

ABSTRACT

This user's guide describes the structure and use of the Configurable Logic Block (CLB) tool. It is assumed that the reader is already familiar with the architecture of the CLB and with the CCS IDE. Information on the architecture of the CLB may be found in the device-specific technical reference manual (TRM).

Table of Contents

1 Introduction	3
2 Getting Started	5
3 Using the CLB Tool	10
4 Examples	25
5 Enabling CLB Tool in Existing DriverLib Projects	41
6 Frequently Asked Questions (FAQs)	45
7 Revision History	45

List of Figures

Figure 1-1. Attaching Tile Design to CLB Instance.....	4
Figure 1-2. CLB Tool Project Structure.....	4
Figure 1-3. CLB Tool Build Process.....	5
Figure 2-1. TDM Compiler Installation Wizard.....	6
Figure 2-2. TDM License Change.....	7
Figure 2-3. TDM Compiler Path.....	7
Figure 2-4. TDM Components.....	8
Figure 2-5. SystemC Directory Creation.....	8
Figure 2-6. SystemC Configuration Output.....	9
Figure 2-7. Make Output.....	9
Figure 2-8. Make Install Output.....	10
Figure 3-1. Import CCS Eclipse Projects.....	10
Figure 3-2. Linked Resources.....	11
Figure 3-3. Build Variables.....	11
Figure 3-4. Build Variable to Generate Diagram.....	12
Figure 3-5. CLB Tool SysConfig Screen.....	12
Figure 3-6. Boundary Input Options.....	13
Figure 3-7. User Description Text Box.....	13
Figure 3-8. Counter Options.....	14
Figure 3-9. Equation Warning.....	14
Figure 3-10. CLB Tool Generated Files.....	15
Figure 3-11. "clb.h" Header File Example.....	15
Figure 3-12. CLB Block Diagram.....	16
Figure 3-13. HLC Configuration Example.....	16
Figure 3-14. GENERATE_DIAGRAM Build Variable.....	17
Figure 3-15. Static Options.....	17
Figure 3-16. Boundary Input 0 to 7.....	18
Figure 3-17. Boundary Input Square Wave.....	18
Figure 3-18. Boundary Input Custom.....	19
Figure 3-19. Boundary Input Tile Output.....	20
Figure 3-20. CLB Simulation Example.....	20
Figure 4-1. Example 3: Generated PWM Waveform.....	25
Figure 4-2. Example 1: EPWM Synchronization.....	27
Figure 4-3. Example 1: PWM Test Pattern.....	28
Figure 4-4. Example 1: Logic Diagram.....	28

Figure 4-5. Example 1: Generated PWM.....	29
Figure 4-6. Example 1: CLB Configuration.....	30
Figure 4-7. Example 2: GPIO Glitch Example.....	31
Figure 4-8. Example 2: CLB Configuration.....	31
Figure 4-9. Example 2: GPIO Glitch Width.....	32
Figure 4-10. Example 5: Event Window Configuration.....	34
Figure 4-11. Example 6: Duty Exceeding Pre-Set Value.....	36
Figure 4-12. Example 6: Period Exceeding Pre-Set Value.....	36
Figure 4-13. Example 17: One-Shot PWM Output.....	38
Figure 4-14. Example 17: Overall CLB configuration.....	38
Figure 5-1. Enable SysConfig.....	41
Figure 5-2. Post-Build Steps.....	41
Figure 5-3. Linked Resources for Enabling CLB Tool.....	42
Figure 5-4. SysConfig SDK Path.....	42
Figure 5-5. epwm_ex1_trip_zone With CLB Tool Support.....	44
Figure 6-1. Example of Simulation Warnings.....	45

List of Tables

Table 3-1. Supported Logical Operations.....	14
Table 3-2. Custom Waveform Code Instructions.....	19
Table 3-3. SystemC Top Level Trace Signals.....	21
Table 3-4. Asynchronous Output Conditioning Block Trace Signals (CLB Type 2).....	21
Table 3-5. Boundary Trace Signals.....	21
Table 3-6. Counter Block Trace Signals.....	21
Table 3-7. Finite State Machine Block Trace Signals.....	23
Table 3-8. High Level Controller Block Trace Signals.....	23
Table 3-9. Look-Up Table Block Trace Signals.....	24
Table 3-10. Output Look-Up Table Block Trace Signals.....	24
Table 4-1. Example 1: Operating Modes.....	28
Table 4-2. PWM Output.....	29
Table 4-3. Example 4: Signal Connections.....	33

Trademarks

C2000™ and Code Composer Studio™ are trademarks of Texas Instruments.
 All trademarks are the property of their respective owners.

1 Introduction

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 an FPGA. The CLB can enhance the functionality of existing device peripherals or create new peripheral functions. There are different versions of the CLB which have been released for devices in the past. Refer to the [C2000 Real-Time Control MCU Peripherals](#) for more information on what CLB type is available for a certain device. The CLB is configured using a software utility, referred to here as the CLB tool.

1.1 CLB Tool Outline

The CLB tool allows the user to configure and connect sub-modules in each CLB tile. The tool is included under the name Tile Design within the SysConfig graphical user interface (GUI) which is part of Code Composer Studio™ (CCS). The tool includes examples intended to help the user explore the features of the tool and 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. To configure the input and output connections between the CLB tile and other device peripherals, including the cross-bars and other CLB tiles, use the CLB module within SysConfig. The configuration of these connections must be done separately from the CLB tool. The CLB tool includes the capability for simulating these inputs and testing the functionality of the tile configuration.

1.2 Overview of the CLB Configuration Process

The CLB tool is based on the SysConfig tool in CCS. The generated C header and source files, combined with the C2000Ware SDK, can configure the CLB. To conduct a simulation of the design a number of third party tools need to be installed, including a compiler and a wave viewer. For more information on the CLB simulator, see [Section 3.5](#).

The CLB tool generates a “.dot” file which shows sub-module inter-connections in diagram form and can help verify the design. This file is created in HTML and SVG formats using CCS post-build steps. These steps use node.js and JavaScript libraries which are 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 done by the “clb_simulation” file which is generated by the tool. This file will be either a batch (.bat) or shell (.sh) file depending on the operating system used (the images used in this user's guide are based on the Windows operating system). This file must be executed to generate the appropriate “.vcd” file.

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 into the CLB module's registers. The CLB module within SysConfig must have the appropriate Tile Design attached for the configuration to actually take place, as shown in [Figure 1-1](#). The independence between Tile Designs and the CLB module in SysConfig allows any number of tile configurations to be created while having a limited number of CLB instances on a device. These Tile Designs are selectively loaded into the CLB.

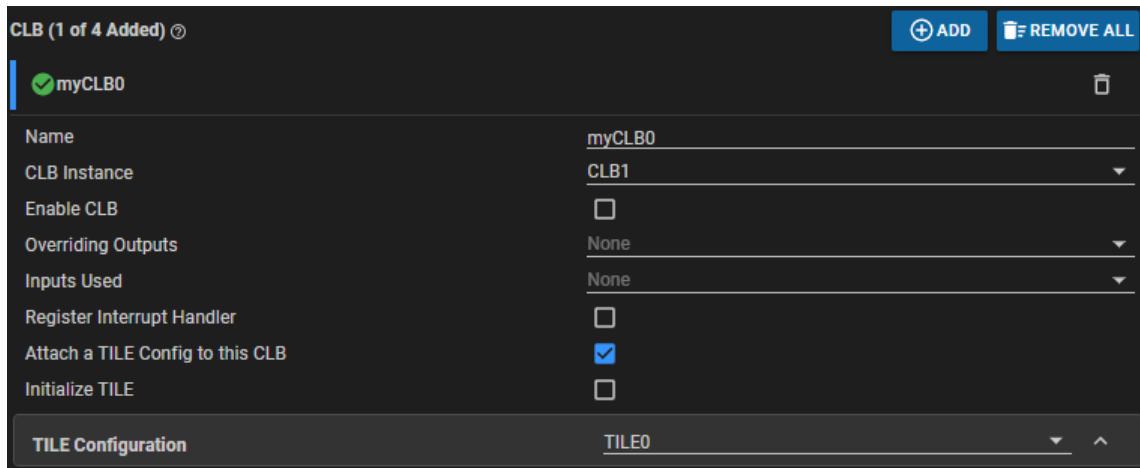


Figure 1-1. Attaching Tile Design to CLB Instance

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 1-2 shows the output of the CLB tool and the post-build steps.

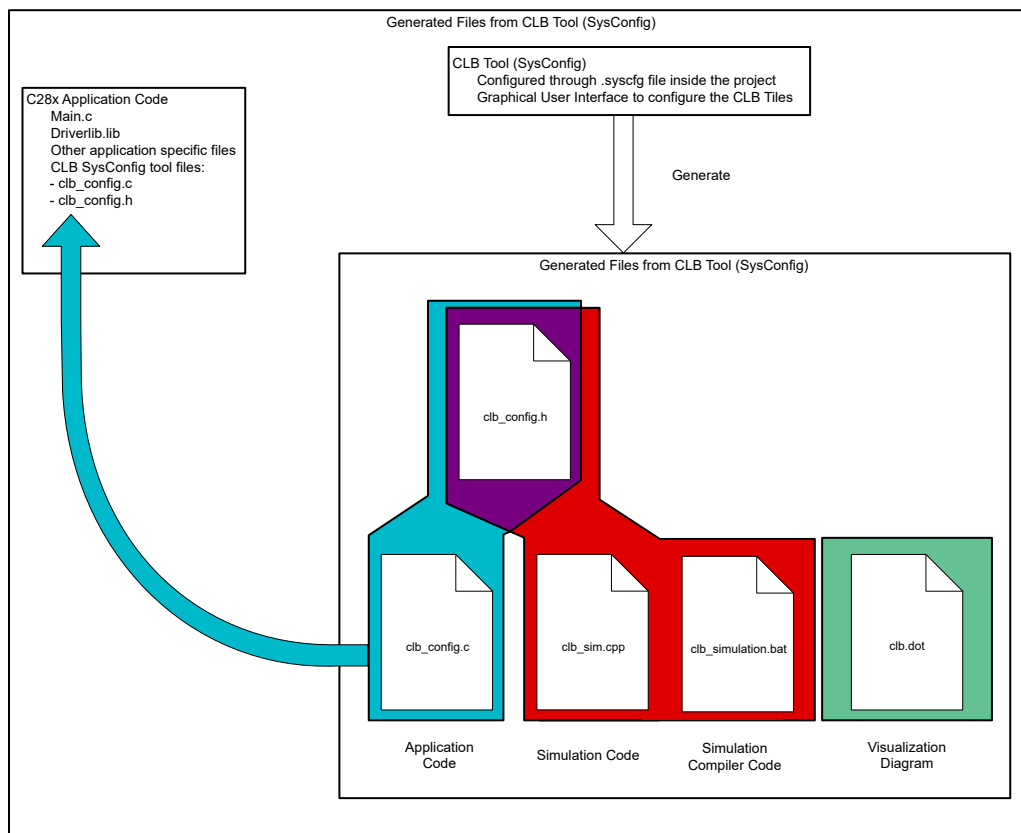


Figure 1-2. CLB Tool Project Structure

In a typical scenario, the user begins by specifying the CLB logic 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 choose 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 can modify the CLB tool settings and repeat the simulation.

Once the simulation produces the correct waveform, the user can download the design into the device using CCS to run or debug the code.

In a CCS project with SysConfig enabled for a C28x device, the steps to create the HTML and SVG block diagram of the Tile Designs is done as part of the post-build steps. When the user builds the CCS project, the user application code and the generated “clb_config.h” and “clb_config.c” are compiled using the C28x compilers and a “.out” file is generated.

The “clb_simulation” file compiles the generated simulation files, “clb_sim.cpp” and “clb_config.h”, and the CLB simulation modes using a GCC compiler. The output of this step is a “.exe” file (“simulation_output.exe”) which is automatically run to generate the “CLB.vcd” file. This file can be viewed using an external graph viewer.

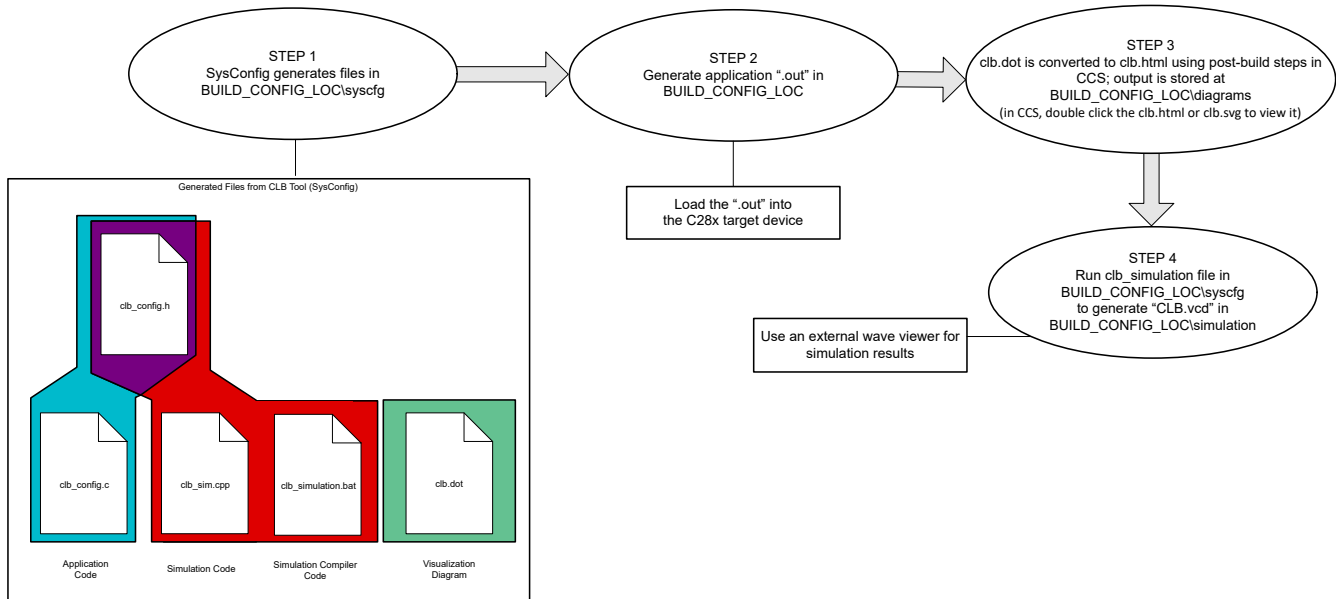


Figure 1-3. CLB Tool Build Process

In [CLB Tool Build Process](#) the BUILD_CONFIG_LOC directory is the directory used as the build configuration output for the project (this can be "Debug", "CPU1_RAM", and so forth), and shares the name of the active build configuration.

2 Getting Started

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

2.1 CLB Related Collateral

Foundational Materials

- [C2000 Academy - CLB](#)
- [C2000™ Configurable Logic Block \(CLB\) Series \(Video\)](#)
- [Customizing on-chip peripherals defies conventional logic](#)
- [Enable Differentiation and win with CLB in various applications Application Report](#)
- [Enable Differentiation with Configurable Logic in Various Automotive Applications \(Video\)](#)

Getting Started Materials

- [C2000™ Position Manager PTO API Reference Guide Application Report](#)
- [CLB Tool User Guide](#)
 - Basic examples are 7 - 15 (start with these). More involved examples are 1-6.
- [Designing With The C2000 Configurable Logic Block Application Report](#)
- [How do I add SYSCONFIG support \(Pinmux and Peripheral Initialization\) to an existing driverlib project?](#)
- [How to Migrate Custom Logic From an FPGA/CPLD to C2000 Microcontrollers Application Report](#)

- Chapters 1-3 are very useful for getting started and learning the CLB. Later chapters are very useful Expert materials for migrating from FPGA/CPLD to C2000's CLB.

Expert Materials

- [Achieve Delayed Protection for Three-Level Inverter With CLB Application Report](#)
- [Diagnosing Delta-Sigma Modulator Bitstream Using C2000™ Configurable Logic Block Application Report](#)
- [How to Implement Custom Serial Interfaces Using Configurable Logic Block \(CLB\) Application Report](#)
- [Tamagawa T-Format Absolute-Encoder Master Interface Reference Design for C2000™ MCUs](#)

2.2 Introduction

In order to use the tool, CCS version 9.0 or later must be installed. Earlier versions of CCS do not contain the SysConfig utility required for CLB configuration. For further information and download options for CCS, visit the [CCS download page](#).

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

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

2.3 Installation

2.3.1 Installation to Compile SystemC

To allow the simulation source file `clb_sim.cpp` to compile for Windows:

1. Download “tdm-gcc” version 5.1.0-2 from [SourceForge](#).
2. Open the downloaded file.
3. Uncheck the "Check for updated files on the TDM-GCC server" option.
4. Select “Create” from the setup wizard.

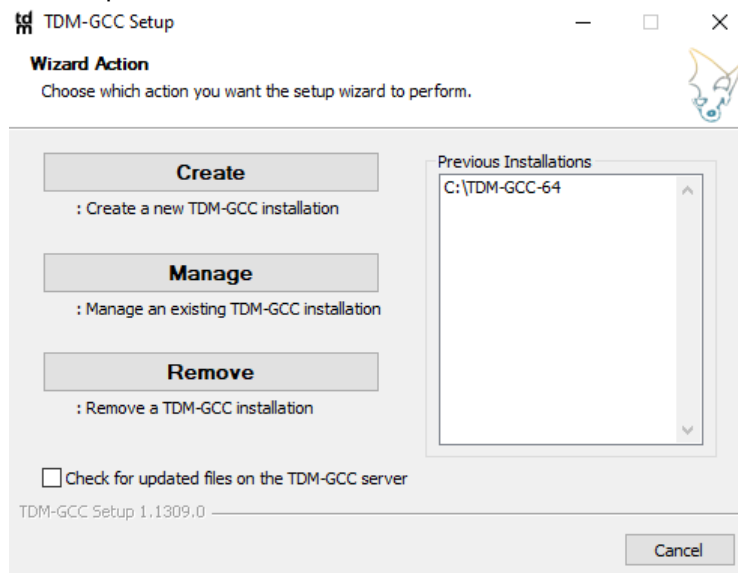


Figure 2-1. TDM Compiler Installation Wizard

5. If the wizard shows this information regarding license changes, select "Next".

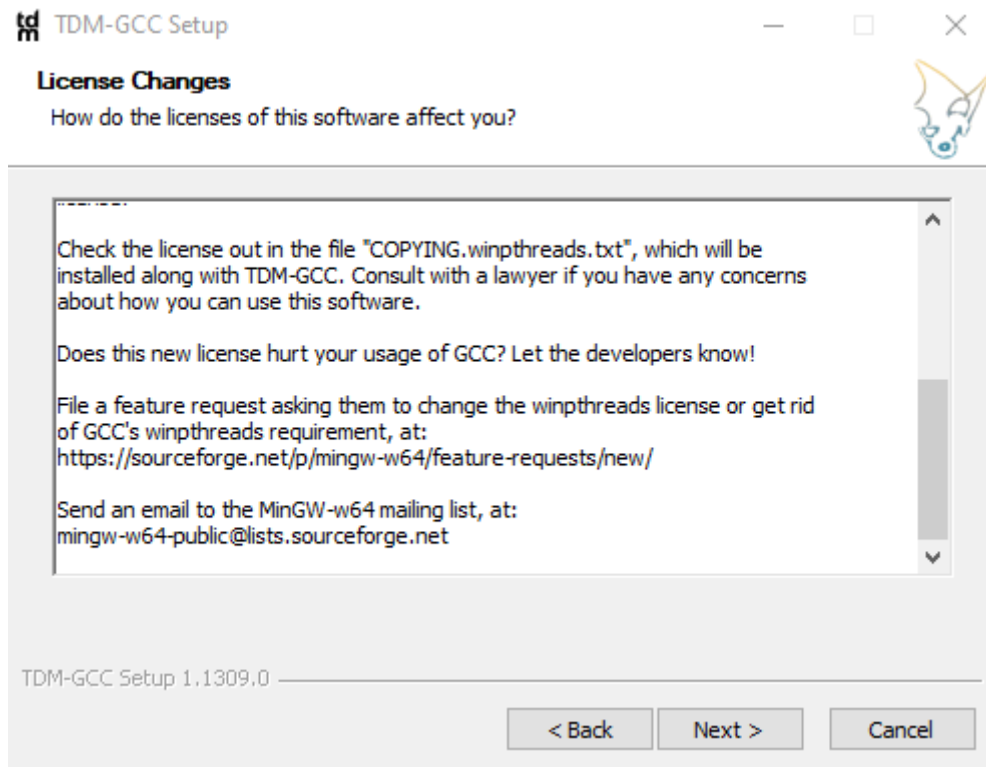


Figure 2-2. TDM License Change

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

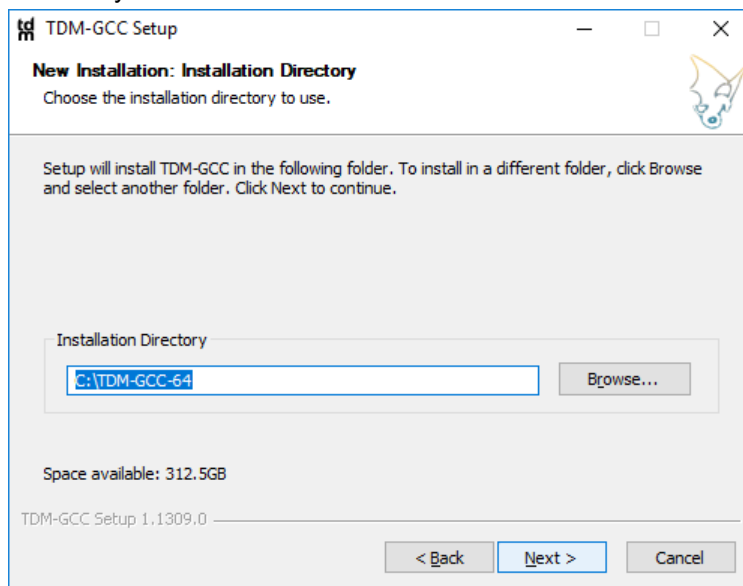


Figure 2-3. TDM Compiler Path

- Verify that the proper components are selected before clicking "Install".

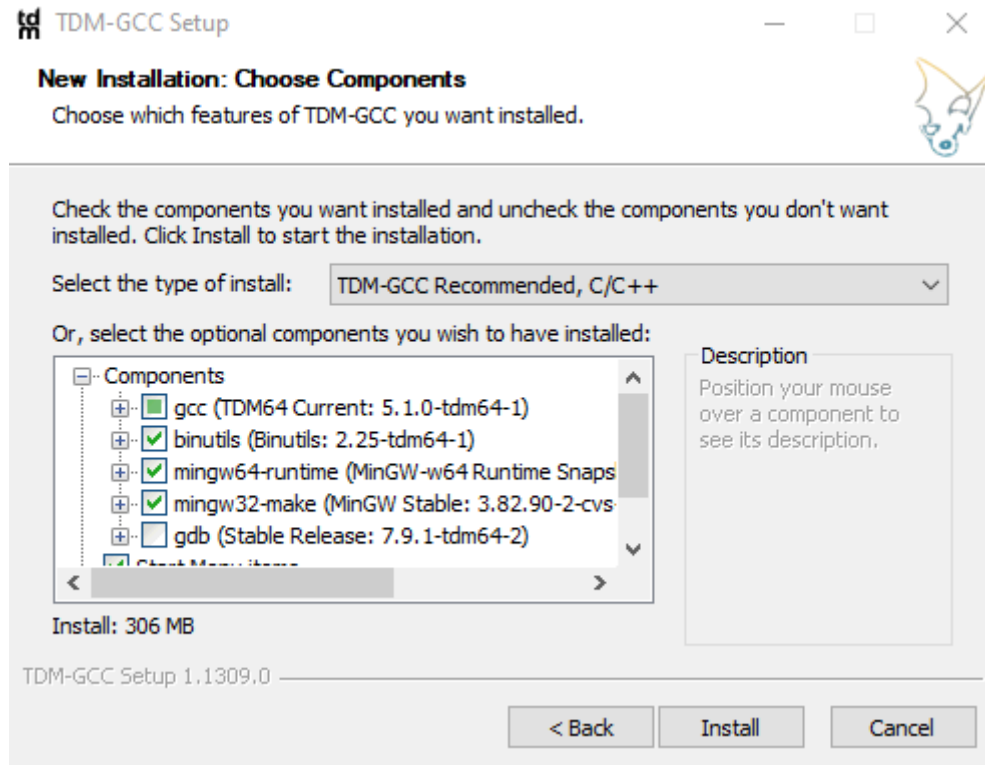


Figure 2-4. TDM Components

- Once the installation completes, select "Next" and "Finish".

For Mac or Linux, the SystemC library needs to be installed, but the G++ compiler does not. Verify that the G++ compiler is up-to-date before proceeding. To install SystemC for Mac or Linux:

- Open the terminal.
- Run `sudo apt-get install build-essential`.
- Install [SystemC 2.3.3](#) from Acclera and extract it by running `tar -xvf systemc-2.3.3.tar.gz` in the terminal.
- Copy this extracted folder into the "/usr/bin" directory by doing `sudo cp -r systemc-2.3.3 /usr/bin`. Go to the directory created by the tar command (not in "/usr/bin") and create a directory called "objdir".

```
~/Downloads$ sudo cp -r systemc-2.3.3 /usr/bin
~/Downloads$ cd systemc-2.3.3
~/Downloads/systemc-2.3.3$ mkdir objdir
~/Downloads/systemc-2.3.3$ cd objdir
```

Figure 2-5. SystemC Directory Creation

5. Run `sudo ./configure --prefix=/usr/bin/systemc-2.3.3/`.

```

-----
Configuration summary of SystemC 2.3.3 for x86_64-unknown-linux-gnu
-----

Directory setup (based on classic layout):
  Installation prefix (aka SYSTEMC_HOME):
    /usr/bin/systemc-2.3.3
  Header files   : <SYSTEMC_HOME>/include
  Libraries      : <SYSTEMC_HOME>/lib-linux64
  Documentation  : <SYSTEMC_HOME>/docs
  Examples       : <SYSTEMC_HOME>/examples

Architecture    : linux64
Compiler        : g++ (C/C++)

Build settings:
  Enable compiler optimizations : yes
  Include debugging symbols     : no
  Coroutine package for processes: QuickThreads
  Enable VCD scopes by default  : yes
  Disable async_request_update  : no
  Phase callbacks (experimental) : no
-----
  
```

Figure 2-6. SystemC Configuration Output

6. Run `sudo make`.

```

CXX      tracing/sc_wif_trace.lo
CXX      utils/sc_hash.lo
CXX      utils/sc_list.lo
CXX      utils/sc_mempool.lo
CXX      utils/sc_pq.lo
CXX      utils/sc_report.lo
CXX      utils/sc_report_handler.lo
CXX      utils/sc_stop_here.lo
CXX      utils/sc_string.lo
CXX      utils/sc_utils_ids.lo
CXX      utils/sc_vector.lo
CXXLD    libsysc_la
make[3]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/src/sysc'
make[2]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/src/sysc'
Making all in tlm_core
make[2]: Entering directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/src/tlm_core'
CXX      tlm_2/tlm_generic_payload/tlm_gp.lo
CXX      tlm_2/tlm_generic_payload/tlm_phase.lo
CXX      tlm_2/tlm_quantum/tlm_global_quantum.lo
CXXLD    libtlm_core_la
make[2]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/src/tlm_core'
Making all in tlm_utils
make[2]: Entering directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/src/tlm_utils'
CXX      convenience_socket_bases.lo
CXX      instance_specific_extensions.lo
CXXLD    libtlm_utils_la
make[2]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/src/tlm_utils'
Making all in .
CXXLD    libsystemc_la
make[2]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/src'
make[1]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/src'
Making all in examples
make[1]: Entering directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples'
Making all in sysc
make[2]: Entering directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples/sysc'
GEN      copy-check-data
To compile and run the examples type
  make check
make[2]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples/sysc'
Making all in tlm
make[2]: Entering directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples/tlm'
Making all in common
make[3]: Entering directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples/tlm/common'
GEN      copy-check-data
To compile the TLM examples library type
  make check
make[3]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples/tlm/common'
Making all in .
make[3]: Entering directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples/tlm'
GEN      copy-check-data
make[3]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples/tlm'
To compile and run the examples type
  make check
make[2]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples/tlm'
make[2]: Entering directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples'
GEN      copy-check-data
make[2]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples'
make[1]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir'
make[1]: Nothing to be done for 'all-am'.
make[1]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir'
~/Downloads/systemc-2.3.3/objdir$
  
```

Figure 2-7. Make Output

7. Run `sudo make install`.

```

make[3]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples'
make[2]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples'
make[1]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples'
make[1]: Entering directory '/home/a0498187/Downloads/systemc-2.3.3/objdir'
make[2]: Entering directory '/home/a0498187/Downloads/systemc-2.3.3/objdir'
make[2]: Nothing to be done for 'install-exec-am'.
/bin/mkdir -p '/usr/bin/systemc-2.3.3'
/usr/bin/install -c -m 644 ../AUTHORS ../NOTICE ../ChangeLog ../INSTALL ../LICENSE ../NEWS ../README ../RELEASENOTES ../cmake/INSTALL_USING_CMAKE '/usr/bin/systemc-2.3.3'
make[2]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir'
make[1]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir'
~/Downloads/systemc-2.3.3/objdir$

```

Figure 2-8. Make Install Output

2.3.2 Install the Simulation Viewer

To install the GTKWave on Windows:

1. Download the waveform viewer GTKwave from [SourceForge](#)
2. Download the native binaries for the correct Windows installation, (for 64-bit Windows, select “gtkwave-3.3.100-bin-win64”), and extract the downloaded zip file into the directory `c:\gtkwave`

To install GTKWave on Mac or Linux:

1. From the terminal, run the command `sudo apt install 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 example projects with the CLB enabled are available in `<C2000WARE_INSTALL>/driverlib/<device>/examples`.

For example, for the F2837xD device the path to the 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’ and click ‘Refresh’, or select the ‘Browse’ button to choose the proper folder directly.
3. Select the ‘clb_empty’ project.
4. Check ‘Copy projects into workspace’.
5. Click ‘Finish’.

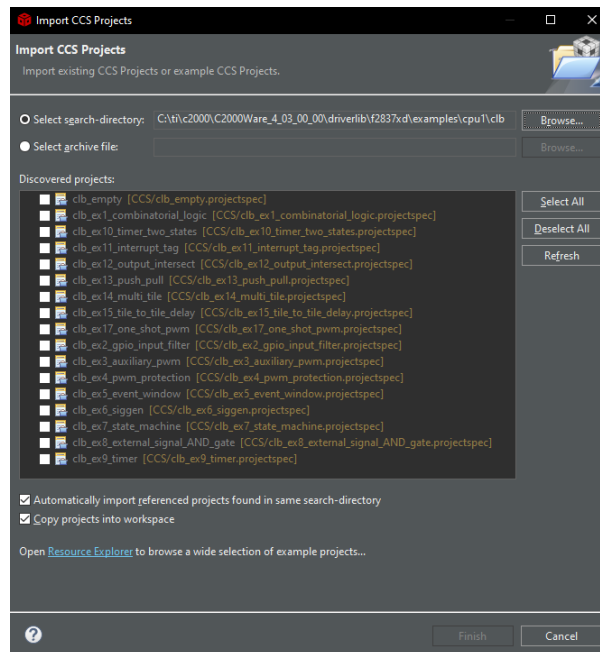


Figure 3-1. Import CCS Eclipse Projects

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 device, but also has the capability to generate the simulation files and the HTML/SVG block diagram of the design. To create the diagrams using the post-build steps, the proper path must be set for the location of the C2000Ware root and Node tool. To double check that these paths are correct:

1. Right click on the project and select ‘Project Properties’.
2. Under ‘Resources’, select ‘Linked Resources’.
3. Check to make sure the path below is correct:
 - a. C2000WARE_ROOT (This is used for the CLB diagrams and for other include paths)

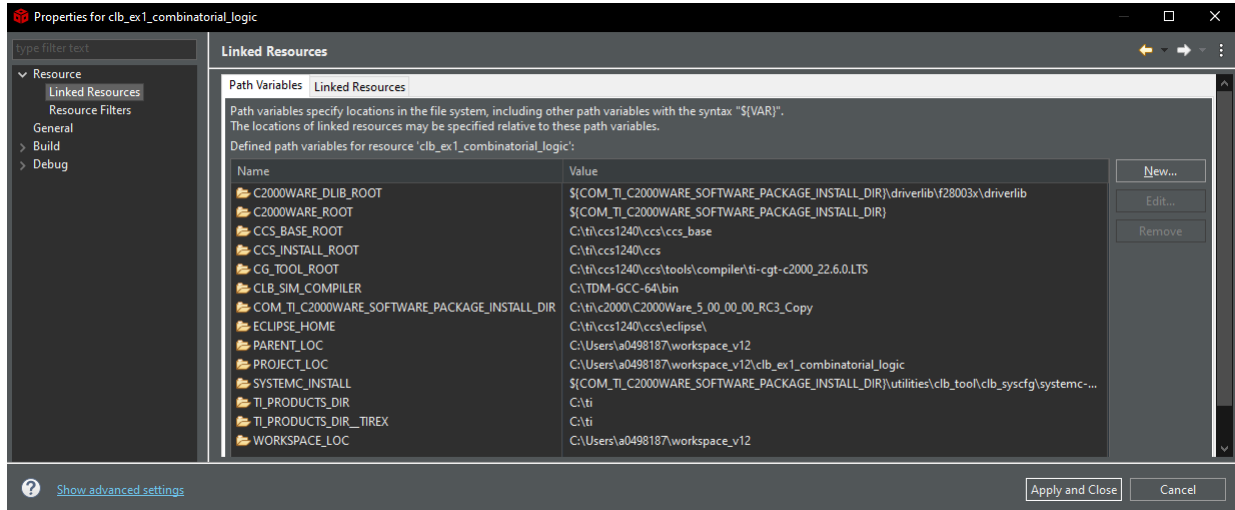


Figure 3-2. Linked Resources

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
5. Check to make sure the path for the below system variable is correct:
 - a. NODE_TOOL

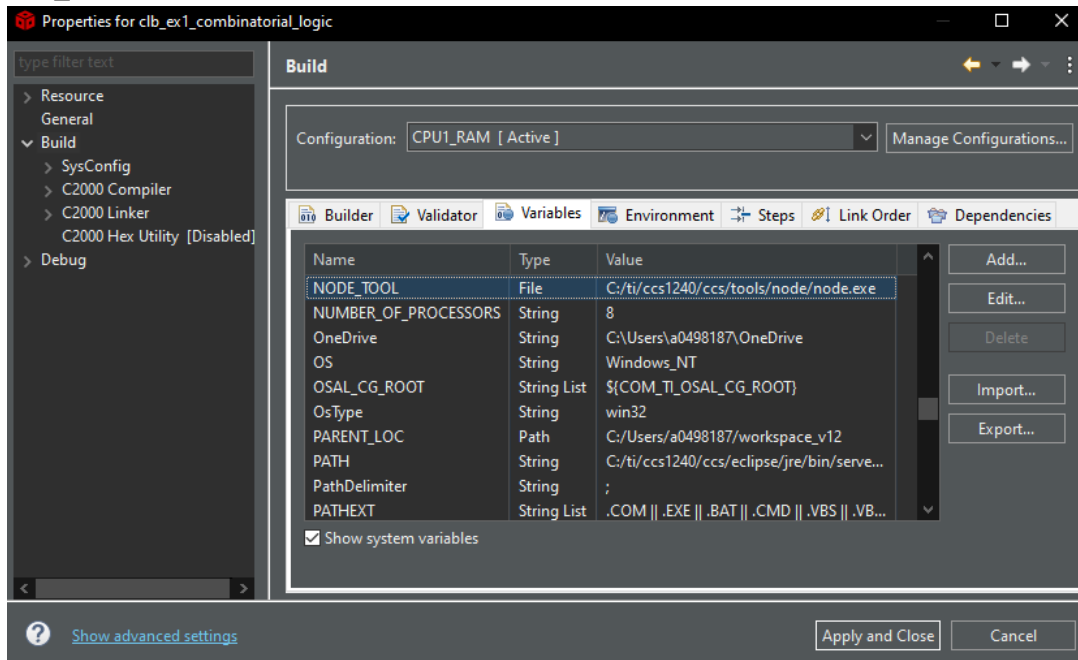


Figure 3-3. Build Variables

To generate the block diagram, the build variable `GENERATE_DIAGRAM` must be set to 1. The build variables for a project can be found by going to Project Properties > Build > Variables, as seen in [Figure 3-4](#). This variable enables the post-build steps listed under 'Steps' to run after the project is built. The block diagrams can be found under the relevant build configuration of the project in the 'diagrams' directory, as seen in [Figure 5-5](#).

Note

For Mac and Linux, the conditionals used in the post-build steps will not be able to execute properly. To generate the diagram, the test `if ${GENERATE_DIAGRAM} == 1` must be removed.

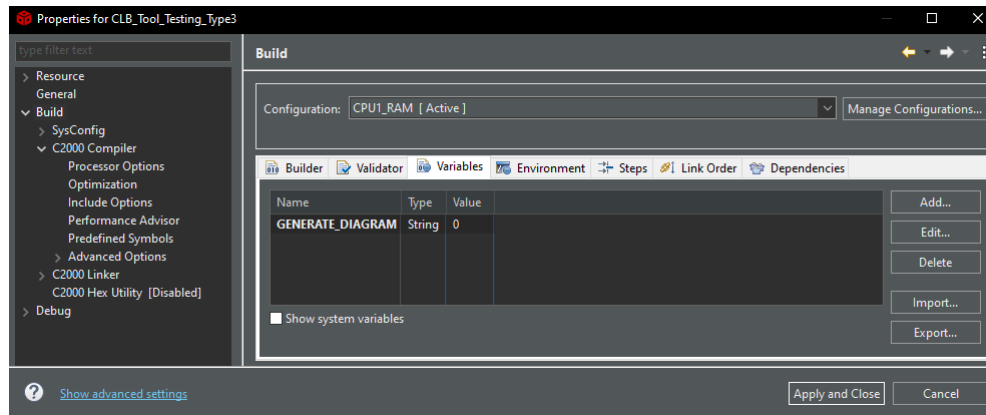


Figure 3-4. Build Variable to Generate Diagram

3.3 Configuring a CLB Tile

To open the SysConfig 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 3-5](#).

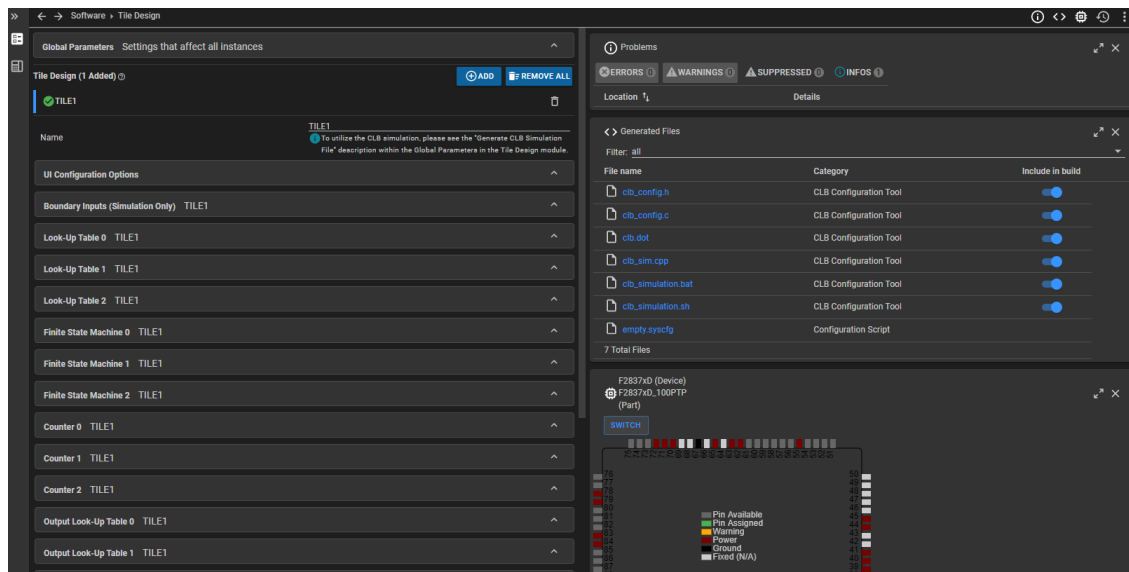


Figure 3-5. CLB Tool SysConfig Screen

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

The configuration of CLB tiles are contained in the .syscfg file. You can change the name of the tile if desired. 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 selecting the sub-module.

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 the CLB module within SysConfig, but for the purposes of simulation the user can specify a square wave signal source, together with a period and duty (both in clock cycles), synchronization, and input pipeline conditions as shown in [Figure 3-6](#). Custom waveform generation for simulation purposes is also supported. For more information on the simulator, see [Section 3.5](#). These options are only for simulation and do not affect the actual CLB configuration or its implementation on the device.

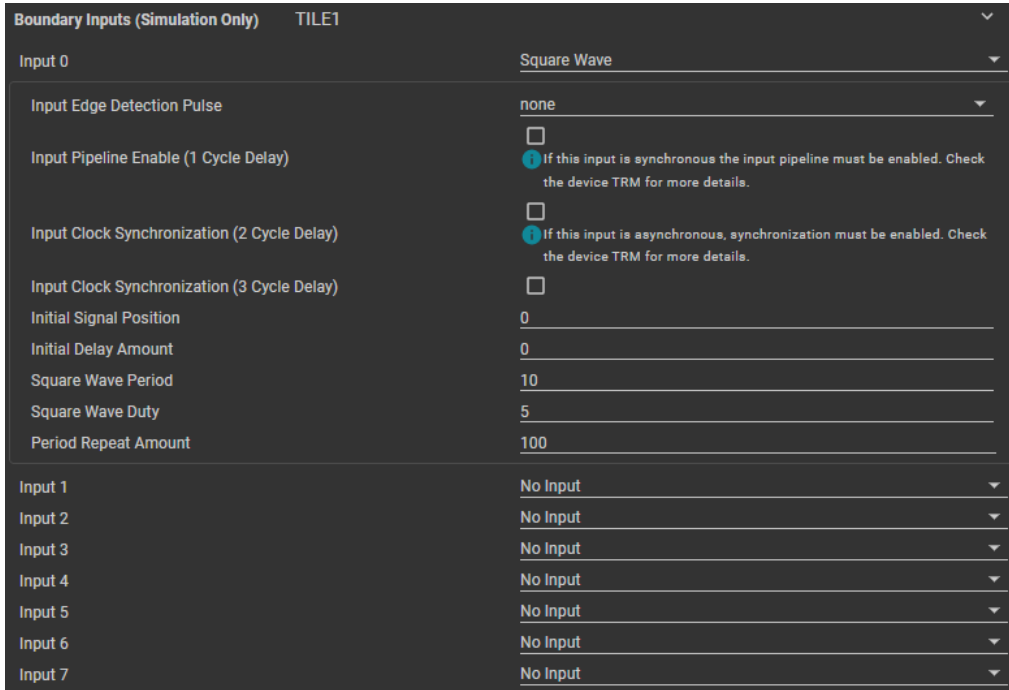


Figure 3-6. Boundary Input Options

The user configures and connects sub-modules in each tile using the check-boxes and drop-down options in the tool. All the sub-modules besides "BOUNDARY" also have a "User Description". This description is a multi-line text box where users can enter comments to help contextualize each part of the CLB.

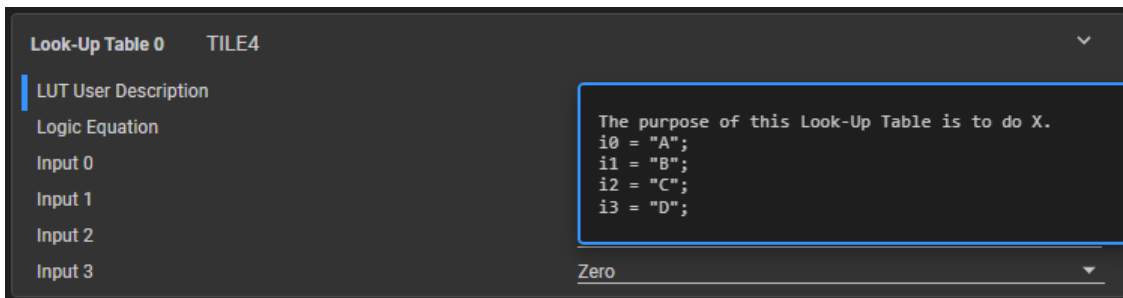


Figure 3-7. User Description Text Box

Context sensitive help appears when the mouse cursor is hovered over each item in the configuration tool. [Figure 3-8](#) shows an example for the "Match Reference 1" field in the "Counter 0" sub-module.



Figure 3-8. Counter Options

Logical equations for the LUTs and FSMs are configured by text entry using C format. [Table 3-1](#) shows the symbols that are allowed in a Boolean equation.

Table 3-1. 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 by an error message below the entry line.

Some unlikely logical combinations generate a warning to the user. [Figure 3-9](#) 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.

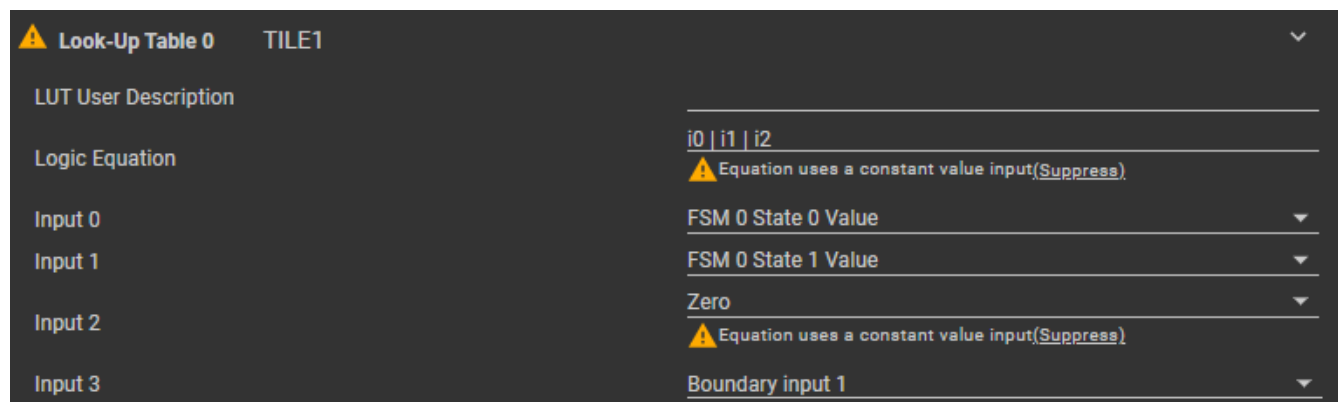


Figure 3-9. Equation Warning

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 because the counter is only 32 bits wide.

The tool automatically generates a 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 and select the filename to open it.

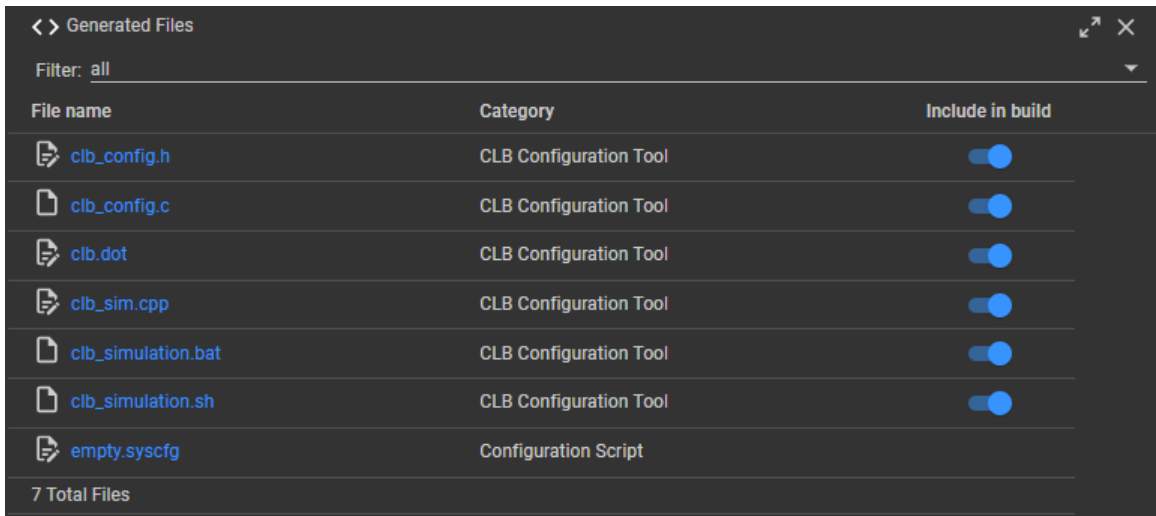


Figure 3-10. CLB Tool Generated Files

CLB register settings are contained in the header file “clb_config.h”, which can be opened by the user by clicking the filename. An example is shown in [Figure 3-11](#). It is important to understand that this file is updated by the tool each time the user changes any CLB Tile Design settings. Therefore, manual changes to the contents of the generated files will be over-written by the tool. If the file is kept open while changing CLB settings, the affected register data changing in the file can be viewed when the 'Unified Diff' option is selected.

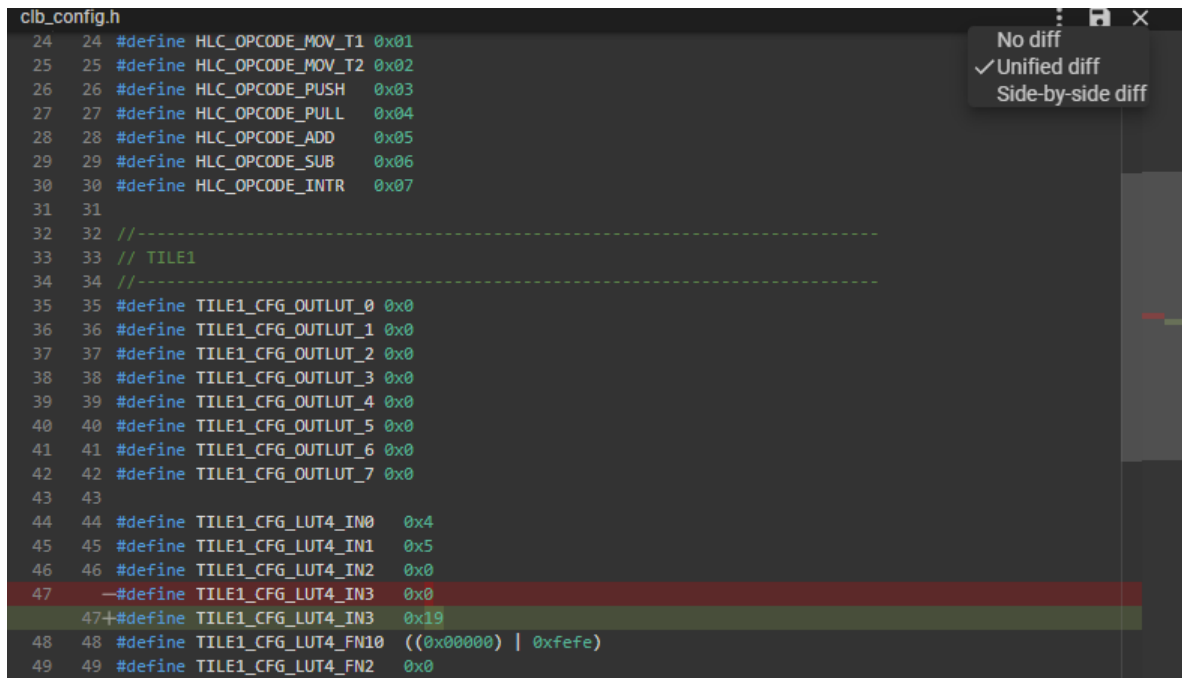


Figure 3-11. “clb.h” Header File Example

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

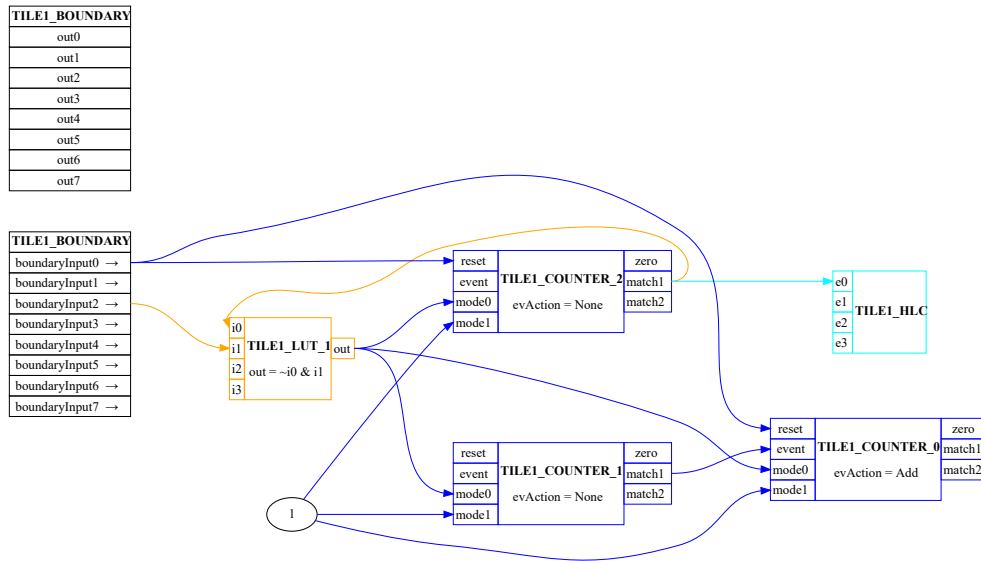


Figure 3-12. CLB Block Diagram

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 the HLC, see the device-specific TRM.

HLC instructions can be entered in the "HLC Program" drop-down in "Other Dependencies". One blank line is always shown until all eight instructions have been used. In Figure 3-13, the user has selected one HLC trigger events and typed in a short program consisting of three instructions.

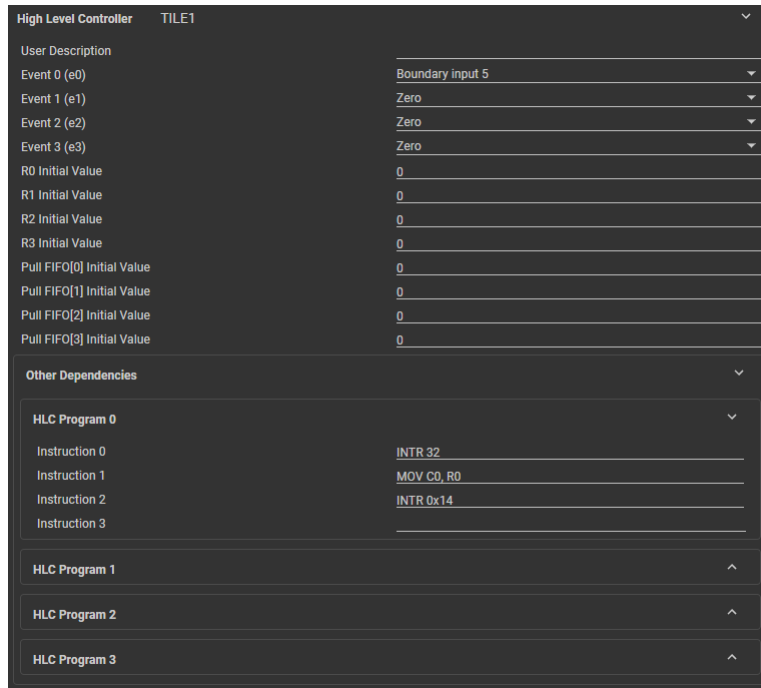


Figure 3-13. HLC Configuration Example

3.4 Creating the CLB Diagram

To create the CLB diagram, follow these steps:

1. Set the GENERATE_DIAGRAM variable in the Build options must be set to 1.

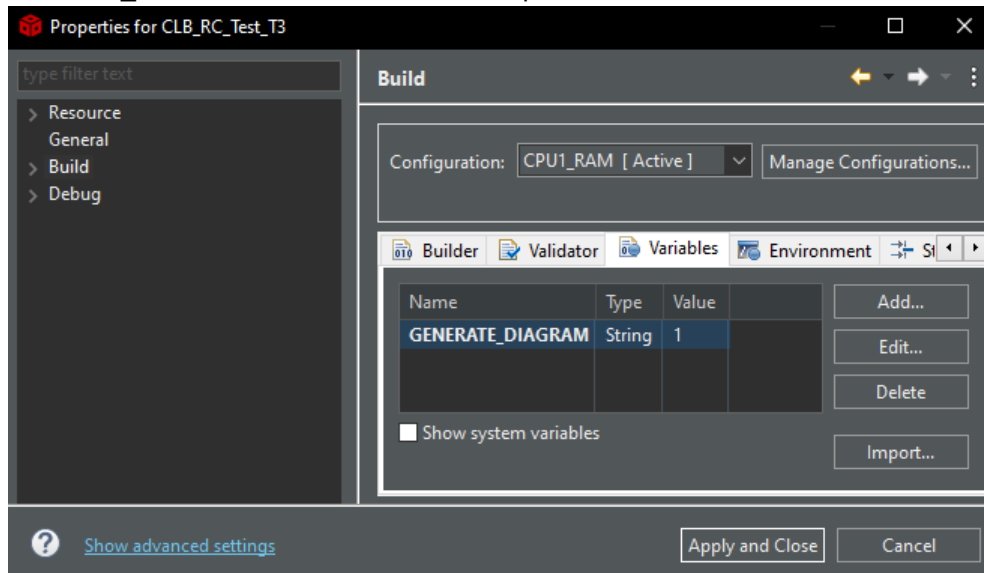


Figure 3-14. GENERATE_DIAGRAM Build Variable

Note

For operating systems such as Linux or Mac, the conditional `if ${GENERATE_DIAGRAM} == 1` need to be removed from the post-build steps to execute properly. Refer to Figure 5-2 for what the post-build steps for a CLB project looks like.

2. Build the project.
3. View the HTML or SVG diagram within the diagrams directory, as shown by Figure 5-5.

3.5 Using the Simulator

3.5.1 The Statics Panel

The top panel in the configuration tool contains “Global Parameters” settings used in simulation. Click the '?' icon to see a short description.

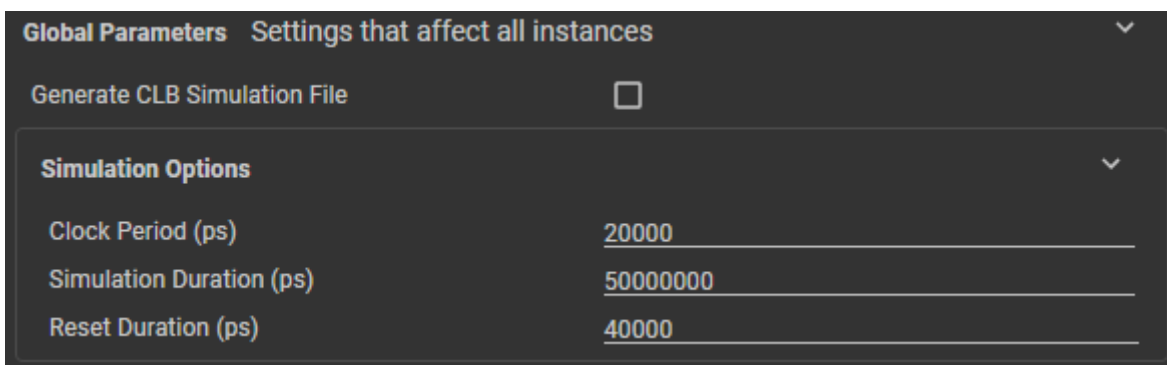


Figure 3-15. Static Options

The "Generate CLB Simulation File" is used to generate the content of the "clb_simulation" file. This file, when executed, will generate the .vcd for all of the CLB Tile Designs. This .vcd file will appear in the directory above where the batch file is executed, within a newly created "simulation" directory.

The “Clock Period (ps)” is the period of the CLB clock in picoseconds used for simulation purposes. The “Simulation Duration (ps)” field allows users to control the duration of the simulation run in picoseconds. The “Reset Duration (ps)” field allows the user to insert a delay in picoseconds to mimic the effect of a device reset.

3.5.2 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 selecting it.

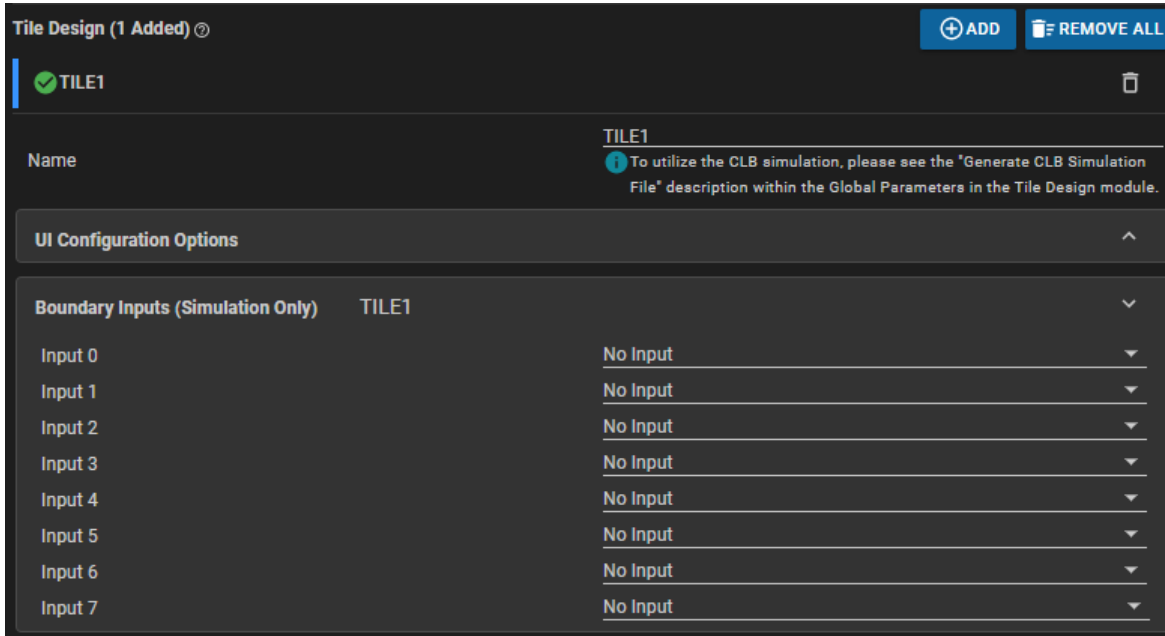


Figure 3-16. Boundary Input 0 to 7

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:

- No Input – Default option, no stimulus is generated.
- Square Wave – Defines a periodic PWM input with configurable initial signal position, initial delay, period, duty, and period repeat amount.

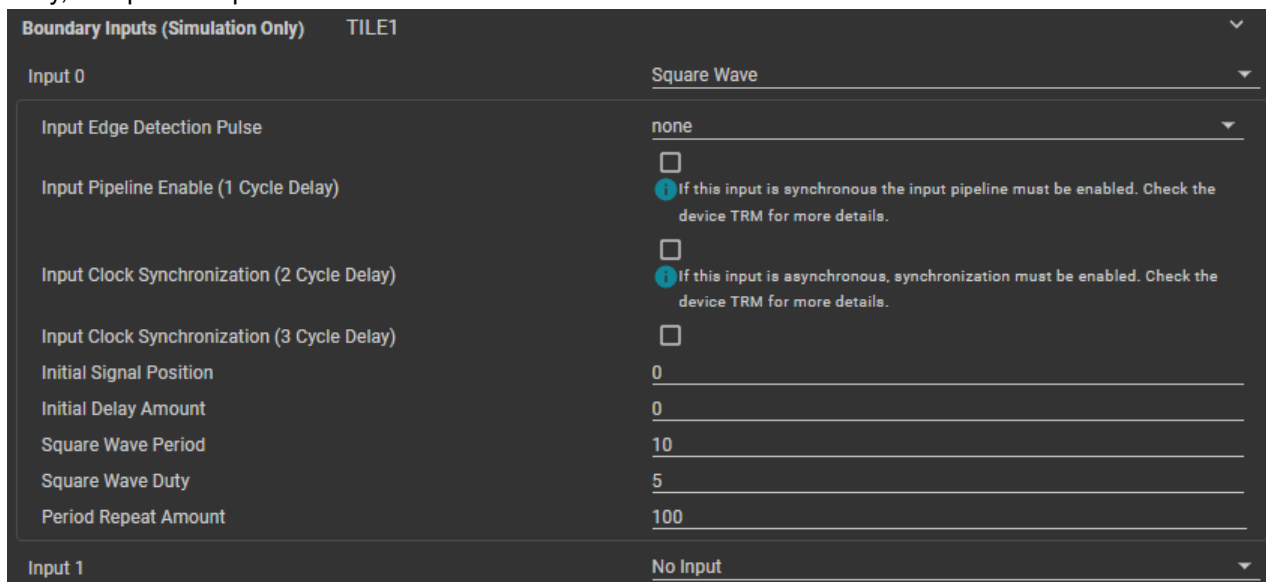


Figure 3-17. Boundary Input Square Wave

The “Input Edge Detection Pulse” 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 3-17](#).

The "Input Pipeline Enable" check-box adds a single cycle delay to the input signal, which is used for synchronized signals which are routed to the CLB as inputs. Note that the pipeline filter is only available on certain CLB types. Check the CLB input mux section in the device-specific TRM for more details.

The “Input Clock Synchronization” check-box forces the input waveform to be synchronized to the CLB clock (the synchronizer creates a 2-3 cycle delay, so there appropriate check-boxes for both timings since the exact delay cannot be predicted). This option is necessary for signals which are coming from asynchronous sources relative to the CLB. For more information, see the CLB input mux section in the device-specific TRM.

- Low (0) or High (1) - Sets the stimulus to a constant low or high, respectively
- Custom Wave Input – Generates a custom stimulus using pseduo-code.

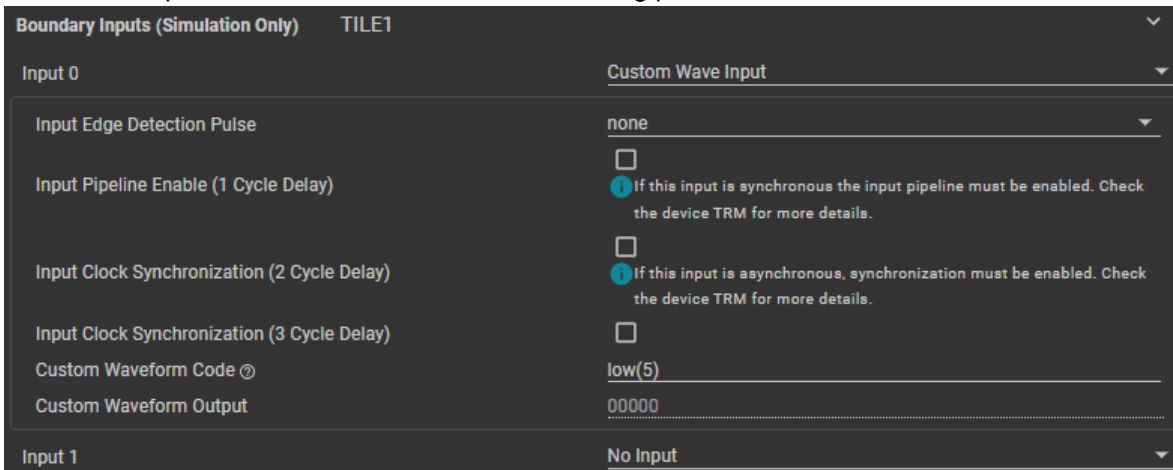


Figure 3-18. Boundary Input Custom

The “Input Edge Detection Pulse”, "Input Pipeline Enable", and “Input Clock Synchronization” work the same as the Square Wave stimulus, with the addition of the custom waveform pseudo-code. The numeric parameter for the 'high', 'low', and 'rpt' instructions can be hexadecimal (0x1A) or decimal (26).

Table 3-2. Custom Waveform Code Instructions

Instruction	Description
#define	Pattern replacer used to define macros
high(N)	Sets waveform high for 'N' CLB cycles
low(N)	Sets waveform low for 'N' CLB cycles
rpt(N)	Starts a repeat block; code encapsulated with rpt(N) and rpt_end will be repeated a total of 'N' times
rpt_end	Denotes the end of a repeat block

- Tile Output - Uses a selected tile output as the input stimulus for the current tile.



Figure 3-19. Boundary Input Tile Output

- "Tile Name" must be the name of a valid tile within the CLB Tool project. The "Input Pipeline Enable" should be enabled since the output of a CLB tile is synchronous. See the *CLB input mux* section in the device-specific TRM for more information.

3.5.3 Running the Simulation

Once the CLB configuration and input stimuli have been defined, the user can compile the project. The full steps for generating the "CLB.vcd" are outlined below.

1. Enable the "Generate CLB Simulation File" check-box in the "Global Parameters" drop-down at the top of the Tile Design module.
2. Build the project to verify there are no errors with the Tile Design settings
3. From the file explorer or command line, execute the "clb_simulation" file. This file is located wherever the generated SysConfig files are saved. For CCS, this is the syscfg directory within the build configuration directory of the project (i.e. "CPU1_RAM/syscfg")
4. Open the "simulation" directory (located one level up from the directory where "clb_simulation" is executed)
5. Double-click on the "CLB.vcd" file

Assuming that the configuration of the waveform viewer has been completed, double-clicking on the "CLB.vcd" file should open the viewer and allow the waveforms to be inspected. Figure 3-20 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.

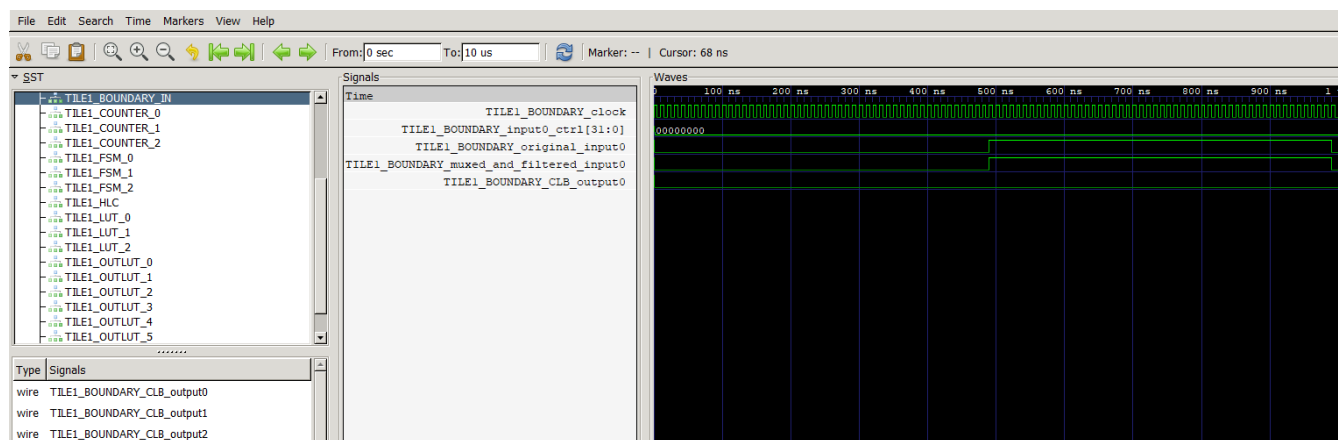


Figure 3-20. CLB Simulation Example

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

3.5.4 Trace Signal Descriptions

This section includes a brief explanation for the different trace signals that are created in the "CLB.vcd" file. Note that some signals or groups are not available depending on the type of CLB available on the device for which the simulation is created.

Note

These conventions are used to abbreviate and shorten the list of trace signals:

1. "TILE#" is the name of the Tile Design instance
2. "N" represents 0 through 7, for multiple instances of the indicated signal
3. "X" represents 0 through 2, for multiple instances of the indicated signal
4. "Y" represents 0 through 3, for multiple instances of the indicated signal

Table 3-3. SystemC Top Level Trace Signals

Trace Signal Name	Description
sc_top_clock	Clock signal of the CLB
sc_top_reset	Reset for the CLB
sc_top_enable	CLB enable signal; used to enable the CLB submodules

Table 3-4. Asynchronous Output Conditioning Block Trace Signals (CLB Type 2)

Trace Signal Name	Description
TILE#_AOC_N_clb_output	Output from the AOC submodule
TILE#_AOC_N_mux_ctrl [15:0]	AOC mux control value; value is in the same format as the CLB_OUTPUT_COND_CTRL_N register
TILE#_AOC_N_release_signal	Release signal; determines when the CLB output is set, cleared, or delayed depending on the release option chosen in TILE#_AOC_N_mux_ctrl (default release signal does not alter CLB output)
TILE#_AOC_N_gate_signal	Gate signal; this signal is logically combined with the CLB output using an AND, OR, or XOR depending on the selection chosen in TILE#_AOC_N_mux_ctrl (default gate signal does not alter output)
TILE#_AOC_N_mux_input_clb_tile_output	CLB output signal corresponding to the same AOC number of the signal; i.e. CLB output 0 is an input option for AOC 0
TILE#_AOC_N_mux_input_boundary_input	CLB boundary input signal corresponding to the same AOC number of the signal; i.e. CLB boundary input 0 is an input option for AOC 0

Table 3-5. Boundary Trace Signals

Trace Signal Name	Description
TILE#_BOUNDARY_CLB_outputN	CLB boundary output signal; this signal is routed out of the CLB peripheral to the rest of the device; this signal has passed through the AOC submodule if the device has this type of CLB (check the CLB Tile section in the device technical reference manual for more details)
TILE#_BOUNDARY_muxed_and_filtered_inputN	Input to the CLB module after passing through any synchronization, input pipeline filters, or edge filters enabled
TILE#_BOUNDARY_inputN_ctrl [31:0]	Control value for enabling synchronization, input pipeline filters, or edge filters
TILE#_BOUNDARY_original_inputN	Input to the CLB module before any modification such as synchronization, pipelining, or edge filtering
TILE#_BOUNDARY_clock	Clock signal of the CLB

Table 3-6. Counter Block Trace Signals

Trace Signal Name	Description
TILE#_COUNTER_X_reconfig_pipeline_en	Enable for reconfigurable pipelining which pipelines the operations of the HLC and counter submodules (this is not the same enable as the pipeline input filter enable)
TILE#_COUNTER_X_counter_equals_match2	This signal is high when the counter value equals the match reference 2 value; if the match reference 2 tap output is enabled for serializer mode, this signal is high when the counter bit position selected is high
TILE#_COUNTER_X_counter_equals_match1	This signal is high when the counter value equals the match reference 1 value; if the match reference 1 tap output is enabled for serializer mode, this signal is high when the counter bit position selected is high

Table 3-6. Counter Block Trace Signals (continued)

Trace Signal Name	Description
TILE#_COUNTER_X_counter_equals_zero	This signal is high when the counter value equals zero
TILE#_COUNTER_X_match2_val [31:0]	Value of the match reference 2 value; can be modified by the HLC
TILE#_COUNTER_X_match1_val [31:0]	Value of the match reference 1 value; can be modified by the HLC
TILE#_COUNTER_X_counter_output [31:0]	Value of the counter itself
TILE#_COUNTER_X_hlc_match2_load_en	Load enable which determines when to load the match reference 2 value from the HLC submodule; this matches the TILE#_HLC_hlc_counterX_match2_load_en signal
TILE#_COUNTER_X_hlc_match1_load_en	Load enable which determines when to load the match reference 1 value from the HLC submodule; this matches the TILE#_HLC_hlc_counterX_match1_load_en signal
TILE#_COUNTER_X_hlc_counter_load_en	Load enable which determines when to load the counter with a value from the HLC submodule; this matches the TILE#_HLC_hlc_counterX_load_en signal
TILE#_COUNTER_X_hlc_counter_load_val [31:0]	Counter value loaded from the HLC submodule; this value is loaded to the match reference 1, match reference 2, or counter value depending on if the appropriate HLC load enable is set (i.e. depending on what instruction is being executed by the HLC). This matches the TILE#_HLC_counter_hlc_load_value signal
TILE#_COUNTER_X_match2_tap [4:0]	Specifies which bit of the counter to tap for match reference 2
TILE#_COUNTER_X_match2_tap_en	Enable which allows the counter to tap a bit specified by TILE#_COUNTER_X_match2_tap; TILE#_COUNTER_X_counter_equals_match2 is high when the appropriate tap bit is set, which effectively brings out a bit position from the counter to the match reference 2 output
TILE#_COUNTER_X_match1_tap [4:0]	Specifies which bit of the counter to tap for match reference 1
TILE#_COUNTER_X_match1_tap_en	Enable which allows the counter to tap a bit specified by TILE#_COUNTER_X_match1_tap; TILE#_COUNTER_X_counter_equals_match1 is high when the appropriate tap bit is set, which effectively brings out a bit position from the counter to the match reference 2 output
TILE#_COUNTER_X_lfsr_en	Enable for the Linear Feedback Shift Register; this allows the counter submodule to compute the CRC on a serial bit stream
TILE#_COUNTER_X_global_serializer_en	Enable for the serializer; when enabled, the counter loads either the next LFSR serial value or the appropriate serial value (see the device technical reference manual for more details)
TILE#_COUNTER_X_mode1	Controls the direction of the counter; counts up when set to 1, counts down when set to 0
TILE#_COUNTER_X_mode0	Controls whether the counter is stopped; enables counting when set to 1
TILE#_COUNTER_X_global_reset	Reset for the CLB
TILE#_COUNTER_X_add_or_shift_dir	When the counter event is enabled and not configured for a load event, this signal is high when the event adds or shifts the counter value left and is set 0 otherwise
TILE#_COUNTER_X_add_or_shift_on_event_en	This signal is set high if the counter is configured to add or shift when an event occurs and is set 0 otherwise
TILE#_COUNTER_X_add_or_shift_mode	This signal is set high if the counter is adding or subtracting and is set 0 otherwise
TILE#_COUNTER_X_global_en	CLB enable signal
TILE#_COUNTER_X_event_load_val [31:0]	When an event occurs and the event action is configured to be 'load', this value is loaded to the counter
TILE#_COUNTER_X_event	This signal is high when an event has occurred or is occurring
TILE#_COUNTER_X_counter_reset	Reset for the counter submodule
TILE#_COUNTER_X_clock	Clock signal of the CLB

Table 3-7. Finite State Machine Block Trace Signals

Trace Signal Name	Description
TILE#_FSM_X_fsm_lut_output	Output for the FSM look-up table output equation
TILE#_FSM_X_fsm_s1_output	Output for the FSM state 1 equation
TILE#_FSM_X_fsm_s0_output	Output for the FSM state 0 equation
TILE#_FSM_X_LUT_output_equation [15:0]	Value representing the FSM look-up table output equation
TILE#_FSM_X_state1_equation_output [15:0]	Value representing the FSM state 1 equation
TILE#_FSM_X_state0_equation_output [15:0]	Value representing the FSM state 0 equation
TILE#_FSM_X_extra_external_input_select1	Selects where the value of external input 3 (e3) is coming from; when high e3 is used, and when low state 1 (s1) is used
TILE#_FSM_X_extra_external_input_select0	Selects where the value of external input 2 (e2) is coming from; when high e2 is used, and when low state 0 (s0) is used
TILE#_FSM_X_extra_external_input1	Extra external input 1 (xe1) to the FSM submodule; the signal chosen as xe1 comes from within the CLB and can be used only in the look-up table output equation
TILE#_FSM_X_extra_external_input0	Extra external input 0 (xe0) to the FSM submodule; the signal chosen as xe0 comes from within the CLB and can be used only in the look-up table output equation
TILE#_FSM_X_external_input1	External input 1 (e1) to the FSM submodule; the signal chosen as e1 comes from within the CLB and can be used in the state 0, state 1, and look-up table output equations
TILE#_FSM_X_external_input0	External input 0 (e0) to the FSM submodule; the signal chosen as e0 comes from within the CLB and can be used in the state 0, state 1, and look-up table output equations
TILE#_FSM_X_global_reset	Reset for the CLB
TILE#_FSM_X_global_en	CLB enable signal
TILE#_FSM_X_clock	Clock signal of the CLB

Table 3-8. High Level Controller Block Trace Signals

Trace Signal Name	Description
TILE#_HLC_spi_export_receive_buffer [15:0]	Representation of the data stored in a SPI RX buffer; this helps validate data exporting through the SPI buffer using the HLC submodule
TILE#_HLC_fifo_overflow_signal	When high, this signal indicates a FIFO overflow has occurred (pushing when the FIFO is full)
TILE#_HLC_fifo_underflow_signal	When high, this signal indicates a FIFO underflow has occurred (pulling when the FIFO is empty)
TILE#_HLC_fifo_write_pointer [1:0]	Current value of the write pointer for the push FIFO (zero-indexed)
TILE#_HLC_fifo_read_pointer [1:0]	Current value of the read pointer for the pull FIFO (zero-indexed)
TILE#_HLC_push_fifo(Y) [31:0]	Represents the current values in the push FIFO
TILE#_HLC_pull_fifo(Y) [31:0]	Represents the current values in the pull FIFO
TILE#_HLC_program_current_instruction [11:0]	Signal which shows the opcode of the current instruction being executed by the HLC
TILE#_HLC_register(Y) [31:0]	Represents the current values of the HLC registers R0 through R3
TILE#_HLC_program_interrupt_number [31:0]	Signal which indicates which interrupt has been triggered by the HLC; the default values is 0xFFFF, otherwise the value represents the interrupt value from the last 6 bits of the interrupt opcode
TILE#_HLC_program_interrupt_flag	This signal is high when the current instruction being executed by the HLC is an interrupt
TILE#_HLC_hlc_counterX_match2_load_en	Enable which goes high when the HLC is loading the corresponding counter match reference 2 value using the MOV_T2 instruction; this matches the appropriate TILE#_COUNTER_X_hlc_match2_load_en signal
TILE#_HLC_hlc_counterX_match1_load_en	Enable which goes high when the HLC is loading the the corresponding counter match reference 1 value using the MOV_T1 instruction; this matches the appropriate TILE#_COUNTER_X_hlc_match1_load_en signal

Table 3-8. High Level Controller Block Trace Signals (continued)

Trace Signal Name	Description
TILE#_HLC_hlc_counterX_load_en	Enable which goes high when the HLC is loading the the corresponding counter value using the MOV instruction; this matches the appropriate TILE#_COUNTER_X_hlc_counter_load_en signal
TILE#_HLC_counter_hlc_load_value [31:0]	Counter value loaded to the match reference 1, match reference 2, or counter value depending on what instruction is being executed by the HLC; this matches the appropriate TILE#_COUNTER_X_hlc_counter_load_val signal
TILE#_HLC_spi_export_enable	This signal indicates that data export via the SPI RX buffer is enabled
TILE#_HLC_reconfig_pipeline_enable	This signal indicates that the reconfigurable pipeline mode is enabled (this affects both the counter and HLC submodules)
TILE#_HLC_alterate_event_clb_async_output(N)	This set of signals represents the asynchronous output of the CLB which comes from the AOC submodules; this is one of the alternate events used for the HLC
TILE#_HLC_alterate_event_clb_output(N)	This set of signals represents the output of the CLB, which comes from the OUTLUT submodules; this is one of the alternate events used for the HLC
TILE#_HLC_hlc_event_trigger(31..0)	A set of signals representing the triggers for the HLC events; to find which event trigger corresponds to which HLC event trigger value, check the device technical reference manual (note that alternate events are not part of this set of signals)
TILE#_HLC_alterate_event_input_selectY	This selection indicates whether the alternate set of events is used for the corresponding HLC event
TILE#_HLC_spi_shift_value [4:0]	Value determining which 16 bits of the 32-bit R0 register are exported to the SPI RX buffer (i.e. a value of 1 selects bits 16:1 of the R0 register)
TILE#_HLC_spi_event_trigger [4:0]	This signal determines which HLC event causes the data export to the SPI RX buffer (note that the event options are from the static switch block as described in the device technical reference manual)
TILE#_HLC_programY_event_source [31:0]	This value indicates which event trigger is being used for the corresponding HLC event; i.e. a value of 1 for TILE#_HLC_program0_event_source (with the alternate event input select set to 0) uses counter 0 match reference 2 for the source of event 0, which triggers HLC program 0
TILE#_HLC_counterX_value [31:0]	Current value of the corresponding counter; this indicates what value is being used in the execution of HLC instructions with C0, C1, or C2 as operands
TILE#_HLC_program_global_load_en	CLB enable signal
TILE#_HLC_program_reset	Reset for the CLB
TILE#_HLC_program_clock	Clock signal of the CLB

Table 3-9. Look-Up Table Block Trace Signals

Trace Signal Name	Description
TILE#_LUT_X_output	Output for the look-up table submodule
TILE#_LUT_X_output_equation [15:0]	Value representing the look-up table logic equation
TILE#_LUT_X_inputY	Input for the look-up table submodule

Table 3-10. Output Look-Up Table Block Trace Signals

Trace Signal Name	Description
TILE#_OUTLUT_N_output	Output for the output look-up table submodule
TILE#_OUTLUT_N_output_equation [7:0]	Value representing the output look-up table logic equation
TILE#_OUTLUT_N_inputX	Input for the look-up table submodule

4 Examples

The CLB tool is supplied with examples in C2000Ware showing how to configure the CLB to implement various use cases. Examples are supplied for all devices with a CLB, although not all examples are applicable to all devices since there are features that are added to newer CLB types that are not available on older versions, such as the Asynchronous Output Conditioning block. Most of the examples below are described for the F28003x device, with the exception of examples 15 which can be found for the F2838x device.

4.1 Foundational Examples

The objective of these examples is to showcase the basic capabilities of the submodules inside each CLB Tile Design. Each example describes a handful of submodules and how to implement simple logic using them in combination. More examples can be found in the links listed in [Section 2.1](#).

For a better understanding of the Tile Design configurations and how the submodules are connected, generate a CLB diagram using the steps laid out in [Section 3.4](#).

4.1.1 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.

4.1.2 Example 3 – PWM Generation

This example configures a CLB tile as an auxiliary PWM generator. The example uses 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 operates at 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 4-1](#) shows in principle what the PWM generator does. Notice how the duty cycle in the next period is changed.

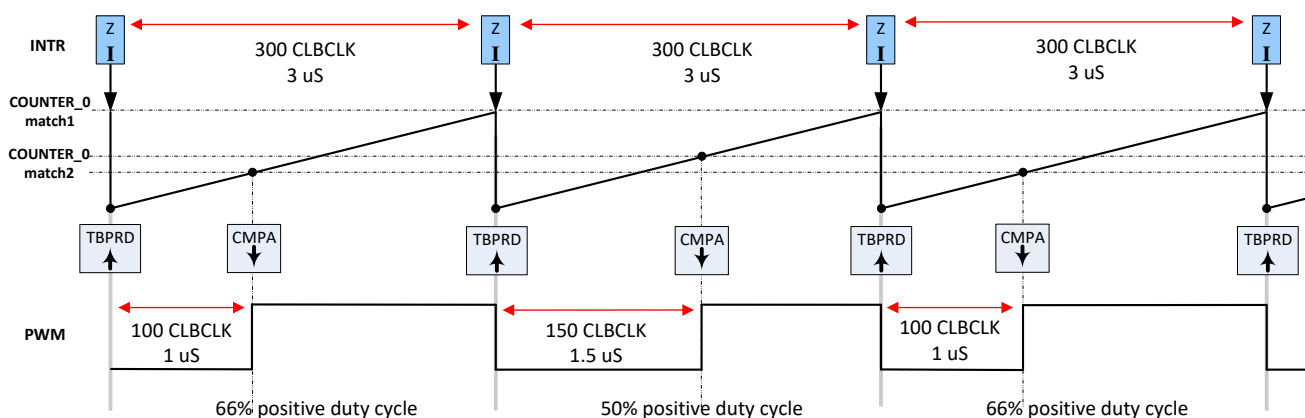


Figure 4-1. Example 3: Generated PWM Waveform

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 μ s). The compare event occurs at either 100 CLBCLK cycles (1 μ s) or 150 CLBCLK cycles (1.5 μ s).

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 3.5.3](#).
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.

4.1.3 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.

4.1.4 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.

4.1.5 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.

4.1.6 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 count the amount of delay in cycles.

4.1.7 Example 16 - Glue Logic

In this example the user is walked through how to migrate custom logic from an FPGA/CPLD to C2000™ microcontrollers.

4.1.8 Exampe 18 - AOC

In this example the Asynchronous Output Conditioning block is used to asynchronously AND gate the input signals to the CLB. This module is only available for CLB types 2 and up.

4.1.9 Example 19 - AOC Release Control

In this example the Asynchronous Output Conditioning block is used to asynchronously set/release the input signals to the CLB. This module is only available for CLB types 2 and up.

4.1.10 Example 20 - CLB XBARs

In this example the CLB INPUTXBAR and CLB OUTPUTXBAR are used to take input signals from GPIOs into the CLB TILES and take output signal from the TILE to GPIOs. The availability of these XBARs are device dependent.

4.2 Getting Started Examples

These examples provide a bit more complexity to create more useful examples and implementations of the CLB peripheral. More examples can be found in the links listed in [Section 2.1](#).

For a better understanding of the Tile Design configurations and how the submodules are connected, generate a CLB diagram using the steps laid out in [Section 3.4](#).

4.2.1 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 4-2](#).

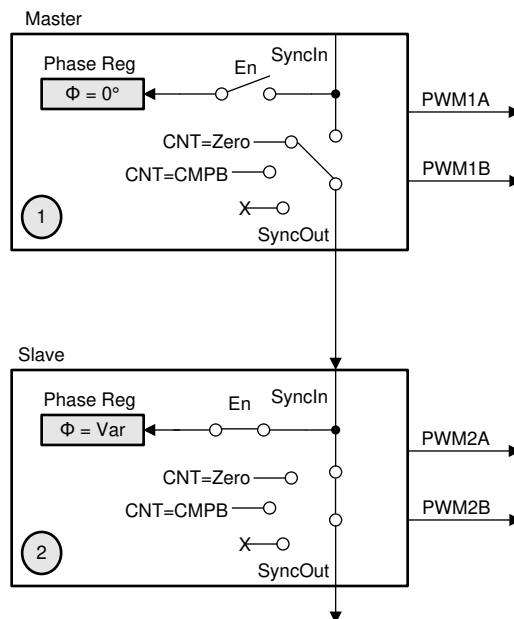


Figure 4-2. Example 1: EPWM Synchronization

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 4-3](#).

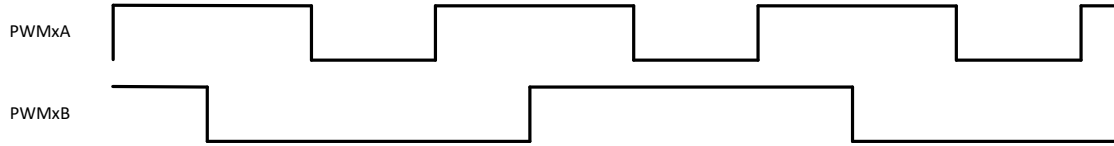


Figure 4-3. Example 1: PWM Test Pattern

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. For instance, if the logic is in active low mode and both PWM signals are low, the output for **both** PWMs will be forced high. Refer to [Table 4-2](#) for more details. The mode is selected using a 2-bit field as shown in [Table 4-1](#).

Table 4-1. 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 4-4](#). Output signals have “_m” appended to the name to indicate they may have been modified by the CLB.

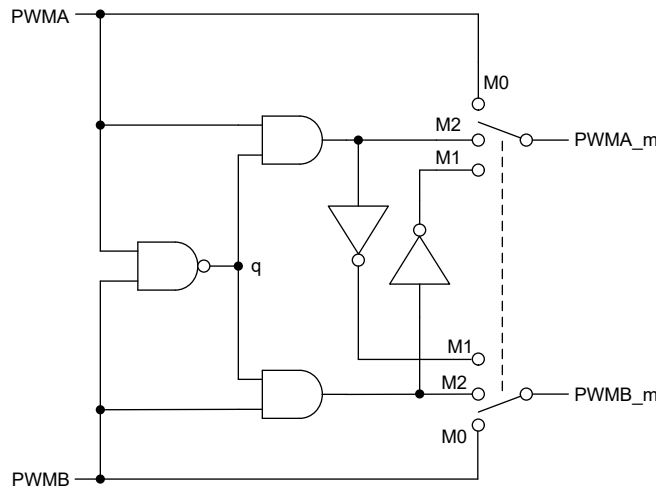


Figure 4-4. Example 1: Logic Diagram

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 4-5](#) (modes 1 and 2). This image highlights in green the areas where the output is forced high or low by the CLB logic.

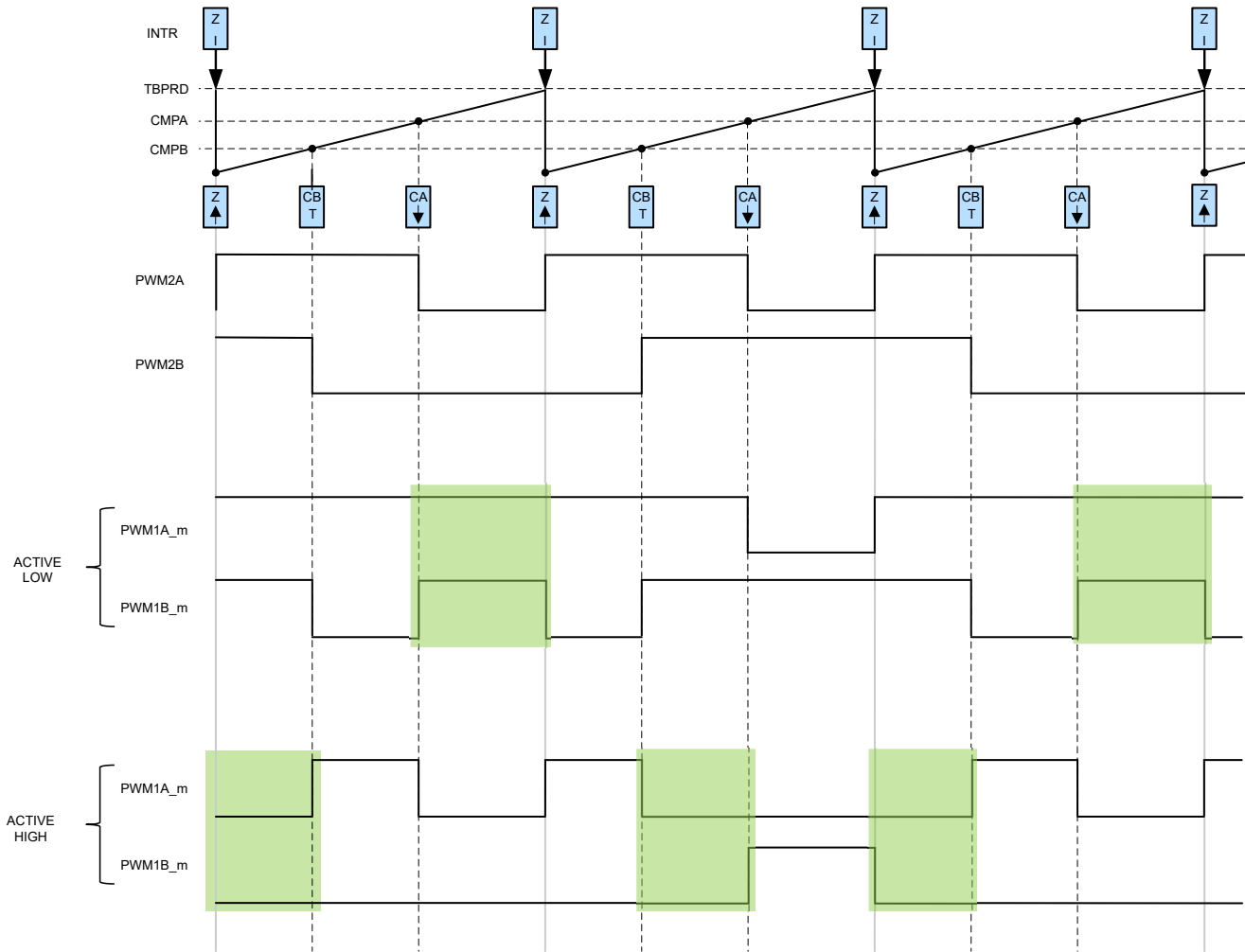


Figure 4-5. Example 1: Generated PWM

Table 4-2. PWM Output

Operating Mode	Original PWM A Output	Original PWM B Output	PWM A Output	PWM B Output
Normal	0	0	0	0
	0	1	0	1
	1	0	1	0
	1	1	1	1
Active Low	0	0	1	1
	0	1	0	1
	1	0	1	0
	1	1	1	1
Active High	0	0	0	0
	0	1	0	1
	1	0	1	0
	1	1	0	0

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 4-6](#) conceptually shows the connections to and from the CLB tile.

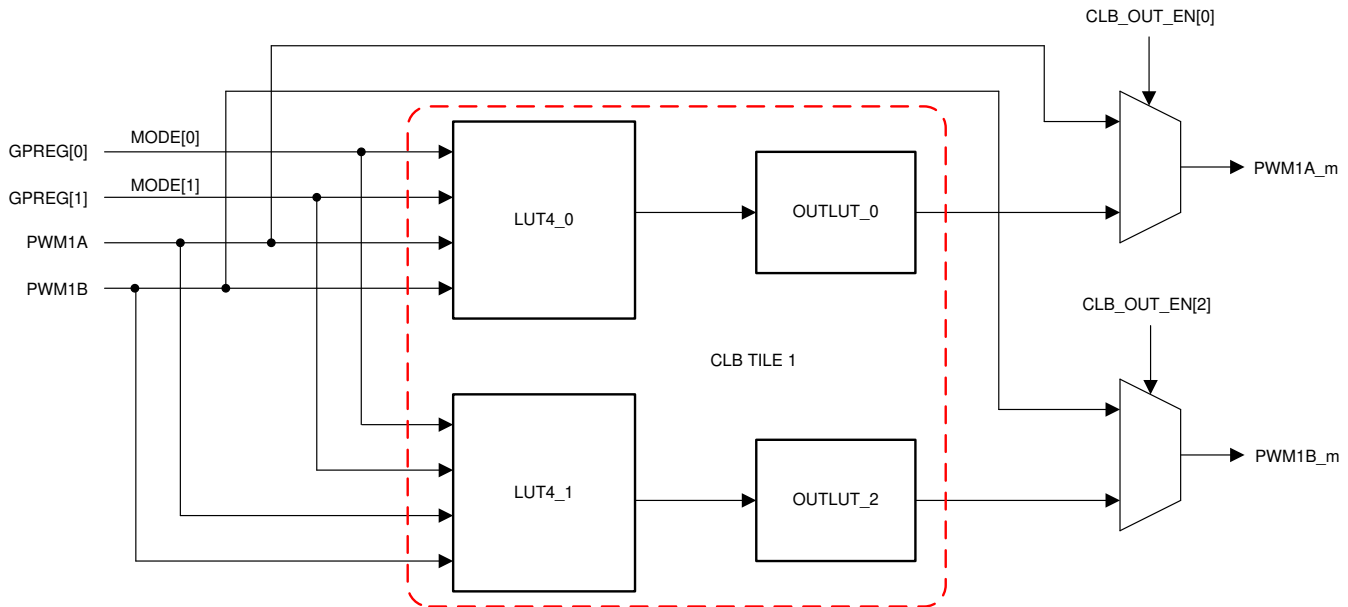


Figure 4-6. Example 1: CLB Configuration

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.
 - a. The Launchpad pins to watch the PWMs for the F28379D and Experimenter kit pins for F28388D are listed, but the Launchpad pins for other devices are not listed. Refer to the device datasheet for more information on available pins and their configurations
8. Open a CCS Expressions window.
9. [Optional] – for instructions on how to run a simulation of the CLB, see [Section 3.5.3](#).

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.

4.2.2 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 4-7 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 4-7 shows in principle what the filter does.

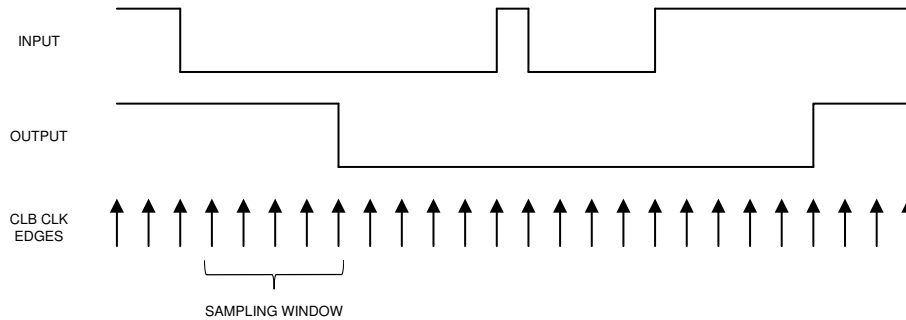


Figure 4-7. Example 2: GPIO Glitch Example

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 4-8.

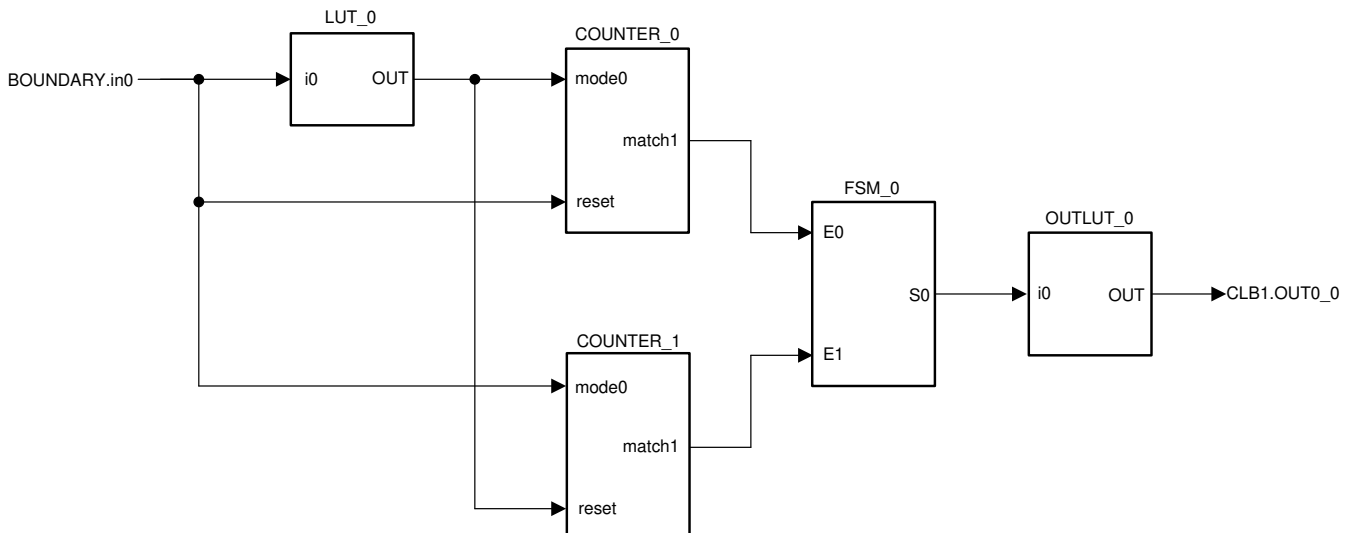


Figure 4-8. Example 2: CLB Configuration

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\2837x\examples\cpu1\clb\ccs, or
 - b. [C2000Ware]\driverlib\28004x\examples\clb\ccs, or
 - c. [C2000Ware]\driverlib\2838x\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 3.5.3](#).

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 4-9](#) shows the expected waveforms at the output pins for glitch widths below and above the sample window setting of 8.

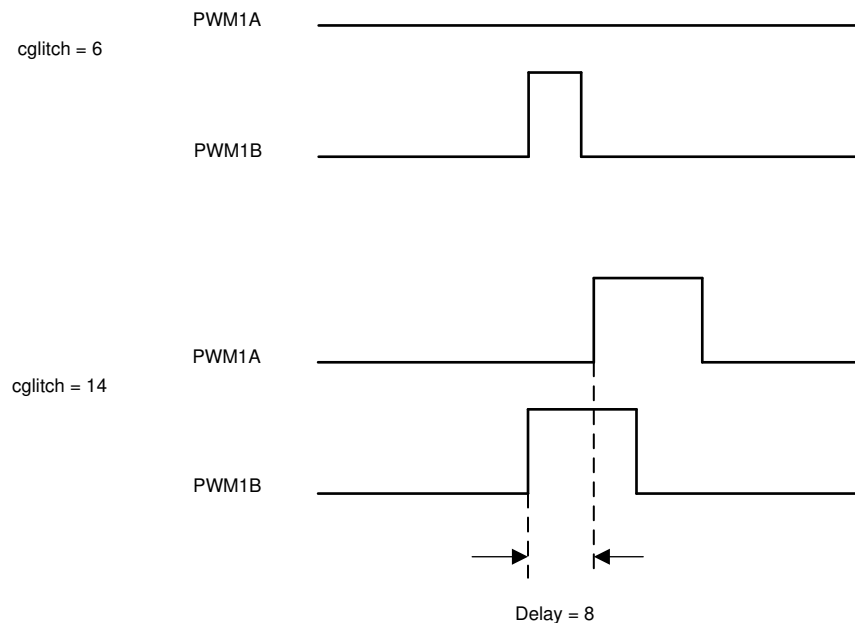


Figure 4-9. Example 2: GPIO Glitch Width

4.2.3 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\f2837x\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 4-3. 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.

4.2.4 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 4-10 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.

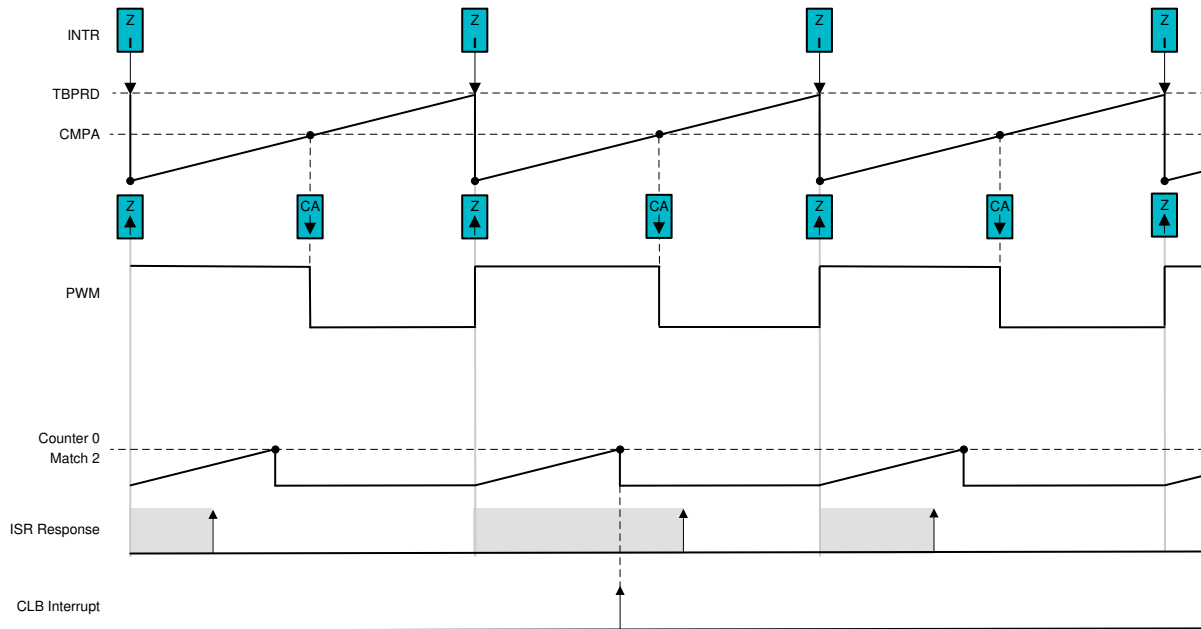


Figure 4-10. Example 5: Event Window Configuration

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 do 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.

4.2.5 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 4-11 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.

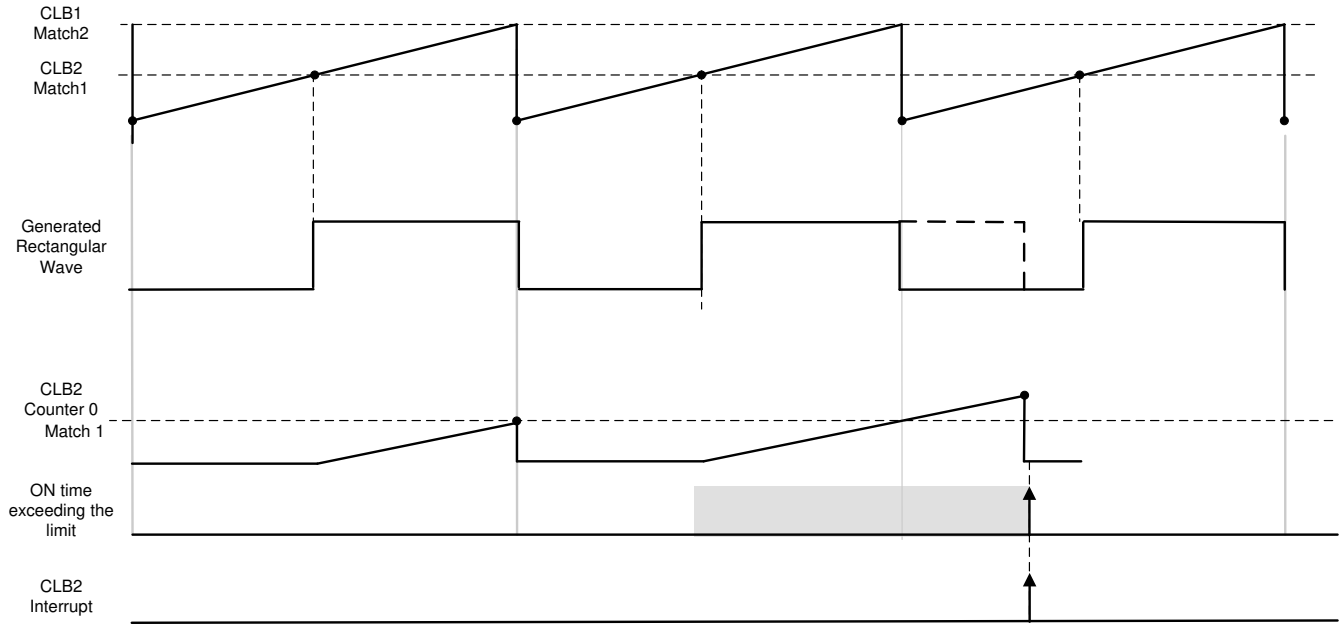


Figure 4-11. Example 6: Duty Exceeding Pre-Set Value

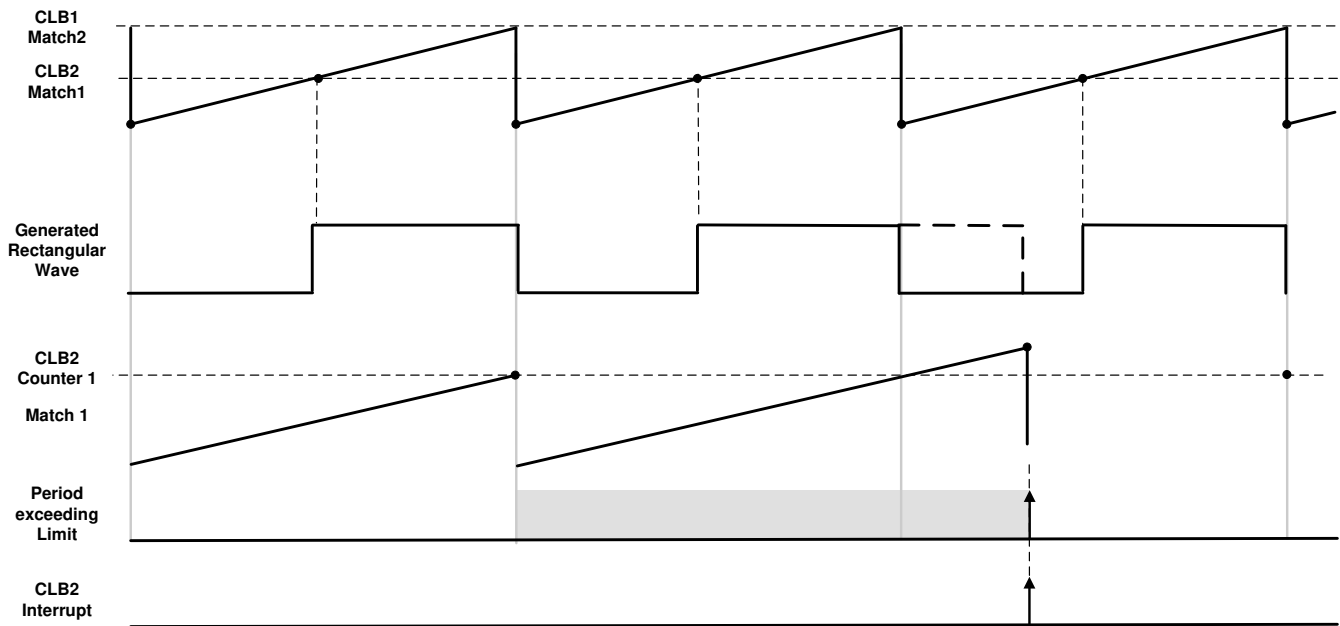


Figure 4-12. Example 6: Period Exceeding Pre-Set Value

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\2837x\examples\cpu1\clb\ccs, or
 - b. [C2000Ware]\driverlib\28004x\examples\clb\ccs, or
 - c. [C2000Ware]\driverlib\2838x\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.

4.2.6 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.

4.2.7 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.

4.2.8 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.

4.2.9 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.

4.2.10 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.

4.2.11 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.

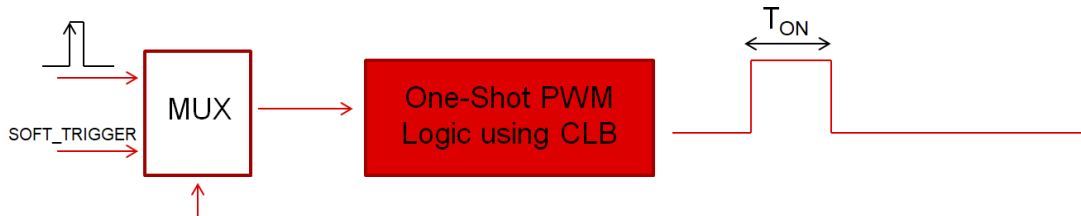


Figure 4-13. Example 17: One-Shot PWM Output

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 μ s - 0.8 μ s 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 4-14.

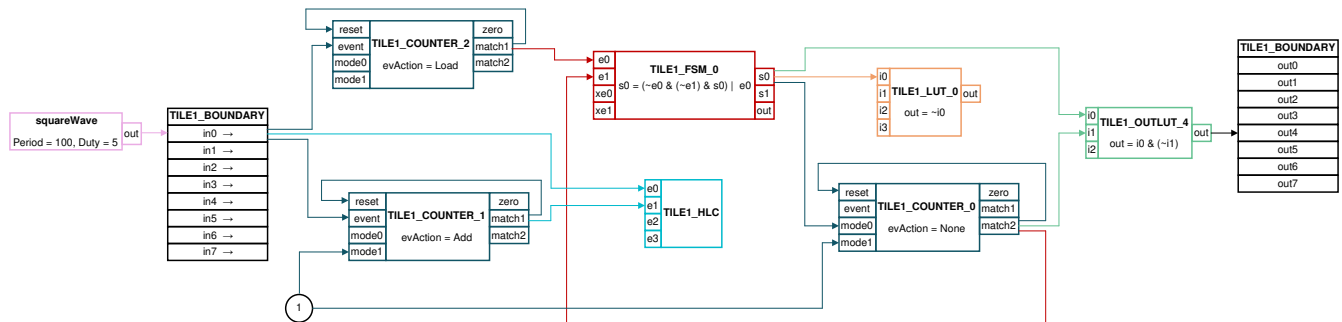


Figure 4-14. Example 17: Overall CLB configuration

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\28004x\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 3.5.3](#).
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.

4.2.12 Example 21 - Clock Prescaler and NMI

In this example the clock prescaler of the CLB module is used to divide down the CLB clock and use it as an input to the TILE logic. Also the HLC module is used to generate NMI interrupts. This module is only available for CLB types 2 and up.

4.2.13 Example 22 - Serializer

In this example the CLB COUNTER is used in serializer mode to act as a shift register. This module is only available for CLB types 2 and up.

4.2.14 Example 23 - LFSR

In this example the CLB COUNTER module is used in Linear Feedback Shift Register (LFSR) mode. This module is only available for CLB types 2 and up.

4.2.15 Example 24 - Lock Output Mask

In this example the lock output mask feature of the CLB is used to lock the selected output signal override settings. This module is only available for CLB types 3 and up.

4.2.16 Example 25 - Input Pipeline Mode

In this example the CLB Input Pipeline mode is enable to delay the input signal by a clock cycle. This module is only available for CLB types 3 and up.

4.2.17 Example 26 - Clocking Pipeline Mode

In this example the CLB pipeline mode is enable and affects the behavior of the CLB COUNTERs and HLC. This module is only available for CLB types 3 and up.

4.3 Expert Examples

The objective of these examples is to provide more in-depth use cases for the CLB. Each example describes how to configure the submodules to achieve a specific implementation. More examples can be found in the links listed in [Section 2.1](#).

For a better understanding of the Tile Design configurations and how the submodules are connected, generate a CLB diagram using the steps laid out in [Section 3.4](#).

4.3.1 Example 27 - SPI Data Export

In this example the high speed data export feature of the CLB is used and one of the HLC registers is exported out of the CLB module using the SPI RX buffer. This module is only available for CLB types 3 and up.

4.3.2 Example 28 - SPI Data Export DMA

In this example the high speed data export feature of the CLB is used and one of the HLC registers is exported out of the CLB module using the SPI RX buffer. The data received in the SPI RX buffer is transferred to memory using DMA. This module is only available for CLB types 3 and up.

4.3.3 Example 29 - Timestamp

This example displays how to timestamp interrupts generated by the CLB. An interrupt is generated when ePWM1 is tripped. ePWM1 is configured to be interrupted by TZ1 and TZ2, both one shot trip sources.

4.3.4 Example 30 - Cyclic Redundancy Check

This example demonstrates how a CLB tile can be configured to act as a cyclic redundancy check (CRC). It emulates the CRC procedure to check the error detection due to unintended changes to raw data. This module is only available for CLB types 2 and up.

NOTE: Bit count for CRC implementation is limited to 31 bits or less.

4.3.5 CLB TDM Serial Port

FILE: clb_ex31_tdm_serial_port.c

For the detailed description of this example, please refer to: [How to Implement Custom Serial Interfaces Using the Configurable Logic Block \(CLB\) Application Note \(SPRAD62\)](#).

In this example a single CLB tile is used to input a TDM stream and generate a TDM output stream. The CLB generates a CPU interrupt when four 32-bit words are received. The CPU can load four 32-bit values to the CLB FIFO for transmission. The CLB and CPU are configured to run at their maximum speed.

This example is only available on C2000 MCU devices with CLB types 2 and up.

External Connections

TDM Input Signal GPIO pin FSYNC_IN GPIO00 BCLK_IN GPIO01 DATA1_IN GPIO02

TDM Output Signal GPIO pin FSYNC_OUT GPIO04 BCLK_OUT GPIO05 DATA1_OUT GPIO06

4.3.6 CLB LED Driver

FILE: clb_ex32_led_driver.c

For the detailed description of this example, please refer to: [How to Implement Custom Serial Interfaces Using the Configurable Logic Block \(CLB\) Application Note \(SPRAD62\)](#).

In this example two CLB tiles are used to communicate with an LP5891-Q1 LED driver. One CCSI bus is used to transmit data using the CCSI bus protocol, while a second tile is used to receive data from the CCSI bus. The C28x CPU communicates with the CLB logic through a hardware-abstraction layer (HAL). This example also utilizes a PWM to generate the required CCSI clocks, and a timer to generate periodic sync events to the LED driver.

This example is only available on C2000 MCU devices with CLB types 2 and up.

External Connections

CCSI Input Signal GPIO pin LED Driver CLB_SIN_1 GPIO17 SOUT

CCSI Output Signal GPIO pin LED Driver CLB_SOUT_1 GPIO08 SIN CLB_SCLK_1 GPIO01 SCLK

4.3.7 FPGA/CPLD to C2000 Examples

Refer to the [How to Migrate Custom Logic From an FPGA/CPLD to C2000 Microcontrollers Application Report](#) document for additional examples on how to translate FPGA- or CPLD-based custom logic to the CLB peripheral on C2000 devices.

5 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_INSTALL>\driverlib\F2837xD\examples\cpu1\clb\empty.syscfg) from the CLB examples folder to the project by copying the file into the project directory.
2. CCS asks the user whether or not to enable SysConfig. Accept and select "Yes".

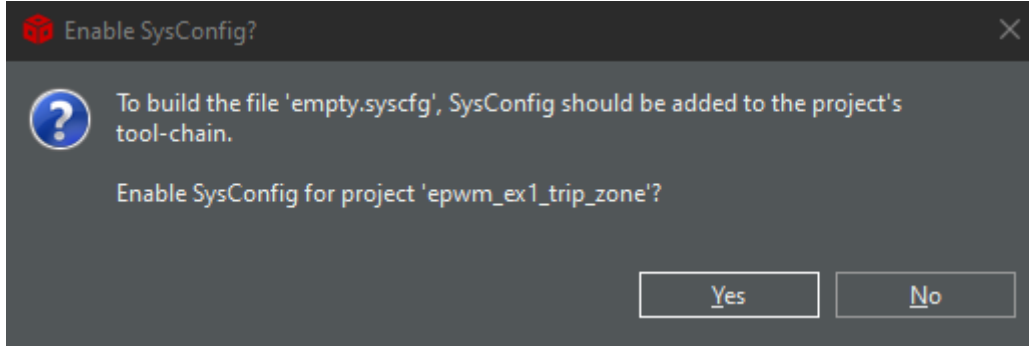


Figure 5-1. Enable SysConfig

3. Open the "Project Properties" and open the Resources → Linked Resources. Add the following Variable Paths:
 - a. C2000WARE_ROOT
[PATH_TO_C2000WARE]
 - b. CLB_SYSCFG_ROOT [PATH_TO_CLB_TOOL]
 - i. See [step 6](#) for more details
4. In the Project Properties window, select Build → Steps.
5. Add the following line to the Post-build steps as seen in [Figure 5-2](#).
 - a. `${NODE_TOOL} "${C2000WARE_ROOT}/dot_file_libraries/clbDotUtility.js" "${C2000WARE_ROOT}" "${BuildDirectory}/syscfg" "${BuildDirectory}/syscfg/clb.dot"`



Figure 5-2. Post-Build Steps

6. Next, open Resources → Linked Resources to verify the correct path for CLB_SYSCFG_ROOT is used. Then open Build → SysConfig → Basic Options and add the proper path to the Root system config meta data list; do this only in the case where the project does not contain SysConfig:
 - a. Make sure that the Linked Resources has the correct path for CLB_SYSCFG_ROOT

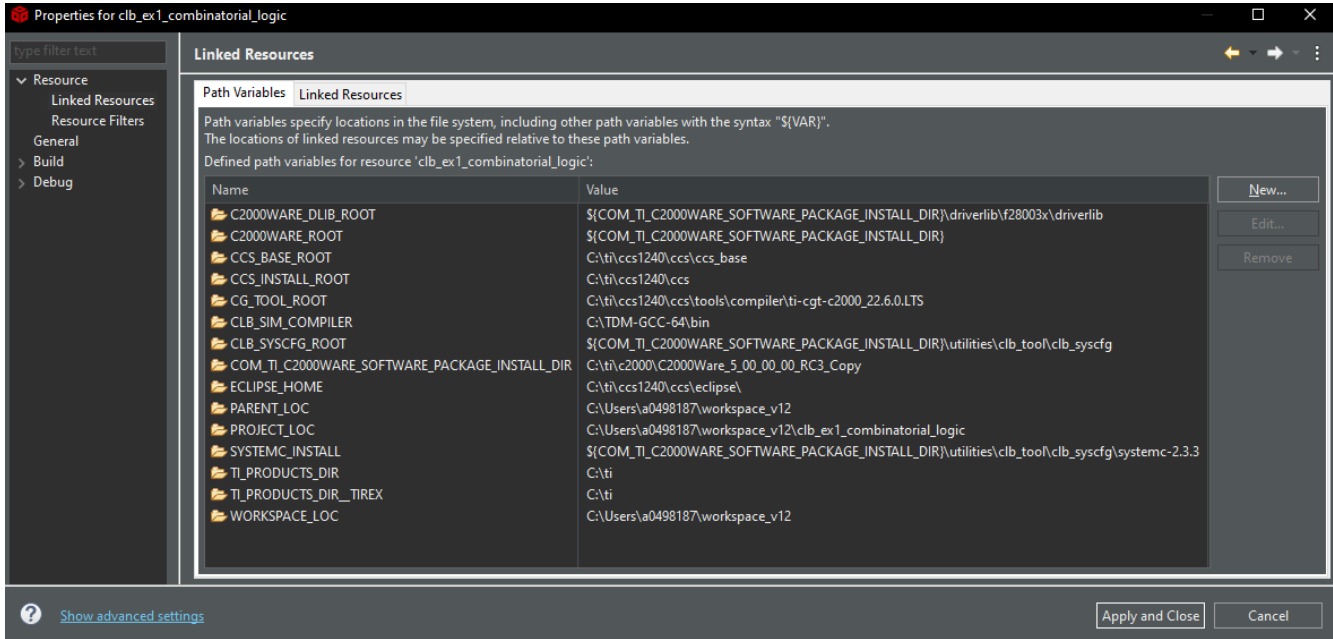


Figure 5-3. Linked Resources for Enabling CLB Tool

- b. `$(CLB_SYSCFG_ROOT)/.metadata/product.json`

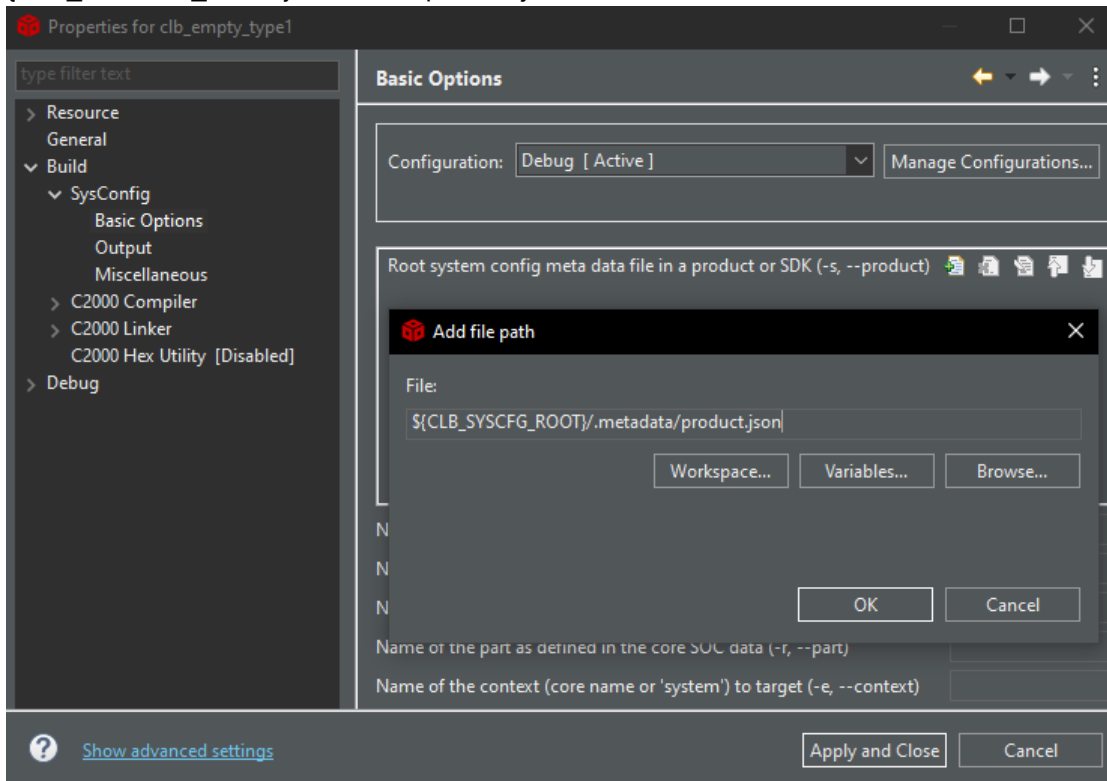


Figure 5-4. SysConfig SDK Path

7. Finally click Apply and Close.

- After building the project, the content generated by the CLB Tool will be present in the build configuration directory. [Figure 5-5](#) shows an example of this after adding CLB support to the epwm_ex1_trip_zone driverlib example.

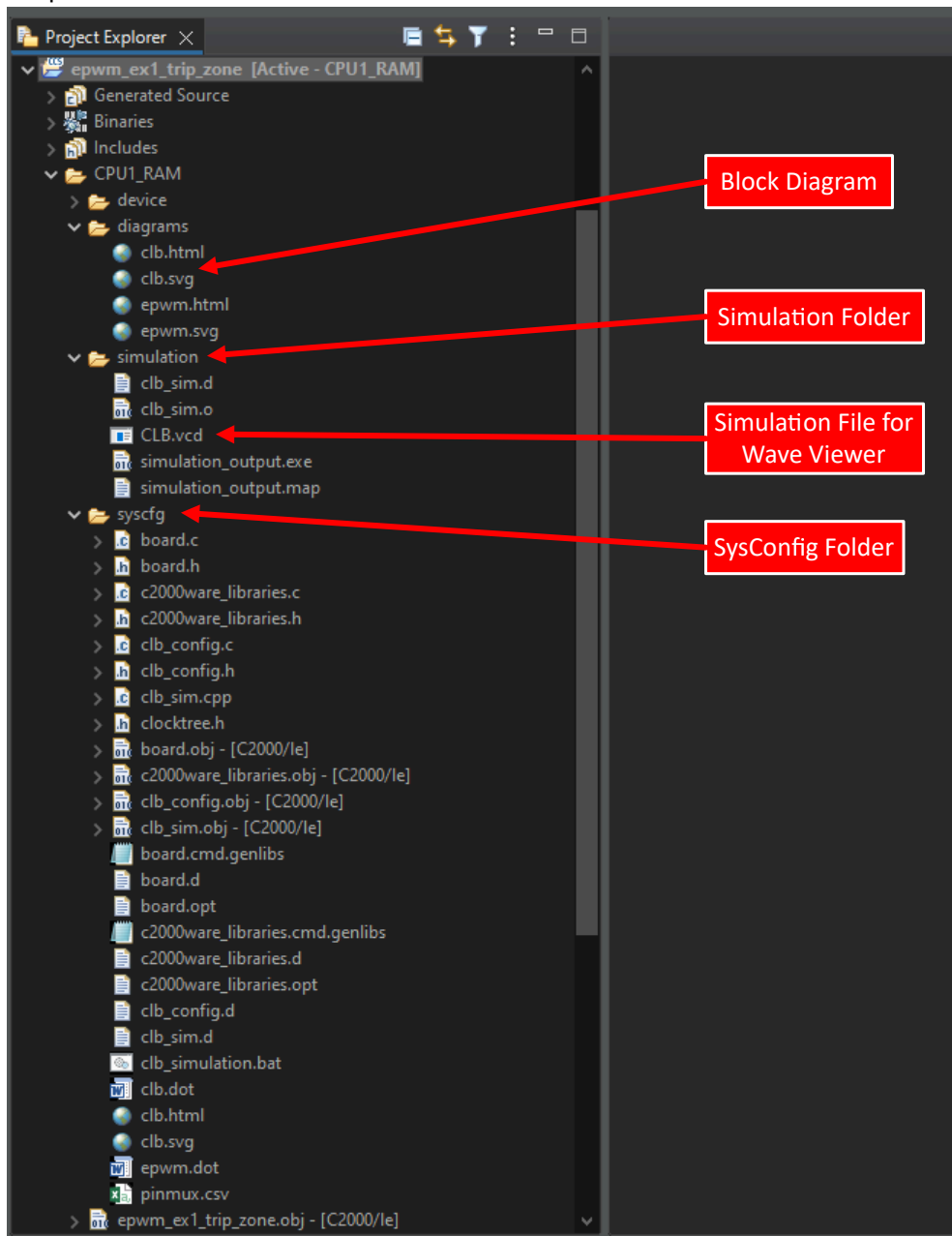


Figure 5-5. epwm_ex1_trip_zone With CLB Tool Support

7 Revision History

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

Changes from Revision A (April 2020) to Revision B (July 2023)	Page
• Updated the numbering format for tables, figures and cross-references throughout the document.....	3
• Changes were made through the entire User's Guide due to major updates to the CLB Tool.....	3

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), 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, regulatory 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](#) or other applicable terms available either on [ti.com](https://www.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.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2023, Texas Instruments Incorporated