Application Report

How to Migrate Custom Logic From an FPGA/CPLD to C2000™ Microcontrollers

Peter Galicki

ABSTRACT

The Configurable Logic Block (CLB) reduces total system cost and enhances functionality by absorbing external logic into C2000™ microcontrollers. CLB uses function calls and a GUI-based programming tool called SysConfig to absorb external logic into the microcontroller without having to learn Hardware Description Language like VHDL or Verilog. This Report shows programmers, hardware engineers and system designers how to translate FPGA- or CPLD-based custom logic (that has been originally defined in HDL) to a CLB form that can be programmed into C2000 MCUs.

Table of Contents

List of Figures

Trademarks

C2000™ are trademarks of Texas Instruments.
All other trademarks are the property of their respective owners.

1 Introduction

Just like a CPLD or FPGA, CLB is composed of programmable logic primitives that can be configured in many ways to implement custom blocks of logic. Instead of using VHDL or Verilog to configure these logic primitives, CLB is programmed with a GUI-based SysConfig tool and function calls. Since the configuration method is different, the CLB is technically not a CPLD or FPGA, but it can be used to achieve identical results.

The CLB holds certain advantages over external CPLDs and FPGAs. Because it resides inside the C2000 device, CLB has direct access to key CPU and peripheral signals without having to account for pin delays. Additionally, a simple built-in HLC processor facilitates data transfer between CLB and C2000 memory allowing the CLB to work hand-in-hand with software running on the C2000 processor(s).

With CLB it is now possible to absorb external custom logic into the C2000 device, create custom peripherals inside the C2000, and modify existing C2000 control peripherals at input stage, output stage or at many pre-defined sites inside the peripheral. The following sections contain step-by-step instructions how to implement the most common use cases, plus low-level functional schematics of CLB building blocks to aid the process of mapping logic from VHDL or Verilog into CLB. Many powerful and flexible CLB features provide you with substantial benefits including reduction in system component count, added flexibility to differentiate products and ability to update custom logic in the field via software after parts have shipped.

This applications report is based on the base-level CLB architecture that is common to several C2000 devices including the F28004x, F2807x, F2837x, and F2838x series. Future versions will include additional features and expanded functionality.

2 CLB Description From the Hardware Perspective

2.1 How Does the CLB Work

CLB is a collection of programmable logic primitives, input and output muxes, that are configurable by the CPU or CLA via Configuration Registers. The CLB block has selectable input and output signals that reach inside selected Control Peripherals (Enhanced Pulse Width Modulator (EPWM), Quadrature Encoder Pulse (QEP) and Enhanced Capture (ECAP)). Based on how the logic primitives are configured, custom logic is applied to selected input signals from inside a Control Peripheral resulting in output signal that is then injected back into a selected spot inside the Control Peripheral.

Note, that the original input and output signals that enter and exit Control Peripherals are not affected by the CLB, only the internal signals can be modified (see Figure 2-1). This method is used to modify the operation of a selected Control Peripheral or to outright replace it with a totally new custom peripheral (when the inputs and outputs are tapped at the peripheral boundary). Note, that even when the internals of a Control Peripherals have been completely replaced by the CLB logic, the inputs and outputs at the peripheral boundary are unchanged (and so are the associated general-purpose input/output (GPIO) assignments).

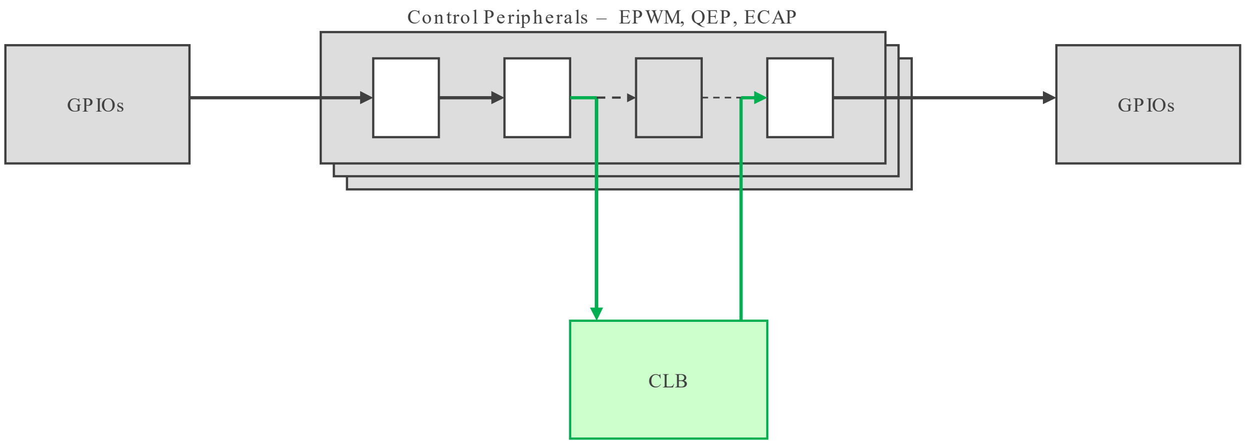


Figure 2-1. CLB Operating Inside Control Peripherals

CLB can also operate without affecting the functionality of the Control Peripherals by driving output signals directly into the Output XBAR (Crossbar) where they can be directed to exit the device through selected GPIO pins. The inputs to the CLB do not have to only come from Control Peripherals, they can also originate from other peripherals, CPU signals, CPU register bits and GPIOs. In cases when GPIOs are the only inputs to the CLB, and the GPIOs are the only outputs from the CLB, the CLB becomes a vehicle for implementing external glue logic that originally may have resided inside an external CPLD of FPGA (see Figure 2-1).

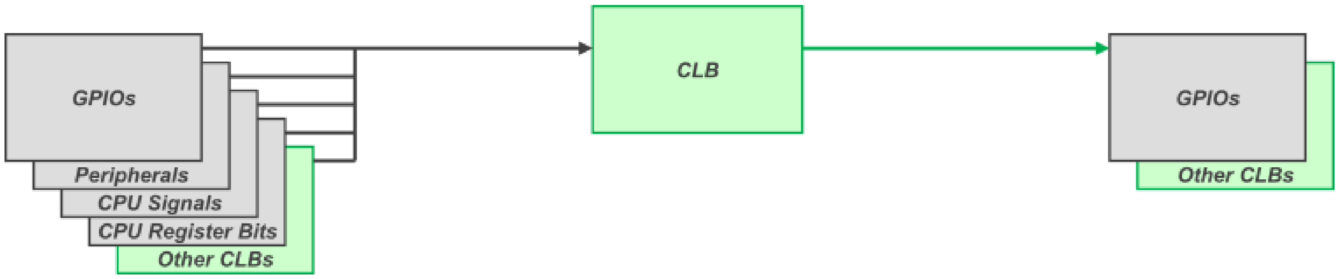


Figure 2-2. CLB Operating Outside Control Peripherals

All CLB use cases described in this document are essentially some combination of these two operating modes (see Figure 2-3). The ability to mix-and-match a variety of inputs and outputs makes the CLB a very flexible and powerful addition to C2000 family. The sections below provide more system level information explaining how the CLB fits within the rest of the C2000 chip. This is followed by a detailed examination of CLB building blocks. Some of the most common examples of CLB usage employing various types of inputs and outputs are shown next. Finally, the first of these examples is described in more detail, showing how two PWM generators and associated glue logic can be absorbed into the C2000 from an external FPGA source, and how both implementations have identical results.

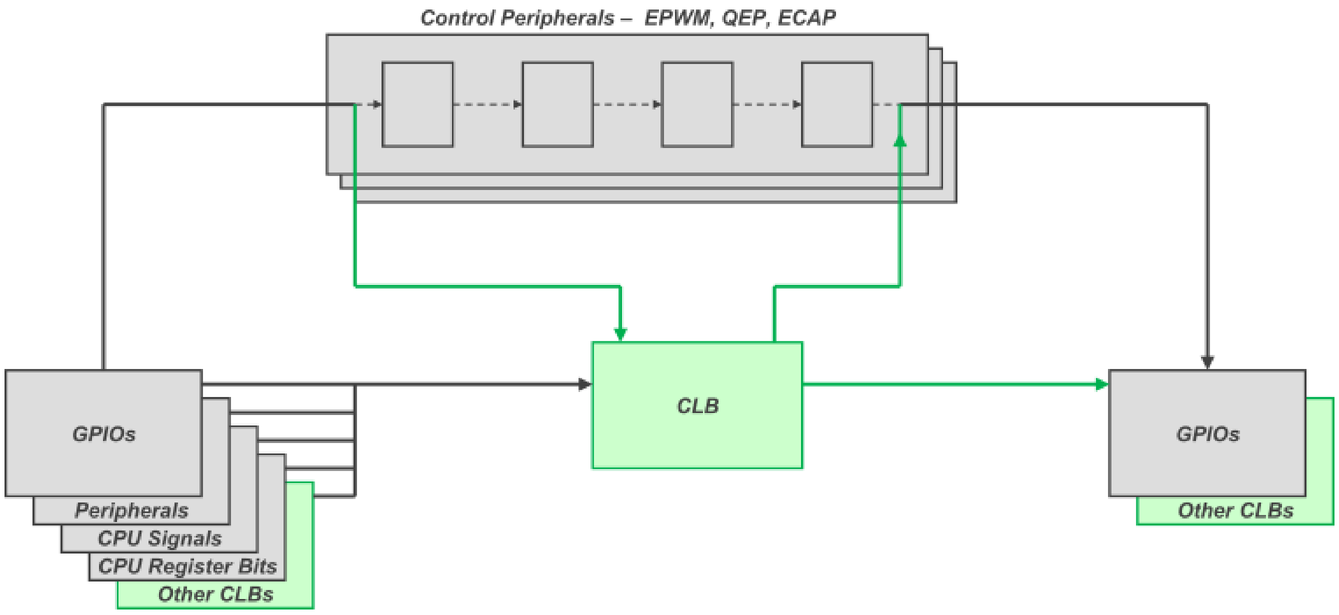


Figure 2-3. CLB Operating Inside and Outside Control Peripherals

2.2 System-Level View of CLB

Figure 2-4 shows the C2000 peripherals, XBars and GPIO Mux. Contrast that with Figure 2-5 showing the same peripherals plus the CLB. By comparing the two drawings, you can see exactly what parts of C2000 are touched by the CLB. In its base-level form, the CLB is composed of CLB1, CLB2, CLB3 and CLB4 blocks. Each CLB block has dedicated connections to corresponding Control Peripherals. For example CLB1 is connected to EPWM1, QEP1 and ECAP1. Likewise, CLB2 is connected to EPWM2, QEP2 and ECAP2, and so on. Additionally, all CLB blocks are connected to a shared group of input signals called Global Signals which originate at all four EPWM modules and the CLB XBar. Every CLB block is also capable of driving a CLB INTR signal to interrupt the CPU or CLA. Finally, every one of the four CLB blocks hangs on CPU and CLA buses allowing both the CPU and CLA to access CLB configuration and data registers.

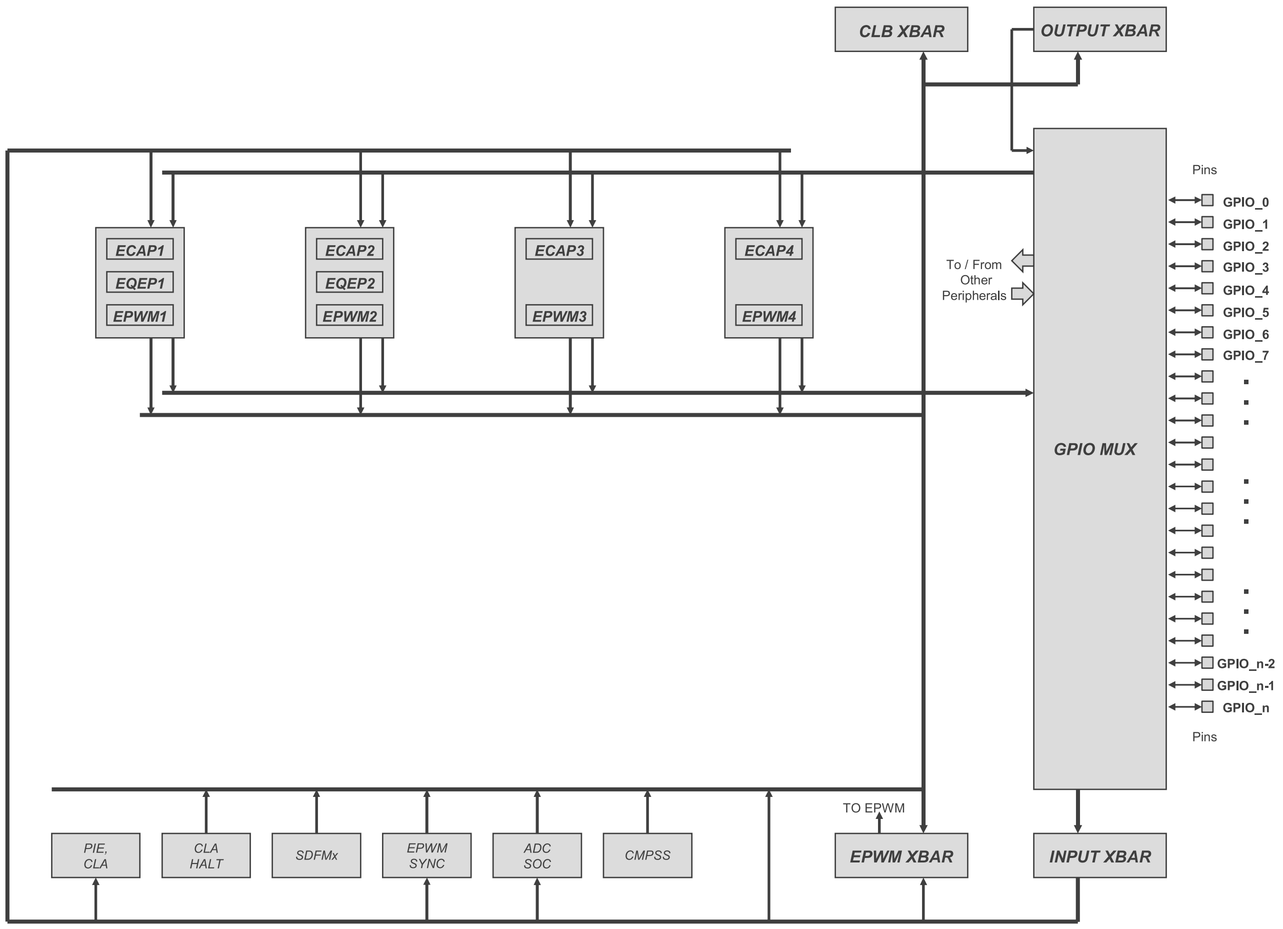


Figure 2-4. C2000 Peripherals Shown Without CLB

Looking at Figure 2-5, the dedicated CLB signals include Local Inputs (black), CLB outputs (green) and Peripheral Signal Mux control signals (orange). Local Inputs for CLB1, for example, contain a mix of signals from EPWM1, QEP1 and ECAP1 plus a couple of CPU signals – CPU TBCLKSYNC and CPU HALT. CLB1 uses Global Signals and Local 1 Signals to generate CLB1 Output Signals. The CLB1 Output Signals can be injected into the EPWM1, QEP1 or ECAP1 Control Peripherals or can be ignored by these peripherals depending on the state of the corresponding (orange) Peripheral Signal Mux. In other words, the (green) CLB Output Signals are replacement signals to be injected in place of an original signal inside a Control Peripheral, while the (orange) Peripheral Signal Mux choose which internal signals of a Control Peripheral are to be replaced (or not) with corresponding CLB Output Signals. Finally, some of the (green) CLB Output Signals are also routed to the CLB/ OUTPUT XBars, where they can be sent via GPIO Mux to selected device pins, or to enter Global Input Signals to become inputs to any of the four CLB modules.

CLB2, CLB3 and CLB4 work the same way, except that CLB3 and CLB4 have fewer Local Signals to work with (missing QEP3 and QEP4 peripherals and no connections to CPU TBCLKSYNC and CPU HALT).

On the inside, each of the four CLB blocks is composed of identical three building blocks – CLB Input Selector, CLB Tile and Peripheral Signal Mux. The CLB Input Selector picks eight signals from the Global and Local Buses. Each CLB Tile applies logic equations to the eight inputs to drive up to eight output signals based on how the Tile has been previously configured via configuration registers. Finally, the eight outputs from a CLB Tile are fed into the Peripheral Signal Mux where they can be selected to replace up to eight of 14 possible internal signals (or none) inside the corresponding Control Peripheral. Again, the operation of the three CLB sections is controlled by corresponding CLB Configuration Registers. While the loading of Configuration Registers for CLB Input Selector and the CLB Peripheral Signal Mux is done by function calls, the Configuration Registers for CLB Tiles are loaded with code generated by the GUI-based SysConfig tool. The following sections explain in more detail how the individual CLB building blocks work together to modify Control Peripherals, implement new peripherals, or generate glue logic with selected GPIOs.

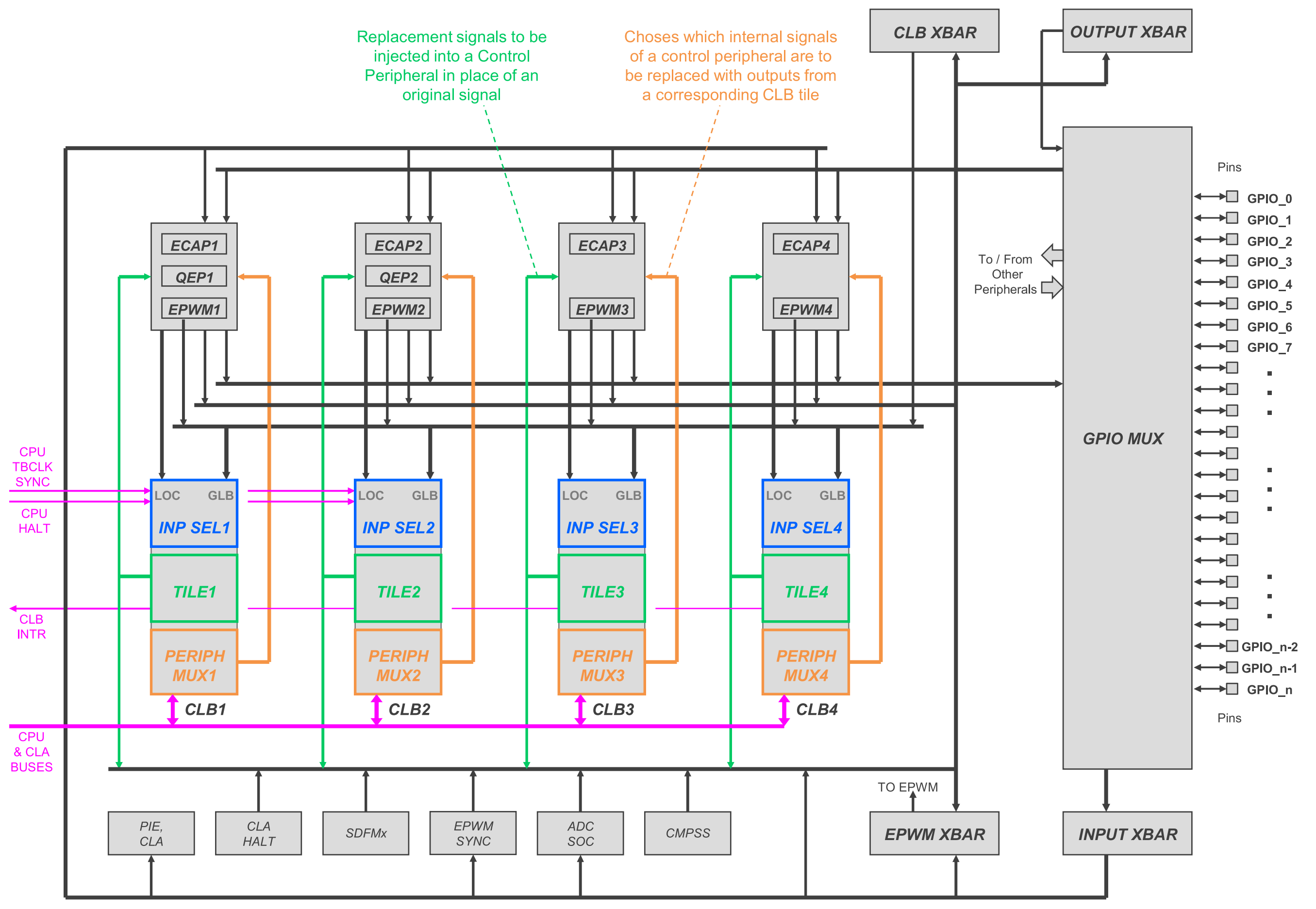


Figure 2-5. C2000 Peripherals Shown With CLB

2.3 Deep-Dive Into the CLB Architecture

Figure 2-6 shows you the internal details of one of the four CLB instances and associated Control Peripherals: EPWM1, QEP1 and ECAP1. The inputs include Global Signals, Local 1 Signals and eight bits from the GPREG Register. Based on how the Input Mux Configuration Registers have been previously configured, the Input Mux Selector picks up to eight signals (blue) from these inputs and sends them to Tile 1 for processing. Based on the state of Tile 1 Configuration Registers, pre-selected logic operations are applied to these eight inputs to produce eight CLB1 Output Signals (green). These are all routed to EPWM1, QEP1, and ECAP1 peripherals, with a couple of the signals also going to the CLB XBar, Output XBar and EPWM XBar where they can be dispatched to EPWM modules, GPIOs or be fed back to any of the four CLB Tiles for further processing (via the Global Signal Bus).

The CLB1 Output Signals that end up at EPWM1, QEP1, and ECAP1 Control Peripherals may be used to displace selected internal signals inside these peripherals, depending on how the Peripheral Signal Mux Registers have been configured. The Peripheral Signal Mux (orange) control multiplexers located at various stages inside Control Peripherals to determine which signals propagate to the next stage – the original internal signal (black) or a new replacement signal from Tile 1 (green). This method provides great flexibility to replace a given stage of a peripheral or to add a new stage (with CLB logic). In the extreme case where inputs to ECAP1 (for example) are immediately fed to Tile 1 and the outputs of Tile 1 are injected at the last stage of ECAP1, the logic inside Tile 1 becomes a new custom peripheral completely displacing the original ECAP1 peripheral. Note that while the internals of the ECPAP1 have now been replaced, the GPIO MUX input and output assignments for ECAP1 remain in place, meaning the new peripheral must use the same input and output GPIOs as the just displaced ECAP1 did.

In other words, when programming CLB1, you should first pick a Control Peripheral to modify, and then determine which internal signals of this peripheral are to be replaced by the outputs from the CLB1 tile (using Peripheral Signal Mux). Once that has been established, you should then use Input Mux Selector to choose the inputs to CLB1 that will be needed by the logic of the new CLB1 function to generate the desired output signal(s). Once the inputs to CLB1 and outputs from CLB1 have been established (using function calls), you are ready to use the GUI-based SysConfig tool to apply necessary logic to the inputs to generate the output(s).

A couple of side notes: both the CLB configuration function calls and the SysConfig tool use CPU or CLA buses (magenta) to load the configuration state into the CLB1 configuration registers. Same buses can also be used to transfer data between the HLC processor inside the Tiles and C2000 memory. The HLC processor can also drive the CLB INTR to interrupt the CPU based on some pre-determined condition reached inside the CLB (more about that later). Furthermore, the Peripheral Signal Mux can be ignored if none of the Control Peripherals are being modified by the CLB (which is the case when CLB is not being used or where CLB is only driving signals to CLB Xbar and other Xbars. The following sections provide additional details about the Input Mux Selectors, Peripheral Signal Mux and CLB Tiles.

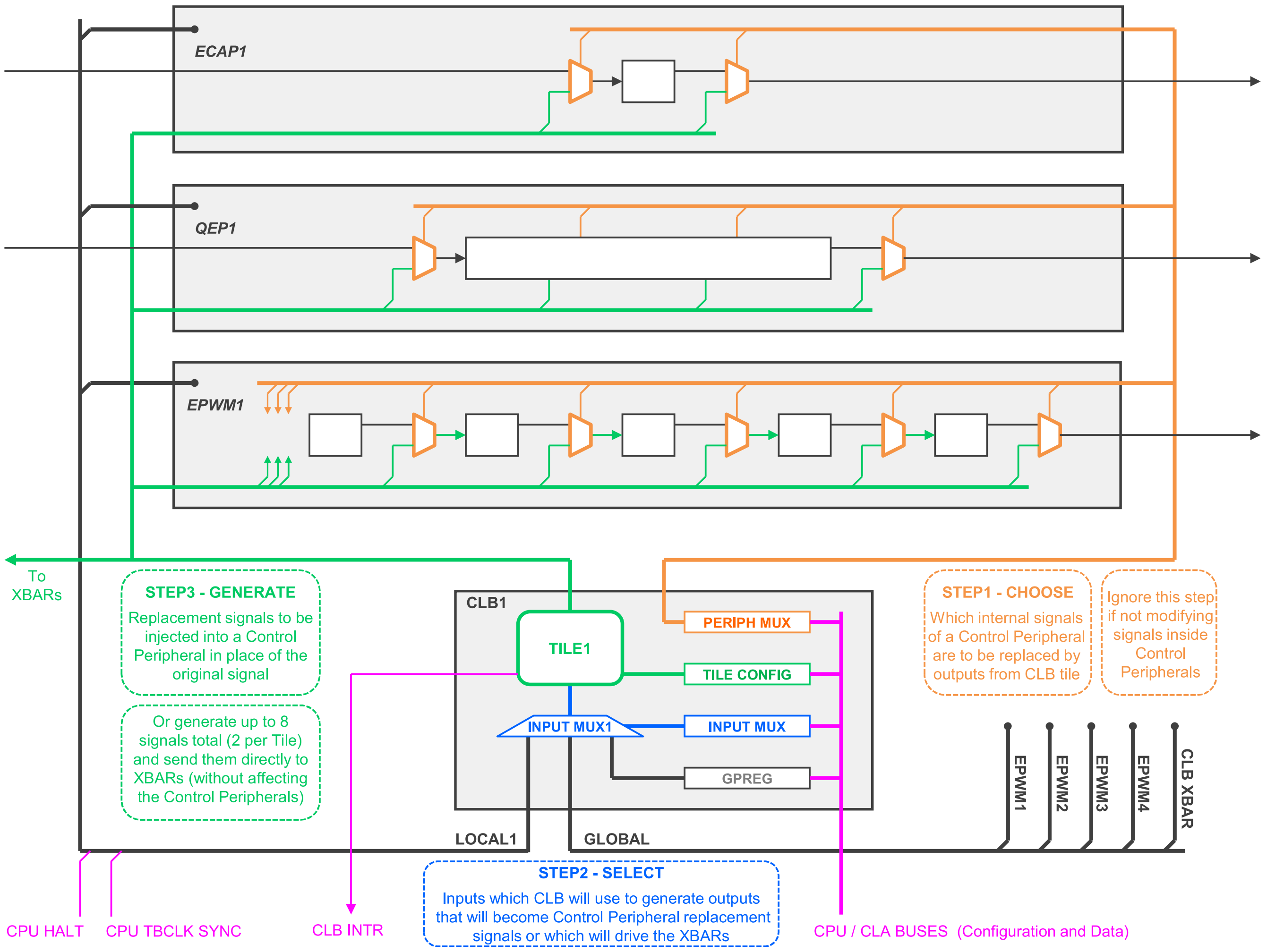

ECAP1
QEP1
EPWM1
To XBARs
STEP3 - GENERATE
Replacement signals to be injected into a Control Peripheral in place of the original signal
Or generate up to 8 signals total (2 per Tile) and send them directly to XBARs (without affecting the Control Peripherals)
CLB1
TILE1
PERIPH MUX
TILE CONFIG
INPUT MUX1
INPUT MUX
GPREG
STEP1 - CHOOSE
Which internal signals of a Control Peripheral are to be replaced by outputs from CLB tile
Ignore this step if not modifying signals inside Control Peripherals
EPWM1
EPWM2
EPWM3
EPWM4
CLB XBAR
LOCAL1
GLOBAL
STEP2 - SELECT
Inputs which CLB will use to generate outputs that will become Control Peripheral replacement signals or which will drive the XBARs
CPU HALT
CPU TBCLK SYNC
CLB INTR
CPU / CLA BUSES (Configuration and Data)

Figure 2-6. CLB Connectivity – CLB1

2.3.1 Input Multiplexers

Figure 2-7 shows a detailed view of Input Mux Selector for Tile 1. Input Mux Selectors for other Tiles are the same with couple of exceptions – Local Inputs for Tiles 3 and 4 have fewer signals, and the 32 bits of GPREG are equally distributed among the four Tiles (one byte per Tile with the low byte assigned to Tile 1). Input Mux Selector 1 has eight identical slices, each generating a single output for the total of eight CLB1 Output Signals. Inside each slice there is a chain of 4 multiplexers starting with the Global Mux that chooses 1 bit from 72 available inputs. That bit becomes bit 0 to the next stage, with Local 1 Input Signal group supplying the remaining 25 bits of stage 2. The single bit output of stage 2 is passed to the next stage unchanged or after passing through clock synchronization circuit. The output of stage 3 passes through a filter where it can become a rising edge pulse, falling edge pulse or remain unchanged. In the final stage 4, the output of stage 3 can be replaced by one of 8-bits from the GPREG Register. Again, the Input Mux Selector is controlled by Input Mux Configuration Registers via function calls. For Global and Local Signal assignments, see the device-specific technical reference manual (TRM) tables.

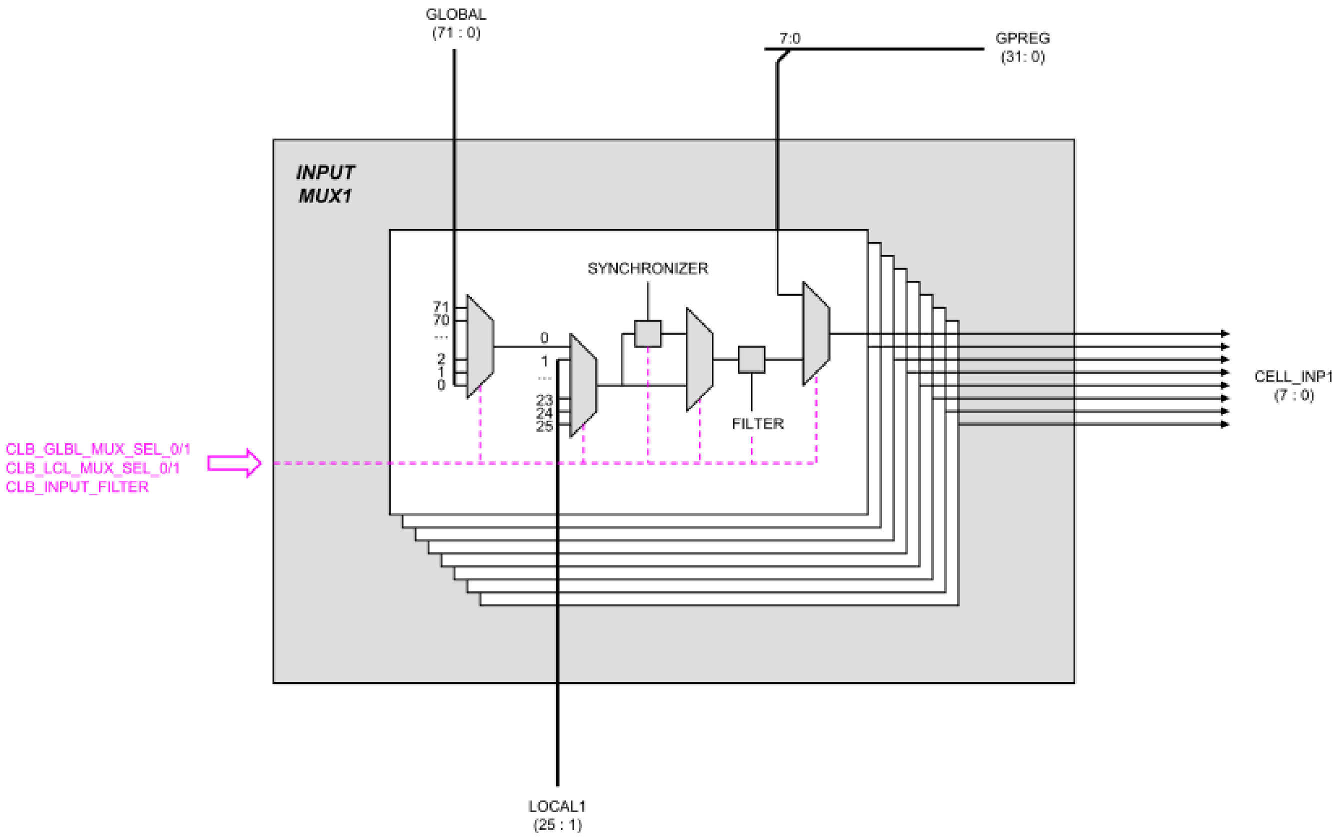


Figure 2-7. Input Selection for CLB1

2.3.2 Peripheral Multiplexers (for Outputs)

Figure 2-8 shows the functional view of Peripheral Signal Multiplexer, where eight CLB1 outputs can replace multiple signals inside selected Control Peripherals.

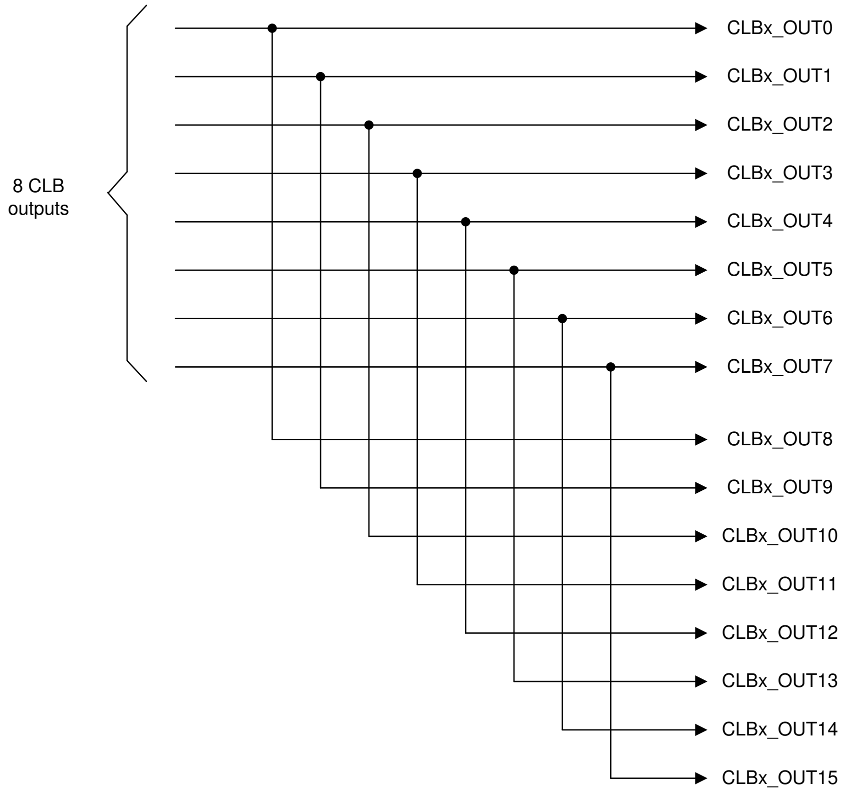


Figure 2-8. CLB1 Outputs

Figure 2-9 shows another view of the CLB1 Peripheral Signal Multiplexer with color-coded sources and destinations included with the signals. There are eight signals coming out of CLB1, of which signals 4 and 5 become direct inputs to CLB XBar, Output XBar and EPWM XBar. Employing the Peripheral Signal Mux, all 8 CLB1 outputs may be used to replace any of the 14 internal Signals of Control Peripheral Group 1 that includes EPWM1, QEP1 and ECAP1. For example, CLB Output Signals 0-7 can replace eight EPWM1 internal signals, CLB Output Signals 0-3 can replace four QEP internal signals, and CLB Output Signals 6-7 can replace two ECAP internal signals. To determine which specific CLB Output Signal replaces which internal signal of EPWM1, QEP1, and ECAP1 peripherals, see the device-specific technical reference manual (TRM) table. Similar signal allocation applies to CLB2, CLB3 and CLB4, each supplying eight dedicated signals to Control Peripheral Groups 2, 3, and 4.

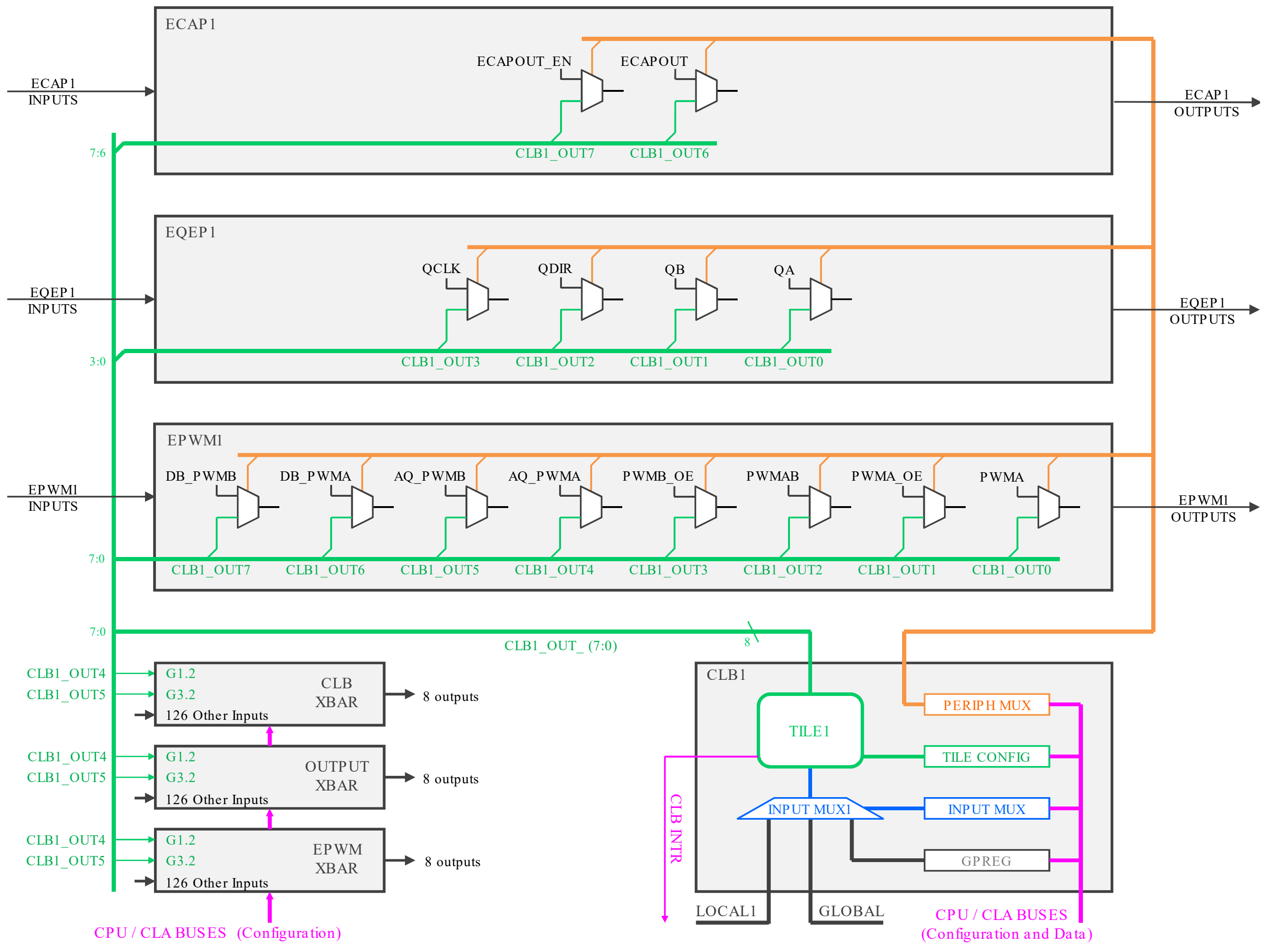


Figure 2-9. Peripheral Signal Mux for CLB1

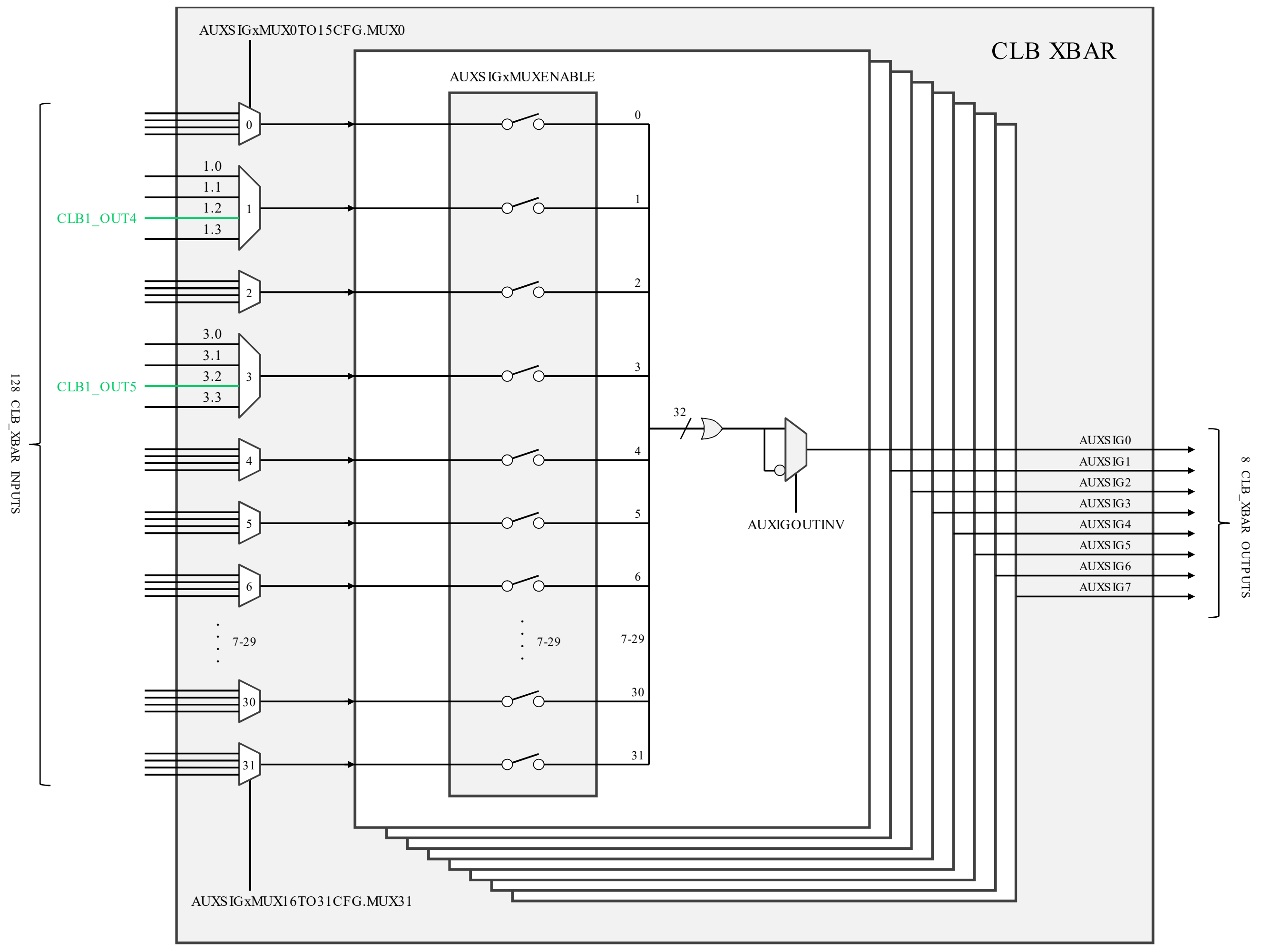


Figure 2-10. CLB XBar

2.3.3 CLB Tiles

Each of the four CLB modules present in the base-level configuration is composed of an Input Signal Selector, CLB Tile and a Peripheral Signal Mux. Input Signal Selector picks which eight signals enter the CLB Tile and the Peripheral Signal Mux allocates eight outputs from the CLB Tile. Inside the Tile itself, logic operations are applied to the eight inputs (as determined by contents of Configuration Registers) to generate the eight output signals. These logic operations can be performed by the HLC block (High Level Controller), three Counters, three LUT4 blocks (Look-Up Table with four inputs), three FSM blocks (Finite State Machine) and eight Output LUT3 blocks (see Figure 2-11). Each of these Logic Primitives is controlled by corresponding Configuration Registers located inside the CLB Tile Register File, which also includes Data Exchange Registers and Control Registers. Data Exchange Registers are used by C2000 CPU to indirectly access HLC Registers and Counter Registers. Data Exchange Registers are also used by the C2000 CPU to load HLC Program Memory. The Tile Configuration Registers are loaded with the CPU using code generated by the SysConfig tool. After configuration, the Data Exchange Registers and Control Registers are accessed by the CPU or the CLA as directed by the application code.

Figure 2-11 shows that one of the main features of a CLB Tile is a 32-bit Logic Bus (brown) that collects results bits from all aforementioned logic primitives. The bottom 8 bits of this bus come from the eight Inputs to the CLB Tile. All 32 bits are subsequently available as inputs to all Logic Primitives, making it possible to easily share intermediate logic results while building the eight outputs from the eight inputs. For example, an output from a Counter may be OR'ed inside the LUT4 block with the output of the FSM block to generate a signal that enters an Output LUT3 to become one of the outputs from the CLB Tile. This 32-bit internal Logic Bus is also available to HLC to trigger up to four different programs on low-to-high transitions of up to four selected Logic Bus signals. The four HLC programs are initially pre-loaded into the HLC Instruction Memory by code generated with the SysConfig tool, and can each contain up to 8 HLC instructions.

While the Logic Primitives are primarily intended to operate on individual signals, the HLC can simultaneously operate on signal vectors (groups of signals) such as exchanging data between four HLC General Purpose Registers (R0, R1, R2, R3) and selected registers (Count, Match 1, Match 2) inside the three CLB Counters. The four General Purpose Registers and the Counter Registers can also be modified by HLC ADD and SUB instructions. Other instructions perform pushing of data from HLC Registers or Counter Registers into the 4-word FIFO, or pulling data from the 4-word FIFO into HLC Registers or Counter Registers. The 4-word FIFO is a form of HLC data memory shared with the host system that facilitates movement of data between the HLC and the C2000 memory. The data movement paths used by the HLC are highlighted blue in Figure 2-11.

On the C2000 side, the pull/push FIFO is accessible via memory-mapped reads and writes. Other registers that can be directly accessed by memory-mapped reads and writes include Tile Logic Configuration Registers, Control Registers and Data Exchange Registers. In Figure 2-11, the paths for these CPU/CLA transfers are highlighted in magenta.

The Data Exchange Registers are used by CPU and CLA to write (but not read) to other resources within the HLC that are not directly accessible via memory-mapped reads and writes. These include HLC Instruction Memory, four HLC General Purpose Registers and three types of Counter Registers: Count, Match 1 and Match 2. These indirect accesses of HLC resources via Data Exchange Registers by CPU and CLA work like this: first, the write address is loaded into the CLB_LOAD_ADDR Register, next the data to be transferred is written into the CLB_LOAD_DATA Register followed by a write to the CLB_LOAD_EN register. This last write triggers a write cycle on the HLC side using Local Interface Bus to complete the transfer. In Figure 2-11, this bus is highlighted in green.

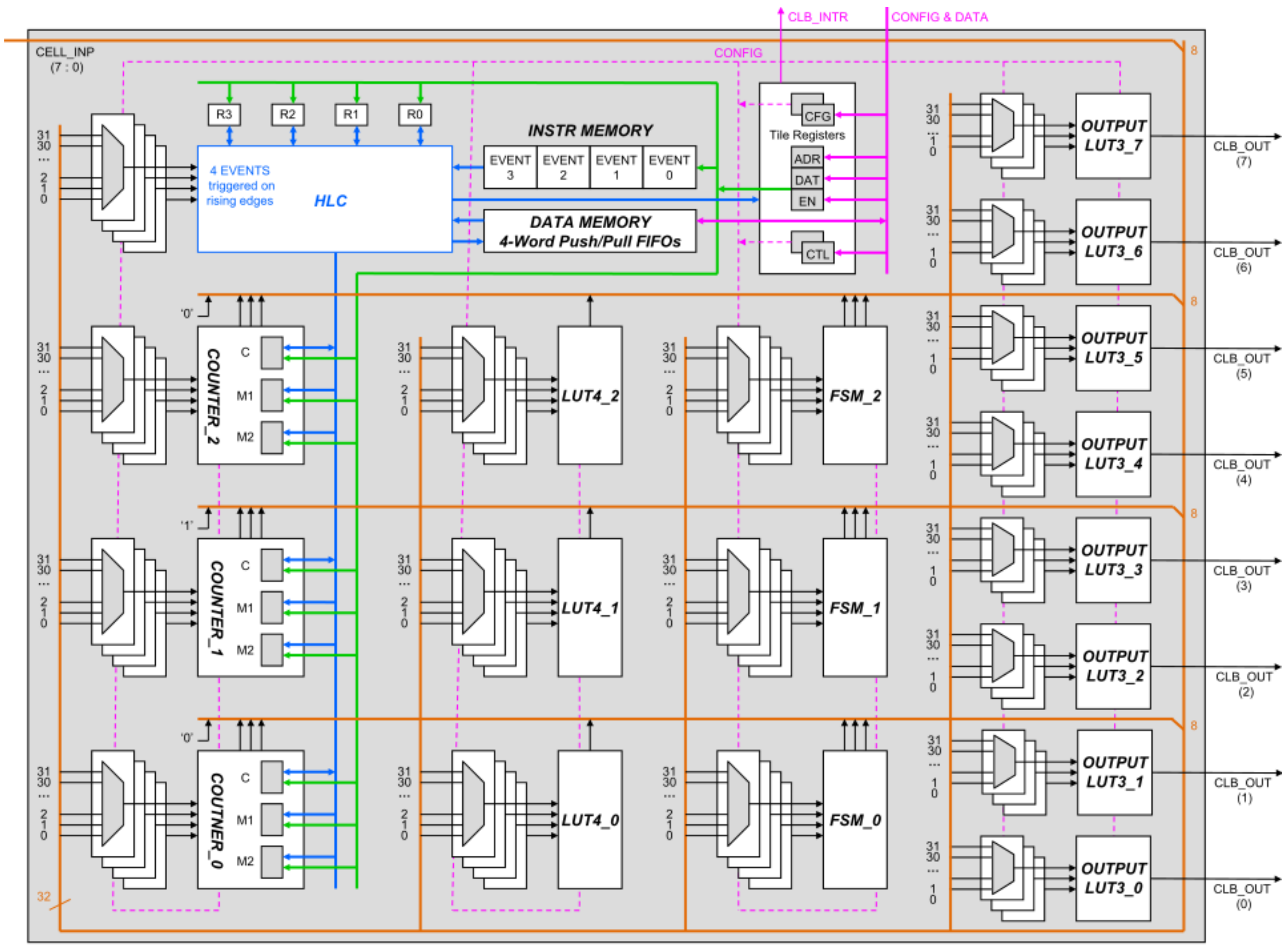


Figure 2-11. CLB Tile

The following sections highlight internal details of the three types of Logic Primitives comprising the CLB Tile: Look-up Tables, Finite State Machines and Counters.

2.3.3.1 Look-up Tables (LUTs)

Figure 2-12 shows two types of LUTs used inside CLB Tiles – The 4-input LUT4 and the 3-input LUT3. Each have one output that can take form of any logical combination of the inputs as prescribed by the corresponding Configuration Registers, which in turn are loaded from code generated by the SysConfig tool. The SysConfig tool also selects which of the 32 bits of the Logic Bus are chosen as inputs to the LUTs. Look-up Tables apply strictly combinatorial logic to the inputs to generate the output (with no clocked registers present). For example, the OUT output can be a logical OR of IN0 input with IN1 input, followed by a logical AND with the IN2 input.

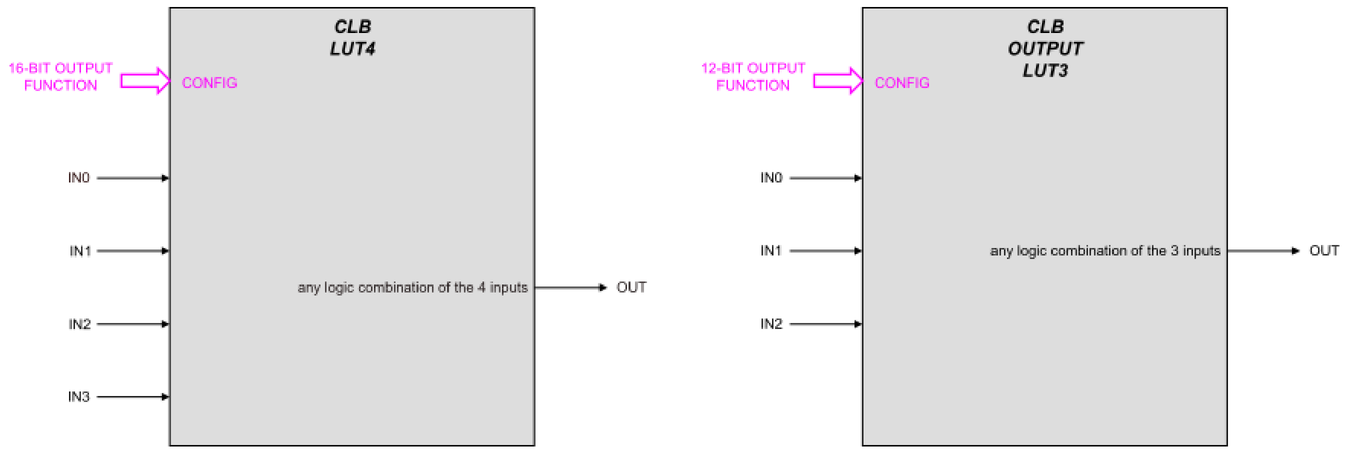


Figure 2-12. CLB Look-Up Tables

2.3.3.2 Finite State Machines (FSMs)

Figure 2-13 shows the Finite State Machine block of the CLB Tile. FSM is composed of three LUT4 combinatorial blocks and two register bits (S0 and S1) clocked by the CLB clock. The two LUT4 blocks feeding new state versions of S0 and S1 get their inputs from FSM EXT_IN0 and EXT_IN1 inputs and the old state versions of S0 and S1. With each rising edge of the CLB clock the new states are updated (according to the logic inside the corresponding LUT4). Both S0 and S1 states are also available as outputs from FSM block to become 2 of 32 bits of the shared CLB Logic Bus. The third LUT4 block operates in two programmable modes. In one mode it gets the same inputs as the other two LUT4 blocks. In the other mode, the S0 or S1 inputs are replaced by EXTRA_EXT_IN0 and EXTRA_EXT_IN1 inputs to expand the number of logical combinations available to drive the FSM_LUT_OUT output. This FSM output also becomes a part of the shared CLB Logic Bus, by which it can be fed to other logic blocks and the HLC. The control bits that determine logic equations inside the three LUT4 blocks and two mode muxes are controlled by corresponding CLB Configuration Registers, which in turn are configured from code generated by the SysConfig tool.

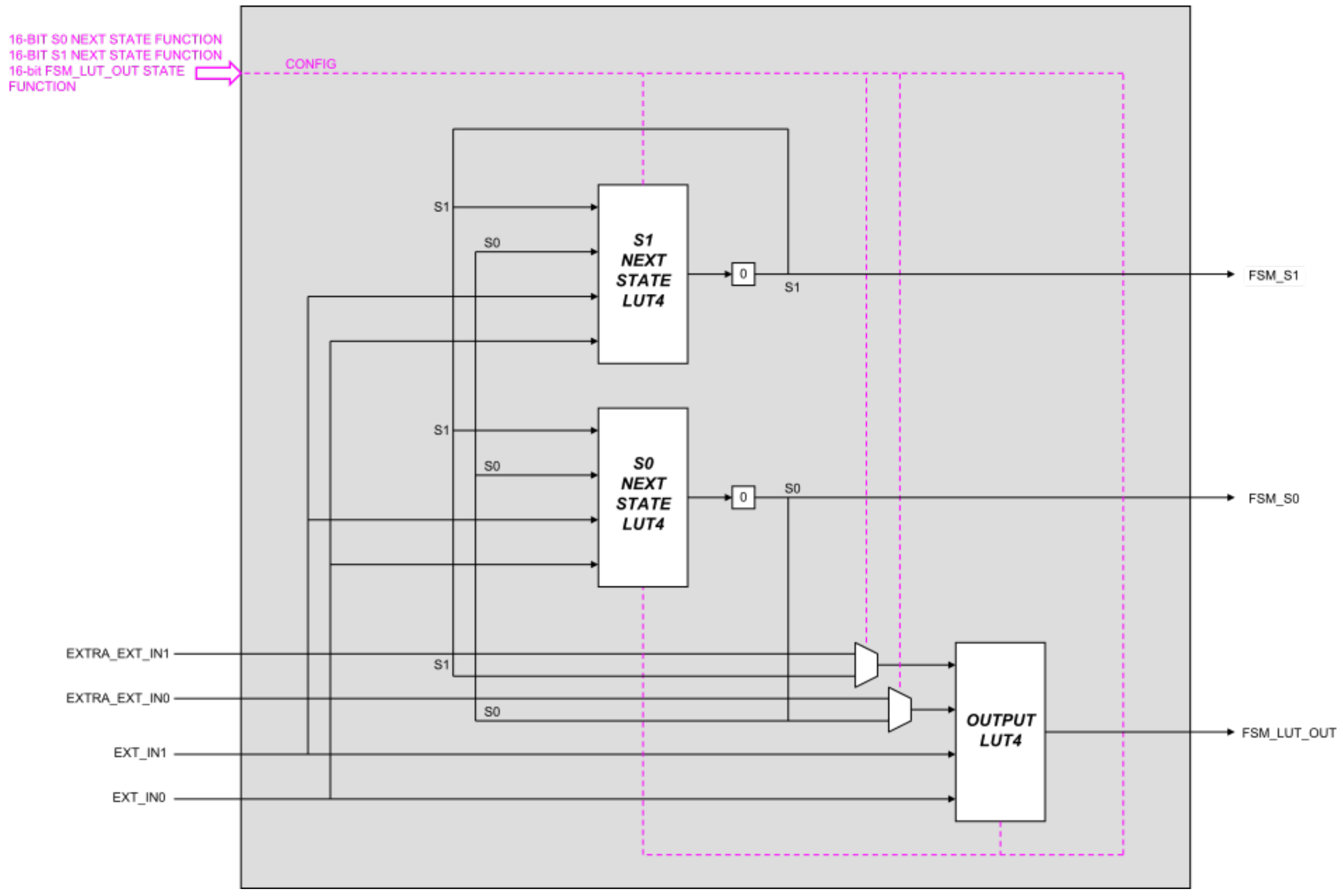


Figure 2-13. CLB FSM Block

2.3.3.3 Counters

Figure 2-14 shows one of three Counters available inside each CLB instance. The core of the counter is the count loop with the Count Register and an Adder that increments or decrements the Count Register by 1 depending on the state of MODE1 input. The count loop also contains a Counter Mux for initializing the Count Register with values other than the incremented or decremented version of the previous count value. For example, the Count Register can be initialized with a new value, a shifted version of the current count value or a version of the current count plus or minus contents of the 32-bit Value Register. The Count Register can only be incremented or decremented on the rising edge of the CLB clock when the ENB signal is asserted. That happens when the MODE0 input is asserted. Loading the Count Register with one of three initial values described above requires a high-to-low transition of the EVENT input. That generates a single-clock pulse that momentarily switches the Counter Mux from increment/decrement mode to load mode and loads the Count Register with the new value. Finally, the Reset input initializes the contents of the Count Register to zero, regardless of the state of other inputs. All four inputs to the Counter are selectable from the 32-bit shared Logic Bus per corresponding Control Registers previously configured by the SysConfig tool.

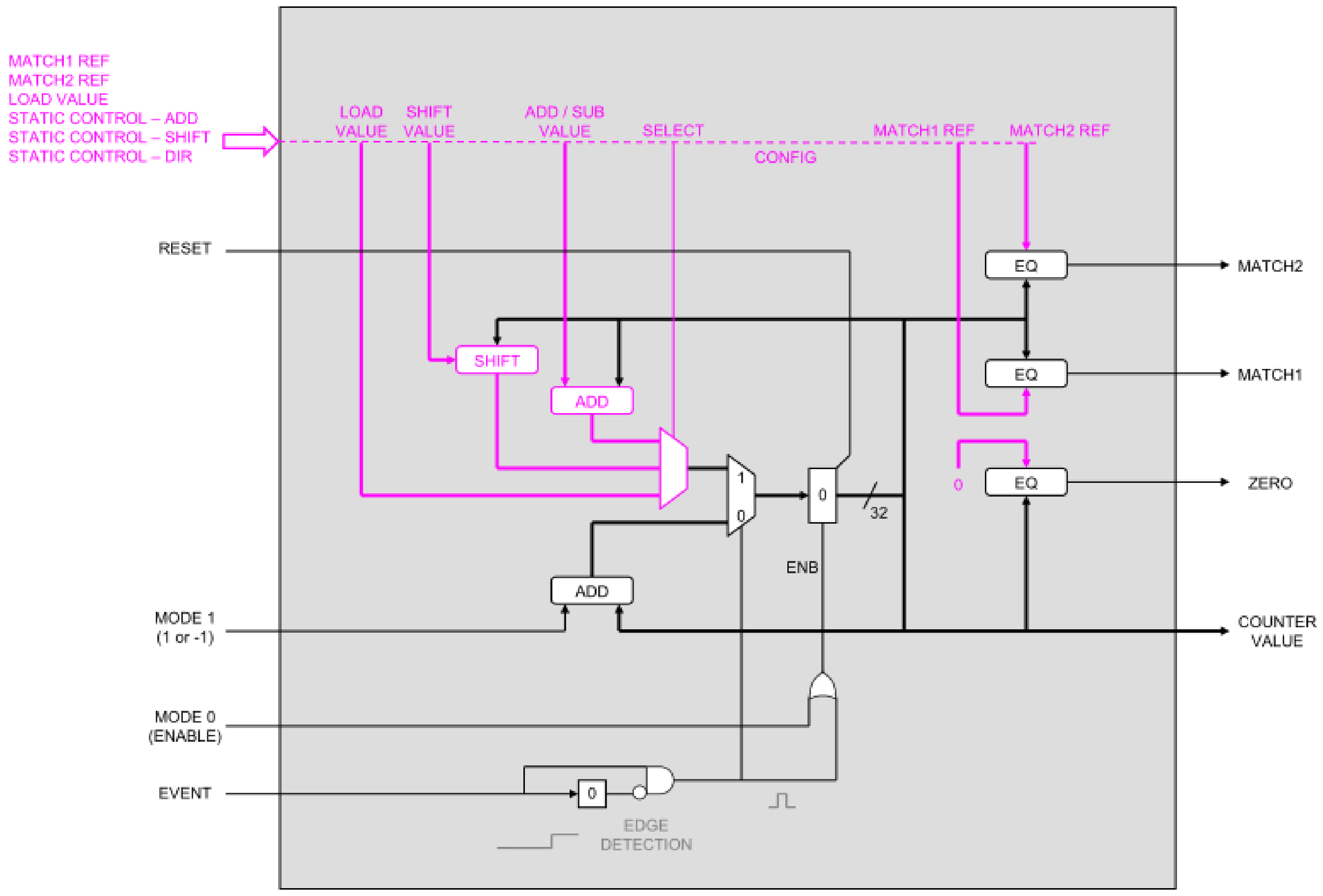


Figure 2-14. CLB Counter Block

The outputs from the Counter Block include the current Counter Value (32-bit wide) and three single-bit status signals from Zero Comparator, Match 1 Comparator and Match 2 Comparator. These comparators issue a logic-1 pulse whenever the current Count Value matches the compare value associated with a given comparator. The three single-bit Counter outputs are mapped to three predefined bits of the 32-bit shared Logic Bus, to be later available as inputs other Counters, LUTs, FSMs and HLC.

The 32-bit Counter Value is directly accessible by the HLC, as are the two Match Registers. The same registers can also be loaded indirectly by the C2000 CPU/CLA via Data Sharing Registers.

3 Overview of CLB Use Cases

Below are four CLB examples that showcase the various ways in which custom logic can be applied to signals internal and external to a C2000 device. Each example is shown in three views that together paint a complete picture of how various portions of CLB are exercised and how the significant signals flow between CLB and other parts of the chip. The first view is a simple schematic block diagram restricted to only showing the signals of interest for a given example and the affected chip components that they flow through. The second view provides the system view of the same transfer with the highlighted signals and blocks of interest superimposed on the dimmed backdrop representing the entire chip Input/Output section. This view provides an instant awareness which portions of the chip need to be configured by function calls and the SysConfig tool for a particular application of custom logic. Finally, the third view dives deeper into the CLB and the Control Peripherals to show exactly which portions of CLB are being used and how they interact with the Control Peripherals. Again, highlighted signals and blocks of interest are superimposed on top of the dimmed background of the local space surrounding the CLB and the Control Peripherals for better understanding of custom logic operations at low level.

3.1 CLB Example 16 – Combine Two EPWM Outputs With a Signal From CPREG Register

In this example, custom logic is applied to outputs of two EPWM blocks and a bit from the GPREG Register, with the resulting output signal exiting the device through a GPIO pin. Figure 3-1 shows a functional schematic diagram for this example. Looking at the signals, the outputs of EPWM1 and EPWM2 modules are combined with GPREG bit inside CLB1 and the result is fed back into EPWM1 to be forwarded to the GPIO MUX and subsequently to the device pin normally used by the standard EPWM1 output. Note that the output of CLB1 is not directly routed to the GPIO MUX, but instead it is returned to the EPWM1 module at the output stage from where it is treated by the rest of the chip just like a normal EPWM1 output. Also note that the EPWM signals that enter CLB1 were extracted from their respective EPWM modules at just before the output stage, but instead being sent to the GPIO MUX they were re-directed to CLB1. For actual implementation of this example in both an FPGA and a C2000 MCU, with identical results for both, see Section 4.

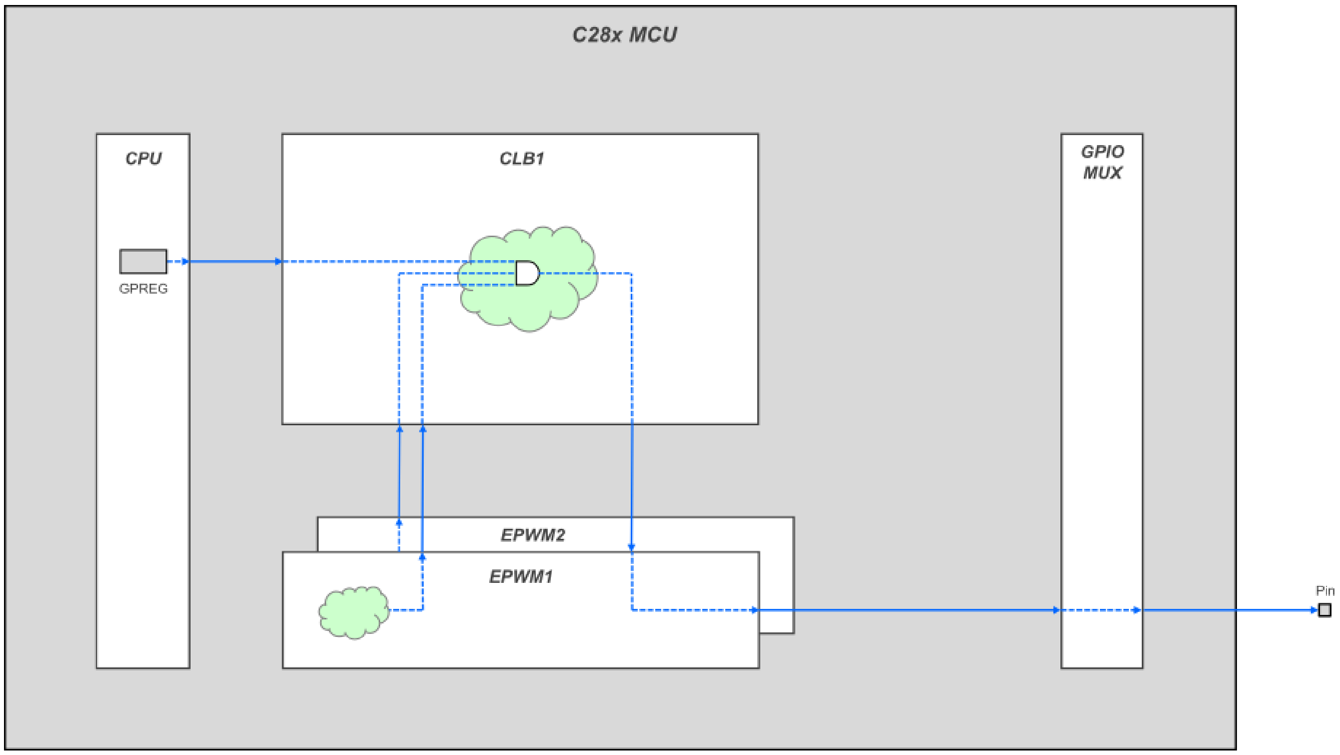


Figure 3-1. CLB Example 16 – Combine Two EPWM Outputs With a Signal From GPREG Register

Figure 3-2 shows the same example through a device-level view to highlight the active blocks and resulting data transfers against the dimmed background of the I/O section of a C2000 device. Here again, it is shown that the generated EPWM1A and EPWM2A signals are diverted into CLB1 by using the Global Signal Bus. As a side note, EPWM1B, EPWM2A and EPWM2B signals go straight from respective EPMW modules into GPIO MUX. After application of custom logic inside CLB1, the resulting signal is fed back into the last stage of EPWM1 via CLB1 Output Bus. Once back inside EPWM1 module, this output signal immediately exits to be forwarded to GPIO MUX as a normal EPWM1A output would. From there it exits the chip though the pin normally assigned to EPWM1A.

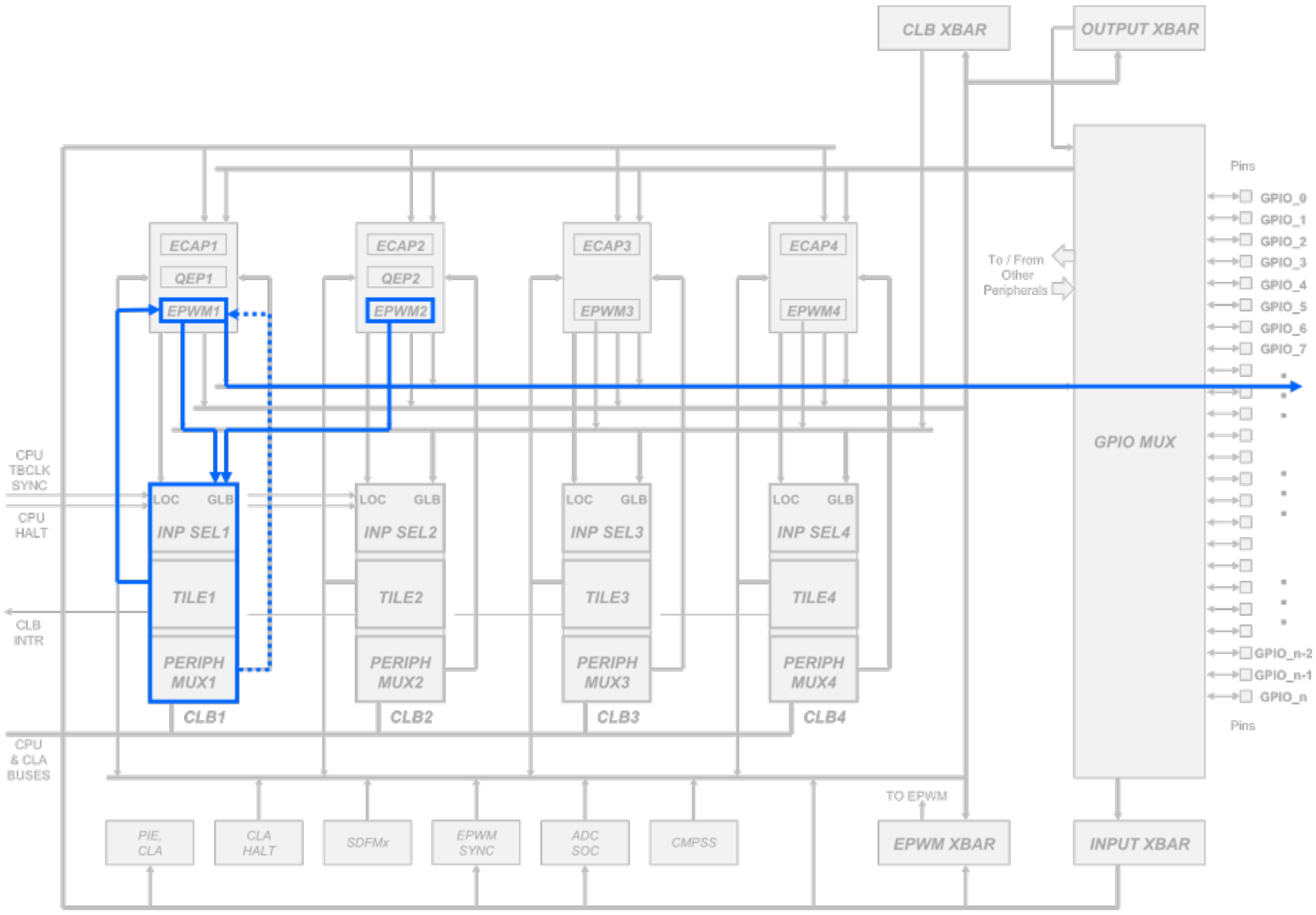


Figure 3-2. Signal Flow in Example 16 – Device View

Figure 3-3 provides the details of signal flow for this example through the lens of CLB1 connectivity. In addition to seeing the inputs and outputs connecting CLB1 with the Local Control Peripherals, this view also provides internal details of CLB1, EPWM1, QEP1 and ECAP1 as well as the data flow inside these. Before going further, note that the Control Peripherals have embedded in them a set of multiplexers that allow substitute signals from CLB1 Local Output Bus to be injected into the peripheral at preselected stages (as directed by CLB1 Peripheral Signal Mux Controls). Also note that signals can be extracted from Control Peripherals at convenient spots to be delivered to CLB1 via Global and Local 1 Buses.

Having set the stage, start tracking the signals for this example. First, the EPWM1A signal is intercepted just before leaving the EPWM1 module to be placed on the Global Signal Bus. Same thing happens to the EPWM2A signal. The Global Signal Bus re-routes both EPWM signals to CLB1 where they are selected to enter TILE1 via Input Mux of Tile 1. The Input Mux also forwards to Tile 1 one of the bits from the GPREG Register. Together, these 3 bits enter Tile 1 of CLB1 while the remaining 5 bits of the 8-bit Input Bus to Tile 1 are left unused. Inside Tile 1, predefined logic is applied to the 3 inputs to produce one output signal (out of eight possible) placing it on the CLB1 Output Signal Bus where it travels back to the EPMM1 module.

Once back inside, the output signal is selected by the Peripheral Signal Mux behind the final stage of EPWM1 module to exit EPWM1 and start making its way toward GPIO Mux where it is placed on the pin assigned to a normal EPWM1A output. Note that at this stage the rest of the chip does not know and does not care whether the signals coming out of Control Peripherals are the original outputs, outputs modified by CLB or entirely new outputs generated by CLB that have nothing to do with the original peripheral. This powerful feature makes is easy for C2000 users to modify peripherals or to create new peripherals without having to rely on external CPLDs or FPGAs, all the while using existing chip resources to deliver signals to and from new/modified peripherals.

Just a reminder, that the path of the signals in this example (and the examples that follow) has been shaped by the CLB1 Configuration Registers controlling Input Mux Selector, Tile 1 Configuration and the Peripheral Signal Mux. The Input Mux Selector and Peripheral Signal Mux control signals being set by application software with function calls, while Tile 1 configuration defined by the SysConfig tool (generating code that sets the Tile Configuration Registers in response to user's interactions with SysConfig Graphical User Interface).

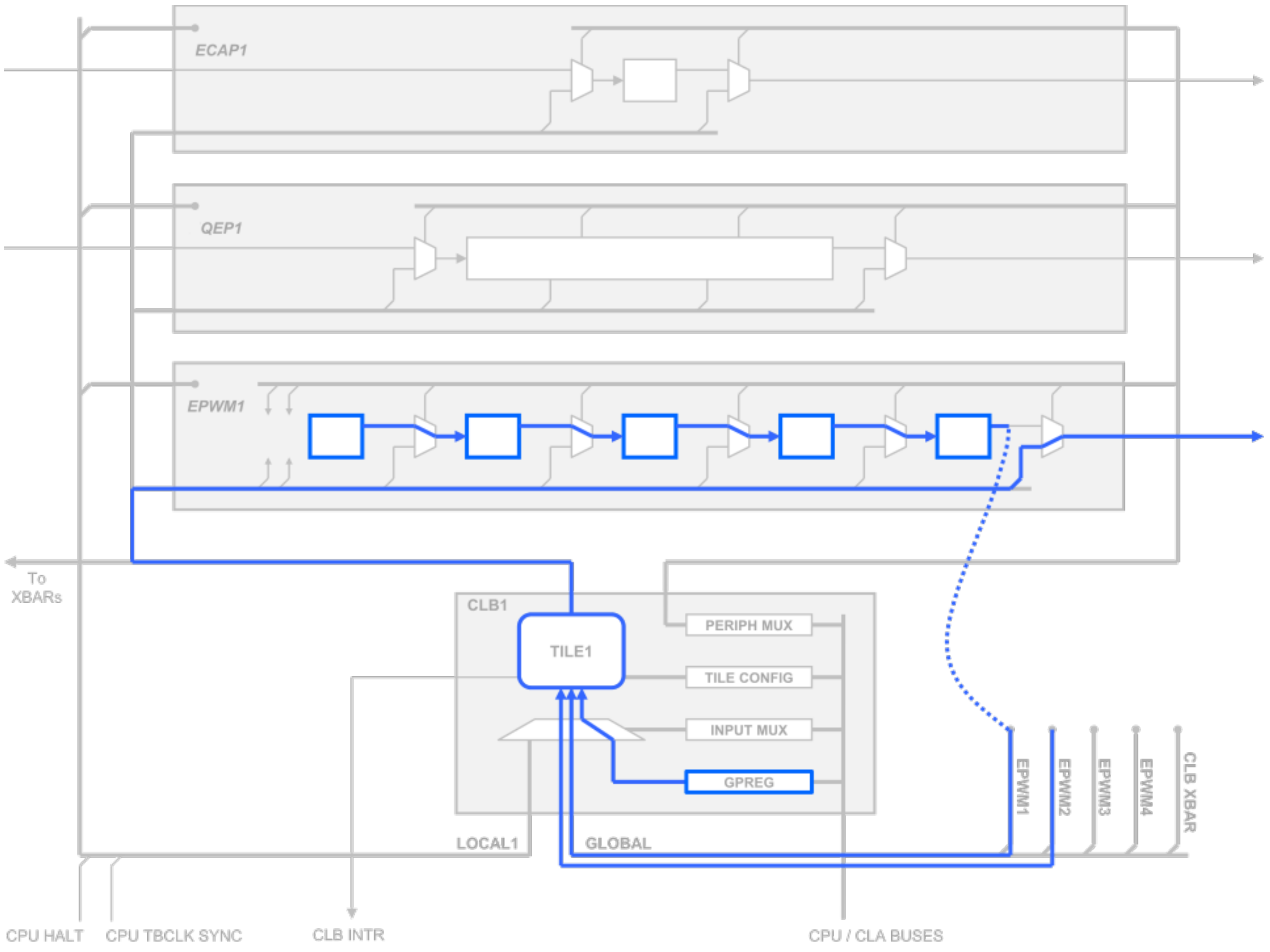


Figure 3-3. Signal Flow in Example 16 – CLB1 Connectivity

3.2 CLB Example 17 – Modify Peripheral Input Signal With a CPU Signal

In this example, some amount of custom logic is applied to the input of QEP2 Control Peripheral. Figure 3-4 shows a functional schematic diagram for this example. Looking at the signals, the input is supplied by a GPIO pin and from there it is passed to QEP2. As soon as it enters QEP2, the input signal is immediately extracted and re-routed to CLB2 where it is combined with the CPU HALT signal. The resulting signal is injected back into QEP just before the first stage of QEP logic just as if it were the original input. From that point on, QEP logic takes over and processes the modified input.

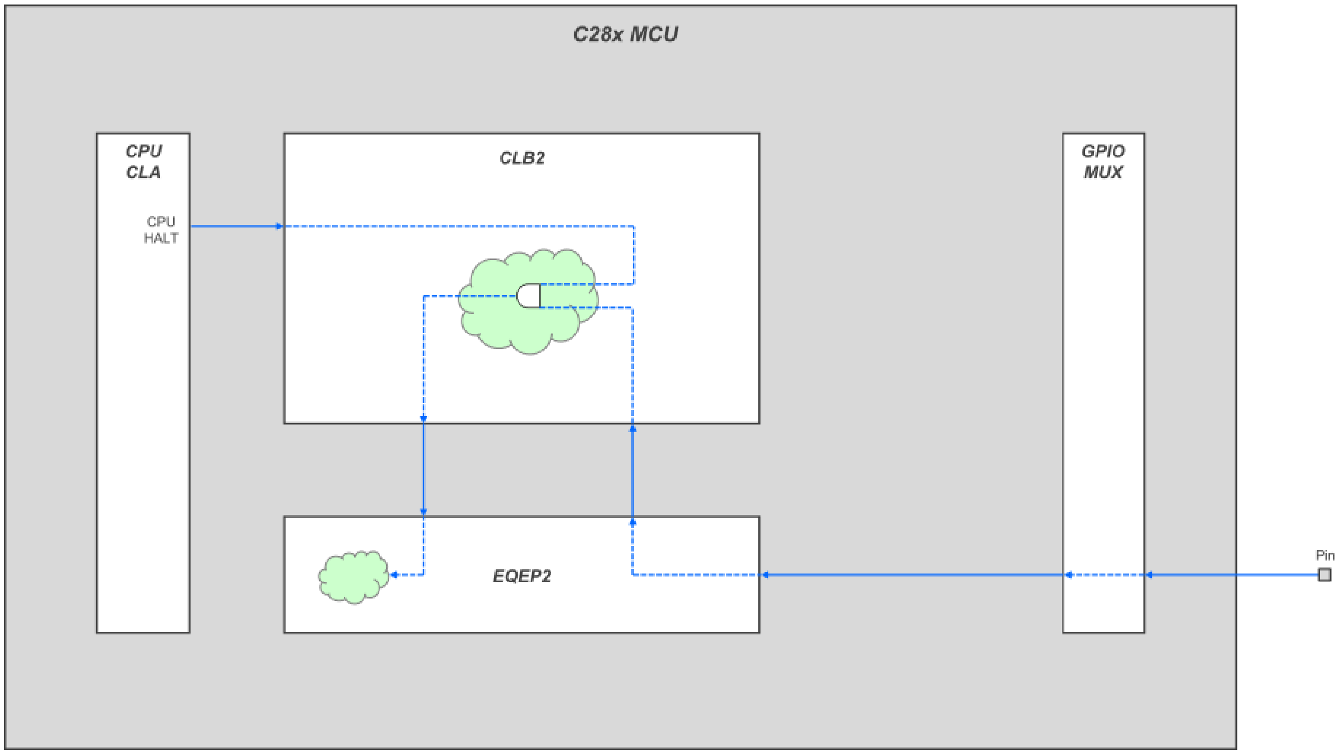


Figure 3-4. CLB Example 17 – Modify Peripheral Input Signal With a CPU Signal

Figure 3-5 shows the same example in a device-level view to highlight the active blocks and resulting data transfers against the dimmed background of the I/O section. Here again, it shows that the GPIO input arrives at QEP2 after passing through GPIO Mux. From there it is placed on the Local2 Signal Bus and forwarded to CLB2. CPU Halt signal also enters CLB2 through the same Local2 Bus. Once inside CLB2, the two signals are selected from the Local2 Bus and passed on to CLB Tile 2. There, they are logically combined (according to Tile 2 Configuration Registers) and placed on the CLB2 Output Bus by which they are fed back to QEP2 peripheral. Simultaneously, the Peripheral Signal Mux 2 block supplies a muxing control signal which tells QEP2 to substitute the newly computed input in place of the original input. After that this replacement input is processed inside QEP2 in the same way the original signal would have been processed.

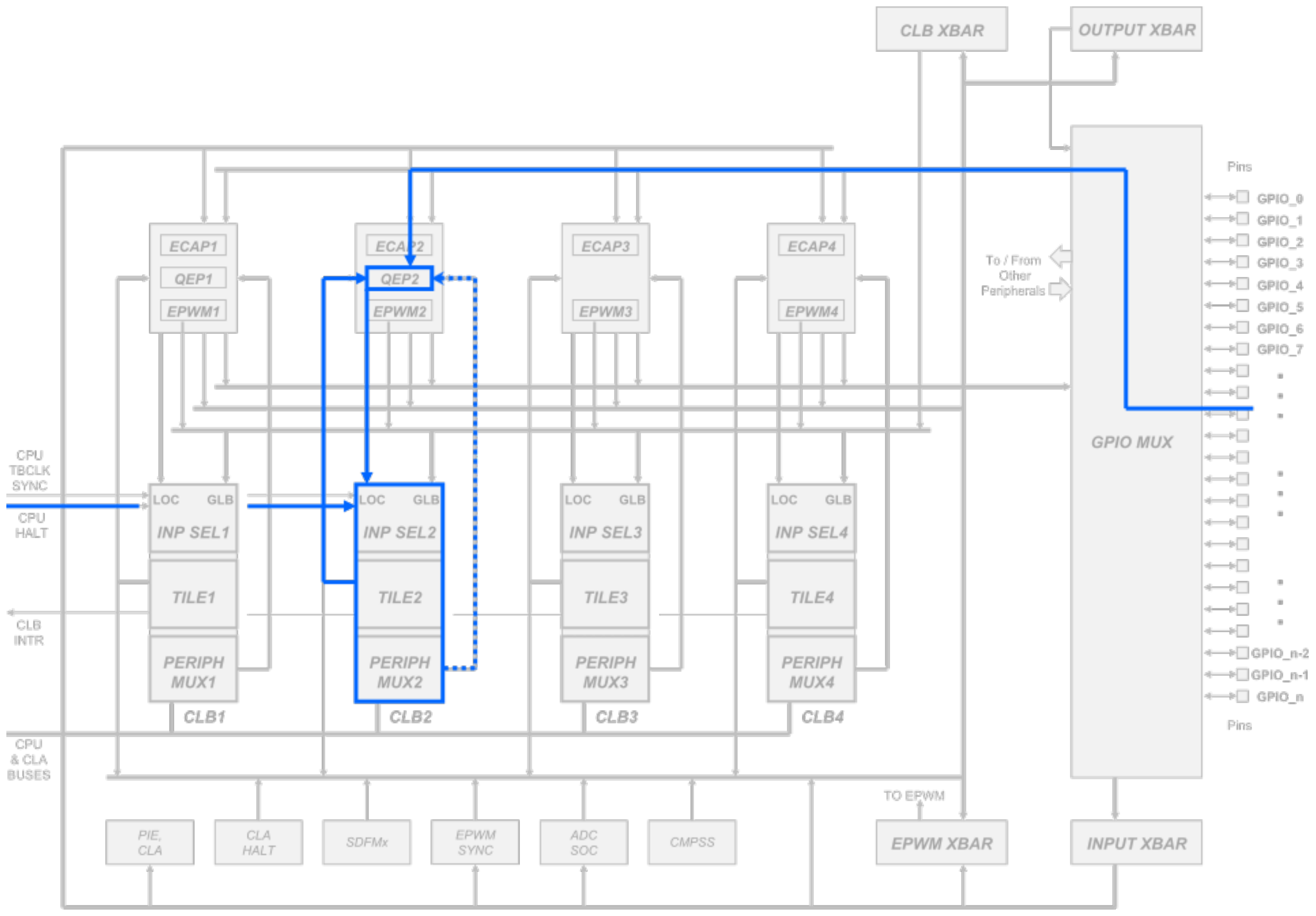


Figure 3-5. Signal Flow in Example 17 – Device View

Figure 3-6 provides signal flow details for this example at the peripheral scale with some visibility inside CLB2 and QEP2 peripheral. Trace the signals again, starting with the original input entering QEP2 being immediately extracted out to be placed on Local2 Bus, which also has the CPU HALT signal. From there, the CLB2 Input Mux sends these two signals to Tile 2 where they are combined and the result is placed on CLB2 Output Bus by which the modified signal is fed back into QEP2 just before the first stage in place of the original input signal. From there on, the modified input is processed by QEP2 as if it were the original input.

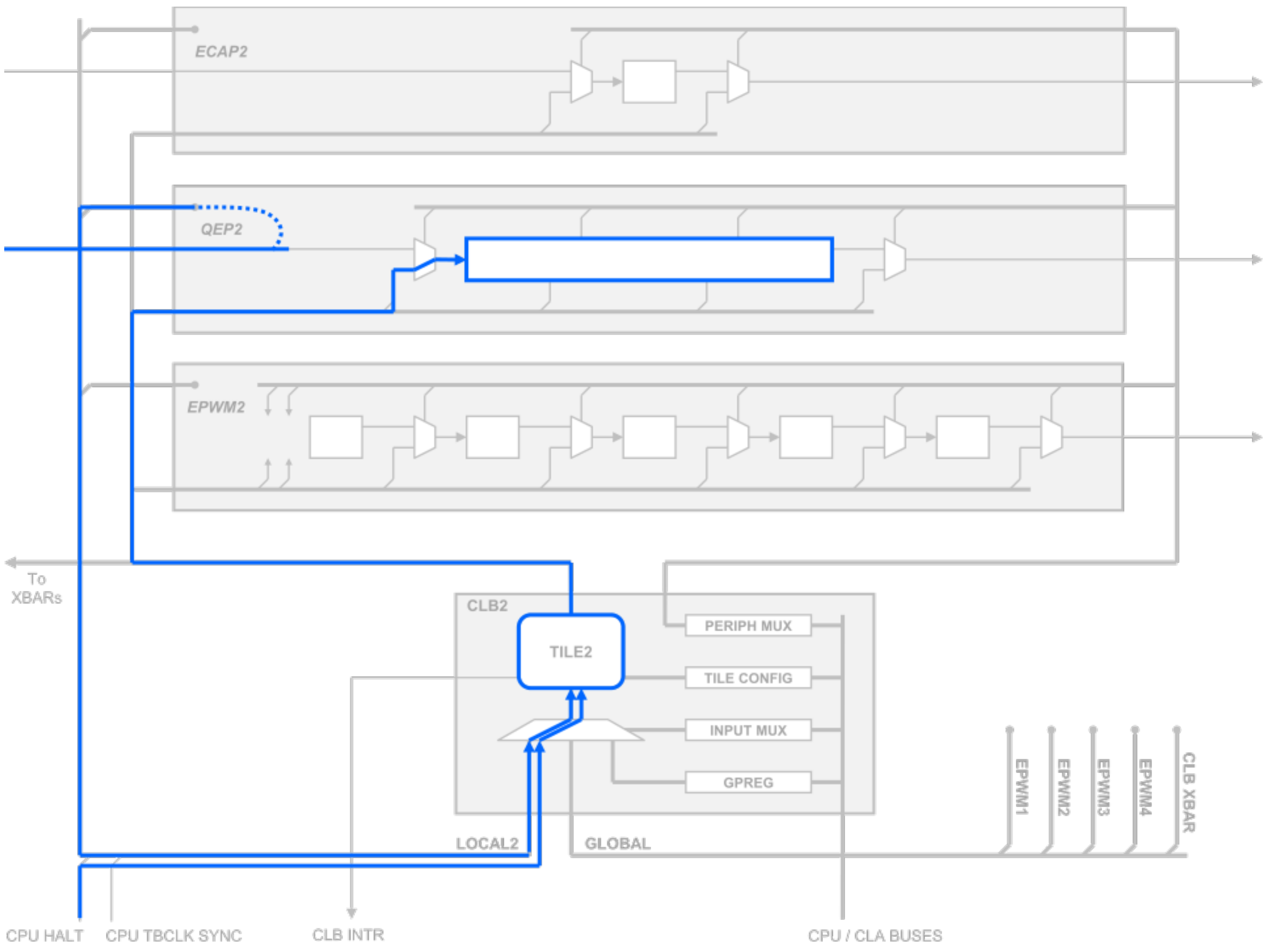


Figure 3-6. Signal Flow in Example 16 – CLB2 Connectivity

3.3 CLB Example 18 – Create Your Own Peripheral in Place of ECAP3

In this example, all logic inside the ECAP3 peripheral is replaced by custom logic that has been programmed inside CLB3. Looking at the signals, the input is supplied by a GPIO pin and from there it is passed on to ECAP3. As soon as it enters ECAP3, the input signal is immediately extracted and re-routed to CLB3 where custom logic is applied to combine it with the output of the CMPSS peripheral (which must first pass through the CLB XBar before entering CLB3). The resulting output is injected back into ECAP3 just after the last stage of ECAP logic, effectively replacing the original ECAP3 logic. From there, this signal travels directly to GPIO Mux where it is subsequently assigned to an output pin (see Figure 3-7).

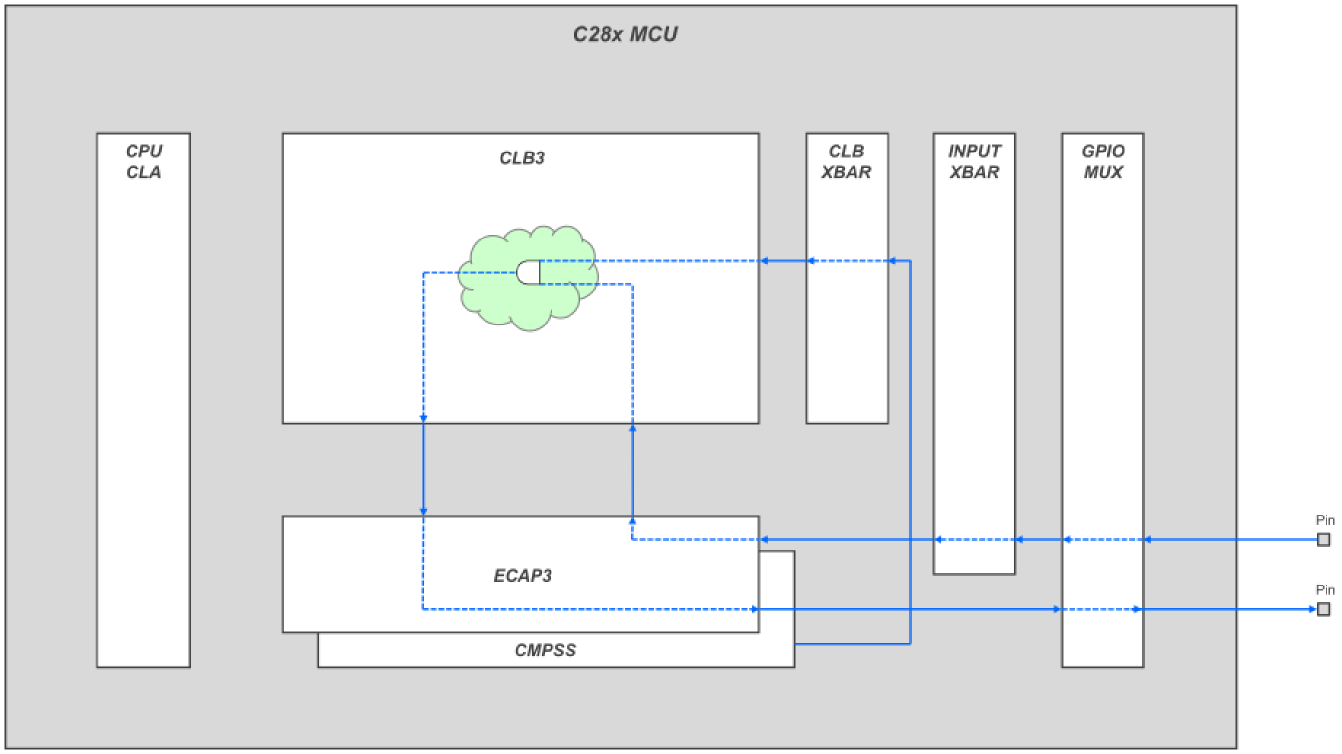


Figure 3-7. CLB Example 18 – Create Your Own Peripheral in Place of ECAP3

Figure 3-8 shows the same example in a device-level view to highlight the active blocks and resulting data transfers against the dimmed background of the I/O section. Here, it shows that the GPIO input arriving at ECAP3 after passing through the Input XBar. From there it is placed on the Local3 Signal Bus and forwarded to CLB3. Meanwhile, the output of CMPSS peripheral also arrives at CLB3 via the Global Signal Bus, after first passing through the CLB XBar. Inside CLB3 the two signals are selected from the Local and Global buses and logically combined inside Tile 3 according to previously defined CLB3 Configuration Registers. The output of that operation is placed on CLB3 Output Bus and sent back to ECAP3. Simultaneously, the Peripheral Signal Mux 3 block supplies a muxing control signal instructing ECAP3 to substitute the just computed signal in place of the original ECAP3 output. After that this replacement output is routed in the same manner as the original ECAP3 output would normaly be – going first into the Output XBar and then into the GPIO Mux where it is placed on the output pin normally assigned to ECAP3.

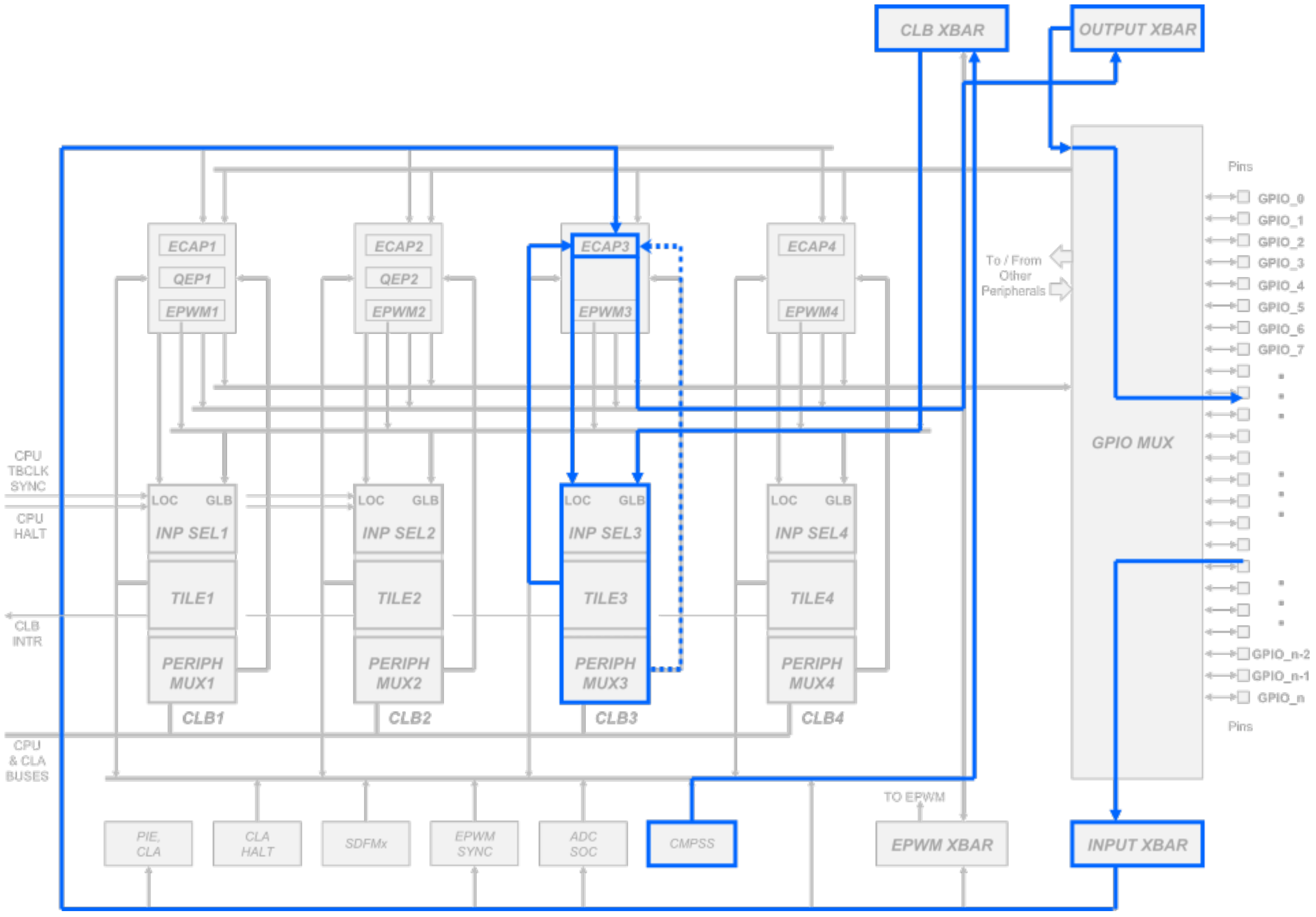


Figure 3-8. Signal Flow in Example 18 – Device View

Figure 3-9 provides signal flow details for this example at the peripheral scale with some visibility shown inside CLB3 and ECAP3 peripheral. The main take-away from this illustration should be that the internal contents of ECAP3 peripheral have been completely replaced by custom logic inside CLB3. This was accomplished by extracting the original input to ECAP3 immediately after it had entered, and injecting the replacement output from CLB3 in place of the original output just before it exits ECAP3. Inside CLB3, the extracted signal enters Tile 3 through the Local3 Bus, where it is combined with the CMPSS output that had arrived at Tile 3 from CLB XBar via the Global Signal Bus. Note that the inputs and outputs to/from ECAP3 have not been altered - the only thing that changed was that the contents of the ECAP3 peripheral have been replaced by custom logic inside Tile3.

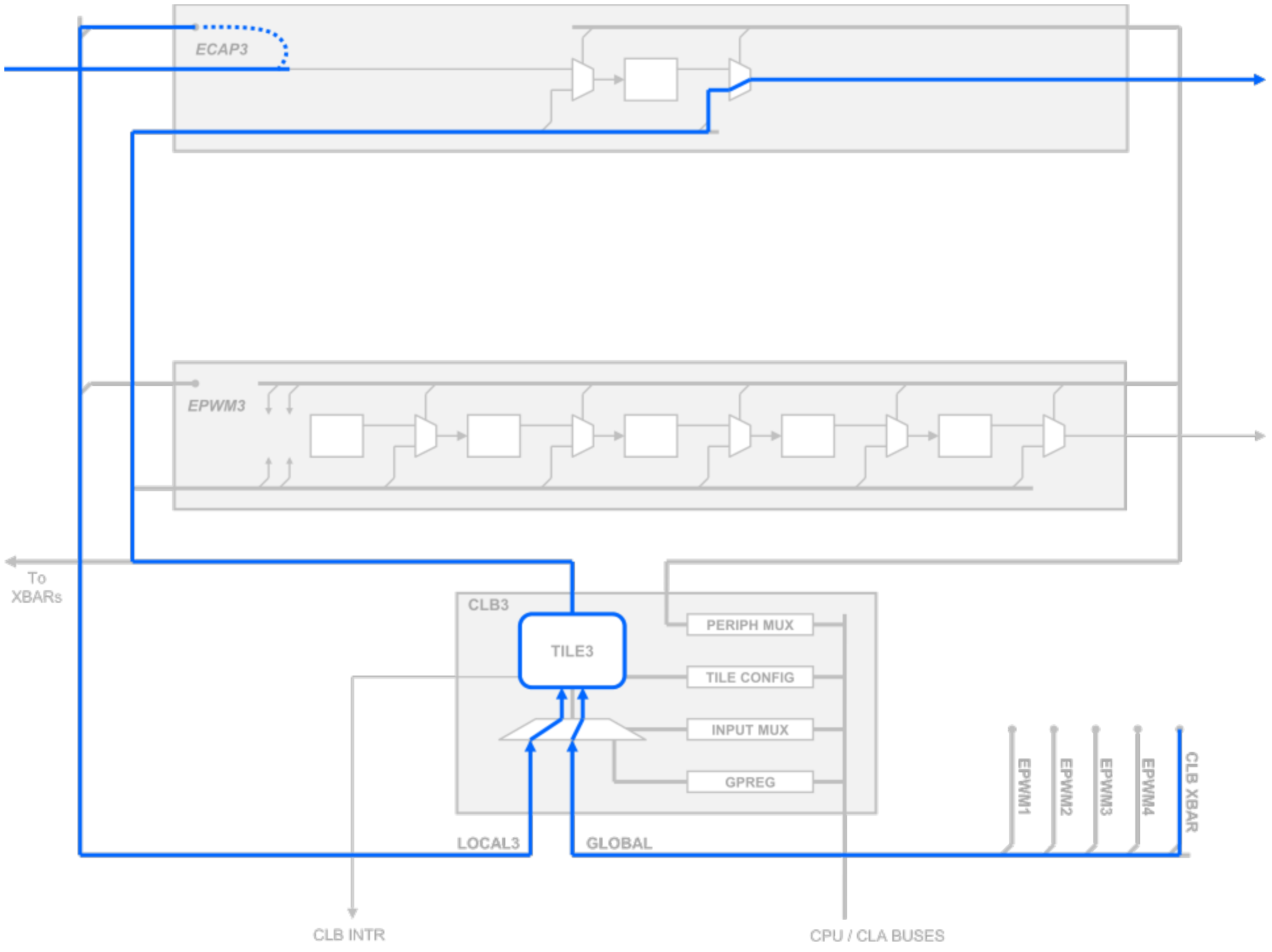


Figure 3-9. Signal Flow in Example 18 – CLB3 Connectivity

3.4 CLB Example 19 – Create Your Own Peripheral Using Only External Signals

This example is different from the previous three examples in that no C2000 peripheral is being used here. Instead, CLB4 operates exclusively on external GPIO inputs to generate a GPIO output. Figure 3-10 shows that there are three external inputs entering GPIO Mux and passing through Input Xbar and CLB XBar on the way to CLB4. Once inside CLB4, the three signals are logically combined according to CLB4 Configuration Registers, and the output is returned via the Output Mux to GPIO Mux where it is placed on the assigned output pin. While the ability to modify internal peripherals is very powerful, this example shows that in order to use CLB you do not have to modify or give-up any peripherals. Custom logic can be simply added on top of all other C2000 functionality to absorb glue logic (for example) from external devices such as CPLDs or FPGAs.

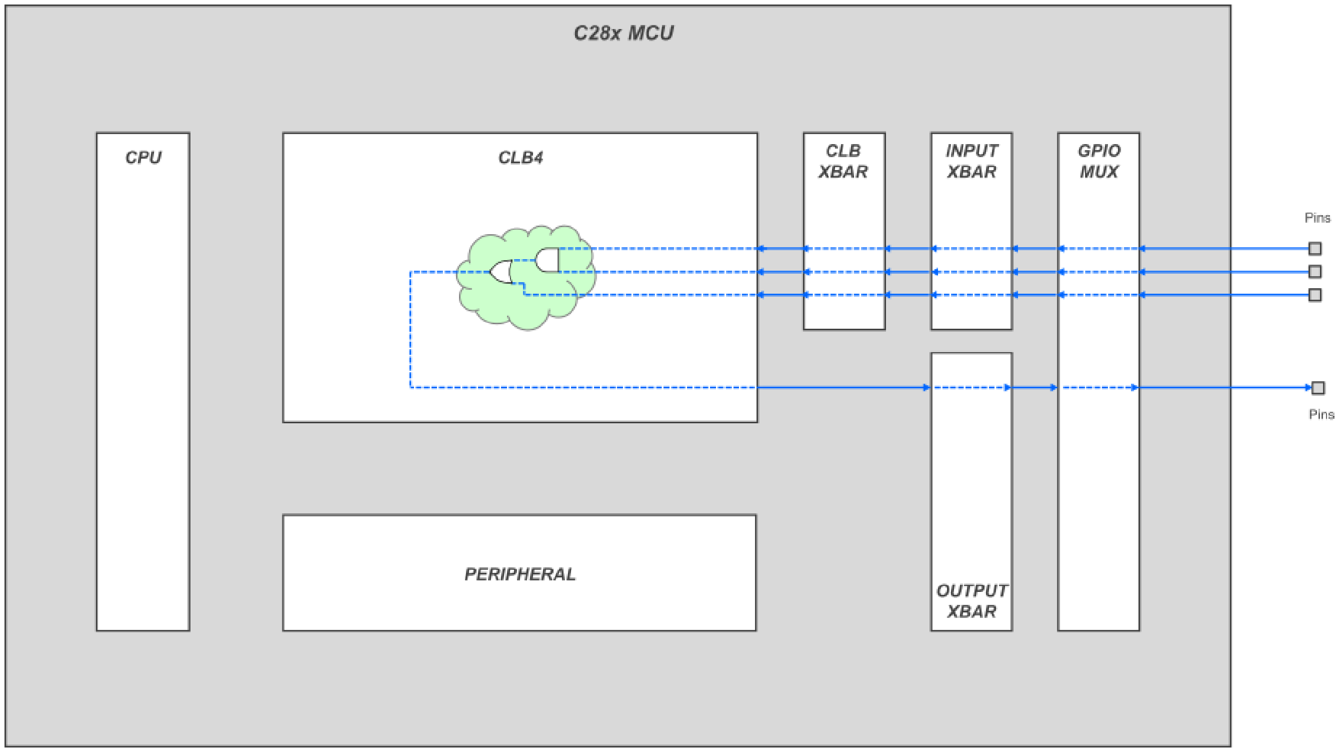


Figure 3-10. CLB Example 19 – Create Your Own Peripheral Using Only External Signals

Figure 3-11 shows the same example in a device-level view to better identify the active blocks and resulting data transfers against the dimmed background of the entire I/O section. Here, it shows three GPIO inputs arriving on the Global Signal Bus via Input XBar and CLB XBar at the Input Select section of CLB4, where they are forwarded to Tile 4 for application of custom logic. The resulting output is routed through the Output XBar to GPIO Mux for placement on the assigned output pin. The Peripheral Signal Mux block of CLB4 is not used in this case and can be ignored because there are no peripherals to be altered (and thus no peripheral signals to be replaced).

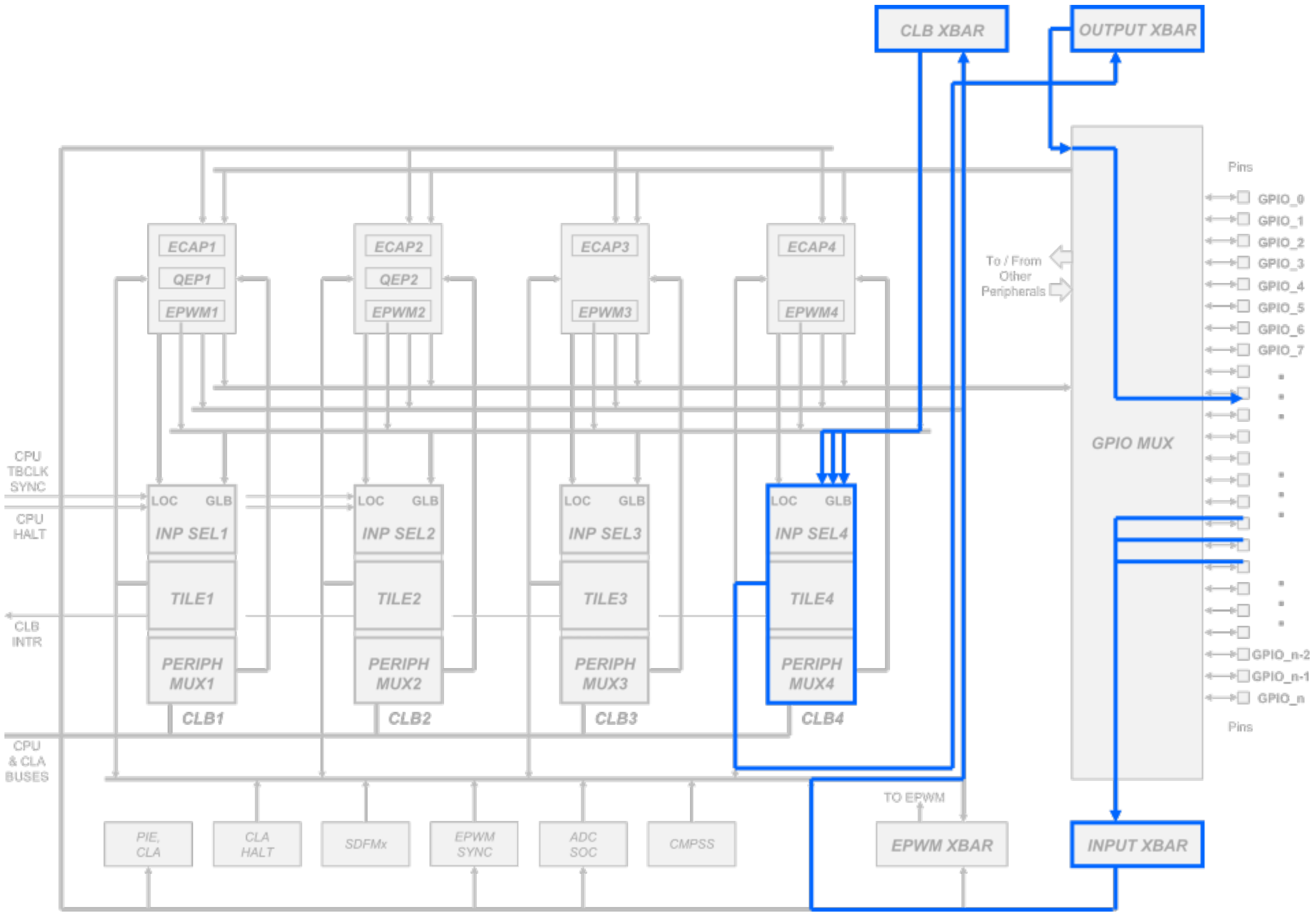


Figure 3-11. Signal Flow in Example 19 – Device View

Figure 3-12 provides signal flow details for this example inside CLB4. The three input signals from CLB XBar (and Input XBar before that) arrive at the Input Mux section of CLB4 using the Global Signal Bus. The Configuration Register contents direct the Input Mux to forward the three inputs to Tile 4 where custom logic is applied to produce the output signal. This output leaves Tile 4 via the CLB4 Output Signal Bus on the way to Output XBar and the GPIO Mux. Again, because no Control Peripherals are involved in this case, the Peripheral Signal Mux is not needed to replace any internal peripheral signals and thus can remain unchanged. The default state of the Peripheral Signal Mux is to retain all peripheral signals (not to substitute any peripheral signals with CLB signals).

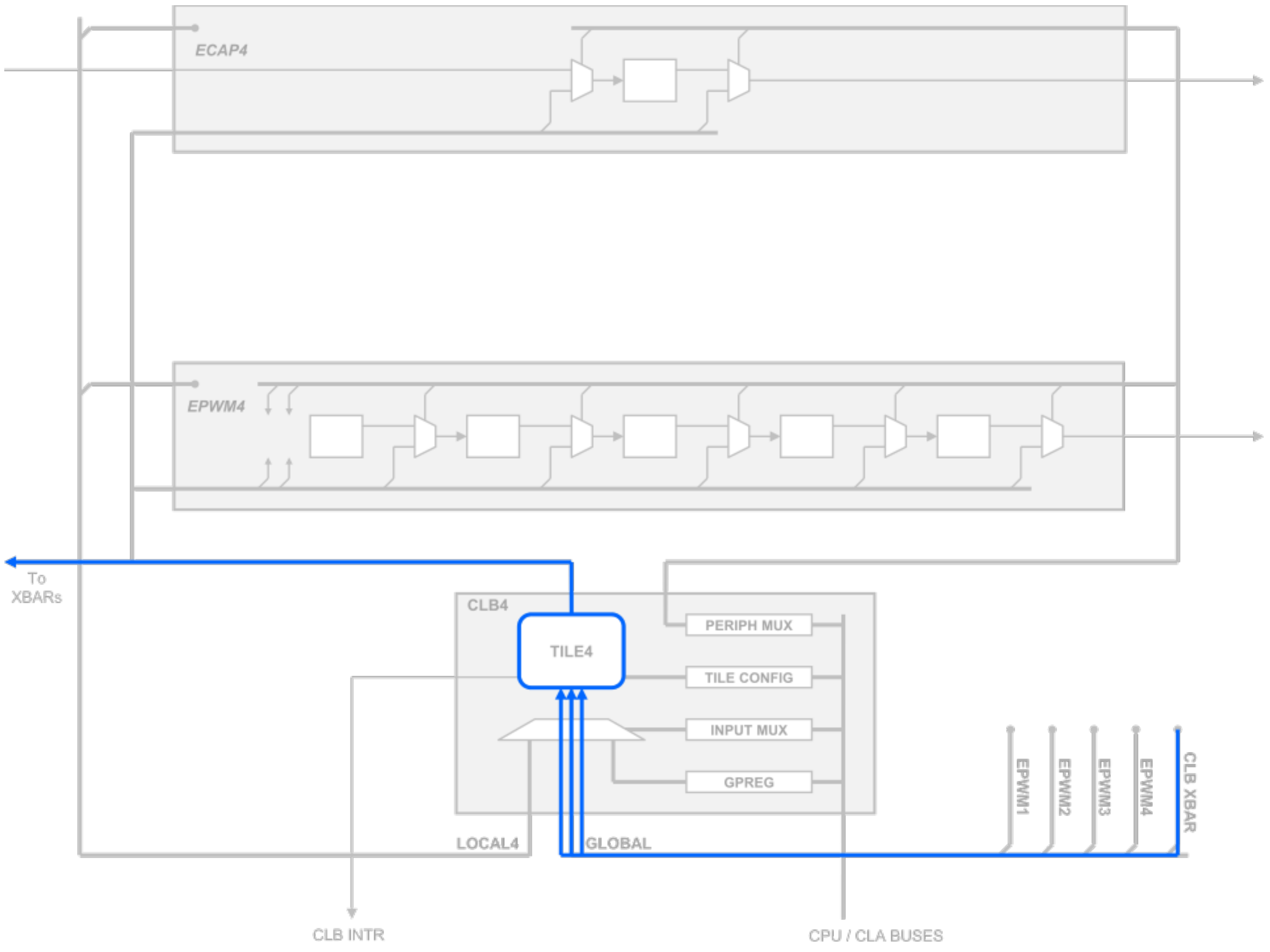


Figure 3-12. Signal Flow in Example 19 – CLB4 Connectivity

4 FPGA to CLB Logic Translation Example 16

This example expands on Example 16 from Section 3.1 to demonstrate how external custom logic can be absorbed into C2000 microcontrollers by using just one of four available CLB Tiles. Figure 4-1 shows an FPGA-based Printed Circuit Board. The FPGA has been programmed with VHDL to include two PWM Generator blocks and a GLUE LOGIC block that combines the two PWM waveforms with an internal signal to drive a single output pin. Taking a closer look at the internal FPGA signals, the two PWM signals INPUT1 and INPUT3 enter the GLUE LOGIC block where they are combined with the static INPUT2 signal (logic 0). The logic function programmed inside the GLUE LOGIC block passes seven pulses of INPUT1 followed by INPUT2 for amount of time equal to three INPUT1 pulses, with the pattern repeating until INPUT3 becomes logic 1, at which point the output goes high (see waveforms of Figure 4-1).

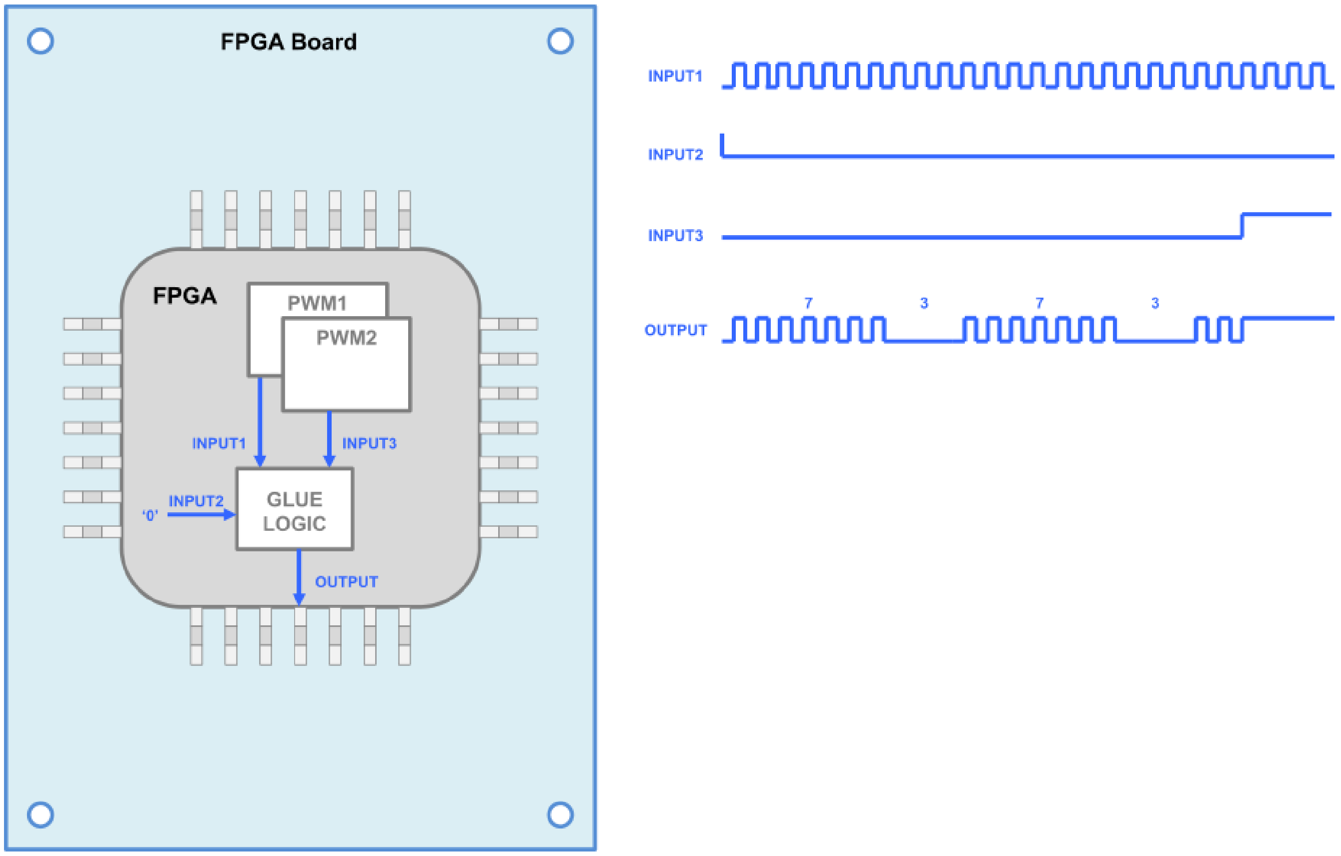


Figure 4-1. System Board With PWM Generators and Glue Logic Inside an FPGA

Figure 4-2 visualizes the translation process to absorb this FPGA logic into a C2000 device, with EPWM1 and EPWM2 Control Peripherals supplying PWM signals and CLB1 providing the GLUE LOGIC. While the GLUE LOGIC inside the FPGA has been programmed using VHDL, the GLUE LOGIC inside CLB1 is programmed with function calls and GUI-based SysConfig tool (no knowledge of VHDL or Verilog is necessary).

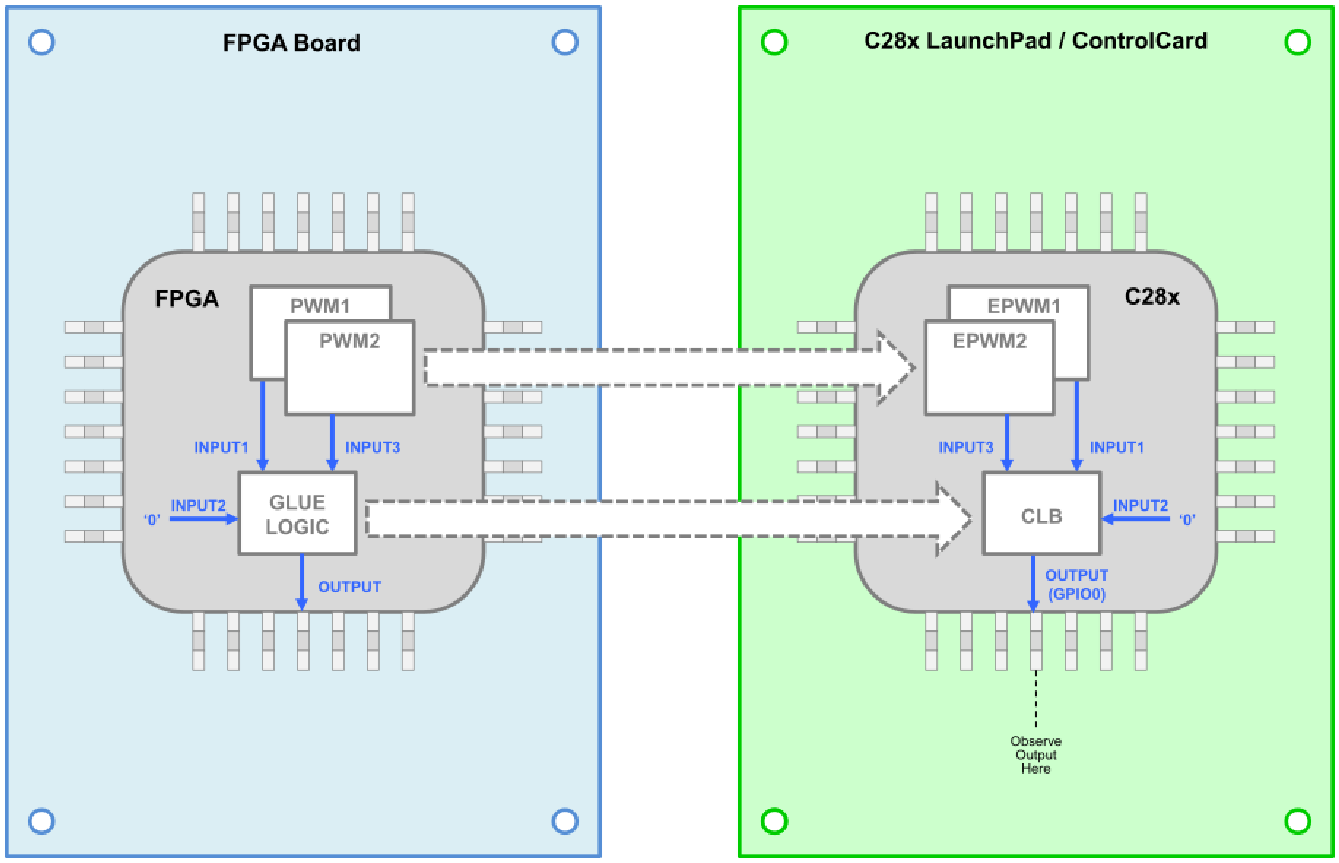


Figure 4-2. Mapping PWM Generators and Glue Logic From FPGA to C2000

The resulting C2000-based board and waveforms are shown in Figure 4-3. Comparing the waveforms, Figure 4-3 shows that they are identical to those of the FPGA board. The remainder of this section covers the details of both the FPGA-based implementation and the C2000-based implementation of this example. These details include schematic diagrams and logic waveforms for both FPGA and C2000 versions of GLUE LOGIC. Also included are FPGA VHDL sources and the C2000 project files.

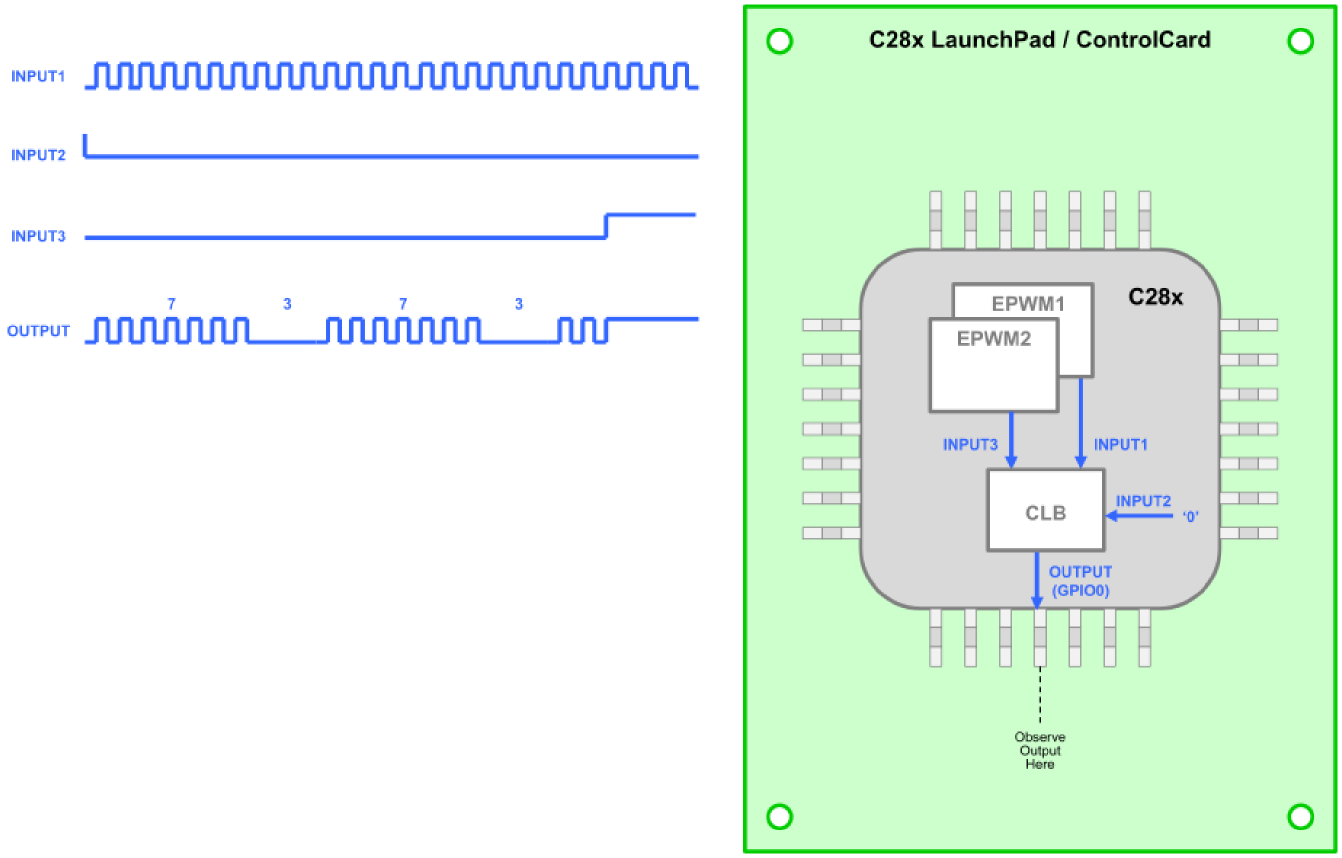


Figure 4-3. Identical Results Obtained With CLB and Two EPWM Peripherals

4.1 The Original FPGA Design

As mentioned above, the FPGA is programmed with two PWM Generators and a GLUE LOGIC block that applies certain logic to PWM outputs and a third signal (logic 0). The logic inside the GLUE LOGIC block is first shown in a schematic form to quickly visualize its function, followed by VHDL source code that mirrors the schematic. Also shown is source code for VHDL Test Bench providing simulation inputs.

4.1.1 Schematic Diagram for FPGA Glue Logic

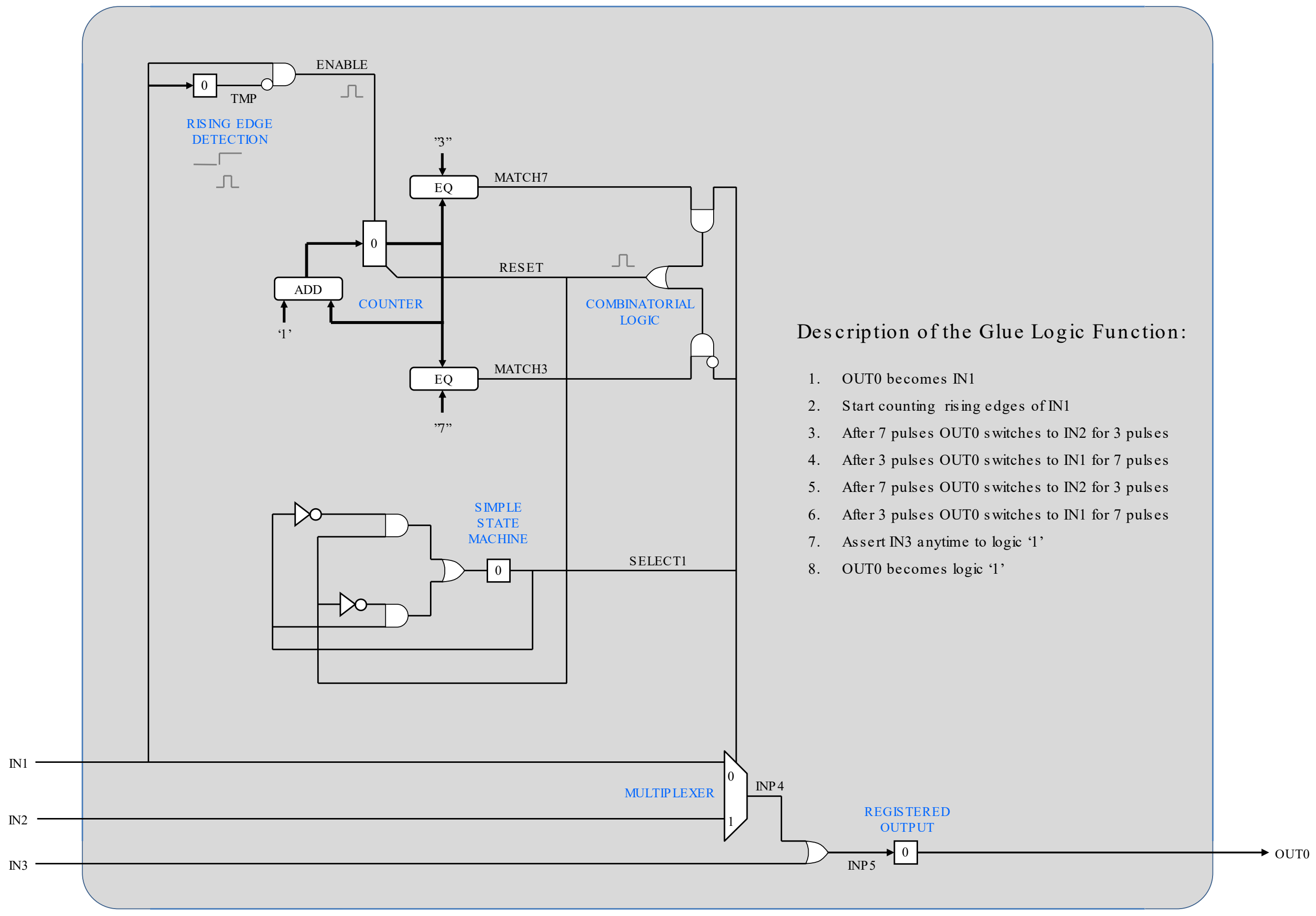
There are three input signals to the GLUE LOGIC block, and one output. Input 1 is a continuous square wave, Input 2 is logic 0 constant and Input 3 is another square wave with lower frequency than Input 1. Output 0 reflects a logical combination of the three inputs according the GLUE LOGIC Function listed in Figure 4-4.

In the beginning, the value of Output 0 reflects Input 1 since the initial SELECT2 output of the Simple State Machine is 0. This causes the Multiplexer circuit to choose Input 1 over Input 2 to drive internal signal IN4. Next, IN4 is logically OR’ed with Input 3 and the result is passed through a Registered Output flip-flop to drive Output 0. Since initially Input 3 is at logic 0, Output 0 starts out as Input 1 delayed by one CLB clock.

Simultaneously, the Falling Edge Detection circuit creates a single-clock pulse ENABLE for each low–to–high transition of Input 1. For each ENABLE pulse, the Counter increments the count value by 1. As it increments, the Counter checks the current count value against two match values – MATCH1 and MATCH2. As soon as either one of the match value is reached, the Combinatorial Logic issues a single RESET pulse to the Counter and the

Simple State Machine. The first time this happens is when the count becomes 7. The resulting RESET pulse resets Counter back to 0 and flips the state of Simple State Machine (SELECT2 signal) from 0 to 1.

This new state causes the Multiplexer to choose Input 2 over Input 1, thus the output becomes 0 after one clock delay. In this new state, the Counter will issue another RESET pulse after only three pulses of Input 1 when MATCH2 is reached, after which the pattern repeats and the Counter starts counting to 7 again. This continues until Input 3 becomes logic 1, at which point Output 0 is forced to logic1 regardless of the logic levels of Input 1 or Input2.



Description of the Glue Logic Function:

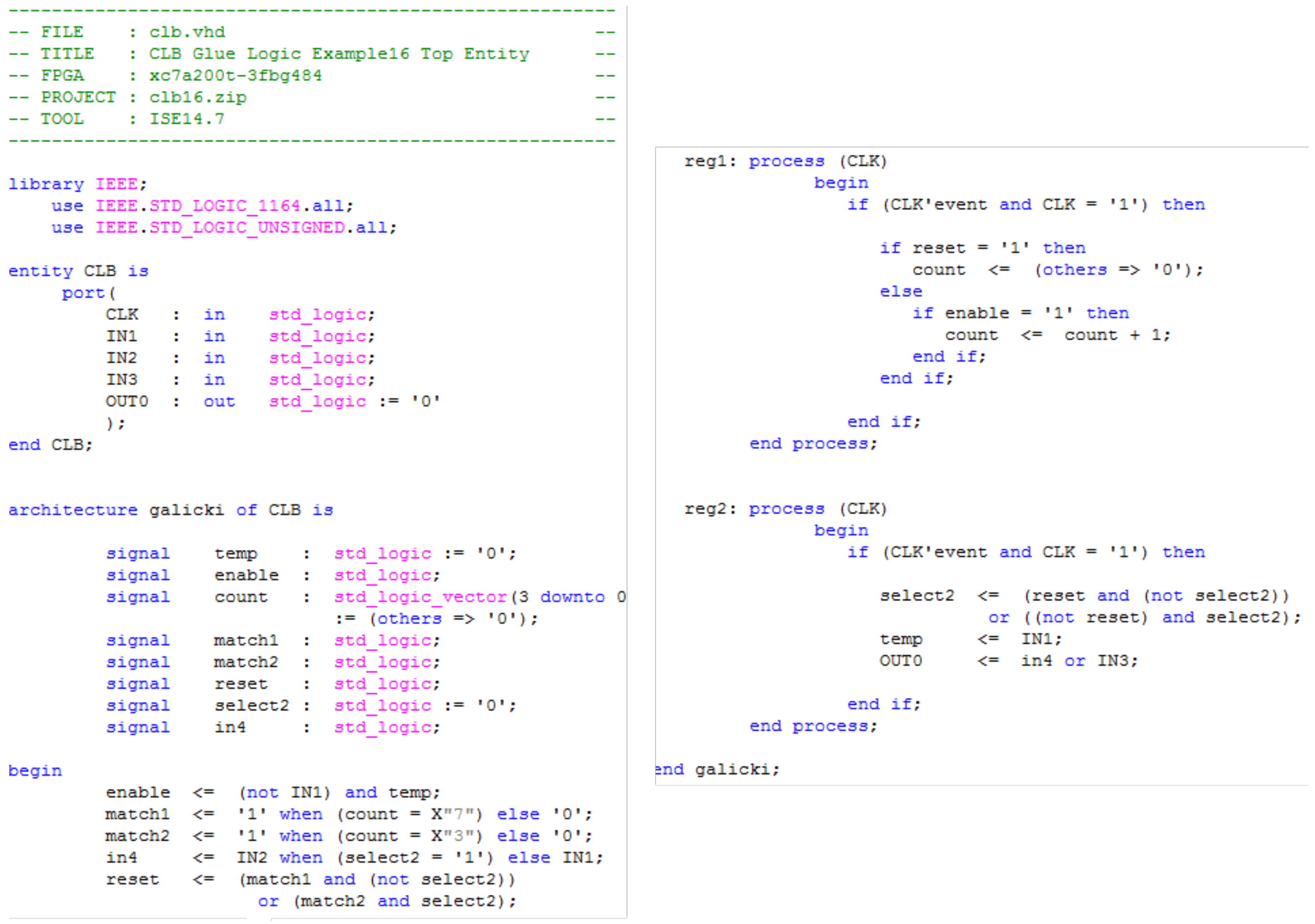
1. OUT0 becomes IN1
2. Start counting rising edges of IN1
3. After 7 pulses OUT0 switches to IN2 for 3 pulses
4. After 3 pulses OUT0 switches to IN1 for 7 pulses
5. After 7 pulses OUT0 switches to IN2 for 3 pulses
6. After 3 pulses OUT0 switches to IN1 for 7 pulses
7. Assert IN3 anytime to logic '1'
8. OUT0 becomes logic '1'

Figure 4-4. CLB Example 16 – Glue Logic Implementation Inside FPGA

4.1.2 VHDL Code for Glue Logic

Figure 4-5 shows VHDL source code that implements the function of schematic diagram of Figure 4-4. First, the entity statement identifies the three inputs and one output as well as the clock source (CLB clock). Next, the signal statements declare the internal signals. Following signal declaration, there are 5 combinatorial signal assignment statements (instantaneous action, no clock delay).

After that, it shows two registered processes for clocked outputs. The first one, reg1, updates the 32-bit Counter at each rising edge of CLB clock when ENABLE is high, but recycles it back to 0 whenever RESET becomes high. The second registered process, reg2, updates the three remaining registers at each rising edge of CLB clock. These include TEMP register used in the Falling Edge Detection circuit, SELECT2 register used in the Simple State Machine, and the Output 0 register used to latch the output.



```
--------------------------------------------------------
-- FILE    : clb.vhd                                  --
-- TITLE   : CLB Glue Logic Example16 Top Entity      --
-- FPGA    : xc7a200t-3fbg484                         --
-- PROJECT : clb16.zip                                --
-- TOOL    : ISE14.7                                  --
--------------------------------------------------------

library IEEE;
    use IEEE.STD_LOGIC_1164.all;
    use IEEE.STD_LOGIC_UNSIGNED.all;

entity CLB is
     port(
         CLK    :  in    std_logic;
         IN1    :  in    std_logic;
         IN2    :  in    std_logic;
         IN3    :  in    std_logic;
         OUT0   :  out   std_logic := '0'
         );
end CLB;


architecture galicki of CLB is

         signal    temp    :  std_logic := '0';
         signal    enable  :  std_logic;
         signal    count   :  std_logic_vector(3 downto 0
                              := (others => '0');
         signal    match1  :  std_logic;
         signal    match2  :  std_logic;
         signal    reset   :  std_logic;
         signal    select2 :  std_logic := '0';
         signal    in4     :  std_logic;

begin
         enable   <=  (not IN1) and temp;
         match1   <=  '1' when (count = X"7") else '0';
         match2   <=  '1' when (count = X"3") else '0';
         in4      <=  IN2 when (select2 = '1') else IN1;
         reset    <=  (match1 and (not select2))
                       or (match2 and select2);

  reg1: process (CLK)
            begin
               if (CLK'event and CLK = '1') then

                  if reset = '1' then
                     count  <=  (others => '0');
                  else
                     if enable = '1' then
                        count  <=  count + 1;
                     end if;
                  end if;

               end if;
        end process;


  reg2: process (CLK)
            begin
               if (CLK'event and CLK = '1') then

                  select2  <=  (reset and (not select2))
                           or ((not reset) and select2);
                  temp     <=  IN1;
                  OUT0     <=  in4 or IN3;

               end if;
        end process;

end galicki;
```

Figure 4-5. CLB Example 16 – VHDL Source Code for FPGA Glue Logic

4.1.3 VHDL Code for Test Inputs

Figure 4-6 shows the simulation test bench for the FPGA GLUE LOGIC code above. This VHDL test bench defines the clock frequency and the state of the inputs that are needed by the VHDL simulator to test the GLUE LOGIC function. Looking at the test bench, the initial statements declare the clock, three inputs and one output. Next, these are matched with the corresponding signals of the GLUE LOGIC entity that is instantiated from within the test bench entity. This is followed by three processes defining the clock and the three inputs. The clock frequency is set at 100 MHz and Input 2 is set to a constant logic 0. The remaining two inputs, Input 1 and Input 3 simulate the outputs of PWM generators with a high frequency square wave for Input 1 (320 ns period) and a low frequency square wave for Input 3 (14.72 µs period). Only one pulse is defined for the Input 3.

```
---------------------------------------------------------------
-- FILE    : tb1.vhd                                          --
-- TITLE   : CLB Glue Logic Example16 Simulation Test Bench --
-- FPGA    : xc7a200t-3fbg484                                 --
-- PROJECT : clb16.zip                                        --
-- TOOL    : ISE14.7                                          --
---------------------------------------------------------------

library IEEE;
    use IEEE.STD_LOGIC_1164.all;
    use IEEE.NUMERIC_STD.all;

entity TB1 is
end TB1;

architecture galicki of TB1 is

            signal   clk   :  std_logic;
            signal   in1   :  std_logic;
            signal   in2   :  std_logic;
            signal   in3   :  std_logic;
            signal   out0  :  std_logic;


component   CLB
       port(
            CLK     :  in    std_logic;
            IN1     :  in    std_logic;
            IN2     :  in    std_logic;
            IN3     :  in    std_logic;
            OUT0    :  out   std_logic
            );
end component;
```

```
begin

clb16:                           CLB
    port map(
            CLK    =>   clk,
            IN1    =>   in1,
            IN2    =>   in2,
            IN3    =>   in3,
            OUT0   =>   out0
            );

clock: process
              begin
                    wait for 0 ns;
                    clk  <=  '0';
                    loop
                       wait for 5 ns; -- 100 MHz
                       clk  <=  not clk;
                    end loop;
         end process;


inp1: process
              begin
                    wait for 0 ns;
                    in1  <=  '0';
                    loop
                       wait for 160 ns;
                       in1  <=  not in1;
                    end loop;
         end process;

                    in2  <=  '0'; -- inp2

inp3:    process
              begin
                    in3  <= '0'; wait for 7360 ns;
                    in3  <= '1'; wait;
         end process;

end;
```

Figure 4-6. CLB Example 16 – VHDL Source Code for Glue Logic Inputs

4.1.4 FPGA Glue Logic Simulation Waveforms

Figure 4-7 shows the simulation results of the GLUE LOGIC function as exercised by the simulation test bench. In addition to the clock, output and the three inputs, the waveforms also show key internal signals: ENABLE, RESET and SELECT 2. Looking at the Output 0 signal, it is evident that it reflects the intended function programmed in the VHDL source above – becoming Input 1 for seven Input 1 cycles, then changing to Input 2 (logic 0) for three Input 1 cycles, with this pattern continuing until Input 1 becomes logic 1 at which point the Output 0 is forced to also logic 1.

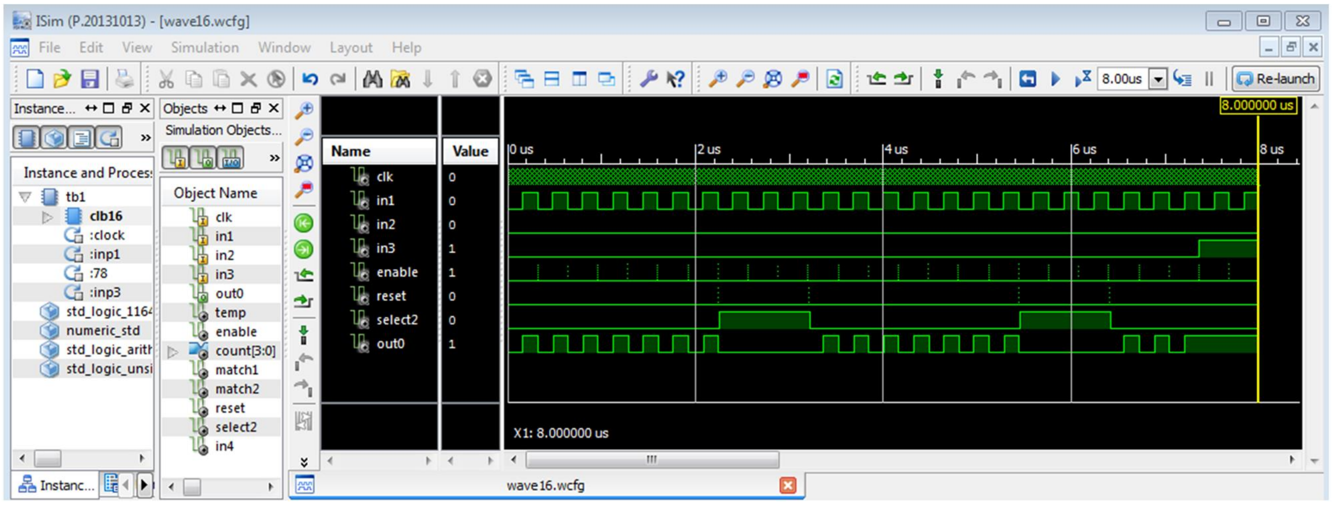


Figure 4-7. CLB Example 16 – VHDL Simulator Waveforms

4.2 FPGA to CLB Translation Process

Custom Logic in Example 16 has two major elements: two PWM Generators and a GLUE LOGIC block that combines the resulting PWM outputs with a static (logic 0) signal to produce a single output. The process of transferring these elements from an FPGA platform to C2000 microcontroller is described below.

4.2.1 Map PWM Generators to EPWM Peripherals

C2000 microcontrollers feature programmable Control Peripherals including multiple instances of Enhanced PWM peripheral (EPWM). EPWM1 and EPWM2 have been chosen to replace the two PWM Generators used in the FPGA instantiation of this example. For code initializing the EPWM modules to generate the corresponding output square waves that match the two FPGA PWM Generators, see the Example 16 project files inside C2000Ware.

4.2.2 Map Glue Logic From VHDL to CLB

Figure 2-11 shows the building blocks comprising CLB Tiles include three Counters, three LUT4 blocks (4-input Look-up Tables), three FSM blocks (Finite State Machine) and eight Output LUT3 blocks. These blocks are programmable through the GUI-based SysConfig tool, which generates code to configure corresponding Tile Configuration Registers based on user's interaction with the GUI. The process of converting HDL code to CLB consists of four steps:

- Using Input Selection function calls to select up to eight inputs to each CLB Tile from Global and Local Signal Buses (or GPREG Register), and using Peripheral Signal Mux function calls to specify whether each of eight possible outputs replaces a signal inside a selected Control Peripheral or not
- Drawing a functional block diagram of HDL code (as shown in Figure 4-4)
- Subdividing this block diagram into a collection smaller sub-sections best suited for mapping into specific CLB building blocks (as shown in Figure 4-8)
- Using the SysConfig GUI to individually implement the logic for each one of the resulting sub-sections.

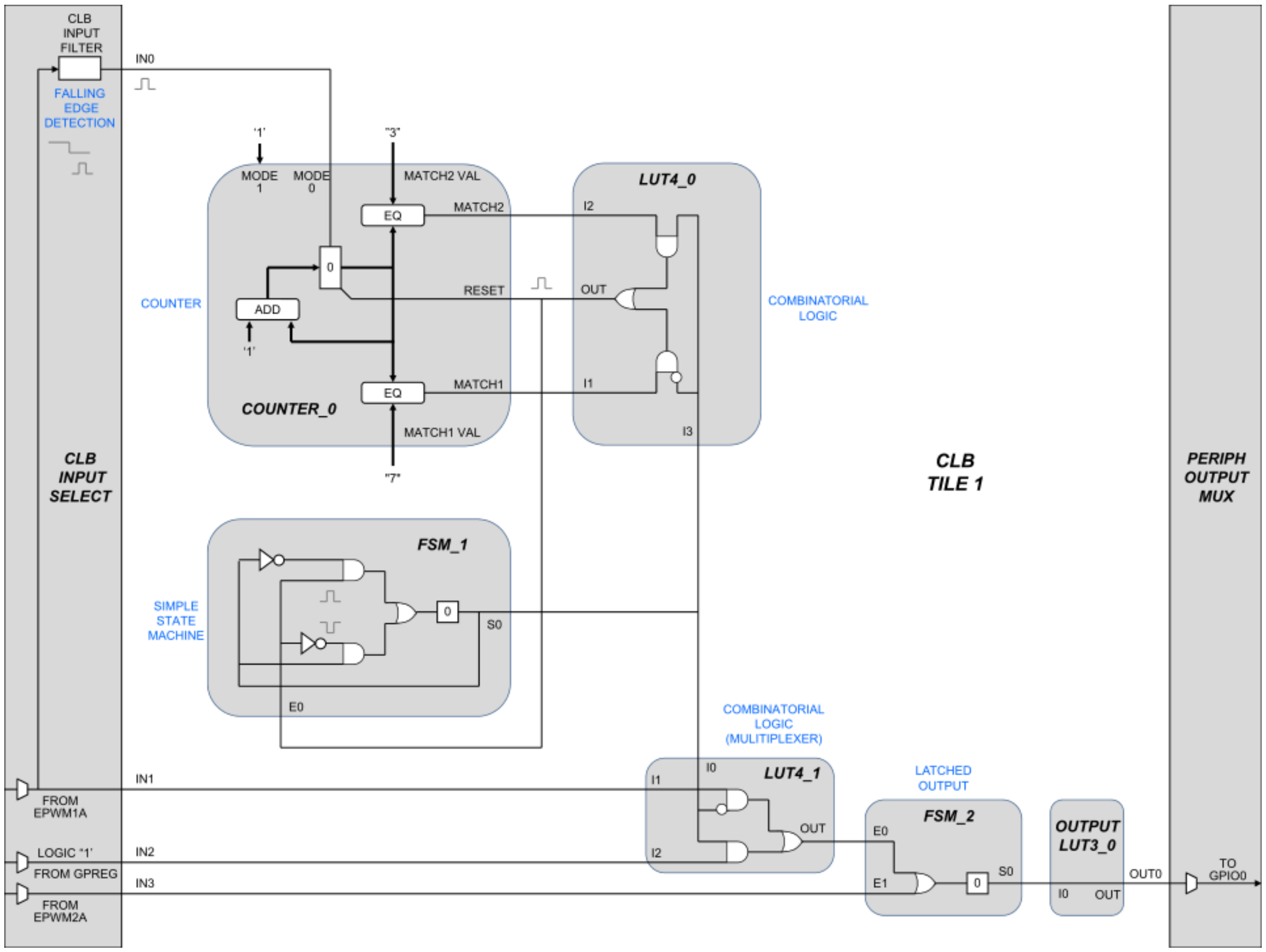


Figure 4-8. CLB Example 16 – Logic Allocation for CLB1

4.2.2.1 Inputs

Just a reminder that the EPWM1 peripheral has two uses in this example: to supply Input 1 signal to CLB1 (to compute the PWM waveform) and also to inject the Output 0 from CLB1 at the last stage of EPWM1 to forward it to GPIO Mux and subsequently to GPIO0 pin using internal signals normally reserved for EPWM1A output (see Figure 3-1).

Figure 2-7 shows that the Input Mux1 has embedded Synchronizer and Filter circuits to, respectively, synchronize the input with CLB clock and to shape the input signal into a pulse on each low-high or high-low transition. This example uses the Filter to generate a single-clock high pulse for each falling edge of Signal1 to enable the Counter counting Signal1 pulses (see Figure 4-4 and Figure 4-5).

Each multiplexer stage of the chain of muxes in Figure 2-7 requires a dedicated function call (including Local Signal function calls even though in this example no Local Signals are used for inputs). The two EPWM inputs for this example (Input 1 and Input 3) arriving at Tile 1 by the Global Signal bus are selected with global input function calls. Input 2 comes from the GPREG Register and is selected with the GPREG input function call.

4.2.2.2 Logic Allocation

With the inputs taken care of, let's now focus on what goes on inside Tile 1 that has been selected to implement the GLUE LOGIC function. Again, the goal here is to shape the contiguous block diagram from Figure 4-4 into a functionally equivalent collection of smaller blocks best suited for mapping into individual CLB logic primitives.

The resulting block diagram is shown is Figure 4-8. Here are the steps that were used to transform Figure 4-4 into Figure 4-8:

- First of all, the Falling Edge Detection has already been implemented inside the Input Selector as described above (so it does not have to be implemented inside the Tile). Next, the counter function was isolated to be implemented inside COUNTER_0 block. Following that, the remaining logic was divided into two sections with registers, and two sections without registers.
- The two sections with registers, representing VHDL registered processes, have been isolated by function and mapped into FSM_1 and FSM_2 blocks. Only the first one of these two is a Finite State Machine where the output at a given clock edge becomes an input on the next clock edge. The second one is just a registered output without any feedback that was mapped into an FSM block because this is the simplest CLB primitive that has a built-in register.
- The two combinatorial logic sections without registers have been isolated by function and mapped into LUT4_0 block and the LUT4_1 block. The LUT4_0 block generates the RESET pulse that resets the Counter to zero and flips the FSM_1 state from 0 to 1 or from 1 to 0. The LUT4_1 block implements the multiplexer seen at the bottom of Figure 4-4 to select Input 1 or Input 2 signal based on current state S0.

4.2.2.3 Output

Finally, OUTPUT LUT3_0 is used to send the resulting output to the mux embedded at the last stage of EPWM1 to continue on through GPIO Mux to drive the GPIO0 pin normally assigned to EPWM1A. As a side note, there are eight OUTPUT LUT3 primitives inside each Tile (see Figure 2-11). Each one can drive its output as a replacement signal into one of 14 pre-selected stages of Control Peripherals to replace the original signal that normally exists at that stage (see Figure 2-9). The decision on whether a given output signal replaces the original signal, or not, is provided by PERIPH SIGNAL MUX function calls as described in step1 at the beginning of this section.

4.3 Resulting C2000 Design

The CLB design for Example 16 now has all the components configured to mimic the original FPGA design inside a C2000 microcontroller. This includes two EPWM peripherals, CLB function calls to configure the Input Selection and Peripheral Signal Mux, plus CLB Tile configuration code generated by SysConfig. At this stage the design is finished. Next, take a look at the resulting signal connections inside CLB Tile 1 and verify that the design outputs match those of the FPGA design by running the CLB Simulator and external Logic Analyzer.

4.3.1 Signal Connectivity

One of the tools provided with C2000Ware is a graphical Tile Viewer showing all CLB components and the associated signal interconnections for a given CLB design. This tool provides a quick check that all the expected CLB components are engaged and properly connected. In addition to identifying the active components and connections, this viewer also shows the input signals, output signals and the logic equations inside the FSM and LUT components. Note that the current version of this viewer is limited to showing the internals of CLB Tiles (it does not currently support the Input Signal Selections to the Tiles or the Peripheral Signal Mux assignments for the Tile outputs). Figure 4-9 shows a screen shot from this viewer as applied to CLB Tile 1 of Example 16. Looking at the Tile 1 components, you see the expected two FSM blocks, two LUT blocks and the Counter, with the FSM and LUT blocks showing the internal logic equations between the inputs and outputs. Taking a closer look at the logic equations, you will see that they match the schematic of Figure 4-8. Also evident are the four input signals entering Tile 1 and one output signal exiting from Tile 1.

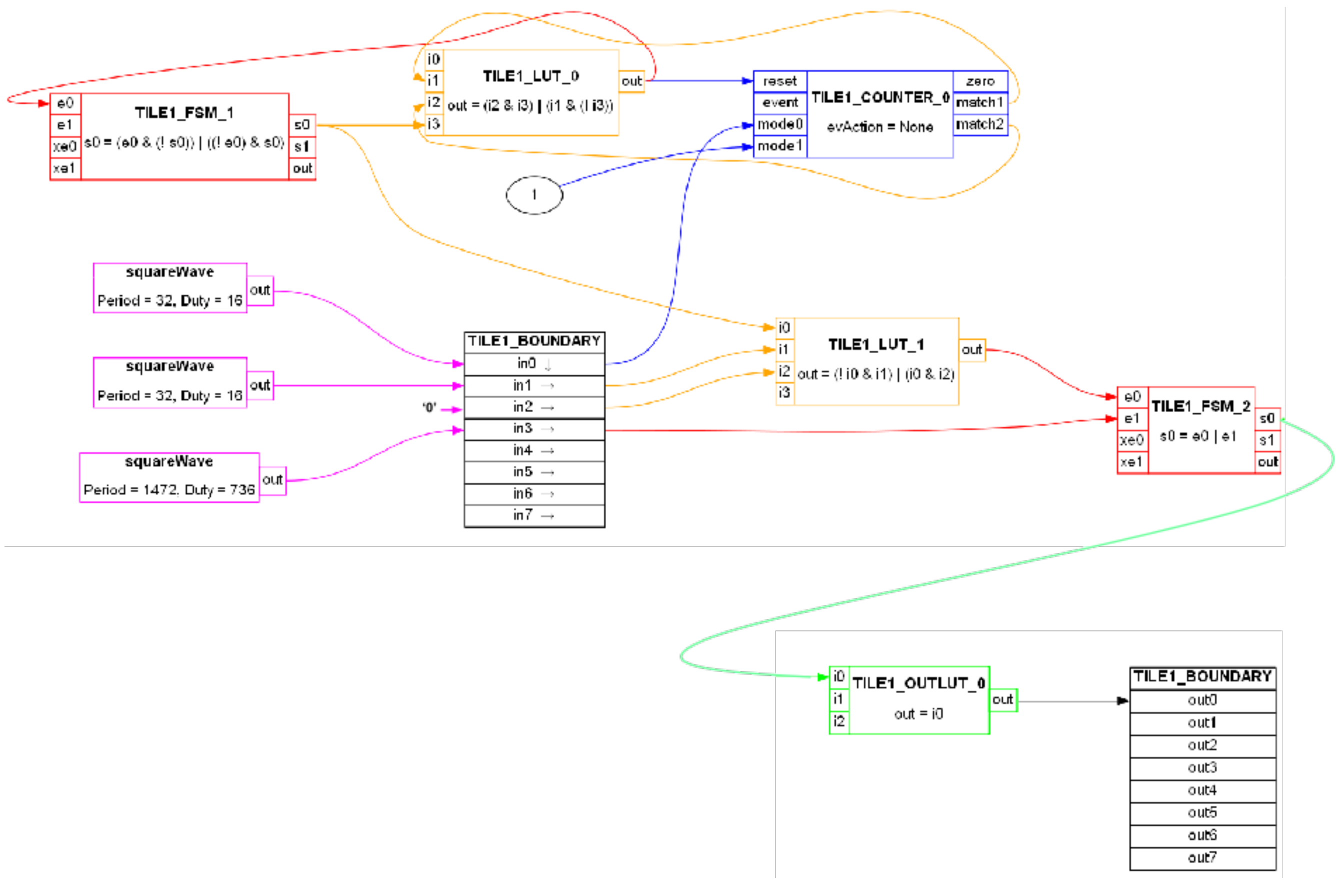
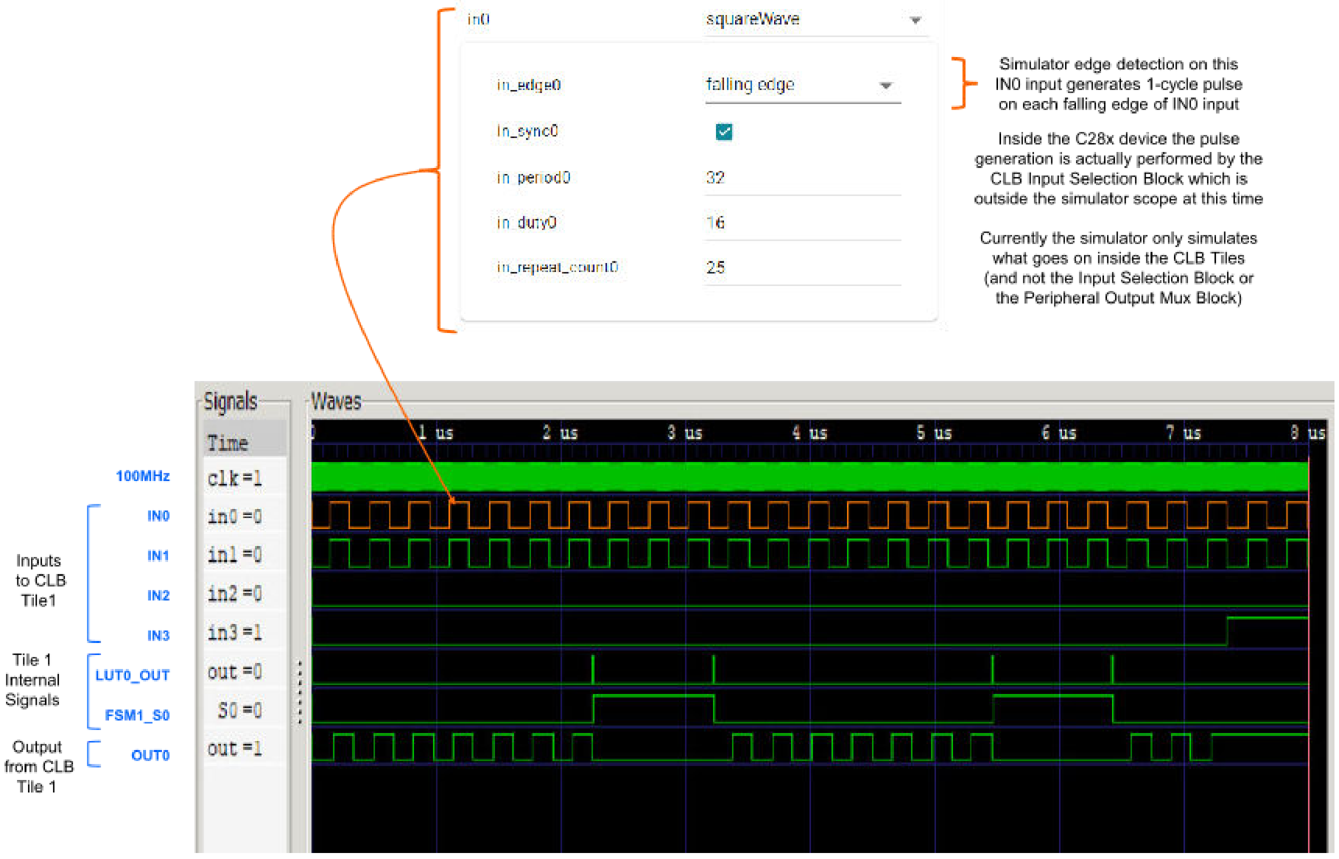


Figure 4-9. CLB Example 16 – Visualizing Signal Connectivity Inside Tile 1

4.3.2 Simulation Waveforms

The set of CLB tools provided with C2000Ware also includes the CLB Simulator. As is the case with the Tile Viewer, the current version of this tool simulates the Tile, but not the Input Selection or Peripheral Signal Muxing for the outputs. To make up for that, the Simulator has means to supply commonly used inputs such as constants, waveforms and pulses. These are defined through the GUI of the SysConfig tool via drop-down menu selections (same method that is used to configure the Tile logic primitives). Figure 4-10 shows a screenshot from the CLB Simulator for CLB Example 16. Examining the waveforms, you can see the four inputs and the output, plus a few of the key internal signals. Comparing these waveforms to those of the FPGA simulator (Figure 4-7) you can see a perfect match, indicating that the C2000 version of Example 16 indeed matches the original FPGA design. For function calls and SysConfig code used above, see the C2000Ware Example 16 project files.



in0	squareWave
in_edge0	falling edge
in_sync0	✓
in_period0	32
in_duty0	16
in_repeat_count0	25

Simulator edge detection on this IN0 input generates 1-cycle pulse on each falling edge of IN0 input

Inside the C28x device the pulse generation is actually performed by the CLB Input Selection Block which is outside the simulator scope at this time

Currently the simulator only simulates what goes on inside the CLB Tiles (and not the Input Selection Block or the Peripheral Output Mux Block)

Figure 4-10. CLB Example 16 – CLB Simulator Waveforms

4.3.3 ControlCard, LaunchPad Waveforms

Once it has been confirmed by simulation that the custom logic implemented inside the C2000 CLB matches that of the original FPGA implementation, it is time to validate the results using actual C2000 hardware. Figure 4-11 contains the screenshot from a Logic Analyzer connected to GPIO_0, GPIO_1 and GPIO_2 of F28004x ControlCard (same waveforms have also been captured using the F28004x LaunchPad). GPIO_0 represents the CLB output that has replaced the original PWM1A output as instructed by the function call that configured the Peripheral Signal Mux of Tile 1. While the original PWM1A signal has been replaced with the CLB1 output, the same function call specified that the other 7 outputs of CLB1 were not to replace their corresponding internal Control Peripheral signals, thus the original PWM1B output can be seen on GPIO_1 pin and the original PWM2A output is visible on GPIO_2.

This completes the transfer of custom logic from an external FPGA into C2000 as prescribed by Example 16. The results confirm that it is possible to transfer complex functions such as PWM generators as well as custom glue logic from external Programmable Logic Devices into C2000 microcontrollers with identical results. This capability reduces the total system cost while enhancing functionality by enabling custom logic to access internal signals of C2000 CPU and peripherals in ways that external programmable logic devices cannot.

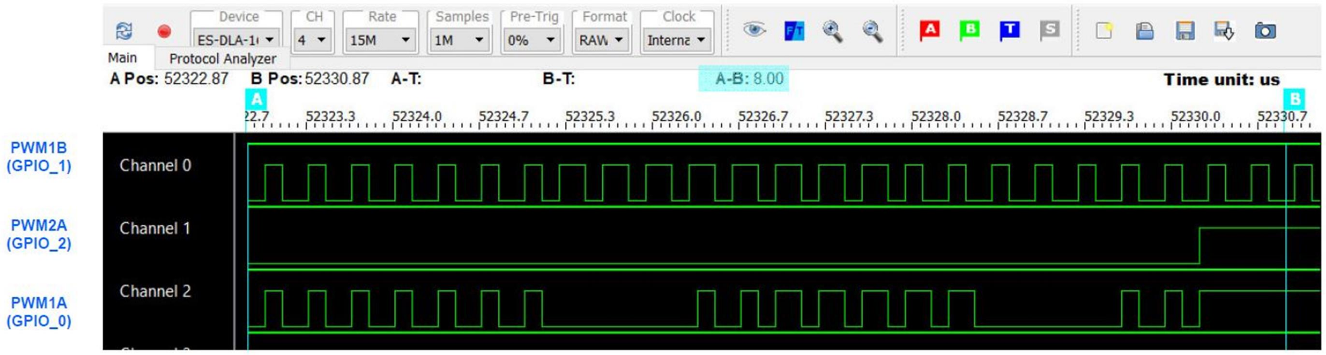


Figure 4-11. CLB Example 16 – C2000 LaunchPad/ControlCard Waveforms

5 References

For more information about CLB modules within specific C2000 devices, see the device-specific C2000 data sheets and the technical reference manual (TRM).

- Texas Instruments: *TMS320F28004x Piccolo™ Microcontrollers Data Sheet*
- Texas Instruments: *TMS320F28004x Technical Reference Manual*
- Texas Instruments: *TMS320F2837xD Dual-Core Delfino™ Microcontrollers Data Sheet*
- Texas Instruments: *TMS320F2837xD Dual-Core Microcontrollers Technical Reference Manual*
- Texas Instruments: *TMS320F2838x Microcontrollers With Connectivity Manager Data Sheet*
- Texas Instruments: *TMS320F2838x Microcontrollers Technical Reference Manual*
- Texas Instruments: *CLB Tool User's Guide*
- Texas Instruments: *TMSF28078x Microcontrollers Data Sheet*

6 Revision History

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

Changes from September 28, 2019 to July 30, 2020 (from Revision * (September 2019) to Revision A (July 2020)) **Page**

- Updates were made in Section 2.2...3
- Updated the numbering format for tables, figures and cross-references throughout the document..........3
- Updates were made in Section 2.2...5
- Updates were made in Section 2.3 7
- Updates were made in Section 2.3.3 13
- Updates were made in Section 2.3.3.2 16
- Update was made in Section 4 30
- Update was made in Section 4.2.2.1 37
- Update was made in Section 4.2.2.3 38
- Update was made in Section 4.3.3 41