CLB Tool

This user's guide describes the structure and use of the Configurable Logic Block (CLB) tool. Information on the architecture of the CLB may be found in the device-specific technical reference manual (TRM).

It is assumed that the reader is already familiar with the architecture of the CLB and with the CCS IDE. For detailed information on the CLB, see the device-specific TRM.

Contents

List of Figures

List of Tables

Trademarks

1 Introduction

1.1 CLB Tool Outline

The CLB is a hardware module integrated into certain C2000™ devices. The CLB contains a set of configurable blocks and inter-connections which allows users to create their own custom digital logic along the lines of what could be done with a FPGA. For example, the CLB might be configured to enhance the functionality of existing device peripherals, or to create new peripheral functions. The CLB is configured using a software utility, referred to here as the “CLB tool”.

The CLB tool allows the user to configure and connect sub-modules in each CLB tile.

The tool makes use of the “SysConfig” graphical user interface (GUI) which is part of Code Composer Studio™ (CCS). The tool includes a small number of examples intended to help the user explore the features of the tool and to create their own projects.

The tool generates a C header file containing a set of constants corresponding to the configuration settings defined by the user in the GUI. The tool also generates a C source file which uses the constants in the C header file to initialize the CLB modules by loading the constants into the CLB registers by a sequence of register load operations. The functions in the C source file must be called during the device initialization. The tool does not configure the input and output connections between the CLB tile and other device peripherals, including the cross-bars and other CLB tiles. The configuration of these registers must be done separately and is the responsibility of the user.

1.2 Overview of the CLB Configuration Process

The CLB tool is based on the “SysConfig” tool in CCS. This, together with files supplied as part of the C2000Ware download, is sufficient to configure the CLB. In order to conduct a simulation of the design it is necessary to install a number of third party tools, including a compiler and a wave viewer. For more information on the CLB simulator, see Section 4.

The CLB tool generates a “.dot” file which shows sub-module inter-connections in diagram form and can be used to verify the design. This file is converted to HTML format in CCS post-build steps using node.js and JavaScript libraries in the provided examples. The tool also generates a “clb_sim.cpp” file. The CPP file, along with other CLB simulation models, is compiled using a GCC compiler. The output of the compilation is an “.exe” file, which must be executed on the local machine in order to generate a “.vcd” file. This “.vcd” file can be used to conduct a timing analysis using an external graph viewer. All these steps are automatically done using post-build steps of CCS.

The CLB configuration is encoded in the generated C header file “clb_config.h”. The “clb_config.c” file generated by the CLB tool uses the generated header file to load the configuration in to the CLB module’s registers. It is important to note that in the application code for the C28x device, the functions in the “clb_config.c” file must be called during the device initialization steps. Figure 1 shows the output of the CLB tool and the post-build steps.

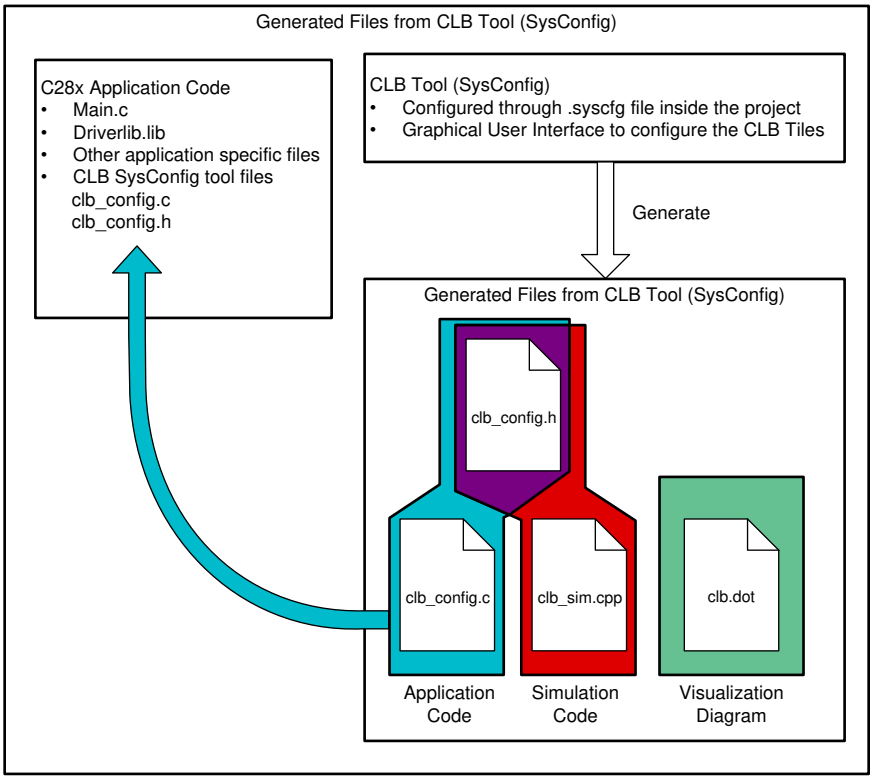

Figure 1. CLB Tool Project Structure

In a typical scenario, the user begins with a specification of the desired CLB logical functionality. This may be in the form of a logic circuit diagram, timing information, written description, VHDL code, or some other form. Having installed the requisite tools, the first step will be to connect the tile sub-modules to implement the desired logic.

The specification may include a set of timing diagrams in which case the user may (optionally) decide to conduct a simulation of the CLB configuration to ensure behavior is as expected. This step includes defining a set of input test stimuli, and building a simulation project to generate simulation waveforms which can be opened in a graph viewer. If the results are not as expected, the user will modify the SysConfig settings and repeat the simulation.

Once correct waveforms are obtained from the simulation, the user can then download the design into the device in the normal way.

In a CLB SysConfig enabled CCS project for a C28x device, the steps to create the HTML block diagram of the CLB Tile configuration and the generation of the “.vcd” simulation waves are automated. When the user builds the CCS project, the user application code, along with the generated “clb_config.h” and “clb_config.c”, are compiler using the C28x compilers and a “.out” file is generated. Post-build steps compile the generated simulation files, “clb_sim.cpp” and “clb_config.h”, along with the CLB simulation modes, using a GCC compiler. The output of this step is a “.exe” file (“simulation_output.exe”). Next in the post-build steps, the “.exe” file is executed and “CLB.vcd” is generated. This file can be viewed using an external graph viewer.

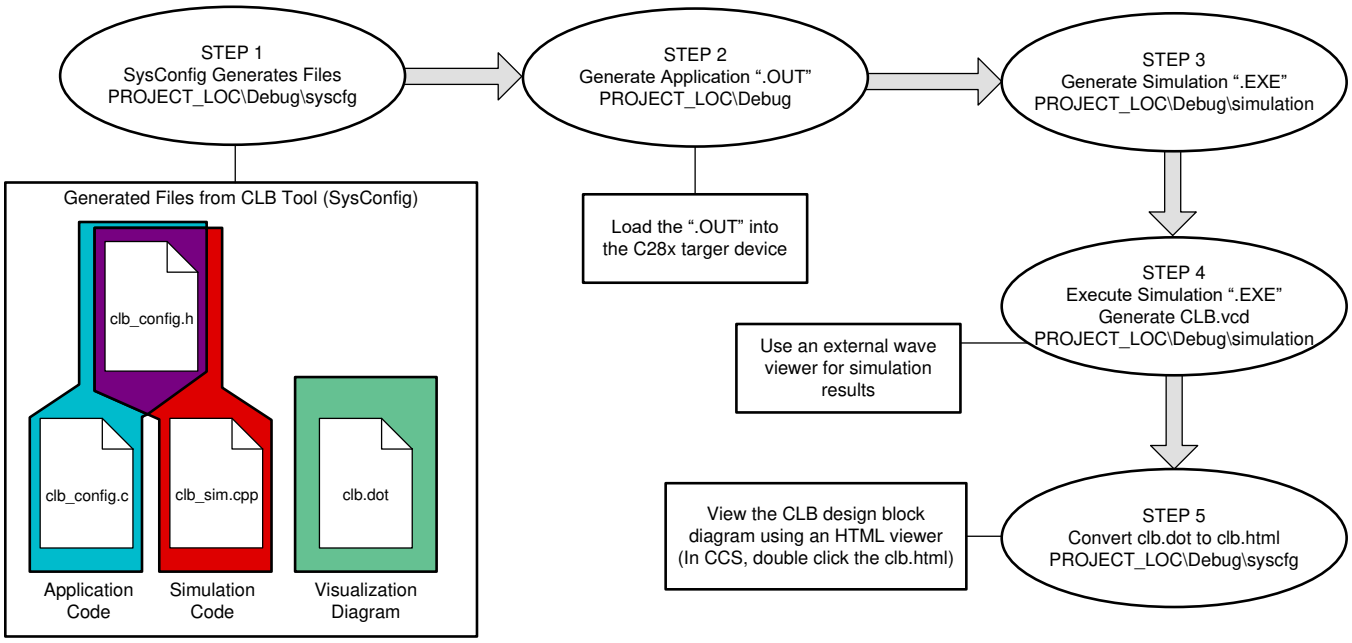

Figure 2. CLB Tool Build Process

2 Getting Started

This section is intended to help new users to get started quickly with the CLB tool.

2.1 Introduction

In order to use the tool, Code Composer Studio (CCS) version 9.0 or later must be installed. Earlier versions of CCS do not contain the SysConfig utility, which is required for CLB configuration. For further information and to download "Code Composer Studio", visit: http://www.ti.com/tool/ccstudio-c2000.

The above tools allow the user to configure the CLB. However, in order to simulate the design the following additional external (non-TI) tools must be installed:

- A GNU compiler (TDM-GCC)
- A simulation viewer (GTK Wave)

2.2 Installation

2.2.1 GNU Compiler

1. Download “tdm-gcc” from the following link: https://sourceforge.net/projects/tdm-gcc/files/TDM-GCC%20Installer/tdm64-gcc-5.1.0-2.exe/download.
2. Open the downloaded file to install the compiler.
3. Uncheck the "Check for updated files on the TDM-GCC server" checkbox.
4. Select “Create” from the setup wizard.

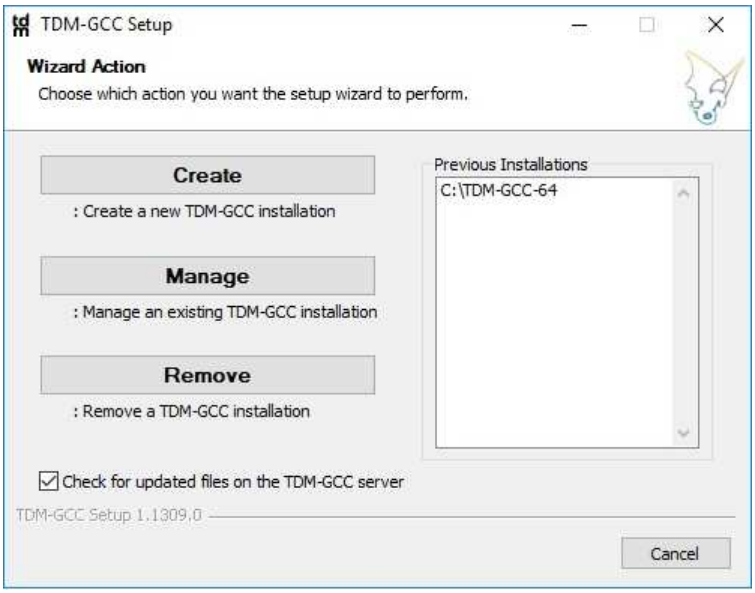

Figure 3. TDM Compiler Installation Wizard

5. Select the 64-bit installation and click “Next”.

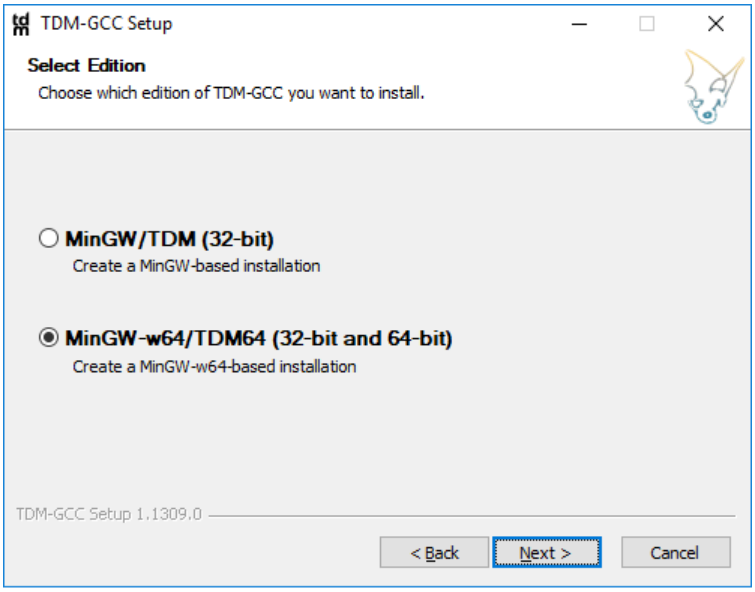

Figure 4. TDM Compiler Installation 64-bit

6. Select the installation directory as C:\TDM-GCC-64 and click “Next”.

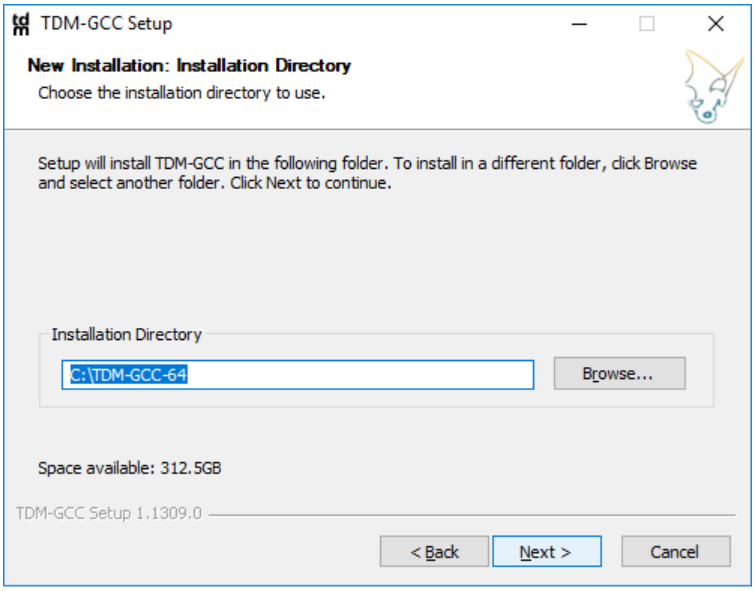

Figure 5. TDM Compiler Path

7. Complete the remaining steps in the installation procedure according to your geographical location.

2.2.2 Install the Simulation Viewer

1. Download the waveform viewer GTKwave from this link: https://sourceforge.net/projects/gtkwave/files/.
2. Download the native binaries for the correct Windows installation, (for example, for 64-bit Windows, select “gtkwave-3.3.100-bin-win64”), and extract the downloaded zip file into the directory c:\gtkwave.

3 Using the CLB Tool

This section describes how to use the CLB tool to configure a CLB tile. The CLB tool requires CCS version 9.0 or newer.

3.1 Import The Empty CLB Project

Driverlib based, CLB enabled example projects are available in <C2000WARE_INSTALL>/driverlib/<device>/examples.

For example the F2837xD devices, the path to the empty CLB project (along with other CLB example projects) is <C2000WARE_INSTALL>/driverlib/f2837xd/examples/cpu1/clb.

1. In the CCS menu, click 'Project -> Import CCS Projects…'
2. Enter the path to the CLB example projects in the 'Select search-directory'.
3. Click 'Refresh'.
4. Select the 'clb_empty' project.
5. Check 'Copy projects into workspace'.
6. Click Finish.

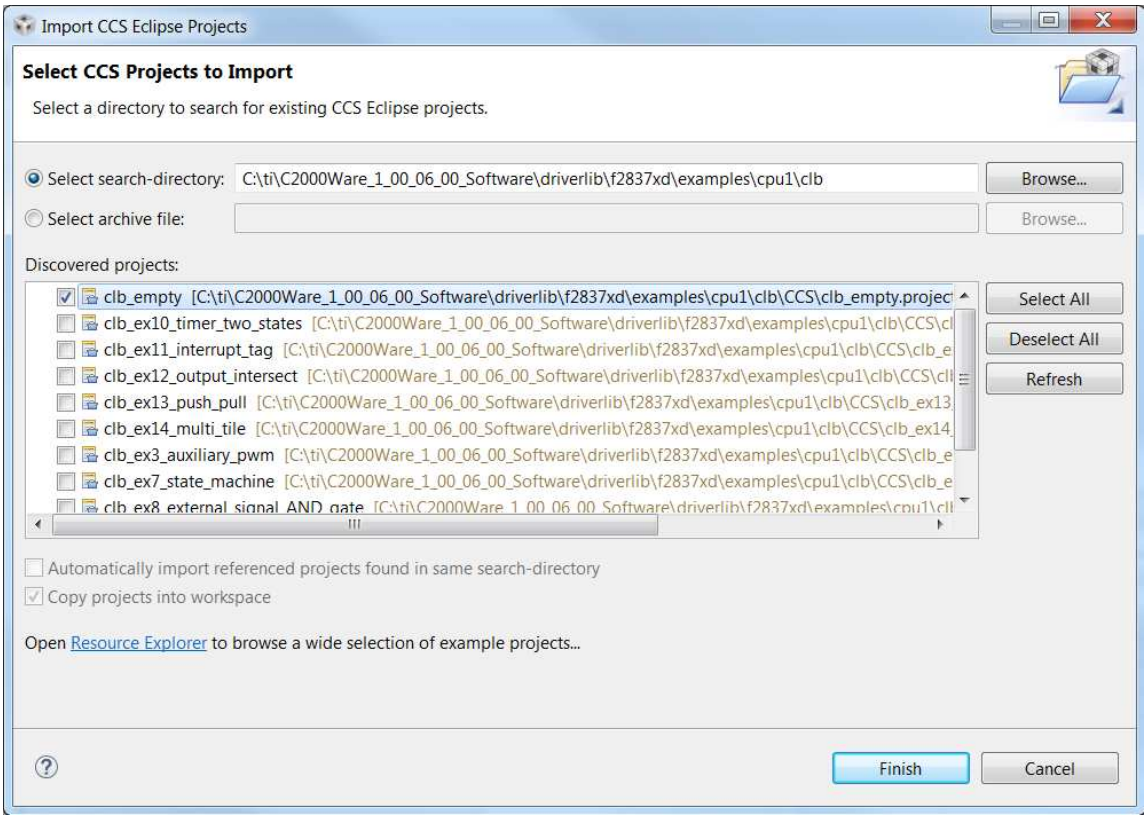

Figure 6. Import CCS Eclipse Projects

3.2 Updating Variable Paths

The empty CLB project imported above has the capability to not only generate the ".OUT" file for the C28x target, but also has the capability to generate the simulation files and the HTML block diagram of the design.

The path to the GCC compiler downloaded above may be different than the one specified in the project. To double check this:

1. Right click on the project and select 'Project Properties'.
2. Under 'Resources', select 'Linked Resources'.
3. Check to make sure all the paths below are correct:
   a. CLB_SYSCFG_ROOT (All CLB components are relative to this path)
   b. CLB_SIM_COMPILER (Important for simulation)
   c. SYSTEMC_INSTALL (Important for simulation)

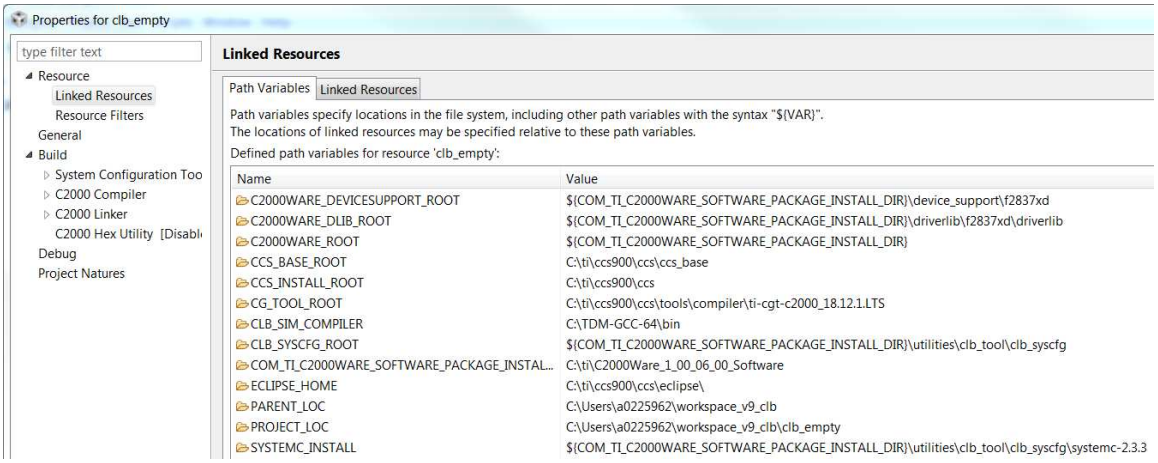

Figure 7. Linked Resources

4. If the icon to the left of the name is not a folder, and is instead an exclamation point, the path does not exist on your system and you must manually select the correct one.

3.3 Configuring a CLB Tile

To open the configuration tool, double-click on the ".syscfg" file you want to edit in the CCS Project Explorer window. A screen like that is shown in Figure 8.

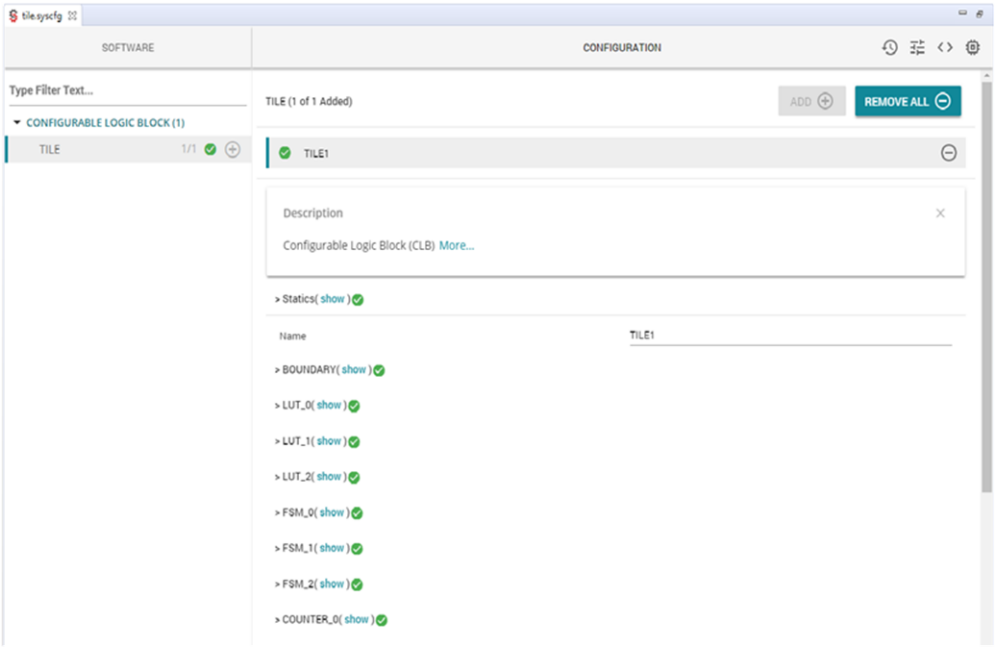

Figure 8. CLB Tool SysConfig Screen

If this screen does not open, be sure you have correctly completed the steps before this.

The configuration of CLB tiles are contained in each .syscfg file. You can change the name of the tile if desired. Multiple .syscfg files can be added to the same project.

For the highlighted tile a list of sub-modules is shown in the pane to the right. The parameters of each sub-system can be inspected and edited by clicking on the word "Show" to the right of its name.

The “BOUNDARY” item is a special case. This group allows the user to select the tile inputs for simulation only. When the tool configuration is generated the CLB inputs always come from global and local mux modules as described in the TRM, however for the purposes of simulation the user can specify a square wave signal source, together with a period and duty cycle (both in clock cycles), and sync conditions as shown in Figure 9. Custom waveform generation for simulation purposes is also supported. For more information on the simulator, see Section 4. These options are only for simulation and do not affect the actual CLB configuration.

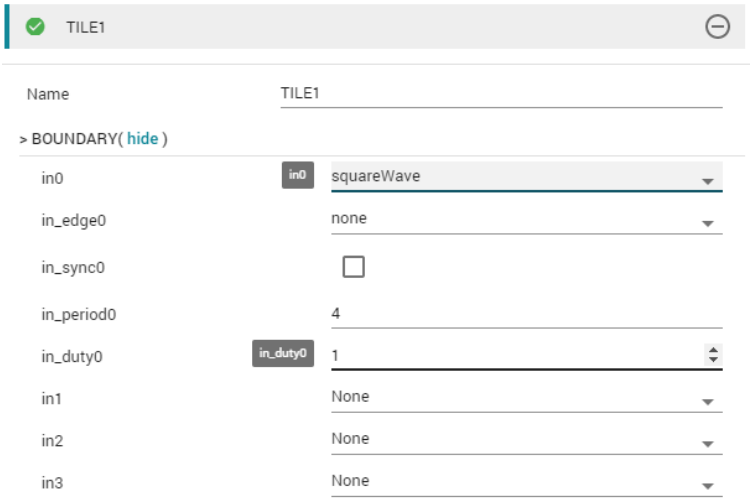

Figure 9. Boundary Input Options

The user configures and connects sub-modules in each tile using the check-boxes and drop-down options in the tool. Context sensitive help appears when the mouse cursor is hovered over each item in the configuration tool. Figure 10 shows an example for the match1_val field in the COUNTER_0 sub-module.

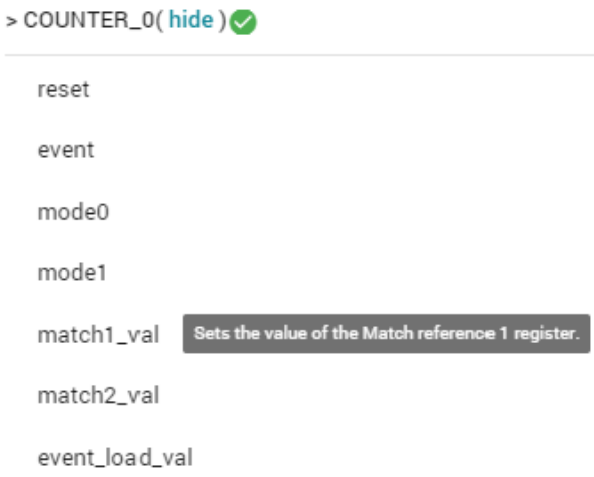

Figure 10. Counter Options

Logical equations for the LUTs and FSMs are configured by text entry using C format. Table 1 shows the symbols that are allowed in a Boolean equation.

Table 1. Supported Logical Operations

Logical Operation	Symbol
AND	&
OR	|
XOR	^^
NOT	!

The use of parentheses is supported: for example, one could write: i1 | !(i2 & i3). The tool performs syntax checking on the equations as they are entered. Invalid equations are indicated as they appear by an error message below the entry line.

Some unlikely logical combinations generate a warning to the user. Figure 11 shows an example in which the user has attempted to use the i2 input in LUT_0 in a Boolean equation. However, i2 is configured to be a constant, which is unlikely to be what the user intended. The warning appears both below the equation and below the input selection.

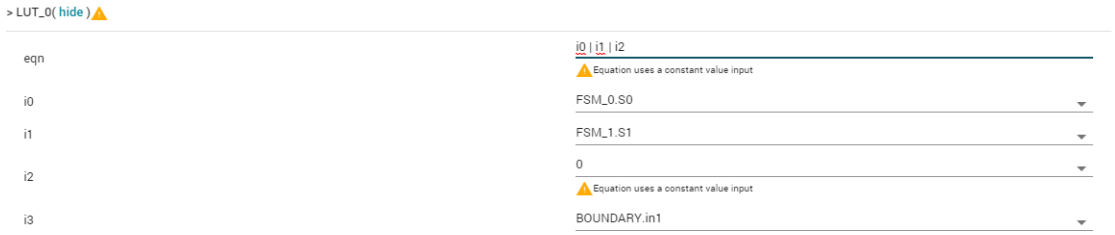

Figure 11. Equation Warning

For some fields the tool performs range checking on numerical entries to ensure they lie within the allowable range. For example, an attempt to load a counter sub-module with a value greater than 2^{32} will produce a warning.

The tool automatically generates number of files as the user enters configuration data. To view the generated files, click on the "<>" symbol in the upper right corner of the tool. Double-click on the filename to open it.

Figure 12. CLB Tool Generated Files

CLB configuration register settings are contained in the header file “clb_config.h”, which can be opened and inspected by the user by clicking the filename. An example is shown in Figure 13. It is important to understand that this file is updated by the tool each time the user changes any CLB settings. Therefore, users should not make any manual changes to the contents of file as these will be over-written by the tool. If the file is kept open while changing CLB settings, the user may observe the affected register data changing in the file.

clb_config.h

```
/*
 *  ======== clb.h ========
 *  DO NOT EDIT - This file is generated by the SysConfig tool.
 */
#ifndef ti_clb_h
#define ti_clb_h

#include <stdint.h>

#ifdef __cplusplus
extern "C" {                // support C++ sources
#endif

// HLC Instruction Register Field definitions
#define HLC_OPCODE_R0 0x0
#define HLC_OPCODE_R1 0x1
#define HLC_OPCODE_R2 0x2
#define HLC_OPCODE_R3 0x3
#define HLC_OPCODE_C0 0x4
#define HLC_OPCODE_C1 0x5
#define HLC_OPCODE_C2 0x6

#define HLC_OPCODE_MOV    0x00
#define HLC_OPCODE_MOV_T1 0x01
#define HLC_OPCODE_MOV_T2 0x02
#define HLC_OPCODE_PUSH   0x03
#define HLC_OPCODE_PULL   0x04
#define HLC_OPCODE_ADD    0x05
#define HLC_OPCODE_SUB    0x06
#define HLC_OPCODE_INTR   0x07

//-----------------------------------------------------------------------------
// TILE1
//-----------------------------------------------------------------------------
#define TILE1_CFG_OUTLUT_0 0x7702c7
#define TILE1_CFG_OUTLUT_1 0x0
#define TILE1_CFG_OUTLUT_2 0x7702cf
#define TILE1_CFG_OUTLUT_3 0x0
#define TILE1_CFG_OUTLUT_4 0x770257
#define TILE1_CFG_OUTLUT_5 0x0
#define TILE1_CFG_OUTLUT_6 0x0
#define TILE1_CFG_OUTLUT_7 0x0

#define TILE1_CFG_LUT4_IN0   0x60a4
```

Figure 13. “clb.h” Header File Example

Fields for the HLC sub-module include those for configuring the events and initial values. Each of the four events can trigger execution of a short program consisting of up to eight instructions. For more information on HLC, see the device-specific TRM.

When a valid event trigger is selected, the tool displays lines where the user can type HLC instructions. One blank line is always shown until all eight instructions have been used. In Figure 14, the user has selected one HLC trigger events and typed in a short program consisting of three instructions.

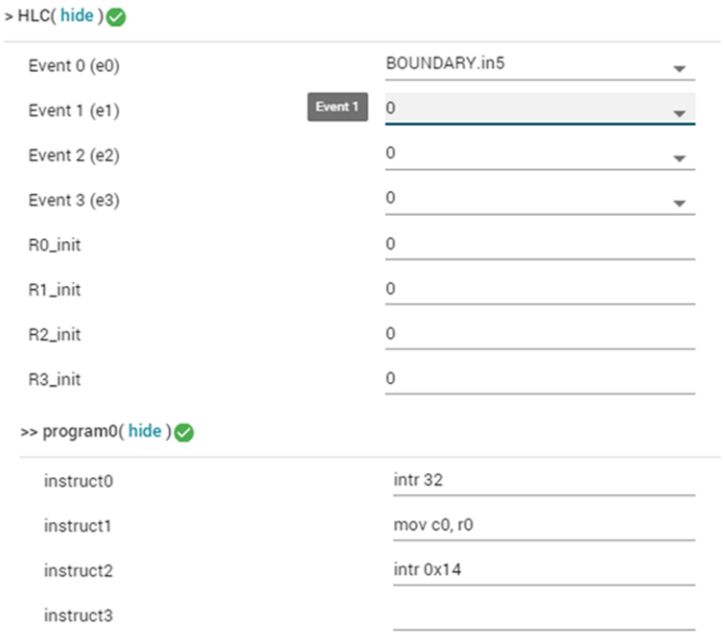

Figure 14. HLC Configuration Example

The file “clb.dot” allows the user to inspect a visual representation of the inter-connection of sub-modules. An HTML version of this block diagram is generated in the post-build steps which can be opened and viewed inside CCS.

4 The CLB Simulator

4.1 Using the Simulator

4.1.1 The Statics Panel

The top panel in the configuration tool contains three “static” settings used in simulation. Hold the mouse over each field to see a short description.

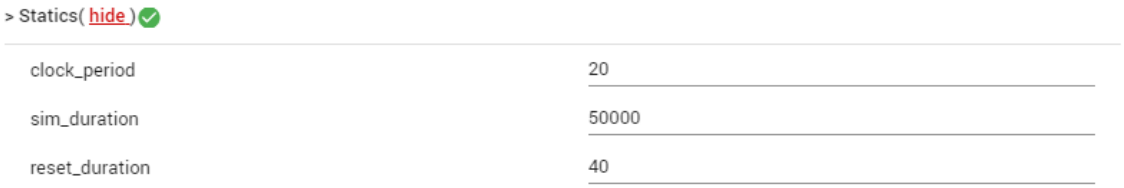

Figure 15. Static Options

The “clock_period” is the period of the CLB clock in nano-seconds for simulation purposes. The “sim_duration” field allows users to control the duration of the simulation run, again in nano-seconds. The “reset_duration” field allows the user to insert a delay (in nano-seconds) before the simulation becomes active to mimic the effect of a device reset.

4.1.2 Creating the Input Stimulus

Open the .syscfg file by double-clicking on the file name in the CCS Project Explorer Window. Expand the "Boundary" category by clicking on "Show".

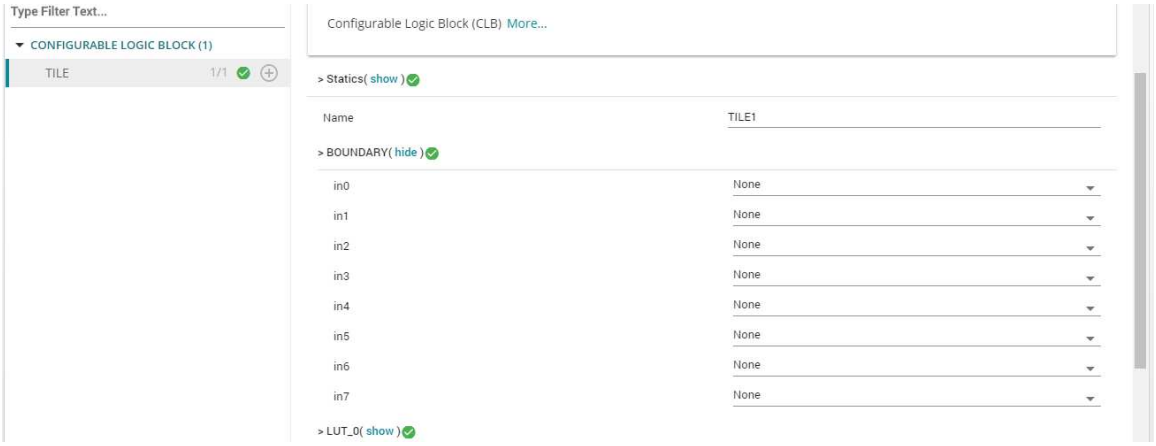

Figure 16. Boundary Input IN0 to IN7

A separate input stimulus can be defined for each of the eight CLB inputs using the drop-down menus. Click on the down-arrow on the right to reveal the options:

- None – the default option, does not generate any stimulus.
- squareWave – allows the user to define a periodic PWM input with configurable duty and phase.

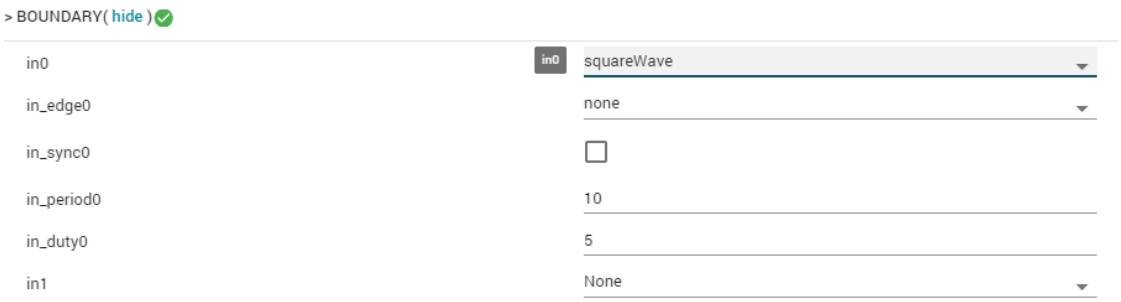

Figure 17. Boundary Input Square Wave

The "in-edge" option offers the user the choice of generating a pulse from the rising and/or falling edges of the PWM wave whose period and duty are set as 10 and 5 CLB clock pulses, respectively, in Figure 17. The "in_sync" check-box forces the input waveform to be synchronized to the CLB clock. For more information, see the *CLB input mux* section in the device-specific TRM.

- Custom – provides the ability for users to generate their own custom stimulus.

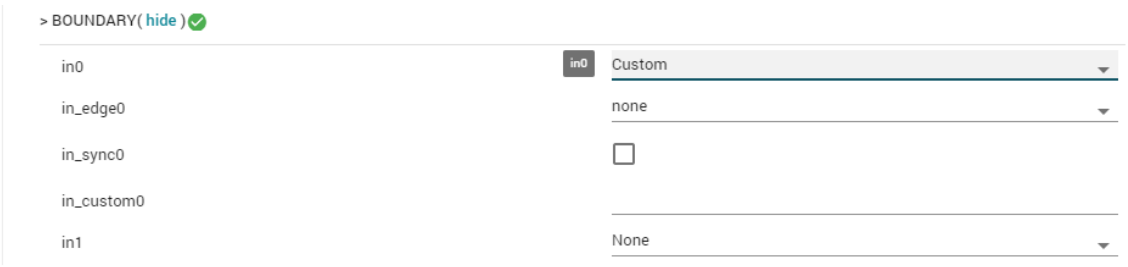

Figure 18. Boundary Input Custom

In order to use this option, the user must enter the systemC commands in the line marked “in_custom”. This is an advanced feature of the tool and no examples are provided in the current version.

4.1.3 Running the Simulation

Once the CLB configuration and input stimuli have been defined, the user can compile the project. The “CLB.vcd” file is generated after the post build steps have completed.

Assuming that the configuration of the waveform viewer has been completed, double-clicking on this file should open the viewer and allow the waveforms to be inspected. Figure 19 shows the GTKwave viewer set up to display a sample of input waveforms. For information on how to add and view signals, see the viewer documentation.

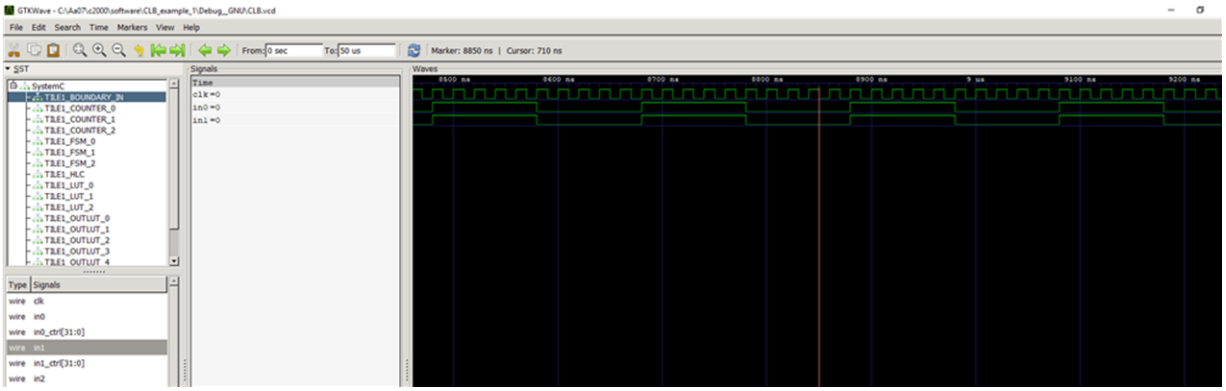

Figure 19. CLB Simulation Example

If the simulated waveforms do not match expectation, the user typically modifies the configuration in the .syscfg file and repeat the simulation.

5 Examples

This version of the CLB tool is supplied with examples showing how to configure the CLB to implement simple combinatorial and sequential logic circuits. Two sets of examples are supplied: for F28379D and F280049. There are minor code differences between these two sets, however, the instructions for running the examples are similar.

5.1 Basic Examples

The objective of these examples is to showcase the capabilities of the submodules inside each CLB TILE. They each describe a handful of submodules and how to implement simple logic using a combination of them.

5.1.1 CLB Empty Project

This example is an empty CLB project with post-build steps to generate the “.OUT” target binary, the simulation “.VCD” and the HTML block diagram.

5.1.2 Example 7 – State Machine

Designing With The C2000 Configurable Logic Block describes how to design an application using CLB by going through the design process step by step. This example uses all submodules inside a CLB TILE in order to implement a complete system.

5.1.3 Example 8 – External AND Gate

In this example, two external signals from two GPIOs are passed through the Input X-BAR and the CLB X-BAR to the CLB TILE. Inside the CLB module these two signals are ANDED. The output of the AND gate is then exported to a GPIO, using Output X-BAR.

5.1.4 Example 9 – Timer

In this example, a COUNTER module is used to create timed events. The use of the GP Register is shown. Through setting/clearing the bits in the GP register, the timer is started, stopped or changes direction. The output of the timer event (1-clock cycle) is exported to a GPIO. Interrupts are generated from the timer event using the HLC module. A GPIO is also toggled inside the CLB ISR. The indirect CLB register access is used to update the timer's event match value and the active counter register to modify the frequency of the timer.

5.1.5 Example 10 – Timer With Two States

In this example, the timer is setup the same as the previous example. The difference is the use of the FSM submodule to toggle the output of the CLB which is then exported to a GPIO. The FSM module acts as a single bit memory block. Interrupts are setup in the same format as the previous example. The interrupt delay of the CLB can be seen by comparing the output of the CLB and the GPIO toggled in the ISR.

5.1.6 Example 11 – Interrupt Tag

In this example, a timer is setup with two different match values. These two events are used by the HLC submodule to generate interrupts. The interrupt TAG is used to differentiate between the interrupt generated due to the match1 event of the CLB counter and the match2 event of the CLB counter.

5.1.7 Example 12 – Output Intersect

In this example, the CLB module is set up the same as the external_AND_gate example. However, instead of the output being exported to the GPIO using Output X-BAR, the output is exported to the GPIO by replacing the output of ePWM1. This is done by configuring the GPIO for EPWM1A output, followed by enabling output intersection.

5.1.8 Example 13 – PUSH-PULL Interface

In this example, the use of the PUSH-PULL interface is shown. Multiple COUNTER submodules, HLC submodule, FSM submodules, and OUTLUT submodules are used. The PUSH-PULL interface is used alongside the GP register to update the COUNTER submodules' event frequencies.

5.1.9 Example 14 – Multi-Tile

In this example the output of a CLB TILE is passed to the input of another CLB TILE. The output of the second CLB TILE is then exported to a GPIO, showcasing how two CLB TILEs can be used in series.

5.1.10 Example 15 – Tile to Tile Delay

In this example the output of a GPIO is taken into the CLB TILE through INPUT XBAR and the CLB XBAR. The signal is forwarded by the TILE to the next TILE. This time the signal only goes through the CLB XBAR and NOT the Input XBAR. This is done to show that delays are added when the signals are passed from TILE to TILE and the delay is NOT characterized. The user should always avoid passing signals with timing requirements between tiles. The COUNTER modules inside the CLBs will count the amount of delay in cycles.

5.2 Example 1 – Combinatorial Logic

The objective of this example is to prevent simultaneous high or low outputs on a PWM pair. PWM modules 1 and 2 are configured to generate identical waveforms based on a fixed frequency up-count mode. The time-base of PWM2 is synchronized to that of PWM1 as shown in Figure 20.

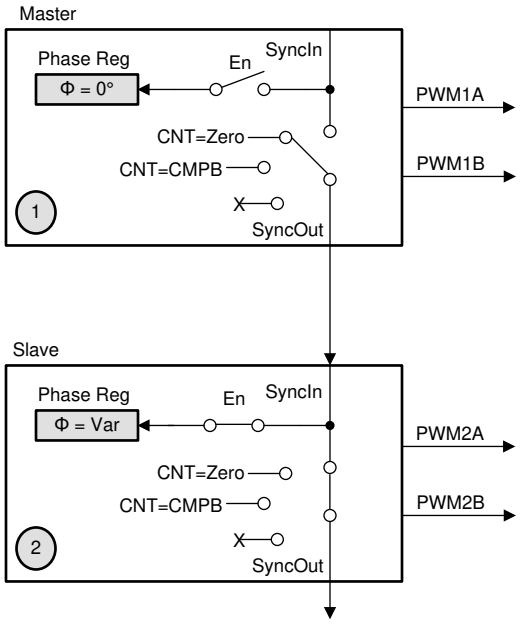

Figure 20. Example 9: EPWM Synchronization

The PWM waveforms are generated to deliberately force both outputs in each module to be simultaneously high and low at different times, as shown in Figure 21.

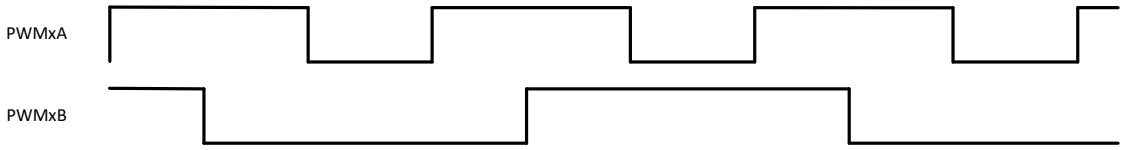

Figure 21. Example 10: PWM Test Pattern

The intention is to modify these waveforms with the CLB to remove either simultaneous high or simultaneous low conditions. This represents a simple combinatorial logic example. The logic operates in three modes: normal, active high, and active low. In normal mode, the PWM signals are passed through the CLB un-modified. In active high mode, the logic prevents logical '1' outputs from simultaneously appearing at the PWM pins. Similarly, in active low mode, logical '0' outputs must not appear on both PWM pins. The mode is selected using a 2-bit field as shown in Table 2.

Table 2. Example 1: Operating Modes

Mode Name	Type	[MODE 1]	[MODE 0]
M0	Normal	0	0
M1	Active Low	0	1
M2	Active High	1	0
M3	Reserved	1	1

The logic circuit which implements the patterns is shown in Figure 22. Output signals have "_m" appended to the name to indicate they may have been modified by the CLB.

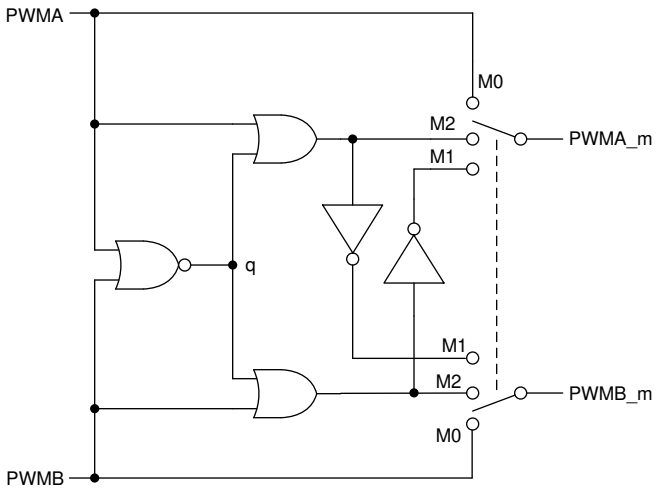

Figure 22. Example 1: Logic Diagram

The logic above can be implemented using two 4-input LUTs: one for each output signal. Therefore, only a small part of one CLB tile is involved. In the example, only the signals from the PWM1 module are modified by the CLB. The signals from PWM2 are carried directly to the device pins for comparison purposes. The input and output waveforms for PWM1 are shown in Figure 23 (modes 1 and 2).

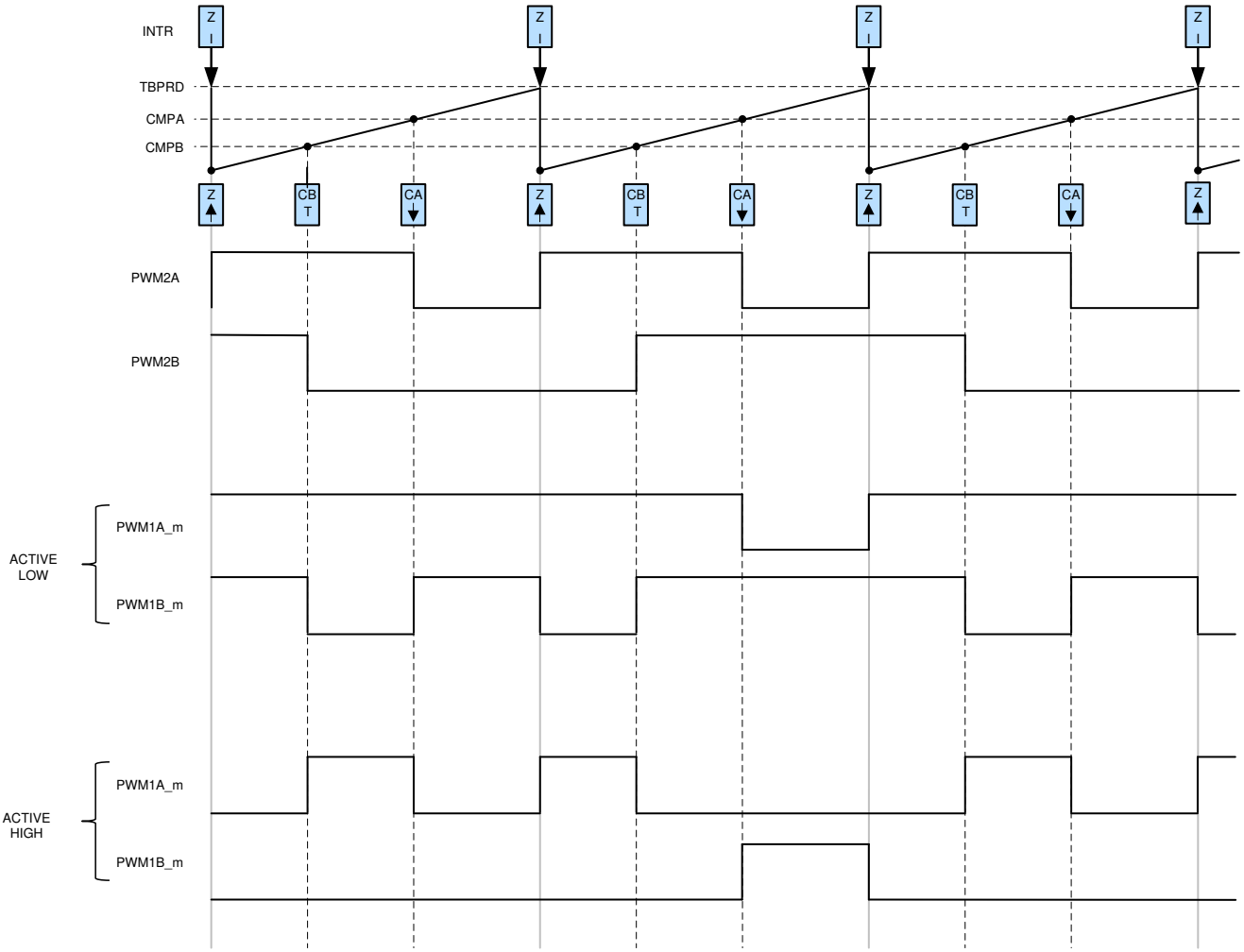

Figure 23. Example 1: Generated PWM

The required logic is implemented using 4-input LUTs 0 and 1. Each of these is connected to the two PWM signals, and the two LSBs of the software “mode” variable, which are written to the GPREG register. The CLB outputs are connected to the PWM1A and PWM1B signals which then go to GPIO pins and 1, respectively. Figure 24 conceptually shows the connections to and from the CLB tile.

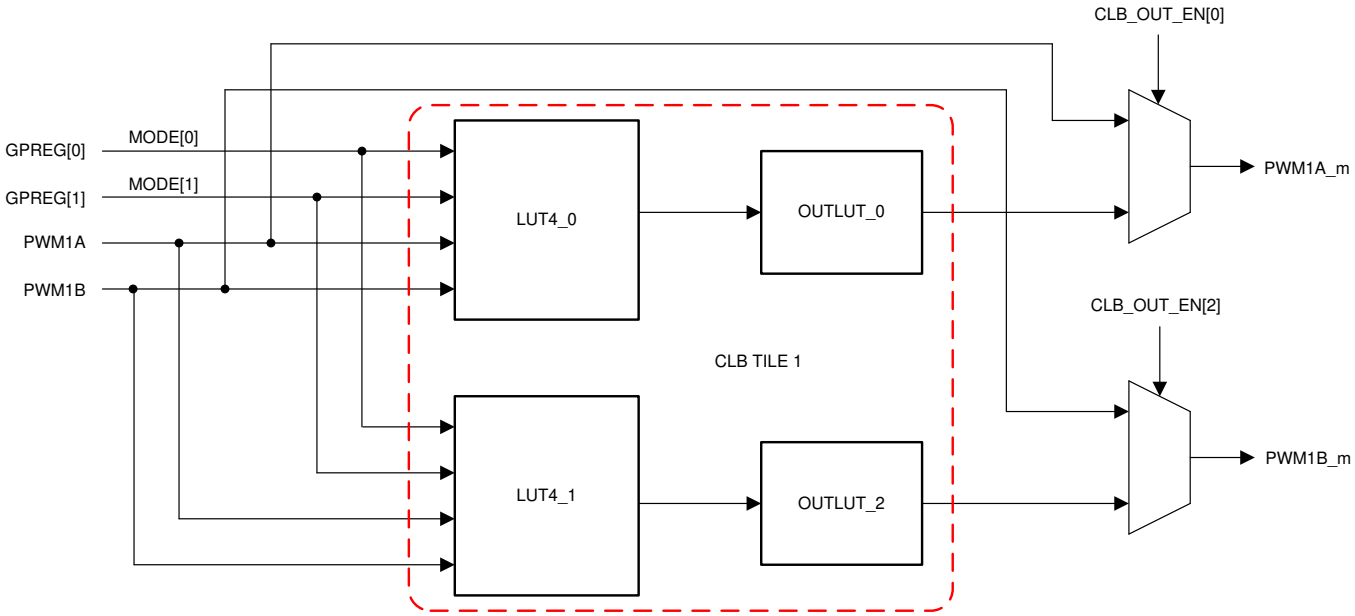

Figure 24. Example 1: CLB Configuration

To run the example, follow this procedure:

1. Click “Project → Import CCS Projects…”
2. Navigate to the CLB tool example directory. The path is:
   a. [C2000Ware]\driverlib\f2837xd\examples\cpu1\clb\ccs, or
   b. [C2000Ware]\driverlib\f28004x\examples\clb\ccs, or
   c. [C2000Ware]\driverlib\f2838x\examples\c28x\clb\ccs

   In the description that follows, it is assumed the C2000Ware directory above is in use.
3. Select the project “clb_ex1_combinatorial_logic” and click “Finish”.
4. In the CCS Project Explorer window, expand the project “clb_ex1_combinatorial_logic” and open the file “clb_ex1_combinatorial_logic.syscfg”.
5. Inspect the configuration of the tile and observe the logical expressions in LUT4_0 and LUT4_1, and the configuration of the output LUTs.
6. From the CCS menu, select “Project → Build Project”.
7. Monitor the pins.

   The Launchpad pins to watch the PWMs for the F28379D and Experimenter kit pins for F28388D are listed, but the Launchpad pins for the F28004x are not listed.
8. Open a CCS Expressions window ....
9. [Optional] – for instructions on how to run a simulation of the CLB, see Section 4.1.3.

If running the program on an F28379D LaunchPad board, PWM signals 1A and 1B can be monitored on pins J4/40 and J4/39, respectively. Set up an oscilloscope to monitor the signals at these pins while the program is running.

If running the program on an experimenter’s kit fitted with a F28388D controlCARD, the signals can be found on pins 49 and 51, respectively.

Open a CCS Expressions window and add the program variable “mode”. With mode set to the default value of 0, the PWM signals pass through the CLB without modification. Stop the program and change mode to 1, then restart the program. The signals should be as shown in the timing diagram above. Repeat this procedure to change the mode to 2 and verify the signals are as shown in the previous timing diagram.

5.3 Example 2 – GPIO Input Filter

This example demonstrates use of finite state machines (FSMs) and counters to implement a simple ‘glitch’ filter which might, for example, be applied to an incoming GPIO signal to remove unwanted short duration pulses.

Figure 25 shows in principle what the glitch filter does. An incoming digital signal is sampled at the CLB clock rate and a counter counts the number of consecutive samples where the input is either high or low. If this number is equal to or greater than a specified sample window, the filter output takes on the same value as the input; otherwise the filter output does not change. Figure 25 shows in principle what the filter does.

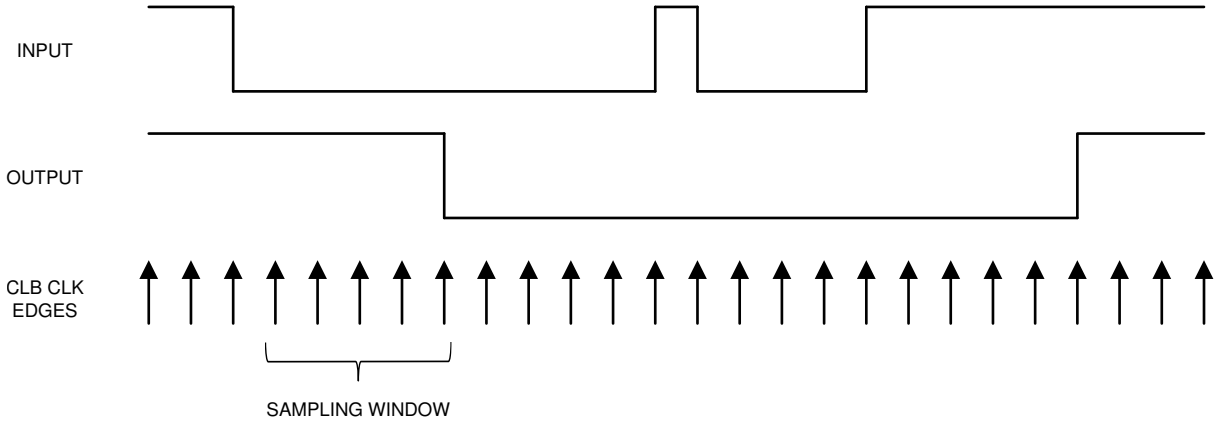

Figure 25. Example 2: GPIO Glitch Example

The CLB configuration uses one LUT4 to invert the incoming signal, and two counters to count the number of pulses: one counter for the high pulses, the other for low pulses. When either counter reaches the sample window length a pulse appears at its ‘match1’ output. In this example the filter sample window length is set to eight. An FSM latches the pulse and implements a simple logic equation to determine the required level at its ‘S0’ state output. One output LUT is used to convey the FWM output to the peripheral signal multiplexer for connection to GPIO0. The CLB configuration is shown in Figure 26.

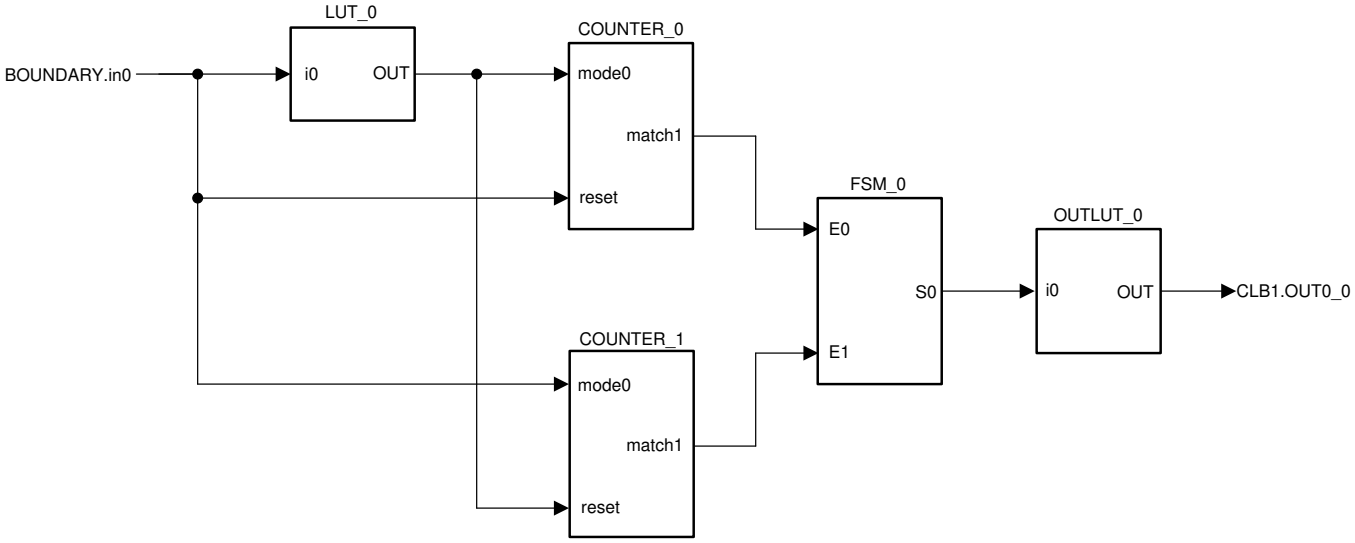

Figure 26. Example 2: CLB Configuration

The example code configures the ePWM1 module to generate the test stimulus.

To run the example, follow this procedure:

1. In CCS v9.0 or higher, click “Project → Import CCS Projects…”
2. Navigate to the CLB tool example directory. The path is:
   a. [C2000Ware]\driverlib\f2837xd\examples\cpu1\clb\ccs, or
   b. [C2000Ware]\driverlib\f28004x\examples\clb\ccs, or
   c. [C2000Ware]\driverlib\f2838x\examples\c28x\clb\ccs

   In the description that follows, it is assumed the C2000Ware directory above is in use.
3. Select the project “glitch_filter” and click “Finish”.
4. In the CCS Project Explorer window, expand the project “glitch_filter” and open the file “tile.syscfg”.
5. Inspect the configuration of the tile and observe the settings of the sub-modules LUT4_0, COUNTER_0, COUNTER_1, and FSM_0. Verify that the configuration matches that in the example description above.
6. From the CCS menu, select “Project → Build Project”.
7. [Optional] – for instructions on how to run a simulation of the CLB, see Section 4.1.3.

If running the program on an F28379D LaunchPad board, PWM signals 1A and 1B can be monitored on pins J4/40 and J4/39, respectively. Set up an oscilloscope to monitor the signals at these pins while the program is running. If running the program on an experimenter’s kit fitted with a F280049 or F28388D controlCARD, the signals can be found on pins 49 and 51, respectively.

Open a CCS Expressions window and add the program variable “cglitch”. Run the program while observing PWM signals 1A and 1B. Pause the program and change the value of “cglitch”, then re-start the program (this process is easier if the expressions window is set to run in “continuous refresh”). For values of 7 or less the glitch should be removed by the filter because its’ width is less than the sample window. When “cglitch” is higher than 7 the glitch should appear on both outputs. Notice also that edges on PWM1A have a small delay compared with those on PWM1B. This is a consequence of the filter method used.

Figure 27 shows the expected waveforms at the output pins for glitch widths below and above the sample window setting of 8.

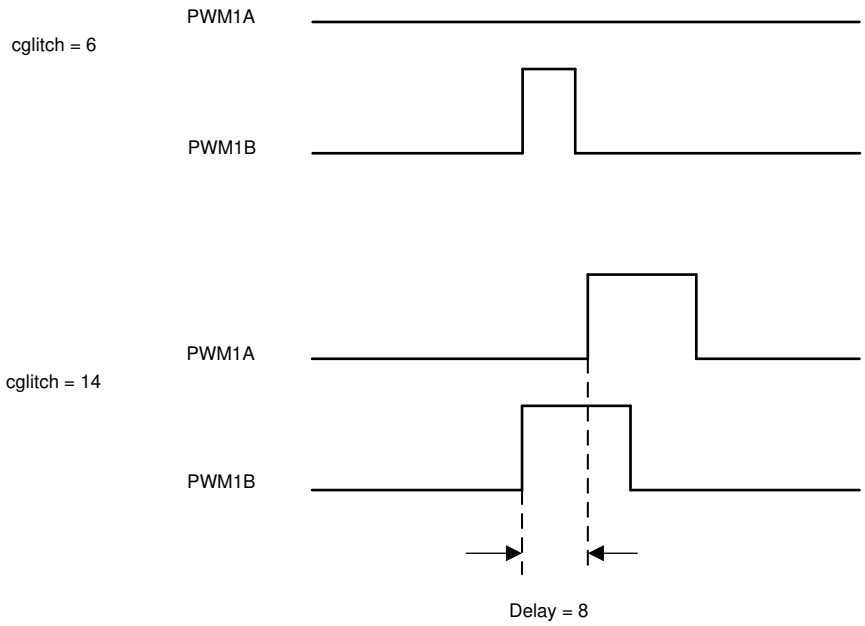

Figure 27. Example 2: GPIO Glitch Width

5.4 Example 3 – PWM Generation

This example configures a CLB tile as an auxiliary PWM generator. The exampleuses combinatorial logic (LUTs), state machines (FSMs), counters, and the high level controller (HLC) to demonstrate the PWM output generation capabilities using CLB.

The PWM generator operatesat the CLBCLK frequency. The FSM is used to set/clear the PWM. The PWM is set on a CMP match event, which is tied to match2 of the COUNTER_0. The PWM is cleared on a Zero match event (Z). This event is tied to the COUNTER_0 match1 output.

The PWM register is configured to use active and shadow registers, which is done using the HLC block. The HLC is used to generate an interrupt on the period match event, match1. When an interrupt occurs, a new counter match value is loaded into the HLC register (R0). The new counter match value is then moved into the match2 register of COUNTER_0. This updates the CMP match value, which in turn updates the value of the positive duty cycle. In this example, the user alternates between two values for the positive duty cycle. Figure 28 shows in principle what the PWM generator does. Notice how the duty cycle in the next period is changed.

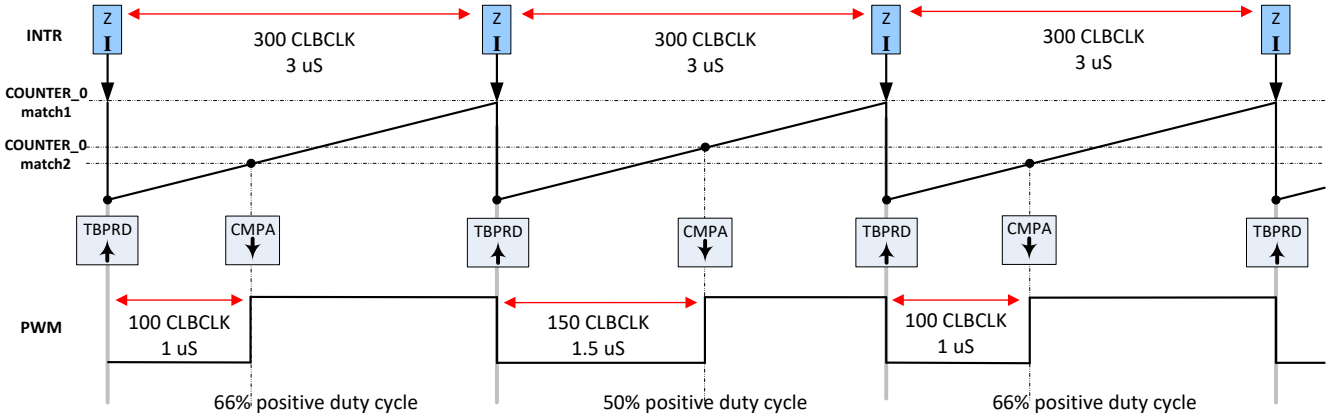

Figure 28. Example 3: Generated PWM Waveform

The CLB tile takes a PWM enable signal as input and generates an interrupt to the CPU. The CLB tile is configured to use a counter to count up until the desired period and compare event values are met. When the counter reaches the compare event match value, at output ‘match2’, the output is driven high and remains high until the counter value for the period match, at output ‘match1’, is met or a counter reset is triggered. When the period event or reset occurs, the counter is reset to 0 and the output is driven low and the counter begins counting up. This output logic is configured using the logical equation entered in the FSM. In this example, the period is 300 CLBCLK cycles (3 µs). The compare event occurs at either 100 CLBCLK cycles (1 µs) or 150 CLBCLK cycles (1.5 µs).

The PWM signal can be viewed by feeding the output of the FSM into OUTLUT_4. In order to view the output on a scope, it has to be transmitted via the Output X-BAR to the GPIO Mux.

To run the example, follow this procedure:

1. In CCS v9.0 or higher, click “Project → Import CCS Projects…”
2. Navigate to the CLB tool example directory. The path is:
   a. [C2000Ware]\driverlib\f2837xd\examples\cpu1\clb\ccs, or
   b. [C2000Ware]\driverlib\f28004x\examples\clb\ccs, or
   c. [C2000Ware]\driverlib\f2838x\examples\c28x\clb\ccs

   In the description that follows, it is assumed the C2000Ware directory above is in use.
3. Select the project “clb_ex3_auxiliary_pwm”, and click “Finish”.
4. In the CCS Project Explorer window, expand the project and open the file “clb_ex3_aux_pwm.syscfg”.
5. Inspect the configuration of the tile and observe the logical expressions in LUT4_0, COUNTER_0, FSM_0, and the configuration of the HLC and the output LUT.
6. From the CCS menu, select “Project → Build Project”.

7. View the CLB Tile block diagram by opening the "Debug/syscfg/clb.html" file
8. [Optional] – for instructions on how to run a simulation of the CLB, see Section 4.1.3.
9. To view the PWM and interrupt signals, set up an oscilloscope and monitor the following pins while the program is running. The table below shows the pin to monitor for each respective board.

Signal	F28379D LaunchPad	F280049 controlCARD	F28388D controlCARD
Interrupt	GPIO0 on pins J4/40	pin 49 (GPIO0)	pin 49 (GPIO0)
Auxiliary PWM	OutputXBAR1 signal on pins J4/34	pin 53 (OutputXBAR1)	pin 53 (OutputXBAR1)

10. Open a CCS Expressions window and add the program variable *dutyValue*. While the program is running, you will notice the CMPA value alternates between 100 and 150 CLBCLK cycles every time the CLB interrupt is serviced. The signals should be as shown in the timing diagram above. Notice that the PWM period remains the same but the positive duty cycle alternates between 50% and 66%. The *dutyValue* variable can be modified within the interrupt service routine.

5.5 Example 4 – PWM Protection

This example extends the features of example 1 to ensure an active high complementary pair PWM configuration always operates with a minimum value of dead-band irrespective of how the generating PWM module is configured. The example illustrates the configuration of four separate PWM tiles to implement PWM protection on four PWM modules. The outputs of PWM modules 1 to 4 are operated on by CLB tiles 1 to 4, respectively.

The protection functionality is enabled by the program variable “mode”. When set to 0 (the default condition), PWM signals are passed un-modified to the output pins. When set to 1, the PWM outputs are modified by the CLB to ensure dead-band.

To run the example, follow this procedure:

1. In CCS v9.0 or higher, click “Project → Import CCS Projects…”
2. Navigate to the CLB tool example directory. The path is:
   a. [C2000Ware]\driverlib\f2837xd\examples\cpu1\clb\ccs, or
   b. [C2000Ware]\driverlib\f28004x\examples\clb\ccs, or
   c. [C2000Ware]\driverlib\f2838x\examples\c28x\clb\ccs

   In the description that follows, it is assumed the C2000Ware directory above is in use.
3. Select the project “clb_ex4_pwm_protection”, and click “Finish”.
4. In the CCS Project Explorer window, expand the project “clb_ex4_pwm_protection” and open the file “clb_ex_pwm_protection.syscfg”.
5. From the CCS menu, select “Project → Build Project”.
6. Use an oscilloscope to observe the PWM signal pairs on the following pins of each LaunchPad/Docking Station board.

Table 3. Example 4: Signal Connections

PWM	Tile	F28379D LaunchPad	F280049 LaunchPad	F28388 Dock Station
1A	1	J4/40	J8/60	49
1B	1	J4/39	J8/59	51
2A	2	J4/38	J8/56	53
2B	2	J4/37	J8/55	55
3A	3	J4/36	J4/36	50
3B	3	J4/35	J4/35	52
4A	4	J8/80	J8/58	54
4B	4	J8/79	J8/57	56

7. Open a CCS Expressions window and add the program variable “mode”.
8. Run the program and verify that no dead-band exists between each PWM pair.
9. Halt the program, change mode to 1, and run the program again. You should now observe rising edge dead-band between each PWM pair. The dead-band time is set by the match_1 values loaded into the two CLB counters, and has been set arbitrarily to 10 in this example.

5.6 Example 5 – Event Window

This example uses the counter, FSM, and HLC sub-modules of the CLB to implement an event timing feature which detects whether an interrupt service routine takes too long to respond to an interrupt. The example configures four PWM modules to operate in up-count mode and generate a low-to-high edge on a timer zero match event. The zero match event also triggers a PWM ISR which, for the purposes of this example, contains a dummy payload of variable length. At the end of the ISR, a write operation takes place to a CLB GP register to indicate the ISR has ended.

The PWM timer zero event is detected by a CLB module where it starts a timer. The timer “match 2” count is set as the maximum expected duration of the corresponding PWM ISR. If the GP register write does not take place before the match 2 count is reached, the HLC triggers a CLB interrupt. Four PWM modules and CLB tiles are configured similarly.

Figure 29 gives an outline of how one tile operates. The upper half shows the configuration of the PWM module to generate a fixed frequency waveform with rising edge on each counter zero match, and falling edge on compare A match. The zero match event generates a CPU interrupt and the objective is to trigger a CLB interrupt if the PWM ISR does not complete within a specified time.

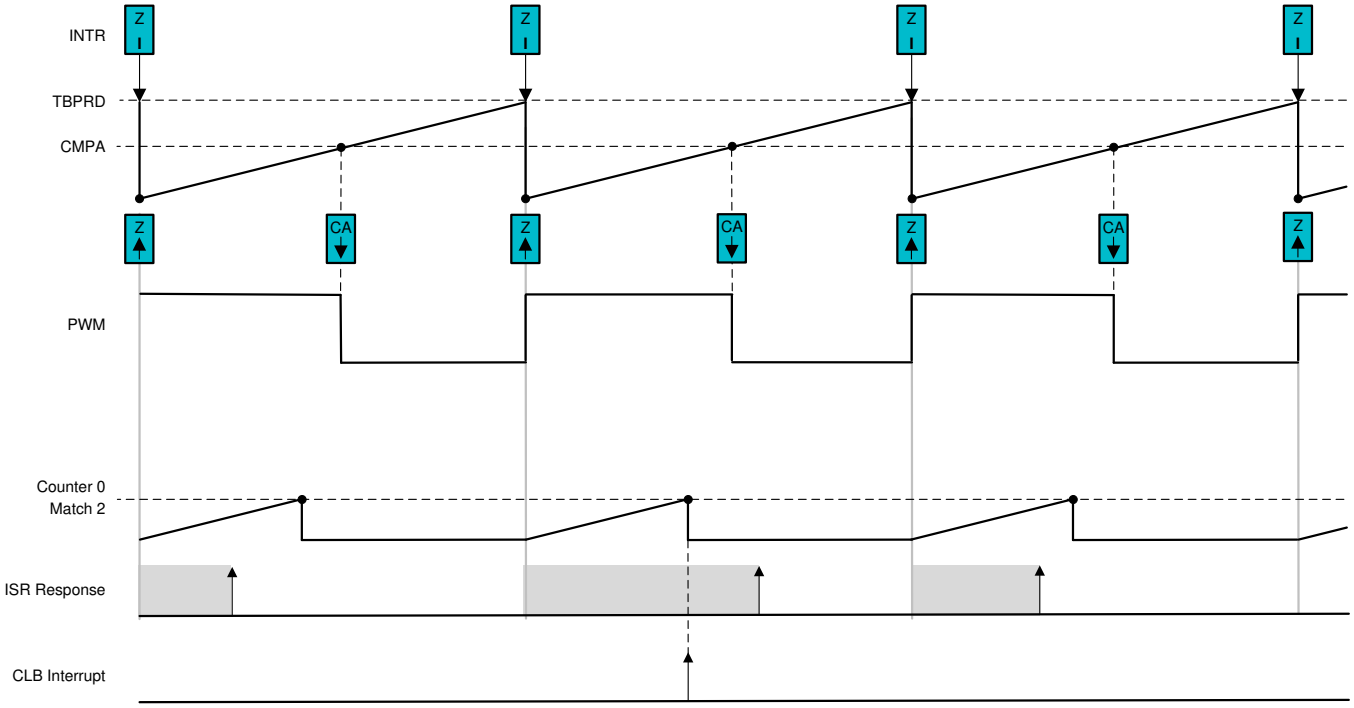

Figure 29. Example 5: Event Window Configuration

The lower half shows the CLB counter, which commences counting at the start of the PWM ISR. If the ISR does not respond before the Match 2 value is reached, an interrupt is generated. The CLB ISR contains an “ESTOP” instruction which acts like a software break-point in the program.

To run the example, follow this procedure:

1. In CCS v9.0 or higher, click “Project → Import CCS Projects…”
2. Navigate to the CLB tool example directory. The path is:
   a. [C2000Ware]\driverlib\f2837xd\examples\cpu1\clb\ccs, or
   b. [C2000Ware]\driverlib\f28004x\examples\clb\ccs, or
   c. [C2000Ware]\driverlib\f2838x\examples\c28x\clb\ccs

   In the description that follows, it is assumed the C2000Ware directory above is in use.
3. Select the project “clb_ex5_event_window”, and click “Finish”.
4. In the CCS Project Explorer window, expand the project “clb_ex5_event_window” and open the file “clb_ex5_event_window.syscfg”.
5. From the CCS menu, select “Project → Build Project”.

Open a CCS Expressions window and add the four program variables: “payload_x”, where ‘x’ is 1 to 4. Observe that at the start of the program, all payload variables have been set to 45. The payload is implemented as a ‘for’ loop in each PWM ISR, each iteration of which takes 12 cycles, so a payload of 45 corresponds to approximately 540 cycles.

Open the .syscfg file and inspect the match 2 settings in counter 0 of the four CLB modules. Notice that all timer limits are set to 3200.

Run the program with the default payloads and verify that the CLB interrupts to not trigger. Then, stop the program and increase any of the payloads. Re-run the program and determine whether any of the ISR limits is exceeded. Keep in mind that since the PWMs are not synchronized, the worst case ISR latency is the cumulative sum of all the payloads plus interrupt overheads.

5.7 Example 6 – Signal Generation and Check

This example uses CLB1 to generate a rectangular wave and CLB2 to check the rectangular wave generated by CLB1 doesn’t exceed the defined duty cycle and period limits.

CLB1: This example uses the counter and FSM sub-modules of the CLB to implement a rectangular pulse generator. The counter0 generates events on Match1 and Match2 values programmed by the user. While Match2 value defines the period of the waveform generated, (Match2 – Match1) value would determine the ON time. State machine uses these events from the counter to generate the waveform – set the output on Match1 and clear the output on Match2 event. Hence the state bit S0 reflects the output waveform generated. This output is in turn brought out on CLB1 output 4 in order to pass this output to CLB2 via CLB X-Bar. In0 is used as an enable from software for the waveform generation. This too is passed to CLB2 via CLB1 Output 5.

CLB2: This example uses the LUTs, counter, FSM, HLC sub-modules of the CLB to implement a checker on the output generated by CLB1. Following is the signal connectivity to CLB2.

CLB1 Output 4 → CLB X-Bar AUXSIG0 → CLB2 in1 (via Global Mux)

CLB1 Output 5 → CLB X-Bar AUXSIG1 → CLB2 in2 (via Global Mux)

The counter0 counts during the ON time of the received signal on In1. Counter0 Match1 value is set to the limit value on the duty cycle. If match1 event occurs it means that the ON time has exceeded the desired value.

The counter1 resets and starts counting on the rising edge of the received signal on In1. Counter1 Match1 value is set to the limit value on the period. If match1 event occurs it means that the Period has exceeded the desired value.

State machine (FSM1 S0) is used to detect the rising edge of the received signal on In1 and in turn used as reset to counter 1.

Whenever either of the counter match1 events described above occur there will be an interrupt generated to CPU using HLC – as an indicator of the error.

Figure 30 gives an outline of how the tiles operate. The match1 event generates a CPU interrupt and the objective is to trigger a CLB interrupt upon error condition detected inside CLB2.

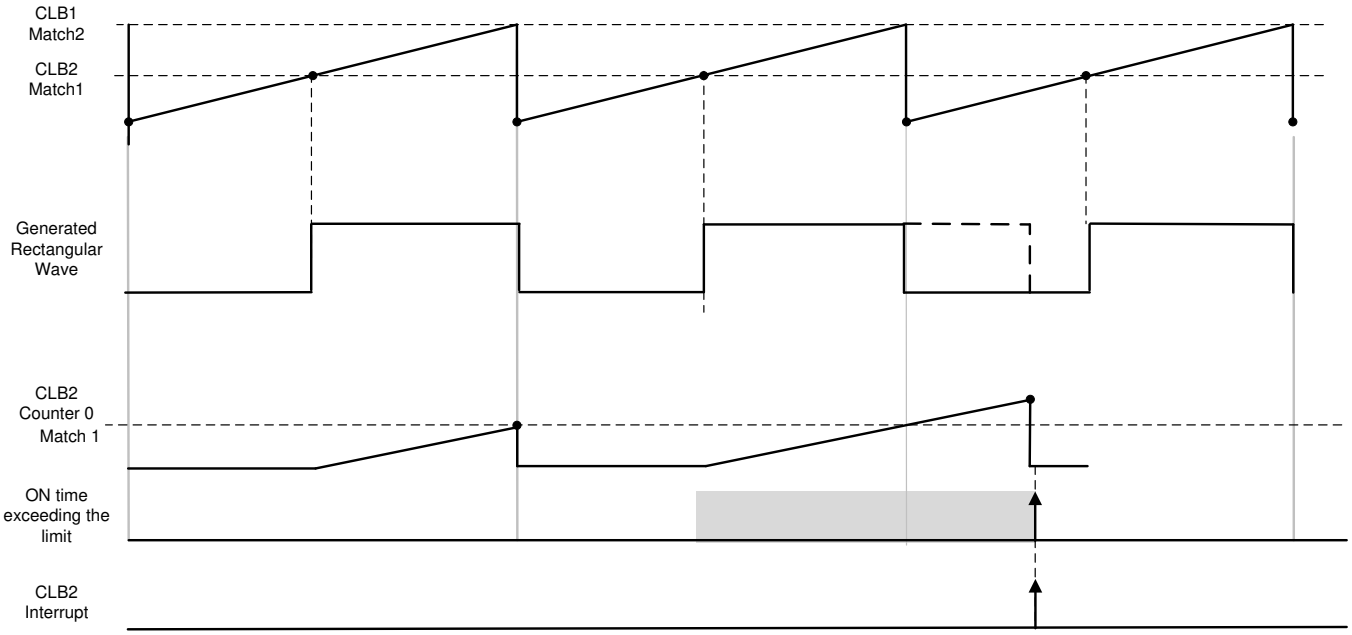

Figure 30. Example 6: Duty Exceeding Pre-Set Value

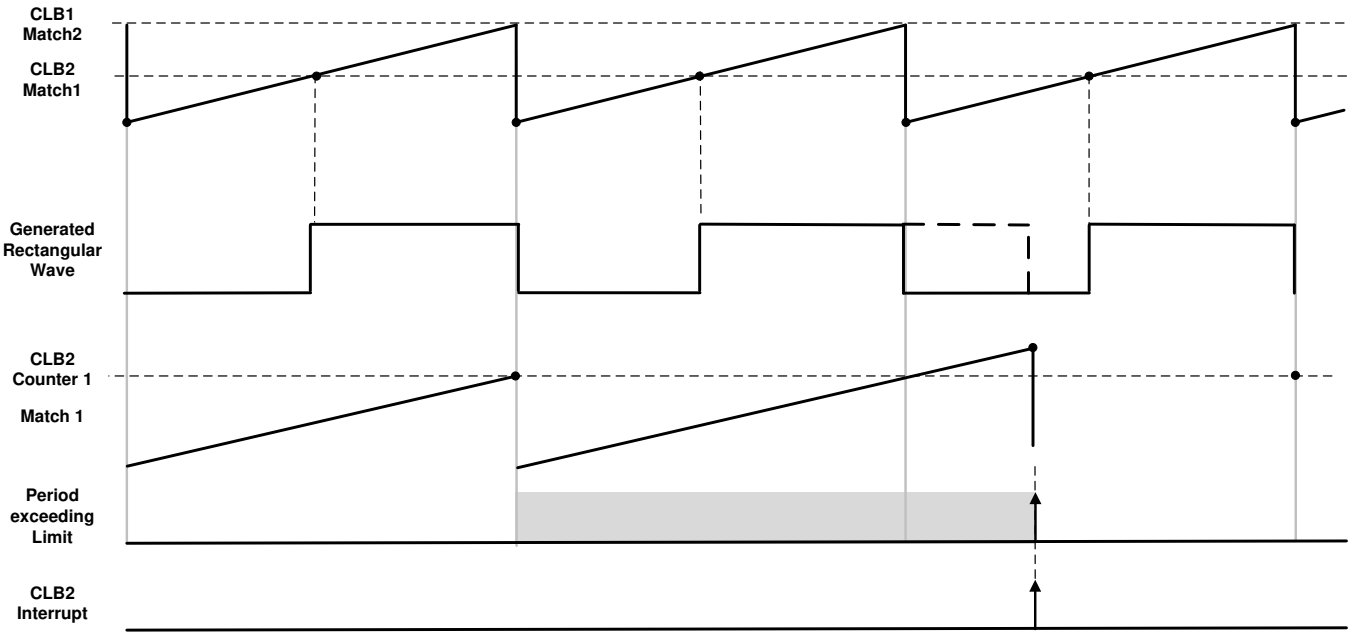

Figure 31. Example 6: Period Exceeding Pre-Set Value

The lower half shows the CLB counter, which commences counting at the start ON time. In the first figure, the duty cycle check and depicted and period check is depicted in the second. If the Match 1 value is reached, an interrupt is generated in either case. The CLB ISR contains an “ESTOP” instruction which acts like a software break-point in the program.

To run the example, follow this procedure:

1. In CCS v9.0 or higher, click “Project → Import CCS Projects…”
2. Navigate to the CLB tool example directory. The path is:
   a. [C2000Ware]\driverlib\f2837xd\examples\cpu1\clb\ccs, or
   b. [C2000Ware]\driverlib\f28004x\examples\clb\ccs, or
   c. [C2000Ware]\driverlib\f2838x\examples\c28x\clb\ccs

   In the description that follows, it is assumed the C2000Ware directory above is in use.
3. 13. Select the project “clb_ex6_siggen”, and click “Finish”.
4. 14. In the CCS Project Explorer window, expand the project “clb_ex6_siggen” and open the file “clb_ex6_siggen.syscfg”.
5. From the CCS menu, select “Project → Build Project”.

Open the SysCfg file (clb_ex6_siggen.syscfg) in the CCS window and inspect the match 1/2 settings in counter 0 of the CLB1 module. Change these values to update the duty and period of the generated output.

Inspect the match 1 settings of counter 0/1 in the CLB2 module. Change these values to update the duty and period values being checked on the generated output.

Run the program with the default values and verify that the CLB interrupts to not trigger. Then, change the values to result in an error (ex: change CLB2 Counter1 Match1 to 400). Rebuild and run the program to see the code stop inside the CLB2 interrupt service routine.

5.8 Example 17 – One-Shot PWM Generation

This example demonstrates how a CLB tile can be configured to act as a one-shot PWM generator. The example makes use of combinatorial logic (LUTs), state machines (FSMs), counters, and the HLC to demonstrate the one-shot PWM output generation capabilities on receipt of an external/software trigger.

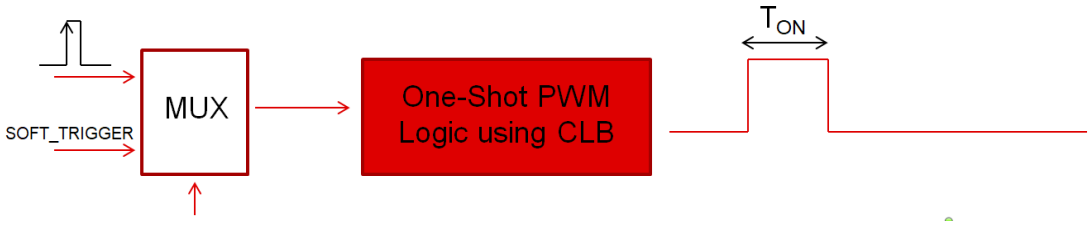

Figure 32. Missing Title

The CLB tile is configured to simulate a one-shot timer on receipt of a trigger the timer starts counting from zero, reaches the MATCH value and then stops counting till the next trigger is received. The output is driven HIGH while counter is counting and is driven LOW when counter reaches the MAX and stops counting. The above logic is implemented using LUT, FSM and counter. Another counter is used to make sure that the following system responds only to a rising edge event instead of input level. The example also supports variable pulse width using HLC submodule and CLB interrupt mechanism. The HLC is used to generate an interrupt after every 3rd trigger event (which is tracked by another counter) and the pulse width is updated by the application. The range of the output pulse width configured in the example is 0.2 µs - 0.8 µs with a step increase of 50 ns in every interrupt ISR. The PWM register is configured to use active and shadow registers, which is also done using the HLC block.

The overall CLB configuration can be visualized as shown in Figure 33.

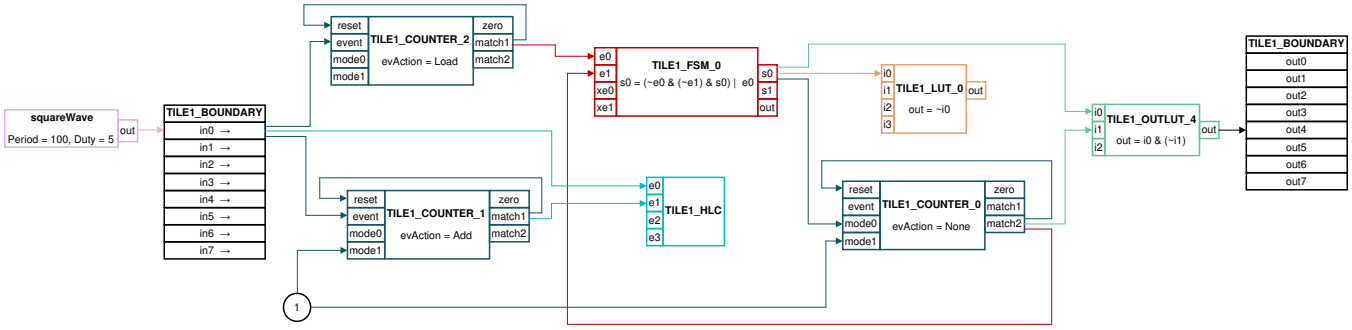

Figure 33. Example 17: Overall CLB configuration

The example supports two modes of configuration: software based trigger and external signal based trigger. The desired mode can be chosen by setting the EXAMPLE_MODE define as 0/1. In case of software based trigger, you can manually update the SOFT_TRIGGER from 0 to 1 in CCS expression window and observe the one-shot pulse output on oscilloscope. Note to make sure that the variable was set to '0' before setting it to '1', because the CLB system responds only to a rising edge. While in the case of external signal based trigger, the EPWM module is configured to generate a trigger signal of 1 MHz with a very short ON time (10% duty). This EPWM generated signal on GPIO0 is routed as the trigger input for CLB internally, thus no external connections are required.

To run the example, follow this procedure:

1. In CCS v9.0 or higher, click "Project -> Import CCS Projects…"
2. Navigate to the CLB tool example directory. The path is:
   a. [C2000Ware]\driverlib\f28004x\examples\clb\ccs, or In the description that follows, it is assumed the C2000Ware directory above is in use.
3. Select the project "clb_ex17_one_shot_pwm", and click "Finish".
4. In the CCS Project Explorer window, expand the project "clb_ex17_one_shot_pwm" and open the file "clb_ex17_one_shot_pwm.syscfg".
5. Inspect the configuration of the tile and observe the logical expressions in LUT_0 and FSM_0, COUNTER_0, COUNTER_1, COUNTER_2,HLC and the output LUT.
6. Configure EXAMPLE_MODE as 0/1 to operate in software/external trigger mode.
7. From the CCS menu, select "Project -> Build Project".
8. [Optional] – for instructions on how to run a simulation of the CLB, see Section 4.1.3.
9. Load the example on F28004x control card.
10. If software trigger mode is chosen:
    a. Add SOFT_TRIGGER variable to CCS expression window.
    b. Connect GPIO2 to oscilloscope and make sure the oscilloscope is in "One-shot" mode instead of "FREE_RUN".
    c. Run the example and set SOFT_TRIGGER = 1 in window, you should be able to observe the single pulse on oscilloscope.
    d. Set SOFT_TRIGGER = 0 first and then SOFT_TRIGGER = 1 to generate the next pulse.
    e. Repeat the above step every time to provide rising edge trigger.
    f. After every three triggers, the pulse would be increased by 50 ns.
11. If external trigger mode is chosen:
    a. Connect GPIO0 (trigger signal) and GPIO2 (output) to the oscilloscope and configure the oscilloscope in FREE_RUN mode.
    b. You should observe a linear variation in output pulse width after every 3rd rising edge of trigger signal.

6 Enabling CLB Tool In Existing DriverLib Projects

Use the following steps to add CLB support to an existing C2000WARE DriverLib Project:

1. Add the "empty.syscfg" file (For F2837xD [C2000Ware]\driverlib\f2837xd\examples\cpu1\clb\ccs\empty.syscfg) from the CLB examples folder to the project by copying the file into the project.
2. CCS will ask the user whether or not to enable SysConfig. Accept and select "Yes"

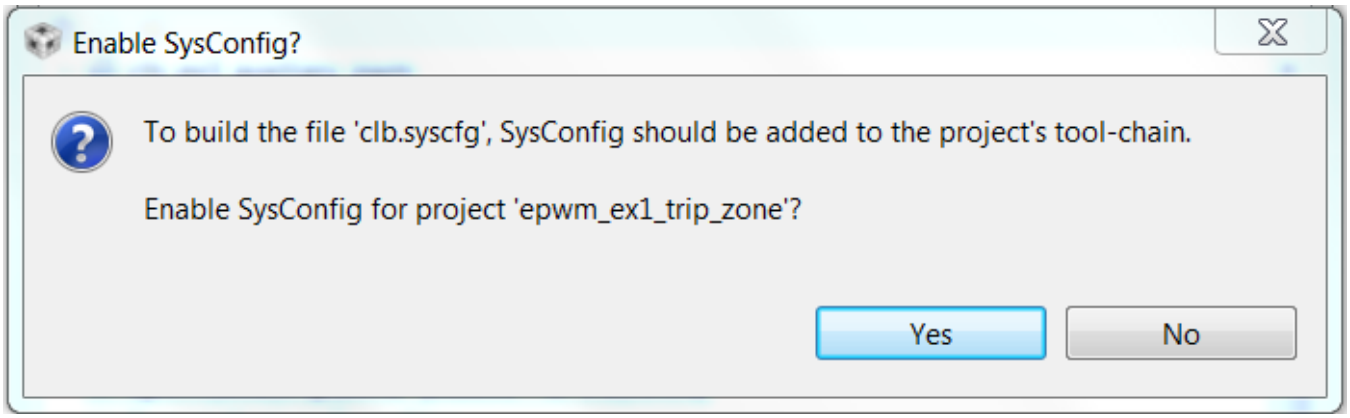

Figure 34. Enable SysConfig

3. Open the "Project Properties" and open the Resources → Linked Resources. Add the following Variable Paths,
   a. CLB_SYSCFG_ROOT
      [PATH_TO_C2000WARE]\utilities\clb_tool\clb_syscfg
   b. CLB_SIM_COMPILER
      C:\TDM-GCC-64\bin
   c. SYSTEMC_INSTALL
      [PATH_TO_C2000WARE]\utilities\clb_tool\clb_syscfg\systemc-2.3.3
   d. C2000WARE_ROOT
      [PATH_TO_C2000WARE]
4. In the Project Properties window, select Build → Steps
5. Add the following lines to the Post-build steps
   a. mkdir "${BuildDirectory}\simulation"
   b. ${CLB_SIM_COMPILER}\g++ -c -DCLB_SIM -I${SYSTEMC_INSTALL}\src -I${C2000WARE_ROOT}\utilities\clb_tool\clb_syscfg\systemc\include -I${PROJECT_ROOT} -I${CLB_SIM_COMPILER}\include -Og -g -gdwarf-3 -gstrict-dwarf -Wall -MMD -MP -MF${BuildDirectory}\simulation\clb_sim.d -MT${BuildDirectory}\simulation\clb_sim.o -I${BuildDirectory}\syscfg -fno-threadsafe-statics -o${BuildDirectory}\simulation\clb_sim.o ${BuildDirectory}\syscfg\clb_sim.cpp
   c. ${CLB_SIM_COMPILER}\g++ -DCLB_SIM -Og -g -gdwarf-3 -gstrict-dwarf -Wall -Wl,-Map,${BuildDirectory}\simulation\simulation_output.map -L${SYSTEMC_INSTALL}\build\src -o${BuildDirectory}\simulation\simulation_output.exe ${BuildDirectory}\simulation\clb_sim.o ${C2000WARE_ROOT}\utilities\clb_tool\clb_syscfg\systemc\src\CLB_FSM_SC_model.o ${C2000WARE_ROOT}\utilities\clb_tool\clb_syscfg\systemc\src\CLB_HLC_SC_model.o ${C2000WARE_ROOT}\utilities\clb_tool\clb_syscfg\systemc\src\CLB_LUT4_SC_model.o ${C2000WARE_ROOT}\utilities\clb_tool\clb_syscfg\systemc\src\CLB_OutputLUT_SC_model.o ${C2000WARE_ROOT}\utilities\clb_tool\clb_syscfg\systemc\src\CLB_counter_SC_model.o -Wl,--start-group -lsystemc -Wl,--end-group
   d. .\simulation\simulation_output.exe
   e. ${NODE_TOOL} "${CLB_SYSCFG_ROOT}/dot_file_libraries/clbDotUtility.js" "${CLB_SYSCFG_ROOT}" "${BuildDirectory}/syscfg" "${BuildDirectory}/syscfg/clb.dot"

6. The final Post-build steps should look similar to those in Figure 35.

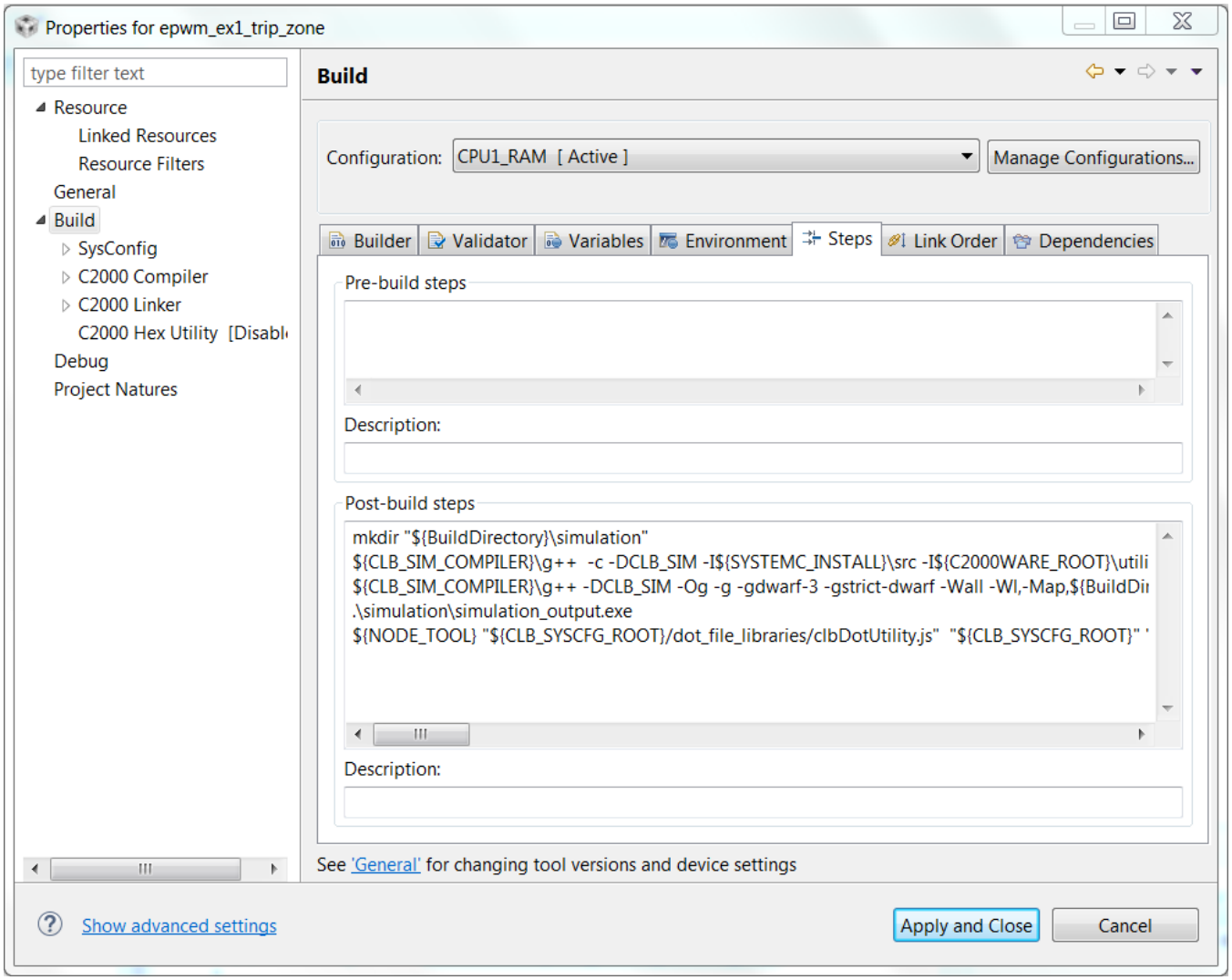

Figure 35. Post-build Steps

7. Next, open Build → SysConfig → Basic Options and add the following to the Root system config meta data list
   a. ${CLB_SYSCFG_ROOT}/.metadata/product.json

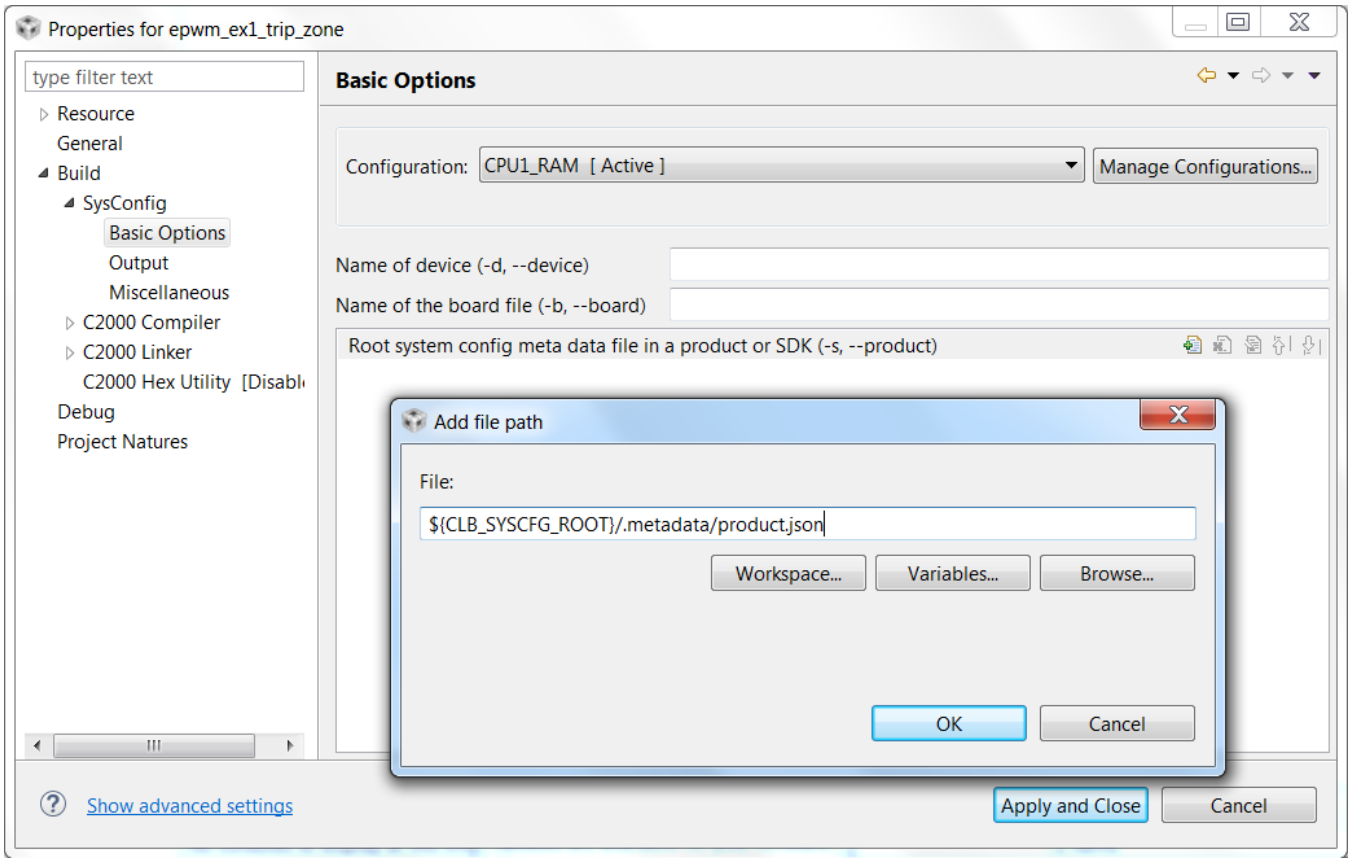

Figure 36. SysConfig SDK Path

8. Finally click Apply and Close

9. After building the project, the content generated by the CLB Tool will be present in the "Build Directory". Figure 37 shows an example of this after adding CLB support to the epwm_ex1_trip_zone driverlib example.

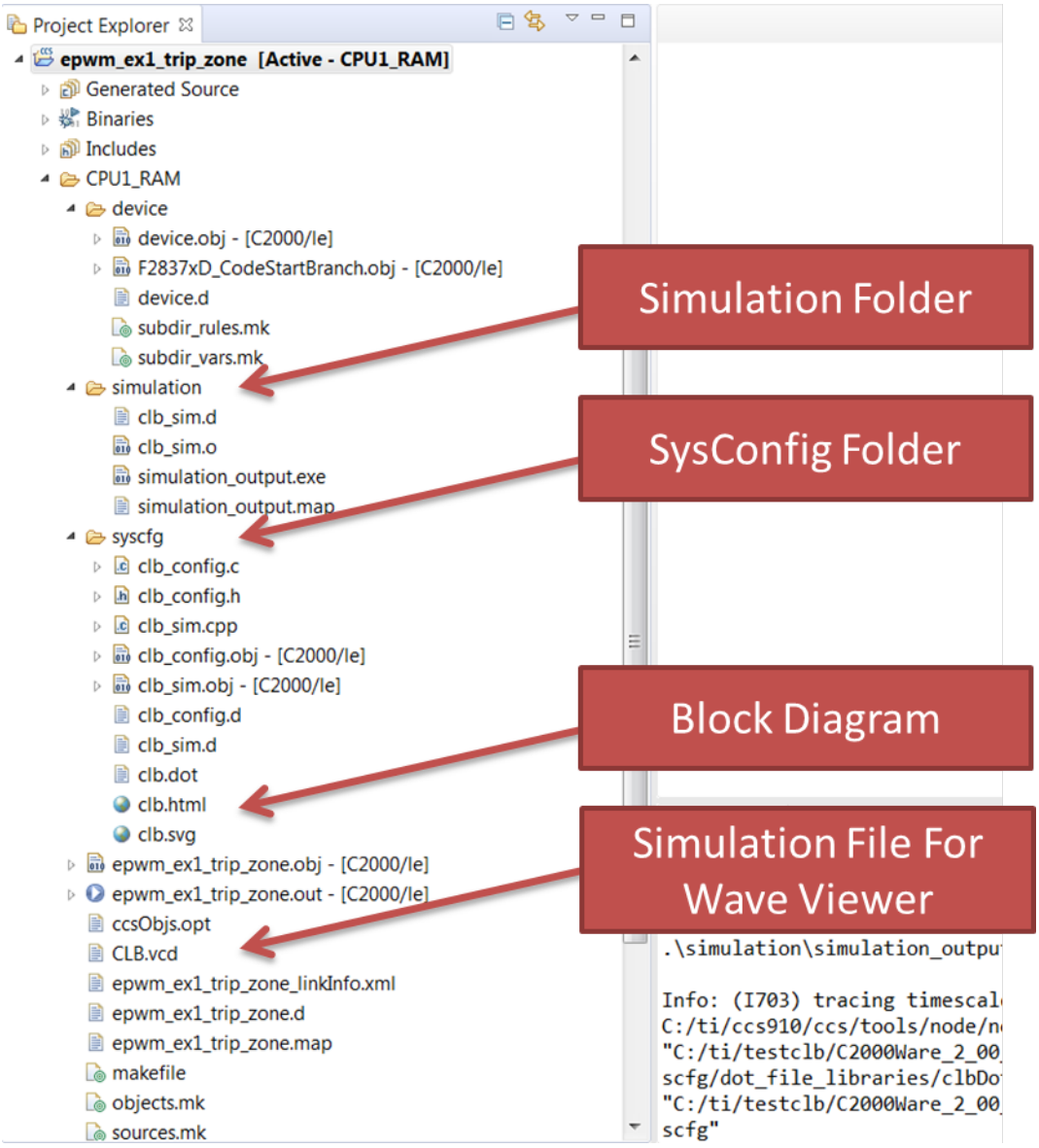

Figure 37. epwm_ex1_trip_zone With CLB Tool Support

7 Frequently Asked Questions (FAQs)

Question: My existing CLB project is not compatible with the latest CLB package. I observe the following build error in CCS problems window: “No such resource: /TILE.syscfg.js”.

Answer: Recent changes in the SysConfig now require the .syscfg file for the project to be modified and updated to reflect the new location of the TILE resource within the CLB package

NOTE: In order to update the file, you will need to modify the .syscfg file with a text editor.

Update the following line of code:

var TILE = scripting.addModule("/TILE");

replace with

var TILE = scripting.addModule("/utilities/clb_tool/clb_syscfg/source/TILE");

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Original (September 2019) to A Revision	Page
• Update was made in Section 2.2.2	6
• Updates were made in Section 5.2	17
• Added new Section 5.8	29

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265