Embedded Application Development

sT Embed Training

Ric Kolk
Altair Engineering
rkolk@altair.com

Topics:

- Software Installation
- Software Installation Video
- Source & Debug Models
- sTE Real Time Operating System (RTOS)
- F28069M LaunchPad
- LED Blink
  - Fixed frequency
  - User set variable frequency
  - Measure Blink ON time
- Hello World LED Blink Video
- Host to Target Communication
- Displaying CPU Usage
- Monitor Buffer
  - Waveform Capture & Real Time Results
  - Oscilloscope Display
- Fixed Point Arithmetic
  - CPU Utilization Example
- Clock Speed, timers, interrupts
- ADC
  - Configuration
  - SOCx Setup
  - Extern Functions, Read, and Write
  - Chip Temperature Example
- 5 Wire Encoder
  - Configuration
  - Encoder Test Model Example
- Order of Execution
- Chip Temperature on the F28069M
- Motor Position Control on the F28069M

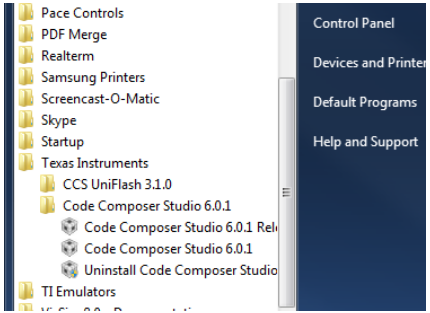
Software Installation

Software installation consists of the following two steps:

Step 1: Install the “code composer” software from Texas Instruments available at the following link:

http://processors.wiki.ti.com/index.php/Download_CCS#Code_Composer_Studio_Version_6_Downloads

Select the "Off-line Install“ and install with all recommended options. After completing, verify the installation by going to the Start menu and confirming you see the following (right)



Step 2: Install “solid Thinking Embed” software available at the following link:

http://www.vissim.com

Confirm VisSim/Embedded has been installed correctly;

NOTE: The following link contains information on the Texas Instrument Launchpad
http://www.ti.com/ww/en/launchpad/launchpads-c2000-launchxl-f28069m.html

Source & Debug Models for Embedded Control

Two types of Embedded Models: Source & Debug

Source Model: A "*.vsm"* model that is CodeGen'd, Compiled, and Downloaded to the target. The Source model executes on the Target with no communication to the Host PC.

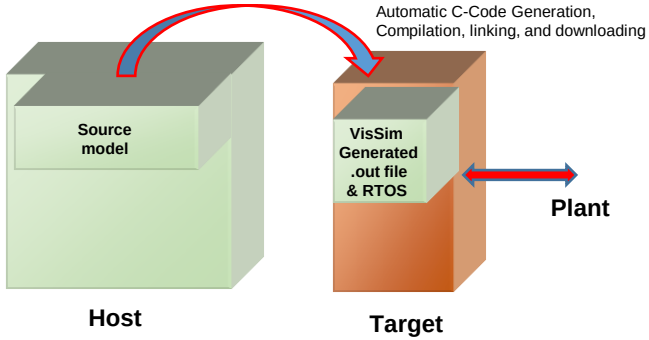
The Target Update time is controlled by the sTE"*Time Step"* value ("*System/System Properties/Range"* menu)

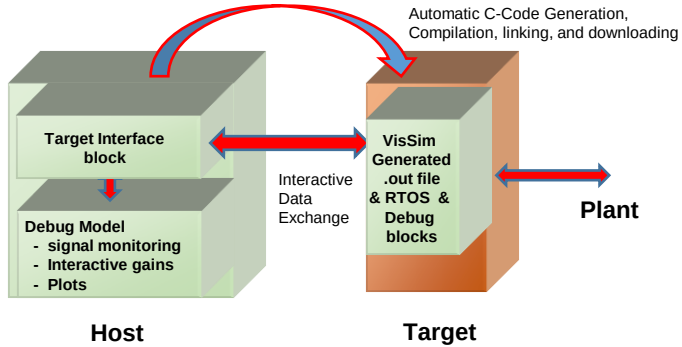
Debug Model: A "*.vsm"* model, part of which is executed on the Target and communicates, in real time, with the remaining part of the Debug model, residing on the Host PC.

The Debug Model part residing on the Host contains a "*Target Interface Block"* whose inputs and outputs communicate with the Target through the Interactive Data Exchange.

The Target Update time is controlled either by "*Time Step"* or as a parameter in the "*Target Interface Block"*.

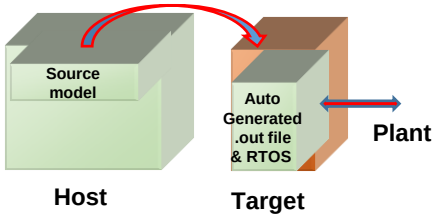
Normally the Debug Model name is the Source Model name appended with "*–d"* (for debug), ex; the source model *myModel.vsm* would have a debug model named *myModel-d.vsm*





When & How to use Source & Debug Models

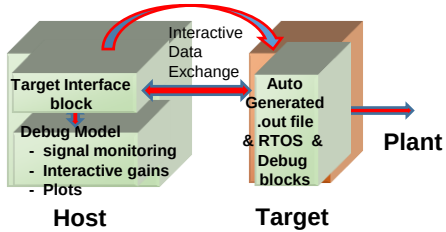
Without Interactive Data Exchange:



ONLY a Source Model is needed:

1. Create the target algorithm as a **Source Model**
2. CodeGen, Compile, and Download the **Source Model** to the Target.
3. The Target will begin executing the Source Model immediately.

With Interactive Data Exchange:



BOTH a Source Model and a Debug Model are needed:

1. Create the target algorithm as a compound block in a **Source Model.** Add Input and Output pins to the compound block to send and receive data from the Host to the Target
2. CodeGen and Compile the **Source Model** to create an executable (.out) file
3. Create a **Debug Model** consisting of a "Target Interface" block configured to read the executable (.out) file from step 2. Onceconfigured, the "Target Interface" block will have the same input and output pins defined in Step 1. You can connect these with "signal producers" and "signal consumers" to sendcommands to the target or plot or display data from the target. Click "Go" in the **Debug Model** to initiate target execution.

No Interactive Data Exchange: Use **Source Model** ONLY

Interactive Data Exchange: Use both a **Source Model** and a **Debug Model**

Real Time Operating System (RTOS)

An **Operating System (OS)** is software that manages a computer's memory, application processes, communication, I/O, and all software and hardware residing on the computer. A **Real-Time Operating System (RTOS)** is an OS that services real time application process data as it occurs with minimal buffering delays.

Terminology & Key Features:

- Process: A computer program that is executed as one or more threads.
- Task: Future promise to perform a process.
- Thread: Smallest sequence of programmed instructions that can be managed by an RTOS. Tasks are executed as one or more threads.
- Thread Switching Latency: Time required for the RTOS to switch executions between threads.
- Jitter: the variability in time required by the RTOS to accept and complete tasks.
- Hard vs. Soft RTOS: A hard RTOS has less jitter than a soft RTOS.
- Hard RTOS: Accepts and completes an application's task deterministically in time.
- Soft RTOS: Accepts and completes an application's task with variability in time.
- Interrupts: An event signal, from hardware or software, that requires immediate attention.
- Interrupt Latency: Time required for the RTOS to act on an interrupt.

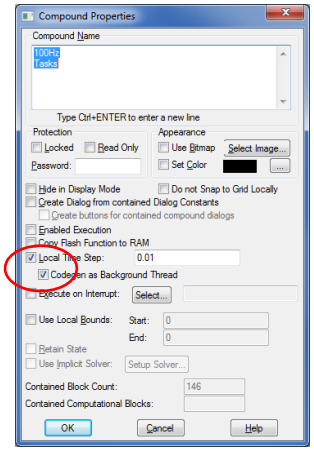
Solid Thinking EMBED RTOS Features:

- Main-Timer 2 control thread runs at rate as set in diagram "*System Properties…"*
- Unlimited number of preemptable (high jitter) background threads – (option in *Compound block*)
- Efficient device drivers for on-chip peripherals
- Handle interrupts directly in sTE (option in *Compound block*)
- Interrupt based soft queued I/O for serial, SPI and I2C
- Instrument individual subsystems for CPU usage

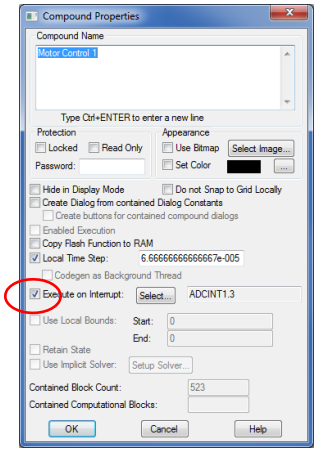
Solid Thinking EMBED RTOS Motor Control Setup

Typical Motor Control Model – Thread Architecture:

Background tasks are captured in 100Hz block



Time critical control operations are captured in an interrupt driven block



F28069M – LaunchPad Develop Kit

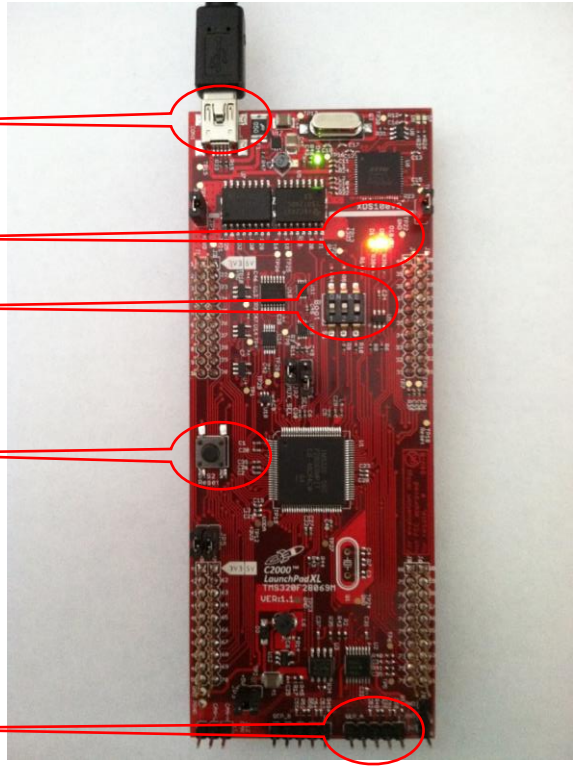
USB connection (used for JTAG communications and power)

LED's (red and blue)

All three microswitches must be set in the upward facing direction

Reset Button

Quadrature Encoder Input (2 channels)



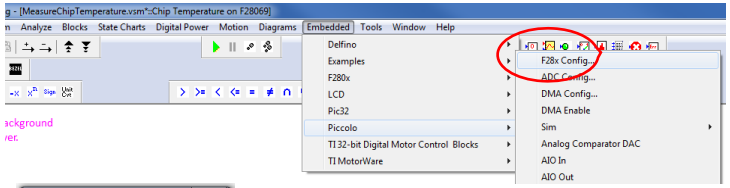
Unless otherwise noted, all examples in this presentation will use the Texas Instrument C2000 F28069M LaunchPad Development Kit (TI Part Number: LAUNCHXL-F28069M) shown below: http://www.ti.com/ww/en/launchpad/launchpads-c2000-launchxl-f28069m.html

LED Blink Example

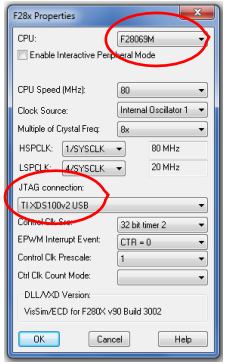
The LED Blink model created in this example is a Source model with no interactive data exchange.

One model is created in this example
Source Model: "*BlinkLED.vsm"*
Debug Model: Not Needed

Step 1. Source Model Creation "*BlinkLED.vsm".* From the "*Embedded/Piccolo"* menu, select and place an "*F28x Config…"* block in the model.



Configure the "*F28x Config …"* block as shown to the right, make sure the "*CPU = F28069M*" and the JTAG connection = "T1XDS100v2USB", other settings may be left at their default values (right).

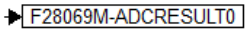


Click "*OK"* and the *"F28x Properties"* block will look like (below).

F28x Config: F28069M@80MHz
TI XDS100v2 USB

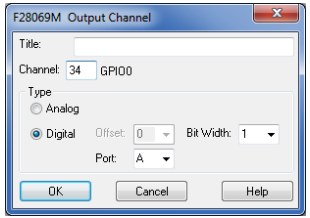
LED Blink – Diagram Construction

From the "*Embedded/Piccolo"* menu, select and place a "*Digital Output for F280x"* block in the model. (right)

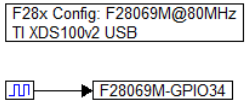


NOTE: The F28069M board has two LED's accessed through the following channel and port information;
Red LED = Channel 34, Port B
Blue LED = Channel 39, Port B
NOTE: Port A has 32 bits, so channel 34 is on Port B

Right click on the *"F28069M-ADCResult0"* block to expose the parameters, configure the block for output to the Red LED by setting the "*Type = Digital:" a*nd select "*Channel: 34 GPIO0"* and "*Port: A"*. The "*Title"* entry can be left blank and all other settings may be left at their default values (right) Click "*OK".*



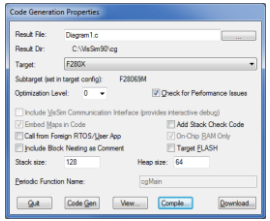
Attach a "*square wave"* to the digital output channel. Set the "*Frequency"* to 0.5 Hz. The completed "*BlinkLED.vsm"* model is shown to the right.



Set the Target update time under the menu "*System/System Properties/Range*" to the desired value, we will select "*Time Step*" = .0001 sec

LED Blink – CodeGen & Target Execution

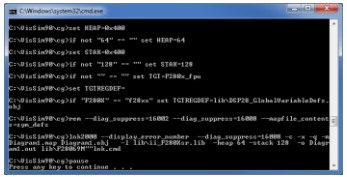
Step 2. Code Generation - Lasso all the blocks in "*BlinkLED.vsm"* model and then;



Step 2a. Click "*Tools/Code Gen"*

Step 2b. Configure the Code Generation Properties (right) as shown.

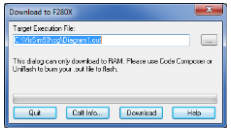
Target is the Launchpad microprocessor family, F280X



Step 2c. Click "*CodeGen" , "Compile…"*

In this step the "*.out*" (executable) file is created. You'll see the compile progress in a DOS window that requires you to "*Press any key to continue…"*

Step 2d. A *Download to f280X* window will appear, click "*Download".* This loads the "*.out"* file to the target. The Target will begin executing immediately.



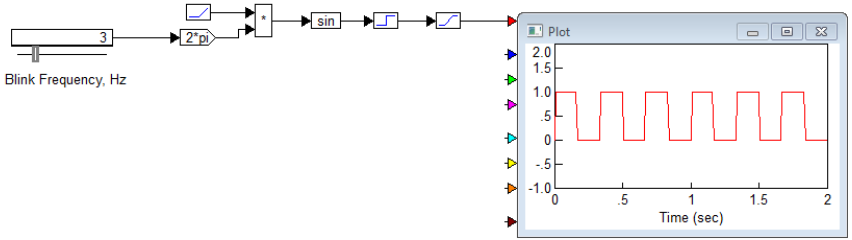
In two steps you have generated code to blink the LED's on the *Target*

View source model in VisSim

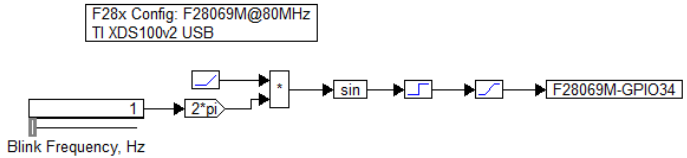
Host to Target Communication Example

This example illustrates interactive data exchange which allows the user to control the red LED blink frequency on the Target using a "*slider"* block on the Host.

Two models are created in this example
Source Model: "*BlinkLEDwithControlledFrequency.vsm"*
Debug Model: "*BlinkLEDwithControlledFrequency-d.vsm"*



Step 1: Source Model **"***BlinkLEDwithControlledFrequency.vsm" .*
Add and configure the "*F28x Config …"* block. A square wave generator model is created using a "*slider"* block to control the frequency. The "*slider* " output is multiplied by "wt", passed through a "*relay*" and then limited to lie between 0 and 1 which creates the square wave.



The "*slider"* controls the desired blink frequency, in Hz, and is configured to provide frequencies between 1 and 10 Hz by setting the *Lower Bound = 1* and *Upper Bound = 10*.

The square wave signal is connected to a "*Digital Output*" block configured to light the red LED (channel 34). Define the Target calculations in a compound block "*Target Calculations*".

View source model in VisSim

Host to Target Communication Example– Code Generation

Set the Target update time under the menu "*System/System Properties/Range*" to the desired value, we will select; "*Time Step*" = .0001 sec

Step 2. Code Generation - Lasso the "*Target Calculations*" compound block and then;

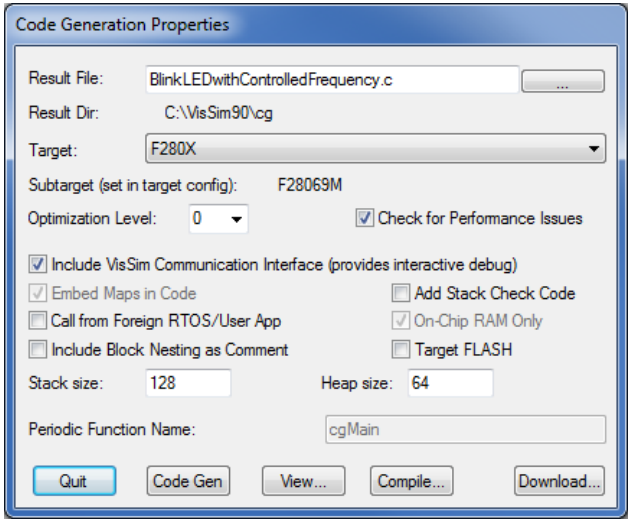
Step 2a. Click "*Tools/Code Gen"*

Step 2b. Configure the Code Generation Properties (right) as shown. Make sure the "*Include VisSim Communication.."* option is checked.

Step 2c. Click "*CodeGen" , "Compile…" , "Quit"*

This step will create the "*.out*" file named as the "*Result File*" with the extension "*.out*" and place the file in the "*Result Dir*" which defaults to "C:\VisSim90\cg".

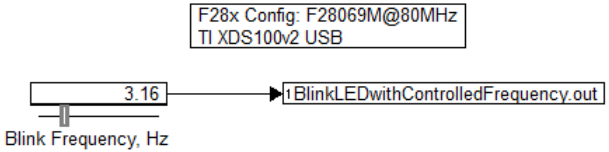
Step 2d. At this point you are finished with the "*BlinkLEDwithControlledFrequency.vsm"* source model, make sure it is saved.



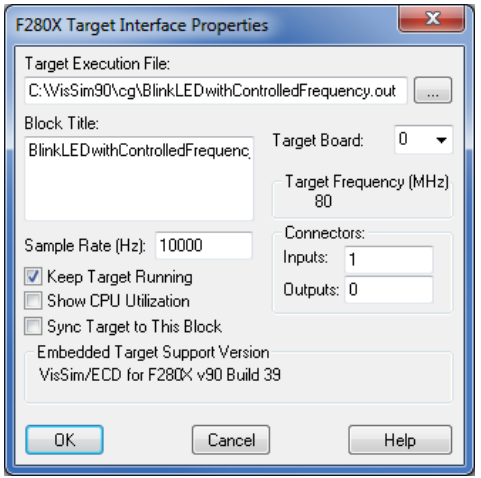
Host to Target Communication Example– Execution

Step 3. Debug Model - Create the debug model by renaming the source model to "*BlinkLEDwithControlledFrequency-d.vsm".*

Edit the Debug model and delete or disconnect the "*Target Calculations"* compound block. In its place, add a "*Target Interface*" block from the ("*Embedded/Picollo/Target Interface")* menu. The "*Target Interface*" block will have the same input and output pins as specified in the "*Target Calculations*" compound block in the source model. Connect the "*slider*" to the input pin of the "*Target Interface*" block.



Configure the "*Target Interface*" block:
"*Target Execution file*" = ".out" file created in Step 2c.
"*Sample Rate (Hz)"* = desired value (it defaults to 1/"*Time Step*" value specified in the source model.
"*Keep Target Running*" = checked to keep target running after VisSim has been stopped.



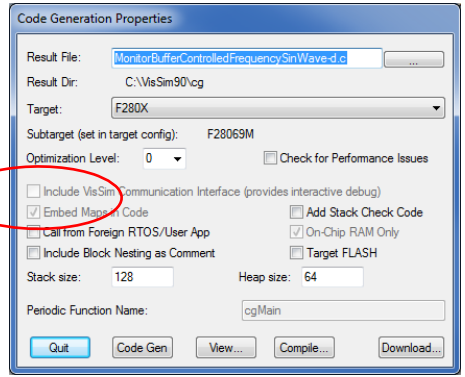
Click "*Go*", and, after a brief handshake, the Target will begin executing blinking the red LED at the frequency specified by the "*slider*".

View debug model in VisSim

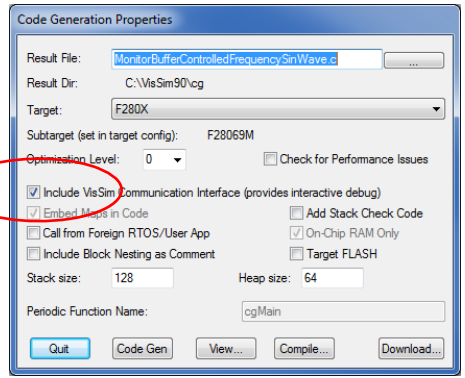
Compound Blocks and CodeGen

If you select the “Include VisSim Communication Interface” then VisSim will generate code ONLY for the selected compound block.

1. If a single compound block is not selected, the “Include VisSim Communication Interface” option will be greyed out. VisSim will generate code for the entire model.

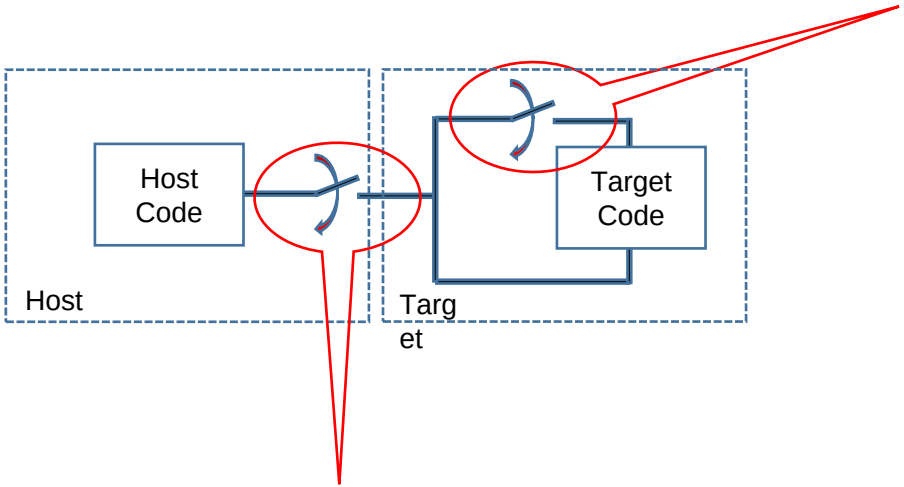


2. If one compound block is selected and the “Include VisSim Communication Interface” is checked, VisSim will generate code for the selected compound block.



Target Update Time

The Target code is executed at the "*Time Step*" value specified in the Source model used to produce the C Code. After compilation the Target "*Sample Rate (Hz)"* value specified in the "*Target Interface*" block in the Debug model will default to 1/"*Time Step*" value specified for the C Code generation, however, it can be modified.



JTAG interface updates at approximately 100Hz. This limits the real time execution of the Host

Three ways to control the Target Update Time:

1. "*Time Step*" in "*System/System Properties…"* sets the update time of the Target from the Source model.

2. "*Sample Rate (Hz)"* in "*Target Interface*" may be used to change the Target update rate during execution of the Debug model.

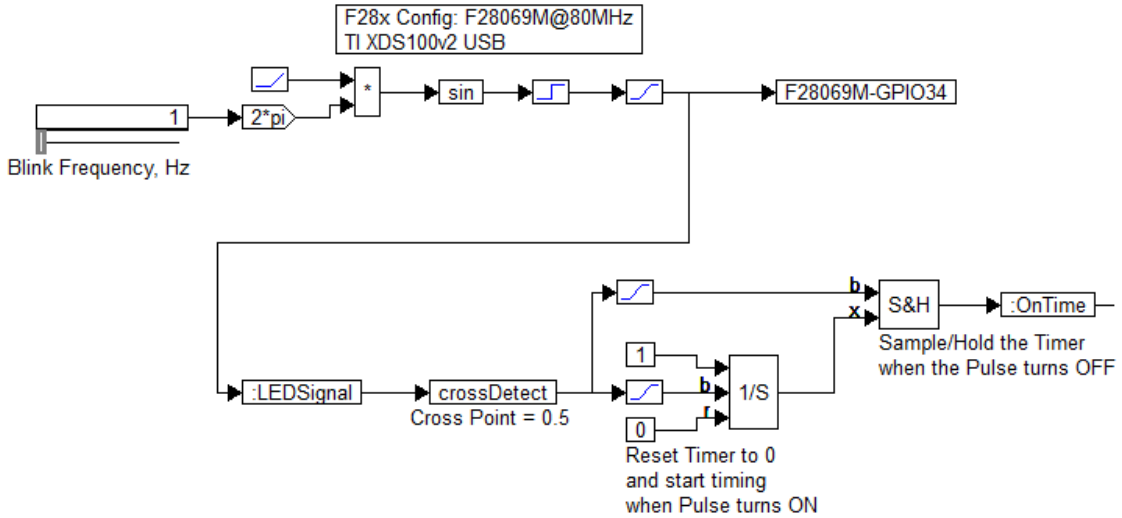
3. "Local Time Step" in Compound block properties. (NOTE: The compound block rate setting holds for all compounds, EXCEPT the topmost one that is selected for "Include Communication Interface")

Host To/From Target Communication Example

This example illustrates bi-directional interactive data exchange between the Host and Target. A "*slider*" block is used to control the red LED blink frequency and a "*plot*" block is used to display the LED "*OnTime*" both on the Host.

Two models are created in this example
Source Model:
"*BlinkLEDwithControlledFrequencyAndOnTimeCalculation.vsm*"
Debug Model:
"*BlinkLEDwithControlledFrequencyAndOnTimeCalculation-d.vsm*"

Step 1: Source Model
"*BlinkLEDwithControlledFrequencyAndOnTimeCalculation.vsm*" .
Add and configure the "*F28x Config …*" block. Add the square wave generator from the previous example. Add the "*OnTime*" calculation using a reset integrator (below)

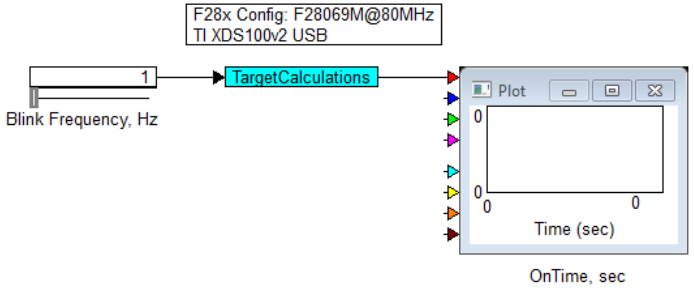


View source model in VisSim

Host To/From Target Communication Example- Codegen

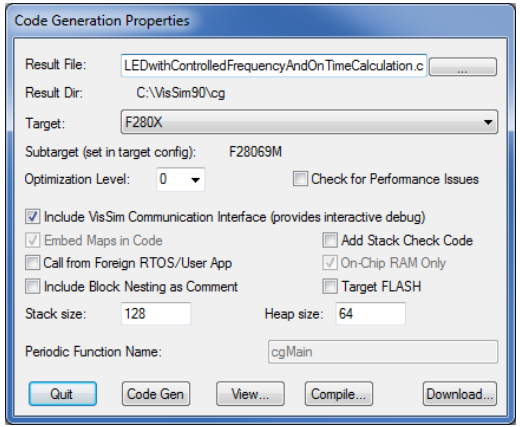
The Target calculations are captured in a compound block named *"TargetCalculations"* (right).

Set the Target update time under the menu "*System/System Properties/Range*" to the desired value, we will select;
"*Time Step*" = .0001 sec



Step 2. Code Generation - Lasso the "*Target Calculations*" compound block and then;

Step 2a. Click "*Tools/Code Gen"*
Step 2b. Configure the Code Generation Properties (right) as shown. Make sure the "*Include VisSim Communication.."* option is checked.
Step 2c. Click "*CodeGen" , "Compile…" , "Quit"*
This step will create the "*.out*" file named as the "*Result File*" with the extension "*.out*" and place the file in the "*Result Dir*" which defaults to "C:\VisSim90\cg".
Step 2d. At this point you are finished with the "*BlinkLEDwithControlledFrequencyAndOnTimeCalculation.vsm"* source model, make sure it is saved.

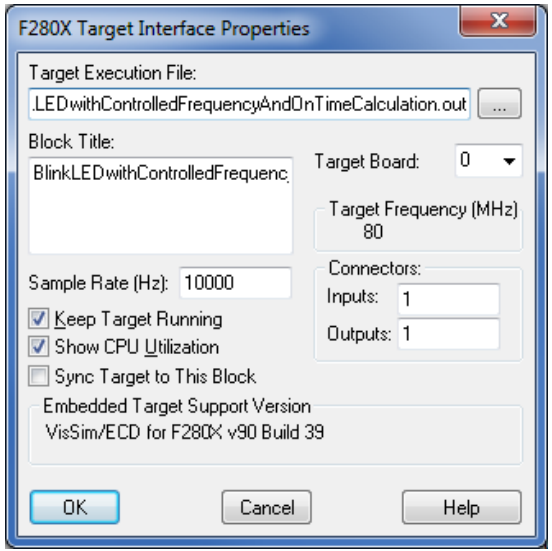


Host To/From Target Communication Example- Execution

Step 3. Debug Model - Create the debug model by renaming the source model to "*BlinkLEDwithControlledFrequencyAndOnTimeCalculation-d.vsm"*

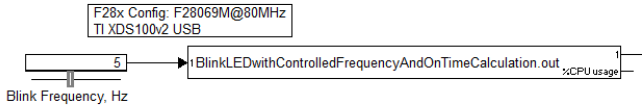
Edit the Debug model and delete or disconnect the "*Target Calculations"* compound block. In its place, add a "*Target Interface*" block from the ("*Embedded/Picollo/Target Interface")* menu. The "*Target Interface*" block will have the same input and output pins as specified in the "*Target Calculations*" compound block in the source model. Connect the "*slider*" to the input pin of the "*Target Interface*" block and the "*plot*" to the output pin.

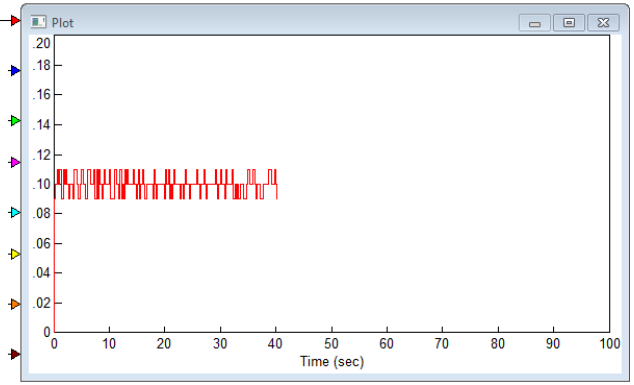
Configure the "*Target Interface*" block:
"*Target Execution file*" = ".out" file created in Step 2c.
"*Sample Rate (Hz)"* = desired value (it defaults to 1/"*Time Step*" value specified in the source model.
"*Keep Target Running*" = checked to for faster startup time



Host To/From Target Communication Example- Results

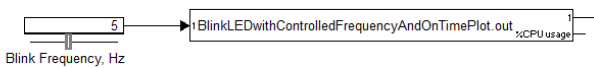
Click "*Go*", and, after a brief handshake, the Target will begin executing blinking the red LED at the frequency specified by the "*slider*".

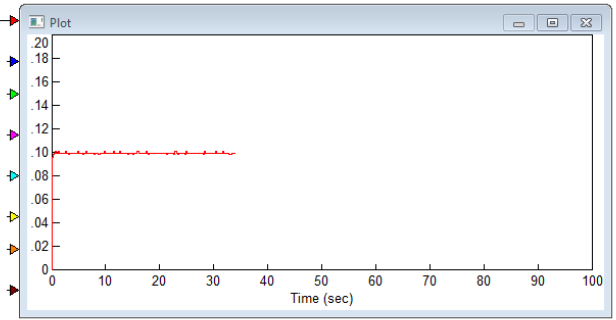




Setting the "*slider"* block at a 5 Hz frequency value, the *"plot"* block (right) displays the measured ":*OnTime*" of 0.1 seconds +/- .01 seconds. The variation is due to the "*Sample Rate (Hz*)" setting being used.

The variation can be reduced by increasing the "*Sample Rate (Hz)"* . Setting the "*Sample Rate (Hz)"* = 1000 Hz in the "*Target Interface*" block reduces the ":*OnTime*" error to +/- 0.01 seconds (below).



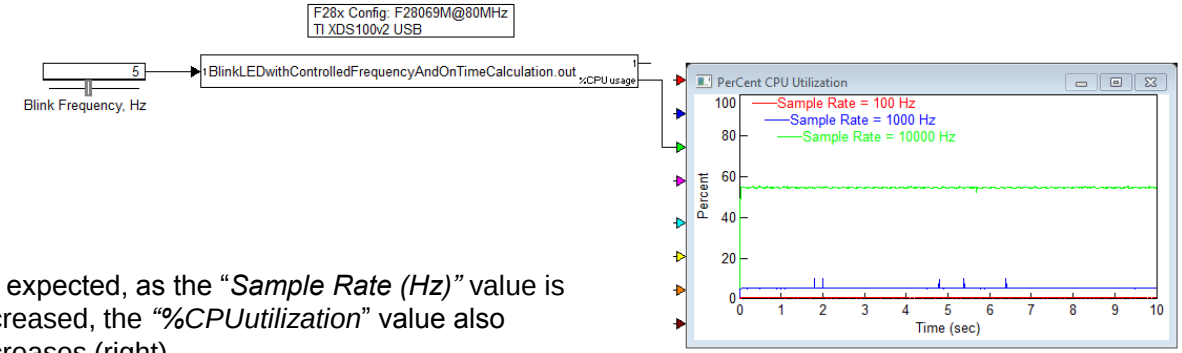


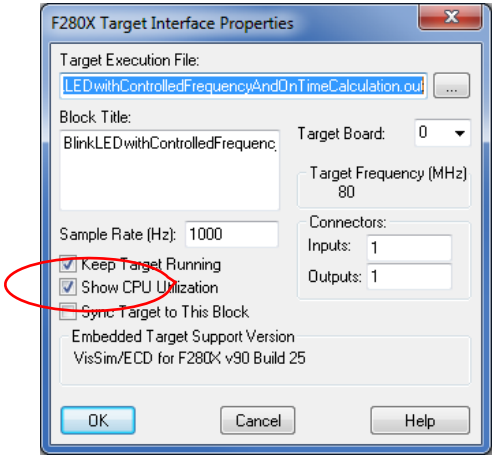
View debug model in VisSim

Displaying the CPU Usage of the Target Application

Checking the "*Show CPU Usage*" option in the "*Target Interface*" block adds an output pin to the "*Target Interface*" block entitled "%CPUusage. This output provides a dynamic value for the Target CPU utilization in percent .

The *BlinkLEDwithControlledFrequencyAndOnTimeCalculation-d.vsm* model (from the previous example) is modified to plot the "%CPU usage" at three "*Sample Rate (Hz)"* values; 100 Hz, 1000 Hz, and 10000 Hz.





As expected, as the "*Sample Rate (Hz)"* value is increased, the *"%CPUutilization*" value also increases (right).

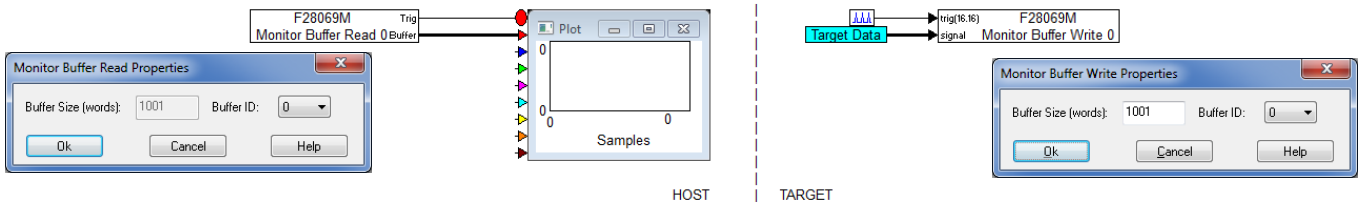
View debug model in VisSim

Target to Host High Speed Communication - Monitor Buffer

The JTAG interface between the Host PC and the Target communicates data at approximately 100 Hz. The JTAG communication rate is often very slow compared with the execution rate of the Target (often in the KHz range).

The "*Monitor Buffer Read*" and "*Monitor Buffer Write*" blocks provide a mechanism for a Debug model to buffer a large volume of data acquired on the Target at the Target "*Sample Rate (Hz)",* transmit the data periodically over the slower JTAG interface from the Target to the Host, and then make the buffer contents available as a vector of data at regular intervals on the Host application.

The following figure illustrates the buffer mechanism to capture, transmit, and display a buffer of 1001 elements using Buffer ID 0. The Target Update Rate = 10,000 Hz, and the Host "Time Step" = 0.01 seconds. The Target "Monitor Buffer Write" is triggered at 0.01 second intervals.



Sequence of Operation:

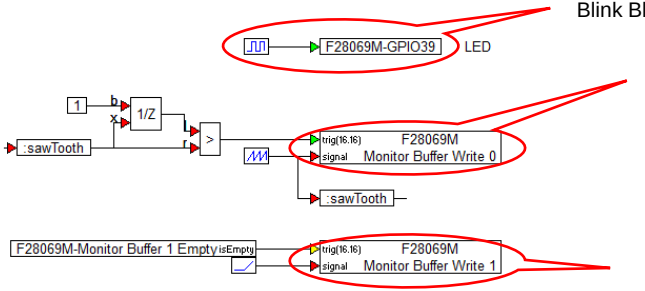
1. "*Monitor Buffer Write 0*" "*trig*" input outputs a pulse every 0.01 seconds
2. "*Monitor Buffer Write 0"* begins recording a new "*buffer*" of data when two conditions are met: (1) "*trig*" = 1 and (2) "*buffer*" is empty. NOTE: recording continues uninterrupted until the "*buffer*" is full.
3. When "*buffer"* is full; "*Monitor Buffer Read 0"* "*Trig*" output produces a "1" pulse and the "*buffer*" is emptied into the "plot" block and "*buffer"* is cleared and ready to accept new data.
4. Steps 2 and 3 are repeated.

Monitor Buffer – Waveform Capture & Real Time Check

This example illustrates the use of the "*Monitor Buffer*" to (1) record a Target waveform and (2) record elapsed time on the Target.

Two models are created in this example
Source Model: "*MonitorBufferTriggerAndTimeCheck.vsm"*
Debug Model: "*MonitorBufferTriggerAndTimeCheck-d.vsm"*

Step 1: Source Model **"***MonitorBufferTriggerAndTimeCheck.vsm"* - Add and configure the "*F28x Config …"* block. "*Time Step*" is set to 0.0001 seconds. A compound block named "*Target Calculations*" is created with the following contents;



Blink Blue LED at 2Hz

Monitor Buffer 0: (for waveform capture)
Buffer size = 100 elements
ID = 0
Input Signal = 200 Hz sawtooth
Trig Input: a 1-pulse is produced when sawtooth amplitude decreases (1x/cycle)

Monitor Buffer 1: (for Target time record)
Buffer size = 250 elements
ID = 1
Input Signal = unit ramp (amplitude = time)
Trig Input: "Monitor Buffer 1 Empty" which is set to 1-pulse when "Monitor Buffer 1" has been read and emptied by the Host

View source model in VisSim

Monitor Buffer – Waveform Capture & Real Time Check

Step 2. Code Generation - Lasso the "*Target Calculations*" compound block and then;

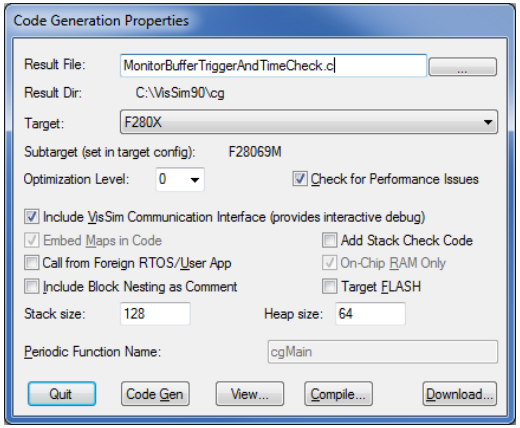
Step 2a. Click "*Tools/Code Gen"*

Step 2b. Configure the Code Generation Properties (right) as shown. Make sure the "*Include VisSim Communication.."* option is checked.

Step 2c. Click "*CodeGen" , "Compile…" , "Quit"*

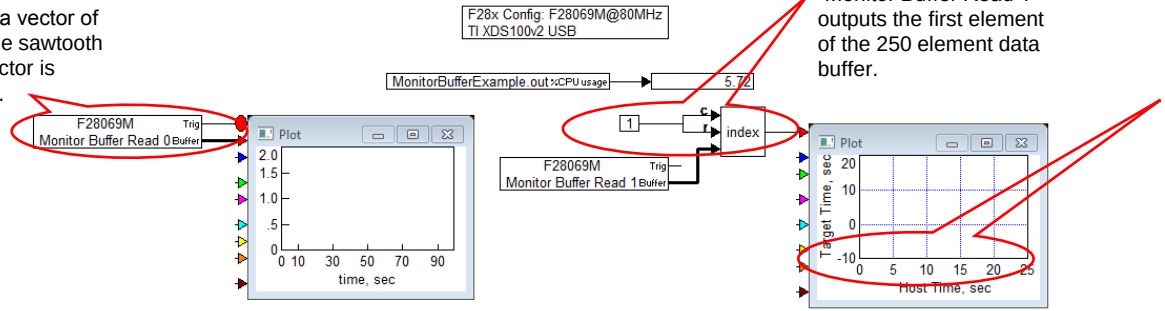
This step will create the ".*out*" file named as the "*Result File*" with the extension ".*out*" and place the file in the "*Result Dir*" which defaults to "C:\VisSim90\cg".

Step 2d. At this point you are finished with the "*BlinkLEDwithControlledFrequencyAndOnTimeCalculation.vsm"* source model, make sure it is saved.



Step 3. Debug Model - Create the debug model by renaming the source model to "*MonitorBufferTriggerAndTimeCheck-d.vsm"* and edit as shown below ("*Time Step*" = 0.01 sec)

"Monitor Buffer Read 0" outputs a vector of 100 elements beginning when the sawtooth first decreases in amplitude. Vector is recorded at 0.0001 sec intervals.

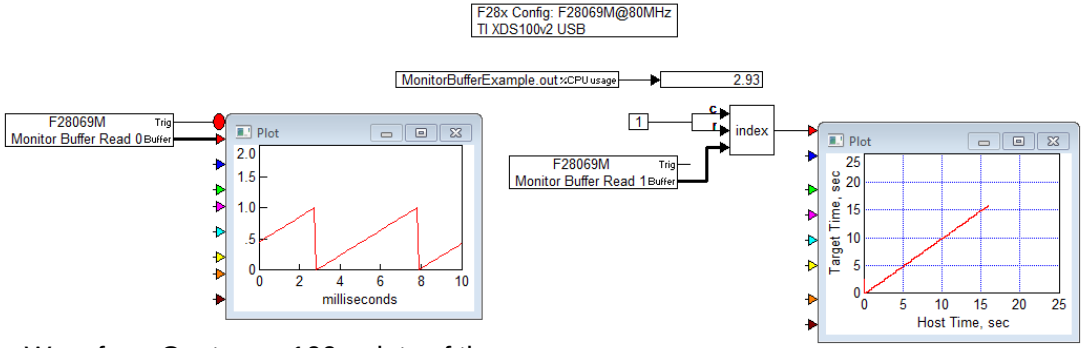


"Monitor Buffer Read 1" outputs the first element of the 250 element data buffer.

Target time (y axis) vs Host real time (x axis) to confirm real time operation.

Monitor Buffer – Waveform Capture & Real Time Results

Click "*Go*", and, after a brief handshake, the Target will begin executing blinking the blue LED at the 2Hz rate.



Waveform Capture: 100 points of the Sawtooth waveform are buffered at a10KHz rate on the Target. The buffer is displayed on the Host at a rate of 100Hz (1/.01 seconds). Each refresh of the buffer contains 10 milliseconds of Target data.

Real Time Results: The Target elapsed time is calculated at a10KHz rate, 250 values are buffered and transmitted to the Host. Even though the Host is executing at 100Hz, Target elapsed time buffer is updated every 25 msec (because the Buffer Size is set to 250 elements updated at 10KHz equivalent to a .1msec update time).

Since the slope of "Target Time, sec" vs "Host Time, sec" = 1 AND since the Host is being forced to run in real time, the Target is therefore executing at a true 10KHz rate.

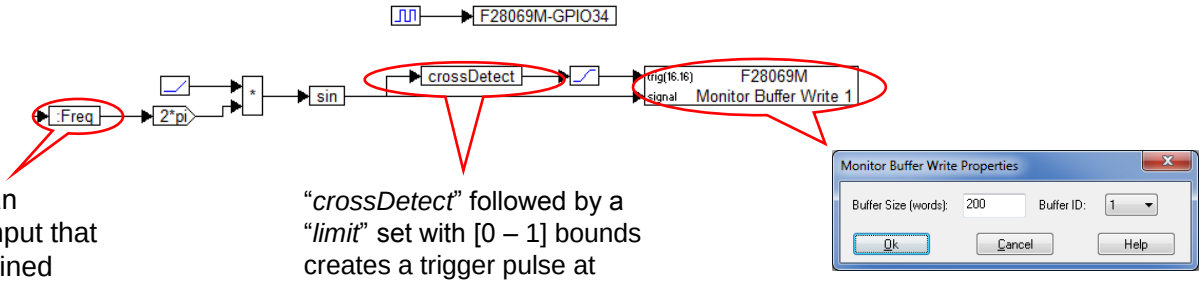
View debug model in VisSim

Monitor Buffer – Oscilloscope Display

This example illustrates the use of the "*Monitor Buffer*" to produce an Oscilloscope display of a sin wave signal generated on the Target. The Oscilloscope is triggered at a negative to positive zero crossing of the sin wave.

Two models are created in this example
Source Model: "*MonitorBufferControlledFrequencySinWave.vsm"*
Debug Model: "*MonitorBufferControlledFrequencySinWave-d.vsm"*

Step 1: Source Model **"***MonitorBufferControlledFrequencySinWavevsm"* - Add and configure the "*F28x Config …"* block. "*Time Step*" is set to 0.0001 seconds. A compound block named "*Target Calculations*" is created with the following contents;



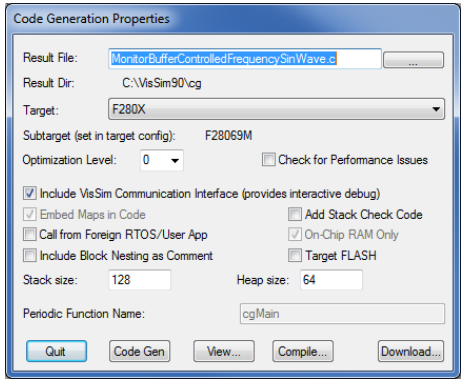
"Freq" is an external input that will be defined using a "*slider*" block on the Host.

"*crossDetect*" followed by a "*limit*" set with [0 – 1] bounds creates a trigger pulse at each negative to positive zero crossing of the sin wave.

View source model in VisSim

Monitor Buffer – Oscilloscope Results

Step 2. Code Generation - Lasso the "*Target Calculations*" compound block and then;



Step 2a. Click "*Tools/Code Gen"*

Step 2b. Configure the Code Generation Properties (right) as shown. Make sure the "*Include VisSim Communication.."* option is checked.

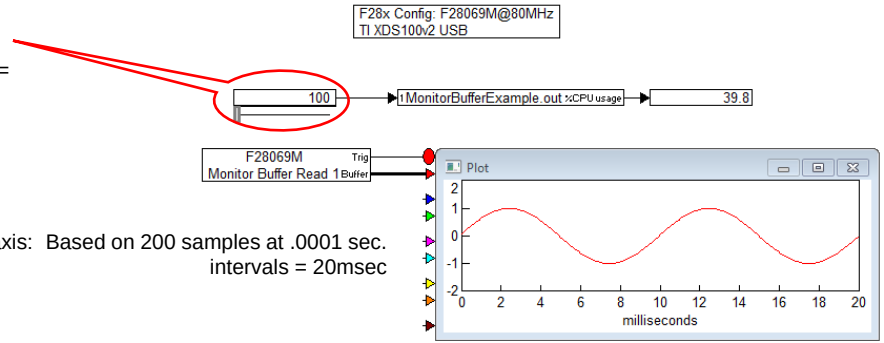
Step 2c. Click "*CodeGen" , "Compile…" , "Quit"*

This step will create the "*.out*" file named as the "*Result File*" with the extension "*.out*" and place the file in the "*Result Dir*" which defaults to "C:\VisSim90\cg".

Step 2d. At this point you are finished with the "*BlinkLEDwithControlledFrequencyAndOnTimeCalculation.vsm"* source model, make sure it is saved.

Step 3. Debug Model - Create the debug model by renaming the source model to "*MonitorBufferControlledFrequencySinWave-d.vsm"* and edit as shown below ("*Time Step*" = 0.001 sec)

"*slider*" defines sin wave frequency (Hz), bounds = [100 – 1000] Hz.



X-axis: Based on 200 samples at .0001 sec. intervals = 20msec

[View debug model in VisSim]

Fixed Point Arithmetic

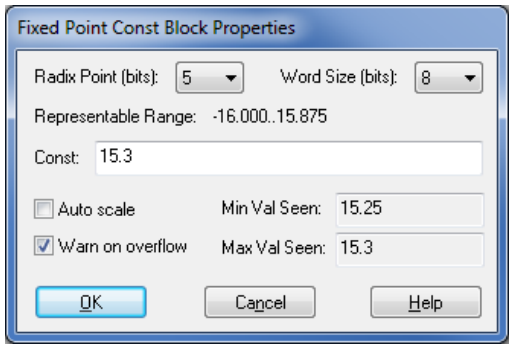
Fixed point arithmetic uses significantly less CPU time than floating point arithmetic on a CPU that does not have an Floating Point Unit (FPU), and for this reason it is widely used in embedded systems where performance is more important than precision.

Why you might not want an FPU:

- Adds more gates to the part = increased cost, increased physical size
- Increased energy consumption (bad for battery powered applications)
- Increases interrupt latency due to save/restore of FPU register set.

The VisSim fixed point block library (“Blocks/Fixed Point”), contains block functions for fixed point operations.

This library contains one “const” signal producer block. The “const” block properties are defined as;



Where:

Const: constant value in decimal form.
Radix Point (bits): Location of the radix point in the binary number (# of bits from the left, or the integer part)
Word Size (bits): Length of the binary number (bits)

Fixed Point Arithmetic - Terminology

Precision is the smallest difference between two consecutive binary values, is determined by the least significant (rightmost) bit.

For example, if a fixed point “const” block were configured as:

Radix Point = 2
WordSize = 16

Then, the number of bits to the right of the radix point = 14, the number of bits to the left = 2, and the precision = 2^-14

And the **notation** used is 2.16

When converting a decimal (floating point) value to a fixed point equivalent, precision determines how accurate the result is. sTE uses truncation if the magnitude of the binary equivalent is less than the original decimal value.

Many of the Texas Instruments blocks use 1.16 format which has a range between -1 to .99997 and the precision = 3.0518e-005

Clock, Clock Ticks, Timers, & Interrupts

In embedded electronics a **clock** is what controls how fast the CPU cycles. The CPU speed is measured in Hz.

The time required for a CPU cycle (or a clock cycle time) is 1/CPU speed, in units of seconds.

For example, an 80MHz CPU speed would have a clock cycle time = 1/80e6 = 1.25e-8 seconds = .0125 microseconds.

One complete clock is defined as a **clock tick.**

A **Timer** keeps track of how many clock ticks occur without having to write specific code to keep track of time.

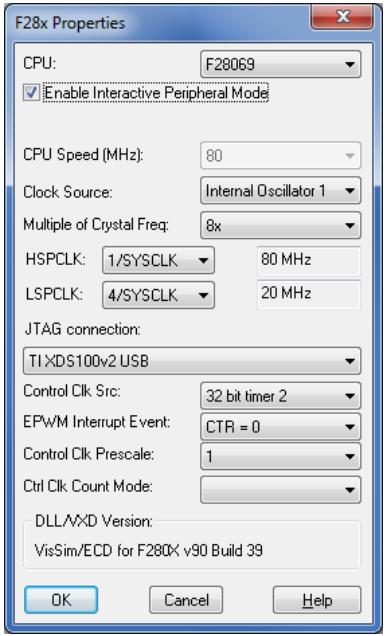
A Timer needs to be initialized and enabled. It will then proceed to count the clock ticks to a predefined value and then start over. You can set the Timer to generate events at multiple times along the way to its end value; these events could be an **interrupt** when it hits a certain number of clock ticks, or it can toggle, set, or clear a specialized pin.

By default, a CPU operates at the manufacturer's clock frequency, however, overclocking can be used to increase the CPU speed. Overclocking will use more power and generate more heat but will improve the speed performance of the CPU.

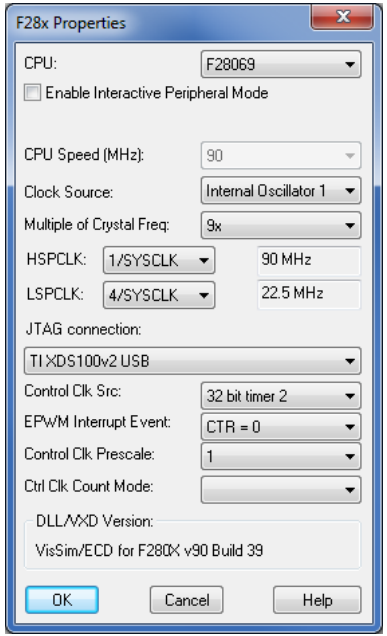
F28069 LaunchPad Clock Speed

The F28069 chip can be setup (using the Embedded/Piccolo/F28 Config… block) to run at its manufacturer's CPU Speed (80MHz) or it can be overclocked.

To set the CPU in overclock mode, the "Multiple of Crystal Freq" (below right) is selected to be 9x instead of the normal 8x value (to produce the 90MHz rate overclock CPU speed).



manufacturer's CPU Speed = 80MHz

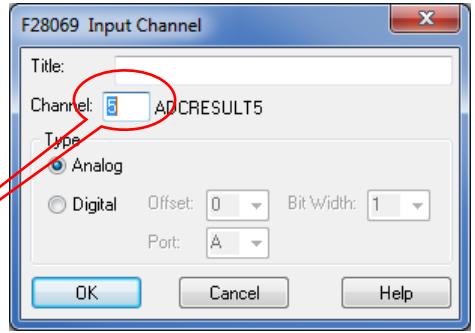


Overclocked CPU Speed = 90MHz

F28069 – Analog to Digital Conversion (ADC)

The Analog Digital Converter (ADC) block (“Embedded/Piccolo/Digital/Analog Input for F280x”) converts an analog (voltage) signal to a digital signal. A PWM signal is used to periodically trigger the ADC to begin its measurement and conversion.

An ADC accepts a voltage input signal and produces a digital output value:
Precision: The number of unique values, example, a 12 bit ADC has 4096 unique values
Range: The maximum and minimum input voltages, example 0 to 3.3volts
Resolution: The smallest detectable input signal change, example 3.3volts/4096 = .81millivolts

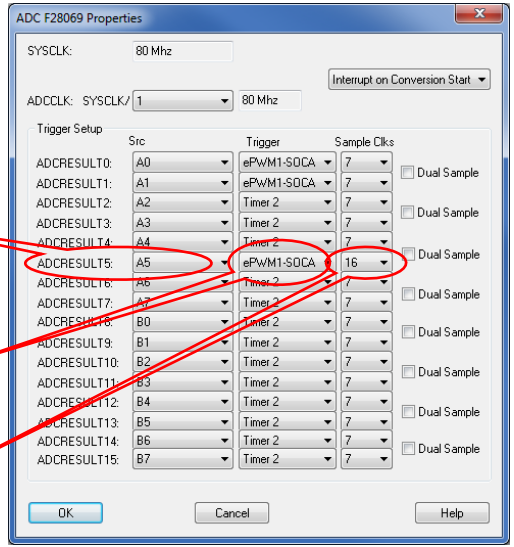


Channel 5 (ADCRESULT5) is selected for the ADC output

The ADC Configuration block is used to associate the ADC with a PWM generated Start Of Conversion (SOC) signal.

F28069 – ADC Configuration

The ADC unit must be configured to start conversions as SOCx pulses are received from the PWM unit. The “ADC F28069 Properties” (“Embedded/Piccolo/ADC Config…”) is shown to the right. The ADC block is using ch5 (ADCRESULT5)



There are 16 result registers; ADCRESULT0 through ADCRESULT15. Each result register is connected to a **“Src”** pin. At the right, ADCRESULT0 is shown connected to pin 0, ADCRESULT1 is connected to pin 1,

“Trigger” is the source of the trigger value for the ADC. For this entry we have set the trigger to “ePWM1-SOCA” (which will be explained next)

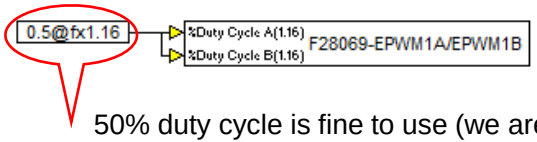
“Sample Clks” is the settling time required for the ADC to converge to a stable value in terms of “ticks”. This is normally set to a value between 7 and 11*. 16 was chosen here.

To sync ADC measurements to the ePWM unit, you must send a SOCx (start of conversion) pulse from the ePWM unit to the ADC and also configure the ADC unit to use the SOCx pulse as the ADC Sample Trigger. The Piccolo parts allow each PWM unit to send a SOC pulse to each ADC channel.

* TI does not allow “Sample Clks” values < 7.

F28069 – ADC SOCx Setup

Example of ePWM setup to produce SOCA



50% duty cycle is fine to use (we are only using 1 PWM, (EPWMA)

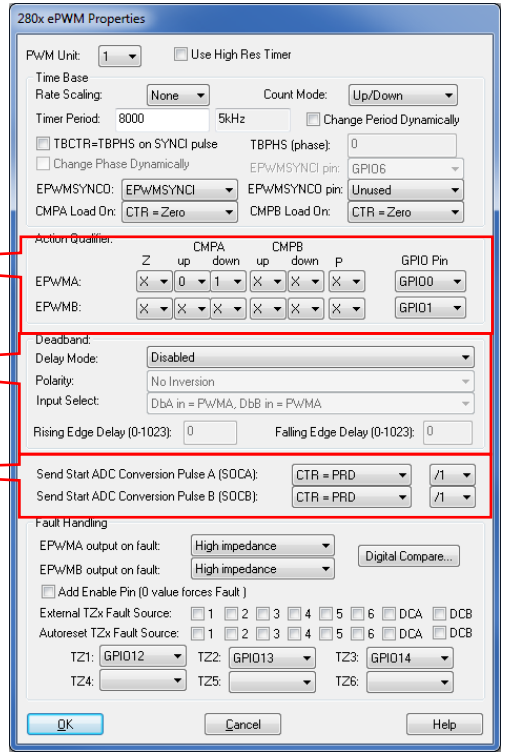
Only EPWMA is being used

No deadband is needed

SOCA is set to repeat each time the PWMA counter reaches a full period

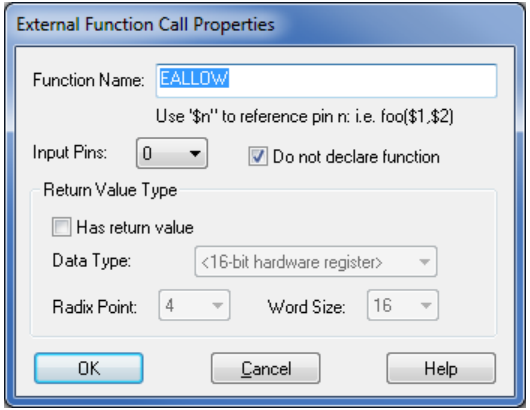
Before we apply the ADC, we need to understand how “functions” are called from VisSim

PWM Configuration:



Extern Function Block

The *Extern Function* block (*Embedded/Picollo/Extern Function*) lets you call an external function. For example, the *Extern Function* is configured to call the C function EALLOW (right).



The configuration parameters are described below:

Function name: Specifies the function call. You can specify arguments to the function that reference the input pins. The pins are referenced using $ notation. For example, Foo($1,$2).

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.

Return Value Type: Specifies the data type of the variable. If you choose hardware register, VisSim Embedded will only create a reference in the code and not an external declaration. The remaining data types are described in the block help file.

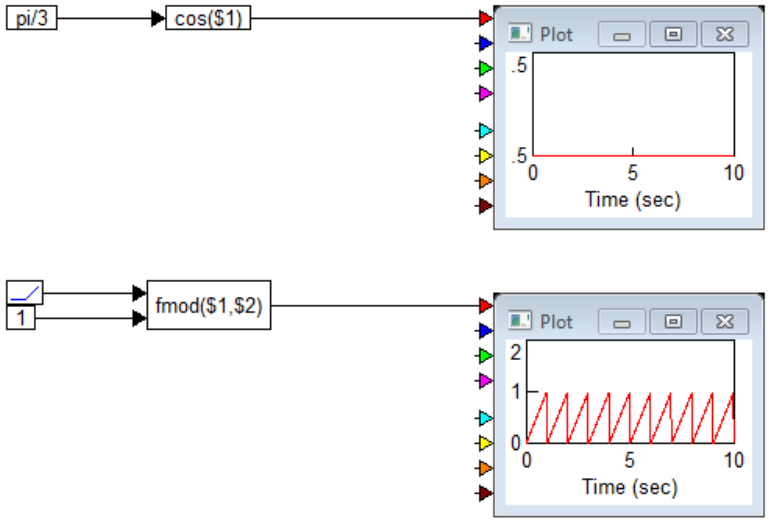
Expression Block

Solid Thinking EMBED recognizes C expressions for numeric data using the "*expression*" block located in the "*Blocks*" menu. Using this block, you are able to include common math and transformations functions from the "*C math.h numerics library*" like acos, asin, atan2, cos, cosh, exp, fabs, log, log10, pow, sin, sinh, sqrt, tan, tanh, … in your model.

The following link presents the functions in the common math and transformations in the "*C math.h numerics library*",

http://www.cplusplus.com/reference/cmath/

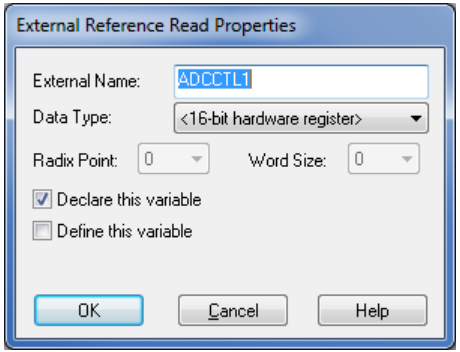
The following example demonstrates the use of the "*fmod*" and "*cos*" functions;



Extern Read & Extern Write Blocks

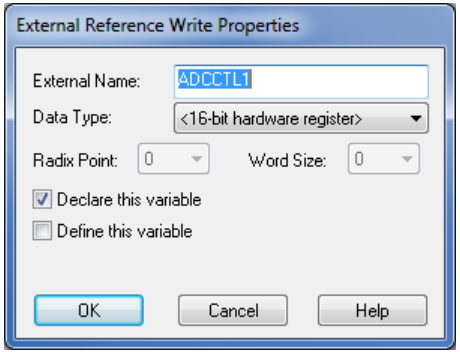
The *Extern Read* block (*Embedded/Picollo/Extern Read*) lets you read an external variable from another C code module into the diagram. If, for the Data Type, you choose hardware register, you can enter a hardware peripheral register name and the block output will produce the value of that register 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.



The *Extern Write* block (*Embedded/Picollo/Extern Write*) 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.



Configuring the ADC Control Register & Execution Order

ADCCTL1 is a control register that lets you configure the ADC unit, the bits are defined as follows:

```
// bit 15      0:    RESET, ADC software reset, 0=no effect, 1=resets the ADC
// bit 14      0:    ADCENABLE, ADC enable, 0=disabled, 1=enabled
// bit 13      0:    ADCBSY, ADC busy, read-only
// bit 12-8    0's:  ADCBSYCHN, ADC busy channel, read-only
// bit 7       1:    ADCPWDN, ADC power down, 0=powered down, 1=powered up
// bit 6       1:    ADCBGPWD, ADC bandgap power down, 0=powered down, 1=powered up
// bit 5       1:    ADCREFPWD, ADC reference power down, 0=powered down, 1=powered up
// bit 4       0:    reserved
// bit 3       0:    ADCREFSEL, ADC reference select, 0=internal, 1=external
// bit 2       1:    INTPULSEPOS, INT pulse generation, 0=start of conversion, 1=end of conversion
// bit 1       0:    VREFLOCONV, VREFLO convert, 0=VREFLO not connected, 1=VREFLO connected to B5
// bit 0       0:    TEMPCONV, 1=temp sensor connected to ADCINA5
```

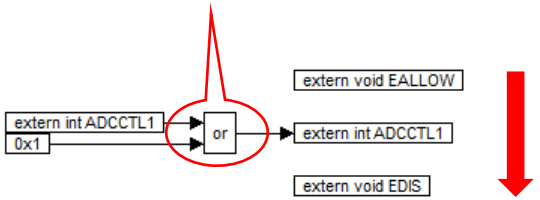
EALLOW is a function that allows writing to configuration registers.
EDIS is a function to disable writing to the configuration registers.

To configure the ADC to read the chip temperature on ADCINA, the following sequence of instructions is executed

EALLOW
ADCCTL1 bit 0 = 1 (bit 0 =TEMPCONV, setting bit 0 =1 causes temp sensor connected to ADCINA5)
EDIS

The equivalent VisSim block diagram is implemented using the “extern function”, “extern write”, and “extern read” block previously discussed

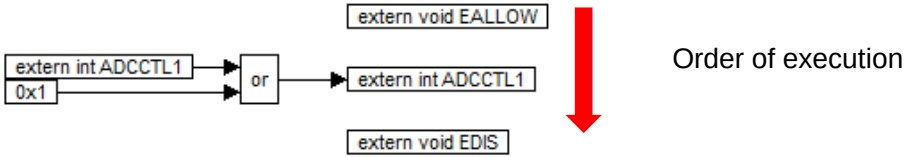
OR with 0x1 sets bit 0 to 1 and leaves the remaining bits unchanged



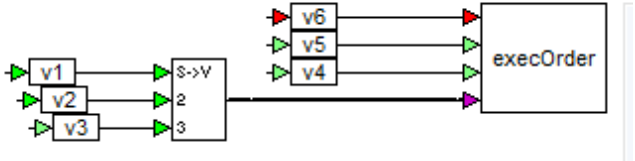
Execution Order: VisSim will execute in “top down” order so the vertical placement of the blocks is critical

Order of Execution

sTE will execute in “top down” order based on the vertical placement of blocks



In situations where the top down ordering is not adequate, sTE provides an “execOrder” block (“Blocks/Signal Consumers”)



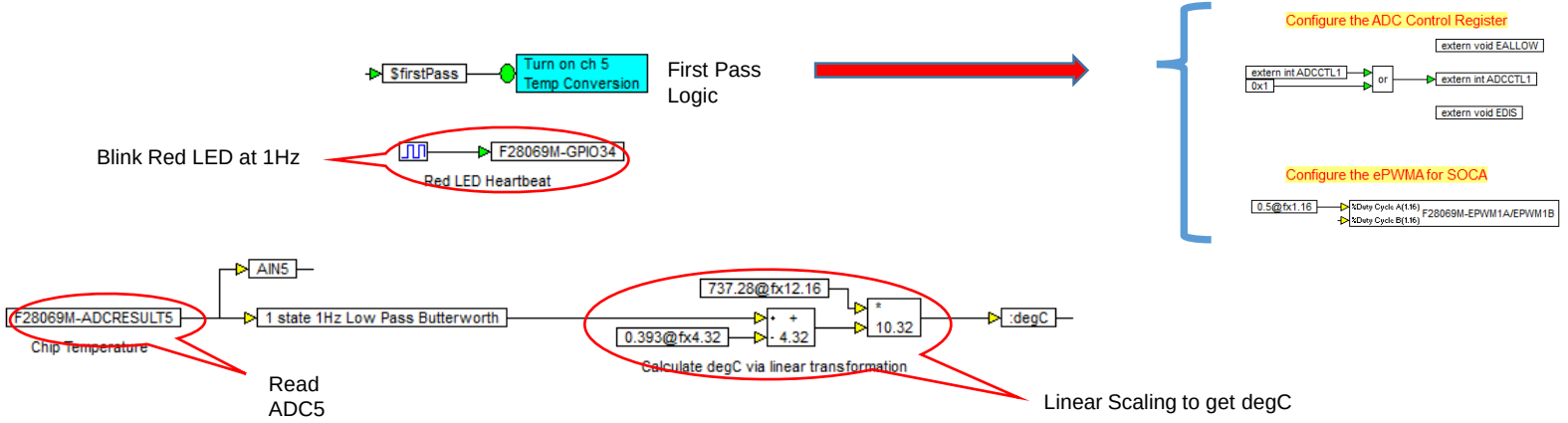
View this example in VisSim

Chip Temperature Example

This example illustrates the use of the “ADC” and “PWM” to measure the chip temperature of the microcontroller chip on the F28069M Launchpad board.

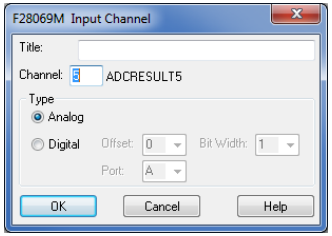
Two models are created in this example
Source Model: “*ChipTempOnF28069M.vsm*”
Debug Model: “*ChipTempOnF28069M-d.vsm*”

Step 1: Source Model **“***ChipTempOnF28069M.vsm”* - Add and configure the “*F28x Config …”* block. “*Time Step*” is set to 0.005 seconds. A compound block named “*Target Calculations*” is created with the following contents;

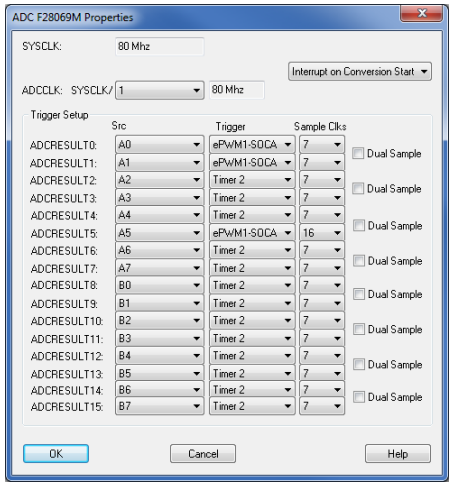


Chip Temperature Example – Setup's

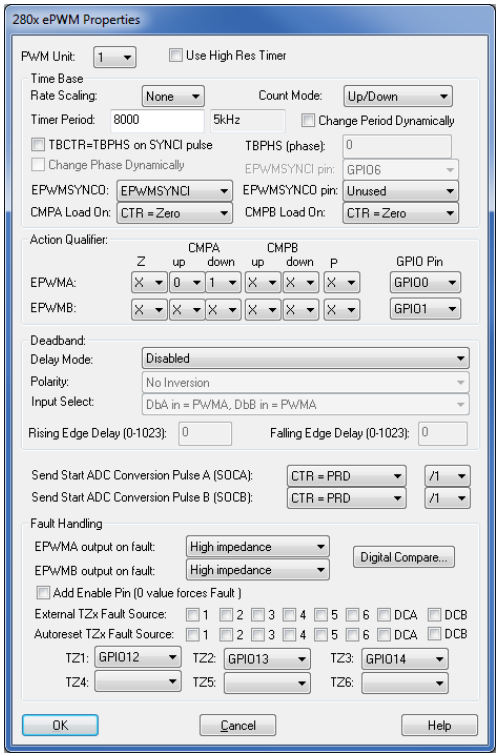
ADC Setup



ADC Configuration Setup



PWM SOCA Setup



[View Source Model in VisSim]

Chip Temperature – CodeGen & Execution

Step 2. Code Generation - Lasso the "*Target Calculations*" compound block and then;

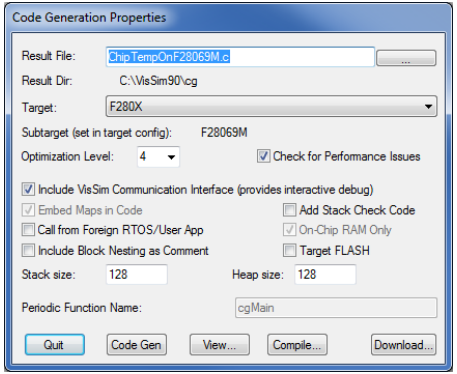
Step 2a. Click "*Tools/Code Gen"*

Step 2b. Configure the Code Generation Properties (right) as shown. Make sure the "*Include VisSim Communication.."* option is checked.

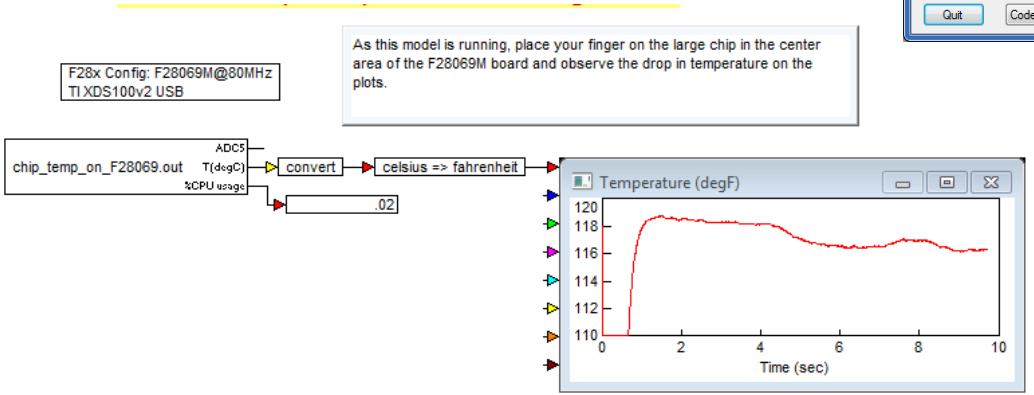
Step 2c. Click "*CodeGen" , "Compile…" , "Quit"*

This step will create the "*.out*" file named as the "*Result File*" with the extension "*.out*" and place the file in the "*Result Dir*" which defaults to "C:\VisSim90\cg".

Step 2d. At this point you are finished with the "*ChipTempOnF28069M.vsm"* source model, make sure it is saved.



Step 3. Debug Model - Create the debug model by renaming the source model to "*ChipTempOnF28069M-d.vsm"* and edit as shown below ("*Time Step*" = 0.005 sec)



View Debug Model in VisSim

Encoder

The F28069M LaunchPad board has inputs for two quadrature encoders. Quadrature encoders measure rotational angles by counting discrete “ticks”.

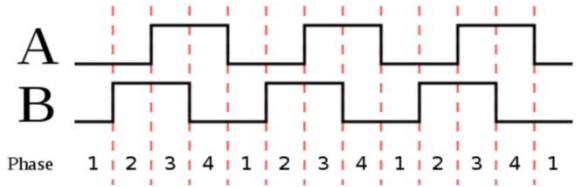
Typically, an encoder will have between 256 to 4000 ticks per revolution.

There are two types of encoders:

- Incremental: Although this type of encoder begins counting “ticks” at power up, it’s information is not accurate until an “index pulse” occurs. The “index pulse” occurs 1x/revolution. When used for motor control, incremental encoders must be rotated initially in “open loop” mode until the “index pulse” is sensed.
- Absolute: This type of encoder begins counting “ticks” at power up and provides accurate angle data immediately.

Encoders have 5 electrical connections: +5v, ground, A, B, index pulse

The A and B outputs consist of discrete values, 1 or 0, and are out of phase by 90 degrees, this allows the direction of rotation to be determined:



Encoder

The F28069M LaunchPad board has encoder peripherals that manage the encoder count value and reset the count value each time an “index pulse” occurs.

The F28069M LaunchPad encoder connections are shown below:

Note: the rectangular pin (viewed from bottom of LaunchPad board) is always pin 1

Encoder 2 Connection

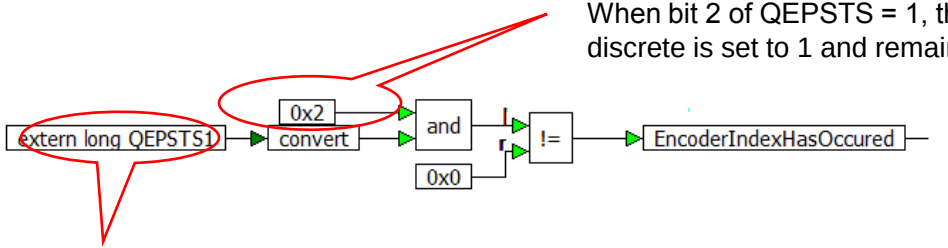
Encoder 1 Connection (pin 1 is leftmost pin)

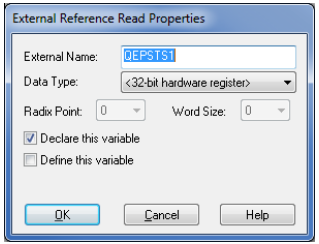
Encoder – Startup

An incremental encoder begins counting “ticks” at power up, it’s information is not accurate until an “index pulse” occurs. The “index pulse” occurs 1x/revolution. When used for motor control, incremental encoders must be rotated initially in “open loop” mode until the “index pulse” is sensed.

A VisSim model to detect the “index pulse” is presented below, In this model, the “index pulse” is named “EncoderIndexHasOccured”

When bit 2 of QEPSTS = 1, the “EncoderIndexHasOccured” discrete is set to 1 and remains there.

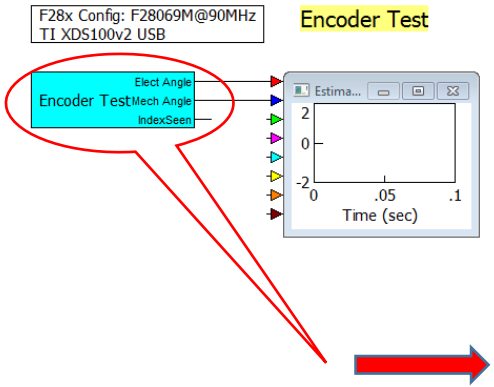


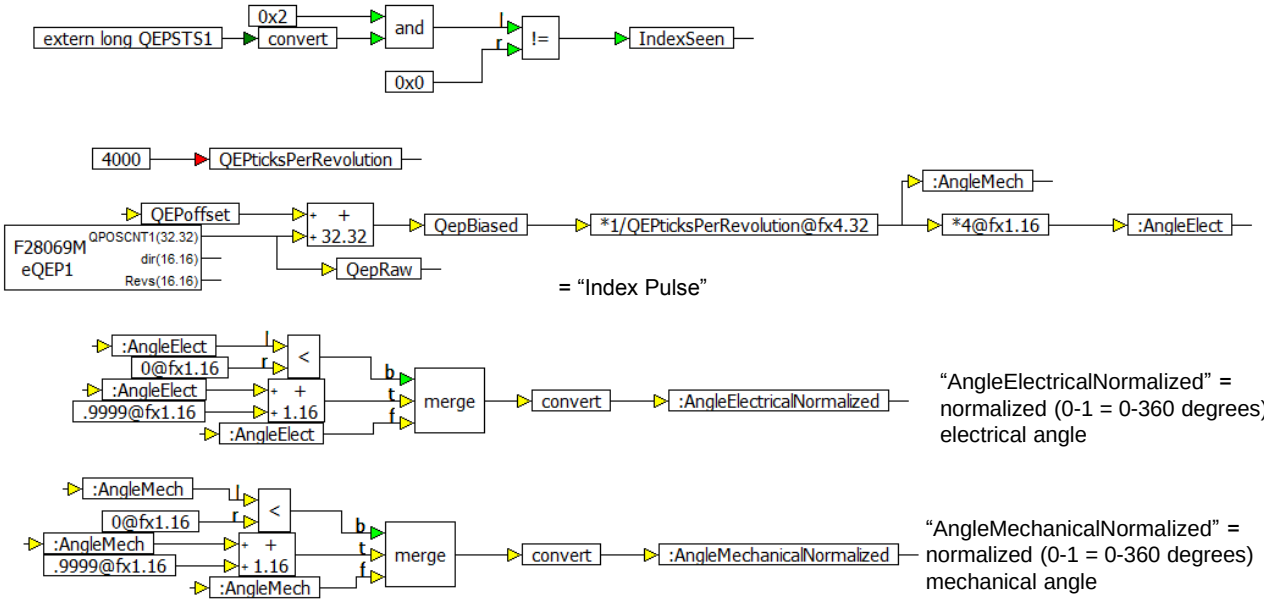


QEPSTS is the quadrature encoder status register

Encoder – Test Model

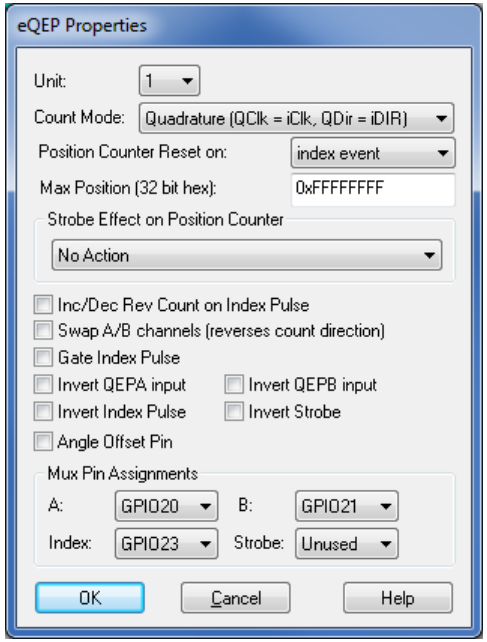
The following model is used on an 8 pole PMSM. It detects the “Index pulse” and measures the electrical and mechanical angles. “Time Step” = .0001 sec.



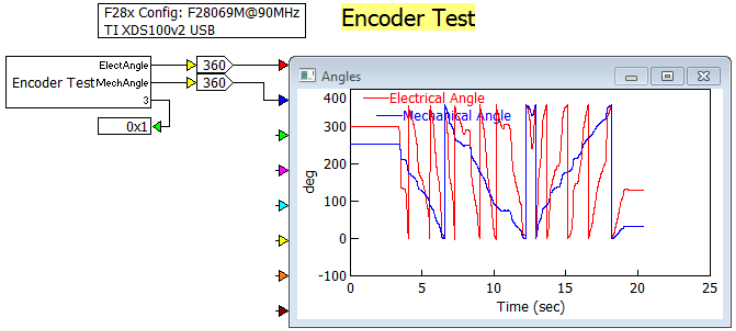


Encoder – TI Peripheral Block & Results

The eQEP Properties are presented below:



Manually turning the PMSM motor shaft produces the following time history results



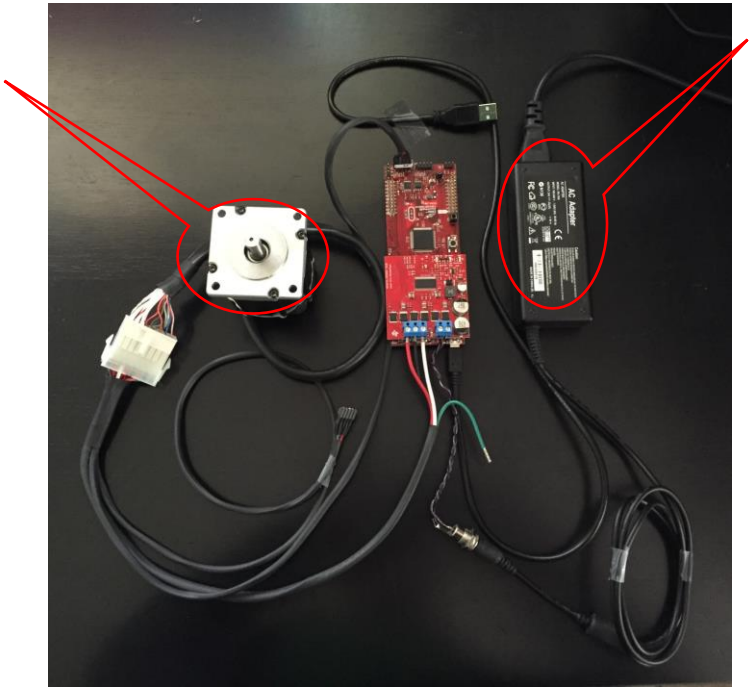
View Source Model in VisSim

View Debug Model in VisSim

Motor Position Control Example

This example illustrates the use of the “ADC”, “PWM”, and encoder to control the position (angle) of a PMSM motor.

Teknic M-2310P-LN-04K Low voltage servo motor with encoder http://www.ti.com/tool/lvservomtr

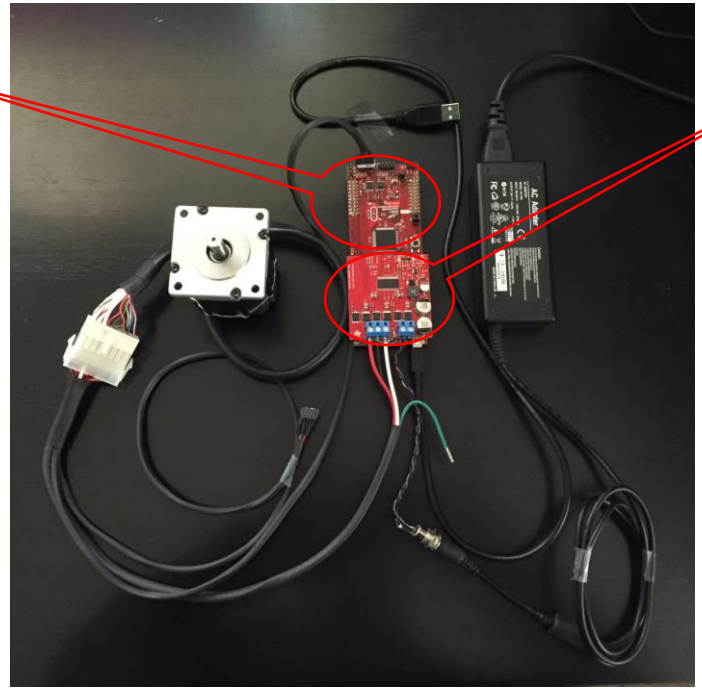


24V, 3A power supply

Motor Position Control Example

TI LAUNCHXL-F28069M

NOTE: J1 and J2 MUST be disconnected as the board is receiving power from the 24V power supply and not the USB



TI BOOSTXL-DRV8301
http://www.ti.com/tool/boostxl-drv8301

Motor Position Control Example

Two models are created in this example
Source Model: "*Motor Position Control - LaunchPadDRV8301-pmsm-28069M.vsm"*
Debug Model: "*Motor Position Control - LaunchPadDRV8301-pmsm-28069M-d.vsm"*

Step 1: Source Model **"***Motor Position Control - LaunchPadDRV8301-pmsm-28069M.vsm"* - Add and configure the "*F28x Config …"* block. "*Time Step*" is set to 0.0001 seconds. A compound block named "*PMSM Control*" is created with the following contents;

Step 2. Code Generation - Lasso the "*Target Calculations*" compound block and then;

Step 2a. Click "*Tools/Code Gen"*
Step 2b. Configure the Code Generation Properties (right) as shown. Make sure the "*Include VisSim Communication.."* option is checked.
Step 2c. Click "*CodeGen" , "Compile…" , "Quit"*
This step will create the ".*out*" file named as the "*Result File*" with the extension ".*out*" and place the file in the "*Result Dir*" which defaults to "C:\VisSim90\cg".
Step 2d. At this point you are finished with the "*Motor Position Control - LaunchPadDRV8301-pmsm-28069M.vsm.vsm"* source model, make sure it is saved.

View Source Model in VisSim

Motor Position Control Example

Step 3. Debug Model - Create the debug model by renaming the source model to "*MonitorBufferTriggerAndTimeCheck-d.vsm"* and edit as shown below ("*Time Step*" = 0.005 sec)
Set the "Sample Rate (Hz)" in the F280x Target Interface block to 20KHz.

Click "*Go*", and, after a brief handshake, the Target will begin executing.

Setting the "***slider"*** block to different position setpoint values will cause the motor to rotate until the setpoint value is achieved.

View Debug Model in VisSim

End of Section

EMBED™
solidThinking