

Altair Embed[®]

Altair Embed 2022.1
User Guide

Copyright © 2022 Altair Engineering , Inc. All rights reserved.



Contents

Introduction	1
What you get with Embed	16
New features	16
Resources for learning Embed	17
Online and local help	17
Videos	18
Sample diagrams	18
Technical support.....	18
The Altair Embed product family	18
Professional and Basic editions	18
Special-purpose add-on modules	20
Embed Viewer.....	20
Quick Start	21
Accessing the Chip Temp sample diagram	21
Compiling the source diagram	23
Downloading and debugging	25
Setting diagram parameters.....	28
Running the diagram and viewing results	28
Model-Based Development with Embed	31
Software-in-the-Loop simulation	31
Embedded diagrams	32
\$isCodeGen flag	32
Processor-in-the-Loop simulation	32
Interfacing with code running on Arduino, ARM Cortex M3, Linux Raspberry Pi, C2000, and STM32 devices	32
Source and debug diagrams for Arduino, ARM Cortex M3, Linux Raspberry Pi, C2000, and STM32 targets.....	33
Communication interfaces	34
Measuring CPU utilization.....	35
Hardware-in-the-Loop simulation.....	35
High power safety concerns.....	36
Automatically Generating Executable Code	37
Target support.....	37
Resources used by targets	37
Target resources managed by Embed	38
Generic MCU target support	38
Preparing a diagram for code generation	39
Configure the target	39
Configure the compound block to communicate with the target.....	39
Targets with no floating-point unit.....	39
Target devices with no file system.....	39
Variable names	39
Speed considerations	39
Code generation considerations for low RAM targets	40
Determine stack and heap use	40

Blocks that generate stand-alone C code	41
Generating and downloading code to target devices	47
Generate and download code to run in RAM on ARM Cortex M3, Linux, and C2000 targets	47
Generate and download code to run in FLASH on Arduino, MSP430, and STM32 targets	50
Generate and download code to run in FLASH in batch mode	51
Using the code generation parameters	51
Displaying Coff information	53
Support library	54
Flashing generated code with UniFlash	54
Controlling execution on embedded targets	58
Create custom-rate functions	58
Set the sample rate for the target application	59
Read and write directly to device registers	59
Control execution order	59
Execute initialization code at boot time	60
Debugging code on embedded targets	60
Debugging techniques	60
Debugging code on Arduino, ARM Cortex M3, Linux, C2000, and STM32 targets	64
Using serial monitor to debug code on Arduino targets	65
Debugging real-time analog waveforms using the Arduino serial port	71
Running generated code on HIL hardware	71
Integrating handwritten code with generated code	71
Calling the generated code from a user application	72
Using Extern Read and Extern Write blocks to merge your code	72
Using Extern Definition and Extern Function blocks to add a C function to your diagram	72
Generating code from custom blocks	72
Event logging	73

Using the Target Support Blocks and Commands 75

Using the target support blocks	75
ADC10/12	75
Analog Comparator DAC	76
Analog In	78
Analog Input	78
CAN Receive	80
CAN Transmit	81
CAN Transmit Ready	82
Comparator	83
DAC	83
DAC12	86
Digital In	87
Digital Input	87
Digital Out	88
Digital Output	89
DMA Enable	90
eCAP	91
eCap PWM	91
ePWM	92
ePWM Action	98
ePWM Action Write	99
ePWM Chopper	100

ePWM Force Action	100
ePWM Force Action Write.....	101
ePWM for simulation.....	101
eQEP.....	102
eQEP for simulation	103
Event Capture	103
Extern Definition.....	104
Extern Function.....	105
Extern Read	107
Extern Write	108
Full Compare Action	109
Full Compare PWM.....	109
Get CPU Usage	111
GPIO In	111
GPIO Input	112
GPIO Out	114
GPIO Output	115
Hall Sensor.....	117
HRCAP.....	117
I/O Memory Read.....	118
I/O Memory Write	118
I2C Read Buffer	118
I2C Start Communication.....	119
I2C Write Buffer	120
JSON.....	120
Monitor Buffer Empty	121
Monitor Buffer Read.....	122
Monitor Buffer Write	122
MQTT Publish	123
MQTT Subscribe	124
Op Amp	126
PWM	127
PWM for simulation.....	135
Quadrature Encoder	136
Read Target Memory	138
SD16	138
SD16A.....	139
segmentLCD	139
Serial UART Read	140
Serial UART Write.....	140
Set PWM Mode.....	141
Sigma Delta Filter Module.....	142
SPI Read.....	144
SPI Write	147
UDP Read	149
UDP Write	150
Target Interface.....	150
Watch Dog	152
Web Server	152
Using the Target Config blocks.....	160
Using Arduino Config.....	161
Using ARM Cortex M3 Config.....	162
Using the Linux Config.....	163
Using the F240X Config.....	165
Using F28X Config.....	165
Using Generic MCU Config.....	167

Using MSP430 Config.....	169
Using STM32 Config.....	173
Using the Peripheral Config blocks.....	174
Using ADC Config.....	174
Using CAN Config.....	182
Using DMA Config.....	183
Using ESP8266WiFi Config.....	184
Using GPIO Qualification.....	185
Using I2C Config.....	186
Using SD16 Config.....	188
Using Serial UART Config.....	189
Using SPI Config.....	190
Using SPI Config for Arduino.....	192
Using SPI Config for Linux.....	194
Using the Target Interface commands.....	195
Using the Get Target Stack and Heap command.....	195
Using the Reset Target command.....	195

Using the TI DMC Block Set 197

Similarities and differences between 16-bit and 32-bit TI DMC block.....	197
ACI Motor.....	197
ACI Flux Estimator.....	199
ACI Speed Estimator.....	200
Clarke Transform.....	200
Current Model.....	201
Inverse Clarke Transform.....	201
Inverse Park Transform.....	202
Park Transform.....	202
Phase Voltage Calc.....	202
QEP Speed.....	203
PID Regulator.....	204
Ramp Generator.....	204
Resolver Decoder.....	205
SMO Position Estimator.....	206
Space Vector Generator (Magnitude/Frequency).....	207
Space Vector Generator (Quadrature Control).....	208
Space Vector PWM.....	208
Speed Calculator.....	209
V/Hz Profile Generator.....	210

Using the TI MotorWare Block Set 211

Angle Estimator.....	211
Controller Read Property.....	212
Controller Write Property.....	213
Estimator Read Property.....	213
Estimator Write Property.....	213
Motor Control.....	214

Using the Fixed Point Block Set 217

Fixed Point block set.....	217
less than.....	218
less than or equal to.....	219
equal to (==).....	219
not equal to (!=).....	222

greater than.....	223
greater than or equal to.....	223
-X (negate)	224
abs	226
and	228
atan2	230
const.....	230
convert	231
cos.....	231
CRC16	231
div.....	232
gain	233
limit.....	234
limitedIntegrator (1/S)	236
merge	236
mul	237
not	237
or	238
PID Regulator.....	239
PI Regulator	240
sampleHold	240
shift.....	242
sign.....	243
sin.....	243
sqrt	244
sum	244
transferFunction	245
unitDelay	248
xor	250
Fixed Point Block Set Configure command	252
Tutorials	253
Implementing an elevator door control system	253
Implementing a PID position controller	262
Generating DLLs	273
Creating a DLL	273
Calling a DLL from an Embed diagram.....	275
Verifying DLL results	276
Comparing simulation speed	276
Building a custom DLL	277
Generating code from automatically-generated DLLs	280
Troubleshooting	281
Generating Simulation Objects	285
Creating a Simulation Object	285
Communicating with an embedded simulation object.....	287
Using the createSim function	287
Using the vsmCgRuntimeCommand	287
Using the vsmCgGetLastErrorString()	288
Sample file with simulation object interface	288
C Support Libraries	291
Object files	291
Targeting C code for unsupported platforms	292

C support library source code	292
Compiling and linking the C support library source code	293
Extending the Arduino Block Set	295
Sample diagrams that use Arduino libraries	295
Importing libraries with the Arduino Library Manager	295
Setting up libraries imported with the Arduino License Manager	296
Using the Extern Definition and Extern Function blocks.....	296
Delay functions	296
Example: Importing an Arduino library that displays text on an SSD1306	296
Arduino Pin Mapping	305
Arduino PWM Frequency Table	309
Index	311

Introduction

Altair Embed® is a visual environment for model-based development of embedded control systems. It combines an intuitive graphical interface with a content-rich environment to help you design real-time applications targeting a broad variety of devices, including Arduino®, Linux® Raspberry Pi™ and AMD64™, STMicroelectronics® STM32®, and Texas Instruments™ microcontrollers and processors.

There are several different versions of the Embed software. Click [here](#) to see what's included in each software package.

Intellectual property rights notice

Intellectual Property Rights Notice:

Copyright © 1986-2022 Altair Engineering Inc. All Rights Reserved.

This Intellectual Property Rights Notice is exemplary, and therefore not exhaustive, of intellectual property rights held by Altair Engineering Inc. or its affiliates. Software, other products, and materials of Altair Engineering Inc. or its affiliates are protected under laws of the United States and laws of other jurisdictions. In addition to intellectual property rights indicated herein, such software, other products, and materials of Altair Engineering Inc. or its affiliates may be further protected by patents, additional copyrights, additional trademarks, trade secrets, and additional other intellectual property rights. For avoidance of doubt, copyright notice does not imply publication. Copyrights in the below are held by Altair Engineering Inc. except where otherwise explicitly stated. Additionally, all non-Altair marks are the property of their respective owners.

This Intellectual Property Rights Notice does not give you any right to any product, such as software, or underlying intellectual property rights of Altair Engineering Inc. or its affiliates. Usage, for example, of software of Altair Engineering Inc. or its affiliates is governed by and dependent on a valid license agreement.

Altair Simulation Products

Altair® AcuSolve® ©1997-2022

Altair® Activate® ©1989-2022

Altair® BatteryDesigner™ ©2019-2022

Altair® Compose® ©2007-2022

Altair® ConnectMe™ ©2014-2022

Altair® EDEM™ © 2005-2022 Altair Engineering Limited, © 2019-2022 Altair Engineering Inc.

Altair® ElectroFlo™ ©1992-2022

Altair® Embed® ©1989-2022

Altair® Embed® SE ©1989-2022

Altair® Embed®/Digital Power Designer ©2012-2022

Altair® Embed® Viewer ©1996-2022

Altair® ESAComp® ©1992-2022

Altair® Feko® ©1999-2022 Altair Development S.A. (Pty) Ltd., ©1999-2022 Altair Engineering Inc.

Altair® Flow Simulator™ ©2016-2022

Altair® Flux® ©1983-2022

Altair® FluxMotor® ©2017-2022

Altair® HyperCrash® ©2001-2022

Altair® HyperGraph® ©1995-2022

Altair® HyperLife® ©1990-2022

Altair® HyperMesh® ©1990-2022

Altair® HyperStudy® ©1999-2022
 Altair® HyperView® ©1999-2022
 Altair® HyperWorks® ©1990-2022
 Altair® HyperXtrude® ©1999-2022
 Altair® Inspire™ ©2009-2022
 Altair® Inspire™ Cast ©2011-2022
 Altair® Inspire™ Extrude Metal ©1996-2022
 Altair® Inspire™ Extrude Polymer ©1996-2022
 Altair® Inspire™ Form ©1998-2022
 Altair® Inspire™ Friction Stir Welding ©1996-2022
 Altair® Inspire™ Mold ©2009-2022
 Altair® Inspire™ PolyFoam ©2009-2022
 Altair® Inspire™ Play ©2009-2022
 Altair® Inspire™ Print3D ©2022
 Altair® Inspire™ Render ©1993-2016 Solid Iris Technologies Software Development One PLLC, © 2016-2022 Altair Engineering Inc.
 Altair® Inspire™ Resin Transfer Molding ©1990-2022
 Altair® Inspire™ Studio ©1993-2022
 Altair® Material Data Center™ ©2019-2022
 Altair® MotionSolve® ©2002-2022
 Altair® MotionView® ©1993-2022
 Altair® Multiscale Designer® ©2011-2022
 Altair® nanoFluidX® ©2013-2022 Altair Engineering GmbH, © 2018-2022 Altair Engineering Inc.
 Altair® OptiStruct® ©1996-2022
 Altair® PolEx™ ©2003-2022
 Altair® Pulse™ ©2020-2022
 Altair® Radioss® ©1986-2022
 Altair® SEAM® © 1985-2019 Cambridge Collaborative, Inc., © 2019-2022 Altair Engineering Inc.
 Altair® SimLab® ©2004-2022
 Altair® SimSolid® ©2015-2022
 Altair® ultraFluidX® ©2010-2022 Altair Engineering GmbH, © 2018-2022 Altair Engineering Inc.
 Altair® Virtual Wind Tunnel™ ©2012-2022
 Altair® WinProp™ ©2000-2022
 Altair® WRAP™ ©1998-2022 Altair Engineering AB
 Altair® S-FRAME® © 1995-2022 Altair Engineering Canada, Ltd., © 2021-2022 Altair Engineering Inc.
 Altair® S-STEEL™ © 1995-2022 Altair Engineering Canada, Ltd., © 2021-2022 Altair Engineering Inc.
 Altair® S-PAD™ © 1995-2022 Altair Engineering Canada, Ltd., © 2021-2022 Altair Engineering Inc.
 Altair® S-CONCRETE™ © 1995-2022 Altair Engineering Canada, Ltd., © 2021-2022 Altair Engineering Inc.
 Altair® S-LINE™ © 1995-2022 Altair Engineering Canada, Ltd., © 2021-2022 Altair Engineering Inc.
 Altair® S-TIMBER™ © 1995-2022 Altair Engineering Canada, Ltd., © 2021-2022 Altair Engineering Inc.
 Altair® S-FOUNDATION™ © 1995-2022 Altair Engineering Canada, Ltd., © 2021-2022 Altair Engineering Inc.
 Altair® S-CALC™ © 1995-2022 Altair Engineering Canada, Ltd., © 2021-2022 Altair Engineering Inc.

Altair Packaged Solution Offerings (PSOs)

Altair® Automated Reporting Director™ ©2008-2022
 Altair® e-Motor Director™ ©2019-2022
 Altair® Geomechanics Director™ ©2011-2022
 Altair® Impact Simulation Director™ ©2010-2022
 Altair® Model Mesher Director™ ©2010-2022
 Altair® NVH Director™ ©2010-2022
 Altair® Squeak and Rattle Director™ ©2012-2022
 Altair® Virtual Gauge Director™ ©2012-2022
 Altair® Weld Certification Director™ ©2014-2022
 Altair® Multi-Disciplinary Optimization Director™ ©2012-2022

Altair HPC & Cloud Products

Altair® PBS Professional® ©1994-2022
 Altair® Control™ ©2008-2022
 Altair® Access™ ©2008-2022
 Altair® Accelerator™ ©1995-2022
 Altair® Accelerator™ Plus ©1995-2022
 Altair® FlowTracer™ ©1995-2022
 Altair® Allocator™ ©1995-2022
 Altair® Monitor™ ©1995-2022
 Altair® Hero™ ©1995-2022
 Altair® Software Asset Optimization (SAO) ©2007-2022
 Altair Mistral™ ©2022

Altair Drive ©2021-2022
Altair® Grid Engine® ©2001, 2011-2022
Altair® DesignAI™ ©2022
Altair Breeze™ ©2022

Altair Data Analytics Products

Altair® Knowledge Studio® © 1994-2022 Altair Engineering Canada, Ltd., © 2018-2022 Altair Engineering Inc.
Altair® Knowledge Studio® for Apache Spark © 1994-2022 Altair Engineering Canada, Ltd., © 2018-2022 Altair Engineering Inc.
Altair® Knowledge Seeker™ © 1994-2022 Altair Engineering Canada, Ltd., © 2018-2022 Altair Engineering Inc.
Altair® Knowledge Hub™ © 2017-2022 Datawatch Corporation, © 2018-2022 Altair Engineering Inc.
Altair® Monarch® © 1996-2022 Datawatch Corporation, © 2018-2022 Altair Engineering Inc.
Altair® Panopticon™ © 2004-2022 Datawatch Corporation, © 2018-2022 Altair Engineering Inc.
Altair® SmartWorks™ © 2021-2022
Altair SmartCore™ © 2011-2022
Altair SmartEdge™ © 2011-2022
Altair SmartSight™ © 2011-2022

Altair One™ ©1994-2022

Third-party software licenses

AcuConsole contains material licensed from Intelligent Light (www.ilight.com) and used by permission.

AES License

The original author is Karl Malbrain.

This work, including the source code, documentation and related data, is placed into the public domain. It is distributed under the AES Software License. See http://www.geocities.ws/malbrain/aestable_c.html.

THIS SOFTWARE IS PROVIDED AS-IS WITHOUT WARRANTY OF ANY KIND, NOT EVEN THE IMPLIED WARRANTY OF MERCHANTABILITY. THE AUTHOR OF THIS SOFTWARE, ASSUMES _NO_ RESPONSIBILITY FOR ANY CONSEQUENCE RESULTING FROM THE USE, MODIFICATION, OR REDISTRIBUTION OF THIS SOFTWARE.

Boost Software License

Copyright (c) 2012 Artyom Beilis (Tonkikh)

Distributed under the Boost Software License, Version 1.0. See accompanying file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt

Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CyberX3D4CC License

Copyright (C) 2002-2003 Satoshi Konno

All rights reserved.

Distributed under the CyberX3D4CC Software License. See <https://github.com/cybergarage/CyberX3D4CC/blob/master/COPYING>.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

ECVT License

Copyright (c) 1993-2018 Texas Instruments Incorporated

<http://www.ti.com>

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

FMILIB License

File: FMILIB_License.txt

License information file for the FMILIB.

Note: Content of this file is used verbatim in doxygen generated documentation.

copyright (C) 2012 Modelon AB

3-Clause BSD License:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.\n
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the Modelon AB nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL MODELON AB BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

GRG2 License

GRG2 (Generalized Reduced Gradient 2) licensed from:

Tom Aird

30 Snowball Ct.

Reno, NV 89511

E-mail: TomAird@aol.com

Phone: 775-852-4641

The owner confirmed by email that VisSim is free to use this code for a one-time fee of \$1000. In exchange for a one-time payment of \$1000 (by Sept. 15) Visual Solutions would get a paid-up unsupported license for OptimizePRO that would allow continued selling of OptimizePRO with VisSim and not pay any further royalties.

GridCell License

GridCell.cpp - part of powerPack.dll

MFC Grid Control - Main grid cell class

Provides the implementation for the "default" cell type of the grid control. Adds in cell editing.

Written by Chris Maunder <cmaunder@mail.com>

Copyright (c) 1998-2002. All Rights Reserved.

This code may be used in compiled form in any way you desire. This file may be redistributed unmodified by any means PROVIDING it is not sold for profit without the authors written consent, and providing that this notice and the authors name and all copyright notices remains intact.

An email letting me know how you are using it would be nice as well.

This file is provided "as is" with no expressed or implied warranty. The author accepts no liability for any damage/loss of business that this product may cause.

For use with CGridCtrl v2.20+

History:

Eric Woodruff - 20 Feb 2000 - Added PrintCell() plus other minor changes

Ken Bertelson - 12 Apr 2000 - Split CGridCell into CGridCell and CgridCellBase <kenbertelson@hotmail.com>

C Maunder - 17 Jun 2000 - Font handling optimised, Added CGridDefaultCell

gstreamer License

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc.

51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

See <https://github.com/GStreamer/gstreamer/blob/master/COPYING> .

librdkafka - Apache Kafka C driver library

Copyright (c) 2012-2020, Magnus Edenhill

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

KEPWARE OPC License

KEPWARE OPC license is provided freely and can be used in your own projects. The programming example is provided "AS IS." As such, KEPWARE, Inc makes no claims to the worthiness of the code and does not warrant code to be error free.

LAPACK (CLAPACK) License

CLAPACK is a freely-available software package. It is available from netlib via anonymous ftp and the World Wide Web. Thus, it can be included in commercial software packages (and has been). We only ask that proper credit be given to the authors. Namely, we ask that you cite the LAPACK Users' Guide, Third Edition.

Authors: Anderson, E.; Bai, Z.; Bischof, C.; Blackford, S.; Demmel, J.; Dongarra, J.; Du Croz, J.; Greenbaum, A.; Hammarling, S.; McKenney, A.; Sorensen, D.

Title: LAPACK Users' Guide, Third Edition

Publisher: Society for Industrial and Applied Mathematics, 1999

Philadelphia, PA

ISBN 0-89871-447-8 (paperback)

Like all software, it is copyrighted. It is not trademarked, but we do ask the following:

If you modify the source for these routines we ask that you change the name of the routine and comment the changes made to the original.

We will gladly answer any questions regarding the software. If a modification is done, however, it is the responsibility of the person who modified the routine to provide support.

LWIP License

LWIP is licensed under the BSD license:

Copyright (c) 2001-2004 Swedish Institute of Computer Science. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

LTOA License

LTOA.C

Converts a long integer to a string.

Copyright 1988-90 by Robert B. Stout dba MicroFirm

Released to public domain, 1991

MIT License

Copyright (c) Microsoft Corporation

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE

MKL License (Intel Simplified Software License)

Intel Simplified Software License

(version January 2017)

This license applies to the following products:

- Intel® Math Kernel Library (intel® MKL)
- Intel® Integrated Performance Primitives (Intel® IPP)
- Intel® Distribution for Python
- Intel® Machine Learning Scaling Library (Intel® MLSL)

Copyright © 2017 Intel Corporation.

Use and Redistribution. You may use and redistribute the software (the "Software"), without modification, provided the following conditions are met:

- Redistributions must reproduce the above copyright notice and the following terms of use in the Software and in the documentation and/or other materials provided with the distribution.
- Neither the name of Intel nor the names of its suppliers may be used to endorse or promote products derived from this Software without specific prior written permission.
- No reverse engineering, decompilation, or disassembly of this Software is permitted.

Limited patent license. Intel grants you a world-wide, royalty-free, non-exclusive license under patents it now or hereafter owns or controls to make, have made, use, import, offer to sell and sell ("Utilize") this Software, but solely to the extent that any such patent is necessary to Utilize the Software alone. The patent license shall not apply to any combinations which include this software. No hardware per se is licensed hereunder.

Third party and other Intel programs. "Third Party Programs" are the files listed in the "third-party-programs.txt" text file that is included with the Software and may include Intel programs under separate license terms. Third Party Programs, even if included with the distribution of the Materials, are governed by separate license terms and those license terms solely govern your use of those programs.

DISCLAIMER. THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT ARE DISCLAIMED. THIS SOFTWARE IS NOT INTENDED NOR AUTHORIZED FOR USE IN SYSTEMS OR APPLICATIONS WHERE FAILURE OF THE SOFTWARE MAY CAUSE PERSONAL INJURY OR DEATH.

LIMITATION OF LIABILITY. IN NO EVENT WILL INTEL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. YOU AGREE TO INDEMNIFY AND HOLD INTEL HARMLESS AGAINST ANY CLAIMS AND EXPENSES RESULTING FROM YOUR USE OR UNAUTHORIZED USE OF THE SOFTWARE.

No support. Intel may make changes to the Software, at any time without notice, and is not obligated to support, update or provide training for the Software.

Termination. Intel may terminate your right to use the Software in the event of your breach of this Agreement and you fail to cure the breach within a reasonable period of time.

Feedback. Should you provide Intel with comments, modifications, corrections, enhancements or other input ("Feedback") related to the Software Intel will be free to use, disclose, reproduce, license or otherwise distribute or exploit the Feedback in its sole discretion without any obligations or restrictions of any kind, including without limitation, intellectual property rights or licensing obligations.

Compliance with laws. You agree to comply with all relevant laws and regulations governing your use, transfer, import or export (or prohibition thereof) of the Software.

Governing law. All disputes will be governed by the laws of the United States of America and the State of Delaware without reference to conflict of law principles and subject to the exclusive jurisdiction of the state or federal courts sitting in the State of Delaware, and each party agrees that it submits to the personal jurisdiction and venue of those courts and waives any objections. The United Nations Convention on Contracts for the International Sale of Goods (1980) is specifically excluded and will not apply to the Software.

*Other names and brands may be claimed as the property of others.

Mosquito License

This project is dual licensed under the Eclipse Public License 2.0 and Eclipse Distribution License 1.0 as described in the epl-v20 and edl-v10 files.

nowide License

Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Open Source Computer Vision Library License

Open Source Computer Vision Library (OpenCV) 3.3.1

(3-clause BSD License)

Copyright (C) 2000-2016, Intel Corporation, all rights reserved.

Copyright (C) 2009-2011, Willow Garage Inc., all rights reserved.

Copyright (C) 2009-2016, NVIDIA Corporation, all rights reserved.

Copyright (C) 2010-2013, Advanced Micro Devices, Inc., all rights reserved.

Copyright (C) 2015-2016, OpenCV Foundation, all rights reserved.

Copyright (C) 2015-2016, Itseez Inc., all rights reserved.

Third party copyrights are property of their respective owners.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the names of the copyright holders nor the names of the contributors may be used to endorse or promote products derived from this software without specific prior written permission.

This software is provided by the copyright holders and contributors "as is" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall copyright holders or contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

OpenModelica Binary FMU License

Copyright (c) 2021 Modelica Association Project "FMI". All rights reserved.

The Reference FMUs are released under the 2-Clause BSD license:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The Reference FMUs are a fork of the Test FMUs (<https://github.com/CATIA-Systems/Test-FMUs>) by Dassault Systemes, which are a fork of the FMU SDK (<https://github.com/qtronic/fmusdk>) by QTronic, both released under the 2-Clause BSD License.

The FMI header files are copyright (c) 2008-2011 MODELISAR consortium, 2012-2021. The Modelica Association Project "FMI" and released under the 2-Clause BSD License.

OpenXLSX License

Copyright (c) 2020, Kenneth Troidal Balslev

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Piggio License

This is free and unencumbered software released into the public domain. Anyone is free to copy, modify, publish, use, compile, sell, or distribute this software, either in source code form or as a compiled binary, for any purpose, commercial or non-commercial, and by any means.

In jurisdictions that recognize copyright laws, the author or authors of this software dedicate any and all copyright interest in the software to the public domain. We make this dedication for the benefit of the public at large and to the detriment of our heirs and successors. We intend this dedication to be an overt act of relinquishment in perpetuity of all present and future rights to this software under copyright law.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

For more information, please refer to <http://unlicense.org/>.

PowrPack License

Author: Samir Khan, 475 Drummerhill Crescent Waterloo, Ontario N2T 1G3 Canada

To Whom it May Concern

I started developing the VisSim PowerPack in July 2006. The PowerPack is an addon for VisSim and is a collection of blocks for

- solution of linear and polynomial equations via LAPACK routines
- smoothing signals via various methods
- curve fitting
- entering and displaying matrix data in a tabular GUI grid.

I donated the software to Visual Solutions for use with VisSim and claim no rights.

Pugixml License

pugixml parser - version 1.10

* Copyright (C) 2006-2019, by Arseny Kapoulkine (arseny.kapoulkine@gmail.com)

* Report bugs and download new versions at <https://pugixml.org/>

This library is distributed under the MIT License. See notice at the end of this file.

This work is based on the pugxml parser, which is:

Copyright (C) 2003, by Kristen Wegner (kristen@tima.net)

Copyright (c) 2006-2019 Arseny Kapoulkine

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

STM32CUBEF0/F1/F2/F3/F4/G0 License

BY INSTALLING COPYING, DOWNLOADING, ACCESSING OR OTHERWISE USING THIS SOFTWARE PACKAGE OR ANY PART THEREOF (AND THE RELATED DOCUMENTATION) FROM STMICROELECTRONICS INTERNATIONAL N.V, SWISS BRANCH AND/OR ITS AFFILIATED COMPANIES (STMICROELECTRONICS), THE RECIPIENT, ON BEHALF OF HIMSELF OR HERSELF, OR ON BEHALF OF ANY ENTITY BY WHICH SUCH RECIPIENT IS EMPLOYED AND/OR ENGAGED AGREES TO BE BOUND BY THIS SOFTWARE PACKAGE LICENSE AGREEMENT.

Under STMicroelectronics' intellectual property rights and subject to applicable licensing terms for any third-party software incorporated in this software package and applicable Open Source Terms (as defined here below), the redistribution, reproduction and use in source and binary forms of the software package or any part thereof, with or without modification, are permitted provided that the following conditions are met:

- 1.Redistribution of source code (modified or not) must retain any copyright notice, this list of conditions and the following disclaimer.
- 2.Redistributions in binary form, except as embedded into microcontroller or microprocessor device manufactured by or for STMicroelectronics or a software update for such device, must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3.Neither the name of STMicroelectronics nor the names of other contributors to this software package may be used to endorse or promote products derived from this software package or part thereof without specific written permission.
- 4.This software package or any part thereof, including modifications and/or derivative works of this software package, must be used and execute solely and exclusively on or in combination with a microcontroller or a microprocessor devices manufactured by or for STMicroelectronics.

- 5.No use, reproduction or redistribution of this software package partially or totally may be done in any manner that would subject this software package to any Open Source Terms (as defined below).
- 6.Some portion of the software package may contain software subject to Open Source Terms (as defined below) applicable for each such portion (“Open Source Software”), as further specified in the software package. Such Open Source Software is supplied under the applicable Open Source Terms and is not subject to the terms and conditions of license hereunder. “Open Source Terms” shall mean any open source license which requires as part of distribution of software that the source code of such software is distributed therewith or otherwise made available, or open source license that substantially complies with the Open Source definition specified at www.opensource.org and any other comparable open source license such as for example GNU General Public License (GPL), Eclipse Public License (EPL), Apache Software License, BSD license and MIT license.
- 7.This software package may also include third party software as expressly specified in the software package subject to specific license terms from such third parties. Such third party software is supplied under such specific license terms and is not subject to the terms and conditions of license hereunder. By installing copying, downloading, accessing or otherwise using this software package, the recipient agrees to be bound by such license terms with regard to such third party software.
- 8.STMicroelectronics has no obligation to provide any maintenance, support or updates for the software package.
- 9.The software package is and will remain the exclusive property of STMicroelectronics and its licensors. The recipient will not take any action that jeopardizes STMicroelectronics and its licensors' proprietary rights or acquire any rights in the software package, except the limited rights specified hereunder.
- 10.The recipient shall comply with all applicable laws and regulations affecting the use of the software package or any part thereof including any applicable export control law or regulation.
- 11.Redistribution and use of this software package partially or any part thereof other than as permitted under this license is void and will automatically terminate your rights under this license.

THIS SOFTWARE PACKAGE IS PROVIDED BY STMICROELECTRONICS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS, IMPLIED OR STATUTORY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED TO THE FULLEST EXTENT PERMITTED BY LAW. IN NO EVENT SHALL STMICROELECTRONICS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE PACKAGE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

EXCEPT AS EXPRESSLY PERMITTED HEREUNDER AND SUBJECT TO THE APPLICABLE LICENSING TERMS FOR ANY THIRD-PARTY SOFTWARE INCORPORATED IN THE SOFTWARE PACKAGE AND OPEN SOURCE TERMS AS APPLICABLE, NO LICENSE OR OTHER RIGHTS, WHETHER EXPRESS OR IMPLIED, ARE GRANTED UNDER ANY PATENT OR OTHER INTELLECTUAL PROPERTY RIGHTS OF STMICROELECTRONICS OR ANY THIRD PARTY.

SUNDIALS License

Copyright (c) 2002, The Regents of the University of California.

Produced at the Lawrence Livermore National Laboratory.

Written by S.D. Cohen, A.C. Hindmarsh, R. Serban,

D. Shumaker, and A.G. Taylor.

UCRL-CODE-155951 (CVODE)

UCRL-CODE-155950 (CVODES)

UCRL-CODE-155952 (IDA)

UCRL-CODE-155953 (KINSOL)

All rights reserved.

This file is part of SUNDIALS.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the disclaimer below.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the disclaimer (as noted below) in the documentation and/or other materials provided with the distribution.

3. Neither the name of the UC/LLNL nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OF THE UNIVERSITY OF CALIFORNIA, THE U.S. DEPARTMENT OF ENERGY OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Additional BSD Notice

1. This notice is required to be provided under our contract with the U.S. Department of Energy (DOE). This work was produced at the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-ENG-48 with the DOE.

2. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately-owned rights.

3. Also, reference herein to any specific commercial products, process, or services by trade name, trademark, manufacturer or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

TI C2000 Compiler/TI ARM/TI MSP430 License

C2000 Code Generation Tools Licenses

--BSD-3-Clause--

Copyright (C) 2013 Texas Instruments Incorporated - <http://www.ti.com/>

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

--BSL-1.0--

Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

--Powell--

Copyright Patrick Powell 1995

This code is based on code written by Patrick Powell (papowell@astart.com). It may be used for any purpose as long as this notice remains intact on all source code distributions.

--Thai Open Source Software Center Ltd--

Copyright (c) 1998, 1999, 2000 Thai Open Source Software Center Ltd and Clark Cooper

Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006 Expat maintainers.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NO NINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

--TI TSPA--

Copyright (c) 1990-2014 Texas Instruments Incorporated

All rights reserved not granted herein.

Limited License.

Texas Instruments Incorporated grants a world-wide, royalty-free, non-exclusive license under copyrights and patents it now or hereafter owns or controls to make, have made, use, import, offer to sell and sell ("Utilize") this software subject to the terms herein. With respect to the foregoing patent license, such license is granted solely to the extent that any such patent is necessary to Utilize the software alone. The patent license shall not apply to any combinations which include this software, other than combinations with devices manufactured by or for TI ("TI Devices"). No hardware patent is licensed hereunder.

Redistributions must preserve existing copyright notices and reproduce this license (including the above copyright notice and the disclaimer and (if applicable) source code license limitations below) in the documentation and/or other materials provided with the distribution

Redistribution and use in binary form, without modification, are permitted provided that the following conditions are met:

* No reverse engineering, decompilation, or disassembly of this software is permitted with respect to any software provided in binary form.

* Any redistribution and use are licensed by TI for use only with TI Devices.

* Nothing shall obligate TI to provide you with source code for the software licensed and provided to you in object code.

If software source code is provided to you, modification and redistribution of the source code are permitted provided that the following conditions are met:

* any redistribution and use of the source code, including any resulting derivative works, are licensed by TI for use only with TI Devices.

* any redistribution and use of any object code compiled from the source code and any resulting derivative works, are licensed by TI for use only with TI Devices.

Neither the name of Texas Instruments Incorporated nor the names of its suppliers may be used to endorse or promote products derived from this software without specific prior written permission.

DISCLAIMER.

THIS SOFTWARE IS PROVIDED BY TI AND TI'S LICENSORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL TI AND TI'S LICENSORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

C2000 RTS Library Licenses

Information on the copyrights and licenses for each RTS file are provided in C2000_RTS_18_1_0_LTS.html.

VsWhere License

Uses the MIT License, as described above.

Software Security Measures:

Altair Engineering Inc. and its subsidiaries and affiliates reserve the right to embed software security mechanisms in the Software for the purpose of detecting the installation and/or use of illegal copies of the Software. The Software may collect and transmit non-proprietary data about those illegal copies. Data collected will not include any customer data created by or used in connection with the Software and will not be provided to any third party, except as may be required by law or legal process or to enforce our rights with respect to the use of any illegal copies of the Software. By using the Software, each user consents to such detection and collection of data, as well as its transmission and use if an illegal copy of the Software is detected. No steps may be taken to avoid or detect the purpose of any such security mechanisms.

WiFiEsp Version 2.2.2 License

GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <<http://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others. For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive

or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps:

(1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those

products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents.

States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification are here: <https://github.com/bportaluri/WiFiEsp/blob/master/LICENSE>.

Use of Master and Slave in hardware-specific block descriptions

Embed supports hardware devices that use the words master and slave to describe relationships in bus modes, communication protocols, and connections. We recognize that these words are inappropriate and offensive. We are evaluating how to change these words without introducing technical confusion.

What you get with Embed

Embed is an extension of Embed SE (Simulation-Only Edition). It includes all the features and capabilities of Embed SE, along with the following embedded development features:

- Enhanced automatic C code generators to produce code that will run on the following devices:
 - Arduino Leonardo, Mega 2560, Nano, and Uno
 - AMD64
 - Raspberry Pi Zero, Zero W, 1A+, 1B+, 2B, 3A+, 3B, 3B+, and 4B
 - STMicroelectronics STM32 F0x, F103x, F3x, F4x, F7x, G0x, G4x, H7x, L4x, and WBx series
 - Texas Instruments ARM Cortex[®] M3, C2000[™], Delfino[™], MSP430[™], and Piccolo[™]
- Peripheral block libraries for:
 - Arduino Leonardo, Mega 2560, and Uno
 - AMD64
 - Raspberry Pi Zero, Zero W, 1A+, 1B+, 2B, 3A+, 3B, 3B+, and 4B
 - STMicroelectronics STM32 F0x, F103x, F3x, F4x, F7x, G0x, G4x, H7x, and L4x
 - Texas Instruments ARM Cortex M3, C2000, Delfino, MSP430, and Piccolo
- Texas Instruments digital motor control (DMC) block libraries
- Fixed-Point block library
- Blocks for communicating with the target and performing analog and digital I/O operations with the analog and digital ports on the target device

New features

The most significant addition to the 2022.1 release is the availability of Embed, Embed SE, and Embed Basic as 64-bit applications.

Additionally, the new eDrives add-on module, available as a separate purchase and install, provides a highly efficient environment covering all phases of developing control systems for electric drives.

The table below summarizes the new features in 2022.1. Note that the Code Generation features are included in Embed and Embed Basic, but not Embed SE.

Blocks	New Feature	Version
Flip	Reverses the elements in a matrix or vector horizontally and vertically	2022.1
Import	Recognizes the semi-colon character as a delimiter imported data files	2022.1
OpenVision: Draw Lines, Draw Rectangles, Draw Circles	Includes line type specification	2022.1
PSIMCoupler	Links an Altair PSIM schematic to an Embed diagram for co-simulation	2022.1
State Chart	Synchronizes default state chart execution with time steps in Embed diagram	2022.1
GUI	New Feature	Version
File > Save	Saves diagrams as VSMX files and uses ZIP compression to save the associated files with the diagrams	2022.1
File > Send	Sends diagrams and automatically attaches all associated files	2022.1
Optimization method	Includes Generalized Reduced Gradient optimization method	2022.1
Code Generation	New Feature	Version
Arduino target support	Expands target support to include Nano boards	2022.1
Raspberry Pi OS support	Expands OS support to include Bullseye	2022.1
STMicroelectronics target support	Expands target support to include STM32 WBx boards	2022.1
STMicroelectronics peripheral support	Expands STMicroelectronics peripheral support to include Hall Sensor and Set PWM Mode blocks	2022.1
Texas Instruments target support	Expands target support to include F280025 and F2838x microcontrollers	2022.1
Texas Instruments peripheral support	<ul style="list-style-type: none"> Provides internal and external voltage reference F2838x supports XBAR, CMPC, and CMPD 	2022.1

Resources for learning Embed

The following free resources are available to help you become an expert Embed user.

Online and local help

Embed offers both online and local Help. By default, Embed is installed with online Help; local Help is an optional, separate installation that you can download from <https://altairone.com>. You can switch between online and local Help with the Help > Set Help Source command. Embed uses your current Browser when displaying Help.

There are three versions of Embed software – Embed, Embed SE, and Embed Basic – described under Embed product family [_D2HLink_12015](#). Help is available for all three versions, and for simplicity, Help always refers to the product as Embed, regardless of the version you are using. Sections that do not apply to a specific version are properly noted.

Videos

The [Embed videos repository](#) contains a growing collection of videos that offer quick and easy ways to learn basic concepts in Embed. Each video focuses on a specific Embed feature. If you are a new Embed user, a good video to start with is [Introduction to Altair Embed](#).

Sample diagrams

Your Embed software includes dozens of fully documented sample diagrams that illustrate both simple and complex embedded systems.

To access sample embedded diagrams

- Click on **Examples > Embedded**.

If you know the chip you're targeting, you can see target-specific examples in the corresponding subcategories.

Technical support

The Altair technical support teams are expert Embed users and can provide assistance and advice for most problems you may encounter. Typically, technical support provides help solving specific problems, rather than providing training on how to use Embed.

Support	How to access
Online forum	Click here to go to the Altair Community webpage. Then click Forums & User Groups and select the Embed product. The Embed online forum is a convenient platform for asking questions and searching for answers.
Email	Send an email to embed-support@altair.com and include the following: <ul style="list-style-type: none"> • Briefly describe your issue • Attach the diagram in question • Specify the version of the software
Phone	Click here to contact your local support office.

The Altair Embed product family

Professional and Basic editions

Feature	Embed Basic Edition	Embed Professional Editions	
		Embed	Embed Simulation-Only (SE)
Restricts block diagram size to 100 blocks	✓		
Drag-and-drop block diagram construction	✓	✓	✓
Continuous, discrete, and hybrid simulations	✓	✓	✓

Conditional execution of subsystems	✓	✓	✓
Interactive, batch, auto-restart, and single-step execution modes	✓	✓	✓
Fixed, adaptive, and stiff integration algorithms	✓	✓	✓
Standard block set with 120+ mathematical, scientific, and engineering blocks	✓	✓	✓
OpenVision block set	✓	✓	✓
State Charts block set	✓	✓	✓
Fixed Point block set	✓	✓	
Digital Motor Control block set	✓	✓	
Toolbox libraries for controls, dynamic systems, electric, electromechanical, eMotor, fixed point, hydraulics, logic, process, quaternion operations, and signal generation	✓	✓	✓
Analysis and linearization with Bode, root locus, and frequency domain plots	✓	✓	✓
Constrained parameter optimization	✓	✓	✓
Visualization with interactive plots and stripCharts, 3D animation, 3D plots, and polar plots	✓	✓	✓
Data I/O using National Instruments and Measurement Computing boards	✓	✓	✓
Data I/O with an OPC server		✓	✓
Data I/O with an RS 232 device	✓	✓	✓
Data I/O with UDP	✓	✓	✓
Data I/O with a PCAN USB Controller Area Network (CAN) device	✓	✓	✓
Code generation	✓	✓	
AMD64 target support	✓	✓	
Arduino Leonardo, Mega 2560, Nano, Uno target support	✓	✓	
AMD64 target support	✓	✓	
Raspberry Pi Zero, Zero W, 1A+, 1B+, 2B, 3A+, 3B, 3B+, 4B target support	✓	✓	

STMicroelectronics STM32 F0x, F103x, F3x, F4x, F7x, G0x, G4x, H7x, L4x, WBx target support	✓	✓	
Texas Instruments ARM Cortex M3, C2000, Delfino, MSP430, Piccolo target support	✓	✓	
Generic MCU target support	✓	✓	
HotLink	✓	✓	

Special-purpose add-on modules

Add-On Module	Description
Embed/Core Source Code Library	Provides source code for core blocks not translated into inline code. Target is the PC. Available only for Embed. Not available for academic (EDU) licenses.
Embed/Target Source Code Library	Provides source code for target-specific blocks not translated into inline code. You choose the target: AMD64; Arduino Leonardo, Mega, Nano, Uno; Raspberry Pi Zero, Zero W, 1A+, 1B+, 2B, 3A+, 3B, 3B+, 4B; STMicroelectronics STM32 F0x, F103x, F3x, F4x, F7x, G0x, G4x, H7x, L4x, WBx; Texas Instruments ARM Cortex M3, C2000, Delfino, MSP430, Piccolo. Available only for Embed. Not available for academic (EDU) licenses.
Embed/Comm	Simulates end-to-end communication systems at the signal level using 200+ communications, signal processing, and RF blocks. Available for Embed and Embed SE. The blocks are for simulation only.
Embed/Digital Power Designer	Provides high-level blocks for simulation and code generation of power supply and digital power components and controls. Available only for Embed.
Embed/eDrives	Provides a highly efficient environment covering all phases of developing control systems for electric drives.

Embed Viewer

The Altair Embed® Viewer is a free, run-time version of Embed for sharing Embed diagrams with colleagues and clients not licensed to use Embed.

The Viewer lets recipients open and simulate Embed diagrams of any size; change block and simulation parameters to test various design scenarios; customize plots and other display blocks to present simulation results in different formats; and make printed copies of diagrams for presentation or archiving.

You can also provide recipients with OUT and ELF files allowing them to simulate your HIL models.

The Viewer preserves model integrity by prohibiting recipients from changing wiring and block diagram structure. Moreover, you can lock sensitive information in password-protected compound and embed blocks, allowing you to share confidential designs without the fear of a security breach.

Quick Start

Your Embed software includes dozens of fully documented sample diagrams that illustrate both simple and complex models spanning a broad range of engineering disciplines.

To access the diagrams, click on Examples > Embedded. The two main subcategories are Digital Motor Control and Digital Power Systems. If you know the chip you're targeting, you can see target-specific examples in the target subcategory.

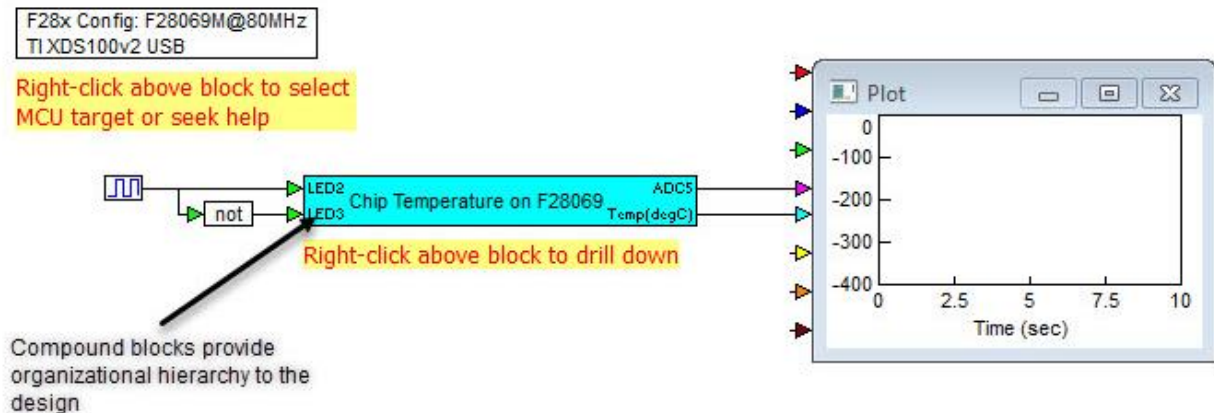
Accessing the Chip Temp sample diagram

This section gets you started using Embed by stepping you through the **Chip Temp on F28069** diagram. This diagram measures the temperature in centigrade of a Texas Instruments Piccolo F28069 device. The ADC channel 5 is redirected from an external pin to the on-chip temperature sensor. The compound block **Turn On Ch 5 Temp Conversion** performs the redirection.

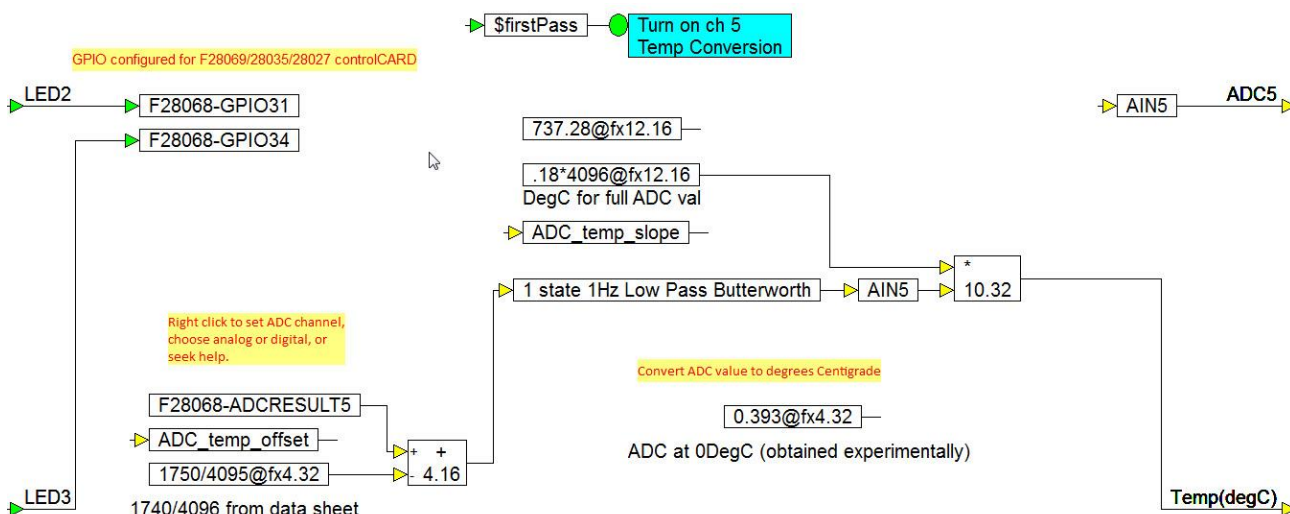
To access the chip temperature model

1. Click **Examples > Embedded > Piccolo > ADC**.
2. Select **Chip Temp on F28069**.

The following diagram appears:

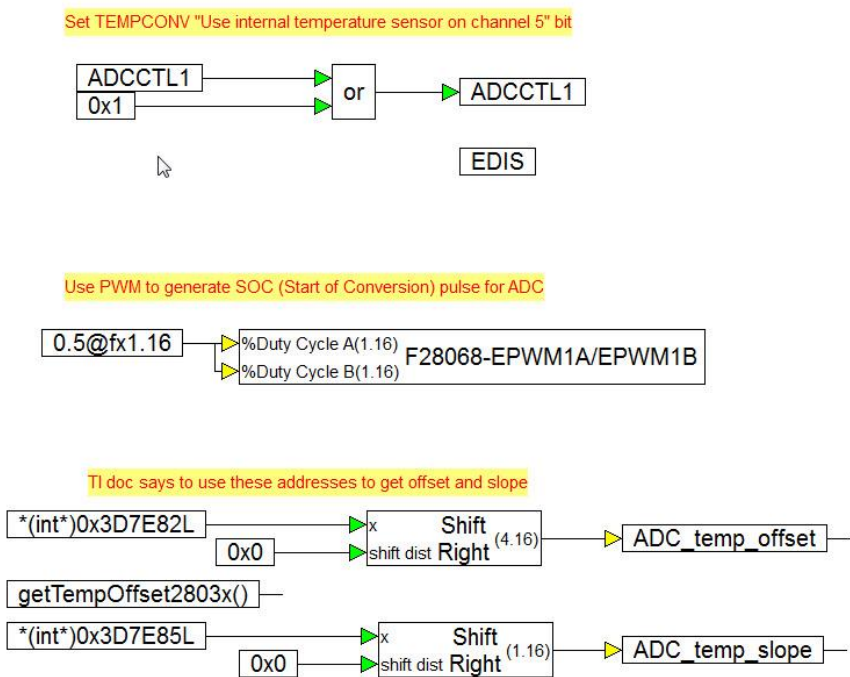


- Right-click **Chip Temperature on F28069** to move down one level of hierarchy.



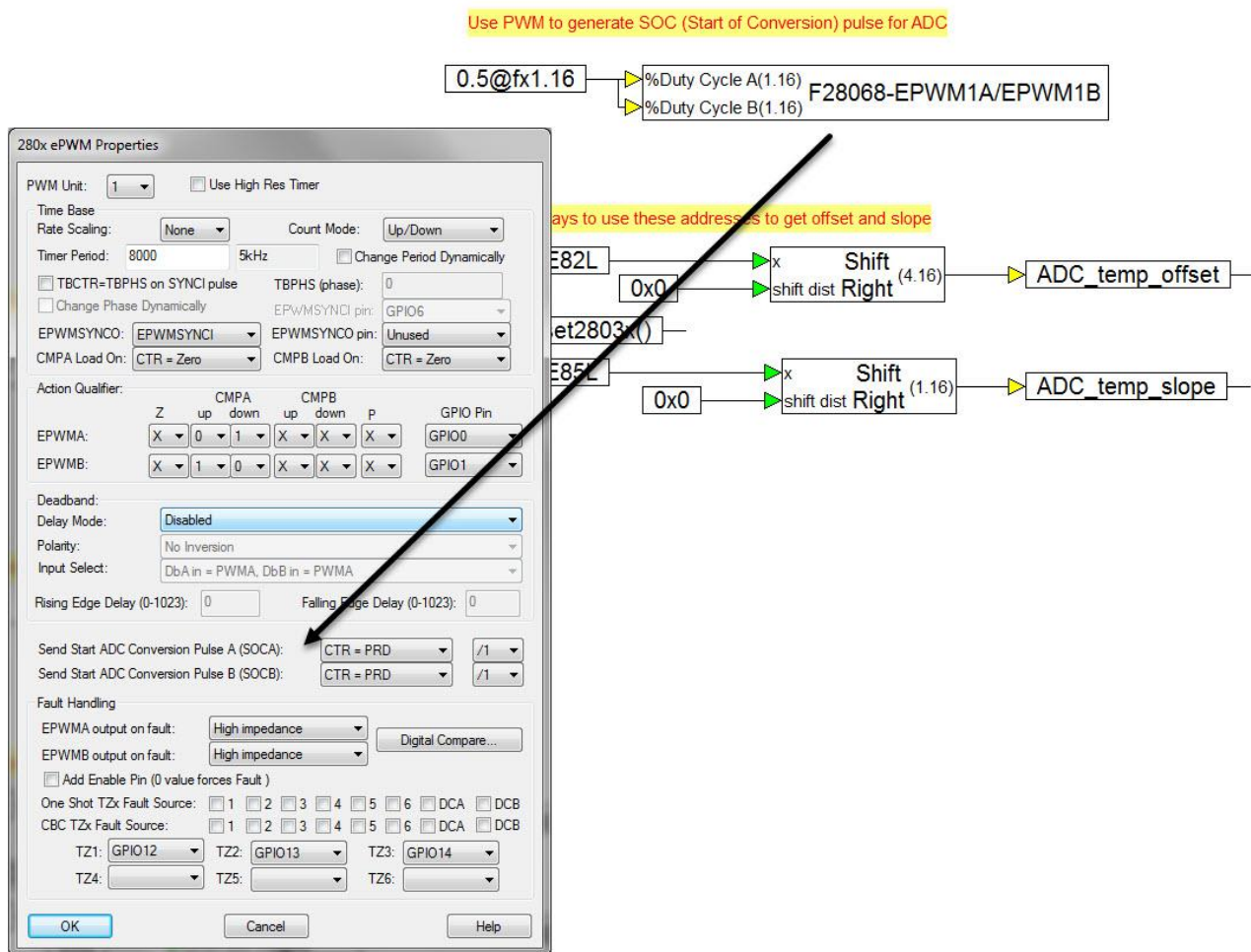
The above diagram reads **ADC channel 5** and applies an offset and gain to convert the reading to degrees centigrade. It also executes the code contained in **Turn on Ch 5 Temp Conversion**, which switches ADC 5 from an external pin to the internal temperature sensor. Note that **Turn on Ch 5 Temp Conversion** is triggered by the built-in variable **\$firstPass**. This means that the block and its contents are executed once at boot time.

- Right-click **Turn on Ch 5 Temp Conversion** to move down one level of hierarchy.



The above diagram enables the internal temperature sensor on ADC A5. The **Extern Read** `*(int*)0x3D7E82L` and **Extern Write** `*(int*)0x3D7E85L` blocks write directly to the hardware registers. To enforce the order of execution, Embed executes parallel flows in top-down order. The **ePWM** block sends Start of Conversion pulses to the ADC A5.

This code also enables the ePWM block to send Start of Conversion pulses to the ACD A5.



Compiling the source diagram

Before you can compile the source diagram, move back up to the top level of the diagram.

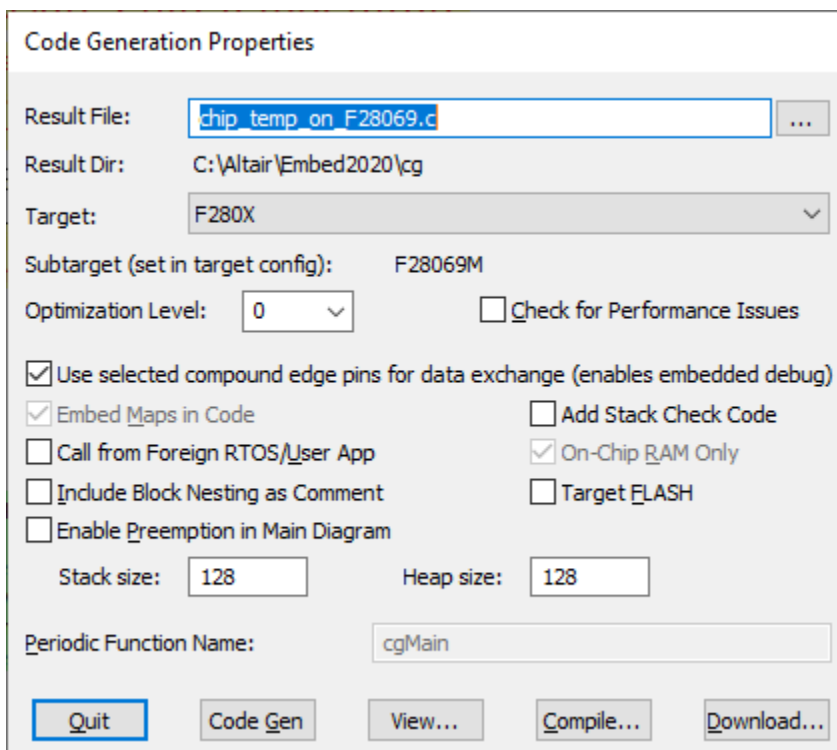
To compile the source diagram

1. At the top level of the diagram, select the compound block **Chip Temperature on F28069**.

The compound block turns red.

2. Click **Tools > Code Gen.**

The C Code Generation dialog box appears.



3. Activate **Use selected compound edge pins for data exchange**. This lets you debug the target executable.
4. Click **Compile** to generate C code and compile it with Code Composer.

The following DOS window appears.

```

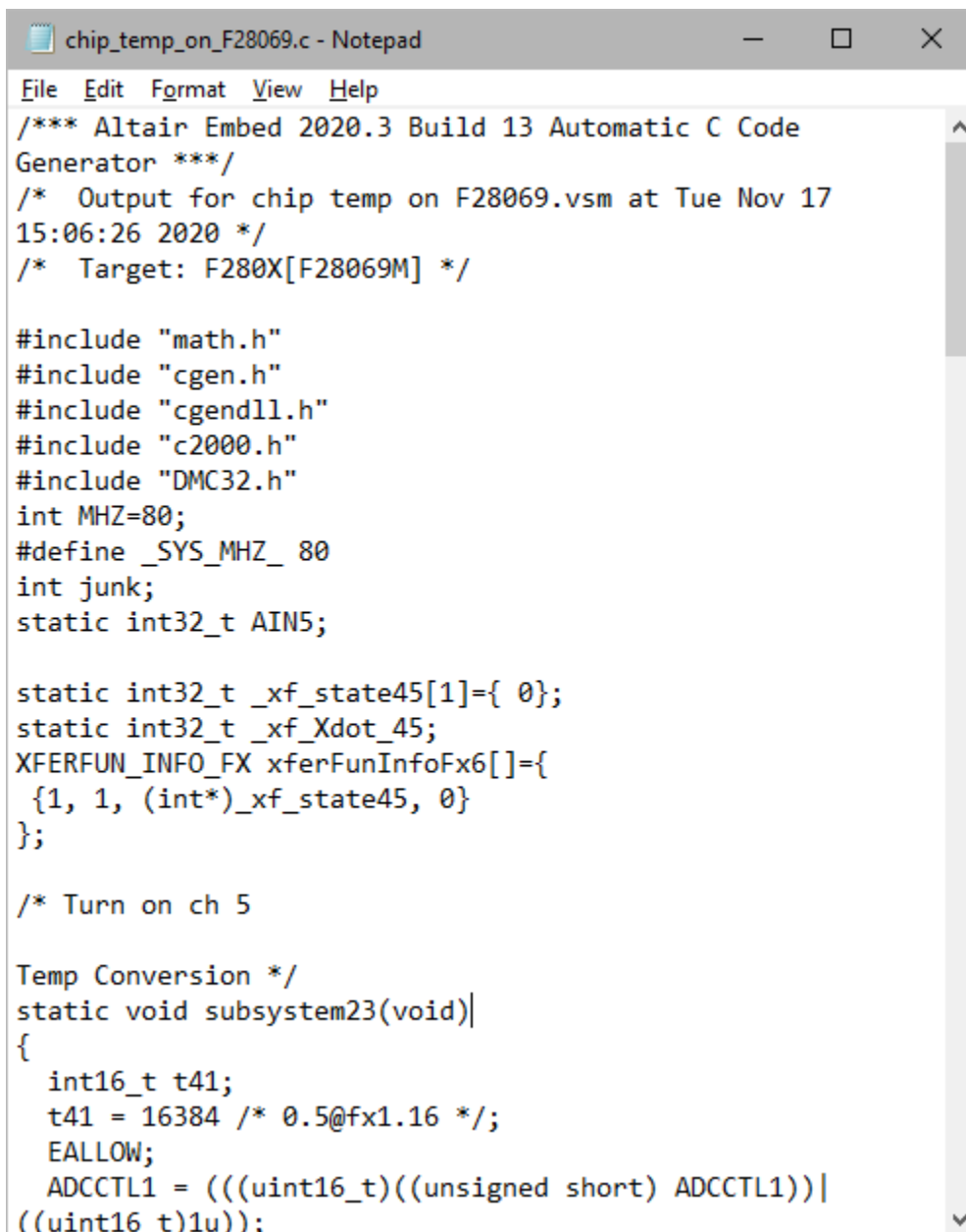
C:\WINDOWS\system32\cmd.exe
C:\Altair\Embed2020\cg>cl2000 -O0 -pds=112 -pds=377 -pds=179 -pds=552 -pden -D_F28069M -g -D_SMALL_RAM -iinclude\TIMot
orWare --define=FAST_ROM_V1p6 -c -fs=tmp -k -ml -mt -v28 -d_DSP -DVERSION_10X=90 -d_F28XX_ -d_F280X_ -fsTMP -i"C:\Altai
r\Embed2020"\vsdk\include -i.\include chip_temp_on_F28069.c

C:\Altair\Embed2020\cg>set HEAP=0x400
C:\Altair\Embed2020\cg>if not "128" == "" set HEAP=128
C:\Altair\Embed2020\cg>set STAK=0x400
C:\Altair\Embed2020\cg>if not "128" == "" set STAK=128
C:\Altair\Embed2020\cg>set FPU_SUFFIX=
C:\Altair\Embed2020\cg>if not "" == "" (
set FPU_SUFFIX=_fpu
set TGT=F280x_fpu
)
C:\Altair\Embed2020\cg>set EXTRALIBS=-l lib\SFO_TI_Build_V6b.lib
C:\Altair\Embed2020\cg>set TGTREGDEF=
C:\Altair\Embed2020\cg>if "2806" == "2806" (if not "9M" == "" set EXTRALIBS=-l lib\HCCal_Type0_V1_fpu32.lib -l lib\SFO_T
I_Build_V6b.lib )
C:\Altair\Embed2020\cg>if "2806" == "2837" (set EXTRALIBS=-l lib\SFO_v8_fpu_lib_build_c28.lib )
C:\Altair\Embed2020\cg>if "F280X" == "f281x" set TGTREGDEF=lib\DSP281x_GlobalVariableDefs.obj

```

The above window displays the output of the Code Composer compiling and linking the diagram.

5. You can check to make sure the compile (cl2000) and link (link2000) are error free, and then press **any key** to continue.
6. Click **Browse** in the C Code Generation dialog box to examine the generated C-code.



```

chip_temp_on_F28069.c - Notepad
File Edit Format View Help
/**** Altair Embed 2020.3 Build 13 Automatic C Code
Generator ****/
/* Output for chip temp on F28069.vsm at Tue Nov 17
15:06:26 2020 */
/* Target: F280X[F28069M] */

#include "math.h"
#include "cgen.h"
#include "cgendll.h"
#include "c2000.h"
#include "DMC32.h"
int MHZ=80;
#define _SYS_MHZ_ 80
int junk;
static int32_t AIN5;

static int32_t _xf_state45[1]={ 0};
static int32_t _xf_Xdot_45;
XFERFUN_INFO_FX xferFunInfoFx6[]={
  {1, 1, (int*)_xf_state45, 0}
};

/* Turn on ch 5

Temp Conversion */
static void subsystem23(void)
{
  int16_t t41;
  t41 = 16384 /* 0.5@fx1.16 */;
  EALLOW;
  ADCCTL1 = (((uint16_t)((unsigned short) ADCCTL1))|
((uint16_t)t)1u));

```

Downloading and debugging

After compiling the source diagram, you can download and debug it using the companion debug diagram **Chip Temp on F28069-d**.

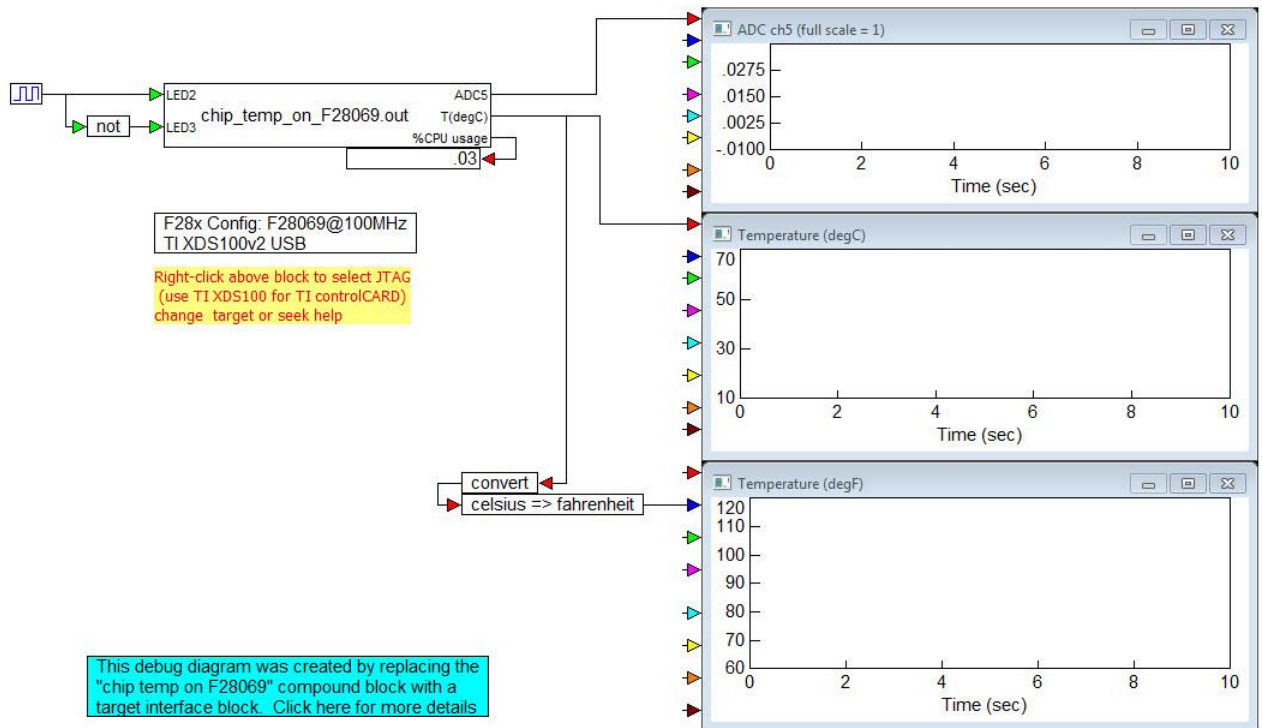
[To download and debug](#)

1. Click **Examples > Embedded > Piccolo > ADC**.
2. Select **Chip Temp on F28069 –d**.

The following diagram appears.

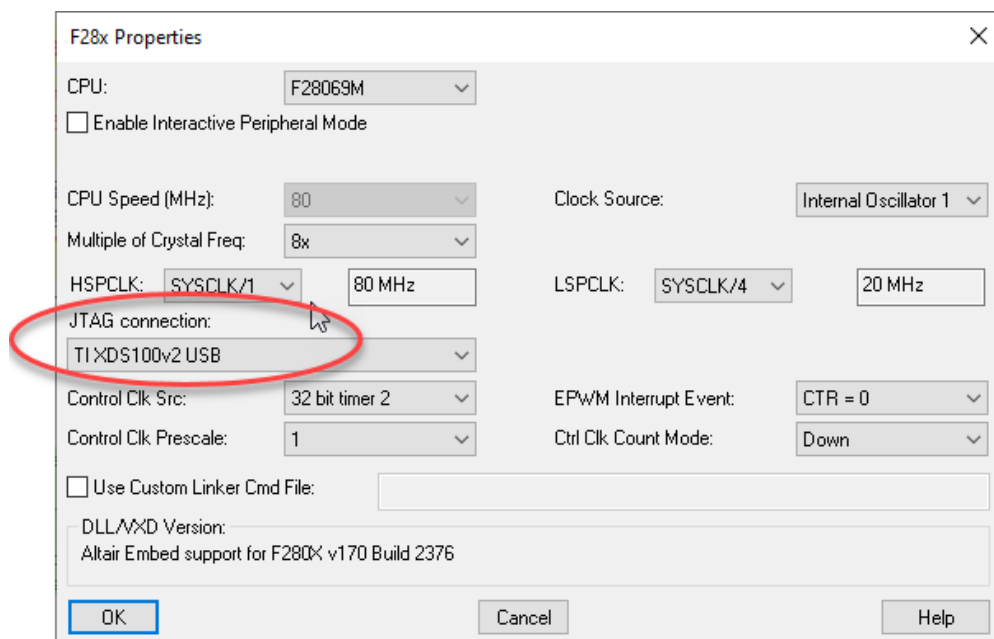
Measure Chip Temperature - companion debug diagram

When you click "Go", the targetInterface block below will download your generated program (chip_temp_on_F28069.out) to the target and then communicate via the JTAG HotLink to send values to your embedded algorithm and receive them back, allowing you to make interactive changes changes in your target algorithms and interactively plot results in Embed.



3. Right-click **F28x Config**.

The F28x Properties dialog box appears.



4. Make sure that the proper JTAG linkage is selected. This example uses XDS100.

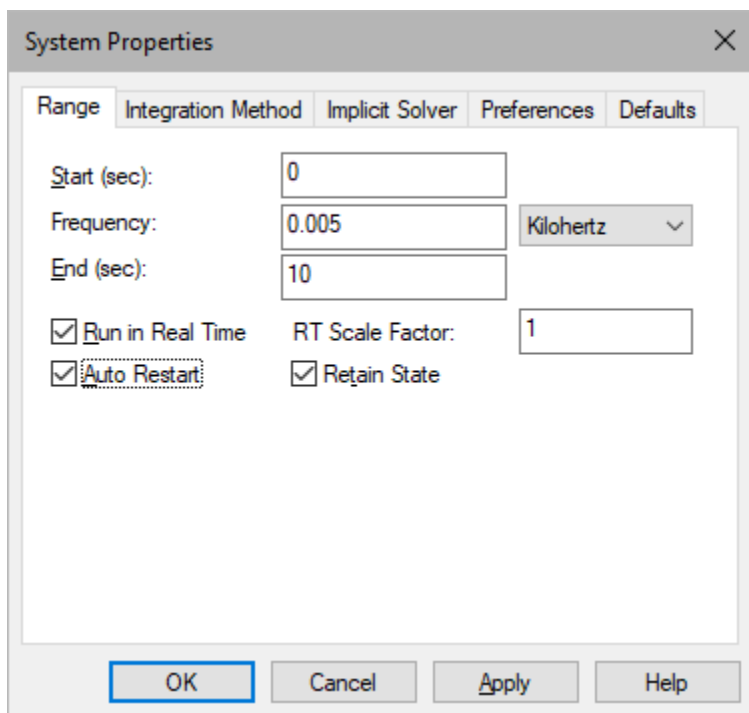
Setting diagram parameters

You set the main run rate of the diagram for both simulation and generated code for the target in the System Properties dialog box. For simulation purposes, you can also set the integration algorithm and duration of the simulation.

To set parameters

1. Choose **System > System Properties**.

The System Properties dialog box appears.



2. The above dialog box is for the **Chip Temp on F28069-d** debug diagram. Notice the options used in the debug diagram:
 - **End** provides a 10 sec interval on plots.
 - **Time Step:** 0.005 provides a 200 Hz update rate to data and plots.
 - **Run in Real Time** executes the diagram in real time, so that Embed runs in sync with the target.
 - **Auto Restart** runs continuously until you stop it.
 - **Retain State** refrains from initializing blocks on restart and prevents reloading of the OUT file.

Running the diagram and viewing results

When you simulate the diagram, Embed downloads and runs the OUT file you created when you compiled the source diagram. After it starts running on the target, Embed provides the following:

- **Interactive inputs:** The 1Hz square wave to the GPIOs blinks the on-board LEDs on the Texas Instruments controlSTICK or controlCARD
- **Interactive plots of on-chip outputs:** The raw ADC A5 reading and adjusted temperature in centigrade

To run the diagram

- Choose **System > Go**, or click  in the toolbar.

The diagram runs until you click Stop in the toolbar.

Model-Based Development with Embed

Model-based development is a process used to design and test embedded systems continuously to reduce defects and development time and improve collaboration between engineers. Embed provides a complete environment for model-based development of embedded systems.

There are typically three phases in the model-based development process:

- **Software-in-the-Loop (SIL):** The plant and controller are modeled
 - and simulated on the host computer
- **Processor-in-the-Loop (PIL):** The controller algorithm is automatically compiled, linked, and downloaded to the target device for execution and testing
- **Hardware-in-the-Loop (HIL):** Plant hardware, sensors, and actuators act as the interface between the plant simulation and the code running on the target

You can generate code at any point in the process to continuously test the validity of your algorithms.

Software-in-the-Loop simulation

During SIL simulation, the plant and controller diagrams are executed entirely on the host computer. It is also common to include the sensor and actuator models as part of the plant diagram, at least initially. Sometimes they are neglected entirely implying they are ideal unity gain models with no dynamics. The controller is separately modelled and interfaced with the plant through the actuator and sensor interface signals. Initial design iterations often use a continuous-time model for the controller, which allows standard frequency domain methods to be employed for the design.

Once the design requirements are satisfied and adequate stability margins are achieved, the controller algorithm is converted to discrete time for further investigation of update time, multi-rate sampling, fixed-point implementations, time jitter, quantization, and time delays. The level of confidence in meeting the design requirements during the SIL phase depends significantly on the accuracy of the plant model. All design changes are made by modifying or augmenting diagrams. The automatically-generated code from a diagram is never directly altered.

Diagrams developed during the SIL phase are reused and possibly supplemented with additional models as development moves into the [PIL](#) and [HIL](#) phases.

Embedded diagrams

With embedded diagrams, you can include version-controlled building block diagrams (root diagrams) within complex models under development (destination diagrams). When embedded in a destination diagram, a read-only version of the root diagram with full navigation capability is inserted into the destination diagram (along with a link to the root diagram). Whenever the root diagram is updated, the changes are propagated to all destination diagrams. As defects and requirement misses are discovered during the PIL [D2HLink 12018](#) and HIL phases, reworks are made to existing root diagrams and new root diagrams may also be created. This automated rework-capture-mechanism maintains the alignment of all SIL, PIL, and HIL diagrams keeping them all up-to-date and accurate.

\$isCodeGen flag

The \$isCodeGen flag is a built-in variable that detects whether you are generating code from a diagram or if you are simply using the diagram in a simulation. In the SIL and PIL phases, the controller (either host- or target-based) receives sensor inputs from the plant diagram residing on the host. When the diagram is migrated to the HIL phase, sensor inputs may be received from the actual sensors. In this situation the \$isCodeGen flag can be used to select when to use actual sensor inputs as opposed to inputs from the plant diagram residing on the host, which in turn, would allow one controller diagram to be used for SIL, PIL, and HIL.

Processor-in-the-Loop simulation

During PIL simulation for C2000, ARM Cortex M3, STM32, and Arduino targets, the controller algorithm is converted to code and executed on a target device while the plant diagram remains in the source diagram on the host. Real-time communication between the target and host is performed via a [HotLink](#) (JTAG, serial, or Ethernet) interface. The Embed GUI is retained while you change controller gains and plot responses from the target.

In most situations, the controller is designed within a compound block with input and output signals flowing through its pins. For C2000, ARM Cortex M3, STM32, and Arduino devices, a small footprint, low jitter, real-time operating system (RTOS) is automatically generated and included in the executable code. After the executable code with RTOS has been created, it is automatically loaded onto the target. Once loaded, the code can run in either stand-alone mode or under host control.

Interfacing with code running on Arduino, ARM Cortex M3, Linux Raspberry Pi, C2000, and STM32 devices

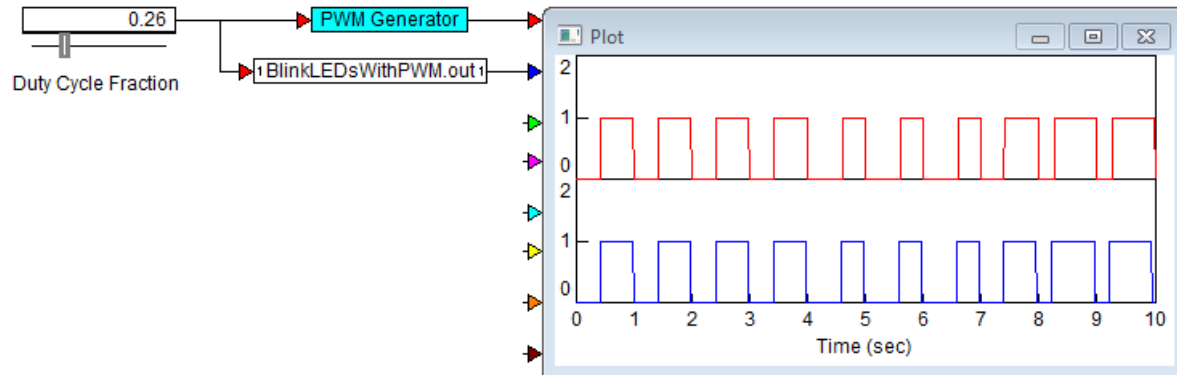
In the source diagram on the host computer, you insert and configure a [Target Interface](#) block for communicating with the generated code running on the target. A Target Interface block is a mirror image of the compound block containing the simulated controller. It has the same number and type of input and output pins; however, it only includes:

- The executable code file name (OUT or ELF file)
- Settings for timing options
- CPU Utilization

Signals applied to the Target Interface input pins travel from the diagram to the executable code running on the target via the [HotLink](#) (JTAG, serial, Ethernet).

The executable code responds to these inputs and produces outputs that travel from the target to the diagram through the [Target Interface](#) block output pins.

A source diagram using the Target Interface block is shown below.



Here, the simulated controller (the compound block named PWM Generator) and the Target Interface block (BlinkLEDsWithPWM.out) are fed into a plot block. A slider provides the duty cycle fraction input signal to the simulated controller and the Target Interface block. When you simulate the diagram, both the simulated controller and the executable code on the target begin executing. After processing the input signal, both the simulated controller and the executable code produce output signals that are made available to the source diagram through the output pins on the compound block and the Target Interface block. These outputs are sent to the plot block allowing one to compare the two responses.

You can use [\\$isCodeGen](#) to detect whether you are generating code from a diagram or if you are simply simulating the diagram. Using this feature allows the same diagram to be used for SIL, PIL, and HIL.

Source and debug diagrams for Arduino, ARM Cortex M3, Linux Raspberry Pi, C2000, and STM32 targets

As models become more complex, the processing and display of model outputs becomes a significant consumer of screen space, often requiring several levels of hierarchy to process and display time history and other information. Due to this, it is convenient to create a special debug diagram from your source diagram:

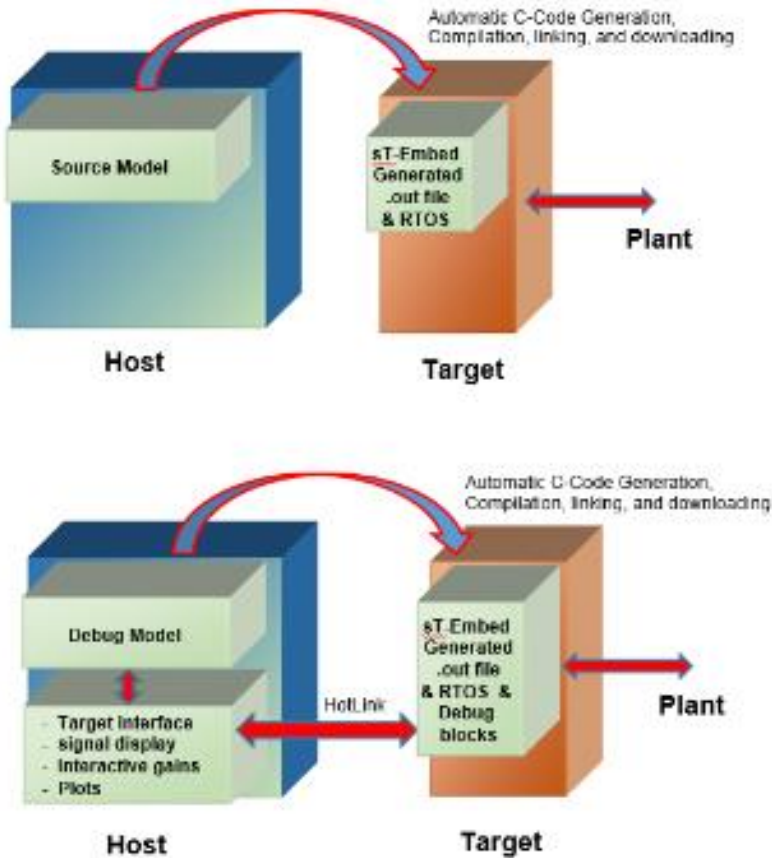
The purpose of the source diagram is twofold:

- To simulate the software that will be executed on the target
- Once the simulation is acceptable, to create the executable code that will be downloaded to the target

The purpose of the debug diagram is threefold:

- To download and run the executable code
- To dynamically adjust parameters in the executable code as it runs on the target using sliders and other signal producers on the host
- To record and process data from the target and present it on the host

The source and debug model functions are shown below.



You create a debug diagram from your source diagram by:

- Saving the [source diagram](#) with “-d” appended to the file name.
- Replacing the compound block from which you generated the executable code with a [Target Interface](#) block. The pin labels on the Target Interface block will be populated with the pin names from the compound block.

When you [simulate the debug diagram](#), the Target Interface block automatically downloads the OUT or ELF file to the target and starts running it. While the target executable runs, you can communicate with the target via the inputs and outputs on the Target Interface block. Note that because the target always runs in real-time, the debug diagram is configured to run in real-time mode when communicating with the target. That way, Embed is in sync with the target.

Communication interfaces

For Texas Instruments C2000 and ARM Cortex M3 devices, and STMicroelectronics STM32 devices, Embed uses a [JTAG HotLink](#) to communicate data between the host and the target device. For Arduino devices, Embed uses a serial HotLink. And for Linux devices, Embed uses an Ethernet or Wi-Fi HotLink.

The HotLinks support both normal and high-speed data collection. In normal mode, commands are sent from the host to the target and data is collected from the target for the host. The signal transfer rate is limited to approximately 200 words/sec.

During the PIL and HIL phases, data must be acquired at a much faster rate. For example, you will have to record data at the frequency that the control algorithm is running at, which could be 10kHz or greater. Embed provides the [Monitor Buffer Read](#) and [Monitor Buffer Write](#) blocks (for all targets except Arduino, Linux, and MSP430) that execute over the HotLink specifically for this purpose. The monitor buffer provides a mechanism for a debug diagram to buffer a large volume of data acquired on the target at the target’s native sample rate, transmit the data periodically over the slower HotLink from

the target to the host, and then make the buffer contents available as a vector of data on the host application. The monitor buffer shown below uses the buffer mechanism to capture and transmit a buffer of 1001 elements collected at 10,000Hz from the target. The host, running at 100Hz, then plots the sample values much like a triggered oscilloscope trace.



Adjusting C2000 and ARM Cortex M3 target update time

Since the JTAG HotLink has bandwidth limitations, Embed lets you adjust the target update time such that the sensor and actuator data transfer between the plant and controller remains synchronized. In addition to the plant diagram, the host model is enhanced with sliders and other signal producers allowing parameters to be interactively adjusted while the controller is running. And, as previously mentioned, the host model can be modified to capture high-speed data collection from the target using the [Monitor Buffer Read](#) and [Write](#) blocks.

Since the modeling uncertainty of the target model used in the SIL phase is removed in the PIL phase, the level of confidence in meeting the design requirements is improved compared with the SIL phase.

A PIL application using a Texas Instruments F28069M LaunchPad microcontroller connected to Embed via USB is shown in the photo on the right.



Measuring CPU utilization

Knowing the level of target CPU utilization is extremely important in embedded applications. Typically, values in the 70% range are considered acceptable for many applications. Applications that consume more are prone to over framing, a situation where not all the control functions are fully executed and completed in the sample time allotted for the controller. The Target Interface block lets you create an output pin that displays the percent CPU utilization on the target while the executable file is running. In addition, you can place Get CPU Usage blocks within any block in the code-generated executable file to output the individual CPU usage for that specific block.

Hardware-in-the-Loop simulation

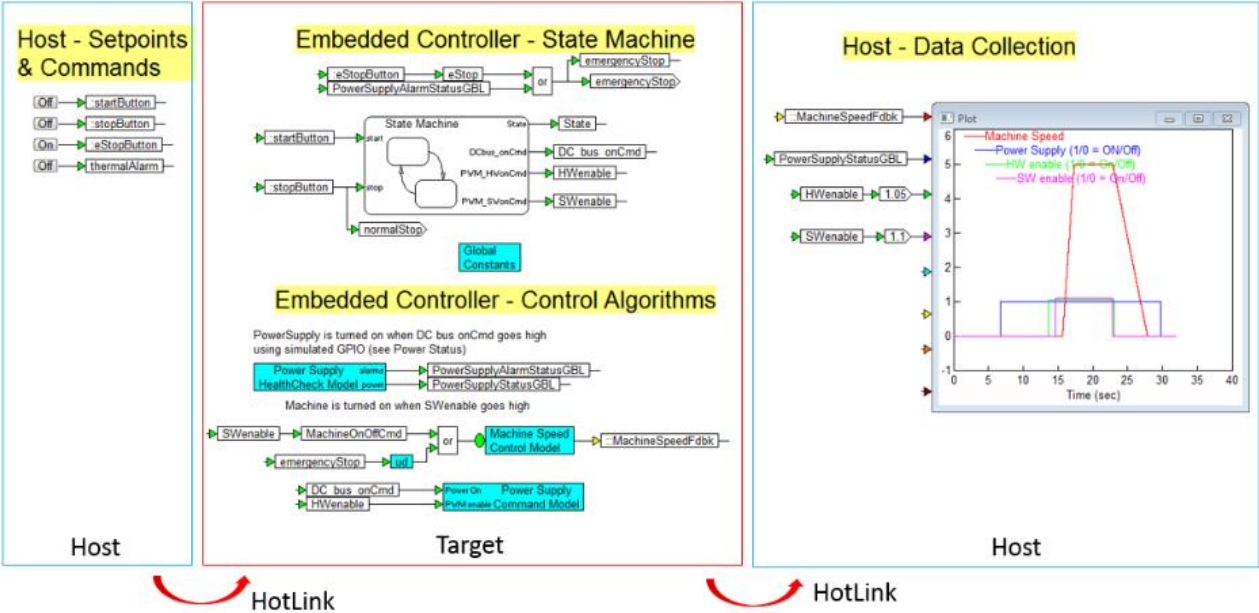
During HIL simulation, the PIL is extended to include the plant hardware, sensors, and actuators. Often, however, in large systems, it may not be feasible or even possible to include the entire plant, all the actuators, and all the sensors. In these situations, some of the sensors, some of the actuators, and parts of the plant (as they become available) are included as HIL devices with the remainder simulated in models executing on the host.

Like in the PIL phase, automatic code generation of the control algorithm is loaded and executed on the target while the host is used to model the parts of the plant for which hardware is not available, as well as to interactively adjust parameters of the control algorithm and capture and present signal time history data.

The HIL phase always executes in real time. Even though the entire plant may not be actual hardware, the ability to test plant components (hydraulic pumps, lines, accumulators, actuators, electric servos, pneumatic actuators, and so on) in the controlled environment of the HIL greatly improves the level of confidence in meeting the design requirements.

High power safety concerns

During the HIL phase, HIL systems will contain high power, which is a serious safety concern. State charts are especially suited to implement control flow logic, including fault detection and fail-safe modes to address faults with respect to both human and machine safety. For example, consider the model below, of an HIL application that starts with a high voltage power supply and applies it to a motor to achieve a setpoint speed.



A state chart is used to sequentially turn on the power supply, turn on the PWM's, and supply the power to the motor. Once the motor is powered, a block diagram controller algorithm maintains its speed. The host model (shown in the blue outlined boxes) is used to control the state machine and plot data collection. The target model (shown in the red outlined box) executes the C code that was automatically generated by the state machine and control algorithm sections of the model.

Automatically Generating Executable Code

Embed generates production-quality C code from any diagram for a variety of platforms including Texas Instruments; STMicroelectronics; Linux AMD64 and Raspberry Pi; and Arduino devices. The generated code is both compact and highly optimized, both of which are essential for low-cost microcontrollers and processors, and high-speed sampling rates.

The C code can also be generated as [DLLs](#) for Microsoft Windows, as well as other [microcontrollers using general support libraries and special drivers](#).

Once code has been generated, Embed automatically builds an executable or DLL (if the Windows target is selected), downloads it to the target device, and establishes an interactive communication link allowing you to control the target execution from the Embed application, as well as collect data from the target to the host during its execution.

When Embed generates code for an embedded target, it uses statically allocated data structures so there is no possibility of failure due to lack of memory resources during program execution.

Target support

Current target support includes the following devices:

- Arduino Leonardo, Mega, and Uno
- AMD64
- Raspberry Pi Zero, Zero W, 1A+, 1B+, 2B, and 3A+, 3B, 3B+, and 4B
- STMicroelectronics STM32 F0x, F103x, F3x, F4x, F7x, G0x, G4x, H7x, and L4x
- Texas Instruments C2000, MSP430, and ARM CortexM3

The generated C code can also be ported to other platforms for compilation and linking, provided you have an ANSI C compiler and the [C-Code Support Library Source Code](#) the platform.

Resources used by targets

Embed target support uses timer and interrupt resources. The following table shows the resources used by the supported targets:

Target	Timer	Interrupt
Arduino	1: Used by Uno and Mega	N/A

	2: Used by Leonardo Main loop is scheduled using the 16-bit timer and not changeable by user	
AMD64 Raspberry Pi	Main loop is scheduled using Linux timer provided by the OS and not changeable by user	N/A
STMicroelectronics STM32 Texas Instruments ARM Cortex M3/M4	SYSTICK (24-bit down count)	N/A
Texas Instruments, Delfino, F280x, F2812, MSP430, Piccolo	User selectable	User selectable
Texas Instruments LF2407	3	2

Embed handles interrupts for the main control rate timer, GPIO pins, DMA, CAN, PWM, QEP, CAP, UART, SCI (serial), I2C, SPI, ADC, watchdog, and power reset.

You can exchange data with Embed from the interrupt handler via the [Extern Read](#) and [Extern Write](#) blocks.

Note: The assembler code for the LF2407 vector tables is supplied in the INSTALL\CGLIB directory so that you can add your own interrupt handlers.

Target resources managed by Embed

Embed lets you access most of the resources available on a target. Managed resources include timer counters, GPIO pins, DMA, CAN, PWM, QEP, CAP, UART, SCI (serial), I2C, SPI, ADC, watchdog, and power reset.

Embed installs an interrupt handler that executes the main body of the C code from your diagram. You can select the source of the interrupt in the corresponding [Target Config dialog box](#). If you are using the [debug diagram](#) the sample rate set in the [Target Interface](#) block is transmitted to the target via the [HotLink interface](#). This allows easy experimentation with the base sampling rate of your algorithm on the target.

Peripheral devices like analog and digital ports on the target can be accessed by simply dragging the corresponding block off the Embedded menu. The blocks can access their corresponding ports immediately without target code generation and will also access the ports when compiled as part of a target system downloaded to execute on the target. For the PC32 board, there is no muxing logic for analog I/O channels, and thus interrupts are enabled across analog I/O ports accessed by Embed.

During a simulation run, data is continuously sent from the PC to the target and target data is continuously sent back to the PC via the [HotLink interface](#). This data movement between PC and target is handled during the target idle time between timer interrupt processing. As the timer interrupt rate increases, less time is available to the idle process to update the PC, and the PC will see less frequent updates. However, since priority is given to the timer interrupt task, the algorithm running on the target will handle analog and digital I/O with no interruption.

Generic MCU target support

The Generic MCU block is used to generate code that will execute code on an embedded target that is not currently supported by Embed. This block generates skeletal embedded code; that is, static RAM and optimized CPU usage. In most cases, when you use the Generic MCU block, you will also need the [C Support Library Source Code](#) add-on module.

Preparing a diagram for code generation

There are several housekeeping chores you should perform before generating code from your source diagram on the host computer.

Configure the target

You use a [Target Config](#) block to choose the target device, crystal multiplier, and processor speed. These settings are saved with the diagram.

There should be only one Target Config block in your diagram.

Configure the compound block to communicate with the target

A Hotlink is used to interactively control the target from the host computer and to collect real-time data from the target to be presented or stored on the host computer.

- C2000, ARM Cortex M3, and STM32 targets use a [JTAG HotLink](#)
- Arduino targets use a serial HotLink
- Linux (AMD64 and Raspberry Pi) targets use an Ethernet or WiFi Hotlink

To use a HotLink, you must encapsulate the controller algorithm in a compound block. It can be handy to label the input and output connectors with unique names.

You can create custom rate functions by [configuring specific parameters](#) in the compound block.

Targets with no floating-point unit

For embedded targets that have limited or no support for floating-point instructions, you can use [fixed-point arithmetic blocks](#). This trade-off has a very limited impact on computational accuracy.

Target devices with no file system

Because many small RAM embedded targets have no file system, you cannot use import or export blocks with them. You can, however, use map blocks. Embed automatically [enables embedding the map data in the generated C code](#) and will automatically select them for certain small targets, like a C2000 device. Map tables will be burned to FLASH memory along with the program code.

Variable names

Variable names in your diagram are retained in the generated code with the following condition: if a variable name includes the punctuation characters + - * # @ ! they are converted to _ .

Consequently, when naming blocks that will eventually be compiled into C code, avoid names that differ only in the above punctuation characters. For example, do not name two blocks Block+ and Block-. They will both be translated into Block_.

Speed considerations

To maximize performance on the target:

- Avoid use of floating-point blocks if you are targeting a fixed-point system. To identify floating-point blocks, activate View > Data Types. Floating-point pins are colored red.
- Avoid use of divide (/) blocks. Division is 10 to 20 times slower than multiplication: when dividing by a constant, multiply by the inverse instead.
- Avoid use of numerical integration (1/S) blocks. These blocks use floating-point operations and invokes the integration engine.

Code generation considerations for low RAM targets

The C2000, ARM Cortex, and STM32 targets support both small and large RAM configurations. The MSP430 and Arduino targets support only small RAM configurations. The code generator emits small RAM code for an embedded target unless it encounters:

- Integration blocks (derivative, integrator, limitedIntegrator, resetIntegrator)
- Linear System blocks (stateSpace, transferFunction)
- Continuous timeDelay blocks
- Matrices used as I/O to enabled compound blocks or top-level compound blocks

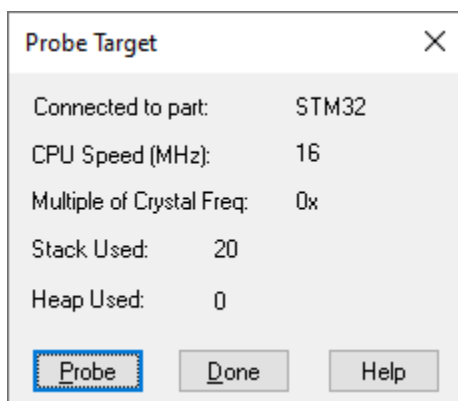
When generating code for C2000, ARM Cortex, or STM32 targets and you include one or more of the above blocks and you activate Check for Performance Issues in the Code Generation dialog box, Embed warns you that large RAM blocks may not run efficiently on the target along with a recommendation on how to update your diagram. You can continue with code generation; however, the generated code may not fit in the target RAM and the code will run slower.

When generating code for Arduino, MSP430, or STM32 targets, you cannot include one or more of the above blocks. If you do, Embed halts code generation and issues a message telling you to replace the blocks.

Note that AMD64 and Raspberry Pi are not low RAM targets. The code generator always emits large RAM code for them.

Determine stack and heap use

For all targets except Arduino, to determine how much stack and heap your diagram uses when running on the target, select **Embedded > your-target-device > Target Interface > Get Target Stack and Heap** after running your diagram on the target in [debug mode](#). If you have conditional subsystems, be sure to exercise all of them before executing this command to get an accurate stack usage report.



Blocks that generate stand-alone C code

Most blocks and State Chart elements do not require a foreign RTOS in order to generate stand-alone C code: this happens automatically during the code generation process. However, some blocks are translated into either an ASCII data stream or a call to an EMPTY function. If you require the generated code for these blocks, you will need to purchase the Embed/Core Source Code Library.

The following tables show how [standard](#) and [OpenVision](#) blocks and [StateChart](#) elements are treated during code generation.

Standard blocks

Block Category	Block Name	Generate C Code	RTOS Dependency
Animation			
	All Animation blocks	NO	YES
Annotation			
	bezel	NO	NO
	comment	NO	NO
	date	YES	NO
	label	NO	NO
	scalarToStruct	YES	YES
	scalarToVec	YES	NO
	structToScalar	YES	NO
	variable	YES	NO
	vecToScalar	YES	NO
	wirePositioner	YES	NO
Arithmetic			
	1/X	YES	NO
	*	YES	NO
	/	YES	NO
	abs	YES	NO
	complexToReIm	YES	YES
	convert	YES	NO
	gain	YES	NO
	magPhase	YES	NO
	pow	YES	NO
	sign	YES	NO
	summingJunction	YES	NO
	unitConversion	YES	NO
	-X	YES	NO
Audio			
	Audioln	YES	YES

	AudioOut	YES	YES
Boolean			
	!=	YES	NO
	==	YES	NO
	<	YES	NO
	<=	YES	NO
	>	YES	NO
	>=	YES	NO
	and	YES	NO
	not	YES	NO
	or	YES	NO
	xor	YES	NO
DDE			
	All DDE blocks	NO	YES
Extensions			
	All Extensions blocks	NO	YES
Fixed Point			
	atan2	YES	YES
	(!=)	YES	NO
	(==)	YES	NO
	(-X)	YES	NO
	<	YES	NO
	<=	YES	NO
	>	YES	NO
	>=	YES	NO
	abs	YES	NO
	and	YES	NO
	convert	YES	NO
	cos	YES	YES
	crc16	YES	YES
	div fixed point	YES	YES
	fixed point constant	YES	NO
	fixed point gain	YES	NO
	fixed point limited integrator	YES	YES
	limit	YES	YES
	merge	YES	NO
	mul fixed point	YES	NO
	not	YES	NO

	or	YES	NO
	PI Regulator	YES	YES
	PID Regulator	YES	YES
	sampleHold	YES	NO
	shift fixed point	YES	NO
	sign fixed point	YES	NO
	sin fixed point	YES	YES
	sqrt	YES	NO
	sum_fp	YES	NO
	transferFunction	YES	YES
	unitDelay	YES	NO
	xor	YES	NO
Integration			
	derivative	YES	YES
	integrator	YES	YES
	limited integrator	YES	YES
	reset integrator	YES	YES
Linear System			
	stateSpace	YES	YES
	TransferFunction	YES	YES
Matrix Operation			
	buffer	YES	YES
	csd	YES	YES
	diag	YES	NO
	dotproduct	YES	NO ¹
	eigen	YES	YES
	fft	YES	YES
	Flip	YES	YES
	ifft	YES	YES
	index	YES	NO
	indexedAssign	YES	YES
	invert	YES	YES
	linearSolve	YES	YES
	matrixSize	YES	NO
	matrixConst	YES	NO
	matrixIn	YES	NO
	matrixMerge	YES	YES
	matrixOut	NO	NO
	maxElement	YES	YES

	meanSmooth	YES	NO
	medianSmooth	YES	YES
	minElement	YES	YES
	mldivide	YES	YES
	multiply	YES	NO
	polyFit	YES	YES
	polyRoots	YES	YES
	psd	YES	NO ²
	reshape	YES	NO
	section	YES	YES
	splineFit	YES	YES
	transpose	YES	YES
	vectorSort	YES	YES
	vsum	YES	YES
Nonlinear			
	case	YES	NO
	crossDetect	YES	YES
	deadband	YES	YES
	delayedSwitch	YES	NO
	demux	YES	NO
	int	YES	NO
	limit	YES	YES
	map	YES	YES
	max	YES	NO
	merge	YES	NO
	min	YES	NO
	quantize	YES	YES
	relay	YES	YES
	sampleHold	YES	NO
OPC			
	All OPC blocks	NO	YES
Optimization			
	All Optimization blocks		
Random Generator			
	Beta	YES	YES
	cauchy	YES	YES
	erlang	YES	YES
	gamma	YES	NO
	gaussian	YES	YES

	pareto	YES	YES
	PRBS	YES	YES
	rayleigh	YES	YES
	triangular	YES	YES
	uniform	YES	YES
	weibull	YES	YES
RealTime			
	All RealTime blocks	NO	YES
Signal Consumer			
	eventLog	YES	YES
	display	YES	YES
	error	NO	NO
	eventDisplay	NO	NO
	execOrder	YES	NO
	export	NO	NO
	histogram	NO	NO
	light	NO	NO
	meter	NO	NO
	plot	NO	NO
	plot3D	NO	NO
	polarPlot	NO	NO
	spectrumDisplay	NO	NO
	stop	NO	NO
	stripChart	NO	NO
	video	NO	NO
Signal Producer			
	button	YES	NO
	constant	YES	NO
	dialogConstant	YES	NO
	dialogTable	YES	NO
	import	YES	YES
	parabola	YES	NO
	pulseTrain	YES	NO
	ramp	YES	NO
	realTime	YES	YES
	sawtooth	YES	NO
	sine	YES	NO
	slider	YES	NO
	Square	YES	NO

	step	YES	NO
	timeOfDay	YES	YES
	timeStamp	NO	YES
	triangle	YES	NO
Time Delay			
	timeDelay	YES	YES
	unitDelay	YES	NO
Transcendental			
	acos	YES	NO
	asin	YES	NO
	atan2	YES	YES
	bessel	YES	YES
	cos	YES	NO
	cosh	YES	NO
	exp	YES	NO
	ln	YES	NO
	log10	YES	NO
	sin	YES	NO
	sinh	YES	NO
	sqrt	YES	NO
	tan	YES	NO
	tanh	YES	NO

1. If the vector size is greater than 6, a foreign RTOS is required.
2. If the inputs are anything except constants, a foreign RTOS is required.

OpenVision blocks

All OpenVision blocks can generate C code; however, they all require a foreign RTOS.

StateChart elements

State Chart Element	Generate C Code	RTOS Dependency
choice	YES	NO ¹
composite state	YES	NO
deep history	YES	YES
entry point	YES	NO
exit point	YES	NO
final state	YES	NO
fork	YES	NO

initial state indicator	YES	NO
junction	YES	NO
state	YES	NO
state chart	YES	NO
terminate	YES	YES

1. If there is an outgoing transition on a choice state with the [else] guard, no foreign RTOS is required; otherwise, there is a call for `stopSimulation()` and a foreign RTOS is required..

Generating and downloading code to target devices

The following code generation options are available for each target.

Target	RAM	FLASH
AMD64	✓	
Arduino		✓
ARM Cortex M3	✓	✓
C2000	✓	✓
MSP430		✓
Raspberry Pi	✓	
STM32		✓

You can also generate C code that can be ported to other platforms for compilation and linking, provided you have an ANSI C compiler and the [C-Code Support Library Source Code](#)

You can also generate code in batch mode that runs in FLASH.

Generate and download code to run in RAM on ARM Cortex M3, Linux, and C2000 targets

During the automatic code generation process, Embed generates a C file from your diagram, then links it with the necessary object files to create an executable file (OUT) to run on the target.

Follow this procedure to generate code to run in RAM on an ARM Cortex M3, AMD64, Raspberry Pi, or C2000 device. Note that when downloading the code to AMD64 or Raspberry Pi, Embed uses the following ports:

- TCP ports 50009 and 57893
- UDP ports 50002, 50003, and 57892

Additionally, when targeting a Raspberry Pi device, you can ensure the executable runs on all Raspberry Pi devices by selecting the slowest Raspberry Pi devices, either 1APlus or 1BPlus.

For the ARM Cortex M3 and C2000 devices, you can alternatively [generate code to run in FLASH](#).

1. Open the diagram.
2. If you are generating code for a subsystem, select the compound block that contains the subsystem.

3. Choose **Tools > Code Gen.**

Code Generation Properties

Result File: ...

Result Dir: C:\Altair\Embed2020\cg

Target: F280X

Subtarget (set in target config): F28069

Optimization Level: 0 Check for Performance Issues

Use selected compound edge pins for data exchange (enables embedded debug)

Embed Maps in Code Add Stack Check Code

Call from Foreign RTOS/User App On-Chip RAM Only

Include Block Nesting as Comment Target FLASH

Enable Preemption in Main Diagram

Stack size: Heap size:

Periodic Function Name:

4. Do the following:

- Under **Result File**, check that the file name displayed is named *currently-open-diagram-name.C*.
- Under **Result Dir**, check that the directory is *<Embed-install-directory>\cg*. If it is not, click ... to update the directory.
- Under **Target**, choose the platform you are targeting. By default, Embed uses the last target you specified for the diagram.

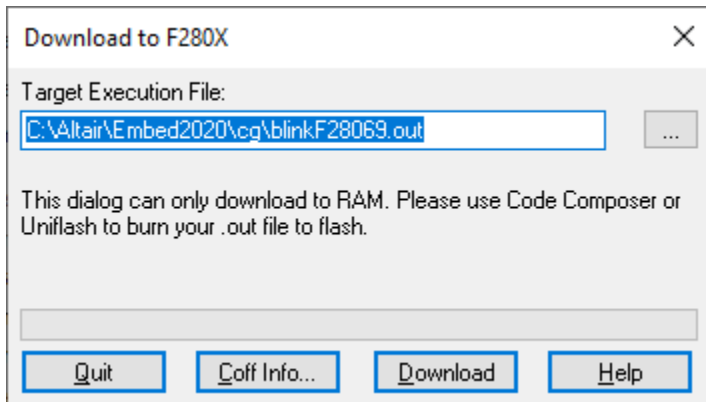
Under **Subtarget**, check that the selected CPU is correct. If it is incorrect, click **Quit** and update the [Target Config](#) block in the diagram. After you save the diagram and click on **Tools > Code Gen**, the subtarget will reflect your change.

5. Choose the remaining [code generation parameters](#).6. Click **Compile**. The following actions are performed:

- The integration algorithm is set to Runge Kutta 4th order if it was previously set to an algorithm more complex than Runge Kutta 4th order.
- A C file is generated. Blocks are translated either directly into one or more C operations or into one or more calls to the Embed C Support [C support library](#). A small set of blocks are unsupported in Embed and are translated into function calls that produce EMPTY returns.
- The C compiler included with the Embed hardware kit is invoked to compile and link the C file, support libraries, and header files to create an OUT file, which is stored in the same location as the C file.
- An MS/DOS window is opened in which to view the code generation, compilation and linking phases.

7. Click **any key** to exit the MS/DOS window.

- Click **Download** in the Code Generation dialog box to start the process of downloading the OUT file to your embedded target. The following dialog box appears.



- Under **Target Execution File**, specify the OUT file to be downloaded, if it is not already set to the file path.
- Click [Coff Info](#) if you want to access information about the sizes of the various linker segments, including data, text, and initialization.
- Click **Download** to download the code to the target RAM.

Generate and download code to run in FLASH on Arduino, MSP430, and STM32 targets

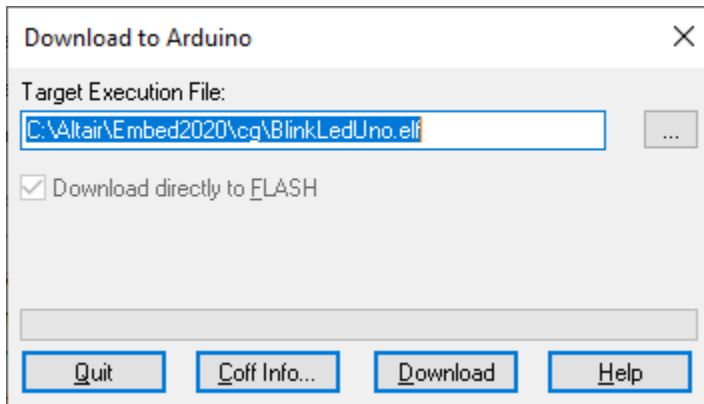
During the automatic code generation process, Embed generates a C file from your diagram, then links it with the necessary object files to create an executable file (OUT or ELF) to run on the target. When you generate executable code for an MSP430 device or Arduino board, the code can only run in FLASH due to the [limited amount of RAM](#) on them.

If you want to flash code to a Texas Instruments C2000 or ARM Cortex M3, see [Flashing generated code with UniFlash](#).

1. Open the diagram.
2. Choose **Tools > Code Gen**.

3. Do the following:
 - Under **Result File**, check that the file name displayed is named *currently-open-diagram-name.C*.
 - Under **Result Dir**, check that the directory is `<Embed-install-directory>\cg`. If it is not, click ... to update the directory.
 - Under **Target**, choose the platform you are targeting. By default, Embed uses the last target you specified for this diagram.
 - Under **Subtarget**, check that the selected CPU is correct. If it is incorrect, click **Quit** and update the [Target Config](#) block in the diagram. After you save the diagram and click on **Tools > Code Gen**, the subtarget will reflect your change.
4. Choose the [code generation parameters](#).
5. Click **Compile**. The following actions are performed:
 - The integration algorithm is set to Runge Kutta 4th order if it was previously set to an algorithm more complex than Runge Kutta 4th order.

- A C file is generated. Blocks are translated either directly into one or more C operations or into one or more calls to the Embed [C support library](#). A small set of blocks are unsupported in Embed and are translated into function calls that produce EMPTY returns.
 - The C compiler included with the Embed hardware kit is invoked to compile and link the C file, support libraries, and header files to create an OUT or ELF file.
 - An MS/DOS window is opened in which to view the code generation, compilation and linking phases.
6. Click **any key** to exit the MS/DOS window and return to the Code Generation dialog box.
 7. Click **Download** in the Code Gen dialog box to start the process of downloading a stand-alone OUT or ELF file to your embedded target. The following dialog box appears.



8. Under **Target Execution File**, specify the OUT or ELF file to be downloaded, if it is not already set to the file path.
9. Click [Coff Info](#) if you want to access information about the sizes of the various linker segments, including data, text, and initialization.
10. Click **Download** to download the code to the target FLASH.

Generate and download code to run in FLASH in batch mode

When you only care about the final results of a simulation or code generation, and you are not interested in looking at the simulation graphically, you can create a batch file that contains commands to start Embed with a specific diagram and execute the diagram, generate code, and download it to FLASH without user interference. To do so, use any editor to create a BAT file that contains the following:

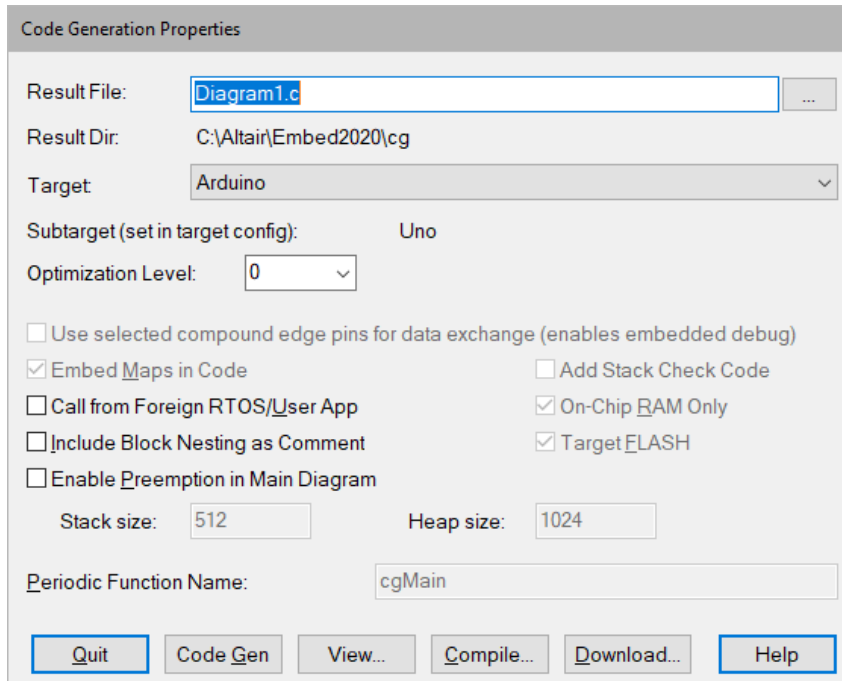
```
<install-path>\vissim32.exe -compile -target=target-name -flash complete-path-to-block-diagram.vsm
```

You must specify the **-flash** argument in conjunction with the **-target** and **-compile** arguments. Specify the *target-name* in the same way that it appears in the Target dropdown in the Tools > Code Generation dialog box. For example, -target=F280X.

Separate arguments with spaces.

Using the code generation parameters

You can select one or more parameters in the Code Gen dialog box to control how the code is generated.



Add Stack Check Code: Causes additional code to be generated that checks stack usage. The stack usage is reported with the [Get Target Stack and Heap](#) command.

Call from Foreign RTOS/User App: Lets you call the generated code from either a foreign RTOS on an embedded system or a user application.

Check for Performance Issues: Alerts you of operations that run slowly on an embedded system and advises how to change your diagram to improve performance. This parameter is not available for MSP430 and Arduino targets.

Embed Maps in Code: Builds the data files from any map block in the generated code. If the target does not have a file system, Embed automatically builds the data file in the generated code. If the target has a file system and you want to change the map file or if you want Embed to read the data file when the embed application starts running, turn this parameter off. This parameter is not available for Arduino targets.

Enable Preemption in Main Diagram: Allows higher priority interrupts to preempt execution of the main diagram.

Heap Size: Allocates memory on the target. The heap is mainly used by matrix blocks and Embed housekeeping routines. Generally, 0x400 is sufficient for initialization of these routines.

To see how much heap is required in your target application, check the `initMatVars()` function in the generated C file. Each Mat Decl uses $\text{arg1} * \text{arg2} * 4$ bytes of heap. For a good estimate of heap usage, add together all the bytes, plus an additional 500 bytes of Embed start-up heap. If your application has not allocated enough heap, Embed issues the following warning:

Insufficient memory to run on target

To correct this situation, increase the heap size and recompile.

This parameter is not available for Arduino targets; the heap is automatically allocated.

Include Block Nesting as Comment: Includes comments with the generated code that indicate the source compound block in the Embed diagram from which the generated code came.

Use selected compound edge pins for data exchange: Enables HIL operation. The generated code running on the target to send and receive data from Embed running on the PC.

This parameter is dimmed when no compound block is selected.

Minimize RAM Usage: Minimizes usage of RAM by omitting numerical integration and floating-point filters. The reward is that the target application will use less memory and fit on a lower-cost part.

On-Chip RAM Only: Some parts (like the F2812) have both off-chip and on-chip RAM. Activating the On-chip RAM Only option will use only on-chip RAM. Note that if you are targeting an Arduino, F280x, or MSP430 device, this parameter is dimmed because the targets do not have off-chip RAM.

Optimization Level: Specifies compiler optimization level, from 0 (no optimization) to 4 (highest level). In rare circumstances, Level 4 may yield inconsistent results, necessitating a lower level of optimization.

Periodic Function Name: Indicates the name of the main Embed time step function. This option can be ignored if you have not activated **Call from Foreign RTOS/User App**.

Result Dir: Displays where the result file will reside. If you want to change the directory, click

Result File: Indicates the C file to be created. By default, the name of the file is the name of the open block diagram. When you press **Compile**, Embed automatically creates the C file, then compiles and links it to create an OUT or ELF file. You can alternatively press **Code Gen** to create only a C file. You can open and browse the C file by clicking on **Browse**.

Stack Size: Allocates memory on the target machine. The stack is used by function calls. Generally, 0x200 is adequate for most applications. This parameter is not available for Arduino targets; the stack is automatically allocated.

Subtarget: Indicates the specific target device. The subtarget is set in the Target Config block in the diagram.

Target: Indicates the embedded platform that the generated code will run on. Only the installed board libraries will appear in this drop-down list box. To create a [Windows DLL](#), select **Host**. To create a [simulation object](#), you must first select a compound block, then select **Simulation Object** from the Target drop-down list.

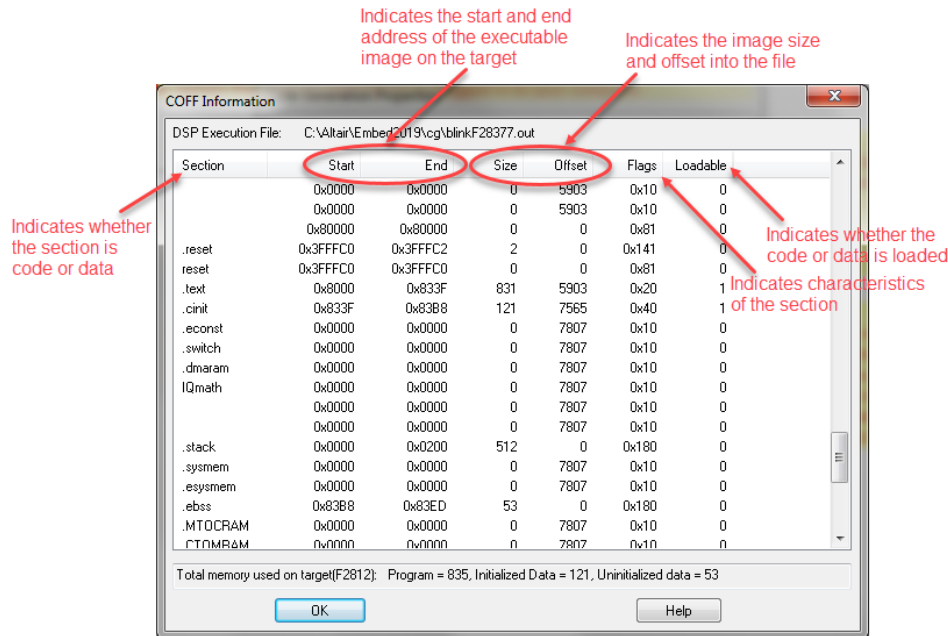
Target FLASH: Most embedded targets have FLASH memory. This is read-only memory that will retain its information even after power has been turned off.

When activated, **Target FLASH** causes the linker to allocate code and constants in FLASH and data in RAM. To burn the resulting executable to FLASH on C2000 and ARM Cortex M3 devices, you need to use a third-party product, like Texas Instruments UniFlash. When **Target FLASH** is not activated, code and data are written to RAM.

For Arduino targets, code is automatically written to FLASH memory; you can ignore this parameter.

Displaying Coff information

When you click on the Coff Info button in the Download dialog box, Embed looks at the generated OUT or ELF and extracts information pertaining to the allocation of FLASH and RAM usage. Because there are limited amounts of FLASH and RAM on a device, it is a good idea to periodically check to see how close you are to exceeding your device's limits. Vendors sell devices with varying amounts of FLASH and RAM. If you are substantially under or approaching your device's limits, you should consider moving to a device that better aligns with your usage requirements.



Support library

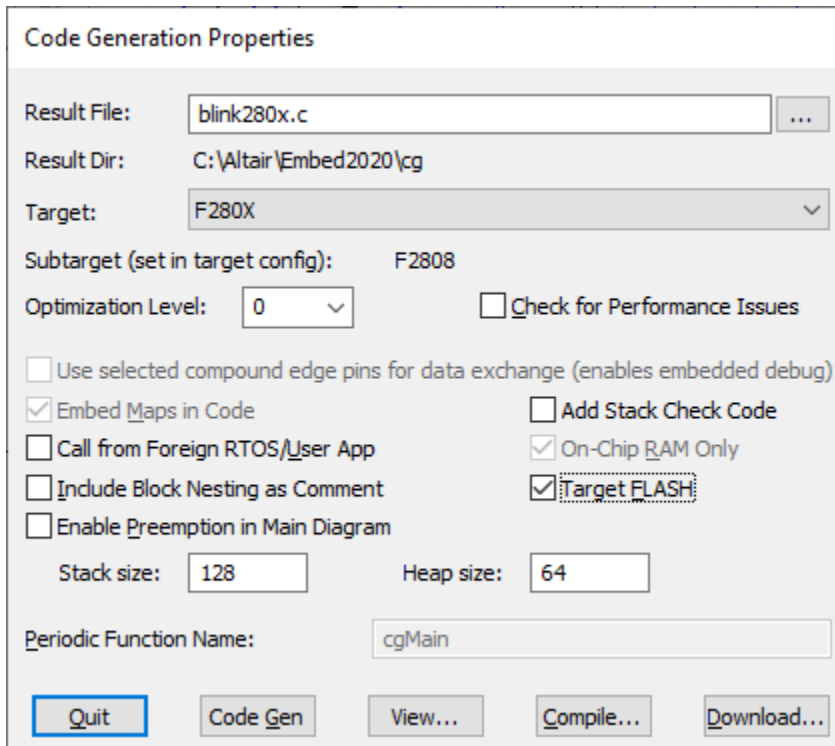
The Embed support library is a collection of precompiled object files that support blocks for which there is no direct translation into C code. These blocks include:

- 1D, 2D and 3D map table lookup
- crossDetect
- deadband
- FFT, inverse FFT, Power Spectral Density
- floating-point transferFunction (fixed-point is generated in-line)
- integrator
- limitedIntegrator
- matrix multiply, inverse, add, power, exponent
- Random number generation
- resetIntegrator
- stateSpace
- timeDelay

Flashing generated code with UniFlash

If you want to flash code to a Texas Instruments C2000 or ARM Cortex M3, you can use the Code Composer Studio™ UniFlash tool. Before you begin, make sure your device is plugged in to your computer and that UniFlash (v3.x, 4.22, or 5.0.0.2289) is installed. If you have not yet installed UniFlash, see the *Altair Embed Installation Guide* or go to the [TI CCS UniFlash](#) web page. We recommend installing UniFlash 5.0.0.2289.

1. Open the diagram.
2. Click **Tools > Code Gen**.

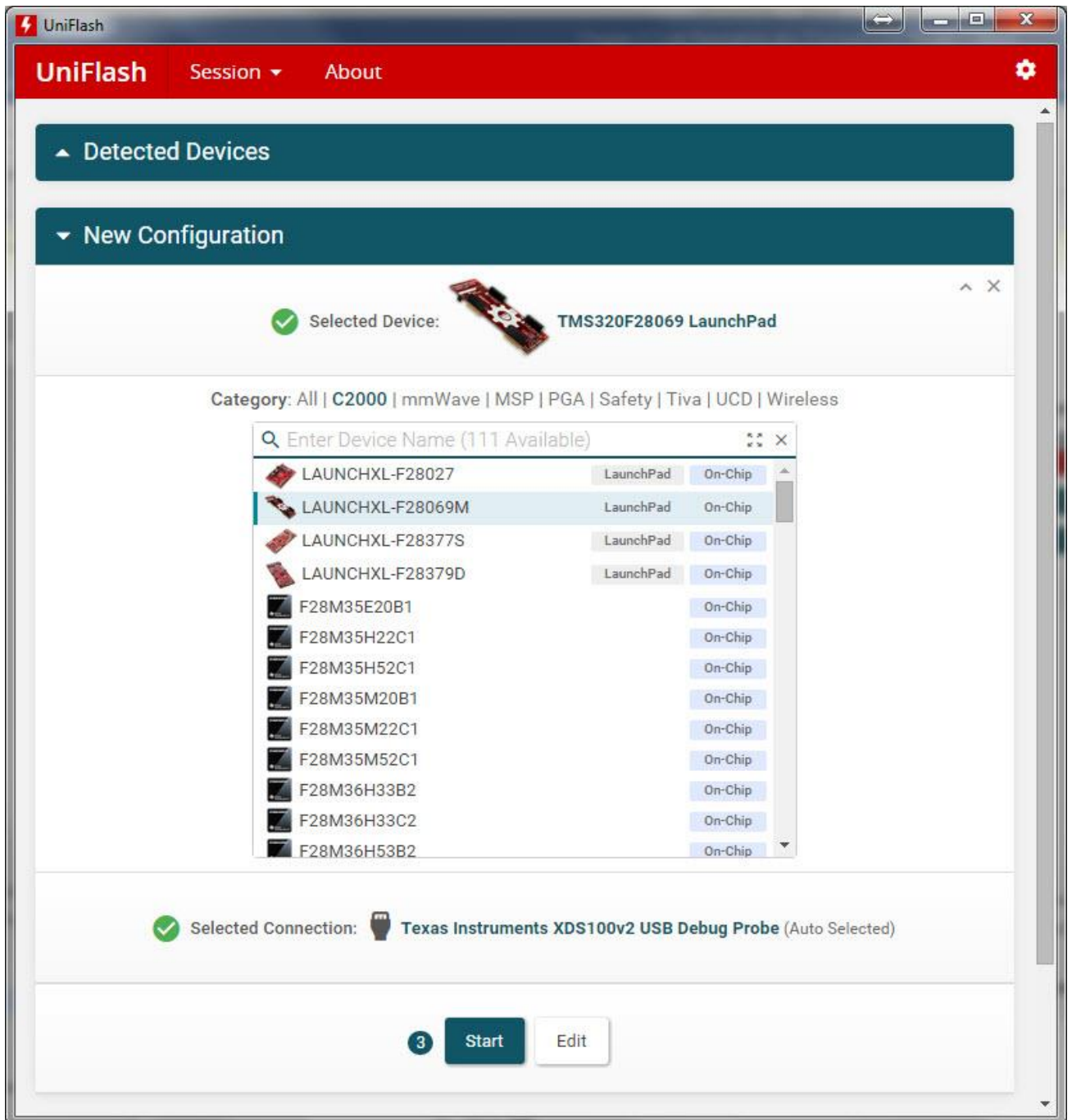


The image shows a 'Code Generation Properties' dialog box with the following settings:

- Result File: blink280x.c
- Result Dir: C:\Altair\Embed2020\cg
- Target: F280X
- Subtarget (set in target config): F2808
- Optimization Level: 0
- Check for Performance Issues
- Use selected compound edge pins for data exchange (enables embedded debug)
- Embed Maps in Code
- Add Stack Check Code
- Call from Foreign RTOS/User App
- On-Chip RAM Only
- Include Block Nesting as Comment
- Target FLASH
- Enable Preemption in Main Diagram
- Stack size: 128
- Heap size: 64
- Periodic Function Name: cgMain

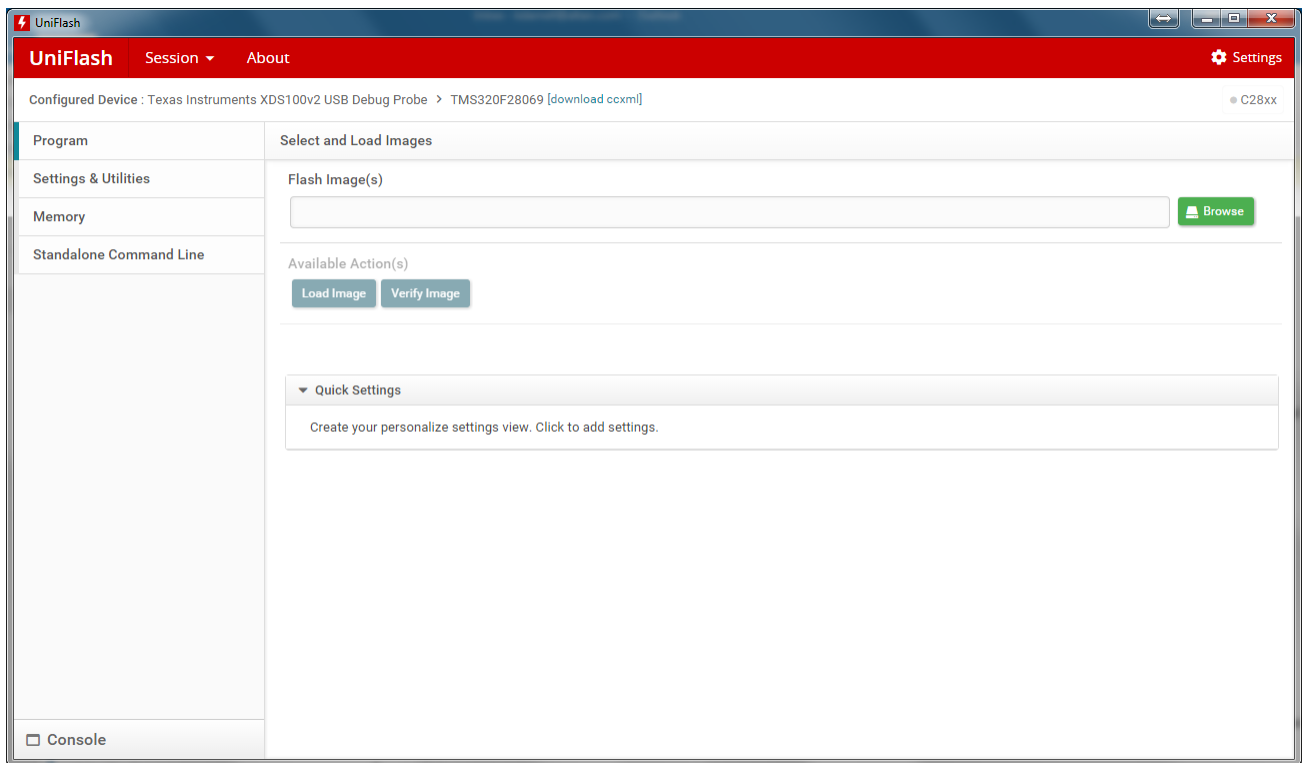
Buttons at the bottom: Quit, Code Gen, View..., Compile..., Download...

3. Activate **Target FLASH**. You can leave the remaining parameters as they are.
4. Click **Compile** to create an OUT file that will run from FLASH. The OUT file is stored in the specified result directory.
5. Click **Quit**.
6. Start **UniFlash**.
7. Under **New Configuration**, select your target device and connection.



In the above window, the **Texas Instruments F28069M LaunchPad** is selected. The corresponding **XDS 100v2 USB Debug Probe** connection is automatically selected.

- Click **Start** to select and download your OUT file to FLASH. The following dialog box appears:



- Under **Flash Image(s)**, click **Browse** and select the OUT file you created in Step 4.
- Under **Available Actions(s)**, click **Load Image**.
- Unplug your device and then plugged it back in; the code will now run in FLASH.

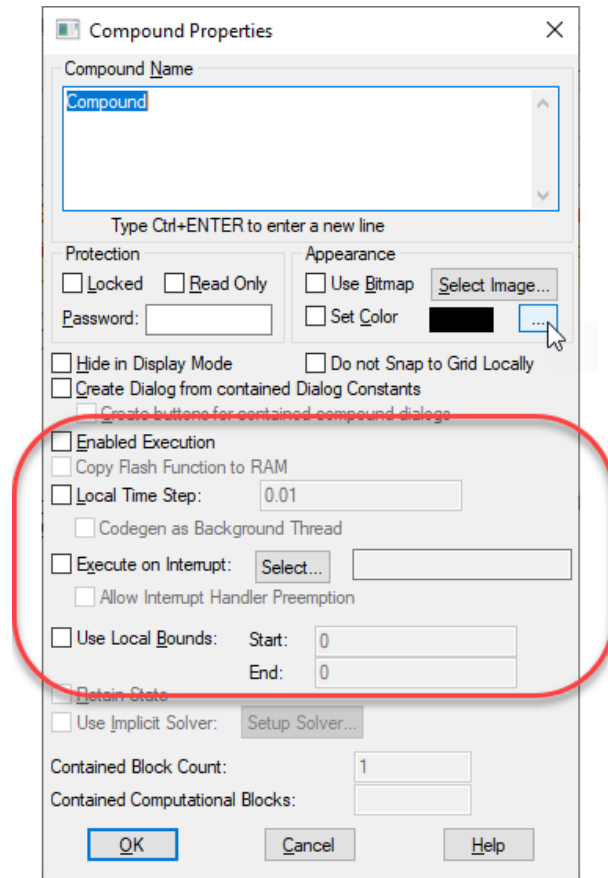
Note: To verify that the OUT file was written to FLASH, click **Verify Image**.

Note: In order to boot from FLASH, the GPIO lines must be set to a specific pattern. For example, to boot from FLASH on the F28069M, GPIO37 and GPIO34 must be High, and SW1 and SW2 on the LaunchPad must be in the ON position. For more information, refer to the Boot Selection Mode in the Texas Instruments Technical Manual for your device.

Controlling execution on embedded targets

Create custom-rate functions

You can control how code is executed on a target by activating specific parameters (show within the red rectangle) in the compound block containing the controller algorithm.



Controlling block execution

With **Enabled Execution** activated, you can control when the block executes in simulation and generates code on the target. For more information, search for *Conditional Execution* under Help > Contents: Modeling and Simulation.

Copying code from FLASH to RAM

Typically, code runs slower in FLASH than RAM. If speed is critical, activate **Copy FLASH Function to RAM** to copy code from FLASH to RAM when you power up the embedded target. If you are [generating code to run in RAM](#), this parameter can be ignored.

Specifying a local step size and local bounds

The Local Time Step parameter lets you specify a step size larger than the base system step size. It must be an integer multiple of the base step. In addition, choosing a local time step will cause the block to appear with a diagonal striping to indicate the nonstandard timing. When used with the Use Local Bounds parameter, the Local Step Size parameter provides additional control over a multi-rate simulation. For more information, search for Multi-rate Simulation under Help > Contents: Modeling and Simulation.

Generating code as preemptible background thread

If you select Code Gen as Background Thread, the contents of the compound block to be executed in a preemptible background thread. It is used for operations that are not as time critical as the main loop. Embed schedules the task to run as close as possible to the local time step rate, but since the task is no longer directly linked to the main task, the time step can be any interval. The code generated from the compound block will be a function call.

Creating and executing interrupt handlers

For Arduino, Delfino, F281X, F280x, MSP430, Piccolo, and STM32 devices, you can create interrupt handlers from a compound block. The compound block contains the logic for the handler; for example, incrementing a counter or sending newly-available data through a serial or SPI port. Generally, you want your interrupt handler to be concise to avoid impacting other operations. When you activate Execute on Interrupt, click Select to choose the source of the interrupt.

If you want to preempt the interrupt for higher priority interrupts, activate the Allow Interrupt Handler Preemption.

Set the sample rate for the target application

The code runs at a timing interval established by an interrupt generated by an onboard clock. Therefore, the generated code runs at a hardware-generated clock rate. This base clock rate is set in the System > System Properties dialog box. It is important to set this rate correctly since it is used to calculate digital filter coefficients and other controller parameters. You can also change this rate in the dialog box for the [Target Interface](#) block.

If the generated code cannot keep up with the specified sample rate, you will see the following message:

Target sampling too fast

Additionally, any background idle loop tasks will be starved of CPU time and will not run.

For example, suppose you request a 10kHz sampling rate (with timer interrupts generated at 100ms), but the diagram takes 150ms to run. This means that by the time the calculation from the diagram has finished, there is already another pending interrupt and you are no longer maintaining the 10kHz rate.

Read and write directly to device registers

You can use the [Extern Read](#) and [Extern Write](#) blocks to read and write hardware registers. The registers have the same names as described in the documentation for your hardware.

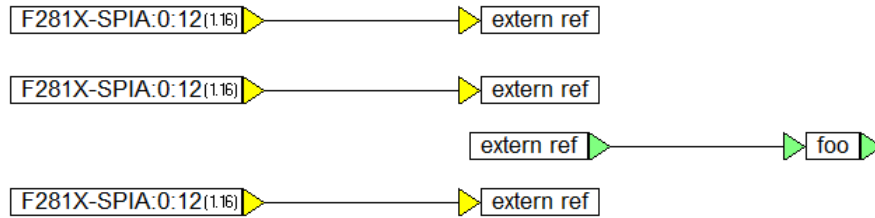
To configure the Extern Read and Extern Write blocks for hardware register access

1. Right-click the **Extern Read** or **Extern Write** block.
2. In the Properties dialog box, under **Data Type**, select **16-bit hardware register** or **32-bit hardware register**.

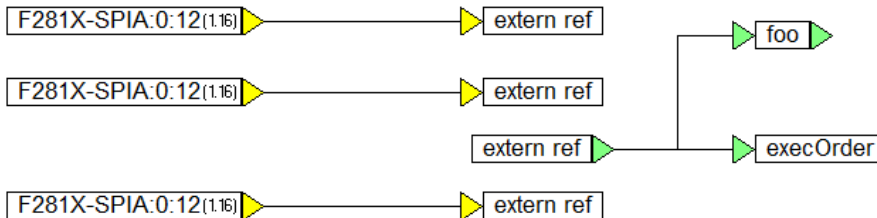
Control execution order

Embed enforces a top-down order of execution. Therefore, if you want to write values to a hardware register in a specific order, simply arrange them from top to bottom.

When you have a combination of Extern Read and Extern Write blocks, and you require a specific order of execution, you can wire the Extern Read block to an execOrder block to control when it is executed. For example, suppose you have the following configuration:



It is ambiguous when the Extern Read block is executed. To enforce the order of execution, wire the Extern Read block into an execOrder block like so:



In this configuration, the Extern Read block will execute third.

Execute initialization code at boot time

For a \$firstPass enabled compound block to run in the boot function, it cannot have any input pins. If there are input pins, Embed expects the inputs must be evaluated in the control routine.

Debugging code on embedded targets

Once the generated code is deployed to the target, it may not execute as expected for several reasons:

- There are timing issues with the data acquisition and actuators
- You may not be able to maintain adequate sample speed
- The code may not fit on the on-chip FLASH or RAM
- The real-world sensor data may not be as precise as expected
- The behavior of the actual plant may be significantly different from the simulated plant

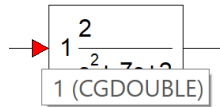
There are different techniques for debugging your diagram depending on your target.

Debugging techniques

Generated code is syntactically error free, however you may still have bugs in your diagrams, depending on such things as algorithm memory requirements, target hardware constraints, and throughput requirements of the algorithm, to name a few. Embed provides a collection of debug tools to investigate and solve these issues.

Controlling execution

The Start, Stop, Step, and Continue controls work on the diagram executing on the PC. If the targetInterface block is set to synchronous operation, the target application responds to these controls; that is, it is stopped and single-stepped as well.



There are no restrictions on when you use the Start, Stop, Step, and Continue controls: you can pause, single-step, and continue execution at any time.

Examining signal values

There are a several ways to examine signal values:

- Hover the mouse over a connector to view signal values.
- Connect display blocks to connector outputs to continuously update displayed values.
- Display output traces interactively in plots.

Setting state chart breakpoints

Within a state chart you can set breakpoints on state events or transitions. After a state chart breakpoint is hit, the simulation pauses, allowing you to single step through the state chart.

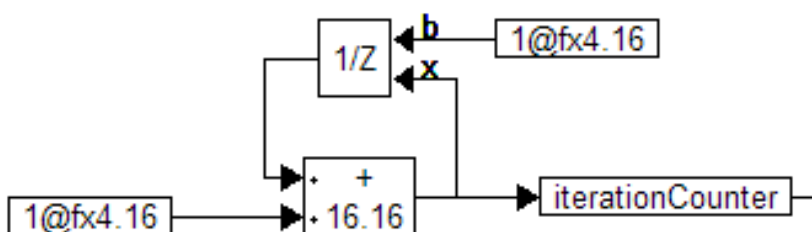
You can also add Watch windows to view variables. When you do so, the active state is highlighted.

Monitoring register values

Use the [Extern Read](#) block with the datatype set to Hardware Register to display register values on the target. These values can be sent over the Hotlink and displayed using plot or display blocks.

Recording event statistics

You can record the number of times a compound block is executed in a simulation or target application by creating an iteration counter in the compounds and sending the counter value back over the HotLink. An example of an iteration counter is shown below.



Execution timing

Knowing the level of target CPU utilization for each conditionally executed subsystem is extremely important in embedded applications. Typically, values in the 70% range are considered acceptable. Applications that consume more are prone to over framing, a situation where not all the control functions are fully executed and completed in the sample time allotted for the controller.

CPU utilization is measured at the system level or within any conditionally executed compound block:

- **System Level CPU Utilization:** The [targetInterface](#) block lets you create an output connector that displays the percent CPU utilization on the target while the target application is running. In addition, you can place [Get CPU Usage](#) blocks within any conditionally executed compound block to output the CPU usage for that specific block.
- **Compound block level CPU Utilization:** You can add a [Get CPU Usage](#) block within any compound block to measure the CPU utilization of that block only. The measured utilization can be captured in a [Monitor Buffer Read](#) block or directly fed out of the compound block for display or plotting.
- **Over Framing Detection:** By activating [Check for Performance Issues](#) during code generation, Embed displays a warning message if the target is not able to execute at its commanded sample rate.

Examining waveforms

Data can be captured on the target and sent to a user-specified buffer by using [Monitor Buffer Read](#) and [Write](#) blocks. Acquisition is triggered using any Boolean operation to start the capture. Once the buffer is full, the data is sent to simulation model on the PC and captured by a [Monitor Buffer Read](#) block connected to a plot block in which you can examine any waveform in the diagram.

Checking for performance degradation

By activating the Check for Performance Issues during code generation, Embed analyzes the diagram for issues that can cause performance degradation in the embedded application. Embed checks for divides, matrix usage, floating point transfer functions, continuous transfer functions, numerical integrations, and more. It then provides suggestions on how to improve performance when these types of issues are detected.

The image shows a screenshot of the 'Code Generation Properties' dialog box. The 'Check for Performance Issues' checkbox is highlighted with a red circle. The dialog box contains the following fields and options:

- Result File: ...
- Result Dir:
- Target:
- Subtarget (set in target config): F2808
- Optimization Level:
- Check for Performance Issues
- Use selected compound edge pins for data exchange (enables embedded debug)
- Embed Maps in Code
- Add Stack Check Code
- Call from Foreign RTOS/User App
- On-Chip RAM Only
- Include Block Nesting as Comment
- Target FLASH
- Enable Preemption in Main Diagram
- Stack size:
- Heap size:
- Periodic Function Name:
- Buttons: Quit, Code Gen, View..., Compile..., Download...

Measuring stack and heap usage

Stack memory is used as scratch space for each thread of execution. Heap memory is used only in rare cases because most memory usage is determined at link time. Heap is used for certain matrix operations (for example, the inverse function) and is tracked so you can be updated continuously on heap usage. To examine the stack and heap:

1. In the **Tools > Code Gen** dialog box, activate **Add Stack Check Code** and click **Compile**.
2. If you view the generated code, the `GET_MAX_STACK_USED();` function appears in the code.

```
static CGDOUBLE _delayOutBuf26=0;
CGDOUBLE _y_16;
CGDOUBLE t32;
CGDOUBLE t27;
CGDOUBLE _uRef_16;
TIMER2TCR = TIMER2TCR;
GET_MAX_STACK_USED();
_y_16 = sim->inSig5[1]->u.Double;
t32 = (( _y_16 +(- _delayOutBuf26))*222.);
t27 = ((0.0001* t32)+ _delayOutBuf26);
_uRef_16 = sim->inSig5[0]->u.Double;
sim->outSig5[0].u.Double = ((( _uRef_16 +(- _y_16))
_delayOutBuf26 = t27;
```

3. Download and run the application on the target.

Allow the target to execute with ample runtime to exercise all significant functions.

4. Under **Embedded > your-device > Target Interface**, choose **Get Target Stack and Heap** to display the maximum stack and heap usage encountered during target execution.

Controlling code placement

Code can be placed in specific memory segments using the [Extern Definition](#) block. This block allows you to declare data and associate it with a specific memory segment using C syntax (compiler dependent).

A common application is allocating DMA RAM. Typically, not all RAM on a target device is capable of DMA, so it is important to attach your DMA data elements to DMA- capable RAM segments. The attachment of a data item to a RAM segment is usually done via `#pragma`.

A second application is the need to copy Flash code to RAM to improve performance. When the code resides in a compound block, selecting the [Copy Flash Function to RAM](#) parameter of the compound block moves the compound block code to RAM prior to execution. The option is enabled when the compound block is configured for Enabled Execution, Local Time Step, or Execute on Interrupt.

Profile matching

Profile matching is one of the most common methods of model validation. The response profiles produced by applying identical command signals to both the simulation model and the target application are compared. Any differences indicate the need for further debugging.

Target application response profiles collected from the target are transmitted to the host over the Hotlink interface and compared with the simulation model response profiles. Embed provides two data collection methods:

- **Direct:** Variable values are transmitted from the target to the host in real time as they are calculated. Overall communication bandwidth is limited to approximately 200Hz.
- **Buffered:** The Embed “Monitor Buffer” records variable values at the target execution rate (this can be 10kHz or greater), stores them in a configurable array, and strobes the array to the host where the data can be saved or plotted.

Debugging code on Arduino, ARM Cortex M3, Linux, C2000, and STM32 targets

When the controller algorithm is running on a target and the plant model is running on the host computer, you use a debug diagram to dynamically adjust parameters in the executable code and to give real-time feedback for the controller responses on the host.

Creating a debug diagram

1. Open the **source diagram** containing the compound block that describes the controller algorithm.
2. Choose **File > Save As** and save the diagram with the same name but with a **-d** suffix. For example, if your source diagram is named `BlinkLED.vsm`, save your debug diagram as `BlinkLED-d.vsm`.

You are now working in your debug diagram.

3. Choose **Embedded > your-target-device > your-target-device Config**.
4. Do one of the following:
 - **For C2000, ARM Cortex M3, and STM32:** Under **JTAG Connection**, select the JTAG connector and click **OK**, or press **ENTER**.
 - **For Arduino:** Under **Virtual Comport**, select the serial port number for your Arduino and click **OK**, or press **ENTER**.
 - **For Linux AMD64 and Raspberry Pi:** Choose the IP address of the target.
5. Choose **System > System Properties** and under the **Range** tab, do the following:
 - Set **Time Step** to 0.01.
 - Activate **Run in Real-time**, **Auto Restart**, and **Retain State**. Since the JTAG bandwidth is around 100-200Hz, there is little sense in running the diagram any faster, even though the target may be running much faster.
 - Click **OK**, or press **ENTER**.
6. From **Embedded > your-target-device**, insert a **Target Interface** block. The Target Interface block is automatically configured with the last OUT or ELF file created for the diagram, along with the same number of inputs and outputs the compound block had when you compiled it. An additional output pin to monitor %CPU usage is created.
7. Wire the **Target Interface** block into the diagram in the same way that the compound block containing the simulated controller is wired.
8. Connect a **display** block to the %CPU usage output pin on the **Target Interface** block.
9. Choose **File > Save**.
10. If the [Target Interface](#) block is not automatically configured, **right-click** the block and set the OUT or ELF file path and input and output pin counts.

JTAG connectors

For C2000, ARM Cortex M3, MSP430, and STM32 devices, Embed provides a built-in USB-based JTAG communication HotLink that lets you interactively control the target from the host, as well as collect data in real time from the target. The supported JTAG connectors are shown in the table below.


JTAG connector	Device
Blackhawk USB2000	C2000, ARM Cortex M3

F280x eZdsp USB	C2000
F283x eZdsp USB	C2000
F2812 eZdsp USB	C2000
Signum JTAGJet	C2000, ARM Cortex M3
Spectrum Digital XDS510 PP	C2000
Spectrum Digital XDS510 USB	C2000, ARM Cortex M3
Stellaris In-Circuit Debug Interface	ARM Cortex M3
TX XDS100v1 USB	C2000, ARM Cortex M3
TI XDS100v2 USB	C2000, ARM Cortex M3
TI XDS100v3 USB	C2000, ARM Cortex M3
TI XDS110 USB	C2000, ARM Cortex M3
TI XDS200 USB	C2000, ARM Cortex M3
TI XDS100v2-M3 USB	C2000, ARM Cortex M3
USB	MSP430
USB ST-LINK/V1+V2+V3	STM32

Simulating with a debug diagram

Normally during development, you will have both the source and debug diagrams open in the Embed work space. You then iterate through the debug cycle by making a change to the source diagram, compiling, then switching to the debug diagram and running the test. Based on test results you go back and modify the source diagram, and so on until you are satisfied with the performance of the code on the target.

When you simulate the diagram, the Target Interface block downloads your generated code to the target and communicates with the target via the JTAG HotLink to send values to your target algorithm and receive them back, allowing you to make interactive changes in your algorithm and interactively plot results in Embed.

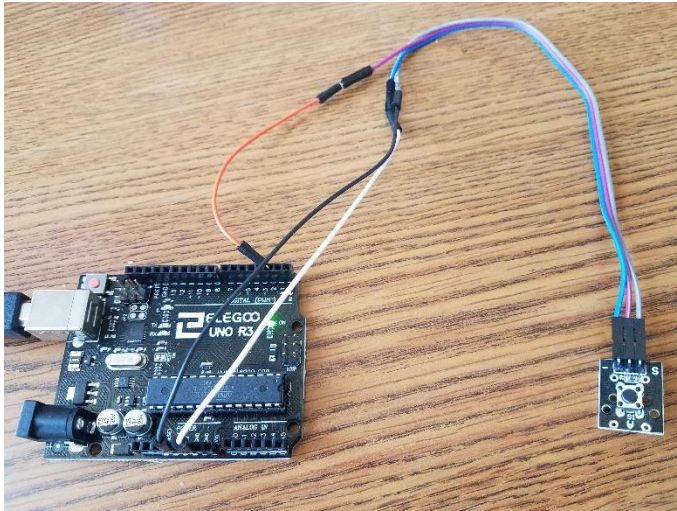
1. Open the [debug diagram](#) containing the **Target Interface** block.
2. Connect your host computer to the target device with a standard USB cable to establish a HotLink connection.
3. Choose **Simulate > Go**, or press  in the toolbar.

Using serial monitor to debug code on Arduino targets

You can also use the serial monitor in the Arduino IDE to debug code on an Arduino target. There is ample information on using the serial monitor on the internet. This section uses a simple example to demonstrate how to use the Arduino IDE serial monitor to debug a faulty diagram. The diagram is supposed to generate code that, when loaded onto an Arduino Uno R3, causes the built-in LED to blink when a push-button sensor attached to the Uno is pressed.

Configure the hardware and the diagram

- Set up the hardware by attaching the pushbutton sensor to the GRND, 3.3V and digital input 2 pins on the Arduino Uno R3.



- Attach the Uno to your computer using a USB cable.
- Start **Embed** and click **File > New**.
- Save the diagram as **BlinkLEDwithPushButton.vsm**.
- Add an **Arduino Config** block, **Digital Input for Arduino** block, and **Digital Output for Arduino** block to your diagram.

Arduino Config: Uno@16MHz

Arduino-PD2 (Arduino pin2)

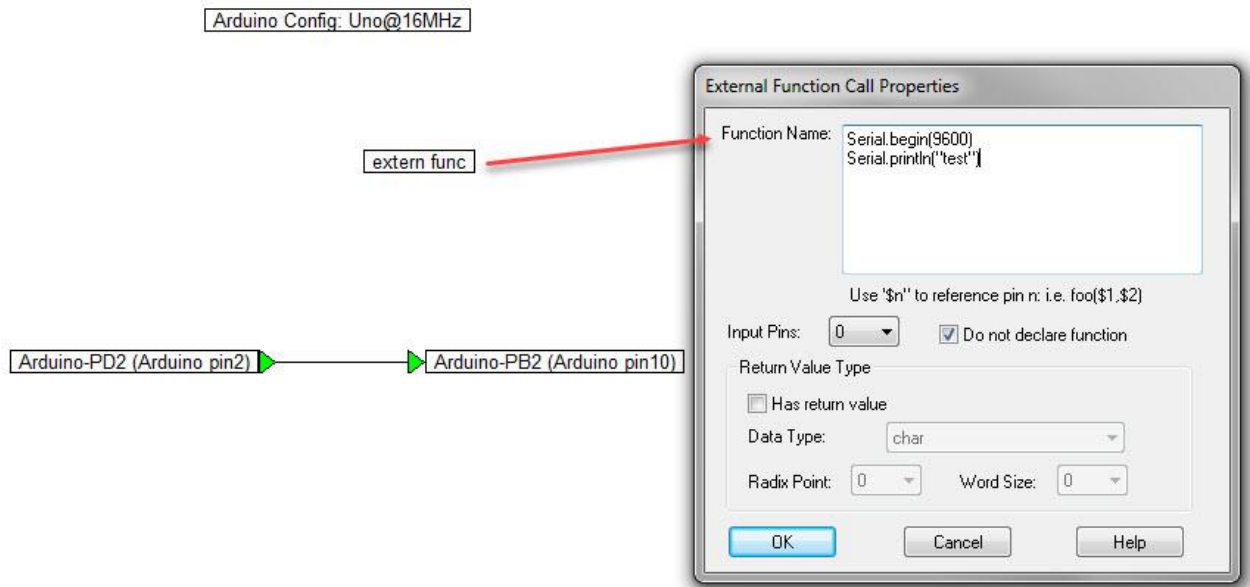
Arduino-PB0 (Arduino pin8)

- Wire the **Digital Input for Arduino** block into the **Digital Output for Arduino** block and make sure:
 - The **Arduino Config** is set to the proper **COMM port**.
 - The **Digital Input for Arduino** is set to **channel 2** and **port PD**.
 - The **Digital Output for Arduino** is set to **channel 2** and **port PB**.
- [Generate code to run on the Arduino](#).
- After the code has been downloaded to the Arduino Uno R3, click the **pushbutton** on the sensor. The built-in LED fails to respond when pushing the sensor button.

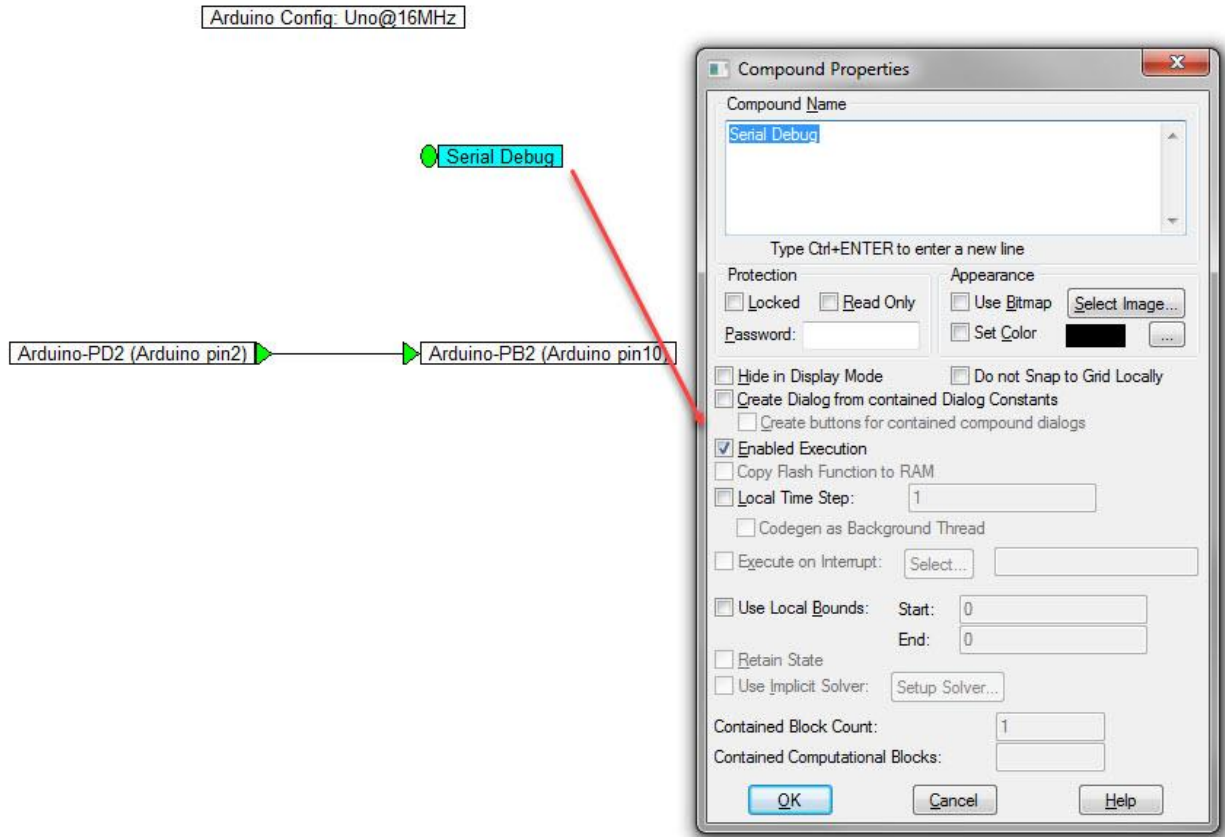
The next several sections step you through how to debug the code using the Arduino serial monitor.

Confirm that data can be printed to the serial monitor

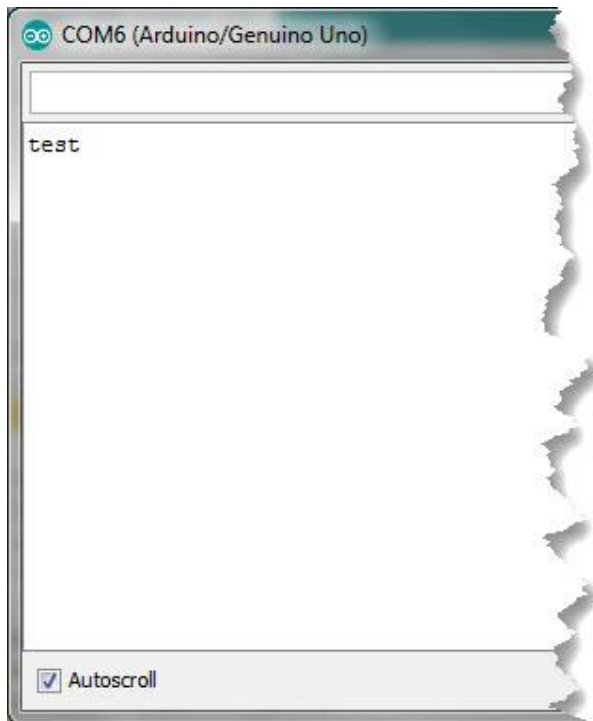
1. Go to **C:\Program Files (x86) > Arduino** and click **Arduino.exe**.
2. Return to the **BlinkLEDwithPushButton** diagram.
3. Check if data can be printed to the serial monitor by doing the following:
 - a. Add an **Extern Function** block to the diagram and call the functions **Serial.begin(9600)**; **Serial.println("test")**.



- b. Encapsulate the **Extern Function** block in a compound block named **Serial Debug** and activate **Enabled Execution**.



- c. Wire a **variable** block set to **\$firstPass** into the **Serial Debug** block.
 - 4. [Generate code to run on the Arduino.](#)
 - 5. Switch to the **Arduino IDE** and click on **Tools > Serial Monitor**.
- The monitor window displays the word *test*, which shows that data communication is working.



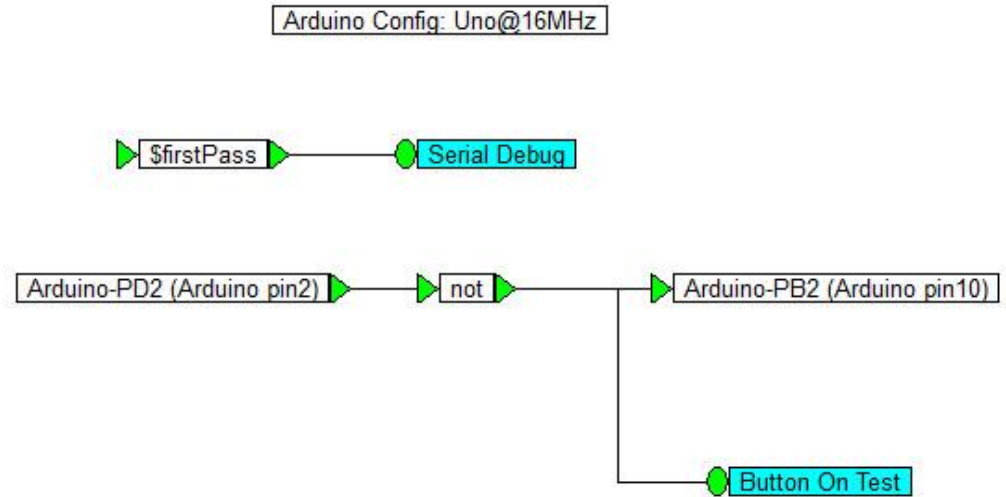
6. Close the **serial monitor**.

Confirm that the pushbutton is working

1. Wire a **Boolean not** block into the diagram:

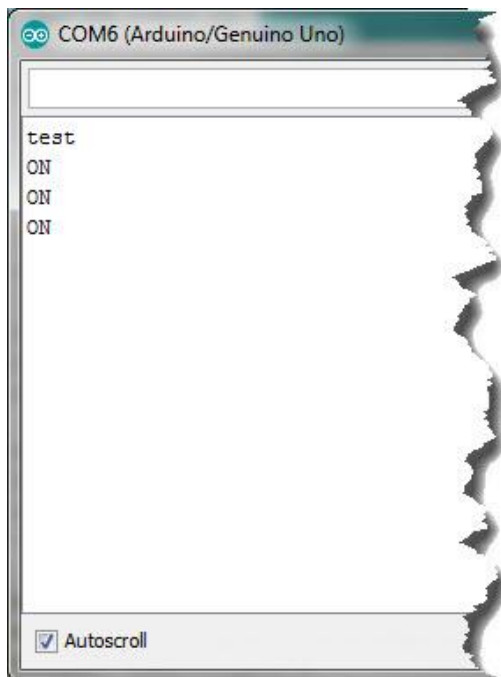


2. Add an **Extern Function** block that calls the function `Serial.println("ON")`.
3. Encapsulate the **Extern Function** block in a compound block named **Button On Test** and activate **Enabled Execution**.
4. Wire the **Boolean not** to the **Button On Test** block.



5. [Generate code to run on the Arduino.](#)
6. Switch to the **Arduino IDE** and click on **Tools > Serial Monitor**.
7. Press the **pushbutton** on the sensor.

The word *ON* is displayed in the serial monitor after each press, which confirms that the pushbutton is working correctly.



8. Close the **serial monitor**.

Check diagram parameters

1. **Right-click** on each block and check that the parameter values are set correctly.
2. The **Channel** parameter for the **Digital Output for Arduino** block is incorrectly set to 2. Set it to **5**, which corresponds to **Uno Pin 13**, the built-in LED.

3. [Generate code to run on the Arduino.](#)
4. Press the **pushbutton** on the sensor.

The built-in LED now blinks each time you press the pushbutton.

Debugging real-time analog waveforms using the Arduino serial port

Two diagrams are included with Embed that show how to use the Arduino serial port for debugging real-time analog waveforms. The diagrams are located under Examples > Embedded > Arduino > Application.

- **genPlotData-Uno:** Sends three channels of packetized waveform data to the serial port
- **serialPlot:** Reads up to eight channels of packetized waveform data from the serial port. It uses the Serial Read (under Blocks > Real Time) to read and parse the data into a vector, then convert the vector to scalar and plot the data in real time.

To see how to use these diagrams for debugging:

1. Attach your Arduino to your computer.
2. Open the **genPlotData-Uno** diagram.
3. Make sure you select the proper COMM port in the **Arduino Config** block.
4. [Compile and flash](#) **genPlotData-Uno** to the Arduino target.
5. Open the **serialPlot** diagram.
6. Right-click the **Serial Read** block and confirm the COMM port is properly set.
7. Run the diagram.

You can change the rate or end time of each run using the Simulate > System Properties command.

Running generated code on HIL hardware

The Target Interface block downloads OUT or ELF files generated with the [Use selected compound edge pins for data exchange](#) parameter at the start of the simulation. The Target Interface block maintains connection to the target and sends and receives data in real time. The data is sent over JTAG, UART or Ethernet depending on the target. The diagram containing the Target Interface block is typically set to run in real-time, with auto restart and retain state set so that debug sessions run until the stop button is clicked.

To perform HIL, you use the [Target Interface block in a debug diagram](#). Place the blocks you wish to debug (typically your controller subsystem) into a single compound block and compile it to an OUT or ELF file. The code generator automatically generates code for the HIL data exchange.

To run the generated OUT or ELF file

- Replace the compound block in your diagram with the Target Interface block. When you place a target Interface block in the diagram, it will automatically be set up with the path to the generated OUT or ELF file.

Integrating handwritten code with generated code

There are two ways to integrate handwritten code with Embed-generated code:

- Make Callable from User App option

- Use the Extern Read and Extern Write blocks
- Use the Extern Definition and Extern Function blocks

Regardless of the method you use, to run independently but communicate with Embed, use the Extern Read and Extern Write blocks.

Calling the generated code from a user application

You supply the operating environment and call Embed-generated code as a task using the [Call from Foreign RTOS/User App](#) parameter in the Code Generation dialog box. This assumes that you are handling timers and interrupts and will call the Embed function at the designed control interrupt rate. You can include the generated C file in a Code Composer project.

Using Extern Read and Extern Write blocks to merge your code

Embed has [Extern Read](#), [Extern Write](#) and [Extern Function](#) blocks to read and write external variables, as well as to call external functions.

To merge your code

1. Use [Extern Read](#) and [Extern Write](#) blocks to read and write external variables.
2. Use [Extern Function](#) blocks to call your functions. You can specify parameters as input pins to the block and access the function result as an output pin on the block.
3. Add your OBJ file to the *targetCL.BAT* file (for example, F280xcl.bat) found in the `\<install-path>\CG` directory. To add your OBJ file to the BAT file, open the BAT file in Notepad and type the path to your OBJ after “set USER_OBJS=” in the first line of the file. If you have more than one file, separate them with commas.

To run your initialization code, put the code in a function called `userStartup()`. This function is called by Embed-generated code at boot time. In this function, you can install your interrupt handler and perform any other initialization task.

Using Extern Definition and Extern Function blocks to add a C function to your diagram

Embed lets you easily convert a C function into an Embed block and add it to the User Block menu. To convert a C function, you use the [Extern Definition](#) and [Extern Function](#) blocks. To add the newly-created Embed block to the User Block menu, you include it in the Addons list under Edit > Preferences. For step-by-step instructions, watch this [online video](#).

Generating code from custom blocks

You can write a custom block in Embed using the Embed add-on API. The Embed install supplies a Microsoft Visual C wizard that will create a project for you. This lets you run pure simulations on the PC and when code generation is requested, your code is emitted in the generated code; Embed sends a message to your DLL at code generation time requesting a code generation string if the standard call is not sufficient. The code generation string may contain %n references that will be expanded by the code generator to the expressions that represent the values flowing into the nth connector.

Event logging

Embed provides a simple function for logging events. You are encouraged to write your own function to send events to an embedded display device of your choice or send them to a log file. The source code for the supplied function is provided below:

```

/*****
 * (C) Copyright 1989-2021 Altair Engineering *
 * All Rights Reserved *
 * This software may not be used, copied or made available to anyone, *
 * except in accordance with the license under which it is furnished. *
 *****/

#include <stdio.h>
#include <stdlib.h>
#include "vsuser.h"

// VisSim generates this call for each event log block
// Write your own version of this function to display log events on your device,
// or log events to a file
// eventType corresponds to the type input, eventName corresponds to the event
input
// eventClass corresponds to the "Label" field in the eventLog dialog.
void vsmLogEvent( int eventType, const char *eventName, const char *eventClass)
{
  const char *messageType="??";
  switch (eventType) {
  case tvsmEventMessage:
    messageType = "Info";
    break;
  case tvsmEventWarning:
    messageType = "Warning";
    break;
  case tvsmEventError:
    messageType = "Error";
    break;
  }

  printf( "%s\t", messageType );
  if (!eventClass)
    eventClass = "Unclassified";
  printf( "%s: %s\n", eventClass, eventName);
}

```


Using the Target Support Blocks and Commands

Using the target support blocks

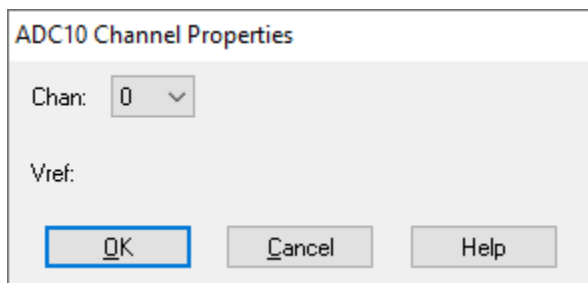
Embed provides on-chip peripheral blocks to allow you to program the following devices and boards:

- **AMD64**
- **Arduino:** Leonardo, Mega2560, Uno
- **Raspberry Pi:** Zero, Zero W, 1A+, 1B+, 2B, 3A+, 3B, 3B+, 4B
- **STMicroelectronics:** STM32 F0x, F3x, F4x, F7x, G0x, G4x, H7x, L4x
- **Texas Instruments:** ARM Cortex M3, C2000 (C2407, Delfino, F280x, F281x, Piccolo), MSP430

ADC10/12

Target Category: MSP430

Description: The MSP430 comes with either an ADC 10-bit precision or ADC 12-bit precision module. The ADC10/12 block allows you to acquire analog data from an MSP430 and convert it into digital data. When you insert an ADC10/12 block into your diagram, it automatically assumes either 10- or 12-bit precision based on the CPU subtype you selected in the dialog box for [MSP430 Config](#).



The image shows a dialog box titled "ADC10 Channel Properties". It features a "Chan:" label with a dropdown menu currently set to "0". Below this is a "Vref:" label with an empty text input field. At the bottom of the dialog are three buttons: "OK", "Cancel", and "Help".

Channel: Indicates a channel number.

Vref: Indicates the reference voltage for the unit. If you have selected a CPU subtype with ADC 10-bit precision on the MSP430, the Vref text box is dimmed.

Analog Comparator DAC

There are different dialog boxes for the Analog Comparator DAC block, depending on your Texas Instruments device. Older devices, like the F2808 do not have a comparator subsystem. Newer devices, like the F280049 have the subsystem.

To compare voltages on an MSP430, use the [Comparator](#) block.

Analog Comparator DAC – No Comparator Subsystem

Target Category: Delfino, F280x, Piccolo

Target Sub-Category: DAC

Description: The Analog Comparator DAC block lets you set up the on-chip comparator that compares an on-chip voltage reference to an external pin.

The Analog Comparator DAC block lets you set the value of the DAC on the comparator unit. It takes an input value between 0 and 1 in FX6.16 format. The voltage produced by the Analog Comparator DAC is the input value * VRefDac.

Drive DAC with Ramp: If input B is selected to be internal DAC, then you can change Drive DAC with ramp.

Input A: Indicates the voltage to be compared on input A. The voltage must be the same as what is presented on ADCINA2.

Input B: Indicates the voltage to be compared on input B. The voltage must be the same as what is presented on internal DAC.

Invert Output: Controls whether the output is inverted.

Mux Pins: Selects the pin that a given function is on.

Output Synchronization: Indicates how often the comparator results are updated and how long the comparator results must be consistent before a change occurs.

Put COMPxOUT on block pin: Indicates that the results of the comparator can be put on a pin.

Ramp Sync Source: Indicates when the ramp goes back to zero for synchronization.

Unit: Indicates the comparator unit.

Analog Comparator DAC - Comparator Subsystem (CMPSS)

Target Category: Delfino, Piccolo

Target Sub-Category: DAC

Description: The Analog Comparator DAC block lets you set up the on-chip comparator that compares an on-chip voltage reference to an external pin.

The Analog Comparator DAC block lets you set the value of the DAC on the comparator unit. It takes an input value between 0 and 1 in FX6.16 format. The voltage produced by the Analog Comparator DAC is the input value * VRefDac.

COMP

COMP+: See Texas Instruments documentation for detailed information on this parameter.

COMP-: See Texas Instruments documentation for detailed information on this parameter.

TRIPH: See Texas Instruments documentation for detailed information on this parameter.

TRIPOUTH: See Texas Instruments documentation for detailed information on this parameter.

Filter Samp Wire: See Texas Instruments documentation for detailed information on this parameter.

Filter Threshold: See Texas Instruments documentation for detailed information on this parameter.

Filter CLK Div: See Texas Instruments documentation for detailed information on this parameter.

Invert Output: Controls whether the output is inverted.

COMPL

COMPL+: See Texas Instruments documentation for detailed information on this parameter.

COMPL-: See Texas Instruments documentation for detailed information on this parameter.

TRIPL: See Texas Instruments documentation for detailed information on this parameter.

TRIPOUTL: See Texas Instruments documentation for detailed information on this parameter.

Filter Samp Win: See Texas Instruments documentation for detailed information on this parameter.

Filter Threshold: See Texas Instruments documentation for detailed information on this parameter.

Filter CLK Div: See Texas Instruments documentation for detailed information on this parameter.

Invert Output: Controls whether the output is inverted.

DACxVALA Load On: See Texas Instruments documentation for detailed information on this parameter.

Hysteresis: Indicates the lag introduced to the system.

Sync Source: See Texas Instruments documentation for detailed information on this parameter.

Unit: Indicates the comparator unit.

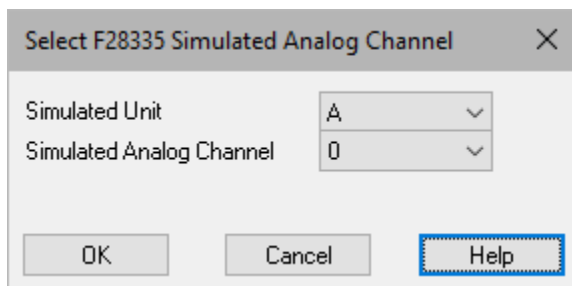
Vref: See Texas Instruments documentation for detailed information on this parameter.

Analog In

Target Category: Arduino, C2407, Cortex M3, Delfino, F280x, F281X, Generic MCU, MSP430, Piccolo, STM32

Target Sub-Category: Sim

Description: The Analog In block receives simulated analog data from the diagram during a simulation. The [Analog Input](#) block produces the data sent to the Analog In block.



Simulated Analog Channel: Indicates the analog channel to be simulated. Click [here](#) for Arduino pin mapping.

Simulated Unit: Indicates the unit. This parameter is available for Texas Instruments dual core, F280025, and F280049 targets.

Unit: Indicates the unit. This parameter is available for STM32 targets.

Analog Input

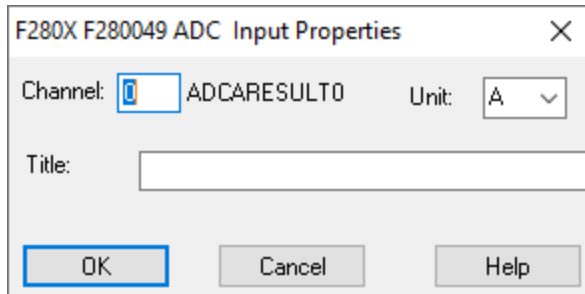
Analog Input for Arduino, C2407, Delfino, F280x, F281X, Piccolo

Target Category: Arduino, C2407, Delfino, F280x, F281X, Piccolo

Target Sub-Category: ADC

Description: The Analog Input block receives analog data from the ADC peripheral on the embedded target. During simulation, Analog Input produces the data supplied by the corresponding [Analog In](#) block.

Note that the block output type is scaled integer and holds fractional values between 0 and 1, where 1 corresponds to a full-scale ADC result regardless of the bits in the ADC converter.



Channel: Indicates the channel number. Click [here](#) for Arduino pin mapping.

Title: Indicates the channel title.

Unit: Indicates the unit number. This parameter is not available for Arduino, C2407, and F281X targets.

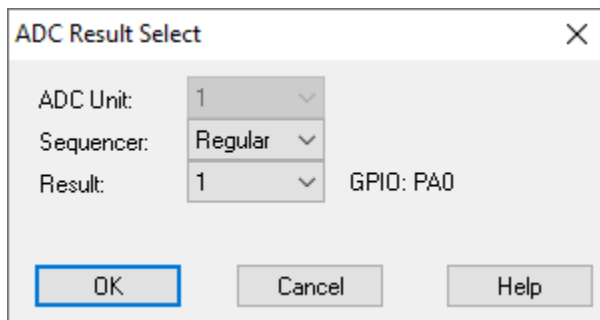
Analog Input for STM32

Target Category: STM32

Target Sub-Category: ADC

Description: The Analog Input block receives analog data from the ADC peripheral on the embedded target. During simulation, Analog Input produces the data supplied by the corresponding [Analog In](#) block.

Note that the block output type is scaled integer and holds fractional values between 0 and 1, where 1 corresponds to a full-scale ADC result regardless of the bits in the ADC converter.



ADC Unit: Indicates the unit number.

Result: Indicates the index into the result vector. The input channel is displayed to the right of the dropdown. To change the input channel for a sequencer result, use the [STM32 ADC Config](#) block.

Sequencer: Indicates a regular or injected mode sequencer. In injected mode, an ADC conversion can be injected — using an external trigger — during the conversion of regular channels. In motor control applications, this is used to delay conversion until after completion of an event — such as transistor switching — so that conversion noise is reduced. An injected conversion has higher priority than a regular conversion and thus interrupts regular conversions.

CAN Receive

Target Category: C2407, Cortex M3, Delfino, F280x, F281X, Piccolo, STM32

Target Sub-Category: CAN

Description: The CAN Receive block implements a controller area network version 2.0. This block generates code to receive up to four 2-byte integers from a CAN bus. Use [CAN Config](#) to configure the network.

CAN Device: Indicates the CAN device.

Data Pins: Controls the number of 2-byte integers to be received.

Data Pin Configuration

Byte Offset into CAN Package: Specifies the byte offset into the 8-byte packet. Offset 0 starts with the first element in the packet.

Pin: Indicates the pin to be configured.

Radix Point: Sets the binary point.

Type: Specifies the data type. Scaled_Int is 16- or 32-bit, depending on word size.

Word Size: Specifies the word size in bits.

Mailbox Number: Defines the mailbox to be used.

Masking Register: Represents a hexadecimal number used to mask off a portion of the identifier that would normally be used to match a CAN message identifier. For STM32 targets, the masking register is optional. In this case, to use it, you must activate the **Use Mask** parameter.

Message ID: Indicates the CAN message ID used to specify the network address.

Mux Pin: Selects the pin which a given function is on.

Note: Some F280x and MSP430 devices have different functions for the same physical pin on the chip. This is referred to as multiplexing, or muxing, for short, and is done because pins are expensive. Because multiple functions compete for a given pin, you must choose what function a pin has. For flexibility, in some cases Texas Instruments provides multiple possible pins for a given function. For instance, the CANTXB function can be on pin 8, 12, or 16. Pin 8 is shared with ePWM5A and ADCSOCA0; pin 12 is shared with TZ1 and SPISIMOB; and pin16 is shared with SPISIMOA and TZ5. If you want ePWM5A on a pin, you cannot use pin 8 for CANTXB, but rather pin 12 or 16.

Number of Outputs: Controls the number of 2-byte integers to be received.

Receive extended frames: When activated, uses CAN 2.0 29-bit identifiers. When **Receive extended frames** is not activated, it uses 11-bit identifiers.

Set address dynamically: Changes address from which to receive as the algorithm runs.

Use Mask: For STM32 targets, you must activate **Use Mask** in order to enter the masking value.

CAN Transmit

Target Category: C2407, Cortex M3, Delfino, F280x, F281X, Piccolo, STM32

Target Sub-Category: CAN

Description: The CAN Transmit block implements a controller area network (CAN) version 2.0. This block generates code to transmit up to four 2-byte integers from a CAN bus. Use [CAN Config](#) to configure the network.

CAN Transmit Properties

CAN Device:

Data Pins:

Bytes in transfer:

Mailbox Number:

Message ID (11 bits):

Mux Pin:

Send Extended Frames

Remote Transmission Request

Auto Answer Mode

Enable Transmit Pin

Set Address Dynamically

Set Data Length Dynamically

Data Pin Configuration

Pin: Type:

Radix Point: Word Size:

Byte offset into CAN packet:

Auto Answer Mode: Only for boxes 2 and 3.

Data Pins: Controls the number of 2-byte integers to be transmitted.

Data Pin Configuration

Byte Offset into CAN Package: Specifies the byte offset into the 8-byte packet. Offset 0 starts with the first element in the packet.

Pin: Indicates the pin to be configured.

Radix Point: Sets the binary point.

Type: Specifies the data type. Scaled_Int is 16- or 32-bit, depending on word size.

Word Size: Specifies the word size in bits.

CAN Device: Indicates the CAN device.

Enable Transmit Pin: Adds an input pin for the CAN address to which you want to transmit. This parameter must be activated.

Mailbox Number: Represents a hexadecimal number used to mask off a portion of the identifier that would normally be used to match a CAN message identifier.

Message ID: Indicates the CAN message ID used to specify the network address.

Mux Pin: Selects the pin which a given function is on.

Note: Some F280x and MSP430 devices have different functions for the same physical pin on the chip. This is referred to as multiplexing, or muxing, for short, and is done because pins are expensive. Because multiple functions compete for a given pin, you must choose what function a pin has. For flexibility, in some cases Texas Instruments provides multiple possible pins for a given function. For instance, the CANTXB function can be on pin 8, 12, or 16. Pin 8 is shared with ePWM5A and ADCSOCA0; pin 12 is shared with TZ1 and SPISIMOB; and pin16 is shared with SPISIMOA and TZ5. If you want ePWM5A on a pin, you cannot use pin 8 for CANTXB, but rather pin 12 or 16.

Number of Inputs: Controls the number of 2-byte integers to be transmitted.

Remote Transmission Request: Sends non-data packet to message ID. If the box at message ID is in auto answer mode, it uses an immediate response.

Send Extended Frames: Allows use of 29-bit identifier.

Set address dynamically: Changes address to transmit to as the algorithm runs.

CAN Transmit Ready

Target Category: C2407, Cortex M3, Delfino, F280X, F281X, Piccolo, STM32

Target Sub-Category: CAN

Description: The CAN Transmit Ready block lets you know when the transmit function is available. Use [CAN Config](#) to configure the network.

The image shows a dialog box titled "CAN Transmit Ready Properties". It has two dropdown menus. The first is labeled "CAN Device:" and has "A" selected. The second is labeled "Mailbox Number:" and has "4" selected. At the bottom of the dialog are three buttons: "OK", "Cancel", and "Help".

CAN Device: Indicates the CAN device.

Mailbox Number: Defines the mailbox to be used.

Comparator

Target Category: MSP430

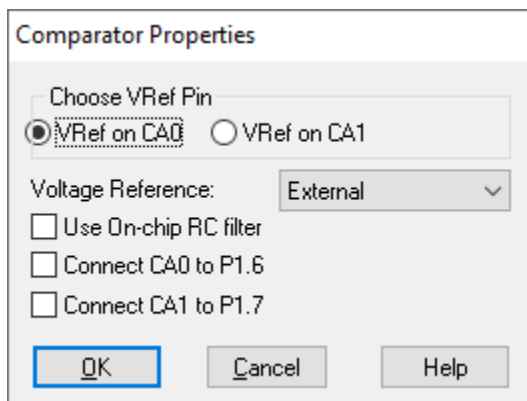
Sub-Target Category: ADC

Description: The Comparator block compares two voltages. It supports precision slope analog-to-digital conversions, supply voltage supervision, and monitoring of external analog signals. Features of the Comparator block include:

- Inverting and non-inverting terminal input multiplexer
- Software selectable RC-filter for the comparator output
- Output provided to Timer_A capture input
- Software control of the port input buffer
- Interrupt capability
- Selectable reference voltage generator
- Comparator and reference generator can be powered down

For more information on the Comparator block, see the MSPUM430.PDF file included with your software.

To compare voltages on a Delfino, F280x, or Piccolo, see [Analog Comparator DAC](#) block.



Choose VRef Pin: Selects the voltage reference pin.

Connect CA0 to P1.6: See the MSPUM430.PDF file included with your software.

Connect CA1 to P1.7: See the MSPUM430.PDF file included with your software.

Use On-Chip RC filter: See the MSPUM430.PDF file included with your software.

Voltage Reference: See the MSPUM430.PDF file included with your software.

DAC

DAC for Delfino, F280x, Piccolo

Target Category: Delfino, F280x, Piccolo

Description: The DAC block performs digital to analog conversions.

The screenshot shows the 'DAC Properties' dialog box. It includes the following fields and controls:

- Unit:** A dropdown menu with 'A' selected.
- Output Pin:** A text input field containing 'A0'.
- Sync Update:** An unchecked checkbox.
- Sync Source:** A dropdown menu with 'PWMSYNC1' selected.
- Gain Mode:** A dropdown menu with '1x' selected.
- Voltage Reference:** A dropdown menu with 'VDAC/VSSA' selected.
- Buttons:** 'OK', 'Cancel', and 'Help' buttons are located at the bottom.

Gain Mode: Selects the output gain with respect to the reference voltage. This parameter is available only for the Texas Instruments F28377.

Output Pin: Specifies the output pin.

Sync Source: Determines which PWM sync will cause the DAC block input to be presented on the hardware. This parameter is available only for the Texas Instruments F28377.

Sync Update: Specifies the sync. This parameter is available only for the Texas Instruments F28377.

Unit: Specifies the unit.

Voltage Reference: Chooses the reference, voltage, and ground. This parameter is available only for the Texas Instruments F28377.

DAC for STM32

Target Category: STM32

Description: The DAC block performs digital to analog voltage conversions.

Unit: Specifies the DAC to be controlled.

OUT1

Pin: Specifies the metal pin on the STM32 device. You have these options:

- **Buffer:** When driving the metal pin, you can buffer the signal, which means the chip is supplying more power.
- **Internal Connection:** Lets you use the voltage internally for a comparator.

Use Trigger Updates: Activate this parameter to use a trigger to control when Embed updates the value on the pin.

Trigger: Specifies the signal that will trigger a conversion.

Add Signal: Lets you add noise or a triangular wave signal to the base DC value. This is a common technique for some digital power control conversions.

Amplitude: Specifies how many bits of noise or triangular wave to add to the signal.

Sample Hold: Performs a sample and hold operation on the signal. This is a power conservation feature available on low-power devices. You control the sample and hold in the following ways:

- **Sample Ticks:** Specifies the number of ticks to sample the initial signal.
- **Hold Ticks:** Specifies the number of ticks to hold the signal before a refresh.
- **Refresh Ticks:** Specifies the number of ticks from digital to analog. Typically, the value is much smaller than the initial Sample Tick value.

OUT2

Pin: Specifies the metal pin on the STM32 device. You have these options:

- **Buffer:** When driving the metal pin, you can buffer the signal, which means the chip is supplying more power.

- **Internal Connection:** Lets you use the voltage internally for a comparator.

Use Trigger Updates: Activate this parameter to use a trigger to control when Embed updates the value on the pin.

Trigger: Specifies the signal that will trigger a conversion.

Add Signal: Lets you add noise or a triangular wave signal to the base DC value. This is a common technique for some digital power control conversions.

Amplitude: Specifies how many bits of noise or triangular wave to add to the signal.

Sample Hold: Performs a sample and hold operation on the signal. This is a power conservation feature available on low-power devices. You control the sample and hold in the following ways:

- **Sample Ticks:** Specifies the number of ticks to sample the initial signal.
- **Hold Ticks:** Specifies the number of ticks to hold the signal before a refresh.
- **Refresh Ticks:** Specifies the number of ticks from digital to analog. Typically, the value is much smaller than the initial Sample Tick value.

DAC12

Target Category: MSP430

Target Sub-Category: DAC

Description: The DAC12 block supports the MSP430 DAC12 peripheral, which performs 12-bit digital to analog conversions. For background information on the DAC12, see the MSP430x4x-SLAU056F.PDF file contained on your installation disk.

Chan: Selects the DAC12 module to be used.

Gain: Selects the output gain with respect to the reference voltage.

Group with next higher channel: Allows you to group together multiple DAC12s with the DAC12GRP bit to synchronize the update of each DAC12 output. Hardware ensures that all DAC12 modules in a group update simultaneously independent of any interrupt or nonmaskable interrupt (NMI) event.

Load Select: Select the trigger for updating the DAC12 voltage output.

Power Level: Select the power level. The more power you use, the faster the DAC12 module responds.

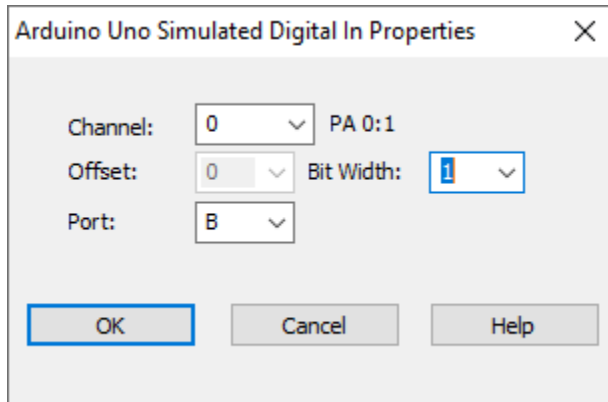
Resolution: Selects 8-bit or 12-bit voltage output resolution.

Digital In

Target Category: Arduino, CortexM3, MSP430

Target Sub-Category: Sim

Description: The values presented to the input of the Digital In block will appear on the output of the actual Digital Input block during a simulation.



Bit Width: Specifies the number of contiguous bits to read in.

Channel: Specifies the channel. Click [here](#) for Arduino pin mapping

Offset: Specifies the offset into the digital port register.

Port: Specifies the digital register. Click [here](#) for Arduino pin mapping.

Digital Input

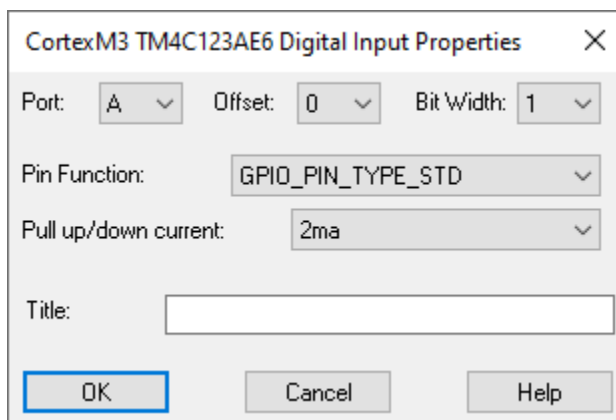
Digital Input for Cortex M3, MSP430

Target Category: Cortex M3, MSP430

Target Sub-Category: Digital I/O

Description: The Digital Input block receives digital data.

The Digital Input block is useful for [hardware-in-the-loop simulation](#).



Bit Width: Specifies the bit width.

Offset: Specifies the offset.

Pin Function: Specifies the pin function.

Port: Specifies the port.

Pull up/down current: Lets you specify the pin draw current in ON or OFF mode. This parameter is typically used for communication protocols, like I2C. This parameter is available only for the Cortex M3 target.

Title: Indicates the channel title.

Digital Input for Arduino

Target Category: Arduino

Target Sub-Category: Digital I/O

Description: The Digital Input block receives digital data.

Bit Width: Specifies the bit width.

Channel: Indicates the channel number. Click [here](#) for Arduino pin mapping.

Enable pull-up resistor: Enables the pull-up resistor on the input pin to 3.3V.

Offset: Specifies the offset.

Port: Specifies the port. Click [here](#) for Arduino pin mapping.

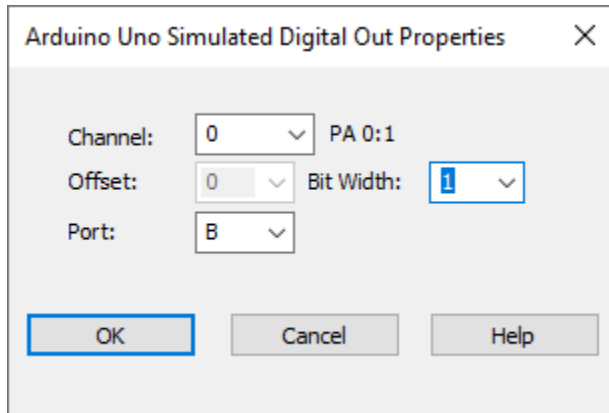
Title: Indicates the channel title.

Digital Out

Target Category: Arduino, Cortex M3, MSP430

Target Sub-Category: Sim

Description: The input values of the actual Digital Output block will appear on the output of the Digital Out block during a simulation.



Bit Width: Specifies the number of contiguous bits to read in.

Channel: Specifies the channel. Click [here](#) for Arduino pin mapping.

Offset: Specifies the offset into the digital port register.

Pin Function: Specifies the pin function.

Port: Specifies the digital register. Click [here](#) for Arduino pin mapping.

Digital Output

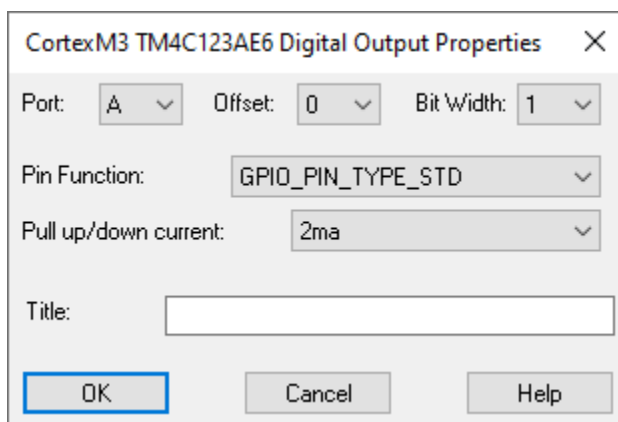
Digital Output for Cortex M3, MSP430

Target Category: Cortex M3, MSP430

Target Sub-Category: Digital I/O

Description: The Digital Output block writes digital data.

The Digital Output block is useful for [hardware-in-the-loop simulation](#).



Bit Width: Specifies the bit width.

Offset: Specifies the offset.

Pin Function: Specifies the pin function.

Port: Specifies the port.

Pull up/down current: Lets you specify the pin draw current in ON or OFF mode. This parameter is typically used for communication protocols, like I2C. This parameter is available only for the Cortex M3 target.

Title: Indicates the channel title.

Digital Output for Arduino

Target Category: Arduino

Target Sub-Category: Digital I/O

Description: The Digital Output block writes digital data.

Bit Width: Specifies the bit width.

Channel: Indicates the channel number. Click [here](#) for Arduino pin mapping.

Port: Specifies the port. Click [here](#) for Arduino pin mapping.

Title: Indicates the channel title.

DMA Enable

Targets: Delfino, F281X, Piccolo, STM32

Target Sub-Category: DMA

Description: The DMA Enable block lets you turn Direct Memory Access (DMA) on and off. DMA Enable writes or reads data to memory automatically while the CPU is performing other tasks. Use [DMA Config](#) to configure the DMA.

DMA Channel: Specifies the channel.

eCAP

Target Category: Delfino, F280x, Piccolo

Target Sub-Category: Capture

Description: The eCAP block lets you configure the enhanced capture peripheral. Input to the eCAP block must have well-defined logic edge transitions to trigger an event. The eCAP unit will record time between events. Up to four sequential event intervals can be tracked.

You can choose the maximum number of events to track. To track both the on time and off time of a PWM signal, set Max Events to 2, Event 1 to Trigger on Falling Edge, and Event 2 to Trigger on Rising Edge. Then activate the corresponding Reset Counter on Capture. With this setup, event pin 1 will produce PWM on time, and event 2 pin will produce PWM off time.

Additional information:

- [How to capture a PWM signal with eCAP block](#)
- Texas Instruments [SPRU807-eCAP](#) document

Capture Unit: Specifies the unit to be configured.

Event1...Event4: Indicates the four independent edge polarity (rising edge/falling edge) selections, one for each capture event.

Input Prescale: Indicates the amount of prescaling.

Max Events: Indicates the maximum number of capture events to be tracked.

Mux Pin: Selects which pin a given function is on.

Note: Some F280x and MSP430 devices have different functions for the same physical pin on the chip. This is referred to as multiplexing, or muxing, for short, and is done because pins are expensive. Because multiple functions compete for a given pin, you must choose what function a pin has. For flexibility, in some cases Texas Instruments provides multiple possible pins for a given function. For instance, the CANTXB function can be on pin 8, 12, or 16. Pin 8 is shared with ePWM5A and ADCSOCA0; pin 12 is shared with TZ1 and SPISIMOB; and pin16 is shared with SPISIMOA and TZ5. If you want ePWM5A on a pin, you cannot use pin 8 for CANTXB, but rather pin 12 or 16.

Reset Counter on Capture: Resets the counter for the specified event.

eCap PWM

Target Category: Delfino, F280x, Piccolo

Target Sub-Category: Capture

Description: The eCap PWM block takes the ecapture unit and configures it as a PWM.

Additional information: Texas Instruments [SPRU791A](#) document.

Capture Unit: Specifies the unit to be configured.

Change Phase Dynamically: Changes the phase dynamically.

CTRPHS (phase): The phase register is loaded into the counter register when a synchronization pulse is received from the prior unit. The phase register can be continuously updated if desired.

Note: Some F280x and MSP430 devices have different functions for the same physical pin on the chip. This is referred to as multiplexing, or muxing, for short, and is done because pins are expensive. Because multiple functions compete for a given pin, you must choose what function a pin has. For flexibility, in some cases Texas Instruments provides multiple possible pins for a given function. For instance, the CANTXB function can be on pin 8, 12, or 16. Pin 8 is shared with ePWM5A and ADCSOCA0; pin 12 is shared with TZ1 and SPISIMOB; and pin16 is shared with SPISIMOA and TZ5. If you want ePWM5A on a pin, you cannot use pin 8 for CANTXB, but rather pin 12 or 16.

Mux Pin: Selects which pin a certain function is on.

Period: Specifies the number of cycles (counter increments) of the board's system clock to reach the end of its period. The carrier frequency of the PWM signal is determined as follows:

$$(\text{System-Clock-Frequency})/(\text{Timer-Frequency}) = \text{Carrier-Frequency}$$

For example, 150MHz/150 = 1MHz.

Polarity: Controls the active high and active low.

Sync-Out: Each eCap PWM block is potentially linked to the next highest one. The link is via a synchronization pulse to load the counter with the contents of the CTRPHS phase register. The pulse can either be passed through the previous unit or generated when the counter equals the period (PRD).

ePWM

Target Category: Delfino, F280x, Piccolo

Target Sub-Category: PWM

Description: The ePWM block lets you configure the enhanced PWM unit. The dynamic phase input operates on the current scaled period.

The Duty Cycle input pins control the duty cycle of the PWM waveform. They are at scaled 1.16. The input value is multiplied by the PWM period and assigned to the PWM compare register to generate a fractional duty cycle. Thus, an input of 0.5 (fx1.16) yields a 50% duty cycle; an input of zero yields 0% duty cycle; and an input of 0.99997 (the largest possible positive value in 1.16 notation) yields 100% duty cycle.

To mirror the input pins on the ePWM with the corresponding output pins on [ePWM for Sim](#) block, activate **Use CMPC/CMPD, TBCTR=TBPHS on SYNCI pulse, Change Phase Dynamically**, and **Change Period Dynamically** in the ePWM Properties dialog box.

You can add an input pin that enables the unit externally. A corresponding outpin pin on the [ePWM for Sim](#) block is also present.

Interactive mode: In this mode, both Duty Cycle input pins are active; however, all parameters in the dialog box are inactive except the PWM Unit parameter.

Additional information:

- [ePWM overview](#)
- [How to make PWM frequency as low as 50Hz](#)
- [How to generate 3-phase PWM for 3-phase inverter](#)
- [How to sync two PWM units 180o out of phase](#)
- [How to capture a PWM signal with eCAP block](#)
- Texas Instruments [SPRU791A](#) document

280x ePWM Properties

PWM Unit: Use High Res Timer Use CMPC/CMPD

Time Base
 Rate Scaling: Count Mode:

Timer Period: Change Period Dynamically

TBCTR=TBPHS on SYNCI pulse TBPHS (phase):
 Change Phase Dynamically EPWMSYNCO pin:

EPWMSYNCO: EPWMSYNCO pin:
 CMPA Load On: CMPB Load On:

Action Qualifier:

	CMPA		CMPB		P	GPIO Pin
	Z	up down	up down			
EPWMA:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="text" value="GPIO0"/>
EPWMB:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="text" value="GPIO1"/>

Deadband:
 Delay Mode:
 Polarity:
 Input Select:

Rising Edge Delay (0-1023): Falling Edge Delay (0-1023):

Send Start ADC Conversion Pulse A (SOCA):
 Send Start ADC Conversion Pulse B (SOCB):

Fault Handling
 EPWMA output on fault:
 EPWMB output on fault:

Add Enable Pin (0 value forces Fault)

One Shot TZx Fault Source: 1 2 3 4 5 6 DCA DCB
 CBC TZx Fault Source: 1 2 3 4 5 6 DCA DCB

TZ1: TZ2: TZ3:
 TZ4: TZ5: TZ6:

Action Qualifier

EPWMA and EPWMB: Determines which events are translated into specific actions that produce the required waveforms at EPWMA and EPWMB. Actions are generated based on the following events:

Z: counter = 0

CMPA Up: counter = CA during up count

CMPA Down: counter = CA during down count

CMPB Up: counter = CB during up count

CMPB Down: counter = CB during down count

P: counter = period

You can choose the following actions for any of the events:

X = Do nothing

0 = Force output to zero

1 = Force output to one

T = Toggle output

Deadband

Delay Mode: Indicates the deadband mode.

Falling Edge Delay Count: Indicates an independent value for falling edge delay.

Input Select: There are two inputs to the Deadband unit: PWMA and PWMB. These two inputs are DbA and DbB. They can be driven by any combination of PWMA and PWMB.

Polarity: Indicates the polarity for the specified deadband.

Rising Edge Delay Count: Indicates an independent value for rising edge delay.

Fault Handling

Add Enable Pin: Adds an input pin that lets you enable or disable the unit externally.

Autoreset TZx Fault Source: Allows the reset of the fault on the next cycle of the PWM.

CBC TZx Fault Source: Enables a cycle-by-cycle trip zone fault and selects the pins that are used. The PWM is put in fault mode until the next cycle.

Digital Compare: Invokes the Digital Compare set up.

EPWMA Output on fault: Specifies the A response to the fault. Your choices are:

Fault Action Disabled: No response to the fault

Forced High: Electrical conductance to 3V

Forced Low: Electrical conductance to ground (0V)

High Impedance: No electrical conductance on the A output

EPWMB Output on fault: Specifies the B response to the fault. Your choices are:

Fault Action Disabled: No response to the fault

Forced High: Electrical conductance to 3V

Forced Low: Electrical conductance to ground (0V)

High Impedance: No electrical conductance on the A output

One Shot TZx Fault Source: Enables a one shot trip zone fault and selects the pins that are used. The PWM is put in fault mode until code is executed to bring it back up.

PWM Unit: Specifies the unit to be configured.

Send Start ADC Conversion Pulse A: Indicates whether the ePWM unit should send a SOC A (start conversion on Pulse A) to the ADC unit.

Send Start ADC Conversion Pulse B: Indicates whether the ePWM unit should send a SOC B (start conversion on Pulse B) to the ADC unit.

Time Base

Change Period Dynamically: Produces an external input pin (%Period (1.16)) that lets you specify a fractional value for the period on-the-fly. The external value will be multiplied by the period and assigned to the TBPRD (period) register. This allows you to dynamically modulate the PWM-based frequency.

Change Phase Dynamically: Produces an external input pin (%Phase (1.16)) that lets you specify a fractional value for the phase on-the-fly. The external value will be multiplied by the phase and assigned to the TBPHS (phase)

register. This allows you to dynamically modulate the PWM-based frequency. To activate **Change Phase Dynamically**, you must also activate **TBCTR=TBPHS on SYNCI pulse**.

CMPA/B Load On: Selects condition to cause the CMPA/B register to be loaded.

Count Mode: Determines the counting mode. Typically, Up/Down is selected.

EPWMSYNCI Pin: Indicates the hardware pin supplying the EPWMSYNCI input pulse for unit 1.

EPWMSYNCO: Indicates the synchronizing signal sent from this unit to unit n+1.

Rate Scaling: Scales the timer source to a slower rate.

TBCTR=TBPHS on SYNCI pulse: The counter for this unit is assigned the value of the TBPHS (phase) register on occurrence of a SYNCI input pulse.

TBPHS (phase): Specifies an offset value added to the time-based counter on occurrence of the SYNCI input pulse.

Timer Period: Specifies the number of PWM clock ticks for one complete PWM waveform. The frequency of the PWM signal is determined as follows:

$$\text{System-Clock-Frequency} / \text{Timer-Frequency} = \text{PWM-Frequency}$$

For example, 150MHz/150 = 1MHz.

Use CMPC/CMPD: Enables compare C and compare D inputs that can be used to send synchronization pulses to the ADC or DMA unit.

Use High Res Timer: Enables the use of a high-resolution timer.

ePWM digital compare

EPWM Digital Compare Properties

Unit: Use Blanking Window Filter (EVTFLT)

Event Src Sel (DCTRIPSEL)		Event Input Select (TZDCSEL)	
DCAL:	<input type="text" value="TZ1 input"/>	EVT1in:	<input type="text" value="Event disabled"/>
DCAH:	<input type="text" value="TZ1 input"/>	EVT2in:	<input type="text" value="Event disabled"/>
DCBL:	<input type="text" value="TZ1 input"/>	EVTB1in:	<input type="text" value="Event disabled"/>
DCBH:	<input type="text" value="TZ1 input"/>	EVTB2in:	<input type="text" value="Event disabled"/>

Event Filtering (EVTFLT)

Source: Invert Blank Window

Start Pulse:

Blanking Window Offset: 0 .. 65535 TBCLKs

Blanking Window Width: 0 .. 255 TBCLKs

Event Output Select (DCACTL/DCBCTL)

EVT1out:	<input type="text" value="EVT1in"/>	<input type="checkbox"/> sync TZ to SYSCLK
EVT2out:	<input type="text" value="EVT2in"/>	<input type="checkbox"/> sync TZ to SYSCLK
EVTB1out:	<input type="text" value="EVTB1in"/>	<input type="checkbox"/> sync TZ to SYSCLK
EVTB2out:	<input type="text" value="EVTB2in"/>	<input type="checkbox"/> sync TZ to SYSCLK

Output Configure (TZCTL/DCACTL/DCBCTL)

EVT1out>	<input type="text" value="EPWMA=Hi Z"/>	<input type="checkbox"/> ADC SOCA	<input type="checkbox"/> xSYNCl	<input type="checkbox"/> Interrupt
EVT2out>	<input type="text" value="EPWMA=Hi Z"/>			<input type="checkbox"/> Interrupt
EVTB1out>	<input type="text" value="EPWMB=Hi Z"/>	<input type="checkbox"/> ADC SOCB	<input type="checkbox"/> xSYNCl	<input type="checkbox"/> Interrupt
EVTB2out>	<input type="text" value="EPWMB=Hi Z"/>			<input type="checkbox"/> Interrupt

Event Filtering

Lets you specify the filter parameters. Activate **Use Blanking Window Filter** to use these parameters.

Blanking Window Offset: Specifies the number of timer counts before the blanking window occurs.

Blanking Window Width: Specifies the duration of the blanking window.

Source Event: Specifies the source event to be filtered.

Invert Blank Window: Inverts the blanking window.

Start Pulse: Specifies the start of the blanking window.

Event Input Sel: Specifies how the event is activated.

Event Output Select: Specifies whether the event is filtered. You can also synchronize the event to the system clock.

Event Src Sel: Lets you select inputs for four possible digital compare events. There are three possibilities: trip zone (TZn), comparator, or, in newer parts, a TRIPIN result. See Texas Instruments documentation for more information.

Output Configure

Specifies the output of the PWM upon occurrence of the event.

EVTa1out: Specifies the value produced by the PWM. You can also specify ADC SOCA, xSYNCl, and if an Interrupt occurs.

EVTa2out: Specifies the value produced by the PWM. You can also specify if an Interrupt occurs.

EVTb1out: Specifies the value produced by the PWM. You can also specify ADC SOCB, xSYNCl, and if an Interrupt occurs.

EVTb2out: Specifies the value produced by the PWM. You can also specify if an Interrupt occurs.

TRIPSEL Config: Lets you select a GPIO pin for each trip source.

Unit: Specifies the compare unit.

Use Blanking Window Filter: Selects the Blanking Window Filter.

ePWM TRIPSEL Config

Label	Value
TRIPIN1/TZ1:	GPIO0
TRIPIN2/TZ2:	GPIO0
TRIPIN3/TZ3:	GPIO0
TRIPIN4/XINT1:	GPIO0
TRIPIN5/XINT2:	GPIO0
TRIPIN6/XINT3:	GPIO0
TRIPIN7/ECAP1:	GPIO0
TRIPIN8/ECAP2:	GPIO0
TRIPIN9/ECAP3:	GPIO0
TRIPIN10/ECAP4:	GPIO0
TRIPIN11/ECAP5:	GPIO0
TRIPIN12/ECAP6:	GPIO0

Buttons:

ePWM Action

Target Category: Delfino, F280x, Piccolo

Description: The ePWM Action block lets you configure the action qualifier for the F280X. This lets you control the PWM waveform.

Interactive mode: The entire block is inactive in this mode.

Additional information: Texas Instruments [SPRU791A](#) document.

ePWM Action Qualifier: Each option represents a possible action the PWM unit may take based on the counter register state. There are two compare registers: CA and CB. Actions are generated based on the following events:

- CA Down:** counter = CA during down count
- CA Up:** counter = CA during up count
- CB Down:** counter = CB during down count
- CB Up:** counter = CB during up count
- P:** counter = period
- Z:** counter = 0

You can choose the following actions for any of the events:

- 0:** Force output to 0
- 1:** Force output to 1
- T:** Toggle output
- X:** Do nothing

ePWM Action Write

Target Category: Delfino, F280x, Piccolo

Description: The ePWM Action Write block lets you set the action qualifier for the ePWM unit by writing the input values to the appropriate hardware registers. The appropriate values come from the ePWM Action block.

Interactive mode: The entire block is inactive in this mode.

Additional information: Texas Instruments [SPRU791A](#) document.

Write Action Qualifier to ePWM Unit: Determines the unit to be configured.

ePWM Chopper

Target Category: Delfino, F280x, Piccolo

Target Sub-Category: PWM

Description: The ePWM Chopper block lets you configure the chopper unit for the ePWM.

Interactive mode: The entire block is inactive in this mode.

Additional information: Texas Instruments [SPRU791A](#) document.

The screenshot shows a dialog box titled "280x ePWM Chopper". It contains three dropdown menus: "ePWM Unit" set to "1", "Chop Frequency" set to "7.50 MHz", and "First Pulse Width" set to "133.33 nS". Below the dropdowns is a note: "Note: You will need to include an ePWM block in your diagram to create a PWM signal to chop." At the bottom are three buttons: "OK", "Cancel", and "Help".

Chop Frequency: Specifies the base frequency of the chopper.

ePWM Unit: Specifies the ePWM module to be configured.

First Pulse Width: Indicates an integer value from one to 16 to set the width of the first pulse. You specify this parameter to provide a high-energy first pulse to ensure hard and fast power switch turn on.

ePWM Force Action

Target Category: Delfino, F280x, Piccolo

Description: The ePWM Force Action block lets you force an output for the ePWM units. This is useful for applications like BLDC motor commutation.

Interactive mode: The entire block is inactive in this mode.

Additional information: Texas Instruments [SPRU791A](#) document.

The screenshot shows a dialog box titled "280x ePWM Force Action Properties". It contains two dropdown menus: "Force Output on A:" and "Force Output on B:", both set to "Forcing disabled". At the bottom are three buttons: "OK", "Cancel", and "Help".

Force Output on A: Performs the selected action:

Forcing Disabled: Forcing is disabled and the ePWM Force Action resumes normal operation.

Force High: Output is 1.

Force Low: Output is 0.

Force Output on B: Performs the selected action:

Forcing Disabled: Forcing is disabled and the ePWM Force Action resumes normal operation.

Force High: Output is 1.

Force Low: Output is 0.

ePWM Force Action Write

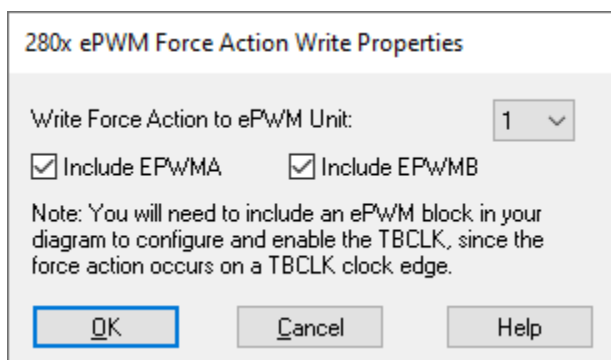
Target Category: Delfino, F280x, Piccolo

Target Sub-Category: PWM

Description: The ePWM Action Write block lets you set the force output mode for the ePWM unit by writing the input values to the appropriate hardware registers. The appropriate values come from the ePWM Force Action.

Interactive mode: The entire block is inactive in this mode.

Additional information: Texas Instruments [SPRU791A](#) document.



Include EPWMA: Specifies the ePWM output pin A.

Include EPWMB: Specifies the ePWM output pin B.

Write Force Action to ePWM Unit: Determines the unit to be configured.

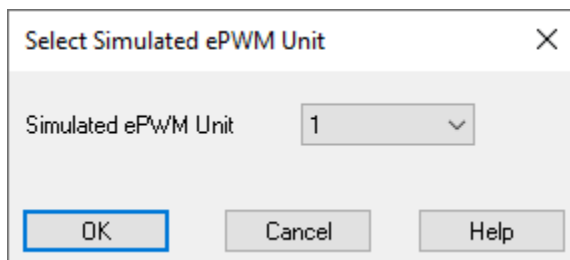
ePWM for simulation

Target Category: Delfino, F280x, Piccolo

Target Sub-Category: Sim

Description: The ePWM for Simulation block is used during a simulation to produce the value between zero and one that is currently being supplied to the actual [ePWM](#) block.

The ePWM for Simulation block has seven outputs. The A, B, C, and D outputs are four compare registers that represent the ratio of the compare register over the period. The period and phase outputs represent the duty cycle for the PWM. The enable output enables the unit externally. The outputs on the ePWM for Simulation block mirror the inputs on the [ePWM](#) block.



Simulated ePWM Channel: Indicates the ePWM channel to be simulated.

eQEP

Target Category: Delfino, F280x, Piccolo

Target Sub-Category: Quadrature Encoder

Description: The eQEP block lets you configure the enhanced quadrature encoder pulse.

Interactive mode: In this mode, only the QPOSCNT1 output pin is active, and all parameters in the dialog box are inactive except Capture Unit.

Additional information: Texas Instruments [SPRU791A](https://www.ti.com/lit/zip/SPRU791A) document.

Count Mode: For quadrature encoder behavior, select Quadrature. For other behaviors, see Texas Instruments documentation.

Gate Index Pulse: Enables gating of index pulse.

Inc/Dec Rev Count on Index Pulse: Enables the counting of index pulses.

Invert Index Pulse: Inverts the index pulse.

Invert QEPA input: Inverts the input on QEPA.

Invert QEPB input: Inverts the input on QEPB.

Invert Strobe: Inverts the strobe signal.

Max Position: Indicates the maximum position.

Mux Pin Assignment: Assigns a peripheral I/O port to a pin.

Position Counter Reset On: Indicates when the position counter is reset. The position counter can be configured to operate in four modes. In each of these modes, the position counter is reset to 0 on overflow and to QPOS MAX on underflow. For more information, see Texas Instruments documentation.

Index Event: If the index event occurs during the forward movement, then position counter is reset to zero on the next eQEP clock. If the index event occurs during the reverse movement, the position counter is reset to the value in the QPOS MAX register on the next eQEP clock.

First Index Event: If the index event occurs during forward movement, the position counter is reset to zero on the next eQEP clock. If the index event occurs during the reverse movement, the position counter is reset to the value in the QPOS MAX register on the next eQEP clock. This is done only on the first occurrence and subsequently the position counter value is not reset on an index event, but rather on maximum position.

Maximum Position: If the position counter is equal to QPOS MAX, then the position counter is reset to zero on the next eQEP clock for forward movement and position counter overflow flag is set. If the position counter is equal to 0, the position counter is reset to QPOS MAX on the next QEP clock for reverse movement and position counter underflow flag is set.

Unit Time Out Event: In this mode, the QPOSCNT value is latched to the QPOSLAT register and then the QPOSCNT is reset to either zero or QPOS MAX, depending on the direction mode selected by QDECCTL[QSRC] bits on a unit time event. This is useful for frequency measurement.

Strobe Effect on Position Counter: Determines the action that occurs when the strobe receives a signal.

Swap A/B Channels: Normally, QEPA input is fed to the QA input of the quadrature decoder and the QEPB input is fed to the QB input of the quadrature decoder. By activating **Swap A/B Channels**, the input to the quadrature decoder is swapped, thereby reversing the counting direction.

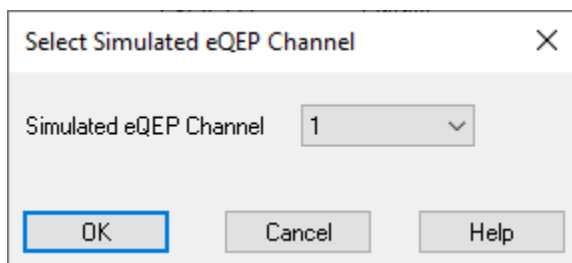
Unit: Specifies the unit to be configured.

eQEP for simulation

Target Category: Delfino, F280x, Piccolo

Target Sub-Category: Sim

Description: The eQEP block lets you provide simulated quadrature encoder counts for a simulation. The encoder counts provided to this block will appear on the output of the corresponding eQEP block.



Simulated eQEP Channel: Indicates the ePWM channel to be simulated.

Event Capture

Target Category: C2407, F281X, MSP430, STM32

Target Sub-Category: Capture

Description: The Event Capture block allows you to monitor an input pin (CAP1, CAP2, CAP3, CAP4, CAP5, CAP6) and capture a transition event (rising edge, falling edge, or both) by sampling the current value of a timer on occurrence of the

event. The Event Capture block counts from zero to 32,767 (hexadecimal 7FFF) and then wraps back. The Timer Rate Scaling parameter allows the base clock rate to be scaled down to up to 1/128 of the base rate. The output of the Event Capture block is a time interval Δt , which is the interval between two successive events. This can be used by the [Speed Calculator](#) block to determine the speed.

Additional information: Texas Instruments [SPRU357](#) document.

Event Trigger: The choices are Rising Edge, Falling Edge, Both Edges, and Quadrature (pin 1 and 2).

Input Pin: Selects the input pin to monitor. Input pins CAP 1-3 are controlled by Event Manager 1; and input pins CAP 4-6 are controlled by Event Manager 2. The input pins CAP 1-2 and 4-5 can also be used as quadrature encoder inputs. CAP 1-3 can use either Timer 1 or 2 as the sampling base, and CAP 4-6 can use either timer 3 or 4 as the sampling base.

Mux Pin: Selects which pin a given function is on.

Note: Some F280x and MSP430 devices have different functions for the same physical pin on the chip. This is referred to as multiplexing, or muxing, for short, and is done because pins are expensive. Because multiple functions compete for a given pin, you must choose what function a pin has. For flexibility, in some cases Texas Instruments provides multiple possible pins for a given function. For instance, the CANTXB function can be on pin 8, 12, or 16. Pin 8 is shared with ePWM5A and ADCSOCA0; pin 12 is shared with TZ1 and SPISIMOB; and pin16 is shared with SPISIMOA and TZ5. If you want ePWM5A on a pin, you cannot use pin 8 for CANTXB, but rather pin 12 or 16.

Timer Rate Scaling: Select from available fractional time rate multipliers to reduce the timer rate. Fractional rates of up to 1/128th the basic rate are possible.

Timer Source: The available choices depend on the choice of the input pin. Input pins 1 – 3 can use either Timer 1 or Timer 2. Input pins 4 – 6 can use Timer 3 or Timer 4. Input pins 4 – 6 are available for C2407 only.

Extern Definition

Target Category: Arduino, C2407, Cortex M3, Delfino, F280x, F281X, Generic MCU, Linux AMD64 and Raspberry Pi, MSP430, Piccolo, STM32

Target Sub-Category: Extern

Description: The Extern Definition block lets you supply any legal file scope C syntax, such as structures with initializers, #includes, #pragmas, typedefs, arrays, and function definitions. For example, you can specify an array of predefined DMA waveforms through the Extern Definition block.

Targeting Arduino boards: With the Extern Definition block, you can also specify functions available in Arduino libraries that are part of the Arduino IDE installed on your computer or available on the internet. Additionally, you can use the Extern Definition block to specify user-created C or CPP code.

External Definition Properties

External Definition:

```
#include "SR04.cpp"

//define trigger and echo pin locations on board
#define TRIG_PIN 12
#define TRIG_PIN 11

SR04 sr04 = SR04ECHO_PIN,TRIG_PIN}
```

External .obj files:

Library Modules:

Select Library Modules

OK Cancel Help

External Definition: Lets you enter C code for defining such things as buffer areas, pre-defined waveforms, and Arduino library modules. In the above dialog, #include and #define directives are preprocessor directives necessary for adding an Arduino library to a diagram.

External .obj files: Lets you add external OBJ files.

Library Modules: Indicates the Arduino library modules that you selected with **Select Library Modules**.

Select Library Modules: Lets you select Arduino library modules installed on your computer.

Extern Function

Target Category: Arduino, C2407, Cortex M3, Delfino, F280x, F281X, Generic MCU, Linux AMD64 and Raspberry Pi, MSP430, Piccolo, STM32

Target Sub-Category: Extern

Description: The Extern Function block lets you call an external function.

The Extern Function block only allows built-in C data types. This means, for example, that you would specify the unsigned short data type in the Extern Function block to match a uint16 user-defined data type.

External Function Call Properties

Function Name:

Use "\$n" to reference pin n: i.e. foo(\$1,\$2)

Input Pins: Do not declare function

Return Value Type

Has return value

Data Type:

Radix Point: Word Size:

Do Not Declare Function: Prevents the code generator from creating a declaration for the function. This is useful if the function is already declared in the header file.

Function Name: Specifies the function call. To specify function calls on new lines, press CTRL+Enter.

You can specify arguments to a function that reference the input pins using \$ notation. For example, Foo(\$1,\$2).

Input Pins: Specifies the number of input pins.

Return Value Type

Data Type: Specifies the data type of the variable. If you choose hardware register, Embed will only create a reference in the code and not an external declaration.

char: Smallest addressable unit. On the MSP430, it is 8 bits; on the C2000, it is 16 bits.

double: 64-bit floating point number.

float: 32-bit floating point number.

int: The natural word length for a given architecture.

long: 32 bits.

MATRIX: Pointer to Embed MATRIX data type. The MATRIX structure is defined in VSUSER.H.

SCALED_INT: 16- or 32-bit depending on word size, which is specified separately.

short: 16 bits.

unsigned char: Smallest addressable unsigned unit. On the MSP430, it is 8 bits; on the C2000, it is 16 bits.

unsigned long: Non-negative 32 bits.

unsigned short: Non-negative 16 bits.

Has Return Value: Lets you return a value. If you have a return value, there will be an output pin to reference it.

Radix Point: Sets the binary point.

Word Size: Specifies the word size in bits.

Extern Read

Target Category: Arduino, C2407, Cortex M3, Delfino, F280x, F281X, Generic MCU, Linux AMD64 and Raspberry Pi, MSP430, Piccolo, STM32

Target Sub-Category: Extern

Description: The Extern Read block lets you read an external variable from another C code module into the diagram. If, for the Data Type, you choose hardware register, you can enter a hardware peripheral register name and the block output will produce the value of that register when compiled.

The Extern Read block only allows built-in C data types. This means, for example, that you would specify the unsigned short data type in the Extern Read block to match a uint16 user-defined data type.

When you have a combination of Extern Read and Extern Write blocks, and you require a specific execution order, use the [execOrder](#) block to control execution.

You can also use the Extern Read block to [integrate handwritten code with generated code](#).

Assign Address: Assigns the variable a specific address. Specify the address in hexadecimal notation. For MSP430 target only.

Data Type: Specifies the data type of the variable. If you choose hardware register, Embed will only create a reference in the code and not an external declaration. The remaining data types are described below:

<32-bit hardware register>: Accesses predefined 32-bit integer. Typically, the name of a device register.

<16-bit hardware register>: Accesses predefined 16-bit integer. Typically, the name of a device register.

char: Smallest addressable unit. On the MSP430, it is 8 bits; on the C2000, it is 16 bits.

double: 64-bit floating point number.

float: 32-bit floating point number.

int: The natural word length for a given architecture.

long: 32 bits.

MATRIX: Pointer to Embed MATRIX data type. The MATRIX structure is defined in VSUSER.H.

SCALED_INT: 16- or 32-bit depending on word size, which is specified separately.

short: 16 bits.

unsigned: Basic unsigned integer type. Contains at least the [0, 65, 535] range.

unsigned char: Smallest addressable unsigned unit. On the MSP430, it is 8 bits; on the C2000, it is 16 bits.

unsigned long: Non-negative 32 bits.

unsigned short: Non-negative 16 bits.

Declare This Variable: Forces Embed to declare the variable during code generation.

Define This Variable: If activated, Embed provides global definition for the variable.

External Name: Specifies the name of the variable in the C code module.

Radix Point: Sets the binary point.

Word Size: Specifies the word size in bits.

Extern Write

Target Category: Arduino, C2407, Cortex M3, Delfino, F280x, F281X, Generic MCU, Linux AMD64 and Raspberry Pi, MSP430, Piccolo, STM32

Target Sub-Category: Extern

Description: The Extern Write block lets you write a value to an external variable in another C code module. If, for the Data Type, you choose hardware register, you can enter a hardware peripheral register name and the block input will be written to that register when compiled.

The Extern Write block only allows built-in C data types. This means, for example, that you would specify the unsigned short data type in the Extern Write block to match a uint16 user-defined data type.

When you have a combination of Extern Read and Extern Write blocks, and you require a specific execution order, use the [execOrder](#) block to control execution.

You can also use the Extern Write block to [integrate handwritten code with generated code](#).

Assign Address: Assigns the variable a specific address. Specify the address in hexadecimal notation. For MSP430 target only.

Data Type: Specifies the data type of the variable. If you choose hardware register, Embed will only create a reference in the code and not an external declaration. The remaining data types are described below:

<32-bit hardware register>: Accesses predefined 32-bit integer. Typically, the name of a device register.

<16-bit hardware register>: Accesses predefined 16-bit integer. Typically, the name of a device register.

char: Smallest addressable unit. On the MSP, it is 8 bits; on the C2000, it is 16 bits.

double: 64-bit floating point number.

float: 32-bit floating point number.

int: The natural word length for a given architecture.

long: 32 bits.

MATRIX: Pointer to Embed MATRIX data type. The MATRIX structure is defined in VSUSER.H.

SCALED_INT: 16- or 32-bit depending on word size, which is specified separately.

short: 16 bits.

unsigned: Basic unsigned integer type. Contains at least the [0, 65, 535] range.

unsigned char: Smallest addressable unsigned unit. On the MSP, it is 8 bits; on the C2000, it is 16 bits.

unsigned long: Non-negative 32 bits.

unsigned short: Non-negative 16 bits.

Declare This Variable: Forces Embed to declare the variable during code generation.

Define This Variable: If activated, Embed provides global definition for the variable.

External Name: Specifies the name of the variable in the C code module.

Radix Point: Sets the binary point.

Word Size: Specifies the word size in bits.

Full Compare Action

Target Category: C2407, F281X

Sub-Target Category: PWM

Description: The Full Compare Action block lets you control the action of the full compare PWM. The Full Compare PWM engine is designed to give you three half H-BRIDGE inverters, as found in the 3-phase brushless DC and AC induction motors. In particular, the Full Compare Action block lets you commutate a brushless DC motor.

PWM Pin1... PWM Pin6: Indicates how the chip activates the specified pin.

Full Compare PWM

Target Category: C2407, F281X

Sub-Target Category: PWM

Description: The Full Compare PWM block lets you set up a full compare PWM unit. You can choose to specify the action at initialization and it will not change during execution.

The input pins control the duty cycle of the PWM waveform. They are at scaled 1.16. The input value is multiplied by the PWM period and assigned to the PWM compare register to generate a fractional duty cycle. Thus, an input of 0.5 (fx1.16) yields a 50% duty cycle; an input of zero yields 0% duty cycle; and an input of 0.99997 (the largest possible positive value in 1.16 notation) yields 100% duty cycle.

Interactive mode: The entire block is inactive in this mode.

Full Compare PWM Properties

Timer Source: 1

Timer Rate Scaling: None

Timer Period (count): 15000

Initial Timer Count: 0

Count Mode: Up/Down

CMP Reg Load On: CTR = Zero

Add Enable Pin (0->Hi Z 1->Normal mode)

Add External Action Pin

PWM Pin: PWM1

Action: Active low

Use Deadband

Deadband prescaling: None

Deadband tick count: 0

OK Cancel Help

Action: Specifies the action to be applied to the selected PWM pin.

Add Enable Pin: Adds an input pin to the block that allows software control to enable or disable the PWM unit.

Add External Action Pin: Defaults to **Active High** if you do not activate.

CMP Reg Load On: Determines the event that causes the CMP register to be loaded. You have three choices:

CTR=Zero: Loads when counter equals 0

CTR=PRD or Zero: Loads when counter equals period or 0

Immediate: Loads immediately

Count Mode: Specify one of four modes: up and down; up; hold; or TDIR control. **Note:** Do not specify hold or TDIR control.

Initial Timer Count: Specifies the initial count for the timer. The timers are started in numerical order, one after the other with no intervening instructions. Order placement on screen will have no effect.

PWM Pin: Determines which pin to which an action is to be supplied. There are six pins.

Timer Period: In conjunction with **Timer Rate Scaling**, determines the base PWM frequency. See Texas Instruments documentation for more information.

Timer Rate Scaling: In conjunction with **Timer Period**, determines the base PWM frequency. See Texas Instruments documentation for more information.

Timer Source: Specifies the timer source. You can choose either 1 or 3.

Use Deadband, Deadband Prescaling, Deadband Tick Count: Sets the amount of deadband between PWM switching to avoid drawing too much current in a power-controlled circuit.

Get CPU Usage

Target Category: Arduino, C2407, Cortex M3, Delfino, F280x, F281X, Linux AMD64 and Raspberry Pi, MSP430, Piccolo, STM32

Target Sub-Category: Target Interface

Description: The Get CPU Usage block provides CPU usage for the enclosing task for a compound block driven off an interrupt or scheduled as a background task or scheduled as a custom rate task.

CPU usage is displayed in raw clock ticks. To translate the raw clock ticks into a percentage of the time slice of the block that it's inside of, use the Get CPU usage diagram under Toolbox > Tools.

Note: For Linux targets, the CPU rate is always considered to be 1 GHz.

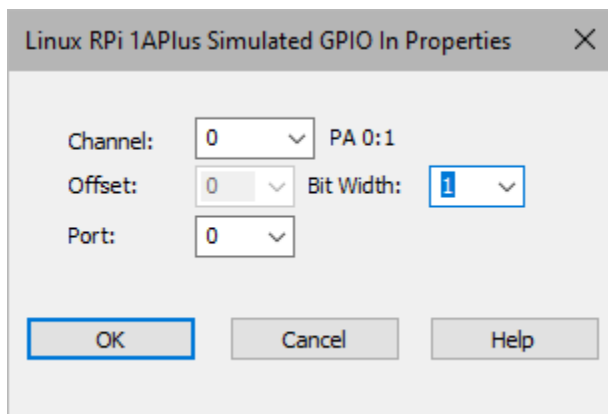
GPIO In

GPIO In for Linux Raspberry Pi

Target Category: Linux Raspberry Pi

Target Sub-Category: Sim

Description: The values presented to the input of the GPIO In block will appear on the output of the actual GPIO In block during a simulation.



Bit Width: Specifies the number of contiguous bits to read in.

Channel: Indicates the channel number. Click [here](#) for Raspberry Pi pin mapping.

Offset: Specifies the offset into the digital port register.

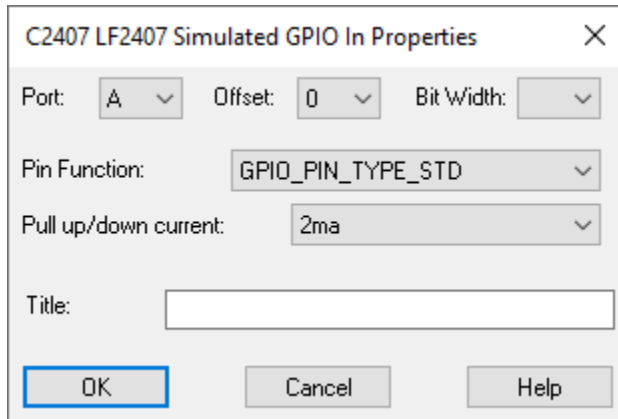
Port: Specifies the digital register. Click [here](#) for Raspberry Pi pin mapping.

GPIO In C2407, Delfino, F280x, F281X, Generic MCU, Piccolo, STM32

Target Category: C2407, Delfino, F280x, F281X, Generic MCU, Piccolo, STM32

Target Sub-Category: Sim

Description: The values presented to the input of the GPIO In block will appear on the output of the actual GPIO In block during a simulation.



Bit Width: Specifies the number of contiguous bits to read in.

Offset: Specifies the offset into the digital port register.

Pin Function: Specifies the pin function.

Port: Specifies the digital register.

Pull Up/Down Current: Lets the specified pin draw current in ON or OFF mode. It is typically used for communication protocols, like I2C.

Title: Indicates the channel title.

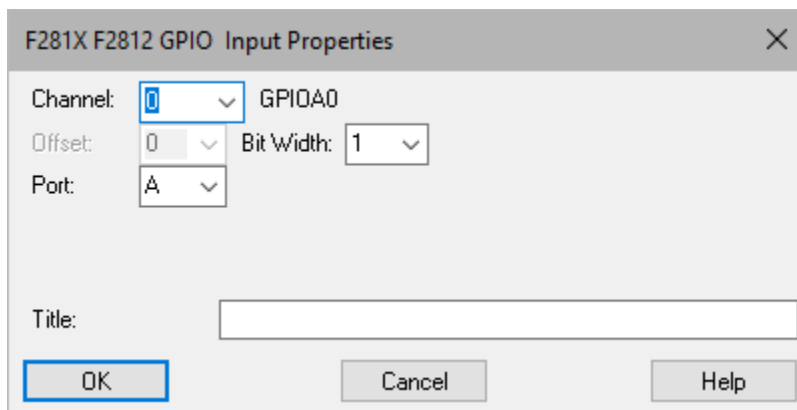
GPIO Input

GPIO Input for C2407, Delfino, F280x, F281X, Generic MCU, Linux Raspberry Pi, Piccolo

Target Category: C2407, Delfino, F280x, F281X, Generic MCU, Linux Raspberry Pi, Piccolo

Target Sub-Category: GPIO

Description: The GPIO Input block reads digital data into the diagram. Use [GPIO Qualification](#) to set up qualification time intervals for GPIO pins on Delfino, F280x, F281X, and Piccolo devices.



Bit Width: Specifies the number of contiguous bits to read in.

Channel: Indicates the channel number. Click [here](#) for Raspberry Pi pin mapping.

Enable pull-up resistor: Enables the pull-up resistor on the input pin to 3.3V. This parameter is not available for Linux Raspberry Pi, Texas Instruments C2407 and F281X, or Generic MCU targets.

Invert Input: Inverts the input. If the input is 3.3V, it is inverted to zero; if the input is zero, it is inverted to one. This parameter is available for Texas Instruments dual core, F280025, and F280049 targets.

Offset: Specifies the offset into the digital port register.

Port: Specifies the digital register. For Linux Raspberry Pi, two ports are available for Raspberry Pi 1A+ and 1B+. For all other Raspberry Pi devices, only one port is available. Click [here](#) for Raspberry Pi pin mapping.

Title: Indicates the channel title.

GPIO Input for STM32

Target Category: STM32

Target Sub-Category: GPIO

Description: The GPIO Input block reads digital data into the diagram.

Bit Width: Specifies the number of contiguous bits to read in.

Channel: Indicates the channel number.

GPIO Mode: Indicates the GPIO mode.

Offset: Specifies the offset into the digital port register.

Port: Specifies the digital register.

Pull up/down: Performs the selected action:

Pull Up: Turns the upper resistor ON and ties the pin to VDD.

Pull Down: Turns the lower resistor ON and ties the pin to ground.

None: Neither resistor is activated.

Speed: Selects the frequency. See the device datasheet for the frequency specifications.

Title: Indicates the channel title.

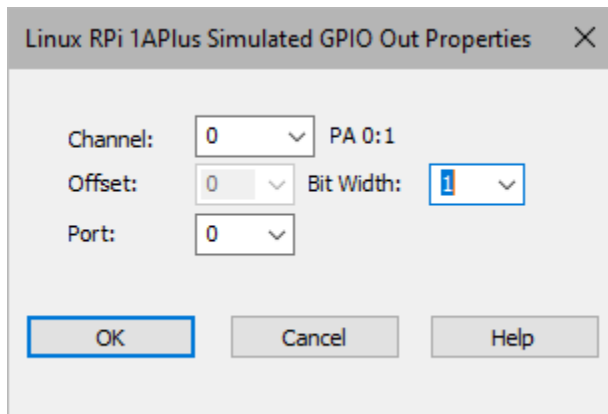
GPIO Out

GPIO Out for Linux Raspberry Pi

Target Category: Linux Raspberry Pi

Target Sub-Category: Sim

Description: The input values of the actual GPIO Out block will appear on the output of the GPIO Out block during a simulation.



Bit Width: Specifies the number of contiguous bits to read in.

Channel: Indicates the channel number. Click [here](#) for Raspberry Pi pin mapping.

Offset: Specifies the offset into the digital port register.

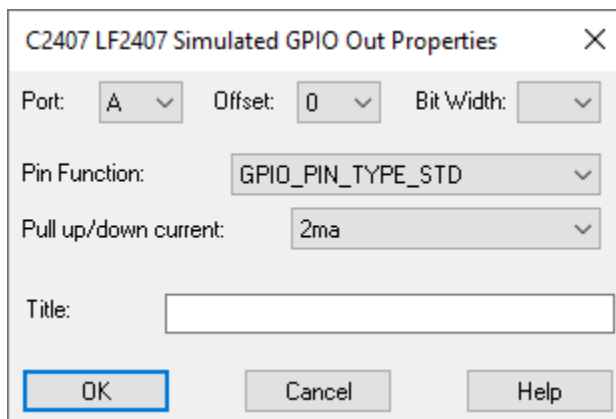
Port: Specifies the digital register. Click [here](#) for Raspberry Pi pin mapping.

GPIO Out for C2407, Delfino, F280x, F281X, Generic MCU, Piccolo, STM32

Target Category: C2407, Delfino, F280x, F281X, Generic MCU, Piccolo, STM32

Target Sub-Category: Sim

Description: The input values of the actual GPIO Out block will appear on the output of the GPIO Out block during a simulation.



Bit Width: Specifies the number of contiguous bits to read in.

Offset: Specifies the offset into the digital port register.

Pin Function: Specifies the pin function.

Port: Specifies the digital register.

Pull Up/Down Current: Lets the specified pin draw current in ON or OFF mode. It is typically used for communication protocols, like I2C.

Title: Indicates the channel title.

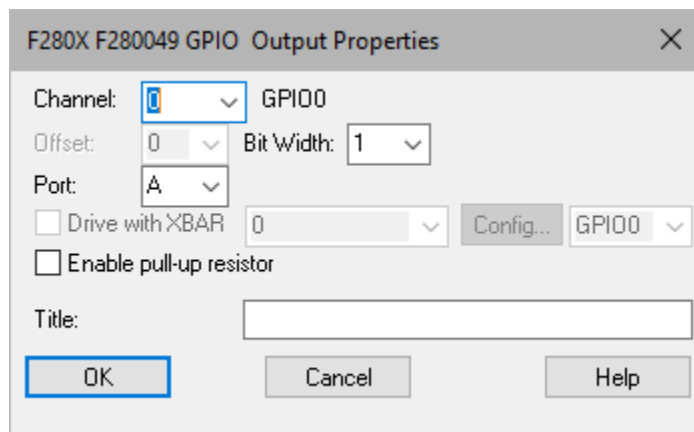
GPIO Output

GPIO Output for C2407, Delfino, F280x, F281X, Generic MCU, Linux Raspberry Pi, Piccolo

Target Category: C2407, Delfino, F280x, F281X, Generic MCU, Linux Raspberry Pi, Piccolo

Target Sub-Category: GPIO

Description: The GPIO Output block outputs digital data. Use [GPIO Qualification](#) to set up qualification time intervals for GPIO pins.



Bit Width: Specifies the number of contiguous bits to read in.

Channel: Indicates the channel number. Click [here](#) for Raspberry Pi pin mapping.

Drive with XBAR: When activated, the GPIO output is taken directly from OUTPUTXBAR, which you can configure to comparator inputs, GPIO inputs, ADC events, eCAP, sigma delta comparators. Note that only certain GPIO outputs are associated with the OUTPUTXBAR.

To configure the OUTPUT XBAR signal source, click **Config**. An XBAR dialog box appears. First, make sure the XBAR register matches the one in the GPIO dialog box. You can then select one source signal from the 128 possible inputs. If you select an INPUT XBAR, click OK to return to the GPIO dialog box and select the input GPIO from the dropdown to the right of the **Config** button. For more information, see Texas Instruments [SPRU712](#) document.

This parameter is available only on Texas Instruments dual core, F280025, and F280049 targets.

Enable pull-up resistor: Enables the pull-up resistor on the output pin to 3.3V. This parameter is not available for Linux Raspberry Pi or Generic MCU targets.

Offset: Specifies the offset into the digital port register.

Port: Specifies the digital register. For Raspberry Pi, two ports are available for Raspberry Pi 1A+ and 1B+. For all other Raspberry Pi devices, only one port is available. Click [here](#) for Raspberry Pi pin mapping.

Title: Indicates the channel title.

GPIO Output for STM32

Target Category: STM32

Target Sub-Category: GPIO

Description: The GPIO Output block outputs digital data.

Bit Width: Specifies the number of contiguous bits to read in.

Channel: Indicates the channel number.

GPIO Mode: Configures the output as push-pull or open-drain.

Push-pull mode: When a logical 1 is presented, the upper switch turns ON and the lower switch turns OFF. When a logical 0 is presented, the upper switch turns OFF and the lower switch turns ON and ties the output pin to ground.

Open-drain mode: 0 ties the output to ground; 1 leaves the output floating (Hi-Z state). In this case, the voltage is defined by the pull up or pull down resistor setting.

Always try to apply the lowest amount of voltage to minimize power usage.

Offset: Specifies the offset into the digital port register.

Port: Specifies the digital register.

Pull up/down: Performs the selected action:

Pull Up: Turns the upper resistor ON and ties the pin to VDD.

Pull Down: Turns the lower resistor ON and ties the pin to ground.

None: Neither resistor is activated.

Speed: Indicates the speed of GPIO output power switches. For more information, consult the STMicroelectronics device specification sheet.

Title: Indicates the channel title.

Hall Sensor

Target Category: STM32

Target Sub-Category: Hall Sensor

Description: The Hall Sensor block uses Hall effect positional sensors to help determine rotor position.

HRCAP

Target Category: Delfino, F280x, Piccolo

Target Sub-Category: Capture

Description: The HRCAP block performs a high-resolution capture.

Capture Unit: Specifies the unit to be configured.

Input Clock: Lets you choose between the System and PLL clock.

Max Events: Indicates the maximum number of events to be tracked.

Mux Pin: Selects which pin a given function is on.

Note: Some F280x and MSP430 devices have different functions for the same physical pin on the chip. This is referred to as multiplexing, or muxing, for short, and is done because pins are expensive. Because multiple functions compete for a given pin, you must choose what function a pin has. For flexibility, in some cases Texas Instruments provides multiple possible pins for a given function. For instance, the CANTXB function can be on pin 8, 12, or 16. Pin 8 is shared with

ePWM5A and ADCSOCA0; pin 12 is shared with TZ1 and SPISIMOB; and pin16 is shared with SPISIMOA and TZ5. If you want ePWM5A on a pin, you cannot use pin 8 for CANTXB, but rather pin 12 or 16.

I/O Memory Read

Target Category: C2407

Description: The I/O Memory Read block is used to read values from I/O memory space.

I/O Port Address: Specifies the address in hexadecimal notation.

I/O Memory Write

Target Category: C2407

Description: The I/O Memory Write block is used to write values to I/O memory space.

I/O Port Address: Specifies the address in hexadecimal notation.

I2C Read Buffer

Target Category: Arduino, Cortex M3, Delfino, F280x, F281X, MSP430, Linux Raspberry Pi, MSP430, Piccolo, STM32

Target Sub-Category: I2C

Description: The I2C Read Buffer block is used to extract data from the I2C read buffer. You can choose the type of data to be extracted.

Use the appropriate [I2C Config](#) blocks to configure the unit.

Block Output: Selects the function of the block.

Bus Busy: Indicates whether the bus is transmitting data.

Data: Extracts the data from the buffer determined by **Data Type**.

Port Status: Extracts the hardware status register determined by the Port parameter.

Receive Queue Empty: 0 if not empty; 1 if empty.

Receive Queue Overrun: 0 if not overrun; 1 if overrun.

Receive Queue Length: Returns the value of the number of data bytes in the receive queue.

Receive Queue Max Length: Returns the maximum length of the receive queue.

Transmit Queue Full: 0 if not full; 1 if full.

Transmit Queue Length: Returns the value of the number of data bytes in the transmit queue.

Transmit Queue Max Length: Returns the maximum length of the transmit queue.

Data Type: Selects the data type.

char: Smallest addressable unit. On the MSP, it is 8 bits; on the C2000, it is 16 bits.

long: 32 bits.

short: 16 bits.

Port: Specifies the hardware unit. Click [here](#) for Arduino pin mapping. Click [here](#) for Raspberry Pi pin mapping.

I2C Start Communication

Target Category: Arduino, Cortex M3, Delfino, F280x, F281X, Linux Raspberry Pi, MSP430, Piccolo, STM32

Target Sub-Category: I2C

Description: The I2C Start Communication block is used to start sending the current contents of the queue.

Use the appropriate [I2C Config](#) blocks to configure the unit.

Action: Lets you choose between starting to send and starting to receive data.

Bus Mode: Lets you choose between master and slave mode. When the target device is Arduino and **Bus Mode** is set to **Slave**, you must encapsulate the slave response block logic in a compound block and enable the I2C Receive/Transmit interrupt. Examples are under Examples > Embedded > Arduino > I2C.

Message Byte Count: Specifies the number of bytes to send or receive before the STOP signal (if checked) is sent.

Port: Specifies the address. Click [here](#) for Arduino pin mapping. Click [here](#) for Raspberry Pi pin mapping.

Send NACK after next byte read: Sends a Not Acknowledged signal at the end of the data packet transmission. This parameter is available only for C2000 blocks.

Send STOP at Transmission End: Sends a STOP signal at the end of the transmission.

Set Data Length Dynamically: When activated, overrides the Message Byte Count parameter and adds a Data Length input pin to the block to dynamically control the data length. This parameter is available only when Bus Mode is set to **Master**.

Slave Address: Specifies the address of the slave to be written to or read from using 7-bit addressing.

I2C Write Buffer

Target Category: Arduino, Cortex M3, Delfino, F280x, F281X, Linux Raspberry Pi, MSP430, Piccolo, STM32

Target Sub-Category: I2C

Description: The I2C Write Buffer block is used to write data to the I2C buffer. You can choose the type of data to be written.

Use the appropriate [I2C Config](#) blocks to configure the unit.

Data Type: Selects the data type.

char: Smallest addressable unit. On the MSP, it is 8 bits; on the C2000, it is 16 bits.

long: 32 bits.

short: 16 bits.

Port: Specifies the hardware unit. Click [here](#) for Arduino pin mapping. Click [here](#) for Raspberry Pi pin mapping.

JSON

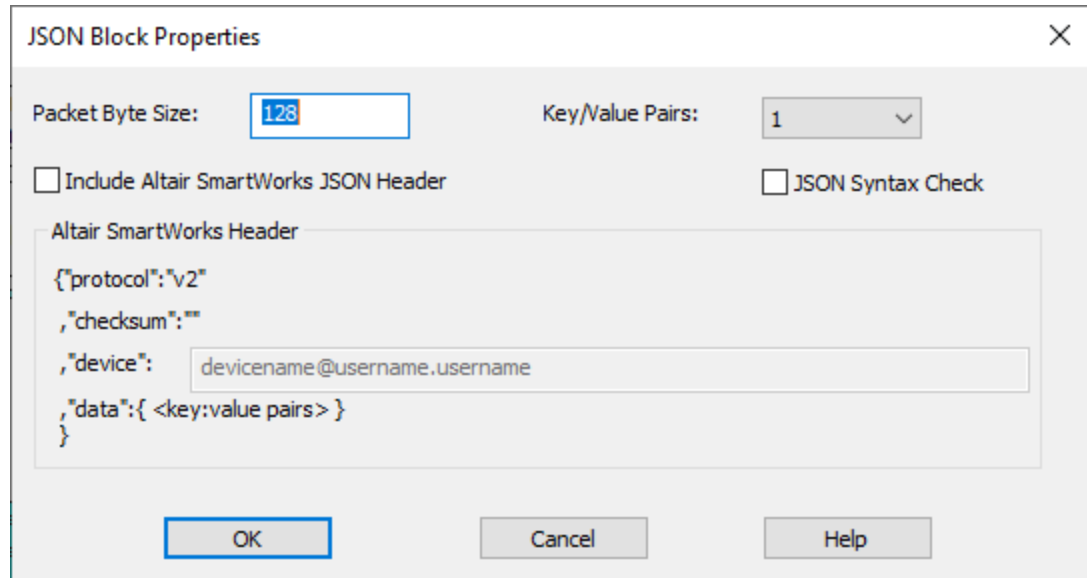
Target Category: Arduino, Delfino, F280x, F281X, Generic MCU, Linux AMD64 and Raspberry Pi, Piccolo, STM32; Blocks > Extensions > IOT

Target Sub-Category: IoT

Description: The JSON block provides an encoding format for translating data into text strings that can be sent and received over the internet.

JSON is used in conjunction with the MQTT blocks to create JSON-formatted text. By using MQTT to send JSON-formatted text to `mqtt.smartcore.com`, you can take advantage of SmartWork's ability to trigger actions and activate rules that look at the data within the JSON data packet. SmartWorks is an Altair IOT development platform. Output from the JSON block is in the following format:

```
{"protocol": "v2", "checksum": "", "device": "devicename@username.username", "at": "now", "data": {"Key": Value}}
```

Altair SmartWorks Header

Specifies the output format of the JSON block. The ,“**device**”: specifies the device that is sending the data. For SmartWorks, the value is specified in the following format:

devicename@username.username

Include Altair SmartWorks JSON Header: When activated, the SmartWorks platform can take actions based on the values of the key pairs.

JSON Syntax Check: Validates the format of the JSON text string.

Key/Value Pairs: Indicates the number of Key/Value input pin pairs on the JSON block. The number you specify is reflected under the Key and Value parameters. Also, when you exit the dialog box, the input pins on the block are updated accordingly. You cannot use the Add Connector and Remove Connector commands on JSON blocks.

Packet Byte Size: Specifies the maximum buffer size of the data string. Larger strings are truncated to the maximum buffer size.

Examples

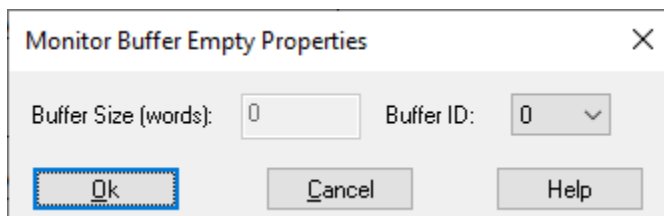
See [Examples > Blocks > Extensions > IOT](#).

Monitor Buffer Empty

Target Category: C2407, Cortex M3, Delfino, F280x, F281X, Piccolo, STM32

Target Sub-Category: Monitor Buffer

Description: The Monitor Buffer Empty block produces a one when the monitor buffer is empty, and zero when it has one or more items.



Buffer ID: Indicates the numeric ID of the buffer to read. This number must correspond to an existing Monitor Buffer Empty block on the target.

Buffer Size: Reserved.

Monitor Buffer Read

Target Category: C2407, Cortex M3, Delfino, F280x, F281X, Piccolo, STM32

Target Sub-Category: Monitor Buffer

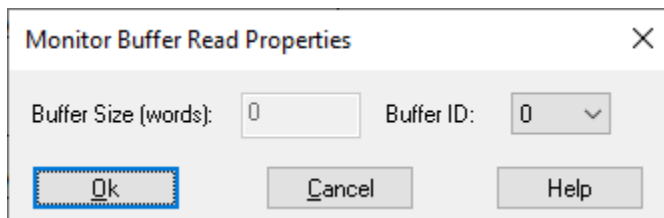
Description: The Monitor Buffer Read block is a debug information block for receiving information from a Monitor Buffer Write block running on the target. The Monitor Buffer Read block only runs on the host PC.

The Monitor Buffer Read block has two outputs:

- **Trig:** A data ready pin
- **Buffer:** A vector of the monitored data received from the target.

When the Trig pin is high, a new vector of monitored values are ready on the Buffer pin. Typically, this block is connected to an enabled plot block with the Trig pin connected to the plot enable pin and the Buffer pin connected to a plot data input pin. In this way, the plot block will appear as an interactive oscilloscope.

Use the [Monitor Buffer Write](#) block on the target to write data to this block.



Buffer ID: Indicates the numeric ID of the buffer to read. This number must correspond to an existing Monitor Buffer Write block on the target.

Buffer Size: Indicates the number of words in the monitor buffer on the target. This is read-only.

Monitor Buffer Write

Target Category: C2407, Cortex M3, Delfino, F280x, F281X, Piccolo

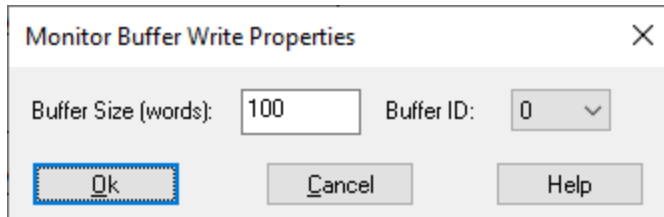
Target Sub-Category: Monitor Buffer

Description: The Monitor Buffer Write block is a debug information block for transmitting information from the target to the host PC.

The Monitor Buffer Write block has two inputs:

- **Trig:** A data ready pin.
- **Signal:** Receive data pin. When the trigger pin is high and the buffer is empty, the values entering the signal pin are written to the monitor buffer until the buffer is full, regardless of the value on the trigger pin. You may use any combination of Embed blocks to form the trigger expression. Non-numeric data is not supported as input to the data pin. You can convert the data by wiring a convert block to the data pin.

Use the [Monitor Buffer Read](#) block on the host PC side to read the data from this block. After the Monitor Buffer Read block reads the monitor buffer, it resets the target buffer to empty state.



Buffer ID: Indicates the numeric ID of this buffer. The Buffer ID will be referenced by the Monitor Buffer Read block.

Buffer Size: Indicates the number of words to be allocated to the monitor buffer.

MQTT Publish

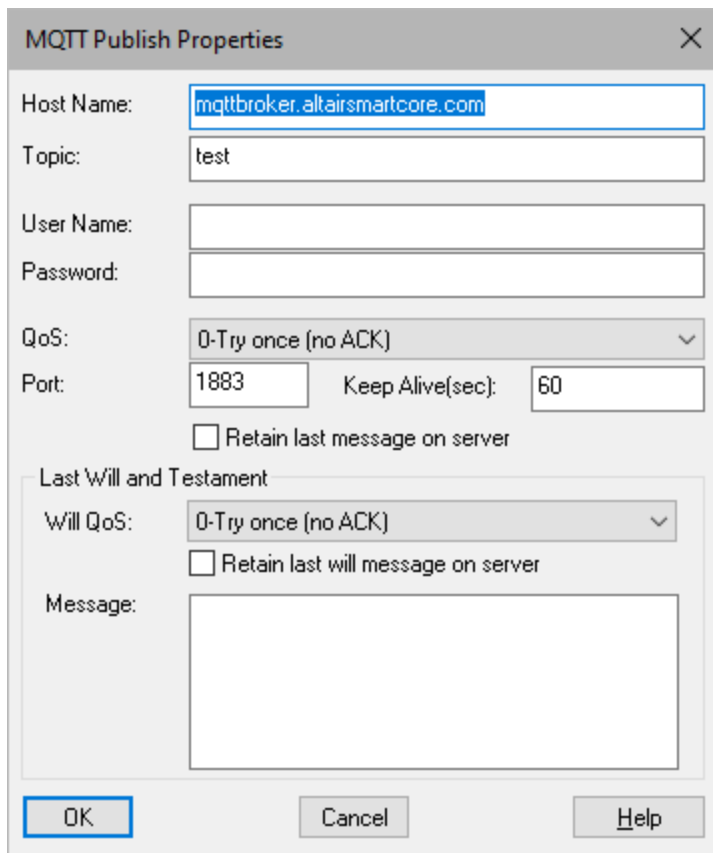
Target Category: Arduino, Generic MCU, Linux AMD64 and Raspberry Pi

Target Sub-Category: IoT, IoT > ESP8266WiFi

Description: The MQTT Publish block publishes messages on the specified topic from your hardware to the Message Queuing Telemetry Transport (MQTT) broker. The broker is primarily responsible for receiving all messages, filtering messages, deciding who is interested in the messages, and publishing the messages to all subscribed clients.

The MQTT Publish block has two inputs:

- **Enable:** When the enable pin is high, the values entering the data pin are presented to the transmit queue. When the enable pin is zero, values entering the data pin are ignored.
- **Data:** Accepts the string to be published to the cloud.



Host Name: Specifies the MQTT broker address. For SmartWorks, use `mqttbroker.altairsmartcore.com` for data exchange and `mqtt.altairsmartcore.com` for API.

Keep Alive: Specifies the maximum time interval between when the client finishes sending a PING message to the broker and starts to send the next PING message. The maximum time interval is 18 hours, 12 minutes, and 15 seconds. When **Keep Alive** is set to 0, the keep alive mechanism is de-activated and **Last Will and Testament** is ignored. The default time interval is 60 sec.

Last Will and Testament

Message: Specifies the last will message to be sent to subscribers. If the broker detects that the publisher has unexpectedly disconnected, it sends the last will message to all subscribers of the specified topic.

Retain last will message on server: Retains the last will message if the publisher unexpectedly disconnects.

Will QoS: Specifies the quality of service to the broker. Your choices are:

0: Sends the message once with no guarantee of delivery. The broker does not acknowledge delivery.

1: Guarantees message delivery, but it could send duplicates. The broker acknowledges delivery.

2: Guarantees message delivery without duplicates. The broker acknowledges delivery, then both the broker and publisher discard the stored messages.

Password: Specifies the user password. This is used for authenticating with the broker. As you type the password into the **Password** parameter, the typed characters are converted to dots.

Port: Indicates the port that the broker is using. The default unencrypted port is 1883; the default encrypted port is 883.

QoS: Specifies the quality of service to the server. Your choices are:

0: Sends the message once with no guarantee of delivery. The server does not acknowledge delivery.

1: Guarantees message delivery, but it could send duplicates. The server acknowledges delivery.

2: Guarantees message delivery without duplicates. The server acknowledges delivery, then both the server and publisher discard the stored messages.

Retain last message on server: Retains the last message if the publisher unexpectedly disconnects.

Topic: Specifies the topic on which the message is to be published. For SmartWorks, enter `your-API-key/streams`.

User Name: Specifies the user ID on the MQTT broker. This is used for authenticating with the broker. For SmartWorks, enter your API key.

Examples

See [Examples > Blocks > Extensions > IOT](#).

MQTT Subscribe

Target Category: Arduino, Generic MCU, Linux AMD64 and Raspberry Pi

Target Sub-Category: IoT, IoT > ESP8266WiFi

Description: The MQTT Subscribe block receives messages on topics to which you have subscribed. When you subscribe to a topic, the Message Queuing Telemetry Transport (MQTT) broker sends messages to the MQTT Subscribe block on that topic. The broker is primarily responsible for receiving all messages, filtering messages, deciding who is interested in the messages, and publishing the messages to all subscribed clients.

The MQTT Subscribe block has two outputs:

- **rdy:** Lets you know when new data has arrived.
- **val:** The data in string format.

Host Name: Specifies the MQTT broker address. For SmartWorks, use **mqttbroker.altairsmartcore.com** for data exchange and **mqtt.altairsmartcore.com** for API.

Keep Alive: Specifies the maximum time interval between when the client finishes sending a PING message to the broker and starts to send the next PING message. The maximum time interval is 18 hours, 12 minutes, and 15 seconds. When **Keep Alive** is set to 0, the keep alive mechanism is de-activated and **Last Will and Testament** is ignored. The default time interval is 60 sec.

Last Will and Testament

Message: Specifies the last will message to be sent to subscribers. If the broker detects that the publisher has unexpectedly disconnected, it sends the last will message to all subscribers of the specified topic.

Retain last will message on server: Retains the last message if the subscriber unexpectedly disconnects.

Will QoS: Specifies the quality of service to the broker. Your choices are:

0: Sends the message once with no guarantee of delivery. The broker does not acknowledge delivery.

1: Guarantees message delivery, but it could send duplicates. The broker acknowledges delivery.

2: Guarantees message delivery without duplicates. The broker acknowledges delivery, then both the broker and publisher discard the stored messages.

Password: Specifies the user password. This is used for authenticating with the broker. As you type the password into the **Password** parameter, the typed characters are converted to dots.

Port: Indicates the port that the broker is using. The default unencrypted port is 1883; the default encrypted port is 883.

QoS: Specifies the quality of service to the server. Your choices are:

0: Sends the message once with no guarantee of delivery. The server does not acknowledge delivery.

1: Guarantees message delivery, but it could send duplicates. The server acknowledges delivery.

2: Guarantees message delivery without duplicates. The server acknowledges delivery, then both the server and publisher discard the stored messages.

Topic: Specifies the topic on which the message is to be published. For SmartWorks, enter *your-API-key/streams*.

User Name: Specifies the user ID on the MQTT broker. This is used for authenticating with the broker. For SmartWorks, enter your API key.

Examples

See **Examples > Blocks > Extensions > IOT**.

Op Amp

Target Category: MSP430

Description: The Op Amp block supports the MSP430 OP AMP peripheral, which performs a variety of operational amplifier functions.

Additional information: Texas Instruments [MSP56F](#) document.

Op Amp Properties

Unit: 0

Function: General purpose opamp

Inverting Input: 0Ax10

Noninverting Input: 0Ax10

Slew Rate: Off, output high Z

Feedback Resistor: Tap 0 - 0R/16R

Output: 0A00OUT->A1

Resistor Connection: Rtop= AVss,Rbot = AVcc

Inverting input externally available

OK Cancel Help

Feedback Resistor Selects the internal feedback resistor configuration.:

Function: Selects the operating mode.

Inverting Input: Selects the device pin for the inverting input.

Inverting Input Externally Available: Checks if inverting input is available on a pin.

Noninverting Input: Selects the device pin for the noninverting input.

Output: Selects the device pin for the output.

Resistor Connection: Selects the feedback resistor.

Slew Rate: Selects the speed of response. (Low Speed = Low Power).

Unit: Specifies the OP AMP peripheral unit number.

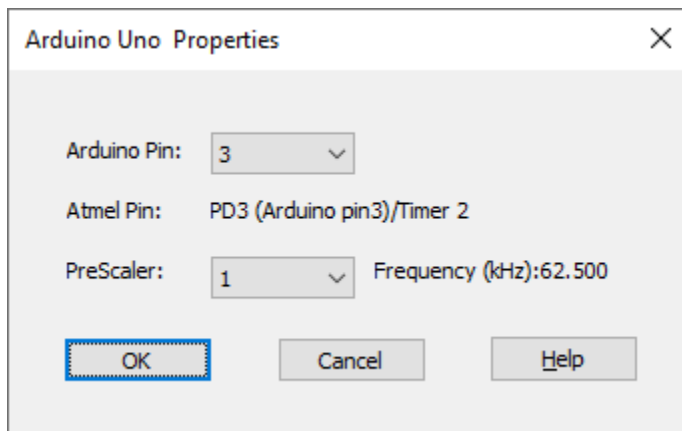
PWM

PWM for Arduino

Target Category: Arduino

Target Sub-Category: PWM

Description: The PWM block can be used for several tasks including dimming an LED, generating modulated and audio signals, and providing variable speed motor control. The PWM block accepts scaled integer (fixed point) 8.16 input.



Arduino Pin: Indicates the pin number on the Arduino board. Click [here](#) for Arduino pin mapping.

Atmel Pin: Indicates the timer associated with the selected PWM pin. Timers 0 and 2 are 8-bit timers; Timer 1 is a 16-bit timer. This is a read-only parameter.

PreScaler: Indicates the prescaler value used to divide the system clock frequency that drives the PWM counter. The frequency is the effective PWM waveform frequency, which is the system clock divided by the prescale value divided by 256 (period register size). Click [here](#) for the Arduino frequency table.

PWM for Linux Raspberry Pi

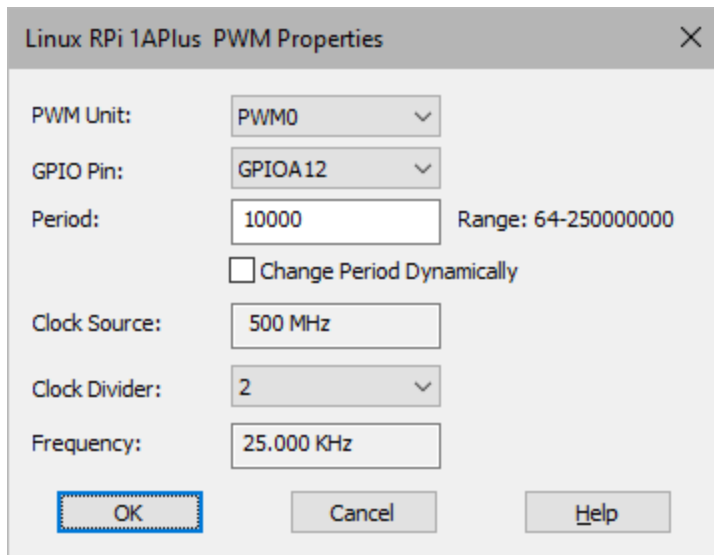
Target Category: Linux Raspberry Pi

Target Sub-Category: PWM

Description: There are two independent output bit-streams, clocked at a fixed frequency, for the PWM block. Additionally, there are two independent PWM channels (0 and 1), each of which can be connected to a limited subset of GPIO pins. Both PWM channels are driven by the same PWM clock, whose clock divider can be varied. Each channel can be separately enabled. The average output of a PWM channel is determined by the ratio of DATA/RANGE for that channel.

The PWM pin available on the GPIO header is shared with the audio system. This means that you cannot use PWM output and play audio through the 3.5mm jack at the same time.

The PWM block accepts scaled integer (fixed point) 1.16 input.



Change Period Dynamically: Produces an external input pin that lets you specify a fractional value for the period on-the-fly. The external value will be multiplied by the period and assigned to the period register. This allows you to dynamically modulate the PWM-based frequency.

Clock Divider: Specifies the value used to divide the system clock frequency that drives the PWM counter.

Clock Source: Indicates the PWM input clock speed. For Raspberry Pi, this is the system clock, which is determined by the CPU selected in the [Target Config](#) block.

Frequency: Indicates the frequency of the generated PWM signal.

GPIO Pin: Specifies the GPIO pin on the device. Click [here](#) for Raspberry Pi pin mapping.

Period: Specifies the number of PWM clock ticks for one complete PWM waveform. The frequency of the PWM signal is determined as follows:

$$\text{Clock-Source-Frequency} / \text{Period} = \text{PWM-Frequency}$$

For example, $150\text{MHz} / 150 = 1\text{MHz}$.

PWM Unit: Specifies the unit to be configured.

PWM for C2407, F2812

Target Category: C2407, F281X

Target Sub-Category: PWM

Description: There are four timers on the 2407, and each one has a PWM. For each PWM, you can specify the timer, timer rate, and timer period. Together, they provide the base frequency for the PWM. The input pin represents the fraction of ON time for the PWM pulse. A value of 0.99997 provides 100% ON time; a value of zero provides no ON time.

Interactive mode: In this mode, the Duty Cycle input pin is active; however, all parameters in the dialog box are inactive except Timer Source.

Additional information: Creating [phase-shifted PWM signals](#).

Count Mode: Specify one of four modes: up and down; up; hold; or TDIR control. Do not specify hold or TDIR control.

Initial Timer Count: Specifies the initial count for the timer. The timers start in numerical order, one after the other with no intervening instructions. Order placement on screen will have no effect.

Timer Period: In conjunction with **Timer Rate Scaling**, determines the frequency of the PWM waveform. For example, if the base clock rate is 32MHz and the Timer Rate Scaling is 1/32, the effective clock rate is 1MHz.

If **Timer Period** is set to 10, the output PWM frequency is 1/10 the effective clock rate of 1MHz (100kHz). However, this is a poor example since if **Timer Period** is set to be only 10, the output PWM frequency can only be varied in steps of 10%.

You must set **Timer Rate Scaling** and **Timer Period** in such a way that the desired output PWM frequency is obtained, while retaining fine control over the output duty cycle.

See Texas Instruments documentation for more information.

Timer Rate Scaling: In conjunction with **Timer Period**, determines the base PWM frequency. Select from available fractional timer rate multipliers to reduce the timer rate. Fractional rates of up to 1/128th the basic rate are possible. See Texas Instruments documentation for more information.

Timer Source: Specifies the timer source. If you choose the same timer for each PWM, then you must also choose the same timer rate and timer period for each PWM. There is a one-to-one correspondence between each EVM timer and the PWM output. For example, PWM1 uses Timer 1 and so on.

PWM for MSP430

Target Category: MSP430

Target Sub-Category: PWM

Description: The PWM block accepts scaled integer (fixed point) 1.16 input.

PWM Properties

Timer Source: TA0

Timer Configure

Timer Rate Scaling: None

Timer Period: 10000

Count Mode: Up

PWM Configure

PWM Out: TA0OUT1

PWM Mode: Software

PWM Pin: P1.2

OK Cancel Help

PWM Configure

PWM Out: Chooses the pin on which the PWM appears.

PWM Pin: See Timer A and Timer B descriptions in the MSP430UM.PDF file supplied with this software.

PWM Mode: See Timer A and Timer B descriptions in the MSP430UM.PDF file supplied with this software.

Timer Configure

Count Mode: Specify one of four modes: Hold, Up/Down, Up, and Continuous.

Timer Period: In conjunction with **Timer Rate Scaling**, determines the frequency of the PWM waveform. For example, if the base clock rate is 32MHz and **Timer Rate Scaling** is 1/32, the effective clock rate is 1MHz.

If **Timer Period** is set to be 10, then the output PWM frequency is 1/10 the effective clock rate of 1MHz (100kHz). However, this is a poor example since if **Timer Period** is set to be only 10, the output PWM frequency can only be varied in steps of 10%.

Timer Rate Scaling: In conjunction with **Timer Period**, determines the base PWM frequency. Select from available fractional timer rate multipliers to reduce the timer rate. Fractional rates of up to 1/8th the basic rate are possible. See Texas Instruments documentation for more information.

Timer Source: Specifies the timer source. If you choose the same timer for each PWM, then you must also choose the same timer rate and timer period for each PWM. Check with your MSP430 part description for the number of PWM outputs that are available.

PWM for STM32

Target Category: STM32

Target Sub-Category: PWM

Channel Configuration

Channel 1-6: Specifies the channel to be configured. Each channel is associated with a compare register that is used to create PWM waveforms.

Dead Time: Sets the amount of deadband between PWM switching to avoid drawing too much current in a power-controlled circuit.

Deadtime Clk Prescale: Specifies the clock divisor for the deadtime. The divisor divides the timer clock to obtain a deadtime clock. For more information, see the STMicroelectronics documentation for the device you are using.

Mode: Determines the action on compare register match. There are a number of modes from which to choose. Normally, **PWM** mode is selected. For more information, see the STMicroelectronics documentation for the device you are using.

Negative Output: Only channels 1 – 3 have negative output.

- **Idle:** Specifies the output state on an occurrence of a break fault. **Idle level on Break** must be enabled. For more information, see TIMx OSSI bit in the STMicroelectronics documentation for the device you are using.
- **GPIO Out:** Specifies the output pin on the chip for the given signal.

Positive Output: Only channels 1 – 4 have positive output.

- **Polarity:** Determines if the active mode is High or Low. For more information, see the STMicroelectronics documentation for the device you are using.
- **Idle:** Specifies the output state on an occurrence of a break fault. **Idle level on Break** must be enabled. For more information, see TIMx OSSI bit in the STMicroelectronics documentation for the device you are using.

- **GPIO Out:** Specifies the output pin on the chip for the given signal.

Fault Handling

Timers can have zero, one, or two break units.

Add Block Enable Pin: Uses hardware faulting mechanism. When you activate Add Block Enable Pin, you invoke breaking to perform analysis.

Auto-Reenable (AOE): Clears the break at the end of the PWM waveform.

Break1 GPIO In: Specifies the pin that will be used to invoke the break fault.

Break1 Polarity: Specifies the polarity of the break input signal.

Break1 Filter: Specifies the duration of the break input signal before the faulting is invoked.

Break2 GPIO In: Specifies the pin that will be used to invoke the break fault.

Break2 Polarity: Specifies the polarity of the break input signal.

Break2 Filter: Specifies the duration of the break input signal before the faulting is invoked.

Inactive on Break: When enabled and a fault occurs, each output line will be set to the inactive level (that is, the opposite of the active level). When neither **Inactive on Break** nor **Idle Level on Break** is enabled, the GPIO outputs go into a Hi-impedance state.

Idle Level on Break: When enabled and a fault occurs, each output line will be set to the idle level as specified per channel under **Positive Output** and **Negative Output**. When neither **Inactive on Break** nor **Idle Level on Break** is enabled, the GPIO outputs go into a Hi-Z state.

Master/Slave Configuration

Count Mode: Selects the counter behavior. For more information, see the STMicroelectronics documentation for the device you are using.

ETR GPIO In: For external trigger connected to a GPIO pin of the device.

Input Trigger: Selects the source of the input trigger.

Invert ETR Extern Trigger: Inverts the ETR GPIO In trigger.

Prescale: Divides the rate of the input trigger.

Sync Master TRGI: You must select a TRGI mode under **Count Mode**. For more information, see the STMicroelectronics documentation for the device you are using.

Trigger Filter: Specifies the duration of the trigger signal before the trigger event is asserted.

Output Trigger Setup

TRGO (to ADC, Timx, Slave): Used by the ADC to start a conversion sequence. It can also be used by another timer as a trigger input.

TRGO2 (to ADC): Used by the ADC to start a conversion sequence.

Update Event Div: Divisor for the update event used to send update event triggers. For example, if you enter 1, the trigger is sent out for every update event; if 2 it sends out every other update event, and so on. An update event occurs when you get a counter overflow or underflow. For more information, see the STMicroelectronics documentation for the device you are using.

Rate Configuration

Count Mode: Determines the counting mode. For more information, see the STMicroelectronics documentation for the device you are using.

Period: Specifies the duration of a PWM waveform in units of timer ticks. In conjunction with the system clock selected for the timer, the timer **Prescale** and **Count Mode**, determines the frequency of the PWM waveform. For example, if the base clock rate is 72MHz and **Prescale** is 720 and the **Count Mode** is Up/Down (divide by 2), the effective clock rate is 50kHz.

The frequency appears to the right of the **Period**. For more information, see the STMicroelectronics documentation for the device you are using.

Prescale: Scales the timer source to a slower rate. Can be any value between one and 65,536. For more information, see the STMicroelectronics documentation for the device you are using.

Timer Unit: Specifies the timer source.

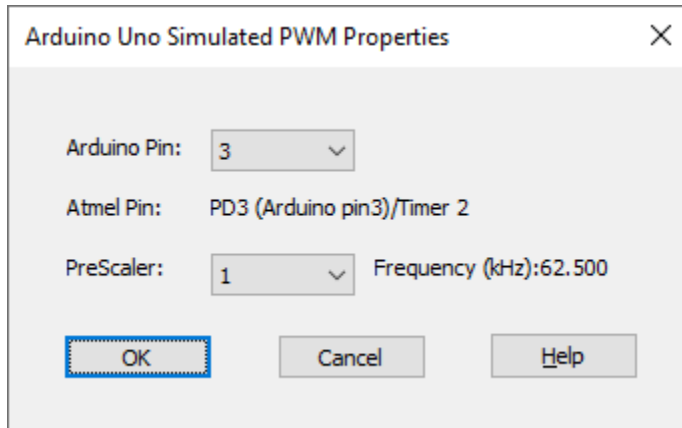
PWM for simulation

PWM for simulation for Arduino

Target Category: Arduino

Target Sub-Category: Sim

Description: The PWM for Simulation block is used during simulation to produce a fixed-point value between 0 and 1 that is supplied to the PWM block.



Arduino Pin: Specifies the Arduino pin. Click [here](#) for Arduino pin mapping. This parameter is available only for Arduino targets.

Atmel Pin: Indicates the Atmel pin that corresponds to the specified Arduino pin. This parameter is available only for Arduino targets.

Prescaler: This parameter is ignored.

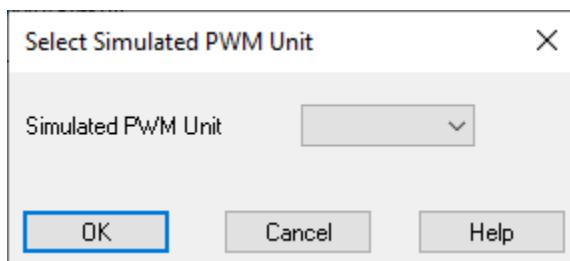
PWM for simulation for C2407, F281X, MSP430, STM32

Target Category: C2407, F281X, MSP430, STM32

Target Sub-Category: Sim

Description: The PWM for Simulation block is used during simulation to produce a fixed-point value between 0 and 1 that is supplied to the PWM block.

When a PWM input is not connected, the corresponding output on the PWM for Simulation displays a 0 value.



Simulated PWM Channel/Unit: Specifies the PWM channel to be simulated.

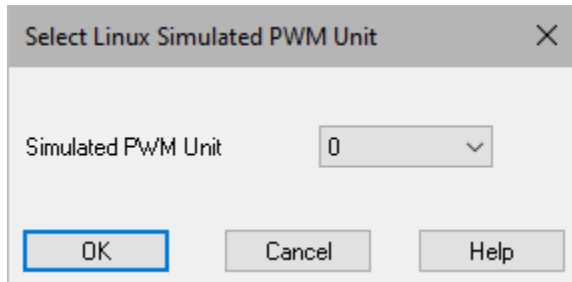
PWM for simulation for Linux Raspberry Pi

Target Category: Linux

Target Sub-Category: Sim

Description: The Linux RPi PWM for Simulation block is used during simulation to produce a fixed-point value between 0 and 1 that is supplied to the PWM block.

When a PWM input is not connected, the corresponding output on the PWM for Simulation displays a 0 value.



Simulated PWM Unit: Specifies the PWM channel to be simulated.

Quadrature Encoder

Quadrature Encoder for F281X

Target Category: F281X

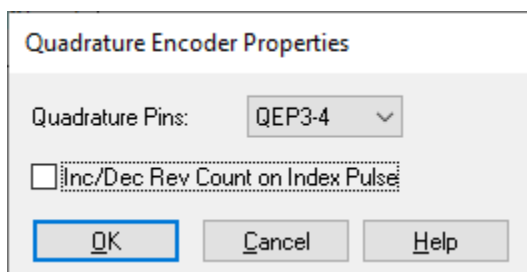
Sub-Target Category: Quadrature Encoder

Description: The Quadrature Encoder block lets you read quadrature encoded counts. The Quadrature Encoder block requires two pins to work.

The Quadrature Encoder block has three outputs. The topmost output is a 16-bit integer representing the number of encoder counts. It cycles back to zero when it reaches 65,535. The second output is the direction, which is +1 for forward and -1 for backward. The third output counts the index pulses when the corresponding parameter in the Quadrature Encoder dialog box is activated.

Refer to the following table for information on encoder logic:

	F2812	F2812
Encoder Channel A	QEP1	QEP3
Encoder Channel B	QEP2	QEP4
Encoder Index	CAP3	CAP5



Inc/Dec Rev Count on Index Pulse: Enables the counting of index pulses.

Quadrature Pins: Specifies the pins from which to read quadrature encoded counts. This parameter is not available for MSP430 targets.

Peripheral Interrupt: Specifies the interrupt for which you want to look.

Quadrature Encoder for STM32

Target Category: STM32

Sub-Target Category: Quadrature Encoder

Description: The Quadrature Encoder block lets you read quadrature encoded counts.

The screenshot shows the 'Quadrature Encoder Properties' dialog box. The fields are as follows:

- Timer Unit: TIM1
- Mode: ch1 edge
- Max Pos: 0
- Reset Position Count on Index Pulse
- Inc/Dec Rev Count on Index Pulse
- Invert A input
- Invert B input
- Input A Filter: None
- Input B Filter: None
- Mux Pin Assignments:
 - A: PA8
 - B: PA9
 - Index: PA0

Inc/Dec Rev Counts on Index Pulse: Enables the counting of index pulses.

Input A Filter: Establishes the minimum duration that signal A stays transitioned in order to be considered a true transition. Use this parameter to prevent erroneous transitions due to small amounts of noise. For more information, see the SMT32 documentation.

Input B Filter: Establishes the minimum duration that signal B stays transitioned in order to be considered a true transition. Use this parameter to prevent erroneous transitions due to small amounts of noise. For more information, see the SMT32 documentation.

Invert A Input: Inverts the input on A.

Invert B Input: Inverts the input on B.

Max Pos: Indicates the maximum position.

Mode: Enables counting edges on a specific channel or both channels.

Mux Pin Assignments: Assigns a peripheral I/O port to a pin.

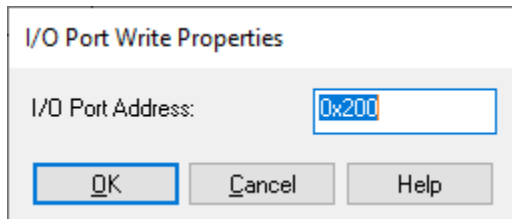
Reset Position Count on Index Pulse: Resets the position count to zero when an index pulse occurs.

Timer Unit: Specifies the timer unit to be configured.

Read Target Memory

Target Category: MSP430

Description: The Read Target Memory block reads a specific memory address on the MSP430.



I/O Port Address: Selects the memory address on the MSP430. Specify the address in hexadecimal notation.

SD16

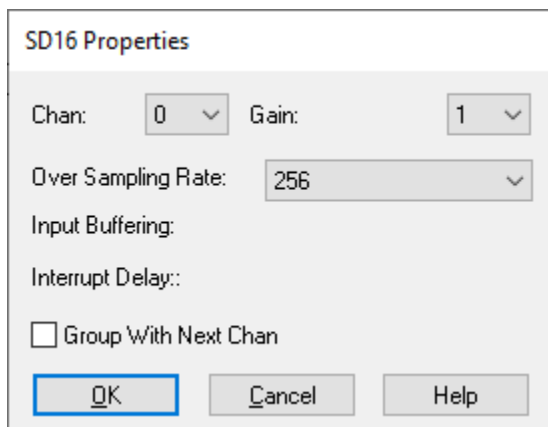
Target Category: MSP430

Sub-Target Category: ADC

Description: The SD16 block lets you perform 16-bit analog to digital conversions. It uses sigma delta oversampling technique to provide high-precision reading. Note that the effective sampling rate is:

$$\frac{SDCLK}{oversample}$$

Use the [SD16 Config](#) command to choose the hardware settings.



Chan: Specifies the analog input channel.

Gain: Specifies the gain to be applied to the channel.

Group With Next Chan: Groups the channel with the next higher channel.

Input Buffering: Not available on this block.

Interrupt Delay: Not available on this block.

Over Sampling Rate: Specifies the oversampling rate. Generally speaking, higher rates are more accurate but take more time.

SD16A

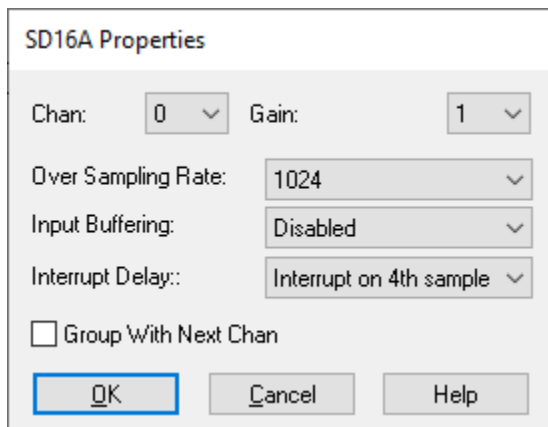
Target Category: MSP430

Sub-Target Category: ADC

Description: The SD16A block lets you perform 16-bit analog to digital conversions. It uses sigma delta oversampling technique to provide high-precision reading. Note that the effective sampling rate is

$$\frac{SDCLK}{oversample}$$

Use the [SD16 Config](#) command to choose the hardware settings.



The image shows a dialog box titled "SD16A Properties". It contains several configuration options:

- Chan:** A dropdown menu set to "0".
- Gain:** A dropdown menu set to "1".
- Over Sampling Rate:** A dropdown menu set to "1024".
- Input Buffering:** A dropdown menu set to "Disabled".
- Interrupt Delay:** A dropdown menu set to "Interrupt on 4th sample".
- Group With Next Chan:** An unchecked checkbox.

At the bottom of the dialog are three buttons: "OK", "Cancel", and "Help". The "OK" button is highlighted with a blue border.

Chan: Specifies the analog input channel.

Gain: Specifies the gain to be applied to the channel.

Group With Next Chan: Groups the channel with the next higher channel.

Input Buffering: Enables up to 1ma to drive the 1.2V internal Vref. This parameter is for external use.

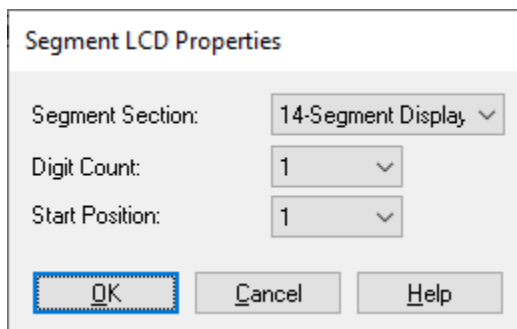
Interrupt Delay: Specifies the number of interrupts to ignore before sampling the data.

Over Sampling Rate: Specifies the oversampling rate. Generally speaking, higher rates are more accurate but take more time.

segmentLCD

Target Category: MSP430

Description: The segmentLCD block lets you write segment control lines on the MSP430.



The image shows a dialog box titled "Segment LCD Properties". It contains three configuration options:

- Segment Section:** A dropdown menu set to "14-Segment Display".
- Digit Count:** A dropdown menu set to "1".
- Start Position:** A dropdown menu set to "1".

At the bottom of the dialog are three buttons: "OK", "Cancel", and "Help". The "OK" button is highlighted with a blue border.

Digit Count: Controls the number of characters to be displayed.

Segment Section: Specifies the segment to which to be written.

Start Position: Specifies the start position.

Serial UART Read

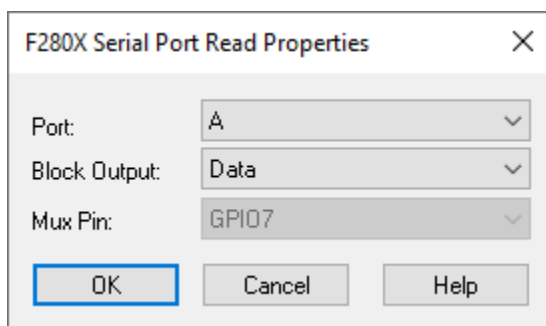
Target Category: AMD64, Arduino, C2407, Cortex M3, Delfino, F280x, F281X, Linux Raspberry Pi, MSP430, Piccolo, STM32

Target Sub-Category: Serial Uart

Description: The Serial UART Read block is a serial communication block for reading RS32 information. There is a 16-bit element transmit and receive queue. There is an interrupt-based driver that places receive characters in a receive queue and transmit characters in a transmit queue for servicing by the interrupt handler. You may read as many bytes as are available in the receive queue. You can query the current receive queue length using the drop down setting in Serial UART Read block.

Use [Serial UART Config](#) to configure the serial port.

Note: If you are performing HIL on the Arduino, you cannot use serial UART blocks in your diagram or include them in Extern Definition blocks. This is because Arduino HIL communication relies on UART-0 on the target. Consequently, it is necessary to remove all UART-0 usage in your diagram when using HIL to debug it.



Block Output: Determines the type of information the block produces.

Data: Eight bits of data received.

Port Status: Actual bits returned by the status register on the peripheral device.

Receive Queue Empty: Outputs 1 if the queue is empty; and 0 if the queue is not empty.

Receive Queue Length: Specifies the length of the receive queue.

Receive Queue Max Length: Specifies the maximum length of the receive queue.

Receive Queue Overrun: Outputs 1 if the queue has been overrun; and 0 if the queue has not been overrun.

Transmit Queue Full: Outputs 1 if the transmit queue is full; and 0 if it is not.

Transmit Queue Length: Specifies the length of the transmit queue.

Transmit Queue Max Length: Specifies the maximum length of the transmit queue.

Mux Pin: Selects which pin a given function is on.

Port: Selects the Comm port on the peripheral device. Click [here](#) for Arduino pin mapping. Click [here](#) for Raspberry Pi pin mapping.

Serial UART Write

Target Category: AMD64, Arduino, C2407, Cortex M3, Delfino, F280x, F281X, Linux Raspberry Pi, MSP430, Piccolo, STM32

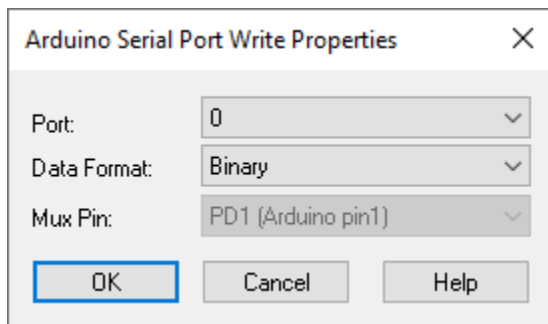
Target Sub-Category: Serial Uart

Description: The Serial UART Write block is a serial communication block for transmitting RS32 information. This block has two inputs: one is an enable pin and the other is for data. When the enable pin is high, the values entering the data pin are presented to the transmit queue. When the enable pin is zero, values entering the data pin are ignored.

The serial port queue is interrupt driven. You may write as many bytes as are free in the transmit queue and the interrupt handler will send the bytes out automatically. You can query the current transmit queue length using the Serial UART Write block.

Use [Serial UART Config](#) to configure the serial port.

Note: If you are performing HIL on the Arduino, you cannot use serial UART blocks in your diagram or include them in Extern Definition blocks. This is because Arduino HIL communication relies on UART-0 on the target. Consequently, it is necessary to remove all UART-0 usage in your diagram when using HIL to debug it.



Data Format: Selects the data format. Binary format sends the data in raw binary format. ASCII format sends the data in human-readable format. This parameter is available only for Arduino targets.

Mux Pin: Selects which pin a given function is on.

Port: Selects the Comm port on the peripheral device. Click [here](#) for Arduino pin mapping. Click [here](#) for Raspberry Pi pin mapping.

Set PWM Mode

Target Category: STM32

Target Sub-Category: PWM

Description: Each STM32 motor control timer (timers 1, 8, 20) has four PWM outputs with optional complementary outputs with deadband. Normally, however, only three are used for 3-phase motor control. The Set PWM Mode block lets you set the output mode of each pin. There are four output modes:

- **HiZ:** High impedance, or unconnected
- **Force 0:** Forced low to ground
- **Force 1:** Forced on to Vcc
- **PWM:** Controlled by PWM duty

This allows you to commute the phases of a BLDC motor based on Hall sensor input.

PWM Output Mode

Timer Unit: TIM1 ▾

CH1: HiZ ▾	CH1N: HiZ ▾
CH2: HiZ ▾	CH2N: HiZ ▾
CH3: HiZ ▾	CH3N: HiZ ▾
CH4: HiZ ▾	CH4N: HiZ ▾

OK
Cancel
Help

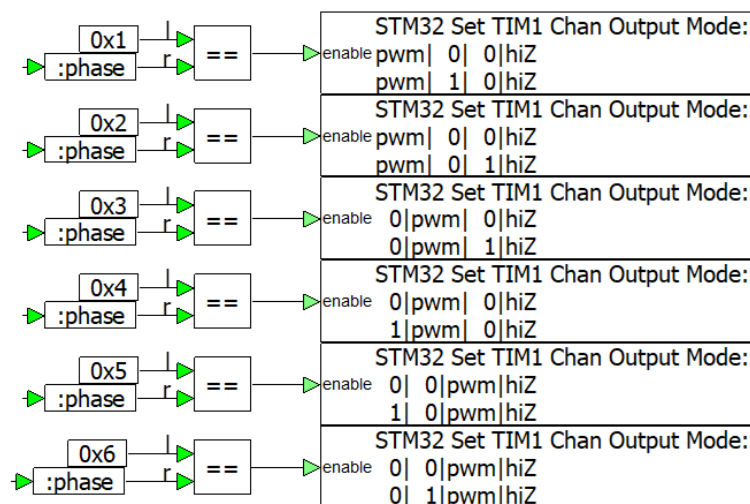
CH1-CH4: Specifies the output mode.

CH1N-CH4N: Specifies the output mode.

Timer Unit: Specifies the timer unit to be configured.

Example

This example uses six Set PWM Mode blocks, one for each Hall sensor value per electrical rotation. Note that the blocks have an enable pin, so they only execute if the input value is 1. The block face shows row 1=name, row 2 = setting of four main outputs, row 3 = setting of four complementary outputs. For any given Hall phase, the Set PWM Mode block will configure the three motor phase half H-bridges so that 1 is off (0/0), 1 is tied to ground (0/1), and 1 is modulated via PWM (pwm/pwm). The fourth column is hiZ/hiZ since it is unused.



Sigma Delta Filter Module

Target Category: Delfino, F280x, Piccolo

Target Sub-Category: ADC

Description: The Sigma Delta Filter Module block is a four-channel digital filter for current measurement and resolver position decoding on newer TI devices, including F2837x and F28004x. This block is used predominately in motor control applications.

Each input channel receives an independent delta-sigma modulator bit stream. The bit streams are processed by four individually-programmable digital decimation filters. The filter set includes a fast comparator filter for:

- Immediate digital threshold comparisons for over- and under-current monitoring
- Zeroes crossing detection

Additional information: Texas Instruments [TI 28377D Technical Reference Manual](#).

Channel: Specifies the input channel.

Comparator Filter

Comparator High Val: Indicates the upper threshold for the signal. If you activate **Enable High Val Interrupt** and the signal crosses the upper threshold, corresponding bits will be set in the Interrupt Flag (SDIFLG) register.

Comparator Low Val: Indicates the lower threshold for the signal. If you activate **Enable Low Val Interrupt** and the signal crosses the lower threshold, corresponding bits will be set in the Interrupt Flag (SDIFLG) register.

Enable High Val Interrupt: Used in conjunction with **Comparator High Val**, it sets the corresponding bits in the Interrupt Flag (SDIFLG) register when the signal crosses the upper threshold.

Enable Low Value Interrupt: Used in conjunction with **Comparator Low Val**, it sets the corresponding bits in the Interrupt Flag (SDIFLG) register when the signal crosses the upper threshold.

Filter Structure: Indicates the filter structure to be used.

Oversampling Ratio: Lets you choose between 1 and 32. Higher values yield higher signal precision, but slower overall sampling rates. The maximum comparator value is displayed with the specified ratio.

Data Filter

Enable Data Filter: Enables the data filter to monitor the bit stream.

Enable Data Ready Acknowledge Interrupt Flag: Sets the Data Ready Interrupt flag.

Filter Structure: Indicates the filter structure to be used.

Max Filter Value: Displays the maximum and minimum data filter values. This is a read-only parameter.

Oversampling Ratio: Lets you choose between 1 and 256. Higher values yield higher signal precision, but slower overall sampling rates.

Sync with PWM11.CMPD: Synchronizes the oversampling ratio with the PWM.

Word Size: Selects the word size of the output.

Enable Modulator Clk Failure Interrupt Flag: Sets the Modulator Clock Failure Interrupt flag.

Modulator Clk Mode: Lets you choose the speed of the modulator clock. If you are using Manchester encoded data, select **No Clk**.

SD1 C1 (clk): Chooses the pin that the clock comes in on.

SD1 D1(data): Chooses the pin that the data comes in on.

Unit: Specifies the unit number.

SPI Read

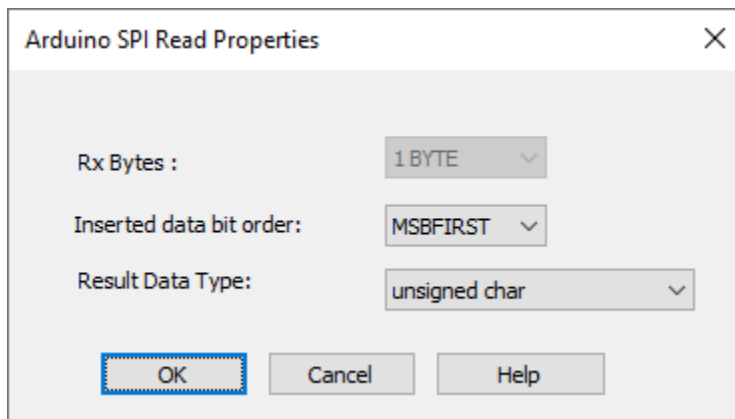
SPI Read for Arduino

Target Category: Arduino

Target Sub-Category: SPI

Description: The SPI Read block is a serial peripheral interface block for receiving data from certain hardware chips, like analog I/O chips. Use [SPI Config](#) to configure the hardware settings.

Additional information: [Using SPI slave to communicate between two Launchpads.](#)



Inserted data bit order: Sets the bit transmission order.

Result Data Type: Specifies the data type of the data value produced.

Rx Bytes: Indicates the number of bytes transmitted.

SPI Read for Linux Raspberry Pi

Target Category: Linux Raspberry Pi

Target Sub-Category: SPI

Description: The SPI Read block is a serial peripheral interface block for receiving data from certain hardware chips, like analog I/O chips. Use [SPI Config](#) to configure the hardware settings.

Additional information: [Using SPI slave to communicate between two Launchpads.](#)

The image shows a dialog box titled "SPI Read Properties". It contains three dropdown menus: "Unit" is set to "SPI0", "Block Output" is set to "Data", and "Result Data Type" is set to "unsigned char". At the bottom of the dialog are three buttons: "OK", "Cancel", and "Help".

Block Output: Determines the type of information the block receives.

Data: Data bits received.

Port Status: Hardware status.

Receive Queue Empty: True if empty.

Receive Queue Length: Specifies the length of the receive queue.

Receive Queue Max Length: Specifies the maximum length of the receive queue.

Receive Queue Overrun: True if queue has overflow.

Transmit Queue Full: True if full.

Transmit Queue Length: Specifies the length of the transmit queue.

Transmit Queue Max Length: Specifies the maximum length of the transmit queue.

Result Data Type: Specifies the data type of the data value produced.

Unit: Specifies the unit number.

SPI Read for C2407, Cortex M3, Delfino, F280x, F281X, MSP430, Piccolo, STM32

Target Category: C2407, Cortex M3, Delfino, F280x, F281X, MSP430, Piccolo, STM32

Target Sub-Category: SPI

Description: The SPI Read block is a serial peripheral interface block for receiving data from certain hardware chips, like analog I/O chips. Use [SPI Config](#) to configure the hardware settings.

Additional information: [Using SPI slave to communicate between two Launchpads.](#)

Block Output: Determines the type of information the block receives.

Data: Data bits received.

Port Status: Hardware status.

Receive Queue Empty: True if empty.

Receive Queue Length: Specifies the length of the receive queue.

Receive Queue Max Length: Specifies the maximum length of the receive queue.

Receive Queue Overrun: True if queue has overflow.

Transmit Queue Full: True if full.

Transmit Queue Length: Specifies the length of the transmit queue.

Transmit Queue Max Length: Specifies the maximum length of the transmit queue.

Data Bits to Extract from Received Bits: Sets the number of data bits, which can be smaller than the number of received bits. The received bit count is set in the SPI Config dialog box.

LSB of Extracted Data: Sets the least significant bit of data, allowing you to locate the data in the received word.

Radix Point: Sets the radix point.

Received Bits: Indicates the number of bits to be received. This value is set in [SPI Config block](#).

Result Data Type: Specifies the data type of the data value produced.

Unit: Specifies the unit number.

Word Size: Specifies the word size.

SPI Write

SPI Write for Arduino

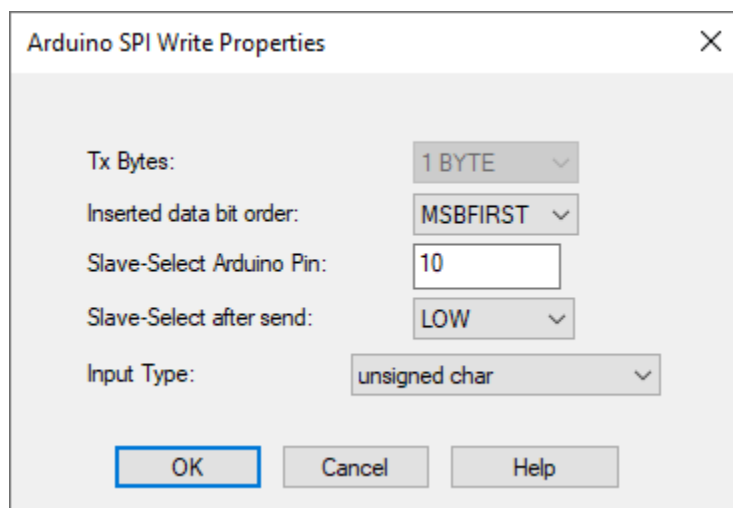
Target Category: Arduino

Target Sub-Category: SPI

Description: The SPI Write for Arduino block is a serial peripheral interface block for transmitting data to an Arduino board. The SPI queue is interrupt driven. You can write as many bytes as are free in the transmit queue and the interrupt handler will send the bytes out automatically. You can query the current transmit queue length using the [SPI Read for Arduino](#) block.

Use [SPI Config for Arduino](#) to configure the hardware settings.

Additional information: [Using SPI slave to communicate between two Launchpads.](#)



Input Type: Specifies the input data type.

Inserted data bit order: Sets the bit transmission order.

Slave Select/Arduino Pin: Selects the pin for the Slave Select signal. The default Slave Select pin is PB2 (Uno) and PB0 (Leonardo and Mega). You can verify the default Slave Select pin in the [SPI Config for Arduino](#) dialog box under Mux Pin Assignment. Click [here](#) for Arduino pin mapping.

Slave Select after send: Specifies the SS pin state upon completion of the byte transfer. The SS pin is set to LOW at the beginning of the transmission. It is set to the selected value after the transmission ends. In this way, you can send multi-byte words by keeping the SS pin LOW until the final byte.

Tx Bytes: Indicates the number of bytes transmitted.

SPI Write for Linux Raspberry Pi

Target Category: Linux Raspberry Pi

Target Sub-Category: SPI

Description: The SPI Write for Linux Raspberry Pi block is a serial peripheral interface block for transmitting data to a Raspberry Pi board. The SPI queue is interrupt driven. You may write as many bytes as are free in the transmit queue and the interrupt handler will send the bytes out automatically. You can query the current transmit queue length using the [SPI Read for ARM-Linux](#) block.

Use [SPI for ARM-Linux Config](#) to configure the hardware settings.

Additional information: [Using SPI slave to communicate between two Launchpads.](#)

Input Type: Specifies the input data type.

Slave Select After send: Specifies the SS pin state upon completion of the byte transfer. The SS pin is set to LOW at the beginning of the transmission. It is set to the selected value after the transmission ends. In this way, you can send multi-byte words by keeping the SS pin LOW until the final byte.

Unit: Specifies the unit number.

SPI Write for C2407, Cortex M3, Delfino, F280x, F281X, MSP430, Piccolo, STM32

Target Category: C2407, Cortex M3, Delfino, F280x, F281X, MSP430, Piccolo, STM32

Target Sub-Category: SPI

Description: The SPI Write block is a serial peripheral interface block for transmitting data to certain hardware chips, like analog I/O chips. The SPI queue is interrupt driven. You may write as many bytes as are free in the transmit queue and the interrupt handler will send the bytes out automatically. You can query the current transmit queue length using the [SPI Read](#) block.

Use [SPI Config](#) to configure the hardware settings.

Additional information: [Using SPI slave to communicate between two Launchpads.](#)

Data Bits to Insert to Tx Word: Specifies the number of data bits to insert into the transmitted word. Normally this is the same size as the transmitted word; however, it can be smaller if the data field is smaller than the transmitted bits. The

transmitted word size is set the SPI Config block. For instance, if you are sending 12 bits of DAC data, in a 16-bit word, set this value to 12, and choose an LSB offset if the 12 data bits are left shifted.

Input Type: Specifies the input data type.

LSB of Inserted Data: Sets the least significant bit of data, allowing you to position the data in the transmitted word.

Radix Point: Sets the radix point.

Transmitted Bits: Sets the number of bits to be transmitted.

Unit: Specifies the unit number.

Use Enable Pin: Produces an enable input pin to the block. If the value on the pin is zero, block operation will be suppressed.

Value ORed to Tx Word: Provides a value that will be logically OR'ed to the transmitted word. This can be useful to set control bits outside of the data bits in the transmitted word. If left to 0, this will have no effect.

Word Size: Specifies the word size.

UDP Read

Target Category: Arduino

Target Sub-Category: IoT > ESP8266WiFi

Description: The UDP Read block reads data from the UDP bus.

Data Element Type: Controls the data type of each output pin and the offset into the packet.

Data Pins: Specifies the number of input pins (128 max).

IP Address: Specifies the IP address of the target.

Pack Offsets: Iterates over all the pins and assigns consecutive ascending offsets to each pin.

Packet Byte Size: Specifies the size of the packet to be read (512 bytes max).

Set All Pin Types = Pin 1: Sets the data type of all pins to be the same as pin 1, and then performs a Pack Offset.

UDP Port: Specifies the port to exchange data. It is recommended to use ports greater than 49151.

UDP Write

Target Category: Arduino

Target Sub-Category: IoT > ESP8266WiFi

Description: The UDP Write block writes data to the UDP port on the Ethernet. The top “Tx” pin must have a value of 1 for the block to send data. The subsequent pins are data pins. The values presented on the data pins are sent to the UDP port.

Data Element Type: Controls the data type of each output pin and the offset into the packet.

Data Pins: Specifies the number of input pins (128 max).

IP Address: Specifies the IP address of the target.

Packet Byte Size: Specifies the size of the packet to written (512 bytes max).

Packet Offsets: Iterates over all pins and assigns consecutive ascending offsets to each pin.

Set All Pin Types = Pin 1: Sets the data type of all pins to be the same as pin 1, and then performs a Packet Offset.

UDP Port: Specifies the port to exchange data. It is recommended to use ports greater than 49151.

Target Interface

Target Category: Arduino, C2407, Cortex M3, Delfino, F280x, F281X, Linux AMD64 and Raspberry Pi, MSP430, Piccolo, STM32

Target Sub-Category: Target Interface

Description: The Target Interface block lets you validate and tune your control algorithm as it executes on a target device. When you simulate the diagram, the Target Interface block automatically downloads the OUT or ELF file to the target and begins executing the code on the target. While the target executable runs, you can communicate with the target via the inputs and outputs on the Target Interface block. Note the following:

- Because the target always runs in real-time, you should configure the diagram to run in real-time mode when communicating with the target. This way, Embed is in sync with the target.

- If you are performing HIL on the Arduino, you cannot use serial UART blocks in your diagram or include them in Extern Definition blocks. This is because Arduino HIL communication relies on UART-0 on the target. Consequently, it is necessary to remove all UART-0 usage in your diagram when using HIL to debug it.

To probe stack and heap usage, use the [Get Target Stack and Heap](#) command.

Block Title: Indicates an optional name that appears on the Target Interface block.

Connectors: Specifies the number of inputs and outputs on the compound block from which the code was generated.

Embedded Target Support Version: Indicates the version number of the driver board in use.

Keep Target Running: Allows the embedded target application to continue running even after you quit Embed.

Sample Rate: Specifies the sample rate at which the embedded algorithm should run. The default is 10kHz.

Show CPU Utilization: Creates an extra output that displays the CPU utilization on the target while your target execution file runs. CPU usage only works with Timer 2 on Delfino, F280x, F281x, and Piccolo targets.

Sync Target to this Block: Synchronizes the target to your simulation. It waits until the data from the PC is presented; then the PC waits for the target to operate for one timer interrupt interval. This parameter is not available for Arduino or Linux targets.

Target Board: Specifies the target card if multiple cards are supported. Board numbers are zero through three.

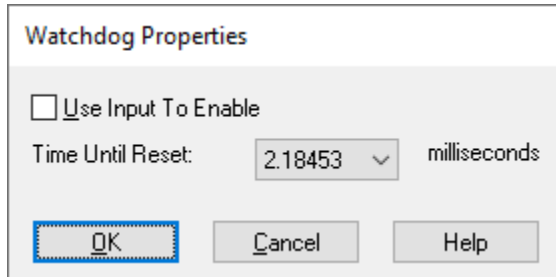
Target Execution File: Indicates the complete pathname of the OUT or ELF file previously generated. If you are not sure of the pathname, click on the ... button to locate the target execution file.

Target Frequency (MHz): Indicates the speed of your CPU. Embed needs to know the speed for the timing to be accurate.

Watch Dog

Target Category: C2407, Cortex M3, Delfino, F280x, F281X, MSP430, Piccolo

Description: The Watch Dog block enables a hardware watchdog. In an embedded system, it's customary to have a hardware watch dog. The watch dog can force an automatic system restart when the system begins to misbehave.



Time Until Reset: Sets the interval for which the Watch Dog block expects input.

Use Input To Enable: Indicates that the block has enabling input that you can use to suppress sending input to the watch dog, which will force a restart.

Web Server

Target Category: Cortex M3

Description: The Web Server block automatically generates code to create a web page running on your embedded target that displays information from your Embed diagram running on the target. You can take data entered on the web page and use it in the Embed diagram running on the target. You can also choose whether to update the web page data automatically at a set rate or have users click a button to update a control on the web page.

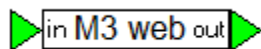
The Web Server block can create a complete web page automatically or let you build a custom web page with graphics and place Embed controls where you like on the page.

An automatically created web page looks like this:




Configuring a sample Web Server block

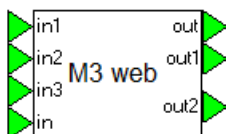
To create a web page, first insert a Web Server block into your diagram.



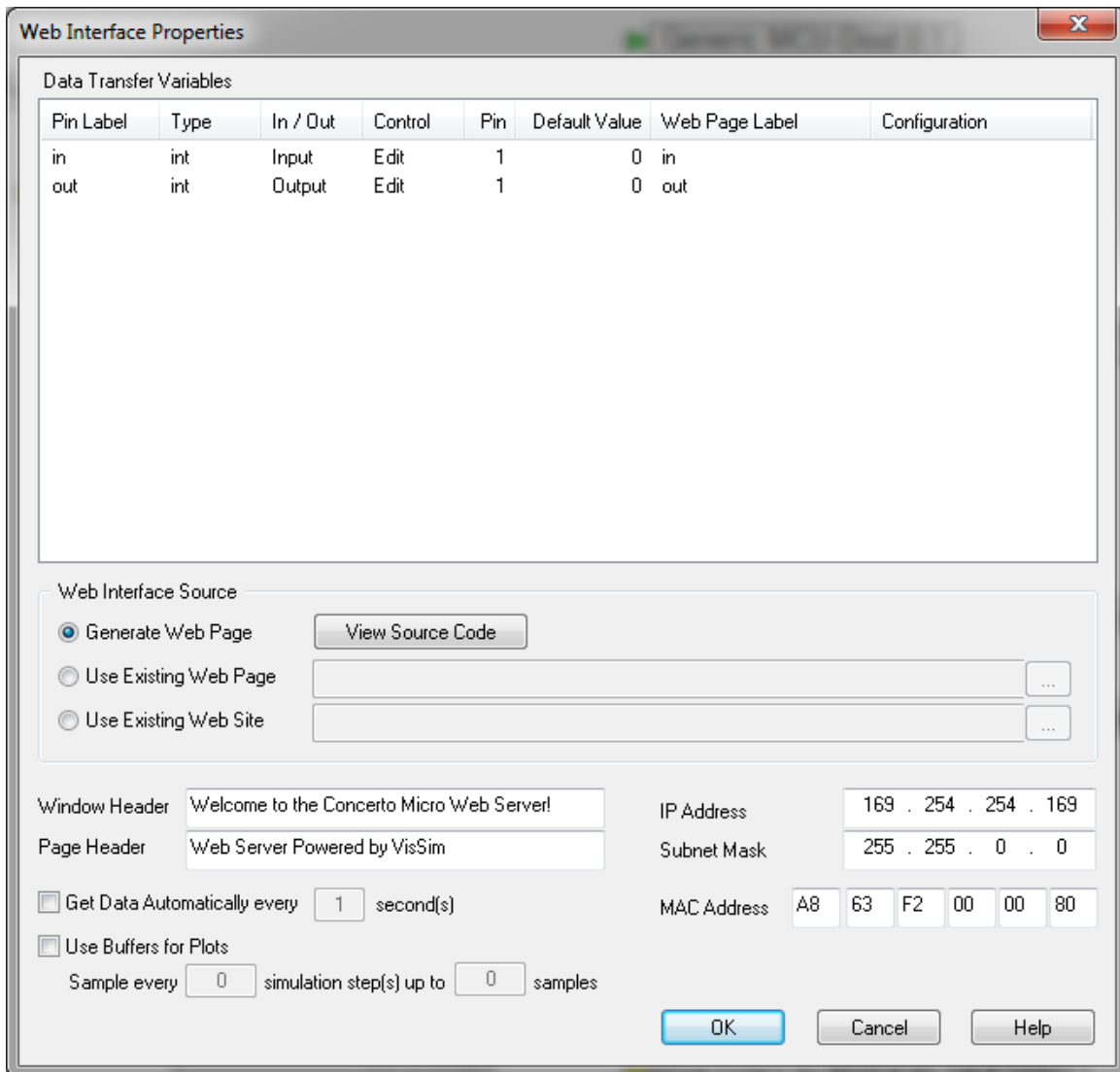
By default, the Web Server block has one input and one output.

To configure the Web Server block

1. Click **Edit > Add Connector** or  to add input and output pins to the block.
2. Click the left side of the block to add input pins and the right side to add output pins.



3. Right-click **Web Server** to display the Web Interface Properties dialog box to configure your web page:



4. Make the appropriate selections. Descriptions of the dialog box selections are in the next several sections.
5. Click **OK**, or press **ENTER**.

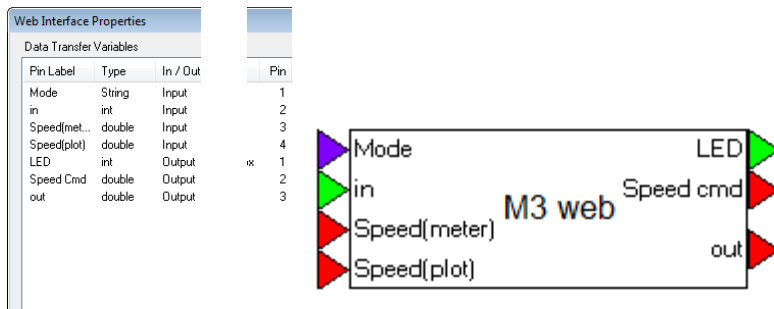
Configuring input and output connector pins

The input and output connector pins are listed in the Data Transfer Variables box in the Web Interface Properties dialog box.

- To edit any property of a pin, double click it.

Pin Labels, Type, In/Out, and Pin

Pin labels appear on the Web Server block in Embed.



The Type column refers to the data type of the pin. There are four data types for connector pins: char, int, double, string.

If you enable View > Data Types, connector pins are colored according to the following table.

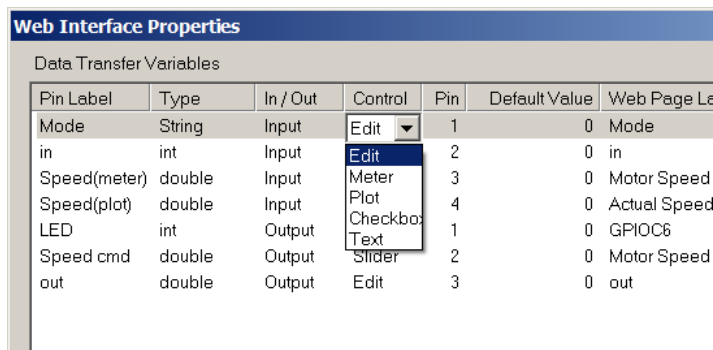
Data Type	Connector Pin Color
char	Green
int	Green
double	Red
string	Purple

The In/Out column indicates whether the connector pin is an input pin or an output pin.

The Pin column indicates the number of the pin. Pin numbers start at 1, which corresponds to the top connector pin.

Control

The Control column lets you create different controls or “widgets” that will be displayed on the web page. The control options are Checkbox, Edit, Meter, Plot, Slider, and Text. When you double-click a control option, a drop-down menu appears with your control choices.



Control	Type	Description
Checkbox	Input	Send a 0 or 1 to the target
Edit	Input and Output	Send the text or numeric data from the web page to the target or receive the data from the target and display it on the web page
Meter	Output	Draw a horizontal thermometer on the web page

Plot	Output	Plot the time history of a variable on the web page
Slider	Input	Dynamically modify a value
Text	Input	Display text on the web page

Web Server Powered by Embed

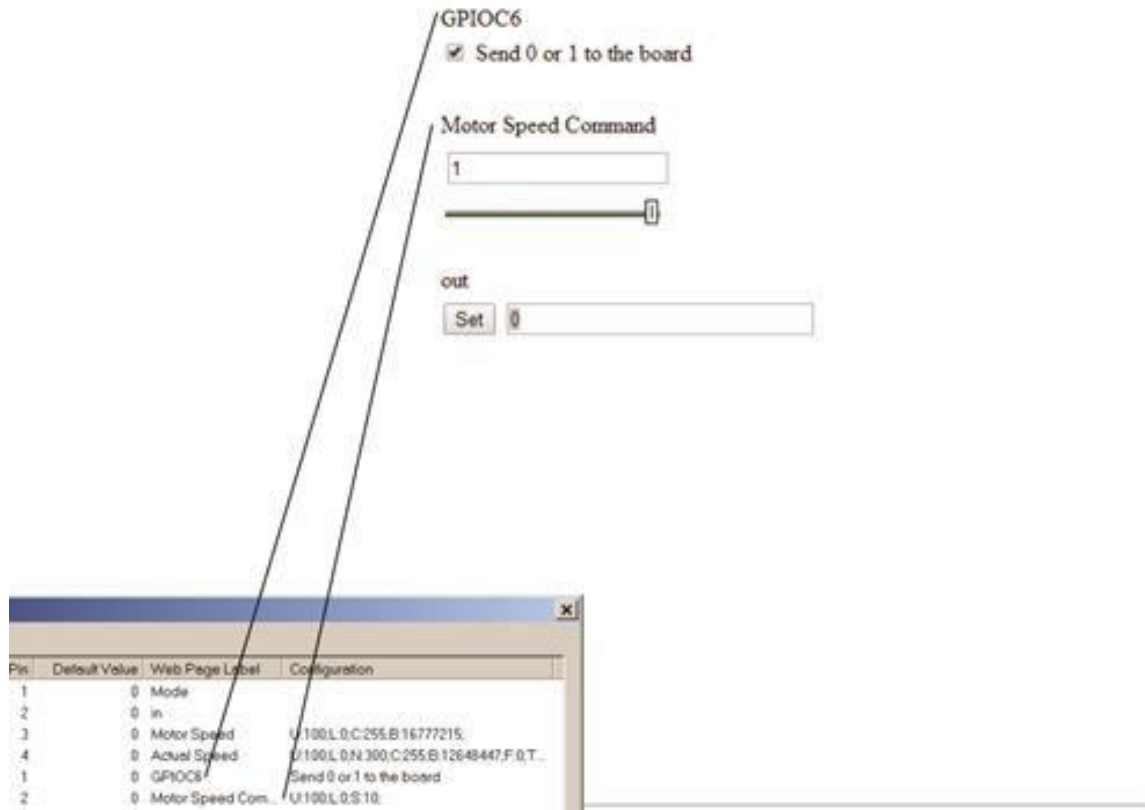
Right click on the meter bar and plot to access dialog boxes to change their appearances.

Web Page Labels

The Web Page Labels column lets you create labels for the controls that will be displayed on the web page. To edit a web page label, simply click the mouse over the label. A box appears around the label to indicate you are in edit mode.

Web Interface Properties							
Data Transfer Variables							
Pin Label	Type	In / Out	Control	Pin	Default Value	Web Page Label	Configurat
Mode	String	Input	Edit	1	0	Mode	
in	int	Input	Edit	2	0	in	
Speed(meter)	double	Input	Meter	3	0	Motor Speed	U:100.L0.0
Speed(plot)	double	Input	Plot	4	0	Actual Speed	U:100.L0.0
LED	int	Output	Check...	1	0	GPIOC6	Send 0 or 1 to the board
Speed cmd	double	Output	Slider	2	0	Motor Speed Com...	U:100.L0.0
out	double	Output	Edit	3	0	out	

Web Server Powered by Embed



Default Value

The Default Value column indicates the initial value to display on the web page before data is sent.

Configuration

The Configuration column lets you choose range and colors for the web page controls that need them.

Configuring the Web Address

You configure the web page address in the lower portion of the Web Interface Properties dialog box.

Window Header	Welcome to the Concerto Micro Web Server!	IP Address	169 . 254 . 254 . 169
Page Header	Web Server Powered by Embed	Subnet Mask	255 . 255 . 0 . 0
<input checked="" type="checkbox"/> Get Data Automatically every	<input type="text" value="1"/> second(s)	MAC Address	A8 63 F2 00 00 80
<input type="checkbox"/> Use Buffers for Plots	Sample every <input type="text" value="0"/> simulation step(s) up to <input type="text" value="0"/> samples		
		<input type="button" value="OK"/> <input type="button" value="Cancel"/> <input type="button" value="Help"/>	

IP Address and Subnet Mask

You must select an **IP address** so you can find the web page on the internet or LAN. Make sure the IP address you assign to the target board is unique on your network. You must also set the **Subnet Mask**. If accessing on a LAN, be sure to use the proper subnet so that the page can be found. You may need to consult your Network Administrator for your IP address and Subnet Mask.

MAC Address

The **MAC Address** is a label on your target board. Make sure the address you enter matches the address on the board.



Automatic Data Update

When you activate **Get Data Automatically**, it allows automatic update of diagram values to the web page and will take data from the page and send it to the diagram at the specified interval. You must also activate the corresponding **Get Data Automatically** parameter on the web page.

Window Header

The **Window Header** specifies the title in the Browser tab or window.

Page Header

The **Page Header** indicates the title on the web page.

Choosing the Web Interface Source

The **Web Interface Source** section gives you the option to have Embed generate the entire web page, or integrate Embed Web Controls into your custom web page along with your graphics. Embed supports a flash-based file system that lets your web page use individual graphic images like PNG and GIF, which can be displayed on your web page.

Web Interface Source

Generate Web Page View Source Code

Use Existing Web Page ...

Use Existing Web Site ...

Generate web page

If you choose this parameter, Embed generates an INDEX.HTM file and places it in the code generation directory (`<install-path>\cg`). Typically, you start by creating a web page using **Generate Web Page** because it provides a basic

script to communicate with the embedded web server. You can view the generated source code by clicking **View Source Code**.

Use existing web page

If you choose this parameter, you must specify the path to your existing single web page. You use **Use Existing Web Page** when you want to edit the web page created with **Generate Web Page**.

Use existing web site

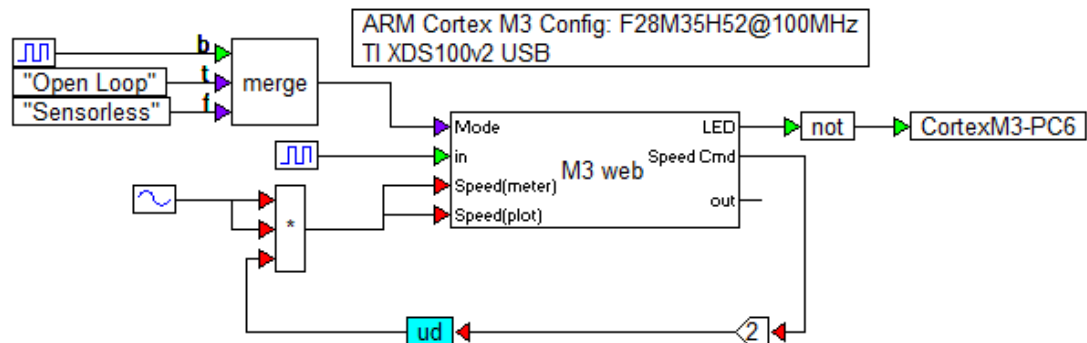
If you choose this parameter, you can create your own web site with several linked web pages with your own controls. To do so:

- There must be an INDEX.HTM
- You must specify the path to the directory containing INDEX.HTM

All the files in the above specified directory and subdirectories will be included in the generated C code.

Using the Web Server block in a diagram

A sample diagram — WebTest05 — with a Web Server block is included under Examples > Embedded > CortexM3 > Webserver.



The Web Server block is treated like any other block in the Embed diagram. It takes input values and supplies output values. The Web Server block operates asynchronously: it posts its input data to the page but does not wait for a response. The output data it provides on the output pins is the contents of the local buffer from the last asynchronous data received from the page.

In the sample diagram above, connectors are color coded to data type:

Green: integer

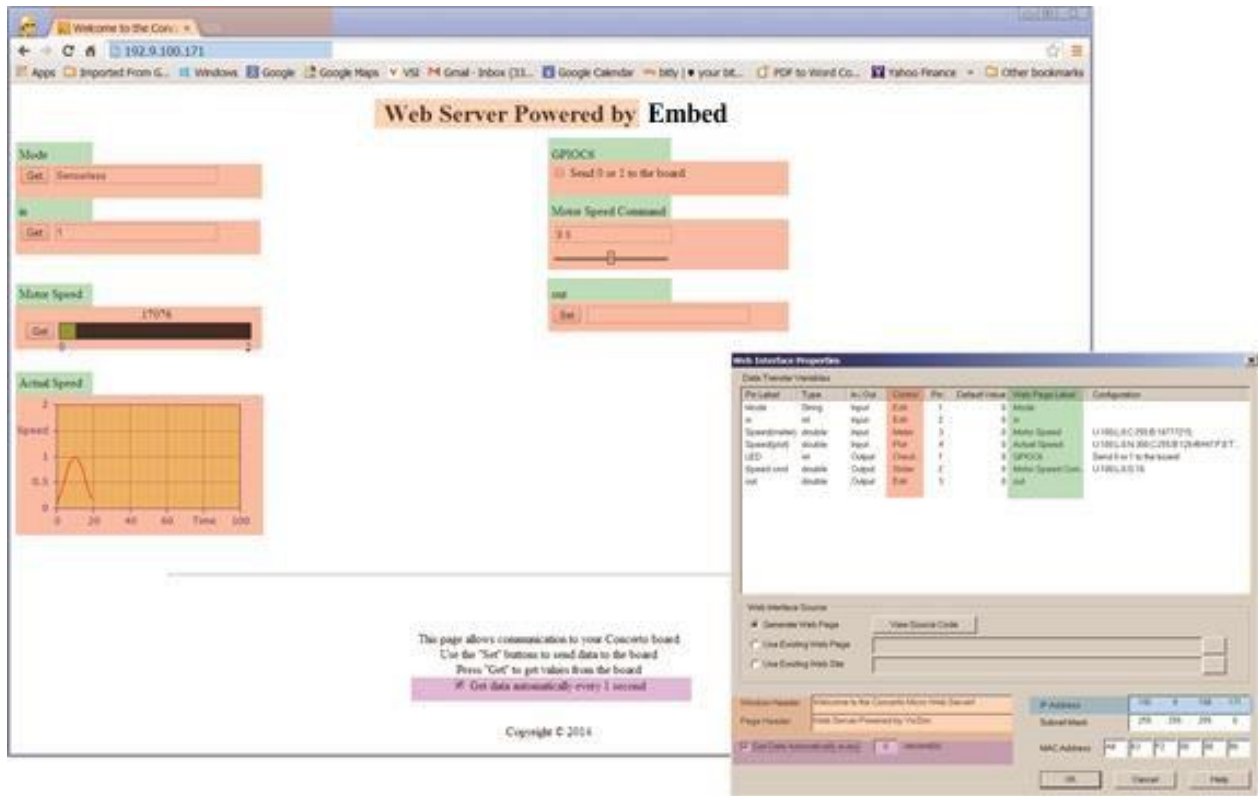
Purple: string

Red: floating point

The diagram generates some simple data merely to exercise the control on the web page.

The Mode input is a string that updates dynamically. The “in” input is an edit display that will show the integer numeric value of the square wave input. The Speed (meter) input is identical to the Speed (plot) input and is a sin squared to produce a value varying between zero and one scaled by the Speed Cmd slider output and a gain of two. The LED output is a check box with value zero or one. This is connected to GPIO PC6 which is wired to an LED on the Texas Instruments Control card. This lets you turn the LED on and off from the web page. The “out” pin represents the floating-point value entered into the Out control on the web page. It is not used in this sample diagram.

When you compile the diagram, the web page below is automatically generated. Using the web page, you can observe and control your application on the target board.



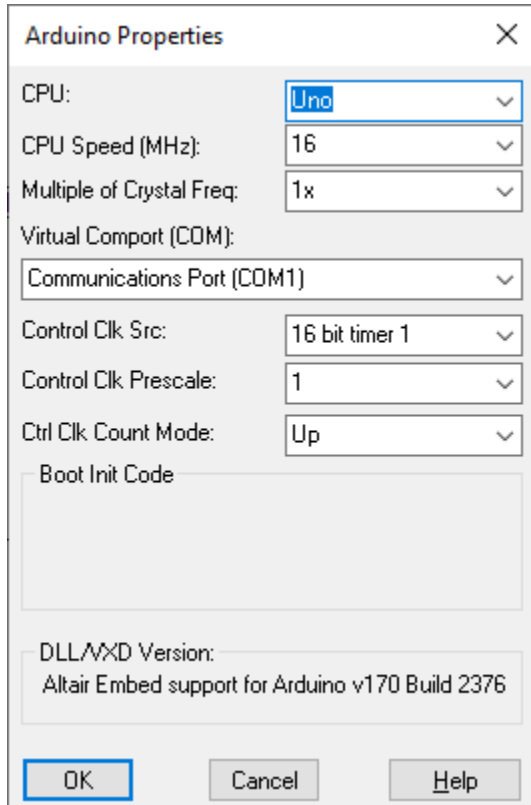
Using the Target Config blocks

The Target Config block configures the diagram for a specific embedded target. There can only be one Target Config block in a diagram.

Using Arduino Config

Target Category: Arduino

The Arduino Config lets you configure settings for the Arduino. After you enter the settings, an Arduino Config block is inserted into the diagram.



Boot Init Code: Specifies user C code to initialize the chip.

Control Clk Prescale: Chooses the prescale factor for the PWM based control clock. If the PWM is also used as an output peripheral, this setting must match the setting of the PWM block in the diagram.

Control Clk Src: Chooses periodic sampling interrupt source for the main control loop in Embed.

CPU: For some targets, there is a family of CPU types. It is important to select the exact CPU type for the specified target.

CPU Speed: Indicates the speed of the CPU.

Ctrl Clk Count Mode: Chooses the count mode for the PW-based control clock. If the PWM is also used as an output peripheral, this setting must match the setting of the PWM block in the diagram.

Multiple of Crystal Freq: Indicates the multiple of crystal frequency.

Virtual Comport: Chooses the serial communications port that corresponds to your Arduino. If you do not know the correct serial port number, start the Arduino IDE and click **Tools > Port** to find it.

Using ARM Cortex M3 Config

Target Category: Cortex M3

The ARM Cortex M3 Config lets you configure settings for the ARM Cortex M3. After you enter the settings, an ARM Cortex M3 Config block is inserted into the diagram.

Clock Source: Chooses the hardware source for the system clock.

Ctrl Clk Count Mode: Chooses the count mode for the PWM-based control clock. If the PWM is also used as an output peripheral, this setting must match the setting of the PWM block in the diagram.

Control Clk Prescale: Chooses the prescale factor for the PWM based control clock. If the PWM is also used as an output peripheral, this setting must match the setting of the PWM block in the diagram.

Control Clk Src: Sets the periodic sampling interrupt source for the main control loop in Embed.

CPU: For some targets, there is a family of CPU types. It is important to select the exact CPU type for the specified target.

CPU Speed: Indicates the speed of the CPU.

Enable Interactive Peripheral Mode: Causes a pre-configured OUT file (whose only task is to read and write peripheral data) to be downloaded to the target. All ADC, Quadrature Encoder, PWM, and GPIO blocks in the diagram will read and write actual values from the on-chip peripherals at the speed of the JTAG (~150Hz).

EPWM Interrupt Event: Chooses PWM event if PWM interrupt is selected as the Control Clk Src.

HSPCLK: Drives the ADC sample timer.

JTAG Connection: Indicates the JTAG driver to be used for HotLink communication.

LSPCLK: Drives the serial and SPI ports.

Multiple of Crystal Freq: Indicates the multiple of crystal frequency.

Preload Out File: Lets you debug the slave core while loading your application in the master core.

	F2812	F2812
Encoder Channel A	QEP1	QEP3
Encoder Channel B	QEP2	QEP4

Encoder Index	CAP3	CAP5
---------------	------	------

Use Custom Linker Cmd File: Overrides using the default linker command file. When activated, you must enter the path to your custom linker command file in the corresponding text box. If there are spaces in the path, enclose the path in double quotation marks.

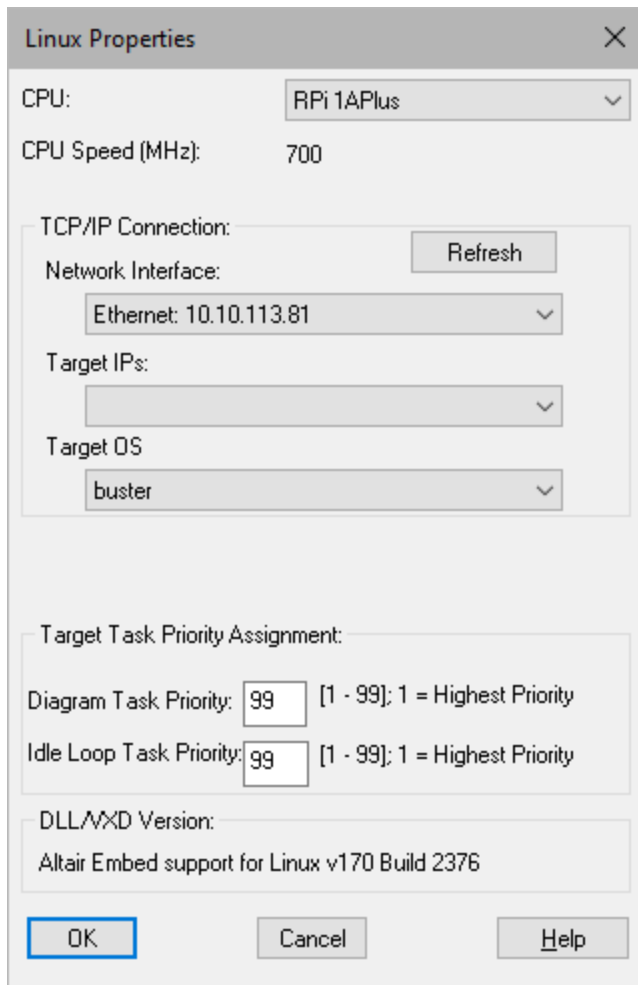
Using the Linux Config

Target Category: Linux AMD64 and Raspberry Pi

The Linux Config lets you configure settings for the AMD64 and Raspberry Pi. After you enter the settings, a Linux Config block is inserted into the diagram.

Linux Config uses TCP/IP to connect to an AMD64 or Raspberry Pi device. A daemon runs on the Raspberry Pi for this purpose. The Linux Config uses these ports:

Protocol	Port
TCP	50009 57893
UDP	50002 50003 57892



CPU: Sets the CPU type for the Raspberry Pi or AMD64.

CPU Speed: Indicates the speed of the CPU.

Target Task Priority Assignment

Diagram Task Priority: Sets the Linux priority of diagram task execution. Specify the priority as an integer in the range of 1 – 99, where 1 is the highest priority.

Idle Loop Priority: Sets the Linux priority of idle loop execution. Specify the priority as an integer in the range of 1 – 99, where 1 is the highest priority.

TCP/IP Connection

Network Interface: Chooses the network interface.

Refresh: Reloads the current IP of the host machine and connected Raspberry Pi or AMD64.

Target IPs: Chooses the IP of the connected Raspberry Pi or AMD64.

Target OS: Chooses the operating system of the connected Raspberry Pi or AMD64. The supported operating system for Raspberry Pi is Buster; the supported operating system for AMD64 is Xenial Xerus.

Using the F240X Config

Target Category: F240X

The F240X Config lets you configure settings for the F240X. After you enter the settings, an F240X Config block is inserted into the diagram.

F240x Properties [X]

CPU: LF2407 [v]

CPU Speed (MHz): 40 [v]

Multiple of Crystal Freq: 4x [v]

JTAG connection: [v]

Control Clk Src: CMP1INT [v]

Control Clk Prescale: /16 [v]

Ctrl Clk Count Mode: Up [v]

Boot Init Code

DLL/AXD Version: Altair Embed support for C2407 v170 Build 2376

OK Cancel Help

Boot Init Code: Specifies user C code to initialize the chip.

Control Clk Prescale: Chooses the prescale factor for the PWM based control clock. If the PWM is also used as an output peripheral, this setting must match the setting of the PWM block in the diagram.

Control Clk Src: Chooses periodic sampling interrupt source for the main control loop in Embed.

CPU: For some targets, there is a family of CPU types. It is important to select the exact CPU type for the specified target.

CPU Speed: Indicates the speed of the CPU.

Ctrl Clk Count Mode: Chooses the count mode for the PWM based control clock. If the PWM is also used as an output peripheral, this setting must match the setting of the PWM block in the diagram.

JTAG Connection: Indicates the JTAG driver to be used for HotLink communication.

Multiple of Crystal Freq: Indicates the multiple of crystal frequency.

Using F28X Config

Target Category: Delfino, F280x, F281X, Piccolo

The F28x Config block lets you choose the setting for the Delfino, F280X, F281X, and Piccolo. After you enter the settings, F28x Config block is inserted into the diagram.

F28x Config for Delfino, F280X, and Piccolo

F28x Properties

CPU: F280049

Enable Interactive Peripheral Mode

CPU Speed (MHz): 100

Multiple of Crystal Freq: 10x

HSPCLK: SYSCLK/1 100 MHz

JTAG connection: TI XDS100v2 USB

Control Clk Src: 32 bit timer 2

Control Clk Prescale: 1

Use Custom Linker Cmd File:

DLL/XXD Version:
Altair Embed support for F280X v170 Build 2376

Clock Source: Internal Oscillator 1

AUXOSCCLK=Internal Oscillator 2

LSPCLK: SYSCLK/4 25 MHz

EPWM Interrupt Event: CTR = 0

Ctrl Clk Count Mode: Down

OK Cancel Help

Auxiliary Oscillator Clock: Chooses the clock for the USB. This parameter is available for newer Texas Instruments devices, like the F280049. Refer to the Texas Instruments specification sheet for more details.

Clock Source: Chooses the hardware source for the system clock.

Control Clk Prescale: Chooses the prescale factor for the PWM-based control clock. If the PWM is also used as an output peripheral, this setting must match the setting of the PWM block in the diagram.

Control Clk Src: Chooses periodic sampling interrupt source for the main control loop in Embed.

CPU: For some targets, there is a family of CPU types. It is important to select the exact CPU type for the specified target.

CPU Speed: Indicates the speed of the CPU.

Ctrl Clk Count Mode: Chooses the count mode for the PWM based control clock. If the PWM is also used as an output peripheral, this setting must match the setting of the PWM block in the diagram.

Enable Interactive Peripheral Mode: Enables Embed to read and write GPIO, ADC, PWM, and quadrature encoder data on a target MCU from Embedded blocks in an Embed diagram. This mode can only support 150 Hz data rate.

EPWM Interrupt Event: Choose PWM event if PWM interrupt is selected as the Control Clk Src.

HSPCLK: Drives the ADC sample timer.

JTAG Connection: Indicates the JTAG driver to be used for HotLink communication.

LSPCLK: Drives the serial and SPI ports.

Preload Out File: If you have a dual-core target, **Preload Out File** will download and start an executable image to the master core. This parameter is available for dual-core devices.

Multiple of Crystal Freq: Indicates the multiple of crystal frequency.

Use custom linker cmd file: Overrides using the default linker command file. When activated, you must enter the path to your custom linker command file in the corresponding text box. If there are spaces in the path, enclose the path in double quotation marks.

F28x Config for F281X

The screenshot shows the 'F281x Properties' dialog box with the following settings:

- CPU: F2812
- CPU Speed (MHz): 150
- Multiple of Crystal Freq: 5x
- JTAG connection: TI XDS100v2 USB
- Control Clk Src: 32-bit timer 0
- Control Clk Prescale: 1
- Ctrl Clk Count Mode: Down
- Boot Init Code: (empty text box)
- DLL/VXD Version: Altair Embed support for F281X v170 Build 2376

Boot Init Code: Specifies user C code to initialize the chip.

Control Clk Prescale: Chooses the prescale factor for the PWM based control clock. If the PWM is also used as an output peripheral, this setting must match the setting of the PWM block in the diagram.

Control Clk Src: Chooses periodic sampling interrupt source for the main control loop in Embed.

CPU: For some targets, there is a family of CPU types. It is important to select the exact CPU type for the specified target.

CPU Speed: Indicates the speed of the CPU.

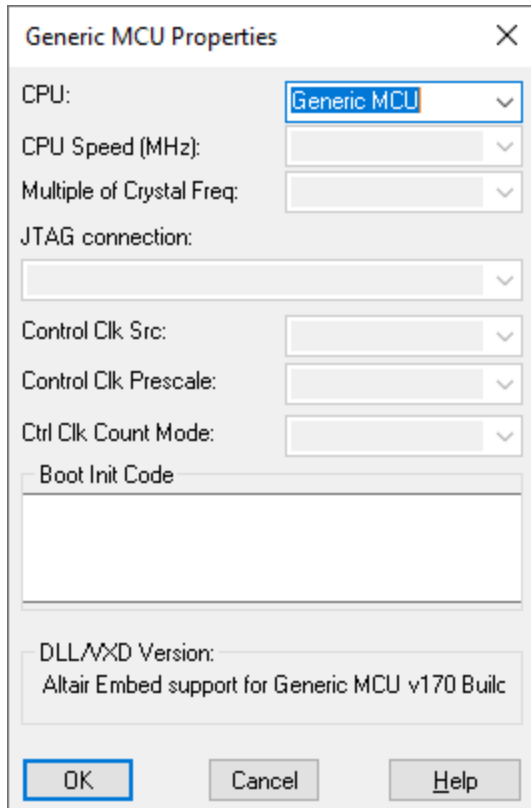
Ctrl Clk Count Mode: Chooses the count mode for the PWM based control clock. If the PWM is also used as an output peripheral, this setting must match the setting of the PWM block in the diagram.

JTAG Connection: Indicates the JTAG driver to be used for HotLink communication.

Multiple of Crystal Freq: Indicates the multiple of crystal frequency.

Using Generic MCU Config

The Generic MCU Config lets you choose the setting for a generic MCU. After you enter the settings, a Generic MCU Config block is inserted into the diagram.



Boot Init Code: Specifies user C code to initialize the chip.

Control Clk Prescale: Chooses the prescale factor for the PWM based control clock. If the PWM is also used as an output peripheral, this setting must match the setting of the PWM block in the diagram.

Control Clk Src: Chooses periodic sampling interrupt source for the main control loop in Embed.

CPU: For some targets, there is a family of CPU types. It is important to select the exact CPU type for the specified target.

CPU Speed: Indicates the speed of the CPU.

Ctrl Clk Count Mode: Chooses the count mode for the PWM based control clock. If the PWM is also used as an output peripheral, this setting must match the setting of the PWM block in the diagram.

JTAG Connection: Indicates the JTAG driver to be used for HotLink communication.

Multiple of Crystal Freq: Indicates the multiple of crystal frequency.

Using MSP430 Config

The MSP430 Config lets you configure settings for the MSP430. After you enter the settings, an MSP430 Config block is inserted into the diagram.

MSP430 Config

CPU sub type: F2274

JTAG connection: USB

Spy Bi Wire

LFX1 Crystal Speed (Hz): 11680

XT2 Crystal Speed (MHz): 0

DCO Speed (MHz): 8.37

Internal Clocks

	Source	Divider
ACLK:	LFX1	/1
MCLK:	DCO	/1
SMCLK:	DCO	/1

Low Power Mode: LPM0: CPU (MCLK) off

DLL/VXD Version: Altair Embed support for MSP430 v170 B

Compiler: Code Composer Studio

7 Segment LCD:

LCD type: SBLCDA2

Leading Digit: LCDM1

Digits Ascend in Memory

Sample Interrupt Src: TIMER TA0

Timers

TIMER TA0 Clk Src: TA0CLK

TIMER TB0 Clk Src: TBOCLK

BASIC TIMER Clk Src: ACLK

Timer A1 Clk Src: ACLK

Timer A1 Clk Src: ACLK

Timer A1 Clk Src:

Timer A1 Clk Src:

7-Segment LCD

Digits Ascend in Memory: Specifies whether the digits ascend or descend in memory.

LCD Type: Chooses one of the Softbaugh LCD devices or a custom device.

Leading Digit: Chooses the LCD memory location for the starting digit of the multi-digit, 7-segment display.

Compiler: Fixed to Code Composer Studio™ for version 14 and 15.

CPU Sub Type: Specifies the specific MSP430 CPU number. When you choose a subtype, note that other dialog box values may change to reflect the default values of the subtype. For example, when you choose F2012, SMCLK Source changes to DCO.

Internal Clocks

ACLK: Lets you select the source and the divider for the auxiliary clock.

MCLK: Lets you select the source and the divider for the main (or master) clock.

SMCLK: Lets you select the source and the divider for the submain clock (SMCLK). This clock is often used to drive peripherals like timers, serial port, I2C, and SPI.

JTAG Connection: Indicates the port to which the JTAG pod is connected. Activate **Spy Bi Wire** to use this method of JTAG HotLink communication.

LFX1 Crystal Speed, XT2 Speed, DCO Speed

DCO Speed: Indicates the speed of the CPU. You can adjust the DCO speed by clicking the **Setup DCO and XT Clocks** button.

LFXT1 Crystal Speed: Lets you specify custom external crystal speed. The default is 32768Hz. Check hardware documentation for the correct crystal speed. You can adjust the LFXT1 crystal speed by clicking the **Setup DCO and XT Clocks** button.

XT2 Speed: Specifies the speed of external crystal 2. This parameter is not available on all CPU subtypes. You can adjust the XT2 speed by clicking the **Setup DCO and XT Clocks** button.

Low Power Mode

LPM0: Turn off CPU (MLCK); all other clocks running

LPM1: Turn off CPU and DCO; ACLK and SMCLK running

LPM2: Turn off CPU and SMCLK; ACLK and DCO running

LPM3: Turn off CPU, SMCLK, and DCO; ACLK running

LPM4: All clocks turned off

None: All clocks running

Sample Interrupt Src: Specifies the periodic interrupt source for the main Embed control loop. (The interrupt rate is automatically set to the Embed sample rate at time of code generation.)

Setup DCO and XT Clocks: Lets you set up the DCO and external clocks. For more information, see [Setting up DCO and external clocks](#).

Timers: Select the clock sources for each timer. The ACLK and SMCLK are internal clocks. The TACLK and TBCLK are external clocks.

Setting up DCO and external clocks

To set up the DCO and external clocks, click on the Setup DCO and XT Clocks button in the MSP430 Config dialog box.

MSP430 1xx, 2xx, and 20xx subtypes

DCO

BCSCTL1.RSELx: Defines a nominal frequency in conjunction with the DCOCTL.DCOx and DCOCTL.MODx. See also Texas Instruments [MSP430X1 User Guide](#).

DCOCTL.DCOx: Defines a nominal frequency in conjunction with the BCSCTL1.RSELx and DCOCTL.MODx. See also Texas Instruments [MSP430X1 User Guide](#).

DCOCTL.MODx: Defines a nominal frequency in conjunction with the DCOCTL.DCOx and BCCTL1.RSELx. See also Texas Instruments [MSP430X1 User Guide](#).

DCO Speed: This is a read-only setting. It calculates the DCO speed based on either the factory calibrated setting or the user-specified calibrated setting, determined by the RSELx, DCOx, and MODx settings.

Use Calibrated Setting: Only for 2xx. Uses factory calibrated setting for the DCO.

LFXT1

Capacitance: Specifies the value of desired on-chip capacitance.

Crystal Resonator ranges: Higher frequency ranges supported by the 2xx, external.

Crystal Speed: Specifies the speed of external crystal one.

High Frequency Mode: Sets to true if LFXT1 is greater than or equal to 400kHz.

Source: For 2xx subtypes, you can specify the source of the XT1 clock. The XT1 clock source can be external or internal. Your choices are:

32768 Hz crystal: Standard low frequency, external

VLOCLK: Very low frequency, internal

Reserved: Do not use

Digital clock: Digital oscillator (not crystal), external

Digital 0.4: Digital oscillator (not crystal), external

XT2

Crystal Speed: Specifies the speed of the second external crystal. If this value is 0, Embed assumes there is not a second external oscillator.

Range: Specifies the frequency range for the second external crystal.

The MSP430 FLL+ Config dialog box

This dialog box lets you configure settings for the MSP430 3xx and 4xx subtypes.

FLL

Additional Multiplier: Specifies an additional multiplier for calculating the DCO speed.

DCO Speed: Indicates the DCO speed, which is based on the **Multiple of Crystal Freq** and **Additional Multiplier** values.

Multiple of Crystal Freq: Indicates the multiple of crystal frequency. It defines the DCO speed by multiplying this value by the crystal speed.

LFXT1

Capacitance: Specifies the value of desired on-chip capacitance.

Crystal Speed: Specifies the speed of external crystal one.

High Frequency Mode: Sets to true if LFXT1 is greater than or equal to 400kHz.

XT2

Crystal Speed: Specifies the speed of the second external crystal. If this value is 0, Embed assumes there is not a second external oscillator.

The MSP430 Unified Clock System (UCS) dialog box

This dialog box lets you configure settings for the MSP430 5xx subtypes.

FLL

DCOCLK: Scaled version of DCOCLKDIV.

DCOCLKDIV: Read-only value, based on the FLL Reference Clk Src, FLL Reference Clk Div, and Multiple of Crystal Freq values.

FLLD Prescaler: Applies a scale factor to the DCOCLK.

FLL Reference Clk Src: Selects the clock source for the frequency lock loop to calculate the value of DCOCLKDIV.

FLL Reference Clock Div: Divides the clock source to calculate the value of DCOCLKDIV.

Multiple of Crystal Freq: Multiplies the clock source to calculate the value of DCOCLKDIV.

XT1

Crystal Speed: Specifies the speed of external crystal one.

Use External Clock: Specifies an external clock.

Turn Off in LPM4: Turns off crystal one in low-power mode four.

Use On-Chip Capacitance: Makes on-chip capacitance externally available. Select the capacitance from the drop-down box.

XT2

Crystal Speed: Specifies the speed of the second external crystal. If this value is zero, Embed assumes there is not a second external oscillator.

Turn Off in LPM4: Turns off crystal two in low power mode four.

Using STM32 Config

The STM32 Config lets you configure settings for the STM32. After you enter the settings, an STM32 Config block is inserted into the diagram.

CLK Sources

HSE: Chooses the speed of the external clock/crystal. Select the HSE mode (clock, crystal, or unused) in the dropdown to the right.

HSI: Indicates the speed of the HSI clock.

LSE: Chooses the speed of the external clock/crystal. Select the LSE mode (clock, crystal, or unused) in the dropdown to the right.

LSI: Indicates the speed of the LSI clock.

MSI: Indicates the speed of the MSI clock.

Control Clk

Count Mode: Displays the count mode. This is a read-only parameter.

Src: Displays the periodic sampling interrupt source for the main control loop in Embed. This is a read-only parameter.

Core: Selects the core device. Only available on STM32 dual core MCUs (H723VE – H735ZG targets).

CPU: For some targets, there is a family of CPU types. It is important to select the exact CPU type for the specified target.

CPU Speed: Indicates the speed and the maximum rated speed of the CPU. This parameter is affected by the **Voltage Regulation** parameter. STM32 H723VE – H735ZG targets are dual core. There is a **CPU2 Speed** parameter for the second core.

CPU and Peripheral Bus CLKs

HCLK=SYSCLK/: Selects the divider for the HCLK.

HSE_PREDIV: Divides the HSE source. It can be divided by 2 – 16. This parameter is available for only F3 series targets that do not end in D or E.

PCLK1=HCLK/: Selects the divider for the PCLK1. Used by the APB1 peripherals.

PCLK2=HCLK/: Selects the divider for the PCLK2. Used by the APB2 peripherals.

PLLM PREDIV / and ***PLLN**: PLLM PREDIV divides the PLL source. It can be divided by 1 – 16. ***PLLN** multiplies the results. If **PLLN** is highlighted in red, the value is either too high or too low. This parameter is available for all targets except F3 series that do not end in D or E.

PLL2DIVM and ***PLLN2**: PLL2DIVM divides the PLL source. It can be divided by 1 – 16. ***PLLN2** multiplies the results. If **PLLN2** is highlighted in red, the value is either too high or too low. This parameter is available for all targets except F3 series that do not end in D or E.

PLL3DIVM and ***PLLN3**: PLL3DIVM divides the PLL source. It can be divided by 1 – 16. ***PLLN3** multiplies the results. If **PLLN3** is highlighted in red, the value is either too high or too low. This parameter is available for all targets except WB series and F3 series that do not end in D or E.

PLLDIVP, PLLDIVQ, PLLDIVR: Specifies the divider values for PLL outputs P, Q, and R. Only available on STM32 F4x, G0x, G4x, and L4x targets.

PLL2DIVP, PLL2DIVQ, PLL2DIVR: Specifies the divider values for PLL2 outputs P, Q, and R. Only available on STM32 F4x, G0x, G4x, and L4x targets.

PLL3DIVP, PLL3DIVQ, PLL3DIVR: Specifies the divider values for PLL3 outputs P, Q, and R. Only available on STM32 F4x, G0x, G4x, and L4x targets.

PLLSRC: Selects the clock source used to drive the PLL input. You can also choose the PLL multiplier and divisor.

SYSCLK: Chooses the input for the system clock.

JTAG Connect: Indicates the JTAG driver to be used for HotLink communication.

Power Supply Config: Configures the power supply. Only available on STM32 F103x and H7x targets.

Toolchain: Selects the compiler to use.

Voltage Regulation: Controls the internal voltage of the chip. The lower the level, the less power the chip uses. When you change the **Voltage Regulation**, the **CPU Max Rated Speed** is changed accordingly.

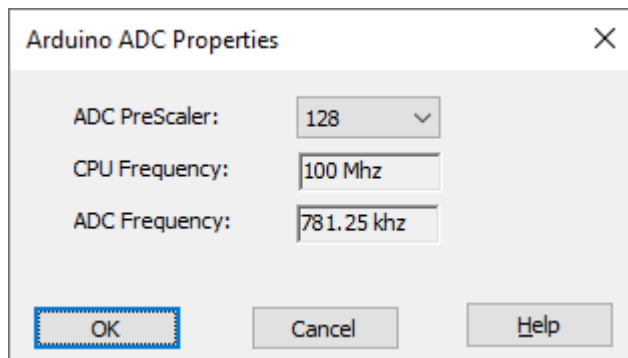
Using the Peripheral Config blocks

Using ADC Config

Target Category: Arduino, C2407, Delfino, F280x, F281X, MSP430, Piccolo, STM32

The ADC Config lets you control the ADC trigger for the supported boards. The ADC Config is essential for motor control algorithms.

ADC Config for Arduino

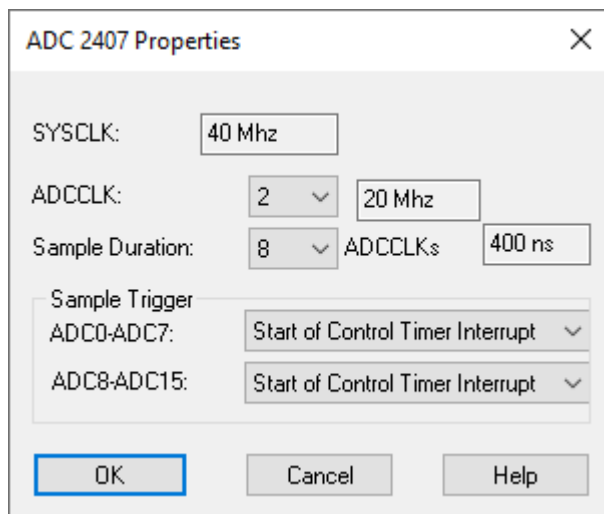


ADC Frequency: Indicates the ADC clock value based on the ADC prescaler.

ADC Prescaler: Prescales down the ADC clock speed by a specified factor. Prescale factors of 2, 4, 8, 16, 32, 64, and 128 are provided. If, for example, you are running your Arduino board at 16MHz with a prescale factor of 2, the ADC clock is set to 8MHz.

CPU Frequency: Indicates the core clock speed of the target device.

ADC Config for C2407



ADCCLK: Specifies the analog-to-digital converter clock.

Sample Duration: Chooses the number of ADC clock ticks for a given ADC sample. High impedance inputs require longer samples. A recommended duration is 20 nsec.

Sample Trigger: Controls when to start sampling channels.

SYSCLK: Indicates the speed of the CPU clock.

ADC Config for F281X

To configure the ADC trigger setup, you must first insert an F281X Config block in your diagram and set the CPU to your device. Only then can you access the ADC Config Properties dialog box.

The screenshot shows the 'ADC F281x Properties' dialog box with the following settings:

- SYSCLK:** 150 Mhz
- HCLK:** SYSCLK / 1 = 150 Mhz
- ADCCLK:** HCLK / 6 = 25 Mhz
- Sample Duration:** 4 ADCCLKs = 160 ns
- Use full scale value = 1 (faster code gen)
- Use Continuous Conversion
- Sample Trigger:**
 - ADC0-ADC7: Start of Control Timer Interrupt
 - ADC8-ADC15: Start of Control Timer Interrupt
 - Sample sequence 1 and 2 simultaneously
 - Cascade sequence 1 and 2
- Channel Sample Order:** 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 (with a 'New...' button)

ADCCLK: Specifies the analog-to-digital converter clock.

Channel Sample Order: Specifies the order in which you sample the channels. Click **New** to specify a new sample order.

HCLK: Specifies the high-speed clock.

Sample Duration: Chooses the number of ADC clock ticks for a given ADC sample. High impedance inputs require longer samples. A recommended duration is 20 nsec.

Sample Trigger

Controls when to start sampling channels.

Cascade Sequence 1 and 2: Scans sequence 1 followed by sequence 2.

Sample Sequence 1 and 2 at the same time: Allows the simultaneous sampling of the two sequences.

SYSCLK: Indicates the speed of the CPU clock.

Use Continuous Conversion: Enables the continuous conversion option in the hardware.

Use Full Scale Value = 1: Provides direct read of the ADC result register with no scaling. When activated, the value will range between zero and 0.00007. When de-activated, Embed adds code to scale the result from zero to three.

ADC Config for Delfino, F280X, and Piccolo

To configure the ADC, it's a good idea to first insert an [F28x Config](#) block in your diagram and set the CPU to your device.

Oversampling the ADC results in a reduction of noise. For every four oversamples, you'll see one additional bit of precision. Examples of how to oversample an ADC channel, see [Examples > Embedded > Piccolo > ADC](#).

Note: The ADC Config dialog box is dependent on the target you select. If you selected an older device, the [dialog box parameters are different](#) from the ones shown below.

ADC F28035 Properties

SYSCLK: Single Ended Interrupt on Conversion Start

ADCCLK: SYSCLK/

Reference:

ADCRESULT	Src	Trigger	Sample Clks	Dual Sample
ADCRESULT0:	ADCINA0	ePWM1-SOCA	7	<input type="checkbox"/>
ADCRESULT1:	ADCINA0	ePWM1-SOCA	12	<input type="checkbox"/>
ADCRESULT2:	ADCINB0	ePWM1-SOCA	12	<input type="checkbox"/>
ADCRESULT3:	ADCINA1	ePWM1-SOCA	12	<input type="checkbox"/>
ADCRESULT4:	ADCINA7	Timer 2	7	<input type="checkbox"/>
ADCRESULT5:	ADCINA5	Software	7	<input type="checkbox"/>
ADCRESULT6:	ADCINA6	Software	7	<input type="checkbox"/>
ADCRESULT7:	ADCINA7	Software	7	<input type="checkbox"/>
ADCRESULT8:	ADCINB0	Software	12	<input type="checkbox"/>
ADCRESULT9:	ADCINB1	ePWM1-SYNC	12	<input type="checkbox"/>
ADCRESULT10:	ADCINB2	ePWM1-SOCA	12	<input type="checkbox"/>
ADCRESULT11:	ADCINB3	Software	7	<input type="checkbox"/>
ADCRESULT12:	ADCINB4	Software	7	<input type="checkbox"/>
ADCRESULT13:	ADCINB5	Software	7	<input type="checkbox"/>
ADCRESULT14:	ADCINB6	Software	7	<input type="checkbox"/>
ADCRESULT15:	ADCINB7	Software	7	<input type="checkbox"/>

12-bit/16-bit: Specifies either a 12-bit or 16-bit analog measurement. This parameter is available on newer Piccolo chips.

ADCCLK: Specifies the analog-to-digital converter clock.

ADC Unit: Specifies the ADC unit to be configured. This parameter is available on newer Piccolo chips.

Dual Sample: If a dual sampling is selected, two consecutive SOCs (SOC_x and SOC_(x+1) or ADCRESULT_x and ADCRESULT_(x+1)) are used. The even numbered ADCRESULT contains the ADCIN_{Ay} value while the next (odd numbered) ADCRESULT contains the corresponding ADCIN_{By} value. Only these ADC inputs can be dual sampled. On clicking the dual sample, the even numbered ADCRESULT Src field must contain only the ADCINA0 to ADCINA7. Once the ADCIN_{Ay} value is selected, the next odd numbered ADCRESULT Src field must contain the ADCIN_{By} ONLY with the selection fixed. This parameter is available on Piccolo, F2802x, F2803x, F2805x, F2806x, and Concerto.

Interrupt on Conversion End: Generates interrupt on conversion end.

Interrupt on Conversion Start: Generates interrupt on conversion start.

Reference: Specifies the voltage reference. This parameter is available on Piccolo, F2802x, F2803x, F2805x, F2806x, and Concerto chips.

Setup PGA Gains: Invokes a dialog box to set the gain mode and output filter resistance for all PGAs.

Single Ended/Differential: When you choose **Single Ended**, Embed compares a single channel relative to analog ground. When you choose **Differential**, Embed measures the voltage difference between pairs of channels. This parameter is available on newer Piccolo chips.

SYSCLK: Indicates the speed of the CPU clock.

Trigger Setup

Dual Sample: Lets you sample pairs of channels simultaneously. This parameter is available for targets that do not have multiple ADC units.

Sample Clks: Specifies the sample and hold duration in units of system clocks.

Src: Specifies the analog pin on which to measure the voltage.

Trigger: Specifies the signal that will trigger a conversion.

ADC Config for F280X – early versions

To configure the ADC, it's a good idea to first insert an [F28x Config](#) block in your diagram and set the CPU to your device. The ADC Config Properties dialog box for the device.

Note: The ADC Config dialog box is dependent on the target you select. If you selected a newer device, the [dialog box parameters are different](#) from the ones shown below.

ADC F280x Properties

SYSCLK: 100 Mhz

HCLK: SYSCLK/ 1 100 Mhz

ADCCLK: HCLK/ 1 100 Mhz

Sample Duration: 1 ADCCLKs 10 ns

Use full scale value = 1 (faster code gen)

Use Continuous Conversion

Sample Trigger

ADC0-ADC7: Start of Control Timer Interrupt

ADC8-ADC15: Start of Control Timer Interrupt

Sample sequence 1 and 2 simultaneously

Cascade sequence 1 and 2

Channel Sample Order:
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 New...

OK Cancel Help

ADCCLK: Specifies the analog-to-digital converter clock.

Sample Duration: Chooses the number of ADC clock ticks for a given ADC sample. High impedance inputs require longer samples. A recommended duration is 20 nsec.

HCLK: Specifies the high-speed clock.

Sample Duration: Chooses the number of ADC clock ticks for a given ADC sample. High impedance inputs require longer samples. A recommended duration is 20 nsec.

Sample Trigger

Controls when to start sampling channels.

Cascade Sequence 1 and 2: Scans sequence 1 followed by sequence 2.

Sample Sequence 1 and 2 at the same time: Allows the simultaneous sampling of the two sequences.

SYSCCLK: Indicates the speed of the CPU clock

Use Continuous Conversion: Enables the continuous conversion option in the hardware.

Use Full Scale Value = 1: Provides direct read of the ADC result register with no scaling. When activated, the value will range between zero and 0.00007. When de-activated, Embed adds code to scale the result from zero to three.

ADC Config for STM32

Each ADC unit on the STM32 supports two independent sequencers that select input channels to be sent to the single AtoD converter for conversion. The Regular Sequencer supports up to 16 ordered elements in the sequence queue. Each sequence element can take its input from any of the 18 possible input channels available to the chip. After conversion, the results are sent to a single data register. To avoid overriding this register, DMA is used to automatically send the register to a memory vector called `_Adc<unit-number>Result[n]`. The memory location is displayed in the dialog box under Regular Sequencer Result.

The Injected Sequencer is similar to the Regular Sequencer, but it has only four ordered places in its sequence queue; however, it has four dedicated registers: one for each of the four input channels after they have been converted (`JDRn`).

The Injected Sequencer has higher priority over the Regular Sequencer and can “inject” its sequencer requests into an ongoing Regular Sequence if need be.

ADC STM32 Properties

ADC Unit: ADC1 / Resolution: 12Bits / Alignment: Right / ADCCLK=: HSI14 / 2 / 7 MHz / Sequencing: SINGLE_CONV

Regular Sequencer

Trigger: TIM1_TRGO / On Edge: None

Result	Input Source
_Adc1Result[0]:	ADC1_IN1 [PA0]
_Adc1Result[1]:	ADC1_IN1 [PA0]
_Adc1Result[2]:	ADC1_IN1 [PA0]
_Adc1Result[3]:	ADC1_IN1 [PA0]
_Adc1Result[4]:	ADC1_IN1 [PA0]
_Adc1Result[5]:	ADC1_IN1 [PA0]
_Adc1Result[6]:	ADC1_IN1 [PA0]
_Adc1Result[7]:	ADC1_IN1 [PA0]
_Adc1Result[8]:	ADC1_IN1 [PA0]
_Adc1Result[9]:	ADC1_IN1 [PA0]
_Adc1Result[10]:	ADC1_IN1 [PA0]
_Adc1Result[11]:	ADC1_IN1 [PA0]
_Adc1Result[12]:	ADC1_IN1 [PA0]
_Adc1Result[13]:	ADC1_IN1 [PA0]
_Adc1Result[14]:	ADC1_IN1 [PA0]
_Adc1Result[15]:	ADC1_IN1 [PA0]

Injected Sequencer

Trigger: / On Edge:

Result	Input Source
JDR1:	
JDR2:	
JDR3:	
JDR4:	

Input Channel Configure: SE/Differential

IN	Sample Time	SE/Differential
IN1:	3 ADCCLKs	Single
IN2:	3 ADCCLKs	Single
IN3:	3 ADCCLKs	Single
IN4:	3 ADCCLKs	Single
IN5:	3 ADCCLKs	Single
IN6:	3 ADCCLKs	Single
IN7:	3 ADCCLKs	Single
IN8:	3 ADCCLKs	Single
IN9:	3 ADCCLKs	Single
IN10:	3 ADCCLKs	Single
IN11:	3 ADCCLKs	Single
IN12:	3 ADCCLKs	Single
IN13:	3 ADCCLKs	Single
IN14:	3 ADCCLKs	Single
IN15:	3 ADCCLKs	
IN16:	3 ADCCLKs	
IN17:		
IN18:		

Buttons: OK, Cancel, Help

ADCCLK= : Selects the source clock and the divider.

ADC Unit: Specifies the ADC unit to be configured.

Alignment: Aligns least significant digit (Right) or most significant digit (Left).

Injected Sequencer

Input Source: Selects the input channel for a given sequencer queue element.

On: Selects the trigger edge event that causes the sequencer to begin converting.

Trigger: Selects the trigger source.

Input Channel Configure

Sample Time: Indicates the sampling time for each input channel. Longer sampling times generally provide more accurate results but take longer to complete.

SE/Differential: Selects if the given channel is single-ended or a differential pair. When you select Differential, the first input is positive and the second input is negative.

Regular Sequencer

Input Source: Selects the input channel for a given sequencer queue element.

On: Selects the trigger edge event that causes the sequencer to begin converting.

Trigger: Selects the trigger source.

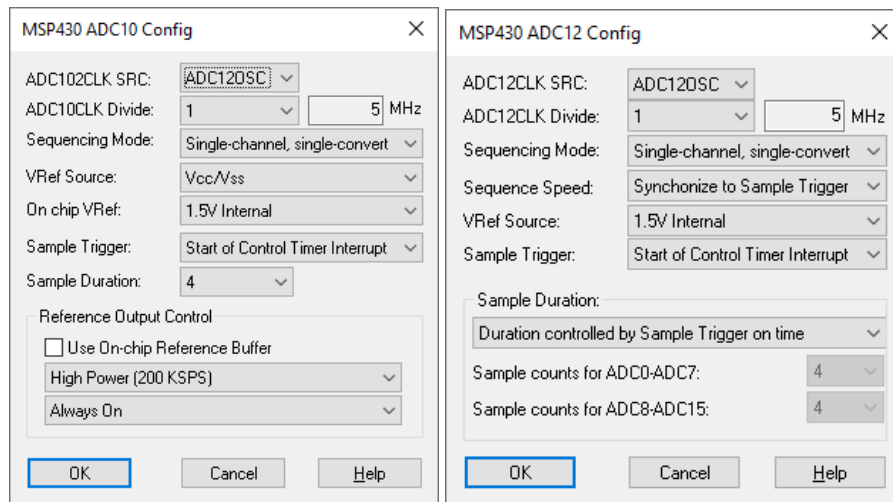
Resolution: Specifies 6-, 8-, 10-, or 12-bits of the result data.

Sequencing: Indicates triggered conversions of a single sequence of channels. This is a read-only parameter.

SYSCLK: Indicates the speed of the CPU clock.

ADC10/12 Config for MSP430

The ADC10/12 Config lets you configure MSP430 ADC clock settings. Note that the dialog box that appears when you click on the ADC10/12 Config command is dependent on the CPU subtype that you specify in the dialog box for the [MSP430 Config](#) command. The following are the two possible dialog boxes that correspond to the ADC10/12 Config command:



ADCxx2CLK SRC: Chooses the source for the ADC10 or ADC12 clock.

ADCxxCLK Divide: Divides the specified ADC10 or ADC12 clock into a slower rate of your choosing.

On chip Vref: Selects the voltage reference.

Reference Output Control (for ADC10 block)

Use On-Chip Reference Buffer: Lets you use the internal reference source on the Vref output pin.

Sample Duration (for ADC10 clock): Indicates the number of ADC10 clocks that comprise the ADC sampling period. The longer you take to sample, the more accurate your measurement.

Sample Duration (for ADC12 clock)

Duration controlled by ADC12CLK count set below: This is the preferred setting. You set the sample control counts in the Sample Control Counts text boxes. The longer the count, the more you sample.

Duration controlled by Sample Trigger on time: This is the default setting. If you choose this setting, you will need an external pin.

Sample Control Counts for ADC0-ADC7: Specifies the sample control counts for ADC0 through ADC7.

Sample Control Counts for ADC8-ADC15: Specifies the sample control counts for ADC8 through ADC15.

Sample Trigger: Controls when to start sampling channels. You can synchronize with the start of the control timer interrupt or with Timer A.

Sequencing Mode

Chooses the operational mode of the ADC engine.

Sequence-of-channels: Samples a number of channels, one after another.

Single-channel, single-convert: Samples one channel once.

Repeat-sequence-of-channels: Samples a number of channels, one after another, nonstop.

Repeat-single-channel: Samples one channel nonstop.

Sequence Speed: If you are in channel sequence mode, this parameter lets you insert delays in the sequence. This parameter is available only on the ADC12 clock.

Vref Source: Lets you select the reference voltage for the unit. Because the internal voltage reference can be inaccurate, this parameter lets choose a high-precision, external signal.

Using CAN Config

Target Category: C2407, Delfino, F280x, F281X, Piccolo, STM32

The CAN Config lets you configure the controller area network.

Baud Rate Prescale: Indicates the divisor applied to system clock to obtain the CAN clock. For more information, see <http://www.bittiming.can-wiki.info/>.

Bus Rate: Indicates the base bit rate of CAN bus messages.

Byte Order: See Texas Instruments or STM32 documentation.

RX Mux Pin: Indicates the RX pin choices for CAN messages. This parameter is not available for C2407 targets.

Sample Point: See Texas Instruments or STM32 documentation.

Synchronization: See Texas Instruments or STM32 documentation.

Sync Jump Width: Indicates the amount the CAN unit can adjust timing to sync with the bus. (1 for CANopen and DeviceNet.)

TSeg1 and TSeg2: The ratio defines the sampling point for the bit: $(Tseg1+1)/(Tseg1+Tseg2+1)$. For CANopen and DeviceNet, ratio=0.875; for ARINC 825, ratio=0.75. For more information, see <http://www.bittiming.can-wiki.info/>.

TX Mux Pin: Indicates the RX pin choices for CAN messages. This parameter is not available for 2407 targets.

Unit: Specifies the unit.

Using DMA Config

Target Category: Delfino, F280x, F281X, Piccolo, STM32

The DMA Config lets you configure a Direct Memory Access (DMA) channel. The DMA peripheral will automatically transfer memory items from a source set of memory locations to a destination set of memory addresses, on the occurrence of an interrupt on the selected trigger source. The memory locations can include peripheral device addresses allowing automatic transfer of peripheral data to memory or memory to peripheral devices. For example, an ADC channel can be automatically sampled and read to a memory buffer, or a memory buffer with a preconfigured waveform can repeatedly be written out at a specific rate to a PWM compare register.

The screenshot shows the 'DMA Config' dialog box with the following settings:

- Channel: 1
- Data Size: 16-bit
- Continuous Mode:
- Trigger Src: None
- One Shot:
- Init Src Address: [Empty]
- Init Dst Address: [Empty]
- Burst Size: 1
- Burst Src Step: 1
- Burst Dst Step: 0
- Transfer Size: 0
- Transfer Src Step: 0
- Transfer Dst Step: 0
- Wrap Src Size: 0
- Wrap Src Step: 0
- Wrap Dst Step: 0

Burst Size: Specifies the count of data items to be transferred.

Burst Dst Step: Specifies the step size between memory locations in the destination.

Burst Src Step: Specifies the step size between memory locations in the source.

Channel: Specifies the DMA channel number.

Continuous Mode: If you do not activate **Continuous Mode**, DMA interrupt generation stops after one interrupt.

Data Size: Indicates the size of a single memory element to be transferred.

Init Dst Address: Specifies the initial destination memory location. This uses standard C address syntax. You can also use Texas Instruments names for hardware devices, for example, `&CMPA1` specifies the address of the first PWM compare register.

Init Src Address: Specifies the initial memory location. This uses standard C address syntax. You can also use Texas Instruments names for hardware devices, for example, `&ADCRESULT0` specifies the address of the first ADC result register.

One Shot: If **One Shot** is activated, the DMA unit immediately begins the next burst after servicing the previous burst. Otherwise, the DMA unit waits for a peripheral interrupt after the first burst.

Transfer Dst Step: Specifies the step size between memory locations in the destination.

Transfer Size: Specifies the count of bursts to be transferred.

Transfer Src Step: Specifies the step size between memory locations in the source.

Trigger Src: Specifies the interrupt signal that will cause a DMA transfer to occur.

Wrap Dst Size: Specifies the wrap destination initial count. If the wrap count decrements to zero, the current destination address is incremented by the wrap step, and the wrap counter is set to the wrap size.

Wrap Dst Step: Specifies the wrap destination initial count. If the destination wrap count decrements to zero, the current destination address is incremented by the destination wrap step, and the destination wrap counter is set to the destination wrap size.

Wrap Src Size: Specifies the wrap source initial count. If the source wrap count decrements to zero, the current source address is incremented by the source wrap step, and the source wrap counter is set to the source wrap size.

Wrap Src Step: Specifies the wrap step. If the source wrap count decrements to zero, the current source address is incremented by the wrap step, and the wrap counter is set to the wrap size.

Using ESP8266WiFi Config

The ESP8266WiFi Config lets you configure the ESP8266 module on your Arduino device.

ESP WiFi Shield Configure

SSID: Specifies the available WiFi name.

Password: Specifies the WiFi security password.

ESP RX Pin -> Arduino: Specifies the pin that connects to the Arduino digital pin 3.

ESP TX Pin -> Arduino: Specifies the pin that connects to the Arduino digital pin 2.

Enable Serial Debug: When activated, displays the debug messages on the serial terminal (serial config 9600, 8, 1, 0, 0).

MQTT Client Configure

Edit MQTT Default Config: When activated, you can edit the default MQTT Client Configuration settings.

MQTT Max Packet Size: Specifies the maximum packet size. The default is 256.

MQTT Keep Alive Seconds: Specifies the keep alive period. That is, how often the client sends a PING message to the broker. The default time interval is 15 sec.

MQTT Socket Time Out Seconds: Lets you set the timeout manually. The default timeout is 15 sec.

Using GPIO Qualification

The GPIO Qualification block lets you choose the qualification time interval for GPIO pins on the Delfino, F280X, and Piccolo. Selection of qualification can help filter noise and remove glitching from external signals.

The screenshot shows a dialog box titled "F280049 GPIO Qualification Properties". It contains several configuration options:

- GPIO Bank:** A dropdown menu with the value "1" selected.
- Qualification Period:** A dropdown menu with the value "SYSCLKOUT/1" selected.
- GPIO0:** A dropdown menu with the value "Sync to SYSCLKOUT" selected.
- GPIO1:** A dropdown menu with the value "Sync to SYSCLKOUT" selected.
- GPIO2:** A dropdown menu with the value "Sync to SYSCLKOUT" selected.
- GPIO3:** A dropdown menu with the value "Sync to SYSCLKOUT" selected.
- GPIO4:** A dropdown menu with the value "Sync to SYSCLKOUT" selected.
- GPIO5:** A dropdown menu with the value "Sync to SYSCLKOUT" selected.
- GPIO6:** A dropdown menu with the value "Sync to SYSCLKOUT" selected.
- GPIO7:** A dropdown menu with the value "Sync to SYSCLKOUT" selected.

At the bottom of the dialog box, there are three buttons: "OK", "Cancel", and "Help". The "OK" button is highlighted with a blue border.

GPIO Bank: Selects the bank of 8 GPIO pins to configure.

GPIO0-7: Selects synchronization or qualification duration for a given GPIO pin.

Qualification Period: Selects the duration of a Qualification sample.

Using I2C Config

I2C Config for Arduino, Cortex M3, Delfino, F280X, Linux Raspberry Pi, Piccolo, and STM32

The I2C Config lets you configure the I2C unit.

Address Mode: In 7-bit address mode (normal address mode), indicates the I2C module transmits 7-bit slave addresses.

Bit Rate: Indicates the bit rate.

Clock Source: Chooses the hardware source for the system clock.

Data Bits: Indicates the number of bits (1 to 8) in the next data byte that is to be received or transmitted by the I2C module.

High Time: Indicates the ICCH- Clock high-time divide-down value. To produce the high-time duration of the master clock, the period of the module clock is multiplied by $(ICCH + d)$, where d is 5, 6, or 7.

These bits must be set to a non-zero value for proper I2C clock operation.

Low Time: Indicates the ICCL- Clock low-time divide-down value. To produce the low-time duration of the master clock, the period of the module clock is multiplied by $(ICCL + d)$. d is 5, 6, or 7.

These bits must be set to a non-zero value for proper I2C clock operation.

Own Address: Indicates the address of this unit on the I2C bus. This is important in slave mode: messages that do not match Own Address will be discarded. This parameter is not available for ARM Linux targets.

Port: Specifies the port. For ARM Linux targets, 1 is the default device available for use; 0 is reserved for VideoCore and HAT EEPROM; and 2 is available for use, but it is dedicated to HDMI interface and should not be used. For Arduino pin mapping, click [here](#).

Prescale: Specifies the value to divide the low speed clock to determine the I2C clock. This parameter is not available for ARM Linux targets.

Rx Queue Length: Specifies the length of the receive queue.

SCL Pin: Specifies the pin on the device for the serial clock line.

SDA Pin: Specifies the pin on the device for the serial data line.

Tx Queue Length: Specifies the length of the transmit queue.

Use Freeform Mode: Indicates that the transfer has no address field. This parameter is not available for Arduino, ARM Linux, and Cortex M3 targets.

I2C Config for MSP430

The I2C Config lets you configure the I2C unit.

Address Len: Specifies the length of the bus address. It can be either 7- or 10-bit.

Baud Rate: Indicates the base clock rate for master mode.

Clk Src

Selects the clock for the I2C unit. You have these choices:

ACLK: Auxiliary clock

Port: Selects the Comm port

SMCLK: Submain clock

UCLK: External clock

Initial Mode: Specifies the mode at boot time.

Module: Choose among USART, USCIB (new school), and USI (minimalist).

Multi-Master Bus: Set to TRUE if addressing is used.

Mux Pin: Selects which pin a given function is on.

Certain MSP430 devices have different functions for the same physical pin on the chip. This is because pins are expensive. Sharing time or space is called multiplexing, or muxing, for short. Because multiple functions compete for a given pin, you must choose what function a pin has. For flexibility, in some cases Texas Instruments has provided multiple possible pins for a given function. For instance, the CANTXB function can be on pin 8, 12, or 16. Pin 8 is shared with ePWM5A and ADCSOCA0, pin 12 is shared with TZ1 and SPISIMOB, pin16 is shared with SPISIMOA and TZ5. So, if you want ePWM5A on a pin, then you cannot use pin 8 for CANTXB, but rather you must use pin 12 or 16 instead.

Own Address: Indicates the address of this unit on the I2C bus. This is important in slave mode: messages that do not match Own Address will be discarded.

Port: Specifies the port.

Respond to General Call: Responds to address 0.

Rx Queue Length: Specifies the length of the receive queue.

SCL Pin/SDA Pin: Specifies mux pins for the I2C clock and data lines.

Tx Queue Length: Specifies the length of the transmit queue.

UCLK: Specifies the external hardware clock rate.

Using SD16 Config

The SD16 Config lets you choose the hardware settings for the [SD16](#) and [SD16A](#) blocks for the MSP430.

Clock Source: Selects the ADC clock.

Clock Scaling: Selects the divider for the ADC clock.

Continuous Conversion: Keeps the SD16 unit running continuously.

Enable 1.2V Internal Vref: Allows you to enable the internal voltage reference.

Enable 1.2V Internal Vref 1mA Buffer: Allows you to increase the power of the enabled internal reference voltage.

Lower Power Mode: Runs in low power mode.

Using Serial UART Config

The Serial UART Config lets you choose the hardware settings for the [serial UART Read](#) and [serial UART Write](#) blocks.

The screenshot shows a 'Serial Port Config' dialog box with the following settings:

- Port: 1
- Parity: None
- Baud Rate: 300
- Data Bits: 9
- Stop Bits: 1
- Clock Sel: SYSCLK
- Tx Queue Length: 64
- Rx Queue Length: 64
- Mux Pin Assignment:
 - SCIRX: PA10
 - SCITX: PA9

Buttons at the bottom: OK, Cancel, Help.

Baud Rate: Sets the baud rate.

Clock Sel: Chooses the hardware source for the system clock.

Data Bits: Sets the number of data bits.

Mux Pin Assignment: Indicates the GPIO pin on which you want the peripheral signal mapped. This parameter is not available for MSP430 targets.

Parity: Sets the parity.

Port: Specifies the port.

Rx Queue Length: Sets the length of the receive queue. The minimum length is 1 byte; the maximum length is 255 bytes.

Stop Bits: Sets the stop bits.

Tx Queue Length: Sets the transmit queue length. The minimum length is 1 byte; the maximum length is 255 bytes.

USCIA: Lets you use the USCIA peripheral module. If you have a part with this module and you want to use it, you must activate this parameter. This parameter is available only for MSP430 targets.

Using SPI Config

The SPI Config lets you choose the hardware settings for the [SPI Read](#) and [SPI Write](#) blocks for C2407, Cortex M3, Delfino, F280x, F281X, MSP430, Piccolo, and STM32 targets.

Bit Rate: Sets the bit rate.

Clk Src

Selects the clock for the serial port unit. This parameter is not available for STM32 targets.

ACLK: Auxiliary clock.

Port: Comm port.

SMCLK: Submain clock.

UCLK: External clock.

CLK Polarity: Clocks the polarity. You have two choices: **Send on rise/Latch on fall** and **Send on fall/Latch on rise**.

Interrupt on RX Fifo Queue Level: Specifies the FIFO level at which an interrupt is generated. A value of n or n-1, where n is the FIFO length, will minimize interrupt occurrences.

Interrupt on TX Fifo Queue Level: Specifies the FIFO level at which an interrupt is generated. Generally, a value of 1 will minimize interrupt occurrence. Note that interrupts are enabled only when the FIFO is filled and additional items have been written to the soft queue.

Length: Specifies the length of the software queue. The queue will allow buffering up to the specified number of elements.

Mux Pin Assignment: Sets the external pins used to carry the SPI signals.

Network Mode: Sets the mode to master or slave.

Prescale: Chooses the prescale factor. This parameter is available only for STM32 targets. It is settable only in master mode.

SPIDiv: Specifies the divider for the system clock to set the SPI clock rate. T

STE/NSS Mode: Enables the slave to transmit data.

Sync Data: Synchronizes the data on the clock edge or $\frac{1}{2}$ cycle before clock edge.

Transmitted Bits: Sets the number of bits transmitted per transaction (1-16).

Unit: Specifies the unit number.

Use FIFO: Selects use of the hardware FIFO (if present).

Use RX Soft Queue: Selects use of a software queue. This queue is interrupt driven and will be filled automatically by the Embed driver.

Use TX Soft Queue: Selects use of a software queue. This queue is interrupt driven and will be drained automatically by the Embed driver.

Using SPI Config for Arduino

The SPI Config for Arduino block lets you choose the hardware settings for the [SPI Read for Arduino](#) and [SPI Write for Arduino](#) blocks.

Bit Rate=SYSClk/SPIDiv: Displays the bit rate.

CLK Polarity: Clocks the polarity. You have two choices: **Send on rise/Latch on fall** and **Send on fall/Latch on rise**.

Mux Pin Assignment: Sets the external pins used to carry the SPI signals. Click [here](#) for Arduino pin mapping. The on-chip Slave Select (SS) pin is available in Slave network mode only.

Network Mode: Sets the mode to master or slave. In Master network mode, the SS pin is selected in the [SPI Write for Arduino](#) block.

SPIDiv: Sets the SPI bit rate. Select from the following:

SPIDiv	SPI Bit rate (Hz)	SYSClk (Hz)
2	8000000	16000000
4	4000000	16000000
8	2000000	16000000

16	1000000	16000000
32	500000	16000000
64	250000	16000000
128	125000	16000000

Sync Data: Synchronizes the data on the clock edge or $\frac{1}{2}$ cycle before clock edge.

Sync Data (CPHA)	CLK Polarity (CPOL)	SPI Mode
on clock edge (0)	Send on rise/Latch on fall (0)	MODE 0
$\frac{1}{2}$ cycle before clock edge (1)	Send on rise/Latch on fall (0)	MODE 1
on clock edge (0)	Send on fall/Latch on rise (1)	MODE 2
$\frac{1}{2}$ cycle before clock edge (1)	Send on fall/Latch on rise (1)	MODE 3

Unit: Specifies the unit number.

Using SPI Config for Linux

The SPI Config for Linux block lets you choose the hardware settings for the [SPI Read for ARM-Linux](#) and [SPI Write for ARM-Linux](#) blocks.

Bit Rate=Clk Src/SPIDiv: Displays the bit rate.

CLK Polarity: Clocks the polarity. You have two choices: **Send on rise/Latch on fall** and **Send on fall/Latch on rise**.

Clk Src: Selects the clock for the serial port unit. Raspberry Pi runs at Advanced Peripheral Bus (APB) clock speed, which is equivalent to core clock speed (250MHz). The supported clock speed ranges from 32kHz to 125MHz.

Mux Pin Assignment: Sets the external pins used to carry the SPI signals.

Network Mode: Indicates that the mode is set to Master. Slave mode does not work on Raspberry Pi.

SPIDiv: Specifies the divider for the system clock to set the SPI clock rate. The clock speed can be divided by any number from 1 to 3900 for the desired speed.

Sync Data: Synchronizes the data on the clock edge or ½ cycle before clock edge.

Sync Data (CPHA)	CLK Polarity (CPOL)	SPI Mode
on clock edge (0)	Send on rise/Latch on fall (0)	MODE 0

½ cycle before clock edge (1)	Send on rise/Latch on fall (0)	MODE 1
on clock edge (0)	Send on fall/Latch on rise (1)	MODE 2
½ cycle before clock edge (1)	Send on fall/Latch on rise (1)	MODE 3

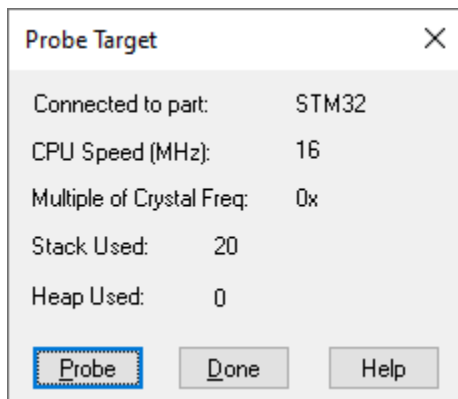
Transmitted Bits: Sets the number of bits transmitted per transaction.

Unit: Specifies the unit number. Raspberry Pi has one SPI interface, referred to as SPI0. It also has a mini SPI interface, referred to as SPI1.

Using the Target Interface commands

Using the Get Target Stack and Heap command

The Get Target Stack and Heap command displays the stack and heap used on the target.



CPU Speed: Indicates the speed of the CPU.

Heap Used: Indicates the heap used on the target.

Multiple of Crystal Freq: Indicates the multiple of crystal frequency.

Stack Used: Indicates the stack used on the target.

Using the Reset Target command

The Reset Target command resets internal registers to their initial values, as defined in the Texas Instruments and STMicroelectronics documentation.

Using the TI DMC Block Set

A 16-bit and 32-bit Texas Instruments DMC block set is included with Embed. The 16-bit block set applies to the C2407 series; the 32-bit block set applies to the 28xx series.

Additional Information: Texas Instruments [C2407](#) document.

Similarities and differences between 16-bit and 32-bit TI DMC block

Most of the 16-bit Texas Instruments DMC blocks use 1.16 scaling, which means you use inputs between -1.0 and 0.9997. Most of the 32-bit Texas Instruments DMC blocks use 8.32 scaling, which means that you use inputs between -256 and 255.99999.

ACI Motor

Embedded Category: C2407, F280x

Block Category: TI 16-bit DMC, TI 32-bit DMC

Operating Mode Availability

- Simulation mode: YES
- C code generation mode: NO

The ACI Motor block implements a discrete equivalent of a 3-phase induction motor using trapezoidal approximation with predictor-corrector. The induction model is normalized by the adjustable base quantities of voltage, current, torque, frequency, and flux linkage. The induction motor is modeled in the stationary reference frame. The outputs of this module are the stator currents, rotor flux linkages, electromagnetic torque, electrically angular velocity, and actual rotor speed in rpm. These outputs are in per-unit except the actual rotor speed.

ACI Motor Properties

Rotor Resistance - R_r (Ohm):	<input type="text" value="0"/>
Stator Resistance - R_s (Ohm):	<input type="text" value="0"/>
Stator Leakage Inductance - L_s (H):	<input type="text" value="0"/>
Rotor Leakage Inductance - L_r (H):	<input type="text" value="0"/>
Magnetic Inductance - L_m (H):	<input type="text" value="0"/>
Base Current - I_B (A):	<input type="text" value="0"/>
Base Phase Voltage - V_b (V):	<input type="text" value="0"/>
Base Torque - T_b (Nm):	<input type="text" value="0"/>
Base Flux Linkage - L_b (Vsec/rad):	<input type="text" value="0"/>
Base electric rotation speed - ω_b (r/s)	<input type="text" value="0"/>
Number of Poles - p	<input type="text" value="0"/>
Damping factor - B (Nm sec/rad)	<input type="text" value="0"/>
Moment of Inertia - J ($\text{kg}\cdot\text{m}^2$)	<input type="text" value="0"/>
Motor Controller	
Sample interval - T_s (sec)	<input type="text" value="0"/>

Base Current: Indicates the maximum current flow at the rated load.

Base Electric Rotation Speed: Indicates the maximum target rotation.

Base Flux Linkage: Indicates the base flux linkage.

Base Phase Voltage: Indicates the base phase voltage.

Base Torque: Indicates the base rated torque.

Damping Factor: Indicates the damping factor.

Magnetic Inductance: Indicates the magnetizing inductance of the load. Specify this value in henries.

Moment of Inertia: Indicates the inertia of the rotor.

Number of Poles: Indicates the number of poles in the motor.

Rotor Leakage Inductance: Indicates the rotor leakage inductance of the load. Specify this value in henries.

Rotor Resistance: Indicates the rotor resistance of the load. Specify this value in henries.

Sample Interval: Indicates the sampling interval of the motor controller.

Stator Leakage Inductance: Indicates the stator leakage.

Stator Resistance: Indicates the stator resistance.

ACI Flux Estimator

Embedded Category: C2407, F280x

Block Category: TI 16-bit DMC, TI 32-bit DMC

Operating Mode Availability

- Simulation mode: YES
- C code generation mode: YES

The ACI Flux Estimator block implements the flux estimator with the rotor flux angle for the 3-ph induction motor based upon the integral of back emf's (voltage model) approach. To reduce the errors due to pure integrator and stator resistance measurement, the compensated voltages produced by PI compensators are introduced. Therefore, this flux estimator can be operating over a wide range of speed, even at very low speed.

ACI Flux Estimator Properties

Rotor Resistance - Rr (Ohm):	<input type="text" value="2.011"/>	Proportional Gain:	<input type="text" value="0.125"/>
Stator Resistance - Rs (Ohm):	<input type="text" value="1.723"/>	(0 - .999)	
Stator Leakage Inductance - Ls (H):	<input type="text" value="0.166619"/>	Integral Gain:	<input type="text" value="0.00017"/>
Rotor Leakage Inductance - Lr (H):	<input type="text" value="0.168964"/>	(sampling period/reset time)	
Magnetic Inductance - Lm (H):	<input type="text" value="0.159232"/>	<input type="text" value="5.96e-08 - .003906"/>	
Base Current - IB (A):	<input type="text" value="5"/>	Sampling Frequency (Hz):	<input type="text" value="2000"/>
Base Phase Voltage - Vb (V):	<input type="text" value="184.752"/>		

NOTE: Greyed parameters are set in the ACI motor block

Base Current: Indicates the maximum current flow at the rated load.

Base Phase Voltage: Indicates the base phase voltage.

Integral Gain: Indicates the controller gain proportional to integral of (cmd-ref).

Rotor Resistance: Indicates the rotor resistance of the load. Specify this value in henries.

Magnetic Inductance: Indicates the magnetizing inductance of the load. Specify this value in henries.

Proportional Gain: Indicates the controller gain proportional to (cmd-ref)Stator Resistance: Indicates the stator resistance.

Rotor Leakage Inductance: Indicates the rotor leakage inductance of the load. Specify this value in henries.

Rotor Resistance: Indicates the electrical resistance of motor rotor.

Sampling Frequency: Indicates sampling rate of the control running on the target.

Stator Leakage Inductance: Indicates the stator leakage.

Stator Resistance: Indicates the electrical resistance of the stator coil.

ACI Speed Estimator

Embedded Category: C2407, F280x

Block Category: TI 16-bit DMC, TI 32-bit DMC

Operating Mode Availability

- Simulation mode: YES
- C code generation mode: YES

The ACI Speed Estimator block implements a speed estimator of the 3-phase induction motor based upon its mathematics model. The estimator's accuracy relies heavily on knowledge of critical motor parameters.

ACI Speed Estimator Properties	
Number of Poles:	4
Rotor Resistance (Ohm):	2.011
Rotor Leakage Inductance (H):	0.168964
Magnetic Inductance (H):	0.159232
Base Electrical Ang. Vel. (rad/s):	376.99111843077E
Sampling Frequency (Hz):	2000
Cut-off Frequency (Hz):	200
NOTE: Greyed parameters are set in the ACI motor block	
<input type="button" value="OK"/> <input type="button" value="Cancel"/> <input type="button" value="Help"/>	

Base Electrical Ang. Vel.: Indicates the rotational speed of the electrical field between poles. **Number of Poles:** Indicates the number of poles in the motor.

Cut-off Frequency: Indicates the cut-off frequency of the internal low-pass filter **Magnetic Inductance:** Indicates the magnetizing inductance of the load. Specify this value in henries.

Magnetic Inductance: Indicates the magnetizing inductance of the stator coil. Specify the value in henries.

Number of Poles: Indicates the number of poles in the motor.

Rotor Leakage Inductance: Indicates the rotor leakage inductance of the load. Specify this value in henries.

Rotor Resistance: Indicates the rotor resistance of the load. Specify this value in henries.

Sampling Frequency: Indicates sampling rate of the control running on the target.

Clarke Transform

Embedded Category: C2407, F280x

Block Category: TI 16-bit DMC, TI 32-bit DMC

$d = a$

$$q = \frac{2b + a}{\sqrt{3}}$$

Operating Mode Availability

- Simulation mode: YES
- C code generation mode: YES

The Clarke Transform block converts balanced three-phase quantities into balanced two-phase quadrature quantities.

Current Model

Embedded Category: C2407, F280x

Block Category: TI 16-bit DMC, TI 32-bit DMC

Operating Mode Availability

- Simulation mode: YES
- C code generation mode: YES

The ACI Current Model block infers rotor position from D and Q (park transformed) current measurements. This is useful for sensorless control of AC induction motors.

ACI Current Model Properties	
Pole Count:	2
Max Flux Speed (Rad/Sec):	100
Magnetizing Inductance (H):	0
Rotor Leakage Inductance (H):	5.5
Rotor Resistance (Ohms):	1.5
Sampling Frequency (Hz):	0
<input type="button" value="OK"/> <input type="button" value="Cancel"/> <input type="button" value="Help"/>	

Magnetizing Inductance (H): Indicates the magnetizing inductance of the load. Specify this value in henries.

Max Flux Speed (Rad/Sec): Indicates the maximum rotor flux speed. Specify this value in rad/sec.

Pole Count: Indicates the number of poles in the motor.

Rotor Leakage Inductance (H): Indicates the rotor leakage inductance of the load. Specify this value in henries.

Rotor Resistance (Ohms): Indicates the rotor resistance of the load. Specify this value in ohms.

Sampling Frequency (Hz): Indicates sampling rate of the control running on the target.

Inverse Clarke Transform

Embedded Category: C2407, F280x

Block Category: TI 16-bit DMC, TI 32-bit DMC

$$a = d$$

$$b = \frac{-d + q\sqrt{3}}{2}$$

$$c = \frac{-d - q\sqrt{3}}{2}$$

Operating Mode Availability

- Simulation mode: YES
- C code generation mode: YES

The Inverse Clarke Transform block converts balanced two-phase quadrature quantities into balanced three-phase quantities.

Inverse Park Transform

Embedded Category: C2407, F280x

Block Category: TI 16-bit DMC, TI 32-bit DMC

$$d = -Q \sin\theta + D \cos\theta$$

$$q = Q \cos\theta + D \sin\theta$$

Operating Mode Availability

- Simulation mode: YES
- C code generation mode: YES

The Inverse Park Transform block projects vectors in orthogonal rotating reference frame into two-phase orthogonal stationary frame.

Park Transform

Embedded Category: C2407, F280x

Block Category: TI 16-bit DMC, TI 32-bit DMC

$$D = d * \cos(\theta) + q * \sin(\theta)$$

$$Q = -d * \cos(\theta) + q * \sin(\theta)$$

Operating Mode Availability

- Simulation mode: YES
- C code generation mode: YES

The Park Transform block converts vectors in balanced two-phase orthogonal stationary system into orthogonal rotating reference frame.

Phase Voltage Calc

Embedded Category: C2407, F280x

Block Category: TI 16-bit DMC, TI 32-bit DMC

cmd = command input

ref = plant measurement input

$out = PID\ output$

Operating Mode Availability

- Simulation mode: YES
- C code generation mode: YES

The Phase Voltage Calc block takes the 3-phase Space Vector Generator as input and produces phase voltages in the time domain or DQ coordinates as output. This is useful for doing sensorless motor control.

An example of the use of the Phase Voltage Calc block can be found in the *PMSM32SIM.vsm* diagram under Diagrams > Examples > Digital Motor Control.

QEP Speed

Embedded Category: C2407, F280x

Block Category: TI 16-bit DMC, TI 32-bit DMC

Block Inputs

- **QE:** fractional value of position between zero and one. The current quadrature encoder count is multiplied by the inverse of the maximum counts per revolution resulting in a value between zero and one.
- **Dir:** 1 = forward direction, -1 = backward

Block Output

- **Wr:** fraction of maximum speed ("Base Electrical Frequency")

Operating Mode Availability

- Simulation mode: YES
- C code generation mode: YES

The QEP Speed block determines the rotational speed based on modulo-encoder tick counts.

Quadrature Encoder Speed Properties	
Base Electrical Freq. (Hz):	200
Control Sampling Rate (Hz):	20000
Low Pass Cutoff (Hz):	30
<input type="button" value="OK"/> <input type="button" value="Cancel"/> <input type="button" value="Help"/>	

Base Electrical Frequency: Indicates the maximum rotational speed of the electrical field between poles.

Control Sampling Rate: Indicates sampling rate of the control running on the target.

Low Pass Cutoff: A low-pass filter is applied to the successive differencing used to determine speed from successive QEP measurements. Speed changes above the cut-off frequency will be attenuated.

PID Regulator

Embedded Category: C2407, F280x

Block Category: TI 16-bit DMC, TI 32-bit DMC

cmd = command input

ref = plant measurement input

out = PID output

Operating Mode Availability

- Simulation mode: YES
- C code generation mode: YES

The PID Regulator block implements a fixed-point PID controller.

Parameter	Value	Range
Kp:	0.5	-128 .. 127.999
Ki:	0.001	-128 .. 127.999
Kd:	0.0001	-128 .. 127.999
Kc:	0.1	-128 .. 127.999
Upper Limit:	0.99997	-128 .. 127.999
Lower Limit:	-1	-128 .. 127.999

Use External Gains

OK Cancel Help

Kc: Indicates the speed at which the PID value comes out of a saturated limit condition.

Kd: Indicates the controller gain proportional to derivative of (cmd-ref).

Ki: Indicates the controller gain proportional to integral of (cmd-ref).

Kp: Indicates the controller gain proportional to (cmd-ref).

Lower Limit: Indicates the lower limit on both output and integral state.

Upper Limit: Indicates the upper limit on both output and integral state.

Use External Gains: Lets you use external gains rather than Kp, Ki, Kd, and Kc.

Ramp Generator

Embedded Category: C2407, F280x

Block Category: TI 16-bit DMC, TI 32-bit DMC

freq = frequency of ramp

offset = DC offset

gain = gain of generated ramp signal

Operating Mode Availability

- Simulation mode: YES
- C code generation mode: YES

The Ramp Generator block generates periodic ramp output of adjustable gain, frequency, and dc offset. The Ramp Generator block has been retired in v2016.3; however, it is more efficient to use the fixed-point Ramp16, Ramp32, and Ramp32-variable freq blocks under Diagrams > Toolbox > Fixed Point.

Ramp Generator Properties

DSP Sampling Frequency (Hz): 100

Ramp Frequency (Hz): 1.52590218966964

Initial Value: 0

OK Cancel Help

DSP Sampling Frequency (Hz): Specifies the sampling rate of the control running on the target.

Initial Value: Specifies the initial value of the ramp.

Ramp Frequency (Hz): Specifies the repetition rate of the ramp.

Resolver Decoder

Embedded Category: F280x

Block Category: TI 32-bit DMC

Operating Mode Availability

- Simulation mode: YES
- C code generation mode: YES

The Resolver Decoder block lets you decode two sinusoidal resolver signals into a single position.

Resolver Position Estimator Properties

<table style="width: 100%; border-collapse: collapse;"> <tr><td style="padding: 2px;">Pole Count:</td><td style="border: 1px solid gray; width: 60px; text-align: center;">4</td></tr> <tr><td style="padding: 2px;">Zero Cross Count:</td><td style="border: 1px solid gray; width: 60px; text-align: center;">1</td></tr> <tr><td style="padding: 2px;">Control Sample Rate:</td><td style="border: 1px solid gray; width: 60px; text-align: center;">12000</td></tr> <tr><td style="padding: 2px;">Resolver Excitation Rate:</td><td style="border: 1px solid gray; width: 60px; text-align: center;">3000</td></tr> <tr><td style="padding: 2px;">Base Frequency:</td><td style="border: 1px solid gray; width: 60px; text-align: center;">100</td></tr> <tr><td style="padding: 2px;">Angle Calibration:</td><td style="border: 1px solid gray; width: 60px; text-align: center;">-0.085</td></tr> </table>	Pole Count:	4	Zero Cross Count:	1	Control Sample Rate:	12000	Resolver Excitation Rate:	3000	Base Frequency:	100	Angle Calibration:	-0.085	<table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th colspan="2" style="text-align: left; padding: 2px;">Low Pass FIR filter Coefficients</th> </tr> <tr> <th style="border: 1px solid gray; padding: 2px;">order</th> <th style="border: 1px solid gray; padding: 2px;">Coef</th> </tr> </thead> <tbody> <tr><td style="border: 1px solid gray; padding: 2px;">0</td><td style="padding: 2px;">1.5999923938e-005</td></tr> <tr><td style="border: 1px solid gray; padding: 2px;">1</td><td style="padding: 2px;">2.2212472684e-005</td></tr> <tr><td style="border: 1px solid gray; padding: 2px;">2</td><td style="padding: 2px;">4.0011068387e-005</td></tr> <tr><td style="border: 1px solid gray; padding: 2px;">3</td><td style="padding: 2px;">6.699193793e-005</td></tr> <tr><td style="border: 1px solid gray; padding: 2px;">4</td><td style="padding: 2px;">9.9511178341e-005</td></tr> <tr><td style="border: 1px solid gray; padding: 2px;">5</td><td style="padding: 2px;">0.000133176887744</td></tr> <tr><td style="border: 1px solid gray; padding: 2px;">6</td><td style="padding: 2px;">0.000163442319334</td></tr> <tr><td style="border: 1px solid gray; padding: 2px;">7</td><td style="padding: 2px;">0.000186219948918</td></tr> <tr><td style="border: 1px solid gray; padding: 2px;">8</td><td style="padding: 2px;">0.000198433521907</td></tr> </tbody> </table>	Low Pass FIR filter Coefficients		order	Coef	0	1.5999923938e-005	1	2.2212472684e-005	2	4.0011068387e-005	3	6.699193793e-005	4	9.9511178341e-005	5	0.000133176887744	6	0.000163442319334	7	0.000186219948918	8	0.000198433521907
Pole Count:	4																																		
Zero Cross Count:	1																																		
Control Sample Rate:	12000																																		
Resolver Excitation Rate:	3000																																		
Base Frequency:	100																																		
Angle Calibration:	-0.085																																		
Low Pass FIR filter Coefficients																																			
order	Coef																																		
0	1.5999923938e-005																																		
1	2.2212472684e-005																																		
2	4.0011068387e-005																																		
3	6.699193793e-005																																		
4	9.9511178341e-005																																		
5	0.000133176887744																																		
6	0.000163442319334																																		
7	0.000186219948918																																		
8	0.000198433521907																																		

OK
Cancel
Help

Angle Calibration: Indicates the angle calibration.

Base Frequency: Indicates the base frequency.

Control Sample Rate: Indicates the control sample rate.

Pole Count: Indicates the number of poles in the motor.

Resolver Excitation Rate: Indicates excitation rate of the resolver.

Zero Cross Count: Indicates the zero-cross count.

SMO Position Estimator

Embedded Category: F280x

Block Category: TI 32-bit DMC

Operating Mode Availability

- Simulation mode: YES
- C code generation mode: YES

The SMO Position Estimator block uses phase currents to estimate the rotor position. This block is useful in sensorless motor control.

Sliding Mode Observer Position Estimator Properties	
Stator Resistance (Ohms):	<input type="text" value="0.9"/>
Stator inductance (H)	<input type="text" value="0.00435"/>
Base Current (Amps)	<input type="text" value="5"/>
Base phase voltage (volts)	<input type="text" value="185"/>
Sampling Frequency (Hz):	<input type="text" value="0"/>
K _{slide}	<input type="text" value="185"/>
K _{slf}	<input type="text" value="185"/>
<input type="button" value="OK"/> <input type="button" value="Cancel"/> <input type="button" value="Help"/>	

Base Current: Indicates the maximum current flow at the rated load.

Base Phase Voltage: Indicates the base phase voltage.

K_{slf}: Indicates the sliding control filter gain.

K_{slide}: Indicates the sliding control gain.

Sampling Frequency: Indicates sampling rate of the control running on the target.

Stator Inductance: Indicates motor parameters.

Stator Resistance: Indicates the stator resistance.

Space Vector Generator (Magnitude/Frequency)

Embedded Category: C2407, F280x

Block Category: TI 16-bit DMC, TI 32-bit DMC

Block Inputs

- **Frequency:** Fractional value to be multiplied by the sample frequency/6 to determine the frequency of the output waveforms (using a frequency scaling of 1).
- **Gain:** Specifies the amplitude of the output waveforms.

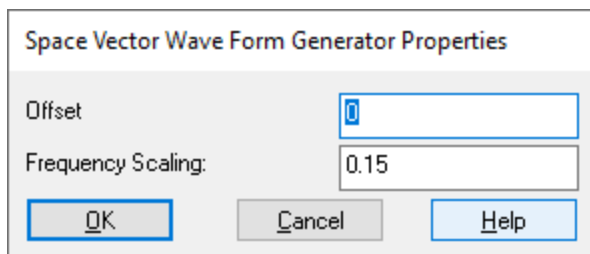
Block Output

- **V_a, V_b, V_c:** Three-phase voltage outputs suitable for driving PWM-based motor control.

Operating Mode Availability

- Simulation mode: YES
- C code generation mode: YES

The Space Vector Generator (Magnitude/Frequency) block calculates the appropriate duty ratios needed to generate a given stator reference voltage using space vector PWM technique. The stator reference voltage is described by its α , β components.



Frequency Scaling: Indicates sampling rate of the control running on the target.

Offset: Indicates the offset that is used in the calculation of the signal.

Space Vector Generator (Quadrature Control)

Embedded Category: C2407, F280x

Block Category: TI 16-bit DMC, TI 32-bit DMC

Operating Mode Availability

- Simulation mode: YES
- C code generation mode: YES

The Space Vector Generator (Quadrature Control) block calculates the appropriate duty ratios needed to generate a given stator reference voltage using space vector PWM technique. The stator reference voltage is described by its magnitude and frequency.

Space Vector PWM

Embedded Category: C2407, F280x

Block Category: TI 16-bit DMC, TI 32-bit DMC

Operating Mode Availability

- Simulation mode: NO
- C code generation mode: YES

The Space Vector PWM block computes a timer period value based on the CPU frequency and the value entered for the Carrier Frequency in the Space Vector PWM Properties dialog box. For example, a carrier frequency setting of 30kHz for a 30MHz CPU yields a timer period of 1000.

The top three inputs (a, b, and c) dynamically determine the duty cycle of each of the six PWM outputs by assigning the proper fraction of the timer period to the compare register. An input value on pin a, b, or c of -1 gives 0% duty cycle; an input value of 0.9999996 gives 100% duty cycle and the PWM varies linearly in between. If 0.5 is supplied to the first input, the compare register would receive 750. The signal on PWM1 would be ON for 750 CPU clock ticks and OFF for 250 CPU clock ticks. If zero is supplied to the first input, the compare register would receive 500. The signal on PWM1 would be ON for 500 CPU clock ticks and OFF for 500 CPU clock ticks.

PWM outputs 2, 4, and 6 are the inverse of PWM outputs 1, 3, and 5, respectively.

The fourth block input (period) is intended for dynamically modulating the carrier frequency. An input value of 0.8 reduces the period by a factor of 0.8%. Thus, the carrier frequency increases by 25% ($1/0.8$).

The Space Vector PWM block performs the following actions:

- Symmetrical pulse-width modulation (PWM) is used to control the inverter with GP timer 1 as PWM time base.

- PWM outputs 1, 3, and 5 control the turn-on and turn-off of the upper power devices.
- PWM outputs 2, 4, and 6 control the turn-on and turn-off of the lower power devices.
- The analog inputs are the amplified and filtered voltage outputs of resistors placed between the sources or emitters of low-side power devices and low-side DC rail.

This also allows PWM period modulation.

Event Manager: For the 2407, you can select EVM B, which is hardwired to the GP timer 3.

Pin Action: Indicates Active Hi or Low. If pin 1 is active high, the deadband unit will cause pin 2 to delay switching high for the deadband interval when pin 1 switches to low. If pin 1 is active low, the deadband unit will cause pin 2 to delay switching low for the deadband interval when pin 1 switches to high.

PWM Frequency: Indicates the base modulation frequency of the PWM. Note that this block is hardwired to GP timer 1 and drives pins PWM 1 - 6.

Speed Calculator

Embedded Category: C2407, F280x

Block Category: TI 16-bit DMC, TI 32-bit DMC

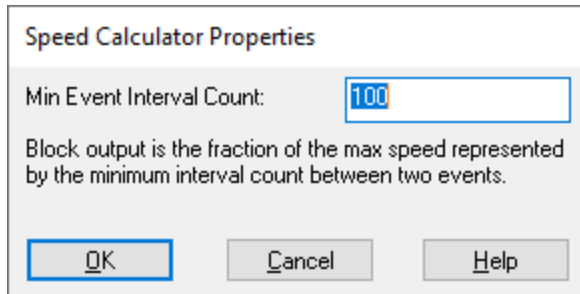
Operating Mode Availability

- Simulation mode: YES
- C code generation mode: YES

The Speed Calculator block calculates the motor speed based on an input time interval, such as the time between two successive events.

The Speed Calculator block is typically used in conjunction with an Event Capture block. The output of the Event Capture block, which represents the time interval between two successive events, is applied as the input to the Speed Calculator block.

The Speed Calculator block determines a 32-bit scaled fixed-point speed of the motor based on the input time interval. To maximize the precision of the calculated speed, you need to know the minimum counts per event interval. You simply enter the minimum counts per event interval that you expect to see (which corresponds to the maximum speed you can measure). In return, Embed automatically calculates the motor speed as a fraction of the maximum speed. The maximum speed is device dependent, based on the actual motion hardware you are using.



Speed Calculator Properties

Min Event Interval Count:

Block output is the fraction of the max speed represented by the minimum interval count between two events.

Min Event Interval Count: Indicates the minimum counts per event interval you expect to see.

V/Hz Profile Generator

Embedded Category: C2407, F280x

Block Category: TI 16-bit DMC, TI 32-bit DMC

The maximum input frequency is normalized to 1. Block input (f_{in}) is a fraction of the maximum frequency.

$$f_{in} \leq LFP: V_{out} = V_{LFP}$$

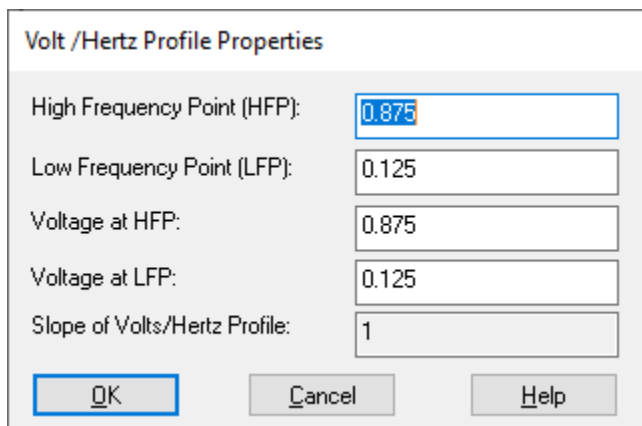
$$LFP \leq f_{in} \leq HFP: V_{out} = V_{LFP} + (f_{in} - V_{LFP}) * slope$$

$$f_{in} > HFP: V_{out} = V_{HFP}$$

Operating Mode Availability

- Simulation mode: YES
- C code generation mode: YES

The V/Hz Profile Generator block generates an output command voltage for a specific input command frequency according to the specified V/Hz profile. This is used for variable speed implementation of AC induction motor drives.



Volt /Hertz Profile Properties

High Frequency Point (HFP):

Low Frequency Point (LFP):

Voltage at HFP:

Voltage at LFP:

Slope of Volts/Hertz Profile:

High Frequency Point: Indicates HFP.

Low Frequency Point: Indicates LFP.

Slope of Volts/Hertz Profile: Indicates slope.

Voltage at HFP: Indicates VHFP.

Voltage at LFP: Indicates VLFP.

Using the TI MotorWare Block Set

The TI MotorWare blocks let you easily set up and use the Texas Instruments InstaSPIN technology.

Angle Estimator

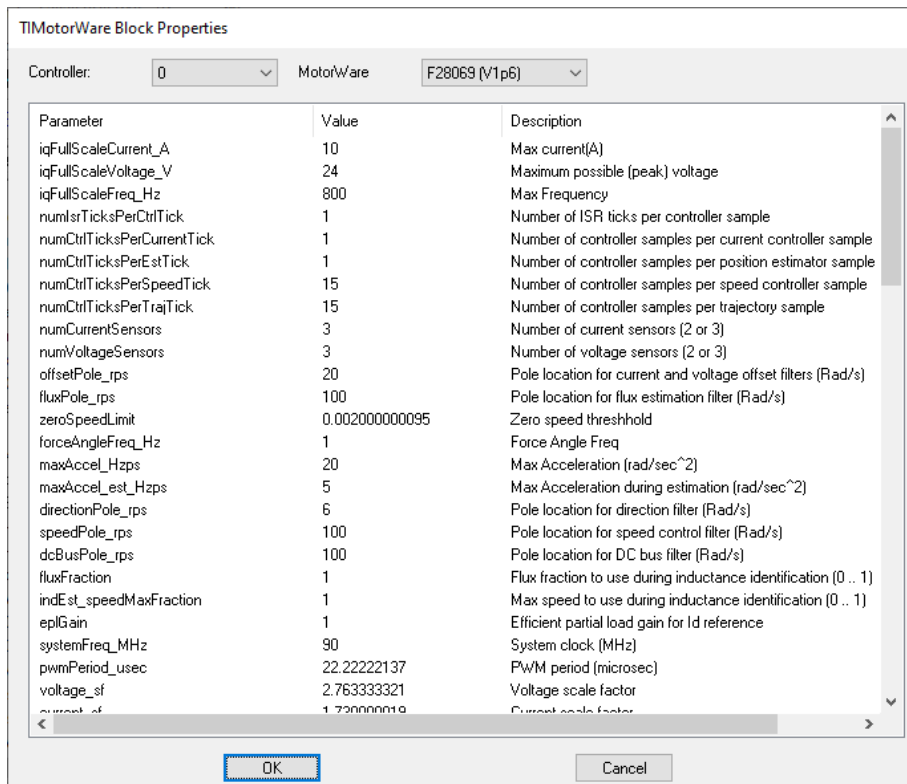
Block Category: TI MotorWare

Block Inputs

- **vbus:** Bus voltage scaled to unity. The value 1 is nominal bus voltage.
- **ia and ib:** Clarke Transform 3-phase current measurement scaled to unity for the peak current.
- **va and vb:** Clarke Transform 3-phase voltage measurement scaled to unity for peak voltage.

You use the Angle Estimator block to define parameters for the motor, motor controller, and motor identification algorithm.

To use the Estimator Read Property or Estimator Write Property blocks, you must have either a Motor Control or Angle Estimator block in your diagram.



Controller: Indicates the controller. This value must match the Controller value in the Estimator Read and Estimator Write blocks.

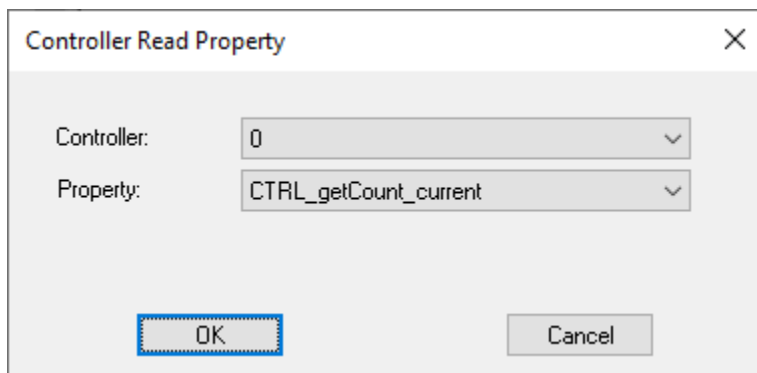
MotorWare: Specifies the version of MotorWare.

Parameter, Value, Description window: Lists the editable motor, motor controller, and motor identification algorithm values. To change the value of a parameter, double click on the value. For more information on the motor, motor controller, and motor identification algorithm values, see the Texas Instruments InstaSPIN documentation.

Controller Read Property

Block Category: TI MotorWare

The Controller Read Property block reads a property of the Texas Instruments motor drive software.



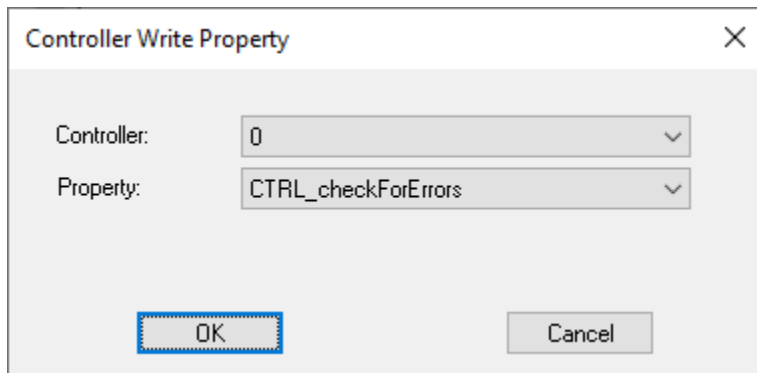
Controller: Indicates the controller instantiation number (0 or 1). This value must match the Controller value in the Motor Control block.

Property: Specifies the property. For more information on the properties, see the Texas Instruments documentation.

Controller Write Property

Block Category: TI MotorWare

The Controller Write Property block writes a property of the Texas Instruments motor drive software.



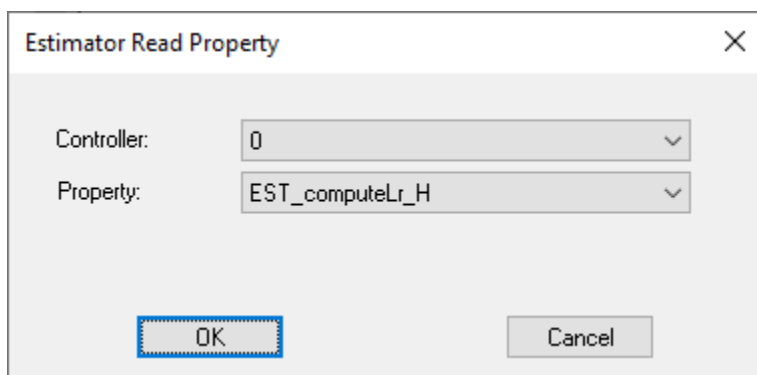
Controller: Indicates the controller instantiation number (0 or 1). This value must match the Controller value in the Motor Control block.

Property: Specifies the property. For more information on the properties, see the Texas Instruments documentation.

Estimator Read Property

Block Category: TI MotorWare

The Estimator Read Property block reads a property of the estimator.



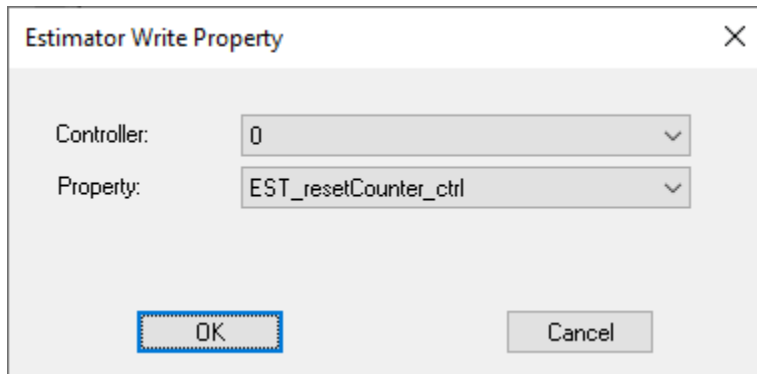
Controller: Indicates the controller instantiation number (0 or 1). This value must match the Controller value in the Angle Estimator block.

Property: Specifies the property. For more information on the properties, see the Texas Instruments documentation.

Estimator Write Property

Block Category: TI MotorWare

The Estimator Read Property block writes a property of the estimator.



Controller: Indicates the controller instantiation number (0 or 1). This value must match the Controller value in the Angle Estimator block.

Property: Specifies the property. For more information on the properties, see the Texas Instruments documentation.

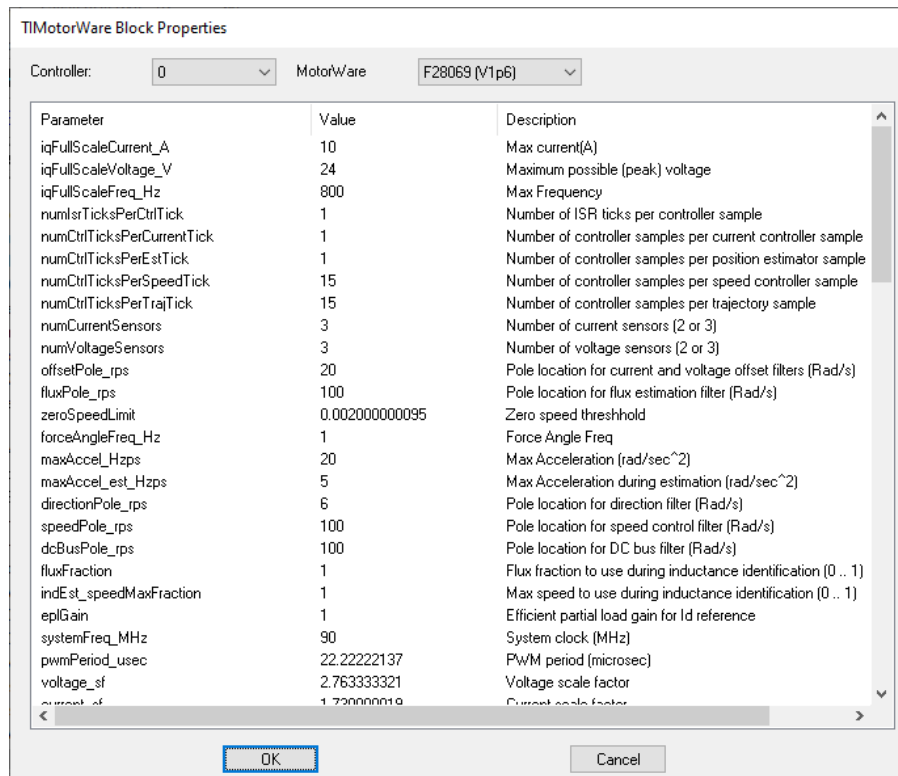
Motor Control

Block Category: TI MotorWare

Block Inputs

- **vbus:** Bus voltage scaled to unity. The value 1 is nominal bus voltage.
- **ia, ib, ic:** Clarke Transform 3-phase current measurement scaled to unity for the peak current.
- **Va, vb, vc:** Clarke Transform 3-phase voltage measurement scaled to unity for peak voltage.

You use the Motor Control block to define parameters for the motor, motor controller, and motor identification algorithm. To use the Controller Read Property or Controller Write Property blocks, you must have a Motor Control block in your diagram.



Controller: Indicates the controller. This value must match the Controller value in the Control Read and Control Write blocks.

MotorWare: Specifies the version of MotorWare.

Parameter, Value, Description window: Lists the editable motor, motor controller, and motor identification algorithm values. To change the value of a parameter, double click on the value. For more information on the motor, motor controller, and motor identification algorithm values, see the Texas Instruments documentation.

Using the Fixed Point Block Set

The Fixed Point block set includes blocks and added capabilities to existing blocks to simulate the effects of scaled fixed-point arithmetic.

Fixed Point block set

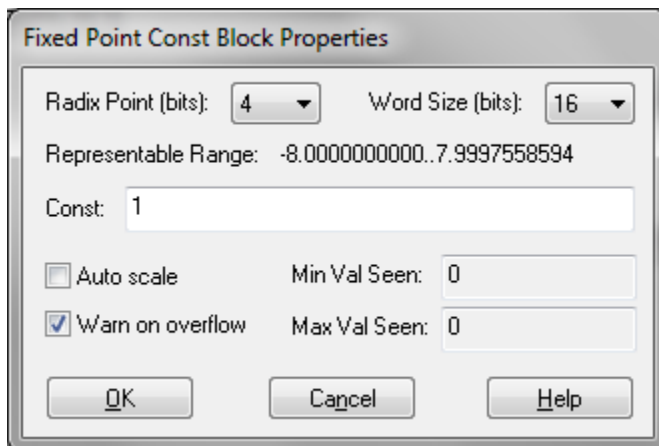
The Fixed Point block set — listed under the Blocks menu — consists of 30+ blocks that let you design and simulate the performance of fixed-point algorithms prior to code generation and execution on an embedded platform.

When you insert a fixed-point block into a diagram, the block displays specific information about the block. For example, the const block appears as follows:



- **1:** Indicates the const value
- **fx:** Indicates fixed point
- **4.16:** Indicates the fixed-point format, with 16 total bits and 4 bits to the left of the radix point. represents the const value
- **Yellow connector:** Indicates Scaled Int. To display connectors, activate **Connector Labels** and **Data Type** in the **View** menu.

The Fixed Point Const dialog box lets you manipulate the const value, word size, and radix point.



The dialog box also shows the **Representable Range**; that is, the range of acceptable values based on the **Radix Point** and **Word Size** parameters. Increasing the **Radix Point** value increases the range; decreasing the value, decreases the range.

The [Learning Center](#) provides several fixed-point videos to help get you started using the Fixed Point block set.

less than

$$y = \begin{cases} 1 & \text{if } x_1 < x_2 \\ 0 & \text{if } x_1 \geq x_2 \end{cases}$$

The less than (<) block produces an output signal of 1 if and only if input signal x_1 is less than input signal x_2 . Otherwise, the output is 0. On the connectors, “l” represents x_1 and “r” represents x_2 .

If you right click the < block, the Boolean block menu appears allowing you to assign a different function to the block.

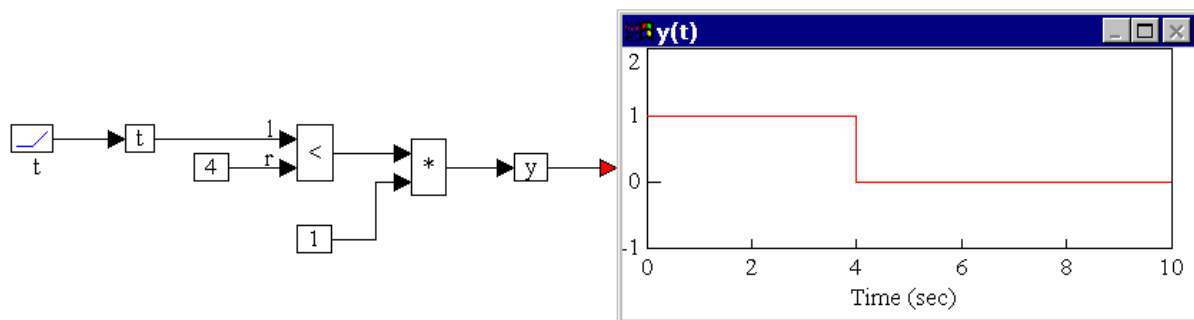
Examples

1. Simple if-then-else construct

Consider a variable y such that:

If $t < 4$ then $y = 1$; else $y = 0$

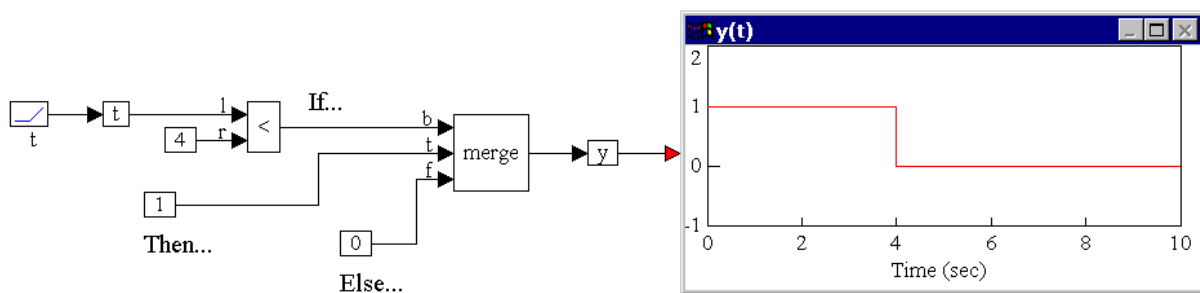
Assume that t is simulation time. This system can be realized as:



By multiplying a constant value one with the output of the < block, y is guaranteed to assume a value of zero until the inequality is true. When the inequality is true, y assumes a value equal to the output of the * block.

2. Modified if-then-else construct

The previous example can also be realized as:



The key difference in implementation is the use of a merge block rather than a * block. The merge block explicitly depicts the if-then-else structure; the * block is a shortcut and can lead to confusion.

less than or equal to

$$y = \begin{cases} 1 & \text{if } x_1 \leq x_2 \\ 0 & \text{if } x_1 > x_2 \end{cases}$$

Block Inputs: Two scalar inputs.

The less than or equal to (\leq) block produces an output signal of 1 if and only if input signal x_1 is less than or equal to input signal x_2 . Otherwise, the output is 0. On the connectors, “l” represents x_1 and “r” represents x_2 .

If you right click the \leq block, the Boolean block menu appears allowing you to assign a different function to the block.

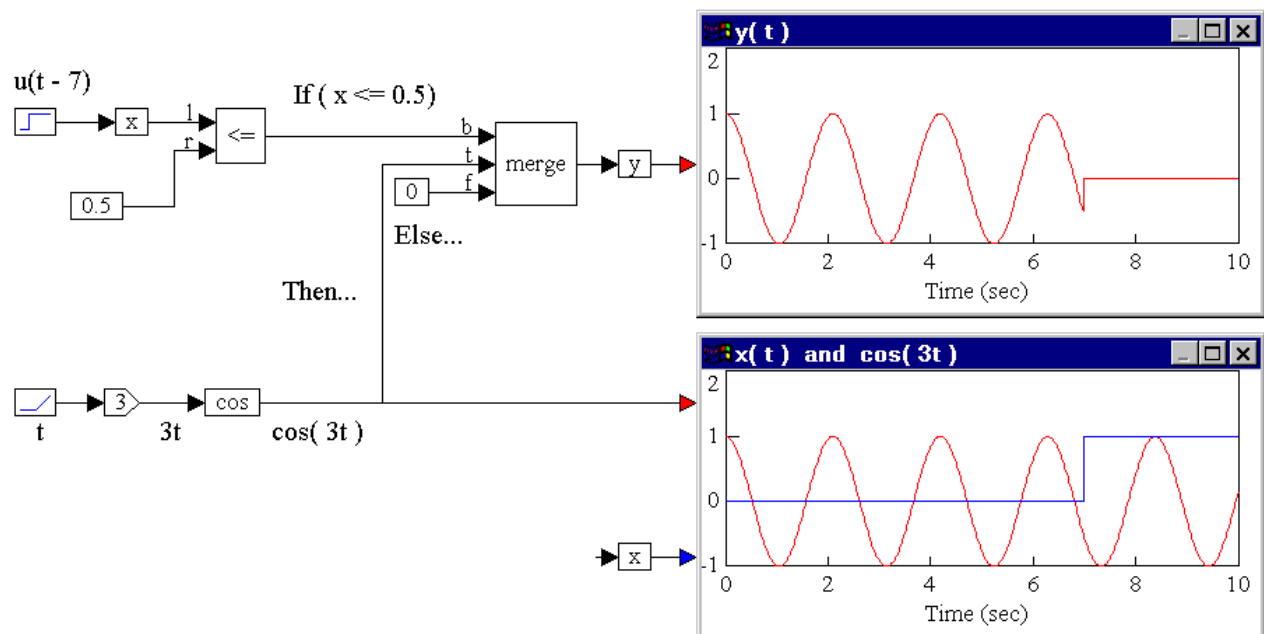
Examples

1. Simple if-then-else construct

Consider a variable y such that:

If $x \leq 0.5$ then $y = \cos(3t)$; else $y = 0$

where t is simulation time. Let x be a unit step delayed by 7s, represented as $u(t - 7)$. This system can be realized as shown below.



Until the onset of the step input at $t = 7$ s, the Boolean inequality $x \leq 0.5$ evaluates to true, and y takes on a value of $\cos(3t)$. At $t = 7$ s, the Boolean inequality evaluates to false and remains false for the duration of the simulation. Consequently, from this point onwards, y takes on the value of 0. The lower plot block monitors the outputs of the \cos and variable x blocks.

equal to (==)

$$y = \begin{cases} 1 & \text{if } x_1 = x_2 \\ 0 & \text{if } x_1 \neq x_2 \end{cases}$$

Block Inputs: Two scalar inputs labeled “l” (for x_1) and “r” (for x_2).

The $==$ block is useful for evaluating the Boolean $==$ equality. The output of the $==$ block is one if and only if input “l” is identically equal to input “r;” otherwise, the output is zero.

If you right click the $==$ block, the Boolean block menu appears allowing you to assign a different function to the block.

Boolean equality comparisons of floating-point variables and non-integer constants: As with programming in any language, it is generally not a good idea to perform Boolean equality comparisons involving floating-point variables and non-integer constants. These types of comparisons should be converted to Boolean inequality comparisons. For example, {If position is equal to π , then ...} should be converted to {If position is greater than or equal to $\langle \pi \text{roundedoff} \rangle$, then ...}. The reason for this is because a floating-point variable, such as position, is rarely exactly equal to a non-zero non-integer value, particularly if it is obtained by solving one or more equations.

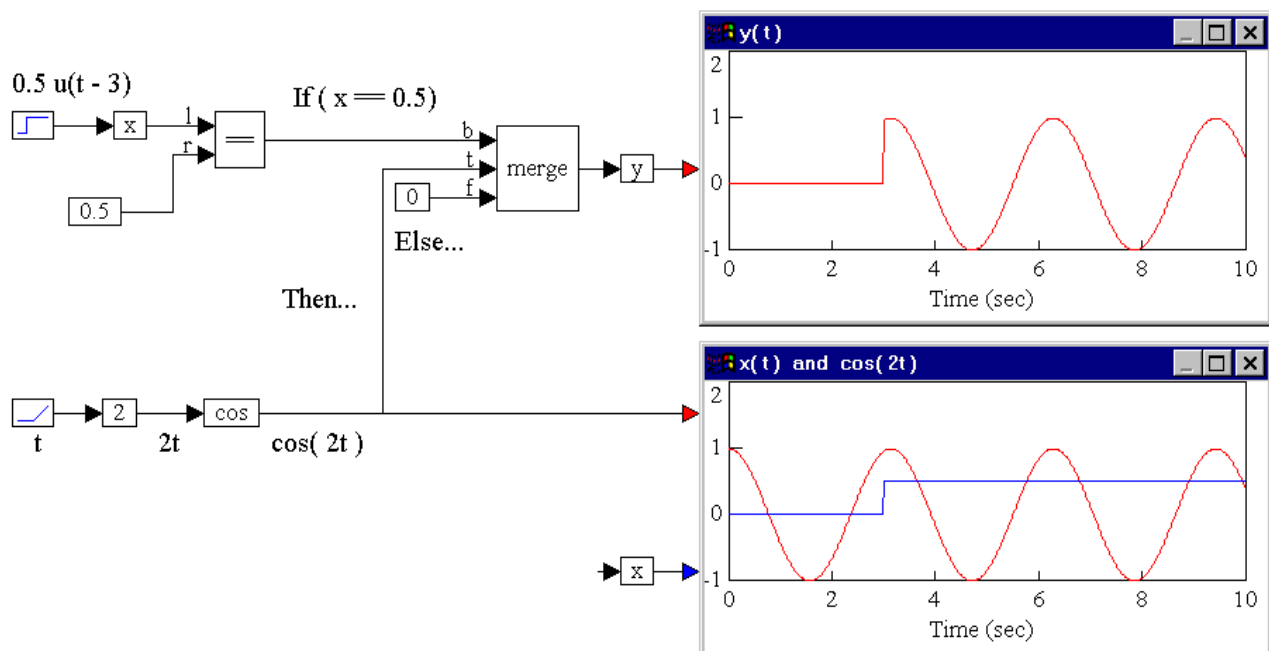
Examples

1. Comparing constants

Consider a variable y such that:

$$\text{If } x = 0.5 \text{ then } y = \cos(2t); \text{ else } y = 0$$

where t is simulation time. Let x be a step function of amplitude 0.5, delayed by 3s. This is usually represented as $0.5 u(t - 3)$. This system can be realized as shown below.



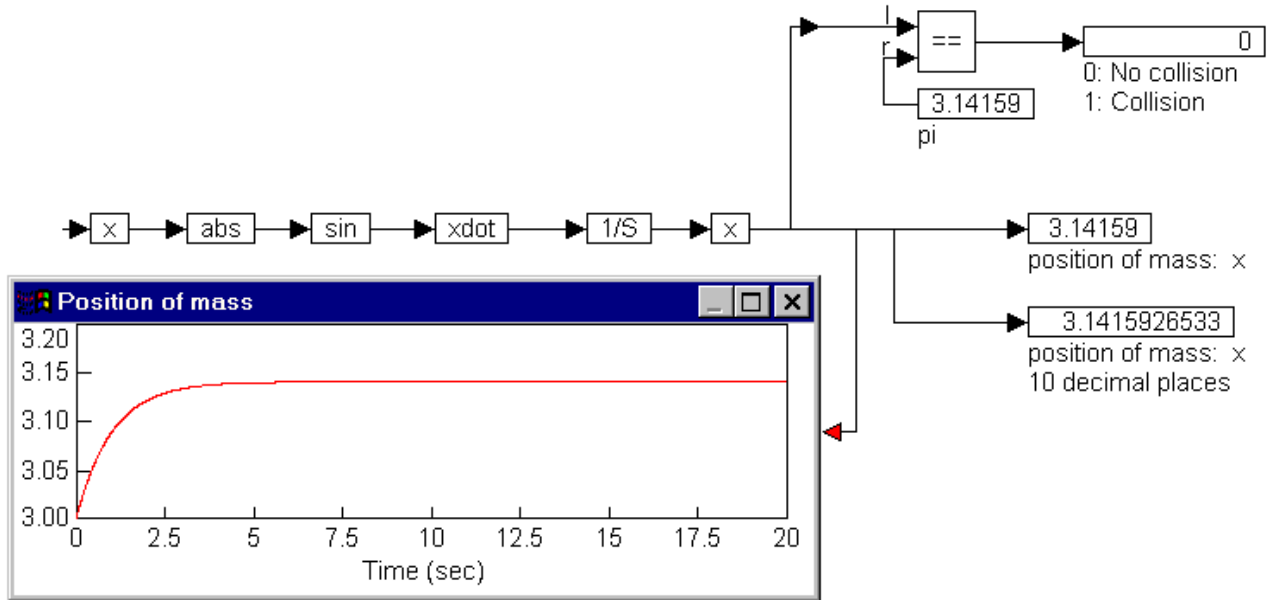
Until the onset of the step input at $t = 3$ s, the Boolean equality $x == 0.5$ evaluates to false, and y takes on a value of 0. At $t = 3$ s, the Boolean equality evaluates to true, and remains true for the duration of the simulation. Consequently, from this point onwards, y takes on the value of $\cos(2t)$. The lower plot block is used to monitor the outputs of the \cos block and the variable x .

2. Comparing a floating-point variable with a non-integer constant

In a collision detection problem, if position x of a mass in motion is equal to π , then a collision is assumed to have occurred with an immovable wall that is located at $x = \pi$. Furthermore, the position of the mass is assumed to be given by the solution of the following first order differential equation:

$$\dot{x} = \sin(|x|)$$

The initial condition is assumed to be $x(0) = 3.0$. It is tempting to realize this system as:



From the result shown in the plot block, at around $t = 7s$, the mass arrives at the boundary located at π . However, the collision detection logic, using an `==` block that compares x with a constant value of π , never detects the collision. This is because the final mass position, as obtained from the output of the integrator, is 3.141592653, which is not equal to 3.14159.

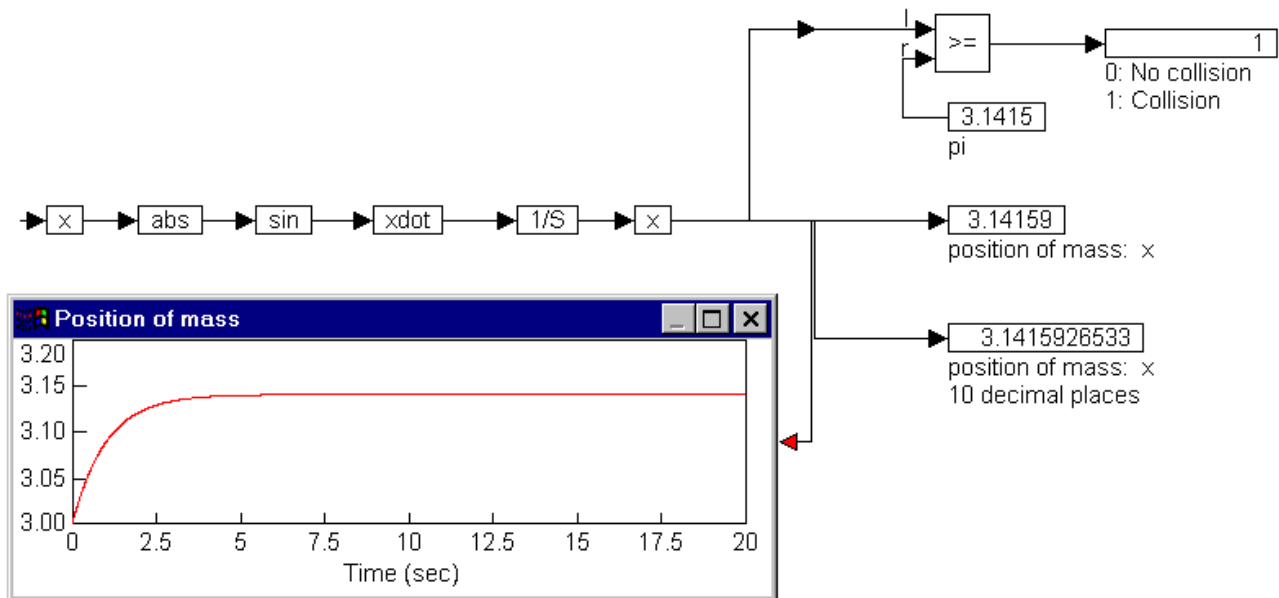
It is clear from the plot block, that for all practical purposes, the mass collided with the wall around $t = 7s$. To capture this reality in the simulation, convert the Boolean equality comparison:

If $x = 3.14159$

Then to a Boolean inequality comparison:

If $x \geq 3.1415$

Then after reducing the const block to four decimal places with no round-off, the system can be realized as:



Except for replacing the == block with the >= block, this diagram is like the previous one. In this case, the collision detection logic detects a collision around $t = 8s$. Obviously, the time at which the collision is detected depends on the number of decimal places retained for the π approximation.

not equal to (!=)

$$y = \begin{cases} 1 & \text{if } x_1 \neq x_2 \\ 0 & \text{if } x_1 = x_2 \end{cases}$$

Block Inputs: Two scalar inputs.

The != block produces an output signal of 1 if and only if the two scalar input signals are not equal. On the connectors, “l” represents x_1 and “r” represents x_2 .

If you right click the != block, the Boolean block menu appears allowing you to assign a different function to the block.

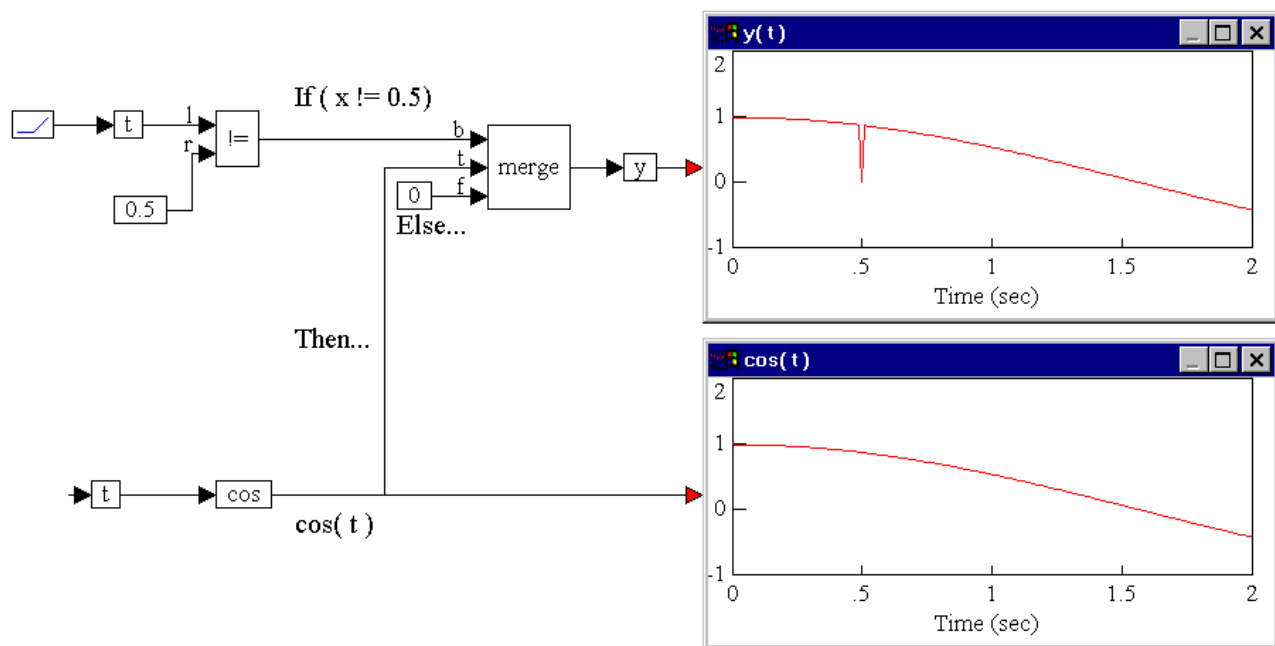
Examples

1. Comparing constants

Consider a variable y such that:

If $t \neq 0.5$ then $y = \cos(t)$; else $y = 0$

where t is simulation time. This system can be realized as shown below.



Until the value of t reaches 0.5, the Boolean inequality $t \neq 0.5$ evaluates to true, and y takes on a value of $\cos(t)$. At $t = 0.5s$, the Boolean inequality evaluates to false, and at the very next time step, returns to true, and remains true for the duration of the simulation. Consequently, at the moment $t = 0.5s$, y takes on the value of 0, and at every other point, y is equal to $\cos(t)$.

greater than

$$y = \begin{cases} 1 & \text{if } x_1 > x_2 \\ 0 & \text{if } x_1 \leq x_2 \end{cases}$$

Block Inputs: Two scalar inputs labeled “l” and “r.”

The greater than (>) block is useful in evaluating the Boolean > inequality. The output of the > block is 1 if and only if input “l” > input “r;” otherwise the output is 0.

If you right click the > block, the Boolean block menu appears allowing you to assign a different function to the block.

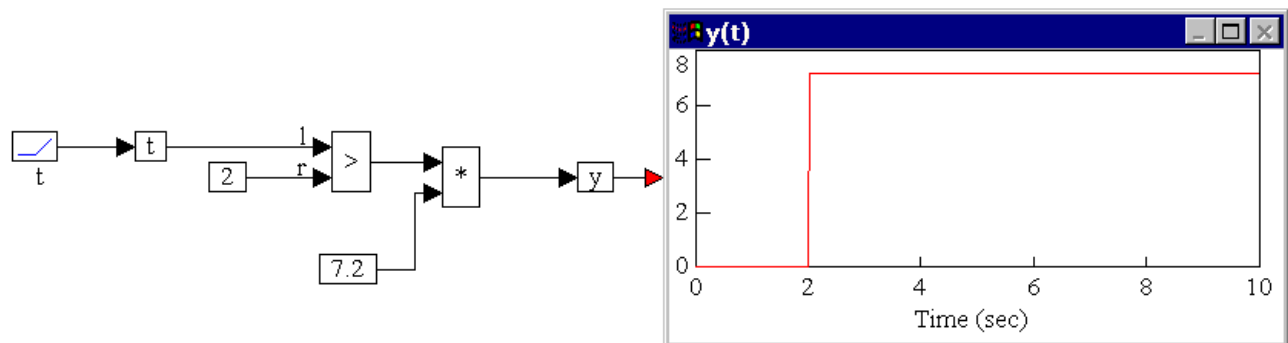
Examples

1. Simple if-then-else construct

Consider a variable y such that:

If $t > 2$ then $y = 7.2$; else $y = 0$

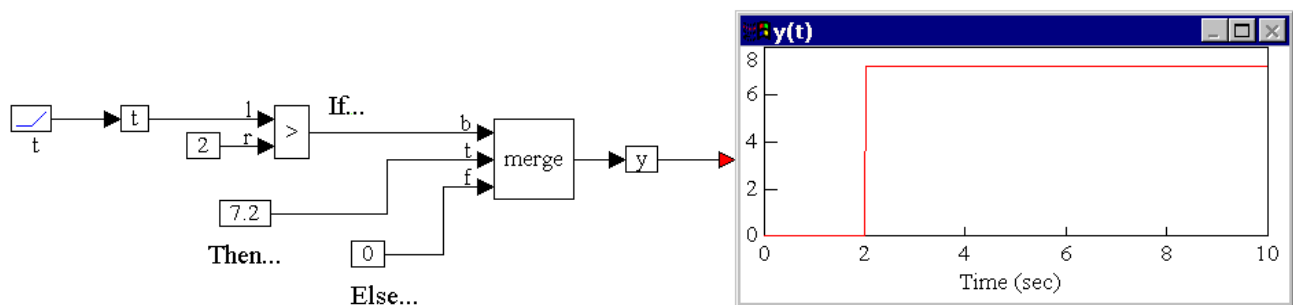
Assume that t is simulation time. This system can be realized as shown below.



By multiplying a constant value of 7.2 with the output of the > block, y is guaranteed to assume a value of zero until the inequality is true. When the inequality is true, y assumes a value equal to the output of the * block.

2. Modified if-then-else construct

Using the above equation, it can also be realized as:



The key difference in implementation is the use of a merge block rather than a * block. The merge block explicitly depicts the if-then-else structure, whereas the * block is a shortcut and can lead to confusion.

greater than or equal to

$$y = \begin{cases} 1 & \text{if } x_1 \geq x_2 \\ 0 & \text{if } x_1 < x_2 \end{cases}$$

Block Inputs: Two scalar inputs.

The greater than or equal to (\geq) block produces an output signal of 1 if and only if input signal x_1 is greater than or equal to input signal x_2 . Otherwise, the output is 0. On the connectors, "l" represents x_1 and "r" represents x_2 .

If you right click the \geq block, the Boolean block menu appears allowing you to assign a different function to the block.

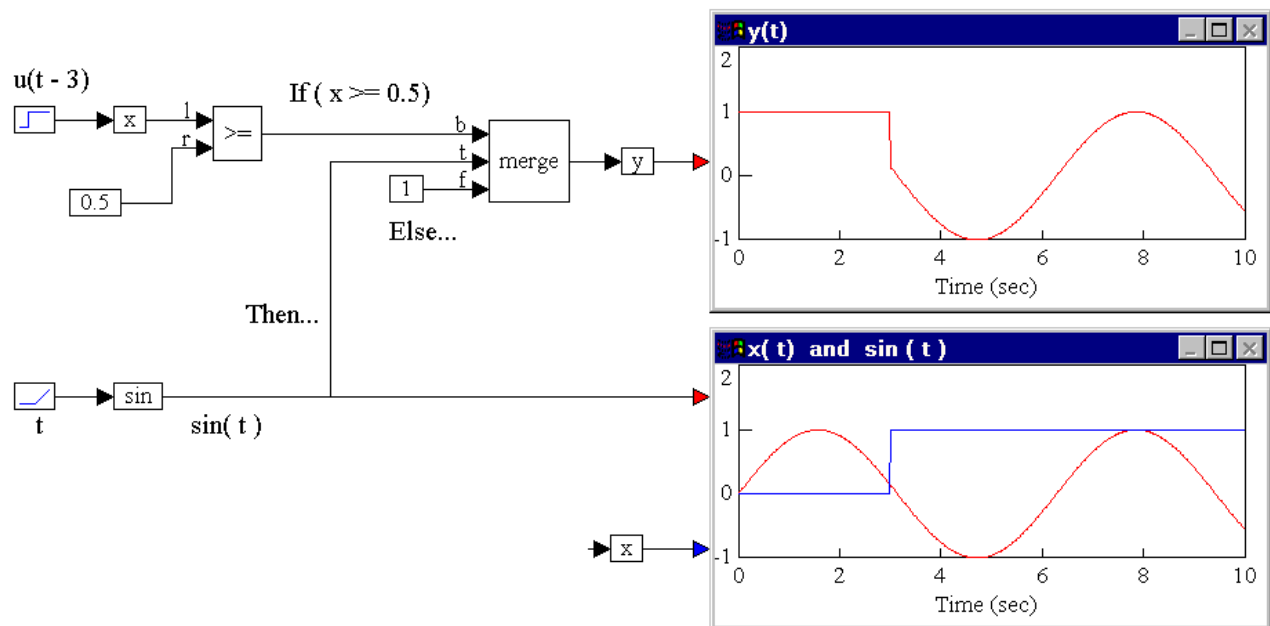
Examples

1. Simple if-then-else construct

Consider a variable y such that:

If $x \geq 0.5$ then $y = \sin(t)$; else $y = 1$

where t is simulation time. Let x be a unit step delayed by 3s. This is usually represented as $u(t - 3)$. This system can be realized as shown below.



Until the onset of the step input at $t = 3$ s, the Boolean inequality $x \geq 0.5$ evaluates to false and y takes on a value of 1. At $t = 3$ s, the Boolean inequality evaluates to true and remains true for the duration of the simulation duration. Consequently, from this point onwards, y takes on the value of $\sin(t)$.

-X (negate)

$y = -x$

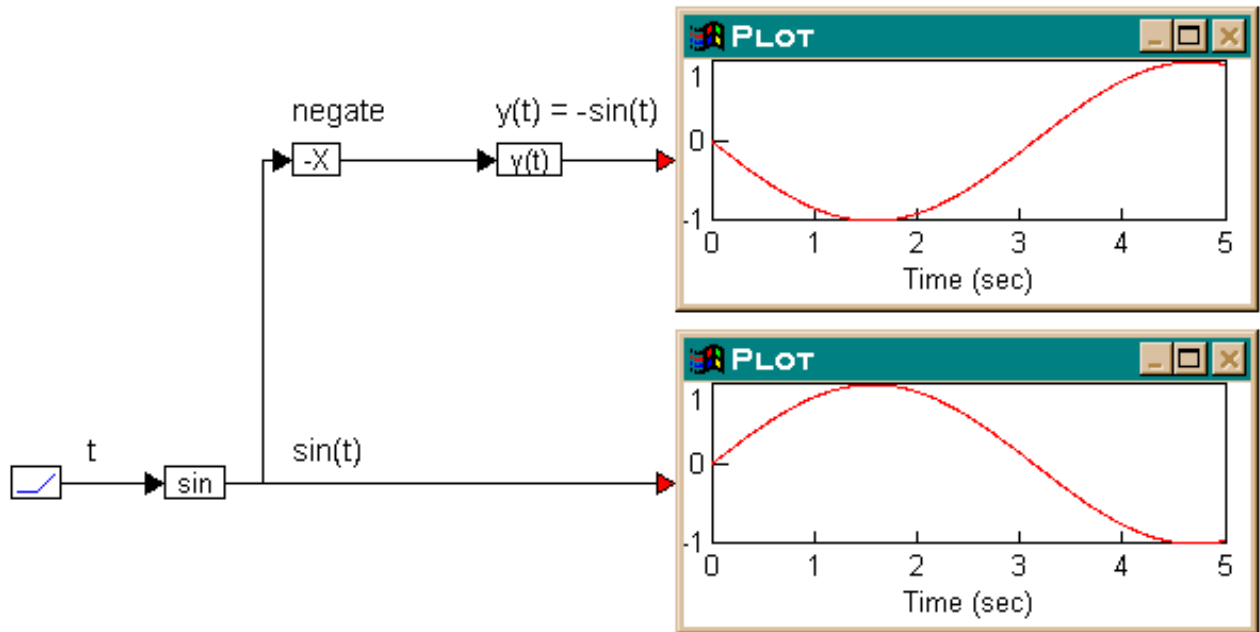
Block Inputs: Scalar, vector, or matrix.

The $-X$ block negates the input signal.

Examples

1. Negation of a scalar

Consider the equation $y(t) = -\sin(t)$, which can be realized as shown below.



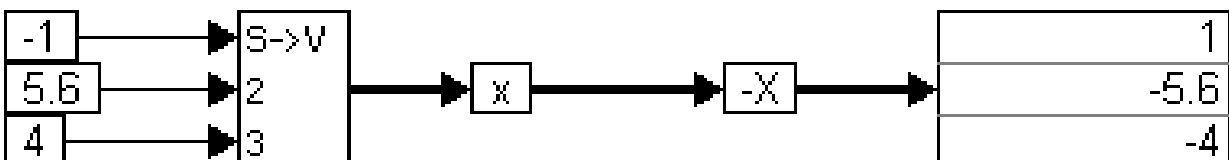
A ramp block is used to access simulation time t , a sin block generates $\sin(t)$, and a $-X$ block converts $\sin(t)$ to $-\sin(t)$. Both $\sin(t)$ and $y(t)$ are plotted for comparison.

2. Negation of a vector

Consider the equation:

$$\mathbf{z} = -\mathbf{x}$$

where $\mathbf{x} = [-1 \ 5.6 \ 4]$. This equation can be realized as:



A scalarToVector block creates a three-element vector from the constant values -1, 5.6, and 4. When the simulation runs, the $-X$ block performs an element-by-element negate operation on the incoming vector.

3. Negation of a matrix

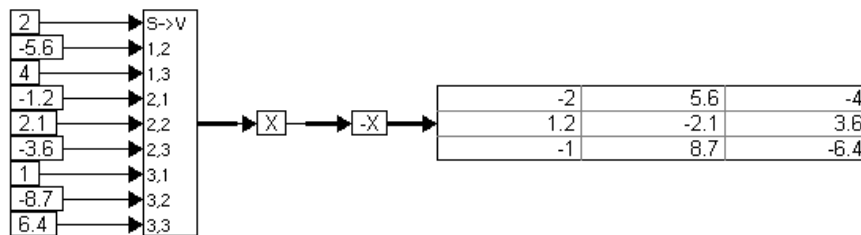
Consider the equation:

$$\mathbf{Z} = -\mathbf{X}$$

where

$$\mathbf{X} = \begin{bmatrix} 2 & -5.6 & 4 \\ -1.2 & 2.1 & -3.6 \\ 1 & -8.7 & 6.4 \end{bmatrix}$$

This equation can be realized as:



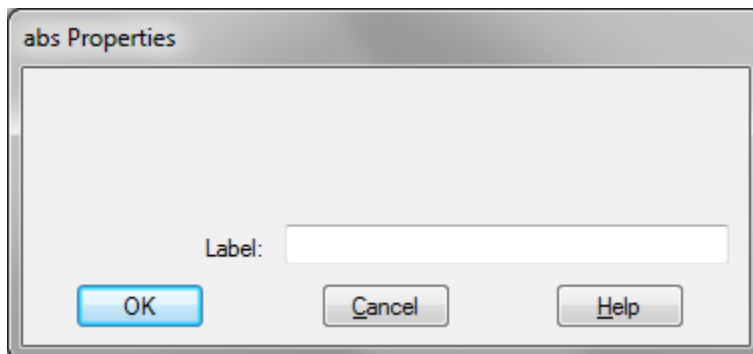
When the simulation runs, the $-X$ block performs an element-by-element negate operation on the incoming matrix.

abs

$$y = |x|$$

Block Inputs: Real, complex, or fixed-point scalars, or vectors or matrices.

The abs block produces the absolute value of the input signal.

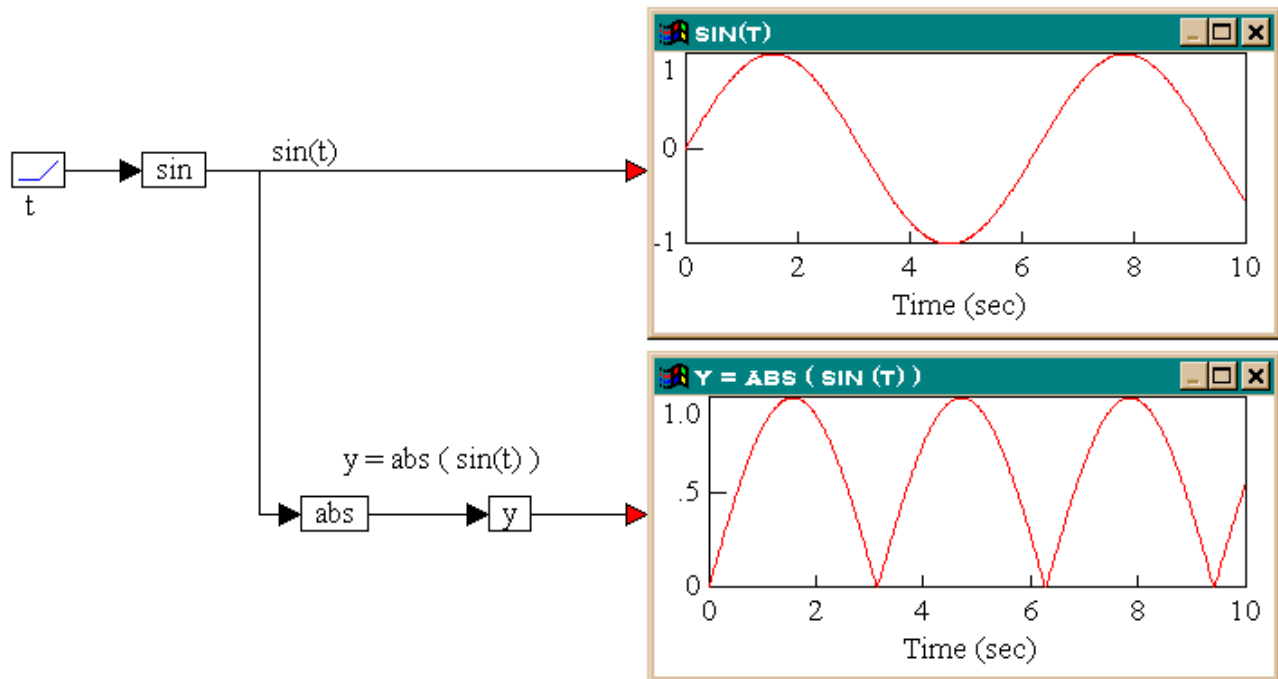


Label: Indicates a user-defined block label.

Examples

1. Absolute value of a scalar

Consider the equation $y = \text{abs}(\sin(t))$, which can be realized as shown below.



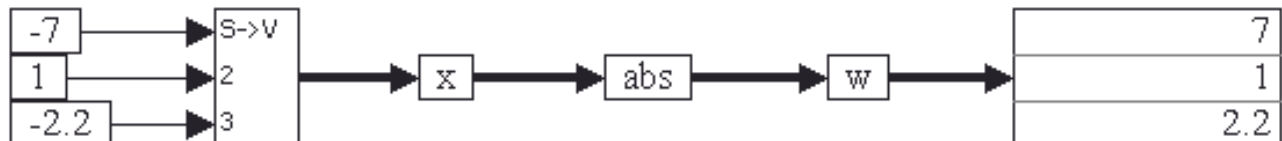
The results in the two plot blocks show that the abs block computes the absolute value of the input signal.

2. Absolute value of a vector

Consider the equation:

$$\mathbf{w} = \text{abs}(\mathbf{x})$$

where $\mathbf{x} = [-7 \ 1 \ -2.2]$. This equation can be realized as:



When the simulation runs, the abs block computes and outputs an element-by-element absolute value of the vector \mathbf{x} .

3. Absolute value of a matrix

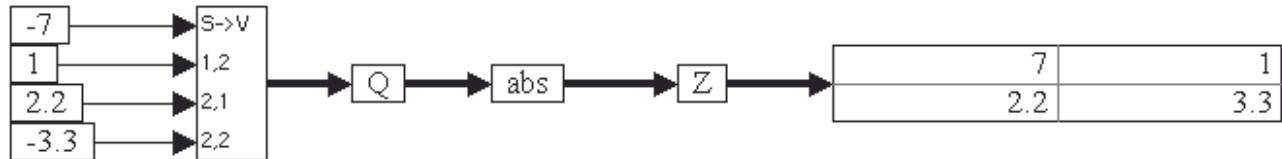
Consider the equation:

$$\mathbf{Z} = \text{abs}(\mathbf{Q})$$

where

$$\mathbf{Q} = \begin{bmatrix} -7 & 1 \\ 2.2 & -3.3 \end{bmatrix}$$

This equation can be realized as shown below.



Four const blocks provide the vector element values of \mathbf{Q} through a scalarToVector block. When the simulation runs, the abs block computes the element-by-element absolute value of the incoming matrix.

and

$$y = x_1 \text{ bitwise AND } x_2$$

Block Inputs: Real, complex, or fixed-point scalars, or vectors or matrices.

The and block produces the bitwise AND of 2 - 256 scalar input signals.

If you right click the and block, the Boolean block menu appears allowing you to assign a different function to the block.

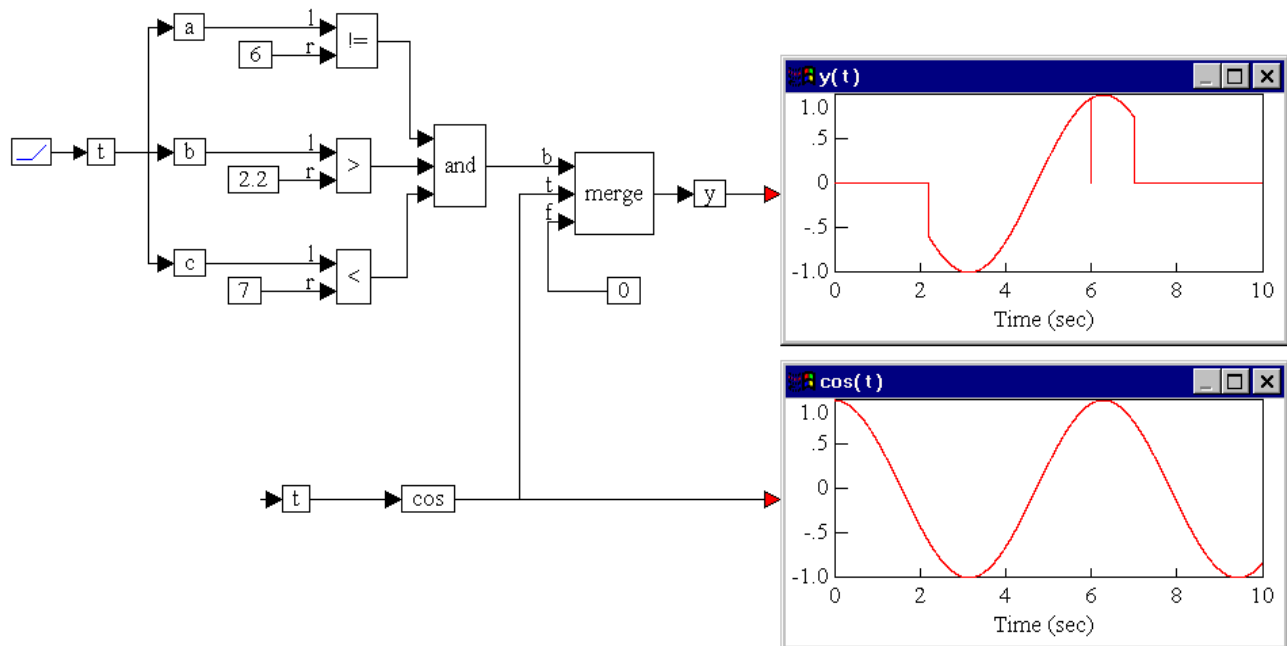
Examples

1. Three variable and

Consider a variable y such that:

If $a \neq 6$ and $b > 2.2$ and $c < 7$, then $y = \cos(t)$; else $y = 0$

where t is simulation time. Furthermore, let t be the input to all three parameters a , b , and c . This system can be realized as shown below.



The output of the and block is true only when all the three inputs are true. This happens in the range $t = (2.2, 7)$, except for the instant $t = 6$. This result is apparent from the top plot block. The variable y is equal to $\cos(t)$ in the range $t = (2.2, 7)$. At the instant $t = 6$, variable a is momentarily false, and consequently, $y = 0$ at $t = 6$, since the output of the and block evaluates to false at that instant.

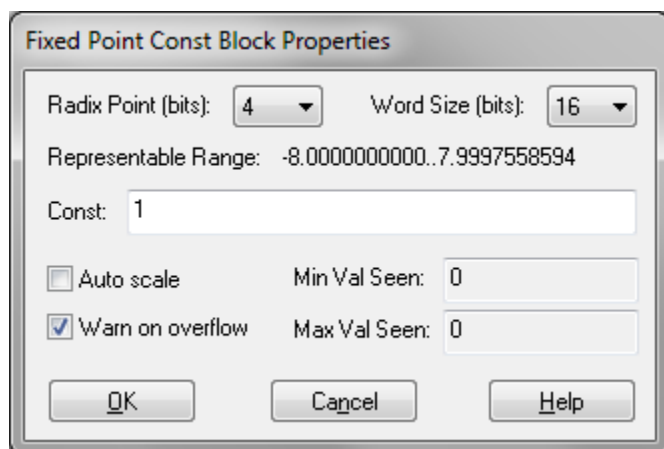
atan2

Block Inputs: fx1.16 or fx1.32 format.

The atan2 block computes the four-quadrant inverse tangent (arctangent) of the input signals in scaled fixed-point notation. The block will automatically take on fx1.16 or fx1.32 based on the data type of the input signal. The atan2 block uses the signs of both input signals to determine the output signal. The block output is in “per unit” angles. This means the angle output is implicitly divided by 2π so that a cycle of sin/cos input will produce a value sweeping from zero to one (instead of zero to 2π). Typically, this block is fed by a sin/cos sensor input to produce a “per unit” angle value.

const

The const block generates a constant signal in scaled fixed-point notation.



Auto Scale: Rescales the range when either the maximum or minimum value is exceeded.

Const: Indicates the value of the output signal.

Max Val Seen: Displays the high watermark of values that come through the block. This is a read-only field.

Min Val Seen: Displays the low watermark of values that come through the block. This is a read-only field.

Radix Point (bits): Sets the binary point.

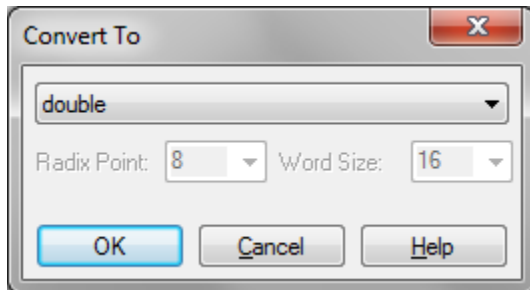
Representable Range: Indicates the range of values based on the selected radix point. This is a read-only field.

Warn on Overflow: Activates a warning dialog box that appears when an overflow error occurs. An overflow error occurs when a result value is outside the currently acceptable range. You must also activate the Enable Overflow Alert Messages in the dialog box for the [Fixed Point Block Set Configure](#) command under the Tools menu.

Word Size (bits): Sets the word size for the target architecture. The word size can be overridden using the Override Word Size option in the dialog box for the [Fixed Point Block Set Configure](#) command under the Tools menu. Most embedded targets use 16- or 32-bit word size.

convert

The convert block converts the data type of the input signal to one of the following: char, unsigned char, short, unsigned short, int, long, unsigned long, float, double, void*, enum Comboltem, matrix double, scaled int, string, complex structure, or matrix complex. To check for overflow errors, activate **Warn Numeric Overflow** under the **Preferences** tab in the dialog box for **System > System Properties**.



Radix Point: This parameter lights up only when Scaled Int is selected. It represents the number of bits provided for the integral part of the number. The difference between the word size and the radix point represents the mantissal (or fractional part of the number).

Word Size: Sets the word size for the target architecture. The word size can be overridden using the Override Word Size option in the dialog box for the [Fixed Point Block Set Configure](#) command under the Tools menu.

COS

$$y = \cos x$$

Block Inputs: fx1.16 or fx1.32 format.

The cos block produces the cosine of the input signal in scaled fixed-point notation. The block automatically uses fx1.16 or fx1.32 based on the data type of the input signal. The input signal is represented in “per unit” angles. This means the angle input is implicitly multiplied by 2π so that a value sweeping from zero to one will produce a complete cycle of sin output. Typically, this block is fed by a repeating fixed-point ramp that runs from zero to one.

CRC16

Block Inputs:

enable: Must be one in order for the block to read "newVal" and apply it to existing CRC value.

reset: If it goes to 1, the internal CRC value is zero.

newVal: New value to be applied to internal CRC value.

The CRC block performs a cyclic redundancy check (CRC) on the data and compares the resulting checksum with the appended checksum. The CRC block lets you select the CRC method and whether the result is available via an output pin.

CRC Output Pin: Indicates current 16-bit CRC value.

CRC Type: Indicates the CRC method. Choose among CCITT, ANSI, and DECT. These methods are described in Wikipedia under [Cyclic Redundancy Check](#).

Initial CRC Value (hex): Indicates initial 16-bit CRC seed.

Polynomial: Indicates polynomial word.

Unit: Each unit maintains an internal CRC buffer. If you need multiple concurrent CRC streams, make sure each one has a unique unit number.

div

The div block produces the quotient of two inputs in scaled fixed notation.

Auto Scale: Rescales the range when either the maximum or minimum value is exceeded.

Max Val Seen: Displays the high watermark of values that come through the block. This is a read-only field.

Min Val Seen: Displays the low watermark of values that come through the block. This is a read-only field.

Radix Point (bits): Sets the binary point.

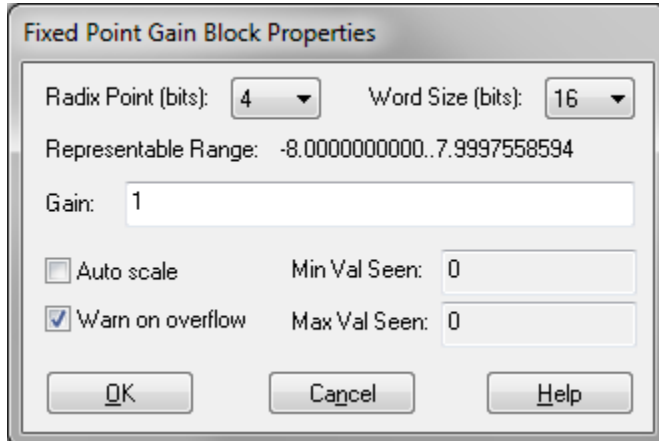
Representable Range: Indicates the range of values based on the selected radix point. This is a read-only field.

Warn on Overflow: Activates a warning dialog box that appears when an overflow error occurs. An overflow error occurs when a result value is outside the currently acceptable range. You must also activate the Enable Overflow Alert Messages in the dialog box for the [Fixed Point Block Set Configure](#) command under the Tools menu.

Word Size (bits): Sets the word size for the target architecture. The word size can be overridden using the Override Word Size option in the dialog box for the [Fixed Point Block Set Configure](#) command under the Tools menu.

gain

The gain block multiplies the input signal by the gain amount in scaled fixed notation.



Auto Scale: Rescales the range when either the maximum or minimum value is exceeded.

Gain: Indicates the constant multiplier of the input signal.

Max Val Seen: Displays the high watermark of values that come through the block. This is a read-only field.

Min Val Seen: Displays the low watermark of values that come through the block. This is a read-only field.

Radix Point (bits): Sets the binary point.

Representable Range: Indicates the range of values based on the selected radix point. This is a read-only field.

Warn on Overflow: Activates a warning dialog box that appears when an overflow error occurs. An overflow error occurs when a result value is outside the currently acceptable range. You must also activate the Enable Overflow Alert Messages in the dialog box for the [Fixed Point Block Set Configure](#) command under the Tools menu.

Word Size (bits): Sets the word size for the target architecture. The word size can be overridden using the Override Word Size option in the dialog box for the [Fixed Point Block Set Configure](#) command under the Tools menu.

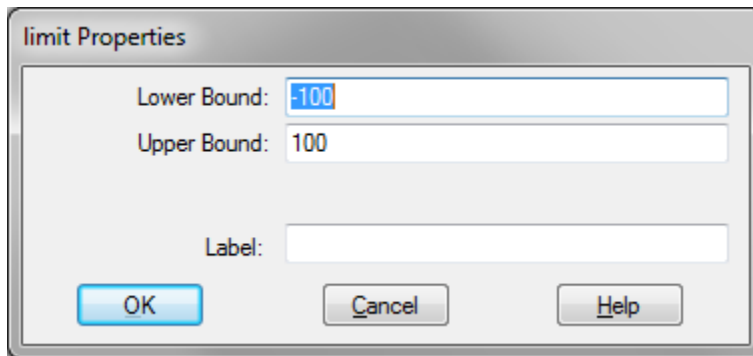
limit

$$y = \begin{cases} x_1 & \text{if } lb \leq x_1 \leq ub \\ lb & \text{if } x_1 < lb \\ ub & \text{if } x_1 > ub \end{cases}$$

Block Inputs: Scalar.

The limit block limits the output signal to a specified upper and lower bound. If the input is less than the lower bound, the limit block limits the output to the lower bound. Similarly, if the input is greater than the upper bound, the limit block limits the output to the upper bound. If the input falls within the specified bounds, the input is transferred to the output unchanged.

The limit block is particularly useful for simulating variables or processes that reach saturation.



Label: Indicates a user-defined block label.

Lower Bound: Indicates the lowest value that the output signal can attain. The default is -100.

Upper Bound: Indicates the highest value that the output signal can attain. The default is 100.

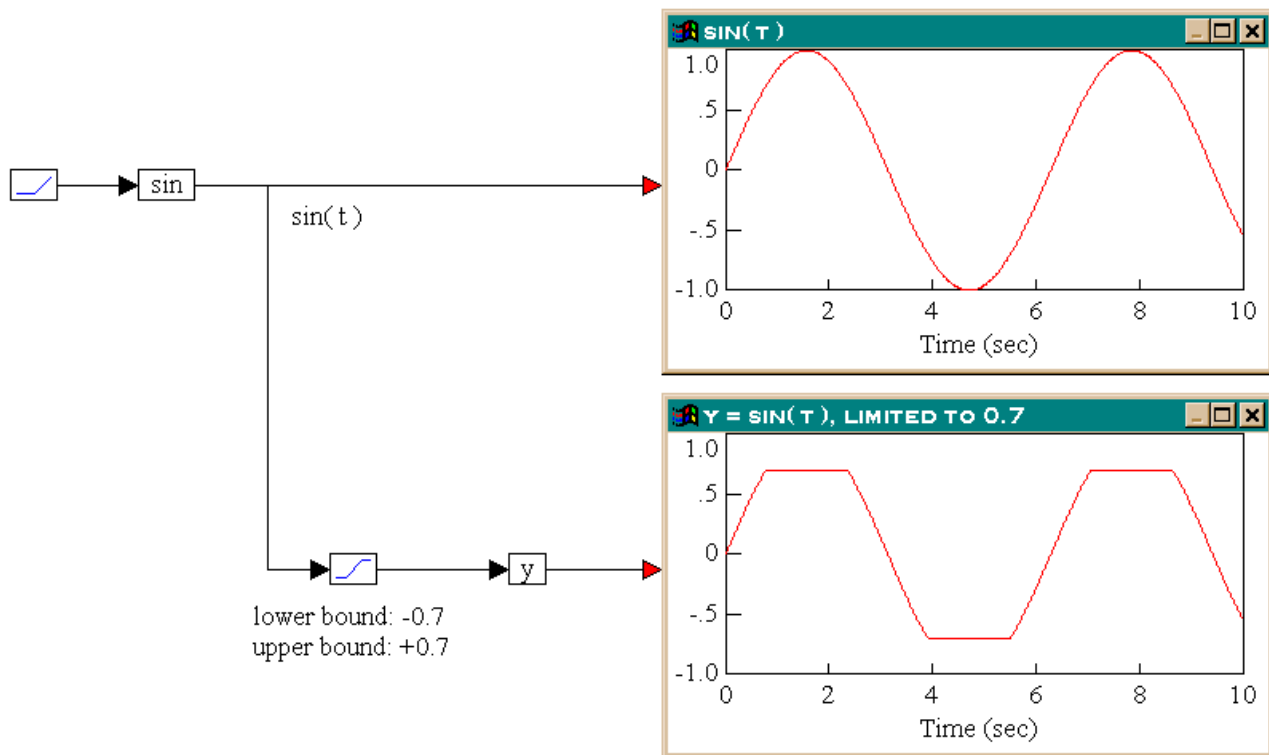
Examples

1. Simulation of saturation

Consider a variable y such that:

$$y = \sin(t)$$

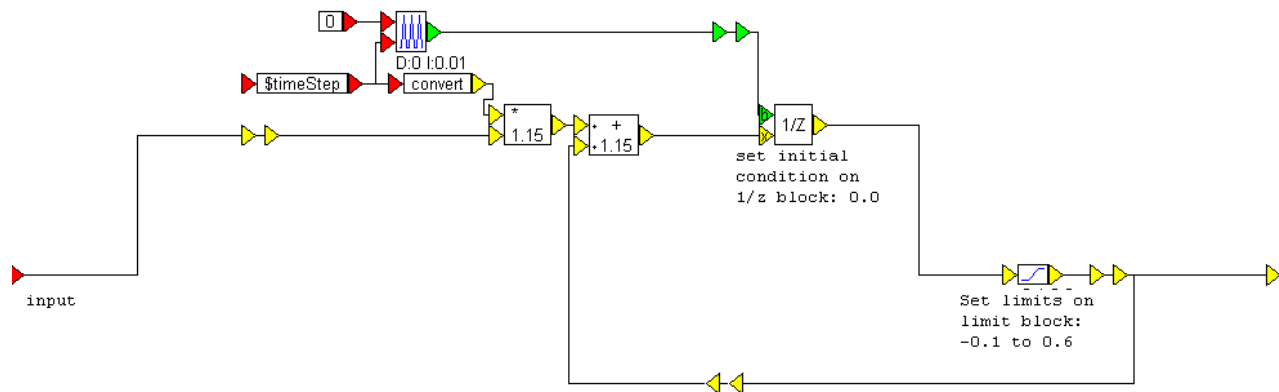
Furthermore, assume that y reaches saturation at $+0.7$ and -0.7 . This equation can be realized as shown below.



From the results in the two plot blocks, the output of the limit block is identical to the input, when the input is within the bounds (-0.7 to +0.7). When the input is out of these bounds, the output is limited to the upper or lower bound values.

limitedIntegrator (1/S)

To see the implementation of the limitedIntegrator block, right click the block.



The high and low limits are displayed on the limit block. You can set the limits in the limit Properties dialog box.

The initial condition for the limitedIntegrator block is set in the [unitDelay](#) block.

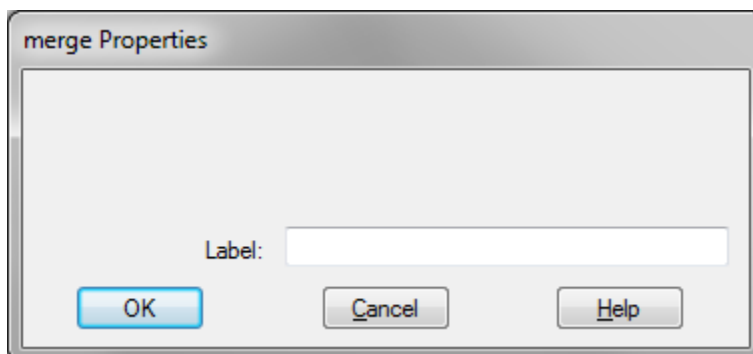
merge

$$y = \begin{cases} x_2 & \text{if } |x_1| \geq 1 \\ x_3 & \text{if } |x_1| < 1 \end{cases}$$

Block Inputs: Scalar, vector, and matrix input.

The merge block examines x_1 (Boolean signal) to determine the output signal. The letters b, t, and f on the input connectors stand for Boolean, True, and False.

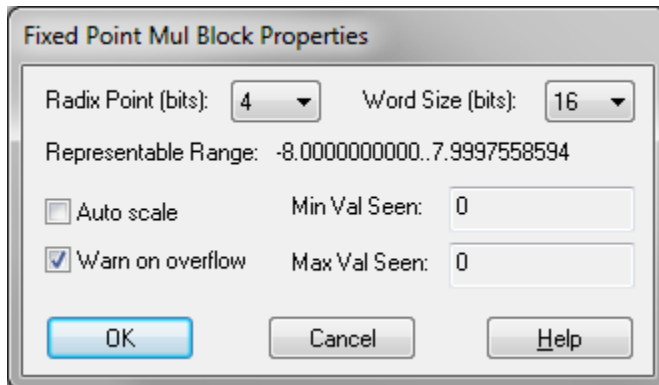
The merge block is particularly well-suited for performing if-then-else decisions.



Label: Indicates a user-defined block label.

mul

The mul block produces the product of the input signals in scaled fixed notation.



Auto Scale: Rescales the range when either the maximum or minimum value is exceeded.

Max Val Seen: Displays the high watermark of values that come through the block. This is a read-only field.

Min Val Seen: Displays the low watermark of values that come through the block. This is a read-only field.

Radix Point (bits): Sets the binary point.

Representable Range: Indicates the range of values based on the selected radix point. This is a read-only field.

Warn on Overflow: Activates a warning dialog box that appears when an overflow error occurs. An overflow error occurs when a result value is outside the currently acceptable range. You must also activate the Enable Overflow Alert Messages in the dialog box for the [Fixed Point Block Set Configure](#) command under the Tools menu.

Word Size (bits): Sets the word size for the target architecture. The word size can be overridden using the Override Word Size option in the dialog box for the [Fixed Point Block Set Configure](#) command under the Tools menu.

not

$$y = \begin{cases} 1 & \text{if } x_1 = 0 \\ 0 & \text{otherwise} \end{cases}$$

The not block produces the Boolean NOT of the input signal. The output is true when the input is false; and the output is false when the input is true.

If you right click the not block, the Boolean block menu appears allowing you to assign a different function to the block.

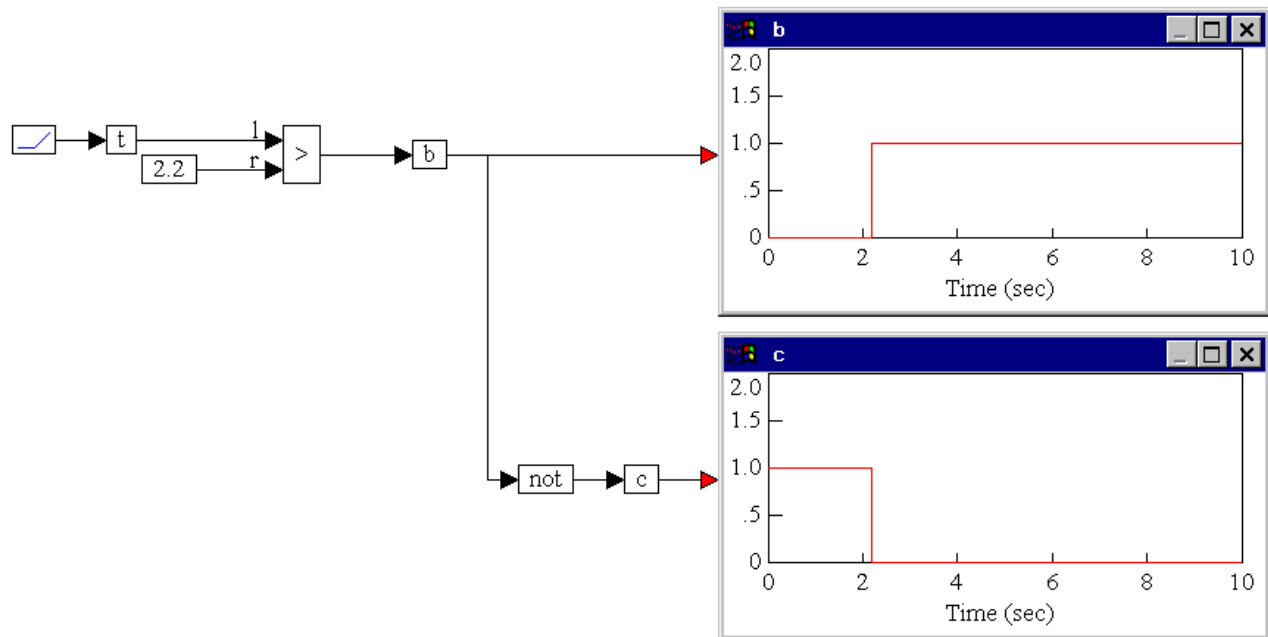
Examples

1. Using a not block

Consider a variable c such that:

$$c = \bar{b}$$

or in other words, $c = \text{not}(b)$. Furthermore, assume that b is true if $t > 2.2$; else b is false, where t is the simulation time. This system can be realized as shown below.



From the outputs obtained in the two plot blocks, b , given by the output of the $>$ block is true only when t is > 2.2 . This requires that c , which is defined to be not (b), be true only the range $t < 2.2$, as obtained in the bottom plot block.

or

$$y = x_1 \text{ bitwise OR } x_2$$

The or block produces the bitwise OR of two to 256 scalar input signals. The output of the or block is true when at least one of the inputs is true. When all the inputs are false, the output is false.

If you right click the or block, the Boolean block menu appears allowing you to assign a different function to the block.

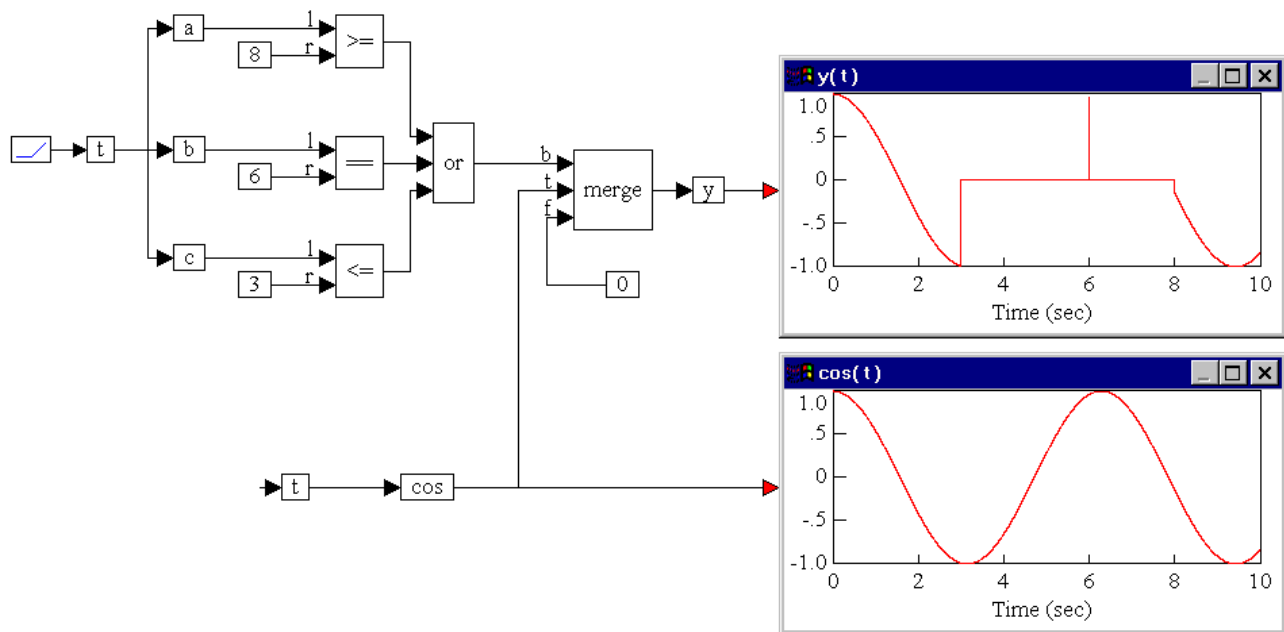
Examples

1. Computation of three inputs

Consider a variable y such that:

$$\text{If } a \geq 8 \text{ or } b = 6 \text{ or } c \leq 3, \text{ then } y = \cos(t); \text{ else } y = 0$$

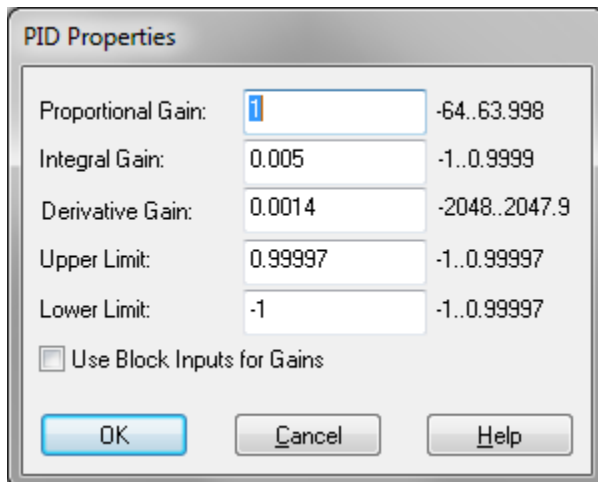
where t is simulation time. Furthermore, let t be the input to all three parameters a , b , and c . This system can be realized as shown below.



During simulation, the or block evaluates to false in the interval $t = (3,8)$, except for the instant $t = 6$. In this case, the variable y takes on the value of zero. The output of or evaluates to true in the remaining parts of the simulation, and as a result, y takes on the value of $\cos(t)$ in these periods, including the instant $t = 6$.

PID Regulator

For implementation, see the Texas Instruments [Digital Motor Control](#) document.



Integral Gain* dt : Indicates the integral gain pre-multiplied by the system dt . For example, for $K_i = 0.5$ and $dt = 0.01$, the coefficient is 0.005.

Lower Limit: Indicates the lower limit of PI output.

Proportional Gain: Indicates the proportional gain.

Upper Limit: Limits internal integrator and PI output.

Use Block Inputs for Gains: Adds additional inputs for the P (proportional) and I (integral) gains.

PI Regulator

For implementation, see the Texas Instruments [Digital Motor Control](#) document.

Property	Value	Range
Proportional Gain:	1	-64..63.998
Integral Gain*dT:	0.005	-1..0.9999
Upper Limit:	0.99997	-1..0.99997
Lower Limit:	-1	-1..0.99997

Use Block Inputs for Gains

OK Cancel Help

Integral Gain*dT: Indicates the integral gain pre-multiplied by the system dt . For example, for $K_i = 0.5$ and $dt = 0.01$, the coefficient is 0.005.

Lower Limit: Indicates the lower limit of PI output.

Proportional Gain: Indicates the proportional gain.

Upper Limit: Limits internal integrator and PI output.

Use Block Inputs for Gains: Adds additional inputs for the P (proportional) and I (integral) gains.

sampleHold

$$y = \begin{cases} x_2 & \text{if } |x_1| \leq 1 \\ y_{previous} & \text{otherwise} \end{cases}$$

The sampleHold block latches an input value under the control of a clock signal, x_1 , which is represented as Boolean input b . When b is true, input signal x_2 , which is represented as input x , is sampled and held until b is true again. Boolean inputs can be regularly or irregularly spaced.

sampleHold Properties

Initial Condition: 0

Label:

OK Cancel Help

Initial Condition: Indicates the initial condition for the sampleHold. The default is zero.

Label: Indicates a user-defined block label.

Examples

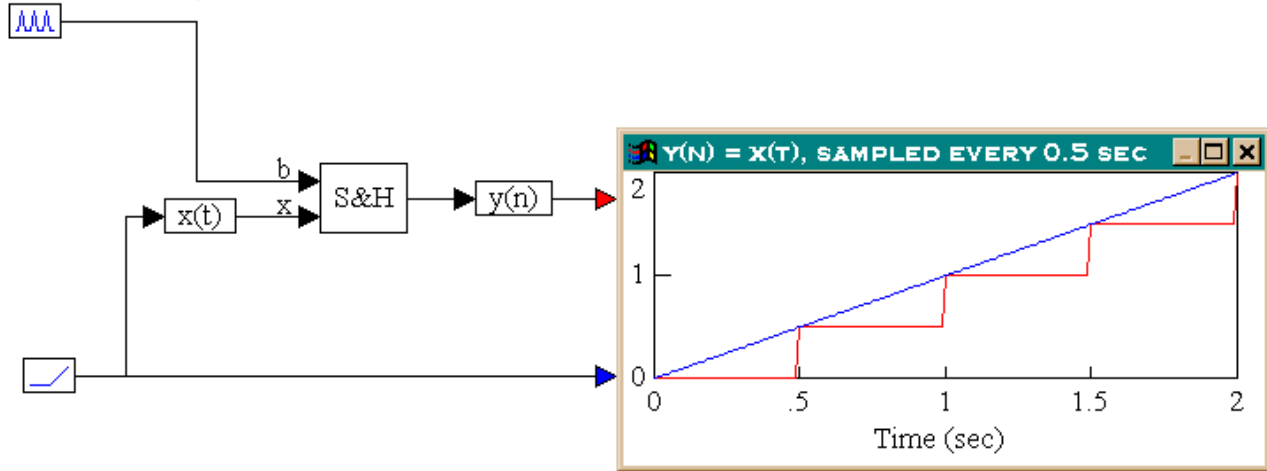
1. Sample and hold with regularly-spaced clock

Consider the equation:

$$y(n) = x(t)$$

sampled every 0.5s. Furthermore, let $x(t)$ be a ramp signal. This system can be realized as shown below.

pulseTrain with
a time between pulses of 0.5



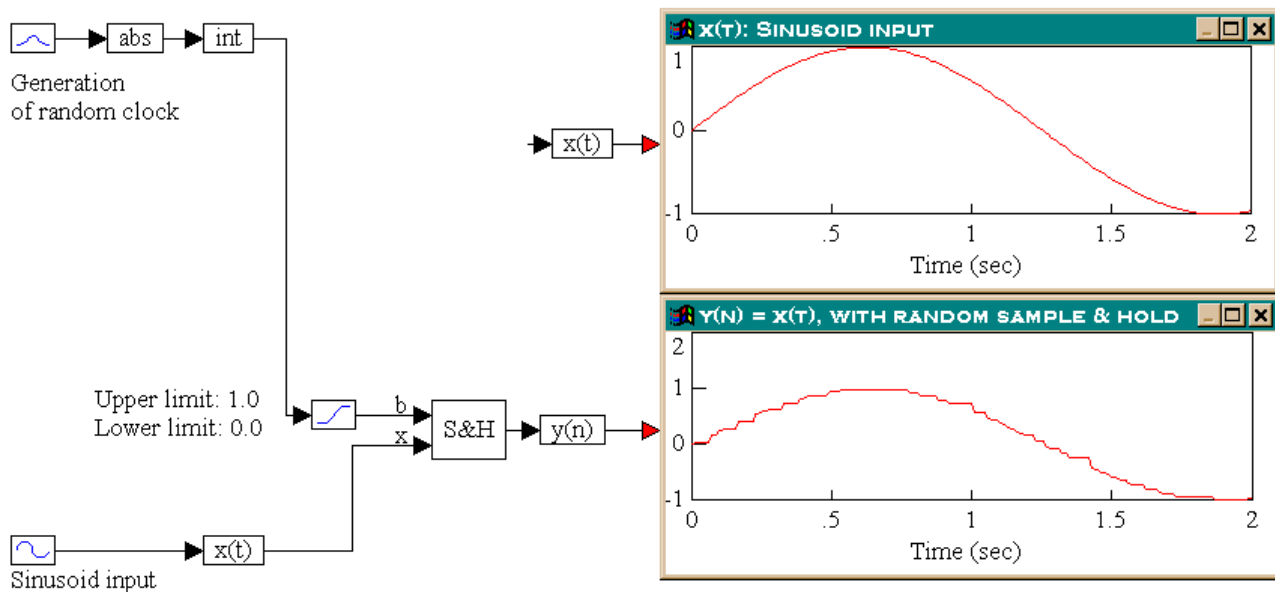
As seen in the plot block, the first clock pulse occurs at 0.5s. Until this time, the output of the sampleHold block is zero. At 0.5s, the input signal is sampled and the value is used as output for the sampleHold block. The output of the sampleHold block is held at this value until the occurrence of the next clock pulse at 1.0s. At this time, the input signal is again sampled and the new value is presented to the output of the sampleHold block, and the process repeats itself.

2. Sample and hold with irregularly-spaced clock

Consider the equation:

$$y(n) = x(t)$$

sampled randomly. Furthermore, let $x(t)$ be a sinusoid signal with a frequency of 2.5 rad/s. This system can be realized as shown below.



A sinusoid block with a frequency of 2.5 rad/s generates the sinusoid signal and a gaussian block produces a randomly varying signal. The randomly varying signal is converted to a random clock by taking the absolute value of the random

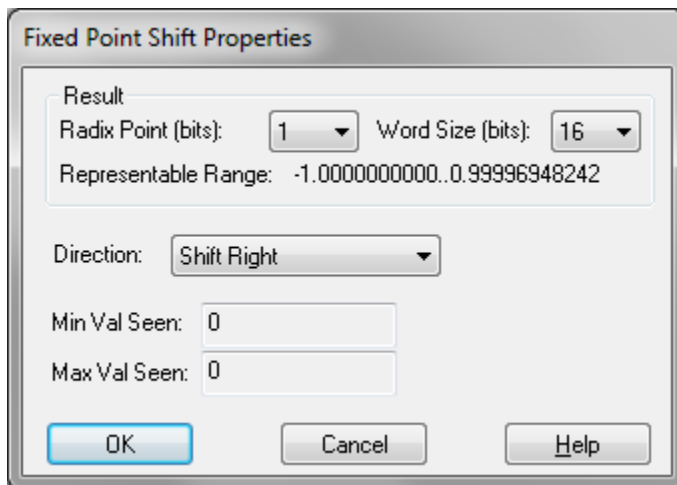
signal and then using only the integer portion of it. The output of the int block is passed through a limit block to restrict the signal to the range (0, 1). The output of the limit block is connected to the top input of the sampleHold block. The output of the sampleHold block is connected to the variable $y(n)$, which is connected to a plot block. The actual input, $x(t)$ is monitored separately in another plot block.

By comparing the outputs in the two plot blocks, the output of the sampleHold block is a randomly sampled and held version of the input sinusoid.

shift

The shift block shifts the input value x by shift dist bits.

Result scaling is set in the dialog box and is independent of the input scaling.



Auto Scale: Rescales the range when either the maximum or minimum value is exceeded.

Direction: Selects whether the shift is left (multiply by 2) or right (divide by 2).

Max Val Seen: Displays the high watermark of values that come through the block. This is a read-only field.

Min Val Seen: Displays the low watermark of values that come through the block. This is a read-only field.

Result

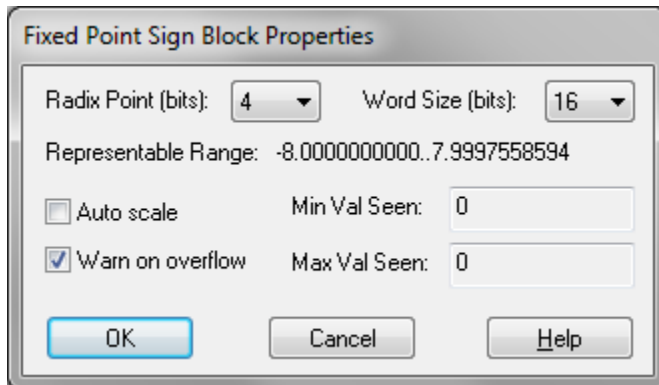
Radix Point (bits): Sets the binary point.

Representable Range: Indicates the range of values based on the selected radix point. This is a read-only field.

Word Size (bits): Sets the word size for the target architecture. The word size can be overridden using the Override Word Size option in the dialog box for the [Fixed Point Block Set Configure](#) command under the Tools menu.

sign

The sign block determines the sign of the input signal in scaled fixed notation.



Auto Scale: Rescales the range when either the maximum or minimum value is exceeded.

Max Val Seen: Displays the high watermark of values that come through the block. This is a read-only field.

Min Val Seen: Displays the low watermark of values that come through the block. This is a read-only field.

Radix Point (bits): Sets the binary point.

Representable Range: Indicates the range of values based on the selected radix point. This is a read-only field.

Warn on Overflow: Activates a warning dialog box that appears when an overflow error occurs. An overflow error occurs when a result value is outside the currently acceptable range. You must also activate the **Enable Overflow Alert Messages** in the dialog box for the [Fixed Point Block Set Configure](#) command under the Tools menu.

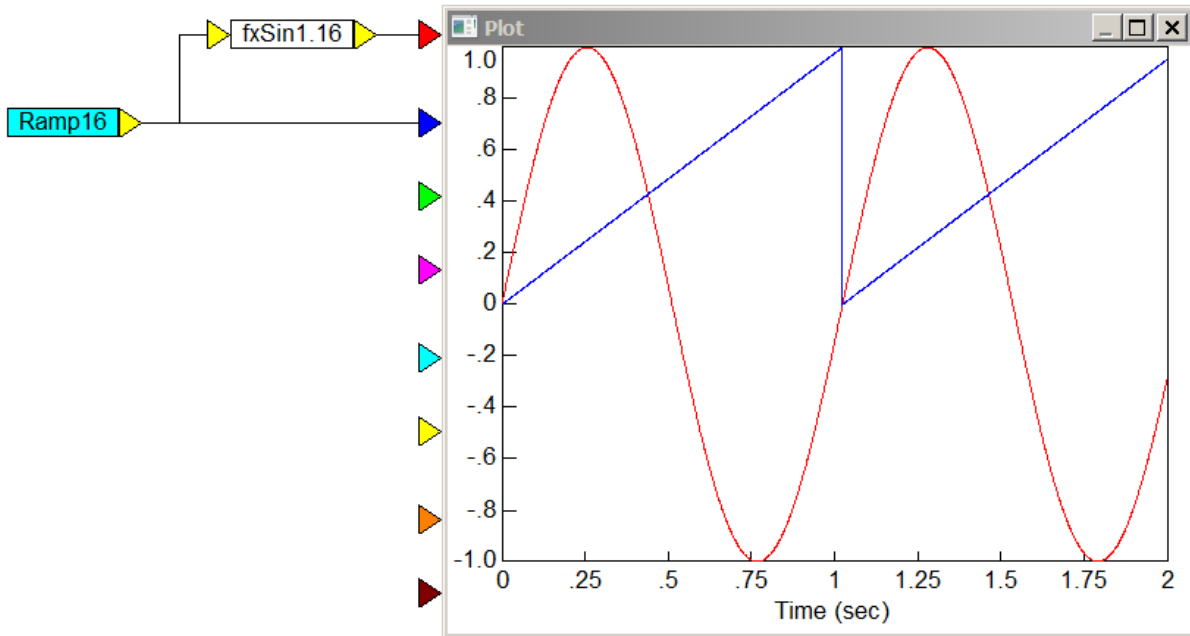
Word Size (bits): Sets the word size for the target architecture. The word size can be overridden using the **Override Word Size** option in the dialog box for the [Fixed Point Block Set Configure](#) command under the Tools menu.

sin

Block Inputs: fx1.16 or fx1.32 format.

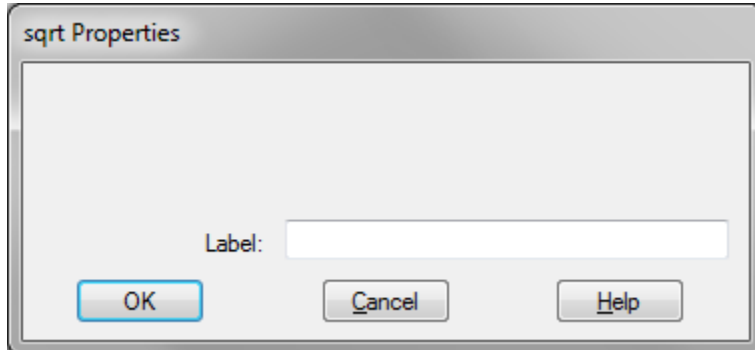
The sin block produces the sine of the input signal in scaled fixed-point notation. The block automatically uses fx1.16 or fx1.32 based on the data type of the input signal. The input signal is represented in “per unit” angles. This means the angle input is implicitly multiplied by 2π so that a value sweeping from zero to one will produce a complete cycle of sin output. Typically, this block is fed by a repeating fixed-point ramp that runs from zero to one.

Example



sqrt

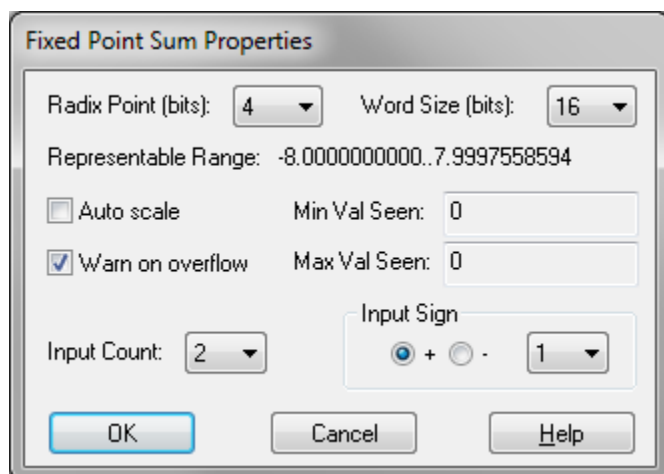
The sqrt block produces an output signal that is the square root of a positive input signal. The sqrt block does not accept negative inputs, and there is no square root of 0.



Label: Indicates a user-defined block label.

sum

The sum block adds two input signals in scaled fixed notation and produces an output.



Auto Scale: Rescales the range when either the maximum or minimum value is exceeded.

Const: Indicates the value of the output signal.

Input Count: Sets the number of inputs for the block.

Input Sign: Sets the sign of each input. To set the sign, select the input number from the drop-down list, then choose either the “-” or “+” radio button. You can display the sign on the block by activating the View > Connector Labels command.

Min Val Seen: Displays the low watermark of values that come through the block. This is a read-only field.

Max Val Seen: Displays the high watermark of values that come through the block. This is a read-only field.

Radix Point (bits): Sets the binary point.

Representable Range: Indicates the range of values based on the selected radix point. This is a read-only field.

Warn on Overflow: Activates a warning dialog box that appears when an overflow error occurs. An overflow error occurs when a result value is outside the currently acceptable range. You must also activate **Enable Overflow Alert Messages** in the dialog box for the [Fixed Point Block Set Configure](#) command under the Tools menu.

Word Size (bits): Sets the word size for the target architecture. The word size can be overridden using **Override Word Size** in the dialog box for the [Fixed Point Block Set Configure](#) command under the Tools menu.

transferFunction

$$y = \frac{a_n s^n + a_{n-1} s^{n-1} \dots a_1 s + a_0}{b_n s^n + b_{n-1} s^{n-1} \dots b_1 s + b_0} x$$

The transferFunction block executes a single-input single-output linear transfer function specified in the following ways:

- **As an M file created with Embed:** The Linearize command in the Analyze menu generates ABCD state-space matrices from the nonlinear system by numerically evaluating the matrix perturbation equations at the time the simulation was halted.
- **As an M file created with a text editor:** The following is an example of a user-written M file:

```
function [a,b,c,d] =vabcd
```

```
a = [-.396175 -1.17336 ; 5.39707 .145023 ];
```

```
b = [-.331182 ; -1.08363 ];
```

```
c = [0 1 ];
```

$d = [0];$

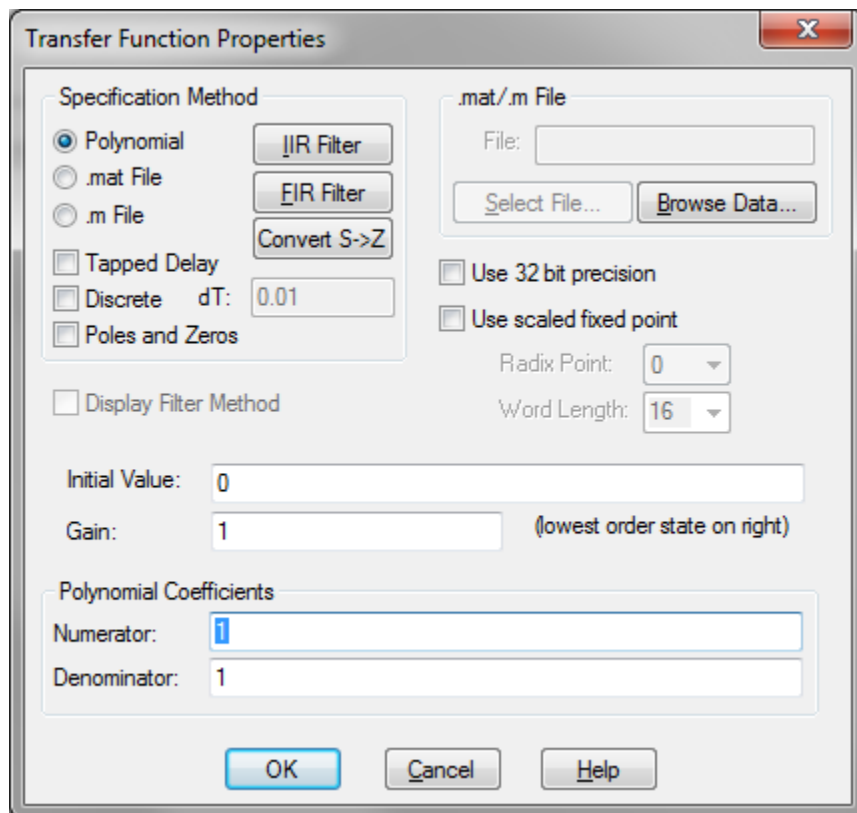
- **As a MAT file created with MATLAB:** Generating MAT files is described in the MATLAB documentation. Note that when you save the ABCD matrices to file, the names of the matrices are not important; however, the order in which they appear is.

When you simulate the block diagram, Embed numerically solves the transferFunction block.

You can set initial conditions for transferFunction blocks using variables. You can also reset transferFunction blocks to zeros using the System > Reset States command.

Digital filter design: The transferFunction block supports IIR and FIR digital filter design.

Setting up a transfer function: The transferFunction block's Properties dialog box allows you to control how the numerator and denominator polynomials are entered.



Display Filter Method: Displays the filter specification on the block. When Display Filter Method is not activated, Embed displays the polynomial coefficients.

Gain: Indicates the transfer function gain. If the leading terms of the numerator and denominator coefficients are not unity, Embed will adjust the gain to make it so. The default value is 1.

Initial Value: Specifies initial values for the states in the block. The values are right-adjusted. The right-most value corresponds to the lowest order state. Unspecified states are set to zero.

.mat/.m File: Indicates the name of the M or MAT file to be used as input to the transferFunction block. You can type the file name directly into this box or select one using the Select File button.

Polynomial Coefficients

Denominator: Indicates the denominator polynomial for the transferFunction block. Embed determines the order of the transfer function by the number of denominator coefficients you enter. For example, an nth order transfer function will have $n + 1$ coefficients. Separate coefficients with spaces.

Numerator: Indicates the numerator polynomial for the transferFunction block. Separate coefficients with spaces.

Radix Point: Sets the binary point. This option can be used only when **Use Scaled Fixed Point** is activated.

Specification Method

Discrete: Indicates a discrete Z-Domain transfer function. Enter the time step for the discrete transfer function in the dT box. By default, Embed uses the step size established with the System > System Properties command. When **Discrete** is de-activated, a continuous transfer function is created.

dT: Specifies the time step for the discrete transfer function. By default, Embed uses step size parameter from the System > System Properties command.

.mat File: Indicates that the transfer function is to be specified as a MAT file. Specify the name of the MAT file in the .mat/.m File group box.

.m File: Indicates that the transfer function is to be specified as an M file. Specify the name of the M file in the .mat/.m File group box.

Poles and Zeros: Lets you enter a transfer function via zeros and poles in the **Numerator** and **Denominator** boxes, respectively. Enter each root in the following format:

(real-part, imaginary-part)

For large order systems, poles and zeros is more numerically accurate.

Polynomial Coefficient: Indicates that the transfer function is to be specified as numerator and denominator polynomials. Supply the numerator and denominator polynomials and gain under the Polynomial Coefficients group box.

Tapped Delay: Provides tapped delay implementation for high order FIR filters.

IIR Filter button: Opens the IIR Filters Setup dialog box to design a suitable filter using analog prototypes.

FIR Filter button: Opens the FIR Filter Setup dialog box to construct Regular Finite Impulse Response filters, differentiators, and Hilbert Transformers.

Convert S ->Z button: Uses bilinear transformation to convert a continuous transfer function to an equivalent discrete transfer function with a sampling interval of dT . Embed requests a discrete sampling rate prior to performing the conversion.

An example of the conversion is shown below.

$$H(s) = \frac{a}{s + a}$$

The bilinear transformation can be implemented by the substitution:

$$\frac{2}{dT} \frac{z - 1}{z + 1} \rightarrow s$$

The above transfer function becomes:

$$H_{dT}(z) = \frac{a}{\left(\frac{2}{dT}\right) \left\{ \frac{z - 1}{z + 1} \right\} + a}$$

Embed automatically simplifies this representation and enters the appropriate coefficients for the numerator and denominator polynomials.

Convert Z ->S button: Uses bilinear transformation to convert a discrete transfer function to an equivalent continuous transfer function. For example, consider:

$$H_{dT}(z) = \frac{z}{z + b}$$

The bilinear transformation can be implemented by the substitution:

$$\frac{2 + dTs}{2 - dTs} \rightarrow z$$

The above discrete transfer function becomes:

$$H_{dT}(s) = \frac{2 + dT \cdot s}{(2 + 2b) + (dT - b \cdot dT)s}$$

Embed automatically simplifies this representation and enters the appropriate coefficients for the numerator and denominator polynomials.

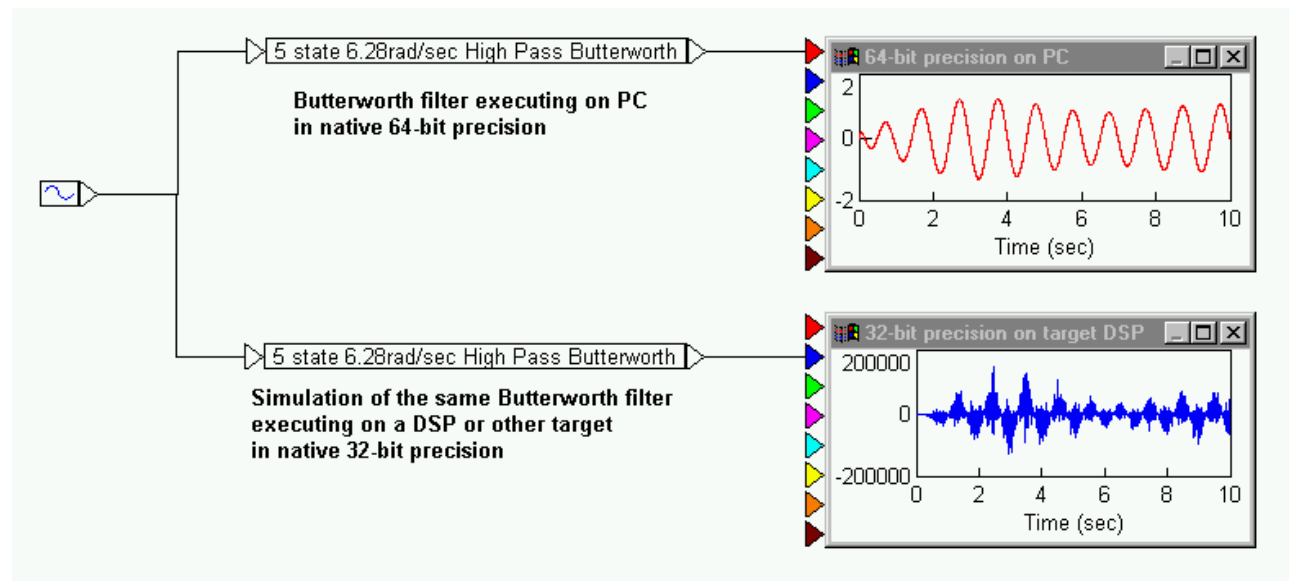
It is important to note that in both transformations, the results obtained are dependent on the sampling interval dT . In other words, for a given continuous or discrete transfer function, an infinite number of equivalent discrete or continuous transfer functions may be obtained by varying the sampling interval dT .

Use 32 Bit Precision: Simulates the behavior of the transfer function at 32-bit precision. Normally, all Embed blocks are 64-bit precision. This parameter allows you to simulate the effect of code running on a floating-point 32-bit target.

Used Scaled Fixed Point: When targeting a fixed-point processor, such as a TI-C2000, activate this option. Embed uses scaled fixed point internal operations and will generate scaled fixed-point code.

Word Length: Sets the word size for the target architecture. This option can be used only when Used Scaled Fixed Point is activated. Note that the word size can be overridden using the Override Word Size option in the dialog box for the [Fixed Point Block Set Configure](#) command under the Tools menu.

Examples



This example simulates a Butterworth filter in reduced precision to simulate the effects of implementing the system on a reduced-precision 32-bit floating-point target. To turn on reduced precision, activate the **32-Bit Precision** parameter transferFunction Properties dialog box.

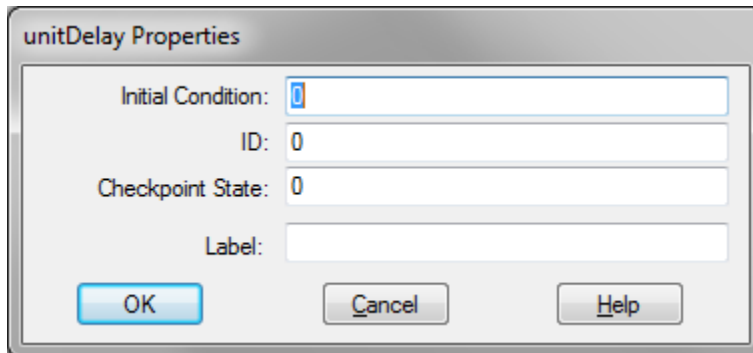
unitDelay

$$y = \begin{cases} y_{buffer}, y_{buffer} = x_2 \text{ if } |x_1| \geq 1 \\ y_{previous} & \text{otherwise} \end{cases}$$

The `unitDelay` block specifies a clocked unit delay. The input connectors are marked `b` (for Boolean clock) and `x` (for main signal). When the Boolean clock does not equal zero, the value contained in the single element buffer is copied to the block output (where it holds this value until the next non-zero Boolean clock). The current value of the main signal is stored in the unit buffer.

The `unitDelay` block is intended for modeling a digital delay in a continuous simulation. A typical digital delay is modeled by wiring a `pulseTrain` block to the Boolean input connector of the `unitDelay` block. Use the `timeDelay` block to model a continuous delay.

You can set the initial conditions for `unitDelay` blocks with variables. You can also reset `unitDelay` blocks to zeros using the `System > Reset States` command.



Checkpoint State: Contains the value of the unit delay at the checkpoint. If you have not checkpointed your simulation via the `System > System Properties` command, the value is zero.

ID: Reserved for future use.

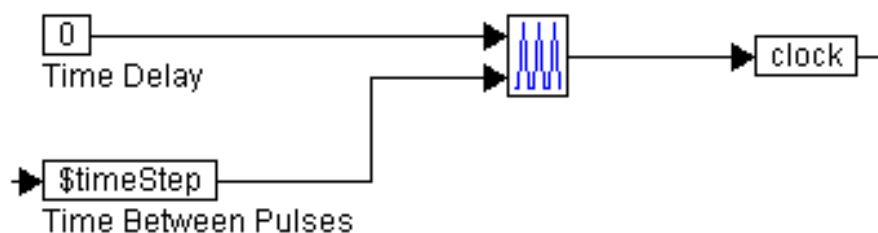
Initial Condition: Sets an initial value for the output signal. The default is zero.

Label: Indicates a user-defined block label.

Examples

1. Clocking the `unitDelay` block

If you are working with `unitDelay` blocks, it is good programming practice to create a clock signal that you can use in every simulation. A typical clock signal can be generated as shown below.



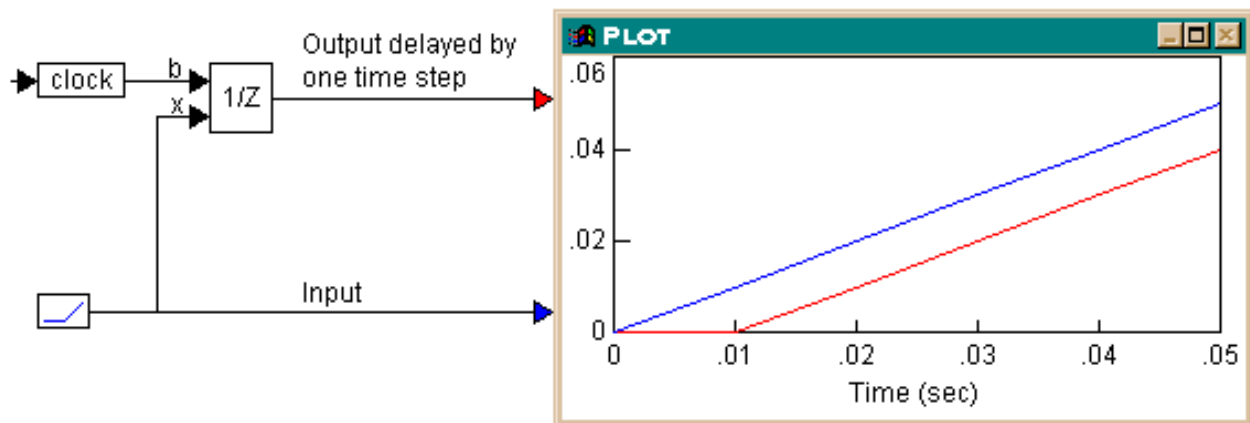
Here, a `pulseTrain` block is assigned two external inputs:

- The top input is the time delay for the `pulseTrain` block. The time delay value for the `pulseTrain` block is the amount of time the `pulseTrain` block waits before producing pulses. This time delay value must not be confused with the amount of time delay generated by the `unitDelay` block.
- The bottom input is the time between pulses.

The output of the `pulseTrain` block is fed to the variable `clock`. This variable can be used anywhere in the simulation to clock `unitDelay` blocks.

2. Introduction of a one-step delay

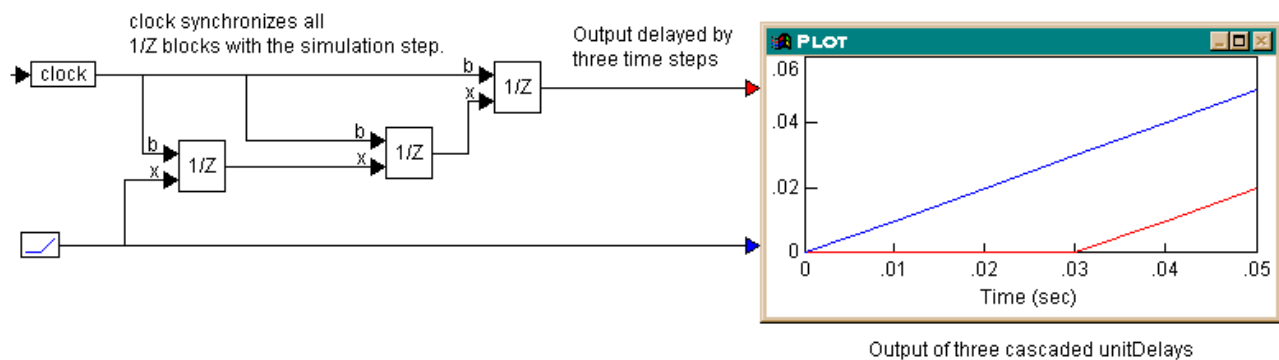
For a given signal, a one-step delay can be introduced as:



During simulation, the actual and delayed signals are plotted in the plot block. The output of the unitDelay block is delayed by one step (equal to 0.01 in this case) as compared to the input.

3. Using a multi-step delay with cascaded unitDelay blocks

To achieve multi-step delays, unitDelay blocks that implement one-step delays, can be cascaded. Consider the example where a three-step delay is introduced.



Three unitDelay blocks, all clocked at the simulation step, are cascaded. Since each unitDelay introduces a one-step delay between its input and output, the output of the third unitDelay block is delayed by three steps compared to the input. The plot block shows this behavior, with a simulation step size of 0.01.

XOR

$$y = x_1 \text{ bitwise XOR } x_2$$

Block Inputs: Scalar.

The xor block produces the bitwise exclusive OR of 2 – 256 scalar input signals.

If you right click the xor block, the Boolean block menu appears allowing you to assign a different function to the block.

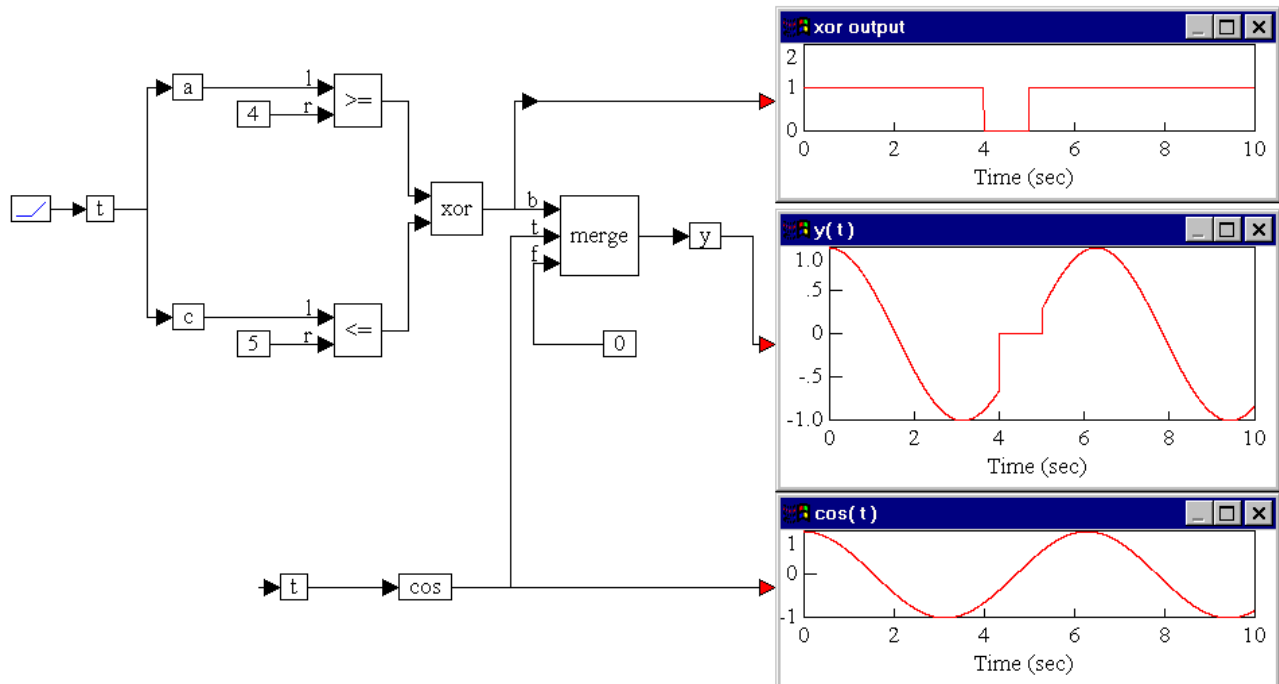
Examples

1. Using the xor block

Consider a variable y such that:

If $a \geq 4$ or $c \leq 5$, then $y = \cos(t)$; else $y = 0$. Also, if $a \geq 4$ and $c \leq 5$, $y = 0$

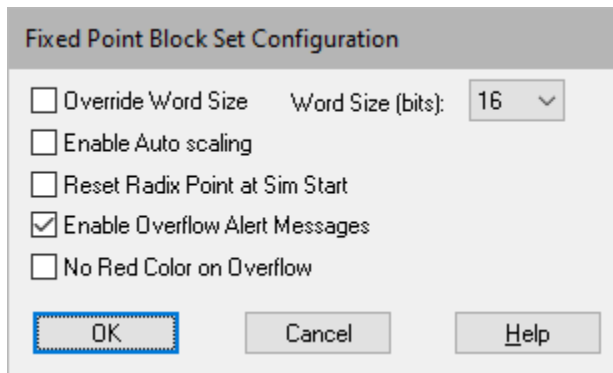
where t is simulation time. Furthermore, let t be the input to parameters a and c . This system can be realized as shown below.



As shown in the two plot blocks, the output of the xor block evaluates to false in the interval $t = (4, 5)$, since both the inputs to the xor block are true in this interval. Consequently, y takes on the value of zero. The output of xor evaluates to true in the remaining parts of the simulation, and as a result, y takes on the value of $\cos(t)$ in these periods.

Fixed Point Block Set Configure command

The Fixed Point Block Set Configure command lets you set options on all Fixed Point blocks. This command is located under the Tools menu.



Enable Autoscaling: Turns on autoscaling for the Fixed Point blocks whose **Autoscale** option is activated.

Enable Overflow Alert Messages: Turns on overflow alert message for the Fixed Point blocks whose **Autoscale** option is activated.

No Red Color on Overflow: Turns off coloring blocks red when overflow occurs.

Override Word Size: Overrides the local word size on all Fixed Point blocks using size specified under **Word Size**.

Reset Radix Point at Sim Start: Resets the radix point on all Fixed Point blocks to zero at the start of the simulation. Typically, this option comes in handy if an algorithm starts to go unstable and the radix point is maxed out. By resetting the radix point to zero, you have an opportunity to fix the algorithm.

Note that this option is available only when **Enable Autoscaling** is activated.

Word Size (bits): Sets the word size for all Fixed Point blocks for the target architecture. This size is in effect only when **Override Word Size** is activated.

Tutorials

The tutorials in this section describe how to convert floating-point implementation to fixed-point implementation. The Learning Center provides several Fixed Point videos to help you get started using the Fixed Point block set.

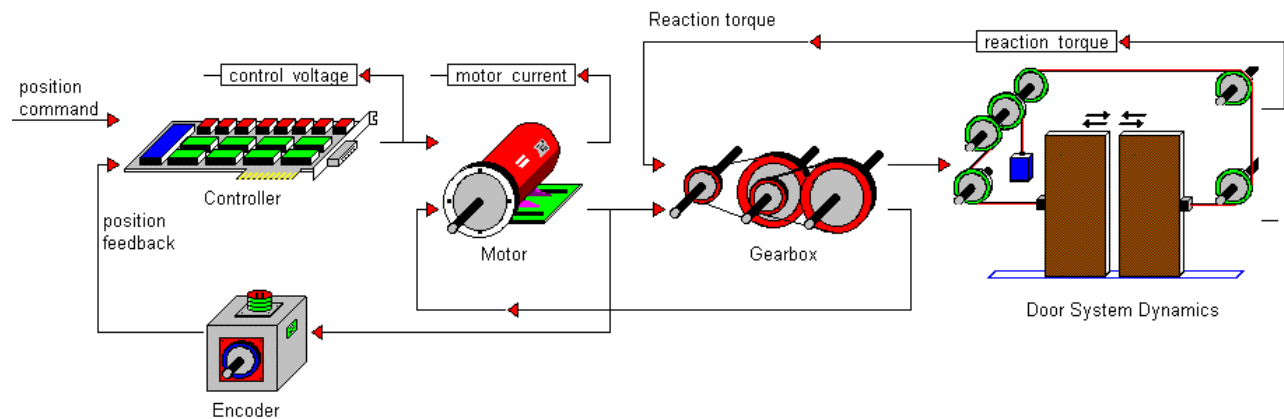
Implementing an elevator door control system

Floating-Point Diagram: `otis_elevator_regular`

Fixed-Point Diagram: `otis_elevator_fixed_point`

Location: Examples > Fixed Point

A simplified elevator door control system — shown below — consists of a DC motor driving a gearbox that in turn manipulates the door position through a series of pulleys. The controller accepts open and close commands as inputs, and controls the magnitude and polarity of voltage that is applied to the DC motor. An encoder provides motor rotor shaft position feedback to the control system.



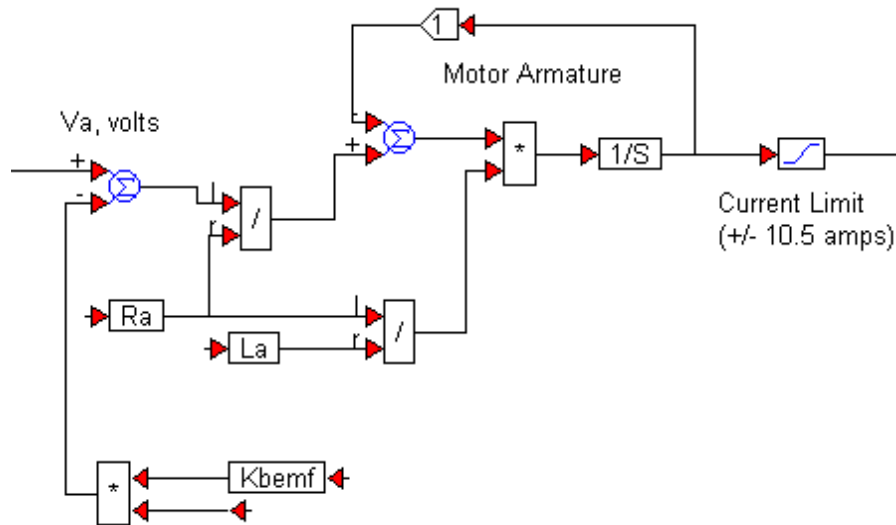
This tutorial describes how to implement an elevator door control system in floating point, then convert the controller to scaled, fixed point.

Floating-point implementation

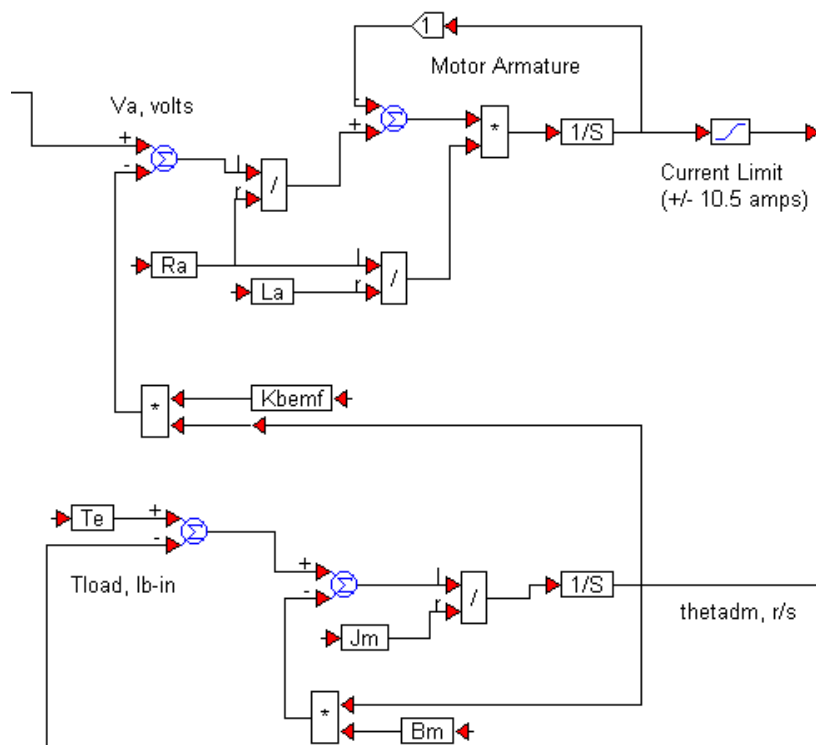
To see the floating-point implementation of the elevator door control system, go to Examples > Fixed Point and open `otis_elevator_regular`.

Constructing the motor

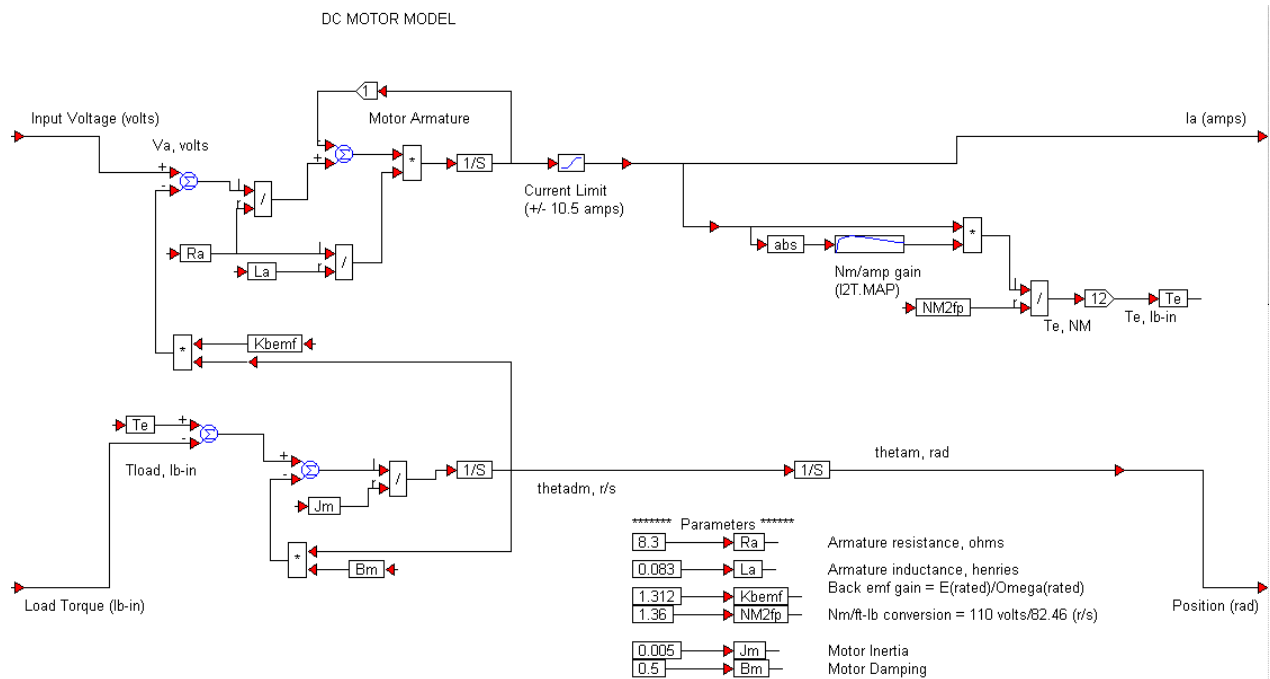
To model the DC motor, the effective voltage is the difference between the applied voltage and back-emf. The motor armature is modeled as a simple first-order system with resistance R_a and inductance L_a . The motor current is limited to $\pm 10.5A$.



To compute the back-emf, the back-emf gain K_{bemf} is multiplied by the angular velocity. Angular velocity is computed using the electrical torque and load torque.



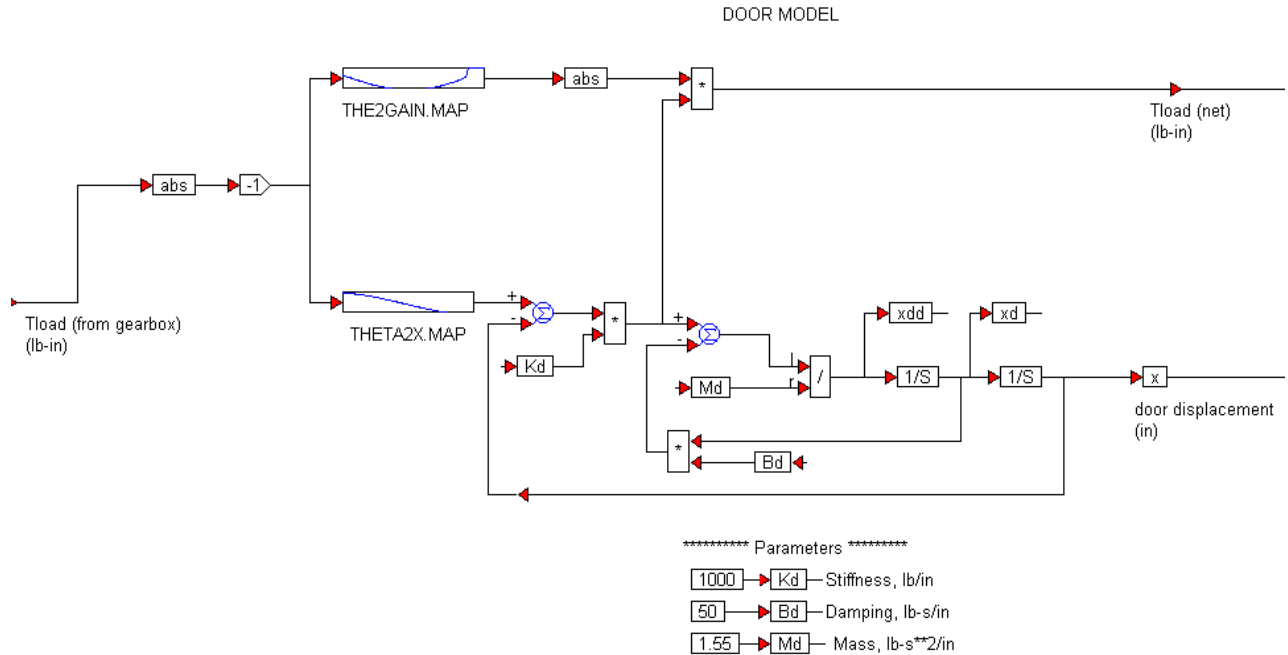
A set of const blocks defines the required motor parameters. The motor angular velocity θ_{adm} is integrated to yield position. The electrical torque T_e is computed using the motor armature current, and the I2T.MAP look-up table contains the motor's current-torque characteristics. This data is obtained from the motor's specification sheets or through the vendor. The complete motor model is shown below.



Note: The eMotor toolbox includes a full set of pre-configured, pre-tested, and ready-to-use blocks, such as motors, amplifiers, loads, sensors, and controllers.

Constructing the gearbox

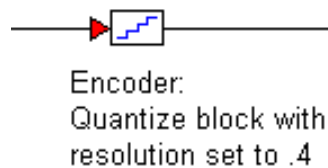
In addition to increasing or decreasing the number of output revolutions relative to the input revolutions, a typical gearbox also introduces additional inertia, stiffness, and damping effects into the system. A basic rotational load model is implemented below. Detailed rotational and translational models are included in the eMotor toolbox. The completed gearbox model can be realized as shown below.



Two look-up tables – THE2GAIN.MAP and THETA2X.MAP – are used for easy modeling of the dependency of the load torque aspects and the relationship between the angular gear-shaft motion and the linear motion of the door-assembly.

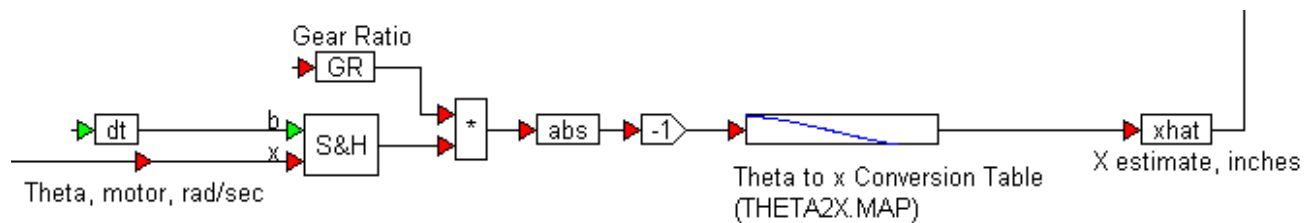
Constructing the encoder

A basic encoder can be modeled simply as a quantizer using the quantize block with the resolution set to 0.4.



Constructing the controller

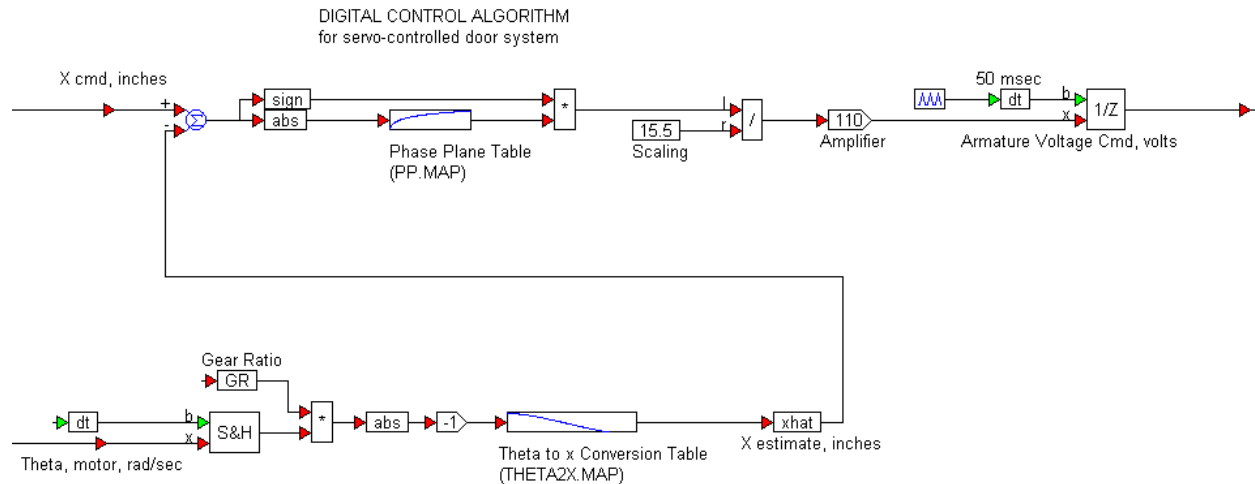
The controller takes two inputs: the open/close command and the actual position of the gear-shaft as estimated by the encoder. The encoder feedback is converted into inches, the same units as the command input using simple arithmetic, and the look-up table THETA2X.MAP gives the relationship between angular gear-shaft position and equivalent door-assembly linear displacement as previously seen. The conversion logic is as shown below.



The error is calculated by subtracting the estimated actual door displacement from the commanded displacement. While the absolute value of the error determines the amplitude of the control voltage to be applied, the output of the sign block is used to determine the polarity of the voltage to be applied (that is, whether the door is to be opened or closed).

Another look-up table (PP.MAP) determines the recommended control voltage ratio. The recommended control voltage is converted into a ratio by scaling it with the maximum value from the table (15.5) and fed to a simple proportional control stage represented by a gain of 110. The output of the proportional stage is sampled at 50ms to represent the physical realities of implementing the control logic on a digital target such as a DSP or a microcontroller.

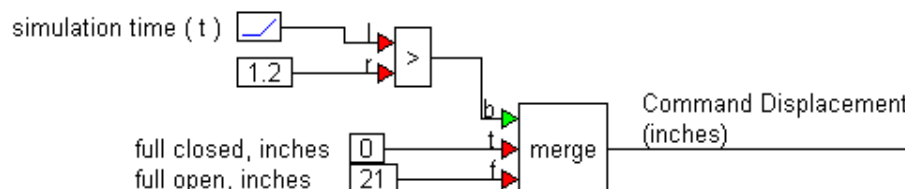
The complete controller structure is as shown below.



Constructing the open/close command

To test the system, an elevator door open-close cycle is constructed. A typical cycle is to open for 1.2s followed with a close command. An open command means the desired door displacement is 21 inches, while the close command implies that the desired door displacement is zero inches.

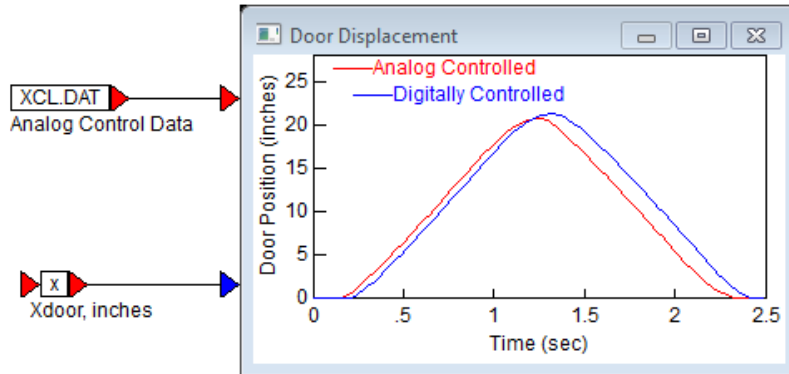
Using a ramp block to access the current value of simulated time (t) and using if-then-else logic with a merge block, you can implement the open/close cycle as shown below.



After connecting all the subsystems and assigning variables for monitoring (*control_voltage*, *motor_current* and *reaction_torque*), the system is complete.

Fixed-point implementation of the controller

When calibrating a new control design, a common way to validate a new design is to compare its performance with an existing floating-point implementation. In the **elevator_door_regular** diagram, system performance of the digital control system being designed is compared with the performance of an existing analog control system.



The door displacement profile of an existing system (XCL.DAT) is brought into the simulation using an import block and compared to the door displacement profile resulting from the current implementation.

By simulating the model, you can make refinements in the control strategy, as well as fine-tune the controller performance.

Once it is determined that the floating-point controller is performing adequately, the next step is to implement the controller using limited precision fixed-point blocks. This lets you simulate the behavior of the controller as it would behave as an embedded system in a fixed-point target, such as a DSP or a microcontroller.

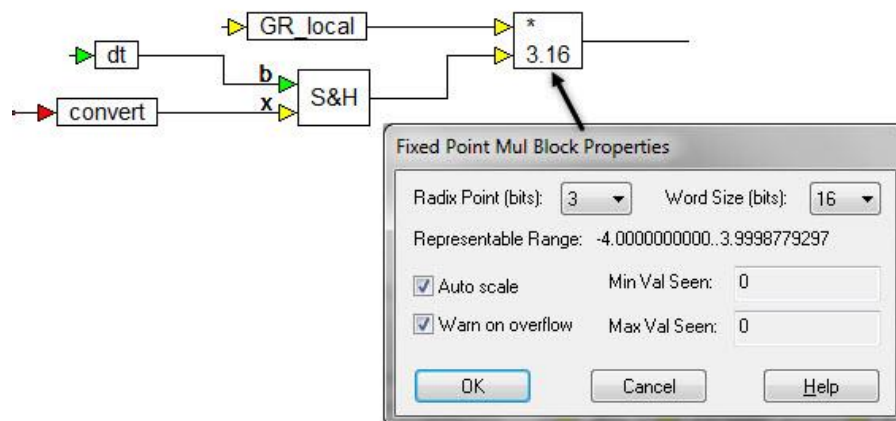
Converting to scaled fixed-point control

The underlying principle in converting an existing floating-point subsystem into an equivalent scaled fixed-point system is that every input, operation, and output be configured to reflect the realities of the target. Fundamental details — such as whether the target is an 8-, 12-, 16-, 24- or 32-bit processor — become important. As you traverse the controller implementation, from left to right, you encounter several operations, including summation, sample-and-hold, absolute value, sign, multiplication, constant, division, gain, and unit delay. Using Fixed Point blocks, each of these operations is replaced with the equivalent fixed-point operator, assuming a 16-bit target.

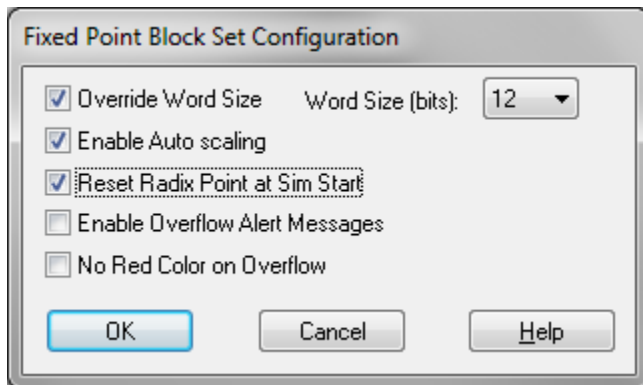
Blocks such as `unitDelay` and `sampleHold` are fixed-point-aware; that is, they automatically adjust to the incoming data type. Consequently, they can be used as is.

Constructing the encoder feedback

The output of variable `GR_local` and the output of the `sampleHold` block are wired into a fixed-point `mul` block with the following configuration:

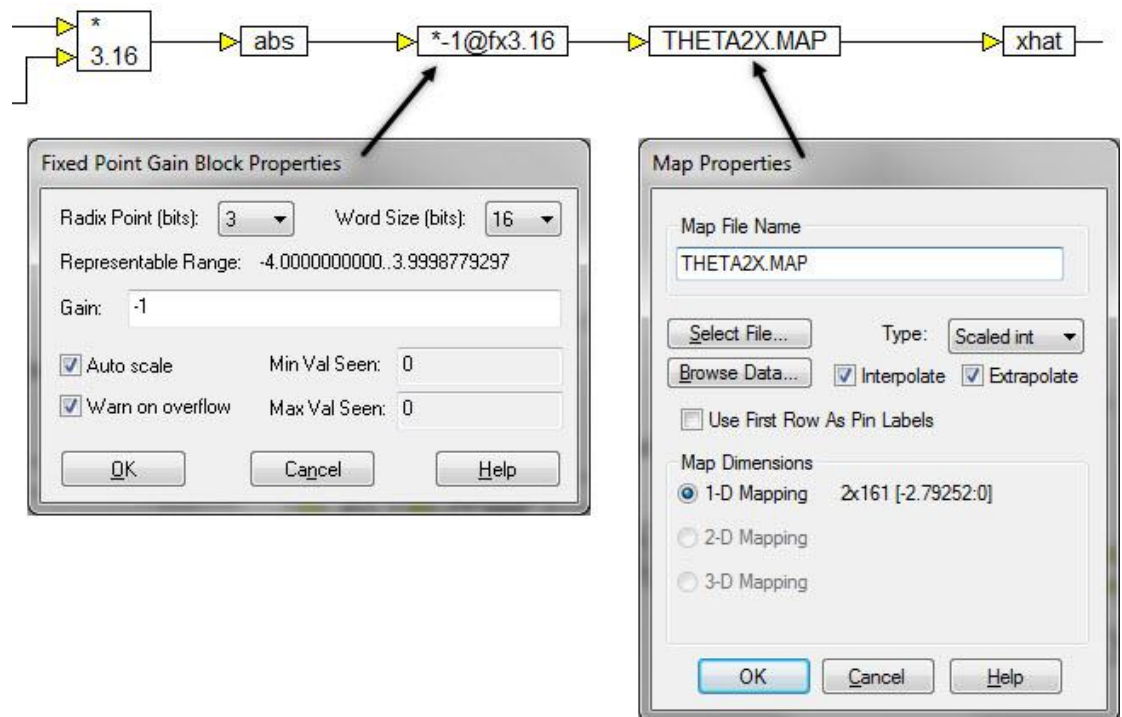


You can choose the radix point bits or select auto scaling. This option, together with the global settings in the Tools > Fixed Point Block Set Configure dialog box, let you automatically monitor the maximum and minimum values seen by the block, and adjust the radix point bits to yield maximum precision while preventing numerical overflow.



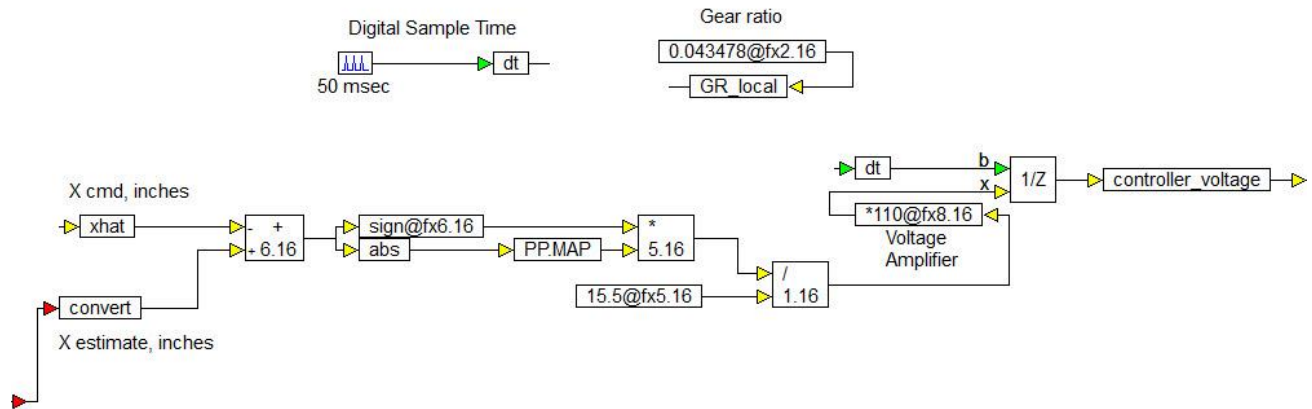
Safely maximizing the dynamic range of each computation is by far the most time-consuming component in the rapid prototyping cycle. Fixed Point blocks reduce this tedious exercise to a few mouse clicks.

Next, the mul output is connected to an abs block to compute the absolute value, which in turn is fed into a fixed-point gain block. The gain output is wired into a map block, which points to the look-up table data file THETA2X.MAP and has a Scaled Int data type. THETA2X.MAP output is fed into the variable *xhat*.



Constructing the control law

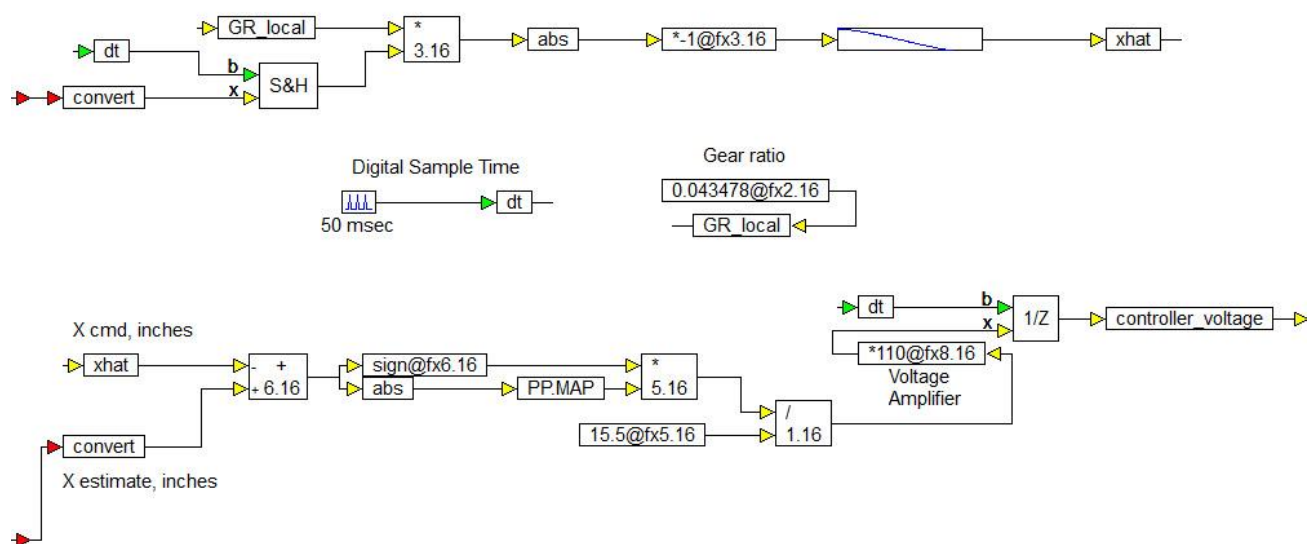
The control law is constructed using fixed-point sum, mul, div, const, and gain blocks. The gear ratio GR_{local} is defined as 0.043478 using a fixed-point const block. A 50ms pulseTrain defines dt . The map block points to PP.MAP and sets the data type to Scaled Int. The resulting control law implementation is:



Completing the controller implementation

To complete the controller implementation, the control law segment is connected to the Encoder feedback segment, and the output of the unitDelay block to the output:

DIGITAL CONTROLALGORITHM
for servo-controlled door system
implemented in scaled fixed point



The two inputs to the fixed-point Controller are scaled to the correct data types using convert blocks. The convert block connected to the encoder feedback needs a radix point precision of at least 8 bits while the convert block connected to the command input can be 6 bits.

When this simulation is executed, the controller is implemented in 16-bit scaled precision, while the rest of the simulation runs in double precision floating-point. This lets you simulate and validate the performance of the controller, as it would execute on the fixed-point target.

Prototyping the embedded control system

The fixed-point controller can easily be prototyped in a hardware-in-the-loop scenario or implemented on a target processor such as a DSP or a microcontroller. Furthermore, integrated solutions let you generate, compile, link,

download, test, debug, and validate the entire application. This dramatically reduces development time and expenses while resulting in a high-quality product that is well tested and very reliable.

Implementing a PID position controller

Floating-Point Diagram: position_control_regular

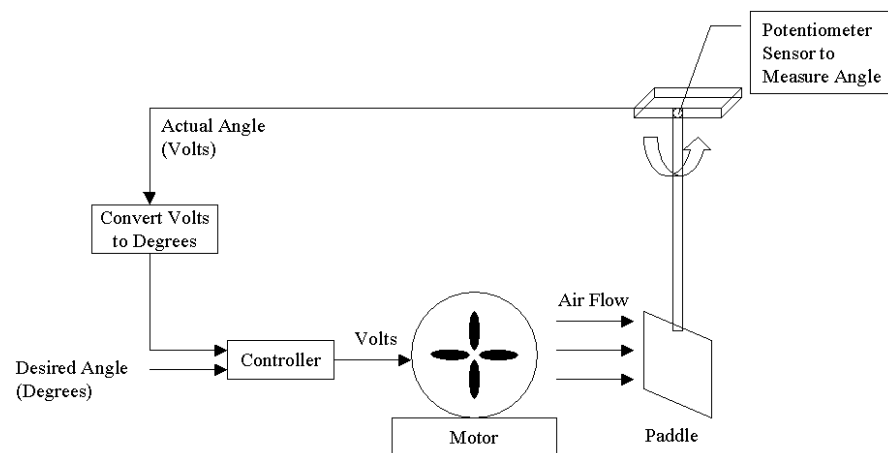
Fixed-Point Diagram: position_control_fixpoint

Location: Examples > Fixed Point

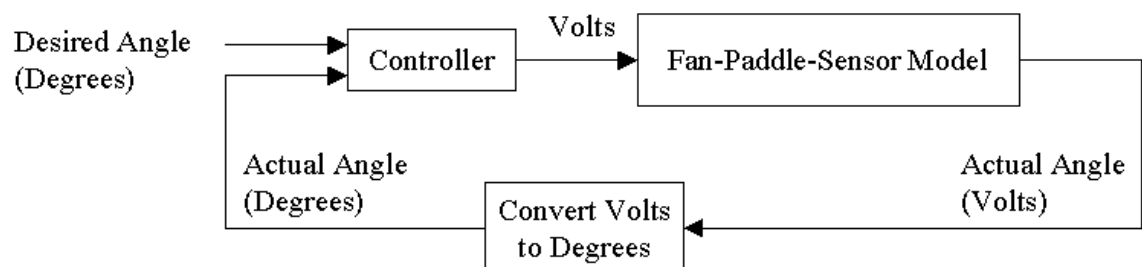
In most real-world cases, a scaled fixed-point-based embedded controller controls a real system, such as automotive brake systems, machine tools, aerospace control surfaces, and other similar systems. In each case, the best way to prototype an embedded controller is to realize the controller in scaled fixed-point implementation that is native to the target platform. The rest of the simulation — such as sensors, plant model components, and actuators — are best simulated in double-precision floating-point to most accurately reflect the real-world application scenario.

This tutorial examines the implementation of a PID controller for a position control application. The plant, controller, and other arithmetic operations are first implemented in [double-precision floating point](#). [D2HLink_12652](#)

The system comprises an electrical motor connected to a small propeller that blows air on a paddle. The paddle is moved at an angle from the vertical. The control problem is to adjust the speed of the motor by varying its input voltage to maintain the paddle at a user-defined angle from the vertical. The system can be schematically represented as shown below.



For the prototyping process, the fan-paddle-sensor subsystem can be collapsed into a single model, as shown below.



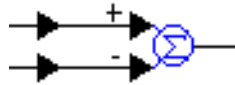
Constructing a floating-point model

The system represented above is built using standard blocks. Each of the three major components — Controller, Fan-Paddle-Sensor Model, and the Convert Volts to Degrees — are developed, linked, and simulated.

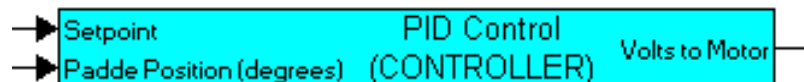
Constructing the controller

The controller has two inputs (desired and actual angles) and one output (voltage) to be applied to the motor.

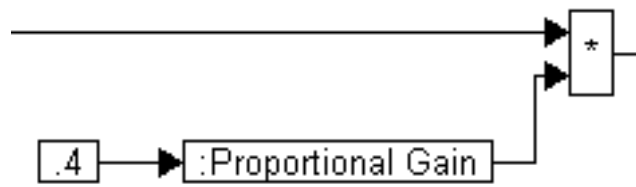
To begin modeling the controller, wire two wirePositioners to the inputs of a summingJunction block, and negate the second input, as shown below.



For ease of implementation, these blocks are encapsulated in a compound block called **PID Control (CONTROLLER)** and the inputs and outputs are labeled appropriately.

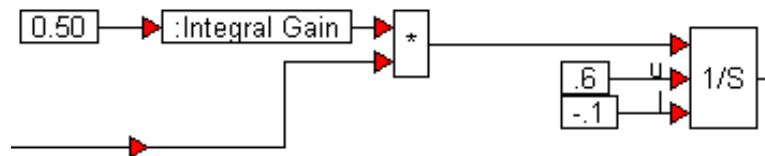


Inside **PID Control (CONTROLLER)**, the output of the summingJunction is passed through a gain of 0.001 and fed as the error input for computing *P* (proportional), *I* (integral), and *D* (derivative) components of the controller. The proportional term, encapsulated in a compound block named *proportional term* is implemented as:



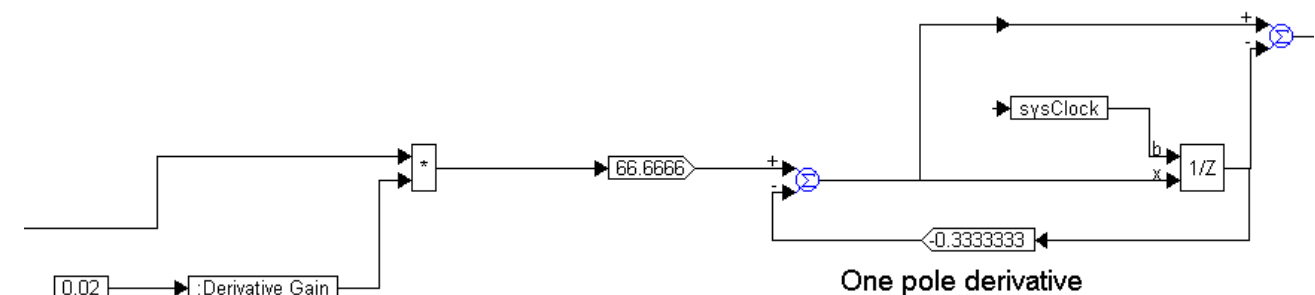
The proportional gain is set to 0.4.

The integral term, encapsulated in a compound block named **integral term** is implemented as:

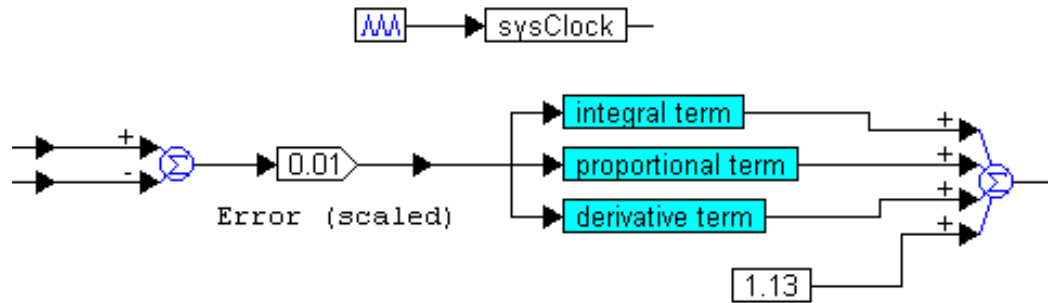


The integration is performed using a limitedIntegrator to prevent windup. The upper and lower limits are set to 0.6 and -0.1 respectively, and the integral gain is set to 0.50.

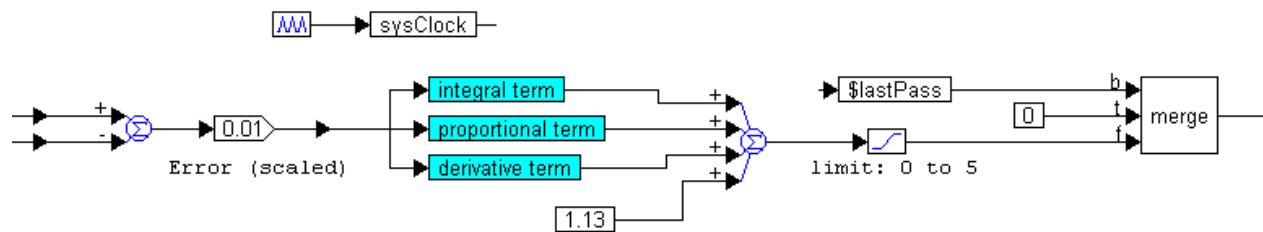
The derivative term, encapsulated as a compound block named **derivative term** is implemented as:



The computation of the derivative is implemented using a unitDelay block, two gain blocks, and two summingJunctions blocks, as shown above. The *sysClock* clock input to the unitDelay is defined using a pulseTrain block, and the contributions of the *P*, *I*, and *D* terms are summed up, as shown below.



Because the motor needs a minimum of 1.13V to turn, a constant bias of 1.13V is added to the mix. To ensure that the voltage applied to the motor is within the rated voltage range, and to shut the motor down when the simulation run is complete, the limit and merge blocks are used, as shown below.



The output of the merge block is forced to zero after the last step of the simulation, represented by the system variable `$lastPass`. For all other steps, the limit block restricts the output to the range 0V to 5V, as shown above.

Constructing the Volts to Degrees Converter

As is the case with many sensors, the potentiometer used in this application produces a voltage proportional to the actual quantity being measured: in this case the angle of the paddle from the vertical. Since the set point is in degrees, you must convert the volts corresponding to the actual angle to degrees of actual angle. The principle for modeling the conversion process is quite simple. You measure the voltage at 0° and 90° angles. Assuming a linear relationship between potentiometer volts and actual angle in degrees, the relationship can be written as:

$$\text{actual angle} = (\text{actual voltage} - 0\text{deg voltage}) * \text{degrees_per_volt}$$

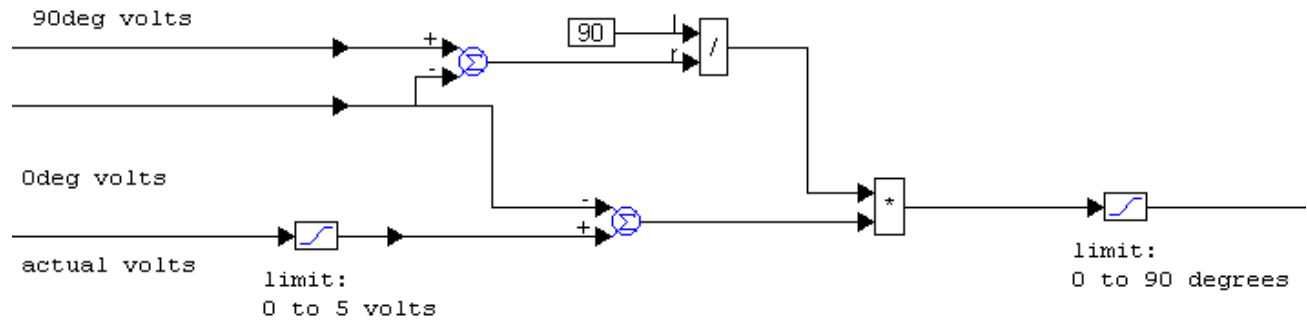
where `degrees_per_volt` is obtained from the two calibrating measurements as:

$$\text{degrees_per_volt} = (90\text{deg voltage} - 0\text{deg voltage}) / 90$$

Combining the two relationships yields:

$$\text{actual angle} = (\text{actual voltage} - 0\text{deg voltage}) * (90\text{deg voltage} - 0\text{deg voltage}) / 90$$

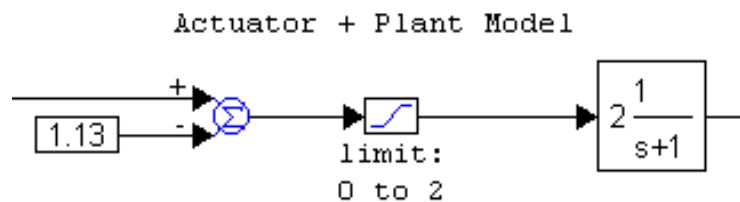
This relationship can be implemented using standard arithmetic blocks.



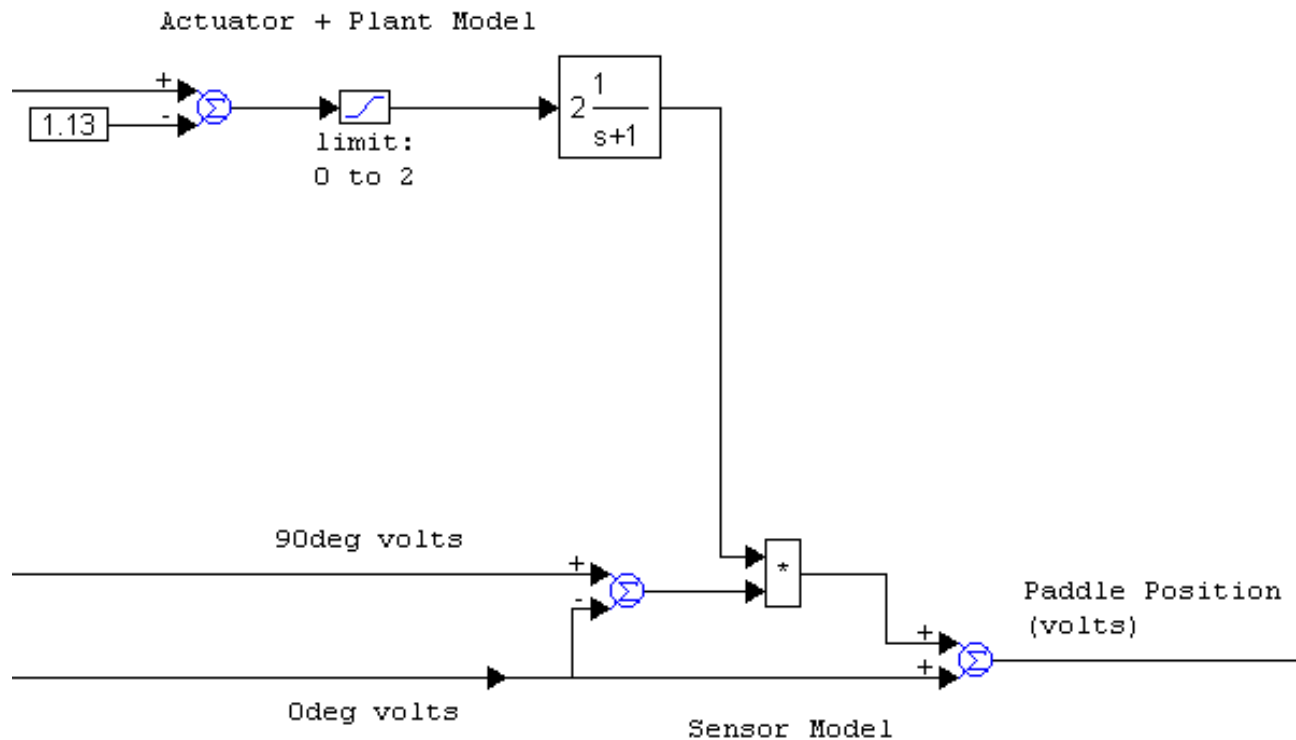
Two limit blocks are used to limit the actual volts to be within zero and five volts and the output to be within 0° and 90°. This prevents the Volts to Degrees Converter subsystem from providing out-of-range values to the controller.

Constructing the fan-paddle-sensor

The key elements to capture in the fan-paddle model are the response profile and the lag between the input and output signals. In other words, when the input changes by a certain amount, how long does it take for the output to show the effects of the change in the input and how do the input and output amplitudes correlate. Based on this approach, subtract the 1.13V that were added in the PID controller as the minimum bias voltage for the motor to run. The remainder is limited to be in the range zero to two. The time delay and response profiles can be modeled easily by a first order transfer function using the transferFunction block, as shown below.



Because the potentiometer converts angular motion into equivalent voltage and the calibrating voltage measurements for 0° and 90° are known, modeling the sensor is a simple arithmetic operation. The complete model for the fan-paddle-sensor subsystem is shown below.



This set of blocks is encapsulated in a compound block named *Fan-Paddle-Sensor Model* (*ACTUATOR+PLANT+SENSOR*). In the dialog box for the System > System Properties command, set the range of the simulation to be zero to 100 with a step size of 0.01. A slider block with range set to zero to 30 is used to specify the set-point angle, and a plot block is used to display the results. Two const blocks are used to specify the 0° and 90° calibration voltages of 1.17 and 0.68, respectively.

Fixed-point implementation of PID position controller

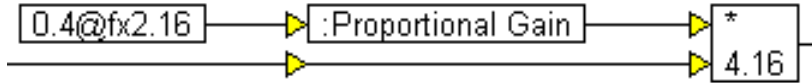
Constructing the Fixed-Point Volts to Degrees Converter

The floating-point implementation of the **Volts to Degrees Converter** was the arithmetic implementation of the equation:

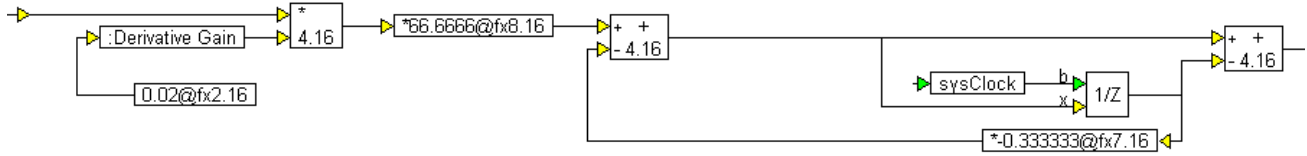
$$\text{actual angle} = (\text{actual voltage} - 0\text{deg voltage}) * (90\text{deg voltage} - 0\text{deg voltage}) / 90$$

Compared to the floating-point implementation, the only differences are the fixed-point const and mul blocks used to define the integral gain and to perform multiplication, respectively. This set of blocks is encapsulated the **Integral Term** compound block.

The **Proportional Term** compound block contains the fixed-point equivalent.



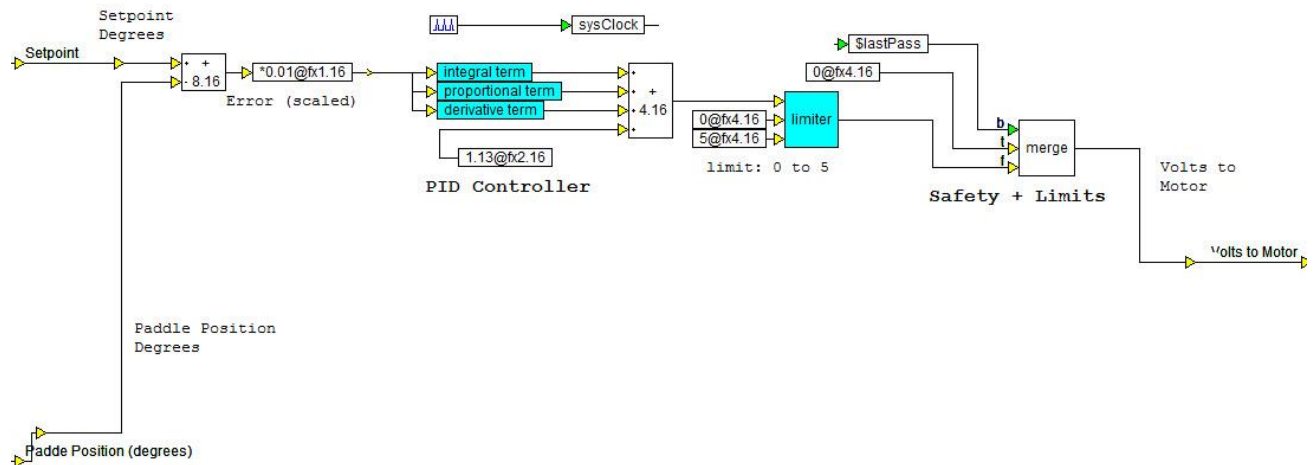
The arithmetic operations of the derivative term are replaced with fixed-point equivalents, const, mul, sum, and gain, as shown below.



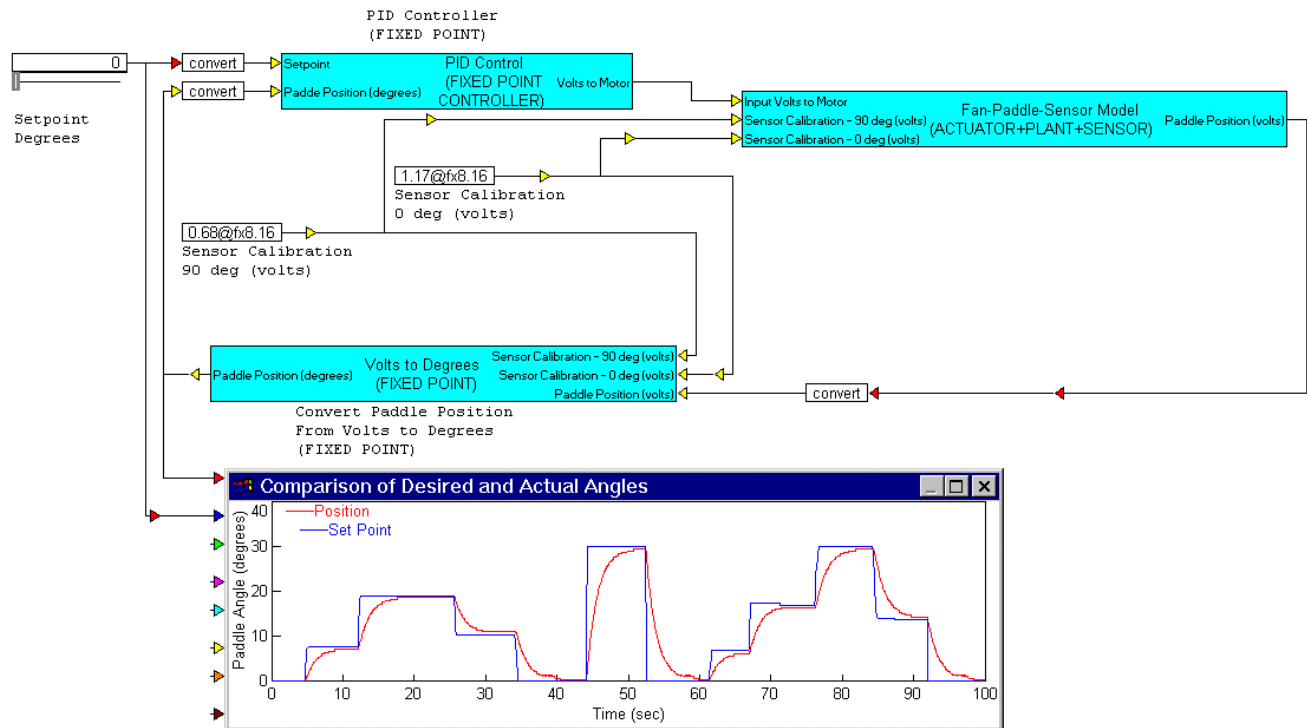
One pole derivative

These blocks are encapsulated in the **Derivative Term** compound block.

Combining the three control terms, the fixed-point PID control can be implemented as:



This set of blocks is encapsulated in the **PID Control (FIXED POINT CONTROLLER)** compound block. The complete system becomes:



Three convert blocks are used to ensure that the inputs to the **Controller** and the **Volts to Degrees** converter are the correct data type. Furthermore, the 0° and 90° calibration voltages are defined using fixed-point const blocks. The simulation parameters remain unchanged from the floating-point implementation.

It is important to note that in the simulation depicted above, the **PID Control** and **Volts to Degrees Converter** are simulated by Embed in scaled fixed-point while the **Fan-Paddle-Sensor** is simulated in floating-point. This means that you can simulate how a given control or logic prototype would execute on a fixed-point embedded system target, such as a DSP or a microcontroller. This lets you answer in a single design and simulation iteration, crucial questions, such as:

- Is it feasible?
- Will it work?
- Will it work on the embedded target that I have chosen or have in mind?
- Am I getting the most dynamic range for each of my variables?
- Can I guarantee that none of the variables will suffer numerical overflow for the entire range of inputs and outputs that I am designing for?
- Does my control system exceed or at least meet the design specifications?

Embedded system prototyping methodology

Diagram: position_control_embedded

Location: Examples > Fixed Point

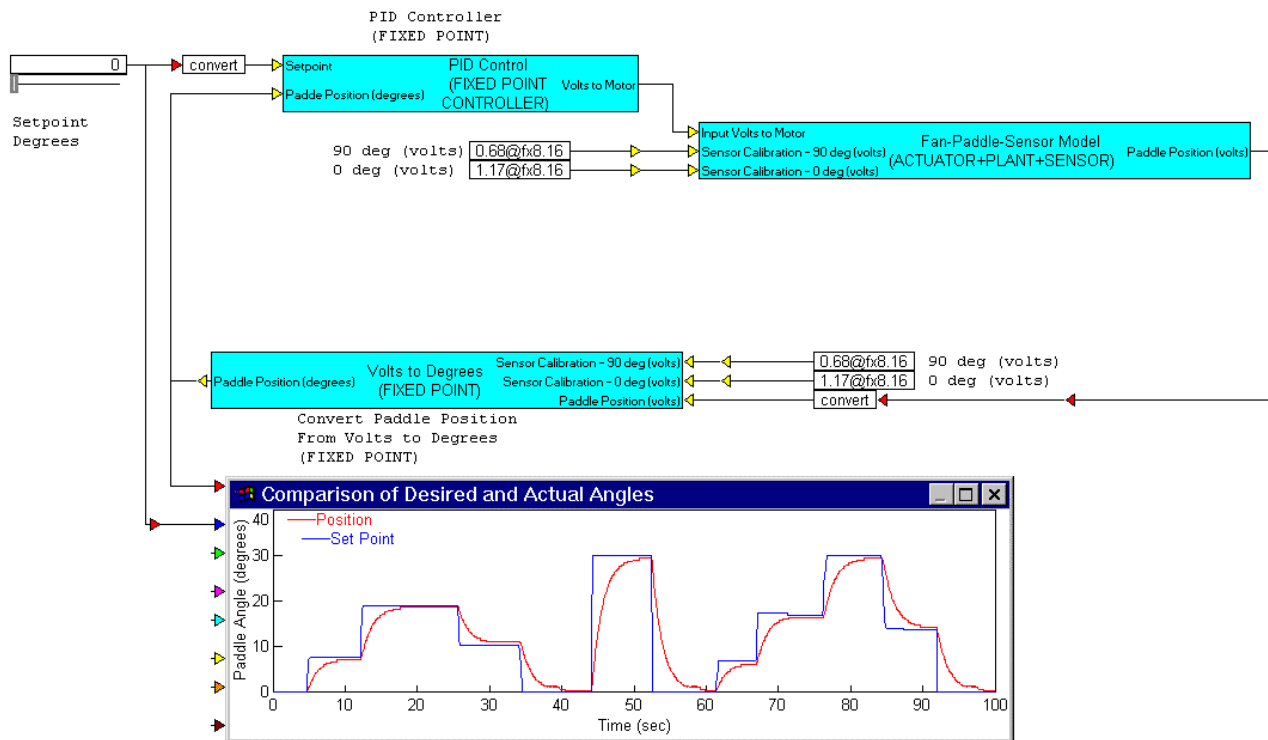
Once the controller and control logic are simulated in scaled fixed-point, and the design issues are addressed satisfactorily, the next step is the actual implementation of the controller and control logic on target hardware.

With Embed, you can quickly and easily implement fixed-point controllers and logic on embedded targets. The embedded code can be exercised and tested by changing set points using slider blocks and observing output in plots while the actual control code is executed on the embedded target.

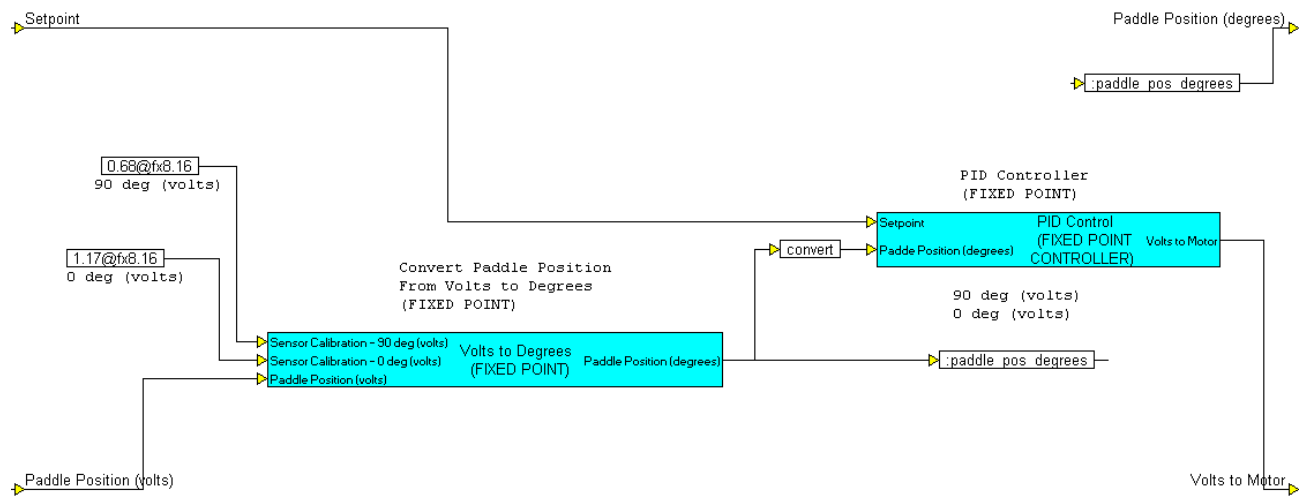
From a methodology point of view, the main concept that is crucial in embedded system prototyping is the principle of adequate and complete encapsulation. That is, all control, logical, input-output (I/O,) and other subsystems that must run on the embedded target, must, at a top level be contained in a single compound block.

In the fixed-point version of the fan-paddle position control system, the **PID Controller (FIXED POINT CONTROLLER)** and **Volts to Degrees (FIXED POINT)** are the control and logic functions for this system. To ready this system for direct implementation on an embedded target, it follows that all you need to do is collapse these two compound blocks into a single compound block.

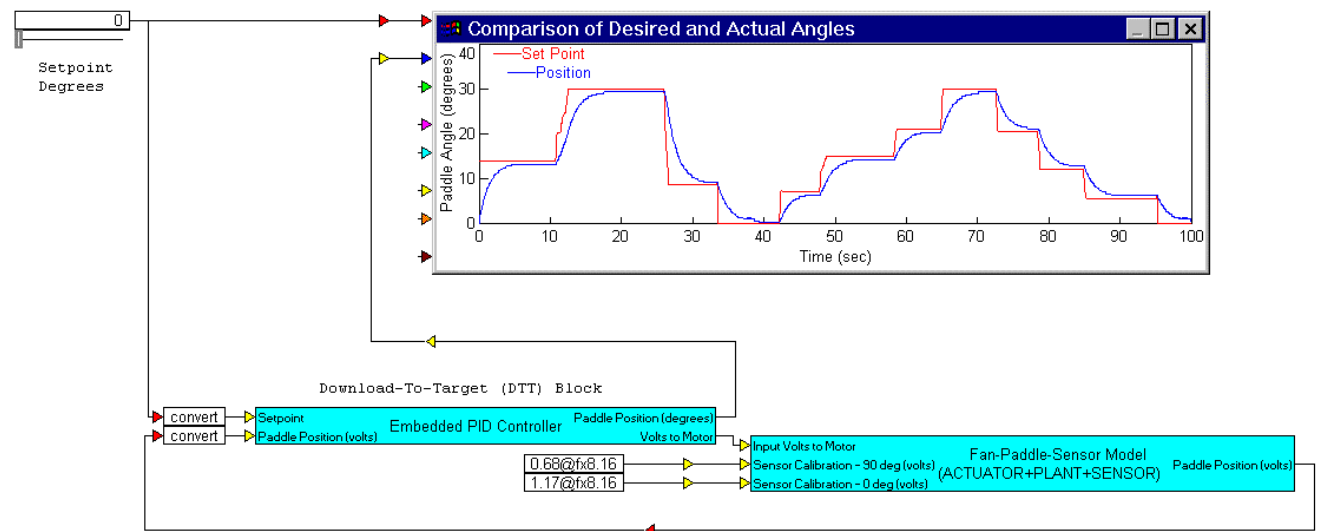
To prepare for encapsulation, begin by duplicating the 0° and 90° calibration value constant, so that **Fan-Paddle-Sensor** and **Volts to Degrees Converter** each has its own set of constants.



Next, encapsulate **PID Controller** and **Volts to Degrees** in a single compound block named **Embedded PID Controller**.



A new local variable called *paddle_pos_degrees* is created and wired to an additional output tab to bring this value out and provide external access to it for monitoring purposes. The complete system representation is as follows:



In this form, **Embedded PID Controller** can generate, compile, link, and download embeddable C code to supported targets and perform hardware-in-the-loop system prototyping and validation.

When performing hardware-in-the-loop validation, the use of analog and digital inputs and outputs is quite common. Embed provides analog and digital I/Os that can be configured just as any other Signal Producer or Consumer block and placed inside the Embedded PID Controller block. Embed supports automatic programming of on-board analog and digital I/O, as well as all the important peripherals. The parameters entered in the configuration of the I/O points and peripherals, such as channel number, and range, are extracted and placed in the embedded control code for the **Embedded PID Controller** block and automatically downloaded to the target.

Generating DLLs

DLLs are useful for:

- **Speeding up simulation time:** When a diagram contains DLLs, it requires less disk space and memory since its executable program files contain the names of the DLL functions but not the code for the functions. For particularly large diagrams, the use of Embed-callable DLLs can significantly increase the speed of your simulations.
- **Performing multi-rate execution:** When a compound block is converted into a DLL, the DLL retains the step size and integration method in use at the time of DLL generation. If the diagram that calls the DLL has a faster or slower step rate, the DLL skips or adds steps to maintain its own clock rate. This allows you to have a low frequency overall diagram with high frequency components compiled as DLLs.
- **Protecting intellectual property:** Because DLLs cannot be reverse-engineered into readable source code, you can be sure that no one can access your intellectual property.
- **Interface with other applications or simulators**

Creating a DLL

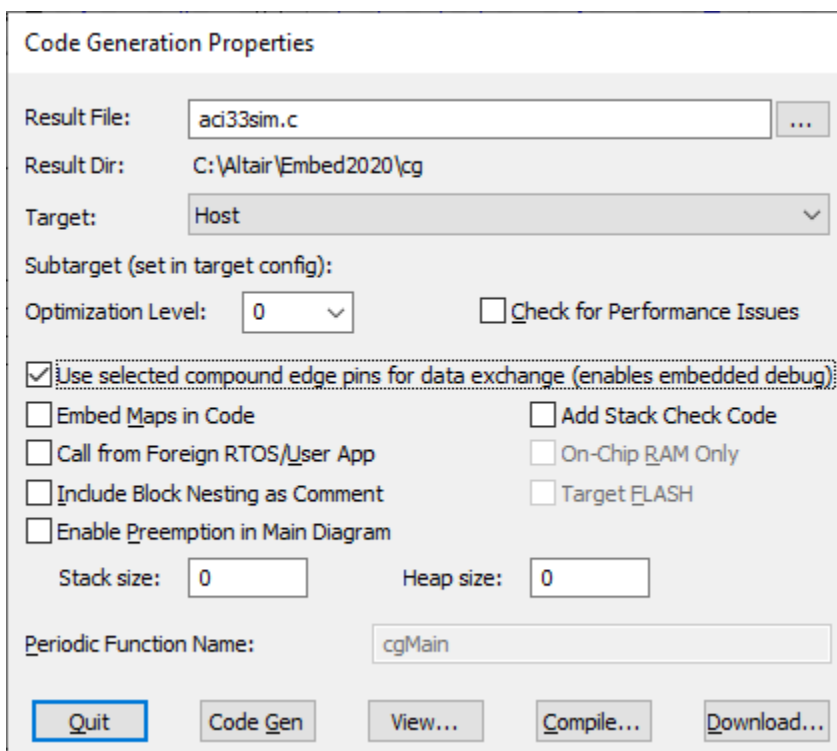
Only compound blocks can be converted into DLLs. When a compound block is converted into a DLL, the DLL retains the step size and the integration method in use at the time of DLL generation, and not those selected or specified by the diagram calling the DLL. The DLL does, however, use the simulation start and end times of the diagram that calls it.

The generated DLL includes the block and connector names. When the resulting DLL has numerous input and output connectors, the connector names make it easy to identify the correct wiring paths

To create a DLL

1. Open the diagram that contains the blocks you want to convert into a DLL.
2. Collapse the blocks into a compound block, if you have not already done.
3. Select the compound block.

4. Choose **Tools > Code Gen.**



The **Result File** box displays *diagram-name.C*, where *diagram-name* is the name of the current diagram. By default, Embed uses the diagram name as the name for the DLL. For example, if you enter ACI33SIM.C, Embed creates a DLL called ACI33SIM.DLL.

5. If you are creating more than one DLL from a single diagram, give each DLL a unique name.
6. The **Result Dir** box indicates where the DLL will be stored. To change the location, click
7. The **Target** box contains the target platform for code generation. Choose **Host**, if it is not already selected.
8. Activate **Use selected compound edge pins for data exchange**. This parameter allows the generated code to include calls to send data to and receive data from Embed. This parameter is dimmed when no compound block is selected.
9. Choose from the following parameters:

Activate this parameter	To
Add Stack Check Code	Do not activate.
Call from Foreign RTOS/User App	Do not activate.
Check for Performance Issues	Do not activate.
Embed Maps in Code	Insert map file contents directly into the generated code. When this parameter is activated, the resulting executable will be portable because the map file is no longer needed.
Heap Size	Does not apply.

Include Block Nesting as Comment	Include comments in the generated code that indicate the compound blocks that correspond to the code.
On-Chip RAM Only	Does not apply.
Optimization Level	Specifies compiler optimization level, from 0 (no optimization) to 4 (highest level). In rare circumstances, Level 4 may yield inconsistent results, necessitating a lower level of optimization.
Periodic Function Name	Does not apply.
Stack Size	Does not apply.
Target FLASH	Does not apply.
Use selected compound edge pins for data exchange	Creates the DLL.

10. Click **Compile**.
11. Embed opens a text window in which it displays DLL creation. When the DLL has been generated, press **any key** to return to the Code Generation Properties dialog box.
12. Click **Done**.

Calling a DLL from an Embed diagram

The process of calling a DLL from an Embed diagram involves binding the DLL to a userFunction block and then wiring the userFunction block into the diagram. During simulation, each time the userFunction block is executed, Embed calls the DLL.

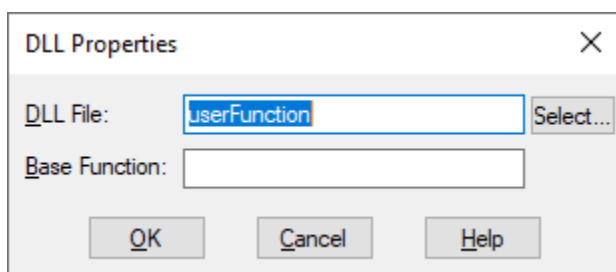
After you bind a DLL to a userFunction block, Embed renames the userFunction block with the DLL name. For example, if you created a DLL named AC Motor, Embed renames the userFunction block to AC Motor.

If you elected to retain connector labels when you created the DLL, you can display the labels on the userFunction block using the View > Connector Labels command. Connector labels make it easy to correctly wire a userFunction block into a diagram.

To bind a DLL to a userFunction block

1. From the **Blocks > Extensions** menu, drag a **userFunction** block into your diagram.
2. Choose **Edit > Block Properties**.
3. Click the **userFunction** block.

The DLL Properties dialog box appears.



4. Do the following:

- In the DLL File box, enter the complete file specification of the DLL. This name corresponds with name specified in the [Result File box in the Code Generation Properties dialog box](#). If you are not sure where the DLL file resides, click **Select** to locate it.
- In the Base Function box, enter **cgMain**.

5. Click **OK**.

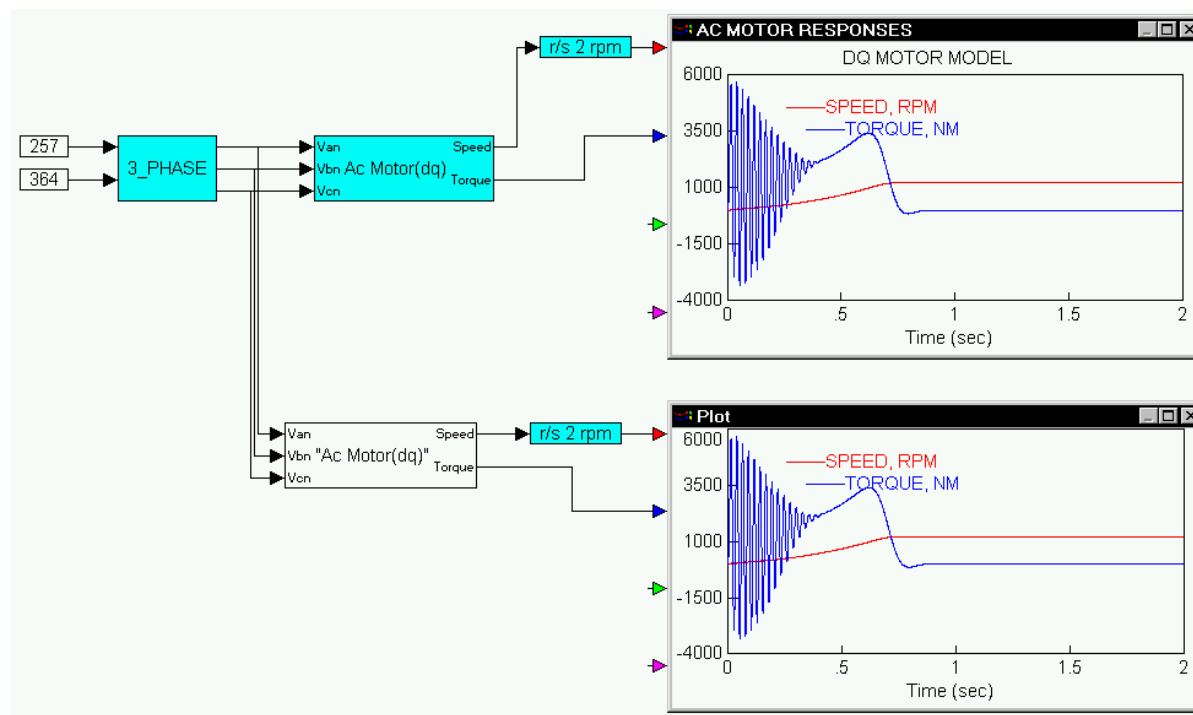
To view connector labels on a DLL

- Choose **View > Connector Labels**.

Verifying DLL results

Before editing the diagram to replace the existing blocks with a corresponding DLL, it is a good idea to verify that the DLL operates correctly.

For example, in following diagram, a DLL is created from the AC Motor (dq) compound block. To verify that the DLL operates correctly, the inputs to the compound block are fed into the DLL. A second plot block, with the same properties as the original plot is placed in the diagram. If both plots register the same results when you simulate the diagram, then the DLL is correct.



Comparing simulation speed

You can easily compare how much faster a simulation runs with a DLL than using Notify At Simulation End. It displays how long it takes to run a simulation in simulated time and real time.

To compare performance

1. Choose **Simulate > Simulation Properties**; then click the **Preferences** tab.
1. Activate **Notify At Simulation End** and click **OK**.
2. Disconnect the DLL from the diagram.

3. Run the simulation.
4. Reconnect the DLL to the diagram and disconnect the corresponding compound block from the diagram.
5. Run the simulation.
6. Compare the real-time simulation results from the two simulation runs.

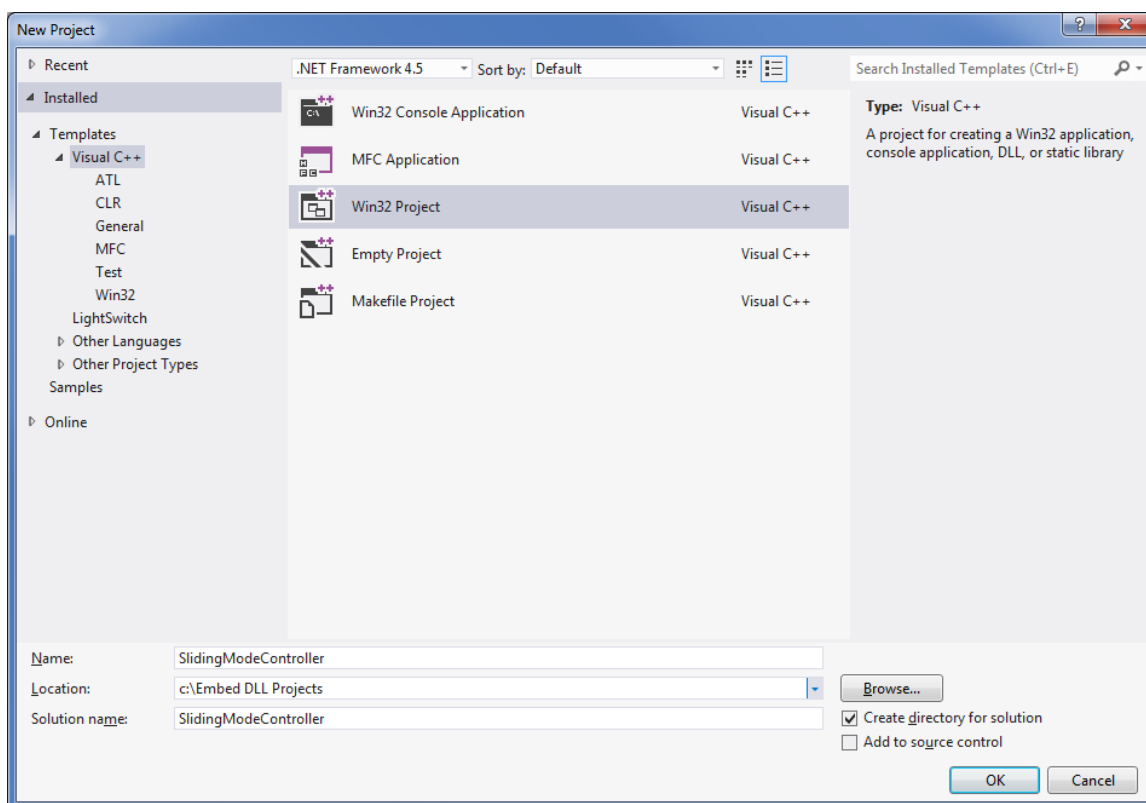
Building a custom DLL

If you want to add a custom dialog box to a DLL, you must compile and link the code manually.

Nowadays, most languages have a Project Build facility that automates the process of building an executable or DLL. We recommend using Microsoft Visual Studio (v6 or higher). Refer to the documentation for the application language you are using for specific instructions.

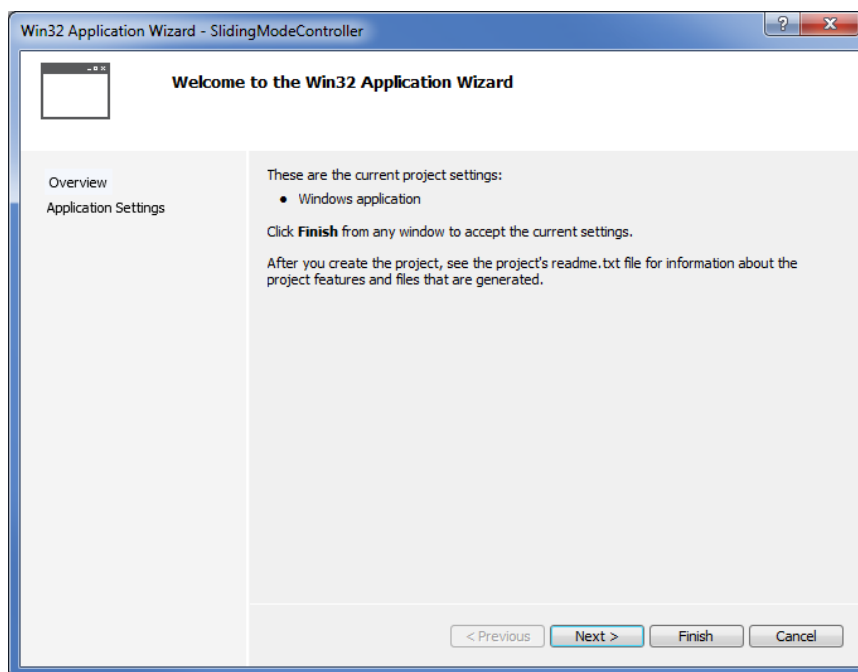
To build a custom DLL with Microsoft Visual Studio v2012

1. Invoke **Microsoft Visual Studio**.
2. Choose **File > New** to create a new project workspace.
3. In the left windowpane, select **Visual C++**. In the right windowpane, select **Win32 Project**. Then enter the **project name** and **location** in the text boxes at the bottom of the window. We recommend activating the **Create directory for solution** parameter. When you do so, the **Solution Name** defaults to the project name.



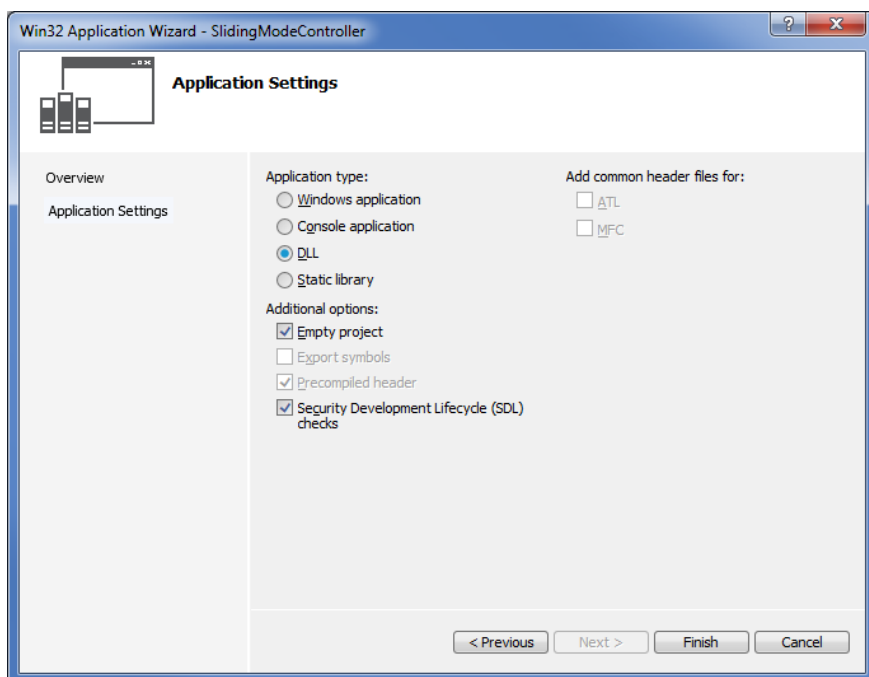
4. Press **OK**.

The Win32 Application Wizard starts.



5. Press **Next**.

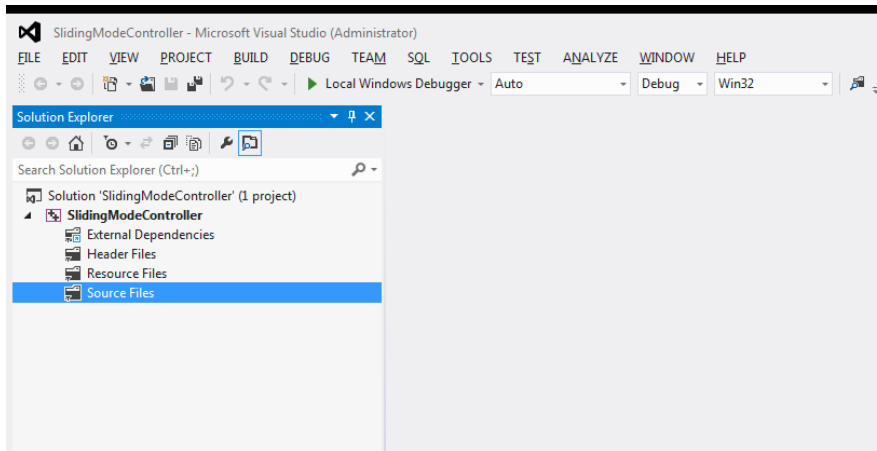
The Application Setting window appears.



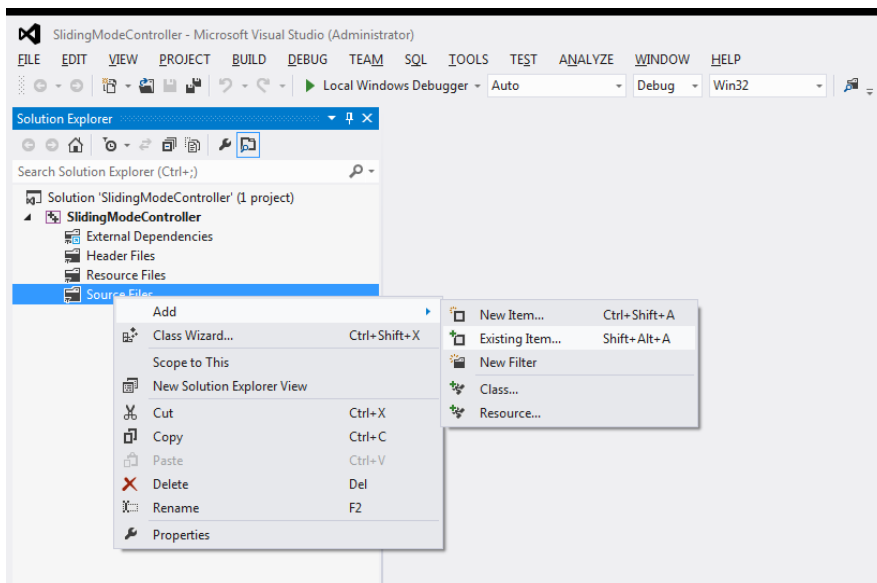
6. Do the following:

- Under **Application Types**, select **DLL**.
- Under **Additional Options**, select **Empty Project**.
- Click **Finish**.

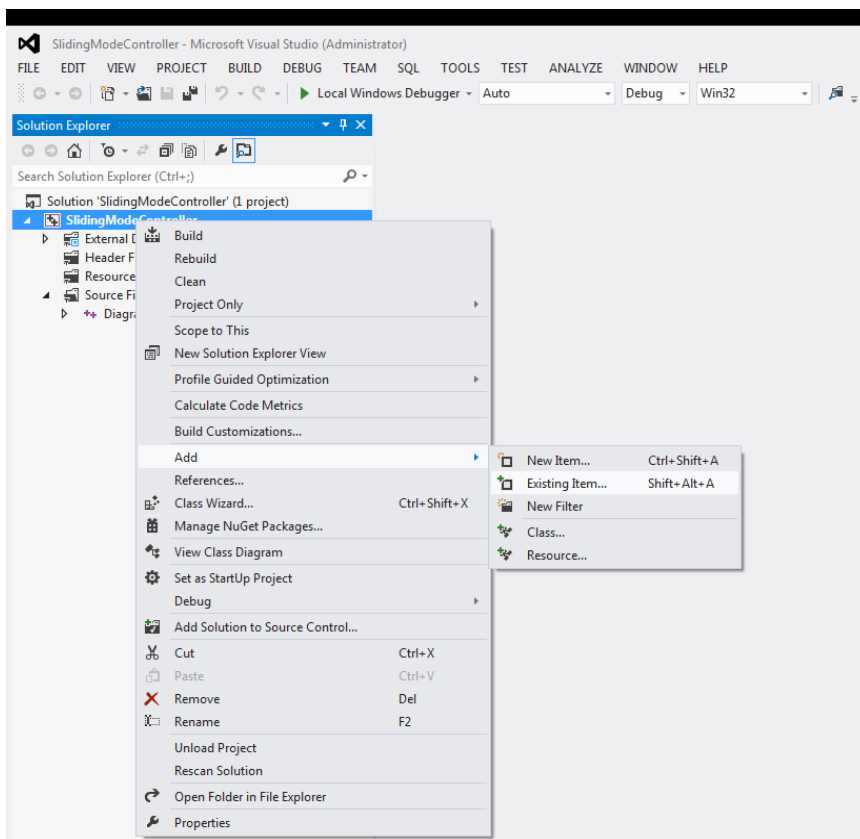
7. The Solution Explorer window appears.



8. Before proceeding, open **Windows Explorer** and copy the custom C or Embed-generated C file to your newly created project location.
9. Go back to the **Solution Explorer** window.
10. Right-click **Source Files**. Then click **Add > Existing Item**.



11. Select your **custom C** or **Embed-generated C** file and press **Add**.
12. Right-click the **Project Name** and select **Add > Existing Item**.



13. Insert the following:

- `<install-directory>\CG\LIB\CGDLL32.LIB`
- `<install-directory>\VSDK\LIB\VISSIM32.LIB`

Note: To specify file names, enter the full pathname, including drive specification for CGDLL32.lib and VISSIM32.lib.

14. Choose **Build > Properties** to establish the settings for the build.

15. In the **Properties Pages** dialog box, do the following:

- Click Configuration Properties > C/C++ > General.
- Select Additional Include Directories; then in the corresponding cell, enter `<install-directory>\VSDK\INCLUDE;`
`<install-directory>\CG\INCLUDE`

Note: To specify directory names, enter the full pathname, including the drive specification for the INCLUDE directories.

- Click OK.

16. Choose **Build > Build Solution** to build the project.

Generating code from automatically-generated DLLs

You can include pre-existing, user-written or automatically-generated DLLs in the portion of the diagram from which you are generating code. To do so, you must declare the DLL functions in USERDLL.H and include the library for the DLL in VSMDLL32.BAT.

For example, to generate a DLL from a compound block in which the DLL named READ_INPUT_FILE is embedded, do the following:

- In *USERDLL.H*, add:

```
__declspec(dllexport) void _stdcall EXPORT READ_INPUT_FILE
(double p[],double in [],double out[]);
```

This line declares the exported DLL function READ_INPUT_FILE so that automatic DLL code generation will make the proper external reference to it.

- In *VSMDLL32.BAT*, add:

```
set userlibs=READ_INPUT_FILE.LIB
```

This line associates the shell variable userlibs with the library file for the exported DLL function.

- To associate multiple libraries with userlibs, separate each library file with an empty space. For example:

```
set userlibs=READ_INPUT_FILE.LIB READ_OUTPUT_FILE.LIB
```

Note: If the user-written file is a CPP file, you must prefix the exported functions in the CPP file with extern "C". For example:

```
extern "C" __declspec(dllexport) void _stdcall EXPORT
READ_INPUT_FILE (double p[],double in [],double out[]);
```

This line causes the external name generated by the CPP file to be compatible Embed DLL naming conventions.

Troubleshooting

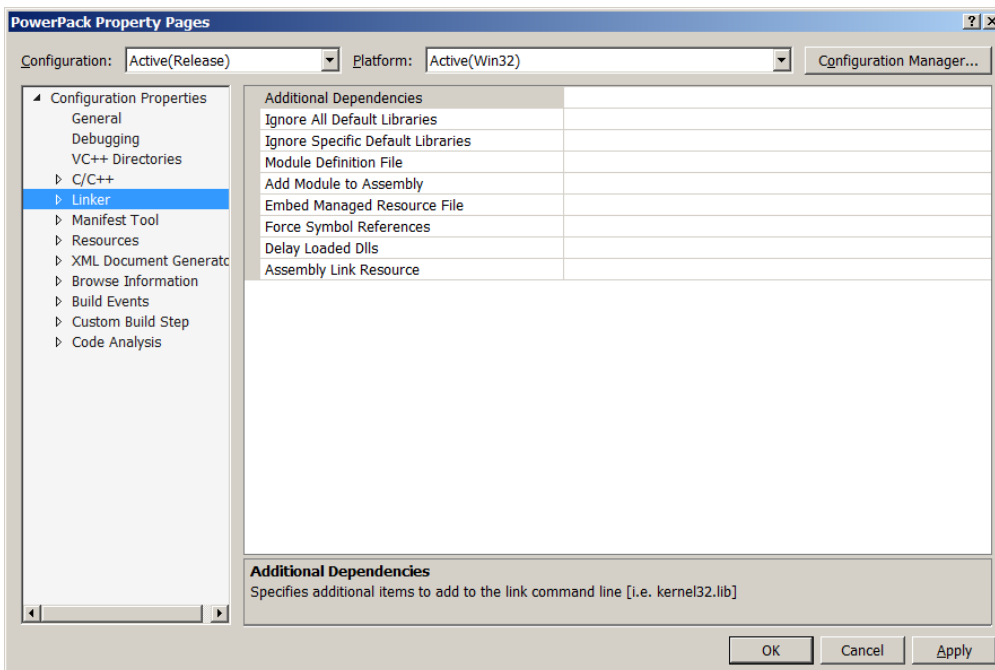
Out of Environment Space error message

The most common problem is an Out of Environment Space error message in the MS-DOS window. To rectify this problem, select the Memory tab in the MS-DOS Prompt Properties, and increase the Initial Environment setting to a larger value. Then try again.

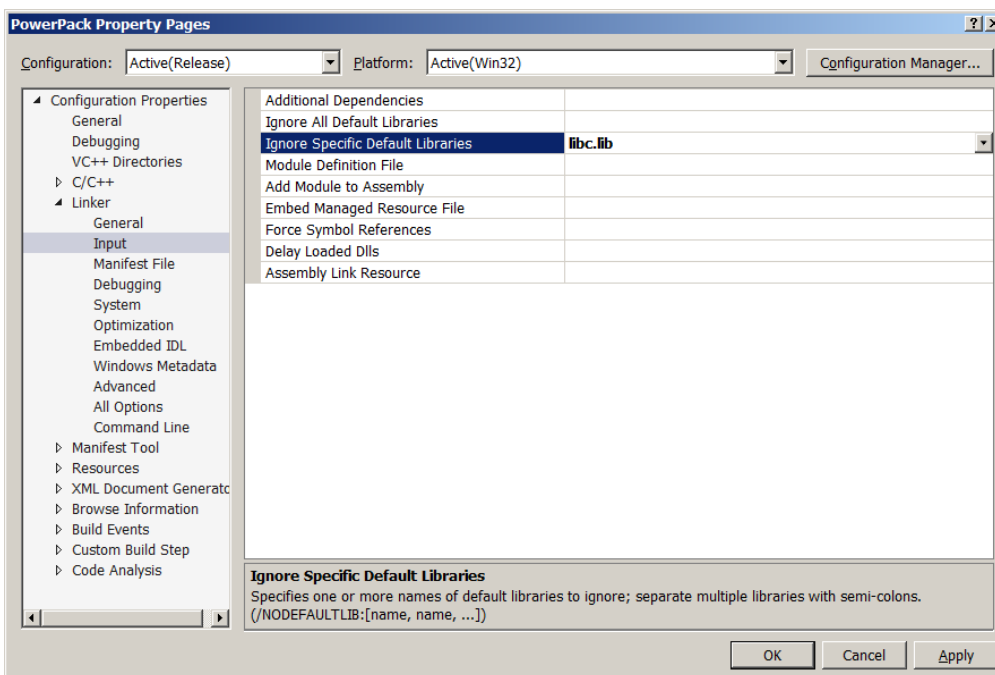
LINK warning about LIBC.LIB

If you receive a Link warning message during the build, you should instruct the Project Build facility to ignore LIBC.LIB, which you can do with Microsoft Visual Studio (v6 or higher). The following example steps you through removing LIBC.LIB using Microsoft Visual Studio v2012:

1. Choose **Project > <project-name> Properties**.
2. Under the Property Pages dialog box, in the left windowpane, choose **Linker** to expand the folder.



3. Under **Linker**, choose **Input**.



4. In the right windowpane, choose **Ignore Specific Default Libraries**, and in the corresponding cell to the right, enter **LIBC.LIB**.
5. Click **OK**.
6. Choose **Build > Rebuild Solution** to build the entire project or **Build > Build Solution** to build only the files that have been modified since the last successful compile and link.

LINK warning messages that can be ignored

You may see additional link errors, such as:

- LINK: warning LNK4099 : PDB vc40.pdb was not found with *ldeb\FILEIO.OBJCgdll32.LIB* or at *c:\src\cg\ldeb\vc40.PDB*
- Linking object not found

These are compiler warnings that there is not sufficient information to create a debug database for the project.

Generating Simulation Objects

Generated code for a simulation object is created in such a way as to allow you to create multiple instances of the generated simulation object. This means that you can create multiple simulation object instances and run them in parallel. This also means that data structures must be dynamically allocated, as opposed to code generated for embedded targets, which uses statically allocated data structures.

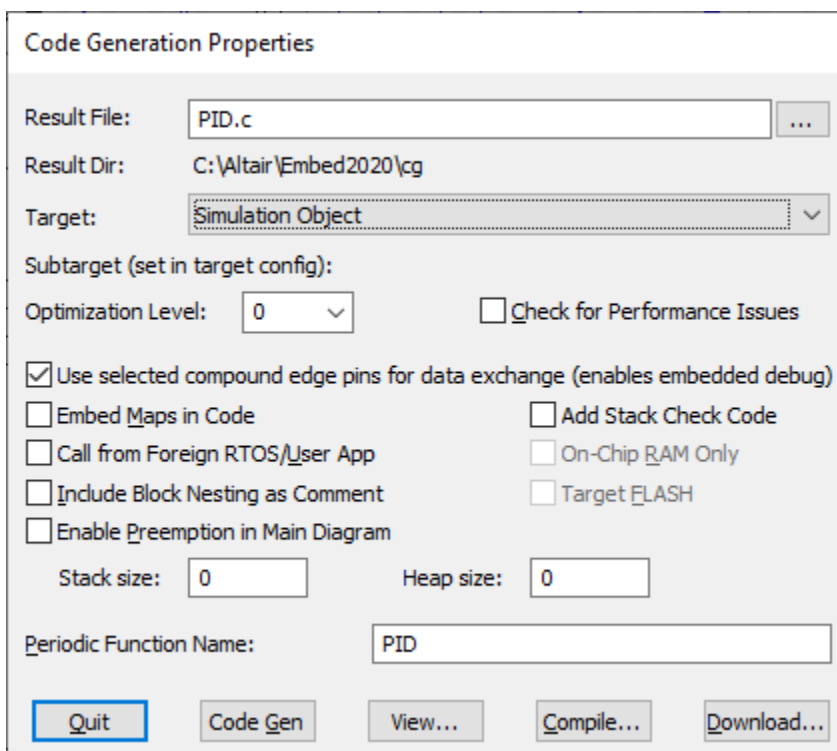
Creating a Simulation Object

Only compound blocks can be converted into simulation objects. When a compound block is converted into a simulation object, it retains the step size and the integration method in use at the time it was generated, and not those selected or specified by the diagram in which the simulation object will be embedded. The simulation object does, however, use the simulation start and end times of the diagram that calls it.

To create a simulation object

1. Open the diagram that contains the blocks you want to convert into a simulation object.
2. Collapse the blocks into a compound block, if you have not already done so.
3. Select the compound block.

4. Choose **Tools > Code Gen.**



The **Result File** box displays *compound-block-name.C*, where *compound-block-name* is the name of selected compound block in the current diagram.

If you are creating more than one simulation object from a single Embed diagram, give each simulation object a unique name.

5. The **Result Dir** box indicates where the simulation object will be stored. To change the location, click
6. The **Target** box contains the target platform for code generation. Choose **Simulation Object**, if it is not already selected.
7. Activate **Use selected compound edge pins for data exchange**; then choose from the following parameters:

Activate this parameter	To
Add Stack Check Code	Do not activate.
Call from Foreign RTOS/User App	Do not activate.
Check for Performance Issues	Do not activate.
Embed Maps in Code	Insert map file contents directly into the generated code. When this parameter is activated, the resulting executable will be portable because the map file is no longer needed.
Heap Size	Does not apply.
Include Block Nesting as Comment	Include comments in the generated code that indicate the compound blocks that correspond to the code.

On-Chip RAM Only	Does not apply.
Optimization Level	Specifies compiler optimization level, from 0 (no optimization) to 4 (highest level). In rare circumstances, Level 4 may yield inconsistent results, necessitating a lower level of optimization.
Periodic Function Name	Specifies the name of the simulation object.
Stack Size	Does not apply.
Target FLASH	Does not apply.
Use selected compound edge pins for data exchange	Activate this parameter.

8. Click **Compile**.
9. Embed opens a text window in which it displays simulation object creation. When the simulation object has been generated, press **any key** to return to the Code Generation Properties dialog box.
10. Click **Done**.

Communicating with an embedded simulation object

Embed provides a support library that contains API functions to support code generation. These include, among other things, numerical integration, transfer function filter, time delay, and pulse train. The [support library](#) resides in the <install-directory>\CG\LIBRARY directory.

Embed also includes a sample file that invokes the API and shows how calls are used. This file, named SIMOBJ.C, resides in \CG\LIBRARY.

To communicate with your simulation object

1. Open your custom application file.
2. Create a handle to your simulation object using the **createSim** function.
3. Make calls to the **vsmCgRuntime** command.

Using the createSim function

The createSim function returns a handle to the simulation. With the handle, you can invoke vsmCgRuntime to run the simulation. Every time you call createSim, you get a unique handle to a new simulation object. This means that you can have multiple simulation objects embedded in your custom application.

```
variable = sim-name createSim();
```

The sim-name is the function name you provided when you generated the code.

The declaration for the createSim function is:

```
SIM_STATE *variable
```

Using the vsmCgRuntimeCommand

This function runs the simulation using the specified input signals.

```
int vsmCgRuntimeCommand(
    IN SIM_STATE * hSim,
    IN int cmd,
    IN double inSig[],
    OUT double outSig[]
    INOUT double * targetTime);
```

Command values

RTE_RUN_TO_TIME Runs the Embedmodel until *targetTime is reached. InSig contains a vector of double precision inputs to the simulation. Note that the order of the inputs is the top-down order of the original compound block connectors. On return, outSig contains the calculated output values.

Return values for RTE_RUN_TO_TIME

VSM_SIM_MATH_ERR Indicates that a math error has occurred.

VSM_SIM_END_TIME_EXCEEDED Indicates that *targetTime is past the current Embed end time.

VSM_SIM_USER_STOP Indicates that a user has pressed the stop button or that the stop block has been actuated.

VSM_ERR_FILE_ACCESS A map or import file could not be opened.

RTE_SET_EXIT_CONDITION Takes the string in inSig and sets it as a C expression to be evaluated at each time step for early simulation termination.

RTE_SET_TIME Moves Embed time forward to *targetTime. Note that blocks do not calculate during this call.

RTE_GET_TASK_TIME Fills *targetTime with the current Embed time.

RTE_GET_TIME_STEP Returns the current Embed time step.

RTE_GET_NEXT_TIME Returns the next time at which Embed calculates new values.

Using the vsmCgGetLastErrorString()

This function returns a text description of the most recent error. If there has not been an error, a NULL pointer is returned.

```
const char* vsmCgGetLastErrorString( IN SIM_STATE * hSim)
```

Sample file with simulation object interface

The following C file shows the interface to Embed simulation object code generation. This is a generic wrapper for a single instantiation of a simulation object. You will replace this file with your own user interface that can instantiate any number of simulation objects.

```
#include <math.h>
#include <memory.h>
#include <stdlib.h>
#include "vsuser.h"
#include "cgen.h"

// Sample file to show how to create and interface to a simulation object
```



```

#include "vsmApp.h" // This file contains the vsmCgRuntimeEvent() commands
#include "cmdApi.h"
#define MAX_VSM_ARG 64
#define TIME_END 1

int main(int argc, char **argv)
{
    SIM_STATE *hSim; // Declare a handle to the sim
    double inSig[MAX_VSM_ARG], outSig[MAX_VSM_ARG], simTime,T, timeStep=.05;
    int a, retVal;
    double endTime=TIME_END*1.000000001;

    hSim = cgMainCreateSim(); // Create sim, return handle (Instantiate a simObject)

    vsmCgRuntimeCommand(hSim,RTE_GET_TIME_STEP,&timeStep,0,0); // Get sim timestep (optional)
    vsmCgRuntimeCommand(hSim,RTE_RESET,0,0,0); // Reset the sim (required before each run)
    for (T=0; T <= endTime; T+=timeStep)
    {
        inSig[0] = 5; // To do: supply your arguments here
        inSig[1] = 5; // We supply some arg values here
        inSig[2] = 0;
        retVal = vsmCgRuntimeCommand(hSim,RTE_RUN_TO_TIME,inSig,outSig,&T);
        if (retVal != VSM_SUCCESSFUL)
        {
            printf("vsmCgRuntimeCommand() returns '%s': ", vsmCgGetLastErrorString(hSim));
            break; // Problem in sim, stop simulating
        }
        printf("T=%g:ST=%g: %g,%g\n",T,simTime, outSig[0],outSig[1]); // To do: make use of Embed results
    }
    return 0;
}

```


C Support Libraries

In addition to the program and utility files necessary to generate C, OBJ, EXE, and DLL files, Embed provides two C support libraries:

- CG32.LIB for the Windows platform
- SIMOBJ.LIB for simulation object generation

During installation, Embed places the C support libraries in the *<install-directory>\CG* and *<install-directory>\CG\LIB*, respectively.

Object files

Both C support libraries (CG32.LIB and SIMOBJ.LIB) contain a collection of object files that contain compiled instructions to support blocks for which there is no direct translation into C source code. These blocks include:

- atan2
- buffer
- crossDetect
- dotProduct
- embed
- error
- export
- import
- integrator
- invert
- limitedIntegrator
- map
- multiply
- pulseTrain
- resetIntegrator
- stateSpace

- stop
- timeDelay
- transferFunction
- transpose
- unitDelay
- vsum

SIMOBJ.LIB also contains API instructions to support code generation.

Targeting C code for unsupported platforms

The source code for the C support library (CG32.LIB) is required for the following reasons:

- To enhance the functionality of the C support library.
- To generate executable files to be run on processors other than the ones supported by the object code version of the C support library shipped with Embed. For example, to embed the source code library in a Hitachi chip, you need to recompile and relink the support library using a Hitachi compiler.

The source code for the CG32.LIB support library is a separate product that is not automatically included when you purchase Embed.

C support library source code

The source code for the C support library comprises the following files:

File name	Description
CG.C	Main driver routines
CG.H	Function prototypes
CGEN.H	Structure definitions
CGIO.C	File I/O
CGIO.H	Function prototypes
CROSSDET.C	Cross detection
CROSSDET.H	Function prototypes
FILEIO.C	File parsing
FILEIO.H	Function prototypes
IMPORT.C	File import/export
IMPORT.H	Function prototypes
MAT.C	Matrix operations
MAT.H	Function prototypes
MATDIV.C	Matrix divide
MATDIV.H	Function prototypes
READCC.TXT	ASCII text file containing additional technical information and corrections to the manual

SIMIO.H	Function prototypes
VCSRC.MAK	C code source makefile for Microsoft Visual Studio
XFER.C	Transfer function support
XFER.H	Function prototypes
UNIX.MAK	Make file for Unix platforms
VCSRC.MAK	Project for Microsoft Visual C

Compiling and linking the C support library source code

To compile and link the support library source code, you can use the makefile named SRC.MAK that was shipped with Embed. This makefile resides in the `<install-directory>\CG`.

Platform	The makefile is configured to use
Windows	Microsoft Visual Studio
UNIX	Gnu and native ANSI C compilers

To compile and link the support library source code

- Enter one of the following commands at the system prompt:

To use this makefile	Use
Microsoft C	Open the project VCSRC.MAK
Gnu C or native ANSI C	make -f unixsrc.mak

Extending the Arduino Block Set

Arduino libraries are a convenient way to add functionality to your Arduino embedded diagrams. There are hundreds of libraries that support common types of hardware, such as servo motors and LCD displays, as well as basic communication functions.

Embed includes several dozen diagrams that use Arduino libraries. Embed also provides an easy way to download libraries from the Arduino website.

You can also copy and paste Arduino sketch code directly into Extern Definition and Extern Function blocks.

Sample diagrams that use Arduino libraries

The sample diagrams that use Arduino libraries are located under Examples > Embedded > Arduino > External Library Import. The diagrams have been pre-configured using [Extern Definition](#) and [Extern Function](#), allowing you to easily download the modules and compile. The sample diagrams are listed in the table below.

Sample diagram using a library	Description
AdafruitOLED-Uno	Prints <i>Hello, world!</i> on an SSD1306 display
DHT11_sensor_library-Leonardo DHT11_sensor_library-Uno	Reads temperature and humidity data from a DHT11 sensor at 0.5Hz rate and prints them to serial output
EsploraAccelerometer-Mega	Reads accelerometer data from Esplora using Extern blocks and prints the data to serial output
LCDminiClick-Mega LCDminiClick-Uno	Communicates with the Arduino board over the SPI interface
LiquidCrystal-Uno	Controls the liquid crystal displays
MPU6050-Uno	Reads temperature, acceleration, and gyro data and angles from the MPU 6050 sensor and prints the data to serial output
Servo-Mega	Controls a motor by writing a position value to the servo
Ultrasonic-Mega UltrasonicBlink-Mega	Reads the distance of an object from the sensor and prints to serial output; the Blink diagram blinks a green or red LED if an object is greater than 10cm away

Importing libraries with the Arduino Library Manager

There are hundreds of libraries that you can import from the Arduino website using the Arduino Library Manager. You can also import libraries from other websites; however, these libraries may not conform to proper coding standards and therefore may not be easy to incorporate into your diagrams.

To use the Library Manager to install Arduino libraries

8. Go to **C:\Program Files (x86)\Arduino\libraries** and double-click **Arduino.exe** to start the Arduino IDE.
9. Click on **Sketch > Include Library > Manage Libraries**.
10. In the **Library Manager Search** box, enter the full or partial name of the library you are searching for.

Libraries that match the name are displayed in the lower window.

11. Select the library.
12. Click on **Install**. For some libraries, you will need to select the version before you can install it.

By default, the library is downloaded to **C:\Users\<user-name>\My Documents\Arduino\libraries**.

The library can now be set up and used in a diagram.

Setting up libraries imported with the Arduino License Manager

When importing Arduino libraries, there are several housekeeping tasks you should perform for the the code to run efficiently.

Using the Extern Definition and Extern Function blocks

After you import libraries, you use the [Extern Definition](#) block to set up the #include, #define, and instantiation declarations from the Arduino example sketch and the [Extern Function](#) block to integrate a setup loop in the diagram. See the [example](#) below.

Delay functions

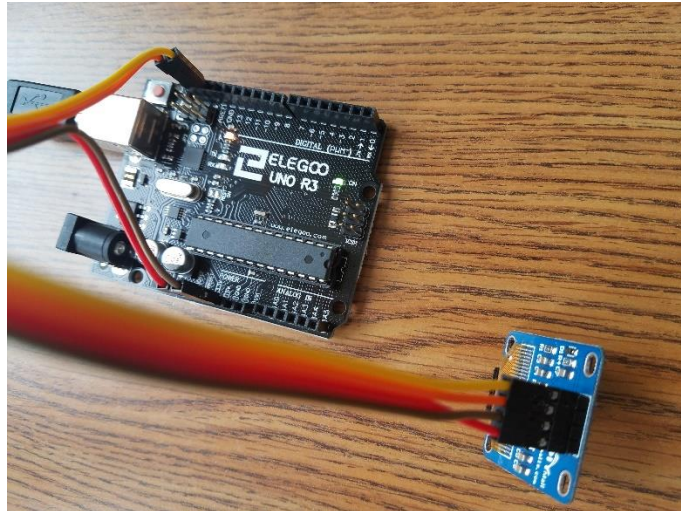
Many Arduino libraries contain Setup() and Loop() functions that contain one or more Delay functions. If the sum of length of the Delay functions is greater than the time step set in Embed (under Systems > Simulation Properties), Embed does not have time to execute one iteration of the library function. The most common way to deal with Delay functions is to encapsulate the library in a compound block and set it to execute as a Background Task with a local time step greater than the sum of all delays in the library.

Alternatively, you can add up the length of all delays in the library and set the Embed time step to a greater value.

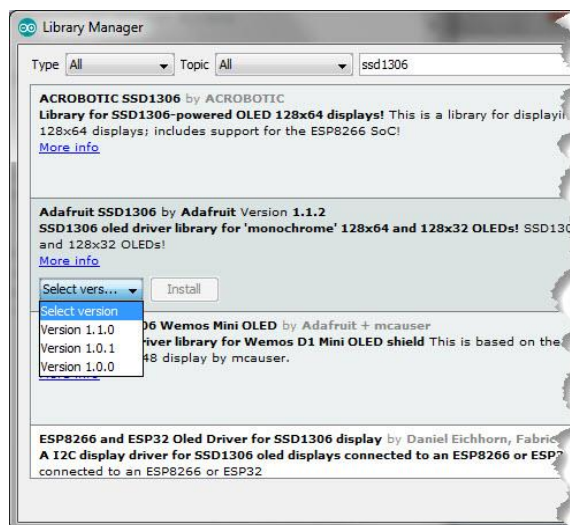
Example: Importing an Arduino library that displays text on an SSD1306

This example describes how to use the Adafruit SSD1306 driver along with the Adafruit GFX general-purpose graphics software to print "Hello, world!" on the SSD1306 on an Arduino Uno coupled with a 128x32-bit display connected via I2C. You can easily modify the steps for a 64-bit display or SPI connection.

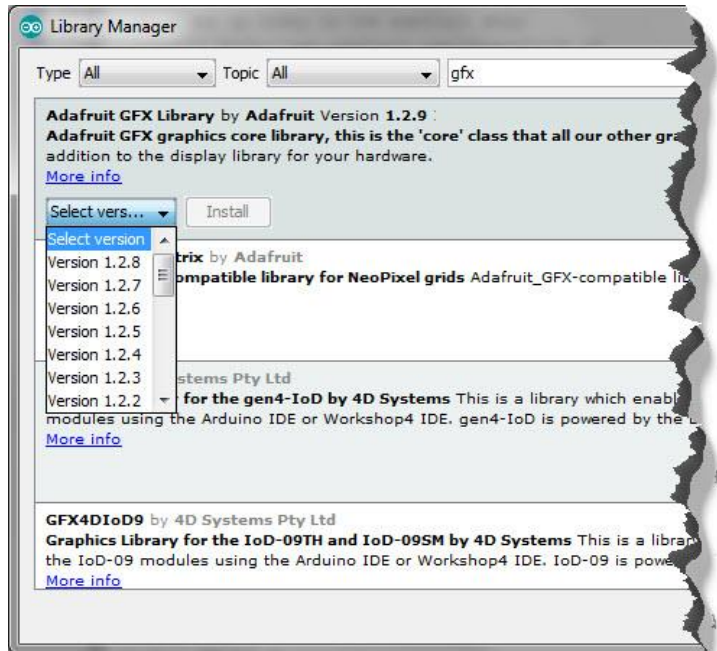
1. Attach the SSD1306 OLED to the Arduino Uno board as shown below. For wiring instructions, go to <https://learn.adafruit.com/monochrome-oled-breakouts/wiring-128x32-spi-oled-display>.



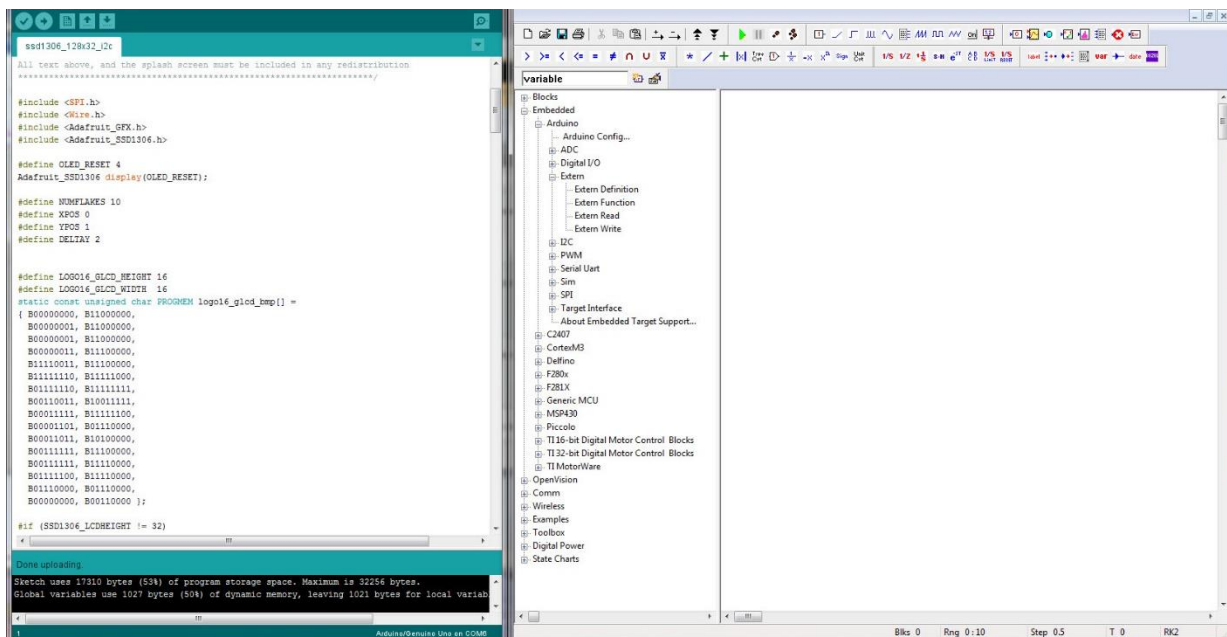
2. Start the Arduino IDE.
3. Click **Sketch > Include Library > Manage Libraries** and do the following:
 - a. In the **Search box**, enter **ssd1306**.



- b. Under **Adafruit SSD1306**, select the most recent version and click **Install**.
 - c. Repeat steps a - b but this time search for and install the most recent version of **Adafruit GFX library**.

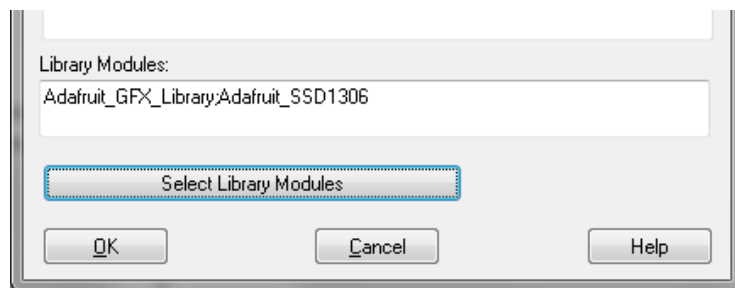


4. Click on **File > Open > Examples > ssd1306_128x32_i2c** and select **ssd1306_128x32_i2c.ino**.
5. To verify that the library modules have been installed correctly, compile the code from the Arduino IDE by clicking on the **checkmark** in the upper left corner of the Arduino window.
6. To verify that the hardware is connected properly and works as expected, click the **right arrow** in the upper left corner of the Arduino window to upload and run the code on your Arduino.
7. Start Embed and position it next to the Arduino window.

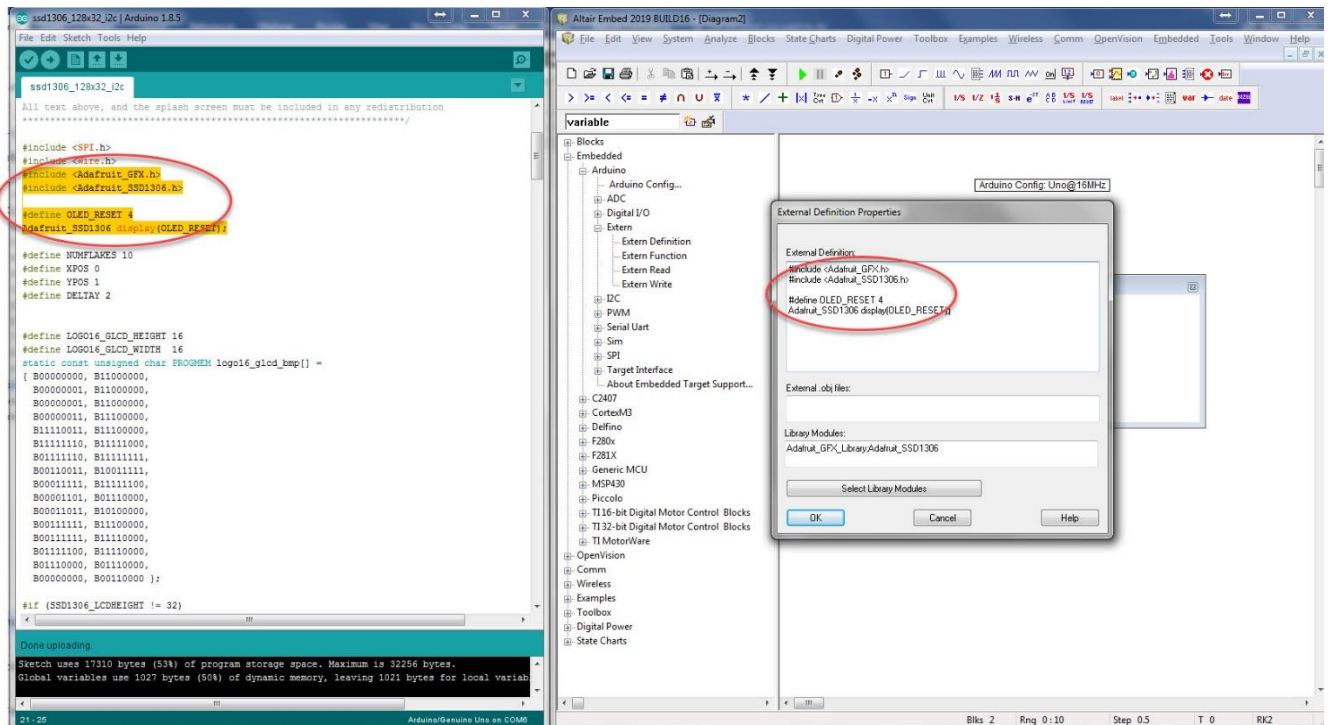


8. Create a new diagram and save it as **OLED2.vsm**.

9. Insert the following blocks into your diagram:
 - **Embedded > Arduino > [Arduino Config](#)** block. Make sure it is configured for an **Uno** and the **Comm port** is set correctly.
 - **Embedded > Arduino > Extern > [Extern Definition](#)** block in your diagram.
10. Right-click the **Extern Definition** block to access its Properties dialog box.
11. Click **Select Library Modules**.
12. In the **External Library Selection** dialog box, select **Adafruit_SSD1306** and **Adafruit_GFX_Library**, then click **OK**.
13. The **Extern Definition** dialog box displays the selected libraries under **Library Modules**.



14. With the Arduino window and Embed window side-by-side, copy the `#include`, `#define`, and instantiation declarations from the Arduino sketch into the External Definition window.



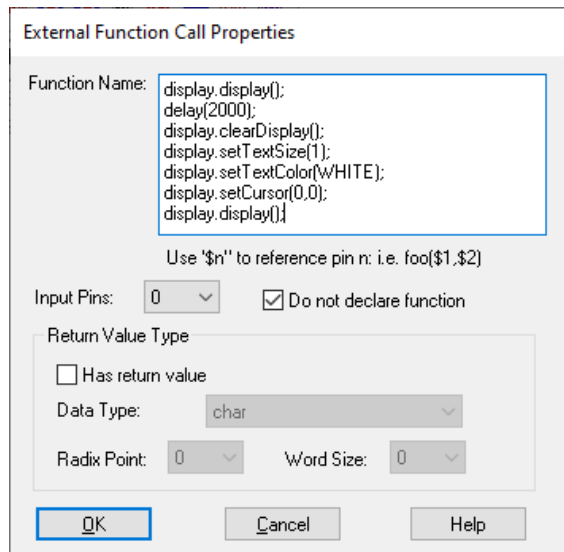
15. In the Extern Definition window, rename the **Adafruit_GFX.h** and **Adafruit_SSD1306.h** to **Adafruit_GFX.cpp** and **Adafruit_SSD1306.cpp**. The CPP files contain all the driver logic.

16. Click **OK**.

17. Integrate a setup loop into the diagram using the [Extern Function](#) block.

- a. Insert an **Extern Function** block into the diagram beneath the **Extern Definition** block.
- b. In the Arduino sketch, copy the following code into the Extern Function block under **Function Name**:

```
display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
display.display();
delay(2000);
display.clearDisplay();
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0,0);
display.display();
```



Your diagram will look like this:

Arduino Config: Uno@16MHz

```

Extern Definition

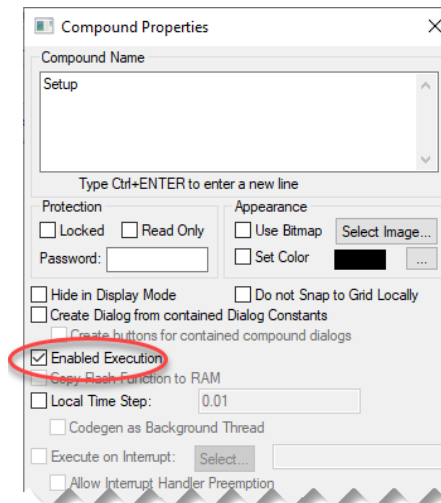
#include <Adafruit_GFX.cpp>
#include <Adafruit_SSD1306.cpp>

#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RESE
    
```

```

display.begin(SSD1306_SWITCHCAPVCC,0x3C);
display.display();
delay(2000);
display.clearDisplay();
display.setTextSize(1);
display.setTextColor(WHITE)
display.setCursor(0,0);
display.display();
    
```

- c. Encapsulate the **Extern Function** block in a **compound block** and name it **Setup**.
- d. CTRL-right-click over **Setup** and in the dialog box, activate **Enabled Execution** and click **OK**.



- e. Wire a **variable block** into **Setup** and set the **variable** block to **\$firstPass**. Because **Setup** runs only once at boot, the **\$firstPass** flag is used to control the enabled compound to run once at boot.

Arduino Config: Uno@16MHz

```
Extern Definition
#include <Adafruit_GFX.cpp>
#include <Adafruit_SSD1306.cpp>

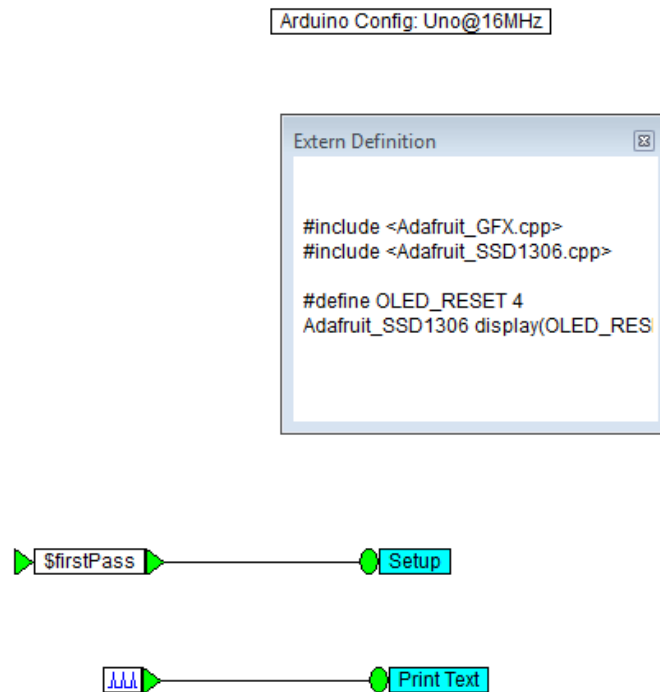
#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RES
```



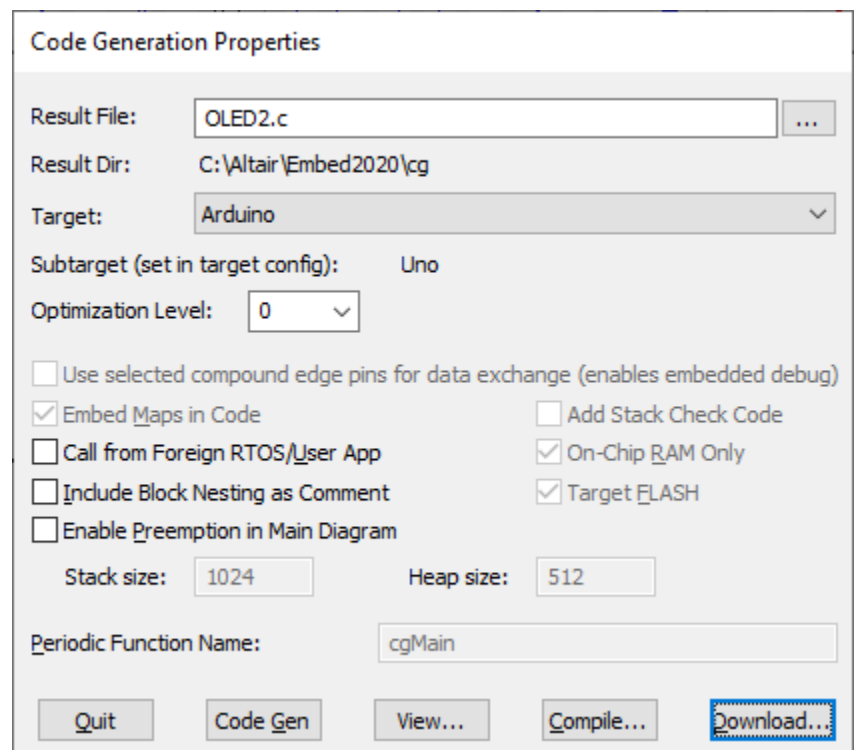
18. Add one more **Extern Function** block to the top level of your diagram.
19. In the Arduino sketch, copy the following code into the **Extern Function** block under **Function Name**:


```
Display.println("Hello, world!");
display.display();
```
20. Encapsulate the **Extern Function** block in a **compound block** and name it **Print Text**.
21. CTRL-right-click over **Print Text** and in the dialog box, activate **Enabled Execution** and click **OK**.
22. Wire a **pulseTrain** block into **Print Text** and set the **Time Delay** to 2s and the **Time Between Pulses** to 1. The pulseTrain block sets the frequency at which to print Hello, world!.

Your diagram will look like this:



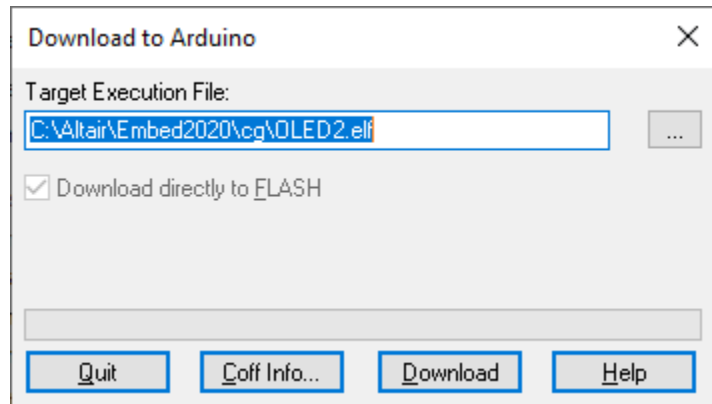
23. To compile the code for the Arduino, click **Tools > Code Gen.**



24. In the **Code Generation Properties** dialog box, click **Compile**.

25. The code is compiled in a DOS window. When the compilation completes, click **Download** in the Code Gen dialog box.

26. In the **Download to Arduino** dialog box, click **Download**.



The text **Hello, world!** is displayed on the **SSD1306**.



Arduino Pin Mapping

Arduino Connector	ATmega2560 (Mega 2560)	(ATmega32u4) (Leonardo)	ATmega168/328 (Uno)
Board Pin	Chip Port and Channel	Chip Port and Channel	Chip Port and Channel
Analog pin 0	PF0 (ADC0) Digital pin 54	PF7 (ADC7/TDI) Digital pin 18	PC0 (ADC0/PCINT8) Digital pin 14
Analog pin 1	PF1 (ADC1) Digital pin 55	PF6 (ADC6/TDO) Digital pin 19	PC1 (ADC1/PCINT9) Digital pin 15
Analog pin 2	PF2 (ADC2) Digital pin 56	PF5 (ADC5/TMS) Digital pin 20	PC2 (ADC2/PCINT10) Digital pin 16
Analog pin 3	PF3 (ADC3) Digital pin 57	PF4 (ADC4/TCK) Digital pin 21	PC3 (ADC3/PCINT11) Digital pin 17
Analog pin 4	PF4 (ADC4 /TMK) Digital pin 58	PF1 (ADC1) Digital pin 22	PC4 (ADC4/SDA/PCINT12) Digital pin 18
Analog pin 5	PF5 (ADC5 /TMS) Digital pin 59	PF0 (ADC0) Digital pin 23	PC5 (ADC5/SCL/PCINT13) Digital pin 19
Analog pin 6	PF6 (ADC6) Digital pin 60		
Analog pin 7	PF7 (ADC7) Digital pin 61		
Analog pin 8	PK0 (ADC8 /PCINT16) Digital pin 62		
Analog pin 9	PK1 (ADC9 /PCINT17) Digital pin 63		
Analog pin 10	PK2 (ADC10 /PCINT18) Digital pin 64		
Analog pin 11	PK3 (ADC11 /PCINT19) Digital pin 65		

Analog pin 12	PK4 (ADC12 /PCINT20) Digital pin 66		
Analog pin 13	PK5 (ADC13 /PCINT21) Digital pin 67		
Analog pin 14	PK6 (ADC14 /PCINT22) Digital pin 68		
Analog pin 15	PK7 (ADC15 /PCINT23) Digital pin 69		
Digital pin 0	PE0 (RXD0/PCINT8) (RX0)	PD2 (RX D1/AIN1/INT2) (RX)	PD0 (PCINT16/RXD) (RX0)
Digital pin 1	PE1 (TXD0) (TX0)	PD3 (TXD1/INT3) (TX)	PD1 (PCINT17/TXD) (TX0)
Digital pin 2	PE4 (OC3B/INT4) (PWM)	PD1 (SDA/INT1)	PD2 (PCINT18/INT0)
Digital pin 3	PE5 (OC3C/INT5) (PWM)	PD0 (OC0B/SCL/INT0) (PWM)	PD3 (PCINT19/OC2B/INT1) (PWM)
Digital pin 4	PG5 (OC0B) (PWM)	PD4 (ICP1/ADC8)	PD4 (PCINT20/XCK/T0)
Digital pin 5	PE3 (OC3A/AIN1) (PWM)	PC6 (OC3A/#0C4A) (PWM)	PD5 (PCINT21/OX0B/T1) (PWM)
Digital pin 6	PH3 (OC4A) (PWM)	PD7 (T0/OC4D/ADC10) (PWM)	PD6 (PCINT22/OC0A/AIN0) (PWM)
Digital pin 7	PH4 (OC4B) (PWM)	PE6 (INT.6/AIN0)	PD 7 (PCINT23/AIN1)
Digital pin 8	PH5 (OC4C) (PWM)	PB4 (ADC11/PCINT4)	PB0 (PCINT0/CLKO/ICP1)
Digital pin 9	PH6 (OC2B) (PWM)	PB5 (PCINT5/OC1A/#OC4B/ADC12) (PWM)	PB1 (OC1A/PCINT1) (PWM)
Digital pin 10	PB4 (OC2A/PCINT4) (PWM)	PB6 (PCINT6/OC1B/OC4B/ADC13) (PWM)	PB2 (SS/OC1B/PCINT2) (PWM)
Digital pin 11	PB5 (OC1A/PCINT5) (PWM)	PB7 (PCINT7/OC0A/OC1C/#RTS) (PWM)	PB3 (MOSI/OC2A/PCINT3) (PWM)
Digital pin 12	PB6 (OC1B/PCINT6) (PWM)	PD6 (T1/#OC4D/ADC9)	PB4 (MISO/PCINT4)
Digital pin 13	PB7 (OC0A/OC1C/PCINT7) (PWM)	PC7 (ICP3/CLK0/#0C4A) (PWM)	PB5 (SCK/PCINT5)
Digital pin 14	PJ1 (TXD3/PCINT10) (TX3)		
Digital pin 15	PJ0 (RXD3/PCINT9) (RX3)		
Digital pin 16	PH1 (TXD2) (TX2)		
Digital pin 17	PH0 (RXD2) (RX2)		
Digital pin 18	PD3 (TXD1/INT3) (TX1)		
Digital pin 19	PD2 (RXDI/INT2) (RX1)		
Digital pin 20	PD1 (SDA/INT1) (SDA)		
Digital pin 21	PD0 (SCL/INT0) (SCL)		
Digital pin 22	PA0 (AD0)		
Digital pin 23	PA1 (AD1)		
Digital pin 24	PA2 (AD2)		

Digital pin 25	PA3 (AD3)		
Digital pin 26	PA4 (AD4)		
Digital pin 27	PA5 (AD5)		
Digital pin 28	PA6 (AD6)		
Digital pin 29	PA7 (AD7)		
Digital pin 30	PC7 (A15)		
Digital pin 31	PC6 (A14)		
Digital pin 32	PC5 (A13)		
Digital pin 33	PC4 (A12)		
Digital pin 34	PC3 (A11)		
Digital pin 35	PC2 (A10)		
Digital pin 36	PC1 (A9)		
Digital pin 37	PC0 (A8)		
Digital pin 38	PD7 (T0)		
Digital pin 39	PG2 (ALE)		
Digital pin 40	PG1 (RD)		
Digital pin 41	PG0 (WR)		
Digital pin 42	PL7		
Digital pin 43	PL6		
Digital pin 44	PL5 (OC5C) (PWM)		
Digital pin 45	PL4 (OC5B) (PWM)		
Digital pin 46	PL3 (OC5A) (PWM)		
Digital pin 47	PL2 (T5)		
Digital pin 48	PL1 (ICP5)		
Digital pin 49	PL0 (ICP4)		
Digital pin 50	PB3 (MISO/PCINT3) (MISO)		
Digital pin 51	PB2 (MOSI/PCINT2) (MOSI)		
Digital pin 52	PB1 (SCK/PCINT1) (SCK)		
Digital pin 53	PB0 (SS/PCINT0) (SS)		

Arduino PWM Frequency Table

Timer	Prescaler	Arduino Target		
		Uno	Leonardo	Mega 2560
		Frequency(kHz)	Frequency(kHz)	Frequency(kHz)
Timer 0	1	62.5	62.5	62.5
	8	07.8125	07.8125	07.8125
	64	00.9765625	00.9765625	00.9765625
	256	00.244140625	00.244140625	00.244140625
	1024	00.061035156	00.061035156	00.061035156
Timer 1	1	Scheduler	31.25	Scheduler
	8		03.90625	
	64		00.48828125	
	256		00.122070313	
	1024		00.030517578	
Timer 2	1	62.5	Not present	62.5
	8	07.8125		07.8125
	32	01.953125		01.953125
	64	00.9765625		00.9765625
	128	00.48828125		00.48828125
	256	00.244140625		00.244140625
	1024	00.061035156		00.061035156
Timer 3	1	Not present	Scheduler	31.25
	8			03.90625
	64			00.48828125
	256			00.122070313
	1024			00.030517578

Timer 4	1	Not present	31.25	31.25
	2		15.625	
	4		07.8125	
	8		03.90625	03.90625
	16		01.953125	
	32		00.9765625	
	64		00.48828125	00.48828125
	128		00.244140625	
	256		00.122070313	00.122070313
	512		00.061035156	
	1024		00.030517578	00.030517578
	2048		00.015258789	
	4096		00.007629395	
	8192		00.003814697	
	16384		00.001907349	
Timer 5	1	Not present	Not present	31.25
	8			03.90625
	64			00.48828125
	256			00.122070313
	1024			00.030517578

Index

\$

\$isCodeGen flag 34

.

.M files
importing 249
.MAT files
importing 250

A

abs 230
Accessing the Chip Temp sample diagram 23
ACI Flux Estimator 201
ACI Motor 199
ACI Speed Estimator 202
ADC Config 176
ADC Config for Arduino 177
ADC Config for C2407 177
ADC Config for Delfino, F280X, and Piccolo 179
ADC Config for F280X – early versions 180
ADC Config for F281X 178
ADC Config for STM32 181
ADC10/12 77
ADC10/12 Config for MSP430 183
Adjusting C2000 and ARM Cortex M3 target
update time 37
Analog Comparator DAC 78
Analog Comparator DAC - Comparator Subsystem
(CMPSS) 79
Analog Comparator DAC – No Comparator
Subsystem 78
Analog In 80
Analog Input 80
Analog Input for Arduino, C2407, Delfino, F280x,
F281X, Piccolo 80
Analog Input for STM32 81
and 232
Angle Estimator 215

Arduino

Analog In 80
Analog Input 80
Digital Input 90
Digital Output 92
Extern Definition 106
Extern Function 107
Extern Read 109
Extern Write 110
GPIO In 113
I2C Read Buffer 120, 121
I2C Write Buffer 122
JSON 122
PWM 137
Serial UART Read 142
Serial UART Write 142, 143
SPI Read 146
SPI Write 149
Arduino Pin Mapping 309
Arduino PWM Frequency Table 313
ARM Cortex M3 Config 164
atan2 234
Automatic Data Update 160
Automatically Generating Executable Code 39

B

Blocks that generate stand-alone C code 43
Building a custom DLL 281

C

C code generation
compound blocks 41
from auto-generated DLLs 284
translation of variable names 41
with custom blocks 74
C Code Support Libraries
object files 295
C Support Libraries 295
C support library source code 296
compiling and linking 297
C2407
Analog In 80
Analog Input 80
CAN Receive 82
CAN Transmit 83
CAN Transmit Ready 84
Event Capture 105
Extern Definition 106
Extern Function 107
Extern Read 109
Extern Write 110
Full Compare Action 111
Full Compare PWM 111
Get CPU Usage 113

- GPIO In 113
- GPIO Input 114
- GPIO Out 116
- I/O Memory Read 120
- I/O Memory Write 120
- Monitor Buffer Empty 123
- Monitor Buffer Read 124
- Monitor Buffer Write 124
- PWM 137
- PWM 131
- Serial UART Read 142
- Serial UART Write 142, 143
- SPI Read 148
- Target Interface 152, 154
- C24x
 - Get CPU Usage 113
 - GPIO In 113
 - GPIO Input 114
 - GPIO Out 116
- C24X
 - Analog In 80
 - Analog Input 80
 - CAN Receive 82
 - CAN Transmit 83
 - CAN Transmit Ready 84
 - Target Interface 152, 154
- Calling a DLL from an Embed diagram 279
- Calling the generated code from a user application 74
- CAN Config 184
- CAN Receive 82
- CAN Transmit 83
- CAN Transmit Ready 84
- Check diagram parameters 72
- Checking for performance degradation 64
- Choosing the Web Interface Source 160
- Clarke Transform 202
- Code Gen command 50, 52
- Code generation considerations for low RAM targets 42
- Communicating with an embedded simulation object 291
- Communication interfaces 36
- Comparator 85
- Comparing simulation speed 280
- Compiling and linking the C support library source code 297
- Compiling the source diagram 25
- Completing the controller implementation 265
- Config commands
 - ADC 176
 - ARM Cortex M3 164
 - CAN 184
 - DMA 185
 - F28X 167
 - Generic MCU 169
 - GPIO 187
 - I2C 188
 - MSP 171
 - SD16 190
 - Serial UART 191
 - SPI 192, 194
- Configuration 159
- Configure the compound block to communicate with the target 41
- Configure the hardware and the diagram 67
- Configure the target 41
- Configuring a sample Web Server block 155
- Configuring input and output connector pins 156
- Configuring the Web Address 159
- Confirm that data can be printed to the serial monitor 69
- Confirm that the pushbutton is working 71
- const 234
- Constructing a floating-point model 266
- Constructing the control law 264
- Constructing the controller 261, 267, 271
- Constructing the door system 260
- Constructing the encoder 261
- Constructing the encoder feedback 263
- Constructing the fan-paddle-sensor 269
- Constructing the Fixed-Point Volts to Degrees Converter 270
- Constructing the gearbox 259
- Constructing the motor 257
- Constructing the open/close command 262
- Constructing the Volts to Degrees Converter 268
- Control 157
- Control execution order 61
- Controller Read Property 216
- Controller Write Property 217
- Controlling block execution 60
- Controlling code placement 65
- Controlling execution 62
- Controlling execution on embedded targets 60
- convert 235
- Converting to scaled fixed-point control 263
- Copying code from FLASH to RAM 60
- Cortex M3
 - Analog In 80
 - Analog Input 80
 - Digital Input 90
 - Digital Output 91
 - Extern Definition 106
 - Extern Function 107
 - Extern Read 109
 - Extern Write 110
 - Get CPU Usage 113
 - I2C Read Buffer 120, 121
 - I2C Write Buffer 122
 - Monitor Buffer Empty 123
 - Monitor Buffer Read 124

- Monitor Buffer Write 124
 - Serial UART Read 142
 - Serial UART Write 142, 143
 - SPI Read 148
 - SPI Write 149, 150
 - Target Interface 152, 154
 - Web Server 154
 - cos 235
 - CRC16 235
 - Create custom-rate functions 60
 - Creating a debug diagram 66
 - Creating a DLL 277
 - Creating a Simulation Object 289
 - Creating and executing interrupt handlers 61
 - Current Model 203
- D**
- DAC 85
 - DAC for Delfino, F280x, Piccolo 86
 - DAC for STM32 87
 - DAC12 88
 - Debugging code on Arduino, ARM Cortex M3, Linux, C2000, and STM32 targets 66
 - Debugging code on embedded targets 62
 - Debugging real-time analog waveforms using the Arduino serial port 73
 - Debugging techniques 62
 - Default Value 159
 - Delay functions 300
 - Delfino
 - Analog Comparator DAC 78
 - Analog In 80
 - Analog Input 80
 - CAN Receive 82
 - CAN Transmit 83
 - CAN Transmit Ready 84
 - DMA Enable 92
 - eCAP 93
 - eCAP Action 100
 - eCAP Action Write 101
 - eCAP Chopper 102
 - eCAP Force Action 102
 - eCAP Force Action Write 103
 - eCAP PWM 94
 - ePWM 95, 103
 - eQEP 104, 105
 - Extern Definition 106
 - Extern Function 107
 - Extern Read 109
 - Extern Write 110
 - Get CPU Usage 113
 - GPIO In 113
 - GPIO Input 114
 - GPIO Out 119
 - HRCAP 119
 - I2C Read Buffer 120, 121
 - I2C Write Buffer 122
 - JSON 122
 - Monitor Buffer Empty 123
 - Monitor Buffer Read 124
 - Monitor Buffer Write 124
 - Serial UART Read 142
 - Serial UART Write 142, 143
 - Sigma Delta Filter Module 144
 - SPI Read 148
 - SPI Write 149, 150
 - Target Interface 152, 154
 - Determine stack and heap use 42
 - Digital In 89
 - Digital Input 90
 - Digital Input for Arduino 90
 - Digital Input for Cortex M3, MSP430 90
 - Digital Motor Control block set *See* DMC block set
 - Digital Out 91
 - Digital Output 91
 - Digital Output for Arduino 92
 - Digital Output for Cortex M3, MSP430 91
 - Displaying Coff information 55
 - div 236
 - DLL generation 277
 - building custom DLLs 281
 - calling from a diagram 279
 - comparing simulation speed 280
 - creating 277
 - LIBC.LIB LINK warning 285
 - LINK messages that can be ignored 287
 - Out of Environment Space message 285
 - troubleshooting 285
 - verifying results 280
 - DMA Config 185
 - DMA Enable 92
 - DMC block set
 - ACI Flux Estimator 201
 - ACI Motor 199
 - ACI Speed Estimator 202
 - Clarke Transform 202
 - Current Model 203
 - Inverse Clarke Transform 203
 - Inverse Park Transform 204
 - Park Transform 204
 - Phase Voltage Current 204
 - PID Regulator 206
 - QEP Speed 205
 - Ramp Generator 206
 - Resolver Decoder 207
 - SMO Position Estimator 208
 - Space Vector Generator (Magnitude/Frequency) 209
 - Space Vector Generator (Quadrature Control) 210
 - Space Vector PWM 210

Speed Calculator 211
 V/Hz Profile Generator 212
 Downloading and debugging 27

E

eCAP 93
 eCap PWM 94
 Embed Viewer 20
 Embedded diagrams 34
 Embedded system prototyping methodology 273
 ePWM 95
 ePWM Action 100
 ePWM Action Write 101
 ePWM Chopper 102
 ePWM digital compare 99
 ePWM for simulation 103
 ePWM Force Action 102
 ePWM Force Action Write 103
 ePWM TRIPSEL Config 100
 eQEP 104
 eQEP for simulation 105
 equal to (==) 223
 Estimator Read Property 217
 Estimator Write Property 217
 Event Capture 105
 Event logging 75
 Examining signal values 63
 Examining waveforms 64
 Example: Importing an Arduino library that displays text on an SSD1306 300
 Execute initialization code at boot time 62
 Execution timing 63
 Extending the Arduino Block Set 299
 Extern Definition 106
 Extern Function 107
 Extern Read 109
 Extern Write 110

F

F280x
 Analog Comparator DAC 78
 Analog In 80
 Analog Input 80
 CAN Receive 82, 83
 CAN Transmit Ready 84
 DAC 86
 eCAP 93
 eCAP Action 100
 eCAP Action Write 101
 eCAP Chopper 102
 eCAP Force Action 102
 eCAP Force Action Write 103
 eCAP PWM 94
 ePWM 103

ePWM 95
 eQEP 104, 105
 Extern Definition 106
 Extern Function 107
 Extern Read 109
 Extern Write 110
 Get CPU Usage 113
 GPIO In 113
 GPIO Input 114
 I2C Read Buffer 120, 121
 I2C Write Buffer 122
 JSON 122
 Monitor Buffer Empty 123
 Monitor Buffer Read 124
 Monitor Buffer Write 124
 Serial UART Read 142
 Serial UART Write 142, 143
 SPI Read 148
 SPI Write 149, 150
 Target Interface 152, 154
 F280xHRCAP 119
 F281X
 Analog In 80
 Analog Input 80
 CAN Receive 82
 CAN Transmit 83
 CAN Transmit Ready 84
 DMA Enable 92
 Event Capture 105
 Extern Definition 106
 Extern Function 107
 Extern Read 109
 Extern Write 110
 Full Compare Action 111
 Full Compare PWM 111
 Get CPU Usage 113
 GPIO In 113
 GPIO Input 114
 I2C Read Buffer 120, 121, 122
 JSON 122
 Monitor Buffer Empty 123
 Monitor Buffer Read 124
 Monitor Buffer Write 124
 PWM 137
 PWM 131
 Quadrature Encoder 138
 Serial UART Read 142
 Serial UART Write 142, 143
 SPI Read 148
 SPI Write 149, 150
 Target Interface 152, 154
 F28X Config 167
 F28x Config for Delfino, F280X, and Piccolo 168
 F28x Config for F281X 169
 Fixed Point block set 221
 Fixed Point Block Set Configure command 256

- Fixed-Point blocks
 - 1/S 240
 - abs 230
 - and 232
 - atan2 234
 - const 234
 - convert 235
 - cos 235
 - CRC16 235
 - div 236
 - equal to 223
 - gain 237
 - greater than 227
 - greater than or equal to 227
 - less than 222
 - less than or equal to 223
 - limit 238
 - limitedIntegrator 240
 - merge 240
 - mul 241
 - negate 228
 - not 241
 - not equal to 226
 - or 242
 - PI Regulator 244
 - PID Regulator 243
 - sampleHold 244
 - shift 246
 - sign 247
 - sin 247
 - sqrt 248
 - sum 248
 - transferFunction 249
 - unitDelay 253
 - xor 255
 - Fixed-point implementation of PID position controller 270
 - Fixed-point implementation of the controller 262
 - Fixed-Point tutorials
 - Fixed-point implementation of controller 257
 - position control application 266
 - Flashing generated code with UniFlash 56
 - Floating-point implementation 257
 - Full Compare Action 111
 - Full Compare PWM 111
- G**
- gain 237
 - Generate and download code to run in FLASH in batch mode 53
 - Generate and download code to run in FLASH on Arduino, MSP430, and STM32 targets 52
 - Generate and download code to run in RAM on ARM Cortex M3, Linux, and C2000 targets 49
 - Generate web page 160
 - Generating and downloading code to target devices 49
 - Generating C code
 - flashing with UniFlash 56
 - integrating handwritten code 73
 - Generating code as preemptible background thread 61
 - Generating code from automatically-generated DLLs 284
 - Generating code from custom blocks 74
 - Generating DLLs 277
 - Generating Simulation Objects 289
 - Generic MCU
 - Analog In 80
 - Analog Input 80
 - Extern Definition 106
 - Extern Function 107
 - Extern Read 109
 - Extern Write 110
 - JSON 122
 - Generic MCU Config 169
 - Generic MCU target support 40
 - Get CPU Usage 113
 - Get Target Stack and Heap command 42, 197
 - GPIO Config 187
 - GPIO In 113
 - GPIO In C2407, Delfino, F280x, F281X, Generic MCU, Piccolo, STM32 113
 - GPIO In for Linux Raspberry Pi 113
 - GPIO Input 114
 - GPIO Input for C2407, Delfino, F280x, F281X, Generic MCU, Linux Raspberry Pi, Piccolo 114
 - GPIO Input for STM32 115
 - GPIO Out 116
 - GPIO Out for C2407, Delfino, F280x, F281X, Generic MCU, Piccolo, STM32 116
 - GPIO Out for Linux Raspberry Pi 116
 - GPIO Output 117
 - GPIO Output for C2407, Delfino, F280x, F281X, Generic MCU, Linux Raspberry Pi, Piccolo 117
 - GPIO Output for STM32 118
 - greater than 227
 - greater than or equal to 227
- H**
- Hall Sensor 119
 - Hardware-in-the-Loop simulation 37
 - Hardware-in-the-Loop simulations *See* HIL simulations
 - High power safety concerns 38
 - HIL simulations 73
 - Digital Input block 73
 - Digital Output block 73

HRCAP 119

I

I/O Memory Read 120
 I/O Memory Write 120
 I2C Config 188
 I2C Config for Arduino, Cortex M3, Delfino, F280X, Linux Raspberry Pi, Piccolo, and STM32 188
 I2C Config for MSP430 189
 I2C Read Buffer 120
 I2C Start Communication 121
 I2C Write Buffer 122
 Implementing a PID position controller 266
 Implementing an elevator door control system 257
 importing data
 transferFunction block 249
 Importing libraries with the Arduino Library Manager 299
 Integrating handwritten code with generated code 73
 Interfacing with code running on Arduino, ARM Cortex M3, Linux Raspberry Pi, C2000, and STM32 devices 34
 Interrupt handlers
 creating 61
 setting sample rate 61
 Introduction 1
 Inverse Clarke Transform 203
 Inverse Park Transform 204
 IP Address and Subnet Mask 160

J

JSON 122
 JTAG connectors 66
 JTAG Hotlink
 enabling 50, 52

L

less than 222
 less than or equal to 223
 limit 238
 limitedIntegrator (1/S) 240
 Linear System blocks
 transferFunction 249
 linearization 249
 linearization data
 .M file 249
 .M file 249
 .MAT file 250
 LINK warning about LIBC.LIB 285
 LINK warning messages that can be ignored 287
 Linux Raspberry Pi

I2C Read Buffer 120, 121
 Serial UART Read 142
 Serial UART Write 142, 143
 SPI Read 146
 SPI Write 149
 Target Interface 152

M

MAC Address 160
 MatLab
 importing data from 250
 Measuring CPU utilization 37
 Measuring stack and heap usage 65
 merge 240
 Model-Based Development with Embed 33
 Monitor Buffer Empty 123
 Monitor Buffer Read 124
 Monitor Buffer Write 124
 Monitoring register values 63
 Motor Control 218
 MQTT Publish 125
 MQTT Subscribe 126
 MSP Config 171
 MSP430
 ADC10/12 77
 Analog In 80
 Analog Input 80
 Comparator 85
 DAC12 88
 Digital Input 90
 Digital Output 91
 Event Capture 105
 Extern Definition 106
 Extern Function 107
 Extern Read 109
 Extern Write 110
 Get CPU Usage 113
 I2C Read Buffer 120, 121
 I2C Write Buffer 122
 opAmp 128
 PWM 137
 Read Target Memory 140
 SD16 140, 142
 SD16A 141
 segmentLCD 141
 Serial UART Write 142, 143
 SPI Read 148
 SPI Write 149, 150
 Target Interface 152, 154
 mul 241

N

New features 17
 not 241

not equal to (!=) 226

O

Object files 295
 Online and local help 18
 Op Amp 128
 OpenVision blocks 48
 or 242
 Out of Environment Space error message 285

P

Page Header 160
 Park Transform 204
 Phase Voltage Calc 204
 PI Regulator 244
 Piccolo
 Analog Comparator DAC 78
 Analog In 80
 Analog Input 80
 CAN Receive 82
 CAN Transmit 83
 CAN Transmit Ready 84
 DAC 86
 DMA Enable 92
 eCAP 93
 eCAP Action 100
 eCAP Action Write 101
 eCAP Chopper 102
 eCAP Force Action 102
 eCAP Force Action Write 103
 eCAP PWM 94
 ePWM 103
 ePWM 95
 eQEP 104, 105
 Extern Definition 106
 Extern Function 107
 Extern Read 109
 Extern Write 110
 Get CPU Usage 113
 GPIO In 113
 GPIO Input 114
 GPIO Out 116
 HRCAP 119
 I2C Read Buffer 120, 121
 I2C Write Buffer 122
 JSON 122
 Monitor Buffer Empty 123
 Monitor Buffer Read 124
 Monitor Buffer Write 124
 Serial UART Read 142
 Serial UART Write 142, 143
 Sigma Delta Filter Module 144
 SPI Read 148
 SPI Write 149, 150

 Target Interface 152, 154
 PID Regulator 206, 243
 Pin Labels, Type, In/Out, and Pin 156
 Preparing a diagram for code generation 41
 Processor-in-the-Loop simulation 34
 Professional and Basic editions 19
 Profile matching 65
 Prototyping the embedded control system 265
 PWM 129
 PWM for Arduino 129
 PWM for C2407, F2812 131
 PWM for Linux Raspberry Pi 129
 PWM for MSP430 131
 PWM for simulation 137
 PWM for simulation for Arduino 137
 PWM for simulation for C2407, F281X, MSP430,
 STM32 137
 PWM for simulation for Linux Raspberry Pi 138
 PWM for STM32 132

Q

QEP Speed 205
 Quadrature Encoder 138
 Quadrature Encoder for F281X 138
 Quadrature Encoder for STM32 139
 Quick Start 23

R

Ramp Generator 206
 Raspberry Pi
 I2C Write Buffer 122
 JSON 122
 Read and write directly to device registers 61
 Read Target Memory 140
 Recording event statistics 63
 Reset Target command 197
 Resolver Decoder 207
 Resources for learning Embed 18
 Resources used by targets 39
 Running generated code on HIL hardware 73
 Running the diagram and viewing results 30

S

Sample diagrams 18
 Sample diagrams that use Arduino libraries 299
 Sample file with simulation object interface 292
 Sample rate
 target sampling too fast 61
 sampleHold 244
 SC16 Config 190
 SD16 140
 SD16A 141
 segmentLCD 141

- Serial UART Config 191
- Serial UART Read 142
- Serial UART Write 142
- Set PWM Mode 143
- Set the sample rate for the target application 61
- Setting diagram parameters 30
- Setting state chart breakpoints 63
- Setting up DCO and external clocks 172
- Setting up libraries imported with the Arduino
 - License Manager 300
- shift 246
- Sigma Delta Filter Module 144
- sign 247
- Similarities and differences between 16-bit and 32-bit TI DMC block 199
- Simulating with a debug diagram 67
- Simulation object generation 289, 299
- sin 247
- SMO Position Estimator 208
- Software-in-the-Loop simulation 33
- Source and debug diagrams for Arduino, ARM Cortex M3, Linux Raspberry Pi, C2000, and STM32 targets 35
- Space Vector Generator (Magnitude/Frequency) 209
- Space Vector Generator (Quadrature Control) 210
- Space Vector PWM 210
- Special-purpose add-on modules 20
- Specifying a local step size and local bounds 60
- Speed Calculator 211
- Speed considerations 41
- SPI Config 192, 194
- SPI Read 146
- SPI Read for Arduino 146
- SPI Read for C2407, Cortex M3, Delfino, F280x, F281X, MSP430, Piccolo, STM32 148
- SPI Read for Linux Raspberry Pi 146
- SPI Write 149
- SPI Write for Arduino 149
- SPI Write for C2407, Cortex M3, Delfino, F280x, F281X, MSP430, Piccolo, STM32 150
- SPI Write for Linux Raspberry Pi 149
- sqrt 248
- Standard blocks 43
- StateChart elements 48
- state-space matrices 249
- STM32
 - JSON 122
 - PWM 137
 - Serial UART Read 142
 - Serial UART Write 142, 143
 - SPI Read 148
 - SPI Write 150
 - Target Interface 152
- sum 248
- Support library 56

T

- Target algorithm
 - controlling execution order 61
 - custom-rate functions 60
 - developing 41
 - executing initialization code at boot time 62
 - reading to/writing from device registers 61
 - speed considerations 41
 - tuning 66
 - validating 66
- Target devices with no file system 41
- Target Interface 152
- Target Interface menu
 - Get Target Stack and Heap command 50, 52
- Target resources managed by Embed 40
- Target sampling too fast 61
- Target support 39
 - Arduino 49
 - ARM Cortex M4 40
 - ARM M4 49
 - Delfino 49
 - F280X 49
 - F2812 49
 - Generic MCU 40
 - LF2407 40, 49
 - Piccolo 49
- Target support
 - ARM Cortex M3 40
 - ARM M3 49
- Targeting C code for unsupported platforms 296
- Targets with no floating-point unit 41
- Technical support 18
- Texas Instruments Digital Motor Control block set
 - See DMC block set
- The Altair Embed product family 19
- Tools menu
 - Code Gen command 50, 52
- transferFunction 249
- transferFunction block 249
- Troubleshooting 285
- Tutorials 257

U

- UDP Read 151
- UDP Write 152
- Uniflash 56
- unitDelay 253
- Use existing web page 161
- Use existing web site 161
- Using ADC Config 176
- Using Arduino Config 163
- Using ARM Cortex M3 Config 164
- Using CAN Config 184
- Using DMA Config 185

- Using ESP8266WiFi Config 186
- Using Extern Definition and Extern Function blocks to add a C function to your diagram 74
- Using Extern Read and Extern Write blocks to merge your code 74
- Using F28X Config 167
- Using Generic MCU Config 169
- Using GPIO Qualification 187
- Using I2C Config 188
- Using MSP430 Config 171
- Using SD16 Config 190
- Using serial monitor to debug code on Arduino targets 67
- Using Serial UART Config 191
- Using SPI Config 192
- Using SPI Config for Arduino 194
- Using SPI Config for Linux 196
- Using STM32 Config 175
- Using the code generation parameters 53
- Using the createSim function 291
- Using the Extern Definition and Extern Function blocks 300
- Using the F240X Config 167
- Using the Fixed Point Block Set 221
- Using the Get Target Stack and Heap command 197
- Using the Linux Config 165
- Using the Peripheral Config blocks 176
- Using the Reset Target command 197
- Using the Target Config blocks 162
- Using the Target Interface commands 197
- Using the target support blocks 77
- Using the Target Support Blocks and Commands 77
- Using the TI DMC Block Set 199
- Using the TI MotorWare Block Set 215
- Using the vsmCgGetLastErrorString() 292
- Using the vsmCgRuntimeCommand 291
- Using the Web Server block in a diagram 161

V

- V/Hz Profile Generator 212
- Variable names 41
- Verifying DLL results 280
- Videos 18
- VSMDLL32.BAT 285

W

- Watch Dog 154
- Web Page Labels 158
- Web Server 154
 - configuring 155
 - example 161
 - Generate Web Page option 160

- Get Data Automatically 160
- input and output pins 156
- IP address 160
- MAC address 160
- Page Header 160
- Subnet Mask 160
- Use Existing Web Page option 161
- Use Existing Web Site option 161
- web address 159
- Web Interface Source 160
- Window Header 160
- What you get with Embed 16
- Window Header 160

X

- X (negate) 228
- xor 255