XAPP1230 (v1.1) November 20, 2015

Configuration Readback Capture in UltraScale FPGAs

Author: Stephanie Tapp

Summary

This document describes the process to readback capture the user state of the internal configurable logic block (CLB) registers of UltraScale™ FPGAs using the Vivado® Design Suite and the UltraScale FPGA JTAG interface. The steps to store the readback capture data in an ASCII file (`.rdbk`) and the steps to identify the design elements in this file using the design logic location file (`.ll`) are provided (Figure 1). Basic knowledge of FPGA configuration described in the *UltraScale Architecture Configuration User Guide* (UG570) [Ref 1] and general usage of the Vivado Design Suite are assumed.

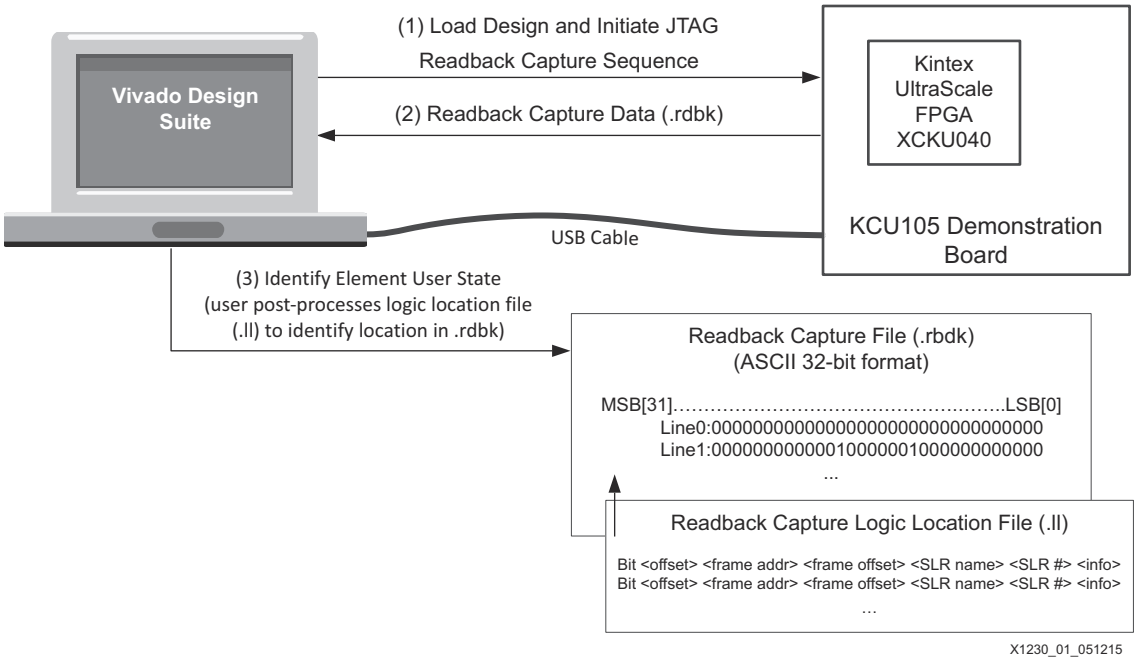

Figure 1: **UltraScale FPGA Readback Capture Flow via JTAG Interface**

You can download the reference design files for this application note from the Xilinx® website. For detailed information about the design files, see Reference Design.

Introduction

UltraScale FPGAs are configured after power-up to define the programmable aspects of the user design that include portions of the input/output blocks (IOBs), CLBs, distributed RAM, shift register logic (SRL), the initial content of block RAM, interconnect routing between clock resources, logic resources, and I/Os. Instructions for the configuration logic and the configuration information are combined to form a bitstream used to configure the UltraScale FPGA. After the FPGA is configured with the user bitstream, the data in internal configuration memory can be read out using a process called readback.

To perform readback, a sequence of commands must be sent to the device. After readback is initiated, the UltraScale FPGA sends the contents of its configuration memory to a supported interface. Readback can be performed from one of three interfaces on UltraScale FPGAs: the internal configuration access port (ICAP), SelectMAP, or JTAG interface.

UltraScale FPGAs support readback verify and readback capture. Readback verify is a comparison that confirms the configuration logic was programmed and remains as the user intended. Readback capture is used to determine the content of user state elements, which by design change during operation.

Readback verify compares the user design bitstream against the readback data using the design's generated mask file (`.msk`/`.msd`). The mask file determines which components have dynamically changing values in the user's design and ignores them during the comparison. Examples of such components are CLBs and look-up tables (LUTs) that have been configured as distributed RAM. The Vivado Design Suite hardware manager supports the verify operation via the JTAG interface as a method to confirm that the user design was correctly programmed into the FPGA.

This application note focuses on readback capture. Readback capture uses the same process as readback verify but requires additional commands to be issued during the readback sequence to read the user state of the internal CLB registers. Readback capture is an identification method in which the unique design readback capture results for CLB registers, block RAM, or LUTs used as distributed RAM or SRLs are mapped by the user design's `.ll` file. The design elements supported by readback capture have unique frame types (Type0, Type1), as shown in Figure 2. The Vivado Design Suite 2015.1 does not support readback capture directly, but a reference script is supplied with this application note that can be run in the Vivado Design Suite Tcl console to readback capture user state data and format it in a readable ASCII file (`.rdbk`). Refer to Implementation for details.

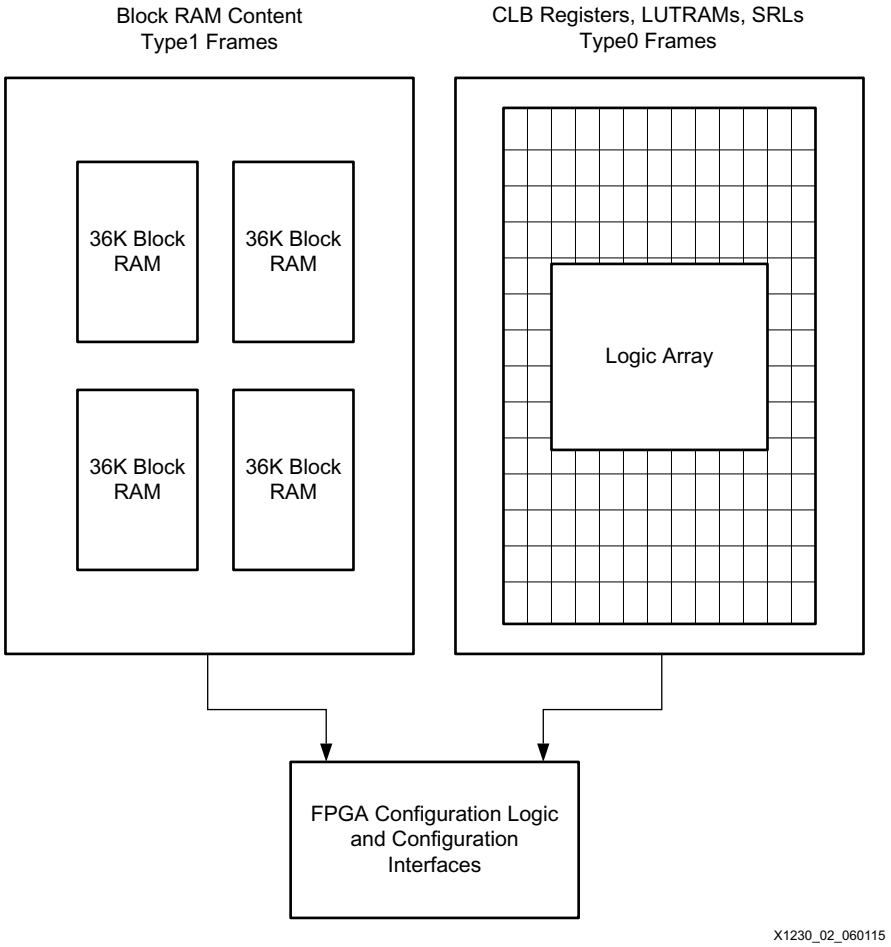

Figure 2: **UltraScale FPGA Design Elements Supported by Readback Capture**

When debugging a new design, the Vivado logic analyzer is another powerful tool and ideal for many applications. This tool should be considered if added debug IP logic is not a concern for timing/resources and if limited trace (block RAM resources needed to increase the depth) is acceptable. In hardware ASIC emulation or co-simulation where a large number of signals need to be analyzed and clock control is available, the readback capture feature is a superior option because it does not require additional logic to be added into the user design and allows for easy access to observe the design state.

Differences from 7 Series FPGAs

The UltraScale FPGAs do not have dedicated capture memory cells, so the CAPTUREE2 primitive and GCAPTURE commands available in 7 series FPGAs are no longer required or supported in UltraScale FPGAs.

For the 7 series FPGA readback capture sequence, the GCAPTURE command was required before reading back the user state contents. In the UltraScale FPGAs, a write to the mask (MSK) register and control 1 (CTL1) register (CAPTURE bit[23]) are required to enable the readback capture of the CLB registers. The JTAG readback capture command sequence in UltraScale FPGAs is described in Table 5.

When performing a readback capture on the 7 series FPGAs, Xilinx recommends stopping the clock of the associated user logic while the GCAPTURE command is being loaded to ensure the current state of the device is readback capture. In the UltraScale FPGAs, you must stop or disable the clock associated with the user state elements being targeted throughout the duration of the readback capture sequence.

The readback capture value difference between 7 series and UltraScale FPGAs is shown in Table 1.

Table 1: **Readback Capture Inversion Summary**

Design Component	Readback Capture Data Inversion in UltraScale FPGAs	Readback Capture Data Inversion in 7 Series FPGAs
CLB registers	Yes. Value captured as `0` when `1`.	Yes. Value captured as `0` when `1`.
Block RAM registers	No. Value captured as `0` when `0`.	Yes. Value captured as `0` when `1`.
Distributed RAM or SRLs	No. Value captured as `0` when `0`.	No. Value captured as `0` when `0`.

Readback Capture Flow Overview

An overview of the readback capture flow is shown in Figure 3. After the readback capture is initiated and the data is read back, the `.ll` file provides a bitmap to identify the design elements of interest. Details on how to generate the files and perform the identification are described in Implementation.

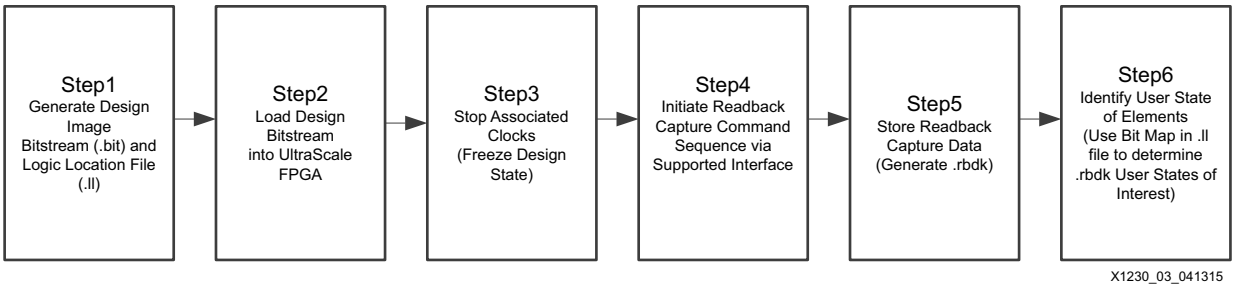

Figure 3: **Readback Capture Flow Overview**

Implementation

The following sections provide instructions for the development board setup, file generation software steps, the readback capture command sequence to read the UltraScale FPGA CLB register user state through the JTAG interface, and Tcl instructions to run the flow on the KCU105 development board.

Demonstration Setup

This readback capture example flow is demonstrated with the following:

- Xilinx KCU105 development board with the Kintex UltraScale FPGA XCKU040 and USB cable
- Vivado Design Suite 2015.1
- Application reference Tcl script (`readback_capture.tcl`)

Before performing the readback capture on the reference design included in this application note, the board setup shown in Figure 4 should be used.

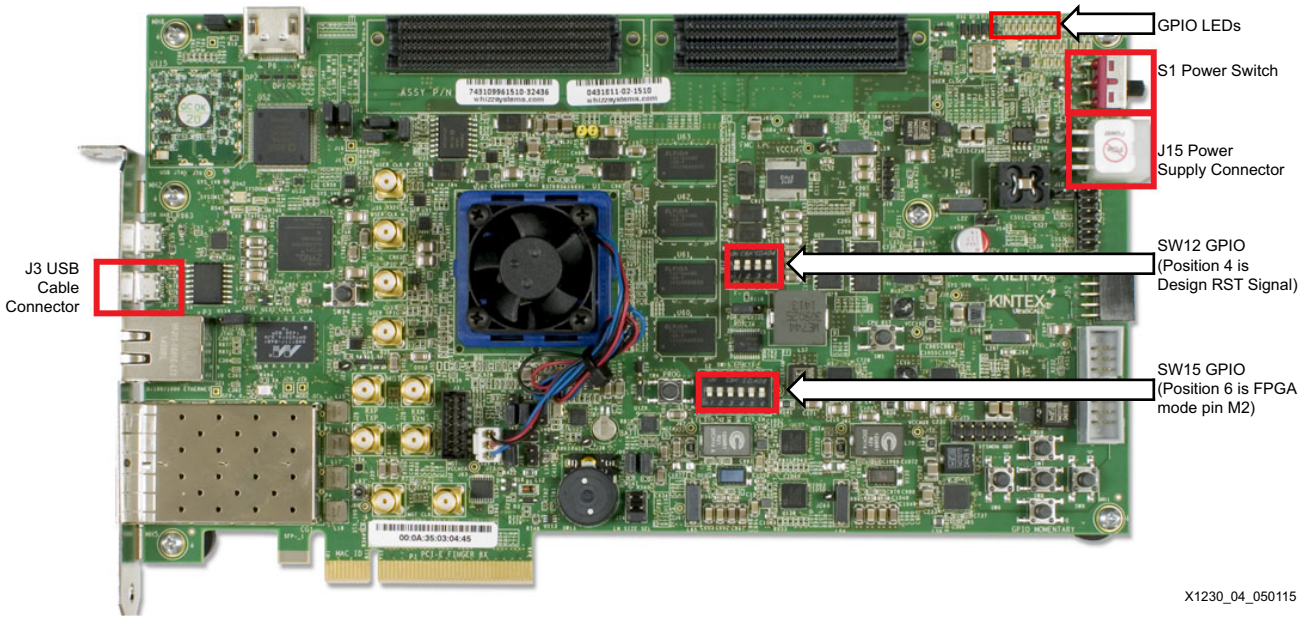

Figure 4: **KCU105 Development Board Setup**

1. Connect one end of the Xilinx platform cable USB II to the computer USB port and the other end to the board USB cable connector J3.
2. Set the general-purpose I/O (GPIO) switch SW12 to `1111` as shown in Figure 4.

   Switch SW12 position 4 is the RST reference design signal and can be used to reset the reference design count. By default, this should be off and set as shown in Figure 4.
3. Set the switch SW15 to `000001` as shown in Figure 4.

FPGA mode pin M2 is position 6, so with this setting the JTAG mode is used M[2:0] = 101.

4. Connect the power supply to the power supply connector J15.
5. Set the power supply switch S1 to the On position.

   Note: The GPIO LEDs display the reference design CLB register's values.

Reference Design Description

This application note includes a simple 8-bit counter reference design to demonstrate how to use the logic location file to identify the user state of the design's eight CLB registers. In this reference design, an 8-bit counter is implemented with CLB registers, and the value is displayed on the eight GPIO LEDs shown in Figure 4.

The reference design 8-bit counter clock is enabled if these conditions are met:

- UltraScale FPGA JTAG USER4 instruction is active
- UltraScale FPGA JTAG TAP is in the RUNTEST with TMS Low

This design allows the user to specify the value of the count by sending two commands (scan_ir_hw_jtag and runtest_hw_jtag described in Enabling the Count and Stopping the Clock) with the Vivado Design Suite via the USB cable. The user issues the scan_ir_hw_jtag command first to activate the USER4 JTAG instruction and then the runtest_hw_jtag command is issued to start the counter. The number of TCKs issued by the RUNTEST command equals the count value that is stored in the CLB registers. For example, if you wanted LED[7:0] = 10101110 captured, this would be reflected in the CLB registers qout_reg[7:0] by selecting the decimal equivalent of 174 for the TCK count. The RST signal can reset the count value when SW12 position4 is set to 0 and a runtest_hw_jtag -tck 1 command is issued.

You must stop the clocks with the associated design elements that the readback capture is to be performed on. In the counter reference design, when the JTAG USER4 instruction is displaced by another JTAG command or the RUNTEST conditions are no longer met, the internal TCK to the counter is disabled. After the reference design internal TCK is disabled, the readback capture can be performed. The reference design is shown in Figure 5.

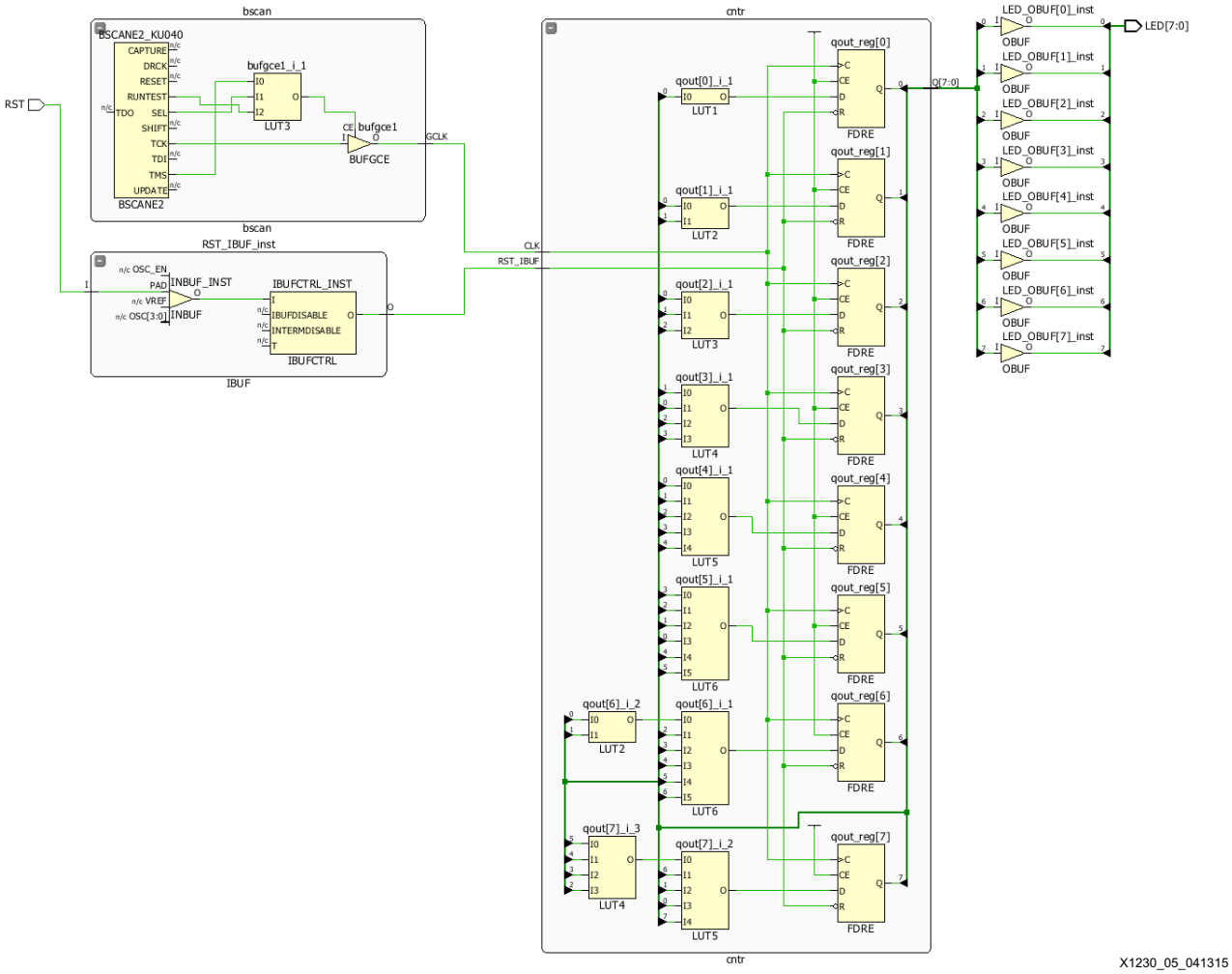

Figure 5: **8-bit Counter Reference Design**

The reference design file that accompanies this application note has the following directory structure that is also referenced in the demonstration flows:

- Design:
  - Design source files are not required to run the demonstration, but they are provided for reference.
  - Verilog files: `LED_Count.v`, `bscan.v`, and `cntr.v`.
  - Constraints file: `LED_Count.xdc`.
- DemoFiles:
  - Bitstream and logic location files created from the design source files are used to demonstrate the readback capture flow on the KCU105 evaluation board in this application note.
  - Bitstream file: `LED_Count.bit` (or `LED_Count.rbt`).
  - Logic location file: `LED_Count.ll`.

- TclScript:
  - Tcl script is provided to readback capture user state data on the KCU105 evaluation board and output an ASCII readback capture file (`.rdbk`) for bit identification.
  - Script file: `readback_capture.tcl`.

Vivado Design Suite Flow

This section discusses the design entry considerations and the readback capture bitstream settings required during the program stage of the Vivado Design Suite flow. For additional details on the bitstream settings, refer to the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 2].

Design Entry

For a design in which the readback capture is performed, the user must stop the clocks with the associated logic to freeze the user state. For the example reference design, the clock used is the BSCANE2 TCK, and this clock is disabled when the USER4 is not the active JTAG instruction.

Bitstream File Generation

The write_bitstream generation properties and file settings that need to be considered are described in this section. Table 2 shows the settings for the output file formats that need to be generated during the bitstream generation for readback capture.

Table 2: **Write_bitstream File Output**

File Extension	File Type	File Description
`.bit/.rbt`	Binary/ASCII	User design bitstream.
`.ll`	ASCII	Logic location file contains location of registers, distributed RAM, SRLs, and block RAM.
`.rbd`	ASCII	(Optional) readback ASCII .rbt-like format, configuration data with padding overhead (133 words) and no configuration commands. This file can be used as a sample file with INIT values. Refer to Debug.

Table 3 describes the write_bitstream settings needed to generate the necessary readback capture files.

Table 3: **Write_Bitstream File Generation Options**

Option	Description
-verbose	Provides log file summary of options used during write_bitstream run
-raw_bitfile	Creates ASCII format user design bitstream (`.rbt`)

Table 3: **Write_Bitstream File Generation Options *(Cont'd)***

Option	Description
-readback_file	(optional) Creates ASCII format `.rbd` file with additional overhead padding to match file readback. No commands are included. The file is typically used with the `.msd` file for readback verify, but optionally can be used during debug for readback capture sequence. Refer to Debug.
-logic_location	Creates ASCII format `.ll` file for bit mapping the locations of the block RAM, distributed RAM, SRLs, and CLB registers.

The bitstream properties set must not prohibit readback. Therefore, the readback security setting set_property BITSTREAM.READBACK.SECURITY with value of Level1 (disables readback) or Level2 (disables both readback and reconfiguration) or the encryption property cannot be used.

The following command line is an example of the settings that should be included during the bitstream generation run from the user project if running from the Vivado Design Suite Tcl console:

```
write_bitstream -verbose -raw_bitfile -logic_location_file -readback_file LED_Count
```

Alternatively, if the Vivado Integrated Design Environment (IDE) is used to generate the bitstream, ensure the settings in Figure 6 are selected. During the bitstream generation, the readback_file setting is optional for debug. See Debug for details.

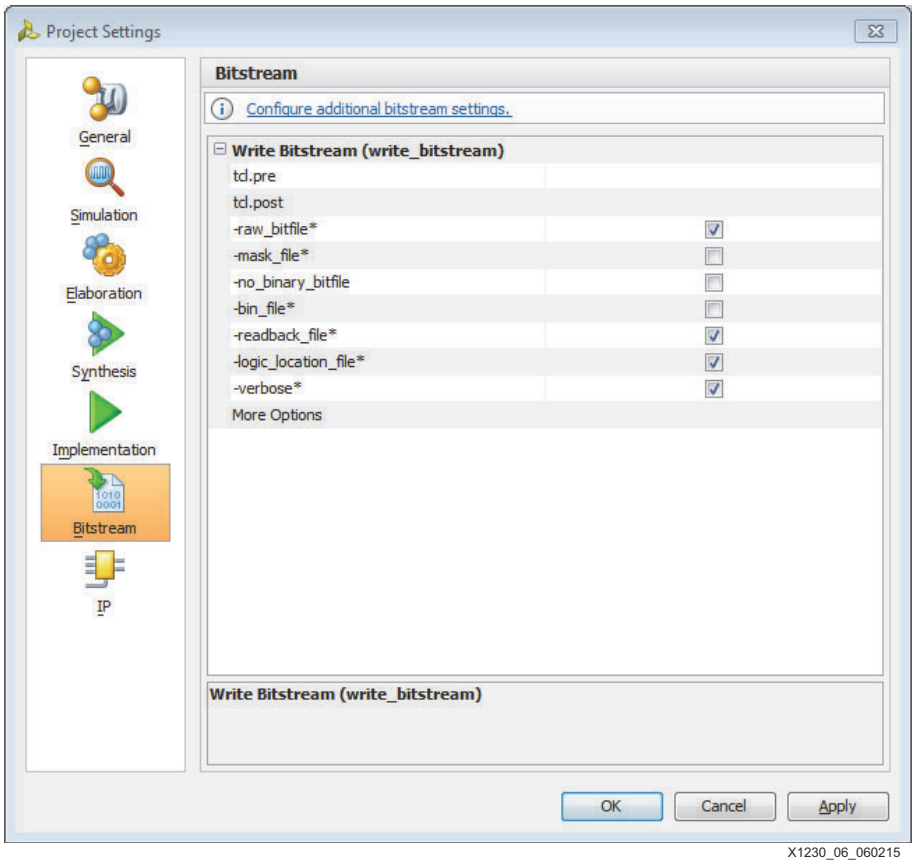

Figure 6: **Vivado Design Suite IDE Bitstream Settings**

Logic Location File

To identify the proper design element from the `.rdbk` file, the bit offset is required from the design `.ll` file. The 8-bit counter `LED_Count.ll` example file included with this application note is shown below. The CLB register's bit locations are shown in this file.

```
Revision 4
; Created by Vivado 2015.1 SW Build 1215546 at Fri May 08 13:54:31 2015
; Bit lines have the following form:
; <offset> <frame address> <frame offset> <SLR name> <SLR number> <information>
; <information> may be zero or more <key>=<value> pairs
; Block=<blockname>    specifies the block associated with this memory cell.
;
; Latch=<name>         specifies the latch associated with this memory cell.
;
; Net=<netname>        specifies the user net associated with this
;                      memory cell.
;
; Ram=<ram id>:<bit>   This is used in cases where a SLICE LUT is used
; Rom=<ram id>:<bit>   as RAM (or ROM) and for Block RAM. <Ram id> will
;                      be a letter indicating which LUT within the SLICE
;                      or a 'B' for Block RAM. For SLICE LUTs <bit> is a
;                      decimal number. For Block RAM this will be BIT or
;                      PARBIT indicating a data or parity bit and a decimal
;                      number.
;
; Info lines have the following form:
; Info <name>=<value>  specifies a bit associated with the FPGA
;                      configuration options, and the value of
;                      that bit. The names of these bits may have
;                      special meaning to software reading the .ll file.
;
Bit   30867264 0x00023204 1152 SLR0 0 Block=SLICE_X49Y78 Latch=AQ Net=cntr/Q[0]
Bit   30867280 0x00023204 1168 SLR0 0 Block=SLICE_X49Y78 Latch=AQ2 Net=cntr/Q[1]
Bit   30867268 0x00023204 1156 SLR0 0 Block=SLICE_X49Y78 Latch=BQ Net=cntr/Q[2]
Bit   30867284 0x00023204 1172 SLR0 0 Block=SLICE_X49Y78 Latch=BQ2 Net=cntr/Q[3]
Bit   30868128 0x00023204 2016 SLR0 0 Block=SLICE_X49Y90 Latch=AQ Net=cntr/Q[4]
Bit   30868144 0x00023204 2032 SLR0 0 Block=SLICE_X49Y90 Latch=AQ2 Net=cntr/Q[5]
Bit   30868132 0x00023204 2020 SLR0 0 Block=SLICE_X49Y90 Latch=BQ Net=cntr/Q[6]
Bit   30868148 0x00023204 2036 SLR0 0 Block=SLICE_X49Y90 Latch=BQ2 Net=cntr/Q[7]
```

The header of the `.ll` file explains all the fields. The file includes bit position columns and logic information. The first column indicates whether the logic is used as a register, RAM, or ROM, and the second column provides the offset. The bit offset count begins with the bit position frame address 0 of the readback data. The frame offset specifies the bit offset within the specified frame address. In the case where a net name is applicable, that name is included for design correlation purposes. In this reference design, the bit offset value is used to determine the exact bit location for a target design element user state identification.

Programming FPGA via JTAG

Before the readback capture is performed, the user design bitstream image must be loaded into the UltraScale FPGA. This can be done through JTAG or another supported configuration interface. For this demonstration, the FPGA reference design is loaded via JTAG. In the Vivado Design Edition or Vivado Lab Edition tool, open the hardware manager. In the Tcl Console

window, issue the following commands to program the FPGA with the reference design, with the user path to the design files specified.

```
>connect_hw_server
>open_hw_target [lindex [get_hw_targets -of_objects [get_hw_servers localhost]] 0]
>current_hw_device [lindex [get_hw_devices] 0]
>refresh_hw_device -update_hw_probes false [lindex [get_hw_devices] 0]
>set_property PROGRAM.FILE {C:/XAPP1230/DemoFiles/LED_Count.bit} [lindex
[get_hw_devices] 0]
>program_hw_devices [lindex [get_hw_devices] 0]
>close_hw_target [lindex [get_hw_targets -of_objects [get_hw_servers localhost]] 0]
```

Enabling the Count and Stopping the Clock

After the FPGA is configured successfully with the reference design, the count value must be selected and the associated clocks stopped to freeze the design state to be captured. In the reference design, you can select the count value you want with JTAG instructions in the Vivado Design Suite Tcl console. Refer to the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 3] for details on the command usage.

To select a specific count value on CLB registers to be displayed on the GPIO LEDs, open the Vivado hardware manager and enter the jtag_mode as shown below from the TCL console:

Setup to open the target in JTAG mode:

```
>open_hw_target -jtag_mode 1
```

Next, issue the USER4 instruction (opcode 23) with the Vivado Design Suite Tcl scan_ir_hw_jtag command and select the count value using the RUNTEST command to enable the counter. The Vivado hardware manager JTAG commands used are:

```
scan_ir_hw_jtag <ir length> -tdi <instruction>
runtest -tck <number of tcks>
```

Shift USER4 opcode in the JTAG instruction register:

```
>scan_ir_hw_jtag 6 -tdi 23
```

The RUNTEST command TCK count is displayed on the LEDs, and that value is stored in the CLB registers. In this example, a RUNTEST of 174 TCKs is selected to display a count value of the hexadecimal equivalent LED[7:0] = `10101110`:

```
>runtest_hw_jtag -tck 174
```

Replace the USER4 JTAG instruction (opcode 23) with BYPASS (opcode 3f) to ensure the clock is stopped:

```
>scan_ir_hw_jtag 6 -tdi 3f
```

JTAG Sequence for Readback Capture of CLB Registers

The process for readback capture of data from the frame data register out (FDRO) register through the JTAG interface is similar to the process for reading from other registers. Registers used during readback capture are listed in Table 4 with their 32-bit command reference. Details

of these registers can be found in the *UltraScale Architecture Configuration User Guide* (UG570) [Ref 1].

Table 4: **Sample Configuration Register Commands Used for Readback Capture**

Register Description	32-bit Configuration Commands
FDRO read packet header (type 1)	0x28006000
FDRO read packet header	0x4800CF00
No operation (NOOP) one word	0x20000000
Synchronization word	0xAA995566
MSK write packet header	0x3000C001
CTL1 write packet header	0x30030001
CMD write packet header	0x30008001
Frame address register (FAR) write packet header	0x30002001
CMD write packet data - reset cyclic redundancy check (RCRC) command	0x00000007
CMD write packet data - read configuration (RCFG) command	0x00000004
MSK write packet data (CTL1 CAPTURE bit)	0x00800000
CTL1 write packet data (CAPTURE bit enabled)	0x00800000
Type 2 FAR write packet data (NULL)	0x00000000

For most applications of readback capture the goal is to get all the user state data that can be read, so it is natural to do a readback of the complete device starting at frame address 0. The procedure for reading CLB registers starting at frame address 0 through the JTAG interface is shown in Table 5. To perform a readback capture on a CLB register, the basic steps performed are:

1. Stop clocks related to the user design state to freeze state.

   Note: Only clocks related to the user state captured must be stopped.

2. Write 1/1 to the configuration logic registers MSK[23]/CTL1[23] for enabling capture.
3. Read frame(s) starting at the FAR register (0x00000000) with the CTL1 CAPTURE bit set to 1.
4. Write 1/0 to the configuration logic registers MSK[23]/CTL1[23] to reset back to default after a readback capture is performed.
5. Unfreeze the associated user design state by enabling clocks.

Table 5: **Sample JTAG Sequence to Readback Capture All Frames in XCKU040**

Step	Description	Step details	TDI	TMS	# of TCK
1	Reset TAP controller	Clock five 1s on TMS to bring the device to the Test-Logic-Reset (TLR) state	X	1	5
		Move into the Run-Test/Idle (RTI) state	X	0	1
		Move into the Select-IR state	X	1	2
		Move into the Shift-IR state	X	0	2
2	Shift in CFG_IN	Shift the first five bits of the CFG_IN instruction, LSB first	00101 (CFG_IN)	0	5
		Shift the MSB of the CFG_IN instruction while exiting Shift-IR	0	1	1
		Move into the Select-DR state	X	1	2
		Move into the Shift-DR state	X	0	2

Table 5: **Sample JTAG Sequence to Readback Capture All Frames in XCKU040 *(Cont'd)***

Step	Description	Step details	TDI	TMS	# of TCK
3	Shift packets in CFG_IN	Shift configuration packets into the CFG_IN data register while in Shift-DR, MSB first.	Packet data [31:0]: 0xFFFFFFFF (dummy word) 0xAA995566 (sync word) 0x20000000 (NOOP) 0x30008001 (CMD write) 0x00000000 (NULL) 0x3000C001 (MSK write) 0x00800000 (CAPTURE bit[23]) 0x30030001 (CTL1 write) 0x00800000 (CAPTURE bit[23]) 0x20000000 (NOOP) 0x20000000 (NOOP) 0x20000000 (NOOP) 0x20000000 (NOOP) 0x20000000 (NOOP) 0x20000000 (NOOP) 0x20000000 (NOOP) 0x20000000 (NOOP) 0x30002001 (FAR write) 0x00000000 (Address 0x0) 0x30008001 (CMD write) 0x00000004 (RCFG) 0x28006000 (FDRO write) 0x483D0E2B (Type 2 packet to indicate number of words to read i.e., XCKU040 32530 frames + 1 frame + 10 words) 0x20000000 (NOOP)	0	767
		Shift the LSB of the last configuration packet while exiting Shift-DR	0	1	1
		Move into the Select-IR state	X	1	3
		Move into the Shift-IR state	X	0	2
4	Shift in CFG_OUT	Shift the first five bits of the CFG_OUT instruction, LSB first.	00100 (CFG_OUT)	0	5
		Shift the MSB of the CFG_OUT instruction while exiting Shift-IR.	X	1	2
		Move into the Select-DR state	X	1	2
		Move into the Shift-DR state	X	0	2

Table 5: **Sample JTAG Sequence to Readback Capture All Frames in XCKU040 *(Cont'd)***

Step	Description	Step details	TDI	TMS	# of TCK
5	Shift frame data from FDRO	Shift the contents of the FDRO register out of the CFG_OUT data register	...	0	# of readback bits -1
		Shift the last bit of the FDRO register out of the CFG_OUT data register while exiting Shift-DR.	X	1	1
		Move into the Select-IR state	X	1	3
		Move into the Shift-IR state	X	0	2
6	Reset TAP Controller	End by placing the test access port (TAP) controller in the TLR state	X	1	3
The readback capture has successfully been completed at this step. The following steps reset the CTL1 register to the original state.					
7	Move to RTI	Move into the RTI state	X	0	
		Move into the Select-IR state	X	1	2
		Move into the Shift-IR state	X	0	2
8	Shift in CFG_IN (update capture bit)	Shift the first five bits of the CFG_IN instruction, LSB first	00101 (CFG_IN)	0	5
		Shift the MSB of the CFG_IN instruction while exiting Shift-IR	0	1	1
		Move into the Select-DR state	X	1	2
		Move into the Shift-DR state	X	0	2
9	Shift packet into CFG_IN	Shift configuration packets into the CFG_IN data register while in Shift-DR, MSB first	Packet data [31:0]: 0xFFFFFFFF (dummy word) 0xAA995566 (sync word) 0x20000000 (NOOP) 0x3000C001 (MSK write) 0x00800000 (CAPTURE bit[23]) 0x30030001 (CTL1 write) 0x00000000 (CAPTURE bit[23]) 0x20000000 (NOOP) 0x20000000 (NOOP)	0	288
		Shift the LSB of the last configuration packet while exiting Shift-DR	0	1	1
		Move into the Select-IR state	X	1	3
		Move into the Shift-IR state	X	0	2
10	Reset TAP controller	End by placing the TAP controller in the TLR state	X	1	3

There is pipeline data that needs to be considered when performing a readback capture. Configuration data coming from the FDRO register pass through a frame buffer plus ten

dummy data words as shown in Figure 9, so this number of words in the readback capture file should be discarded as the pipeline. The next sections describe how to use the `readback_capture.tcl` script to read all the XCKU040 frame data starting at FAR `0x00000000` to a `.rdbk` file and then how to use the `.ll` file <offset> field to determine the CLB registers' user states.

For applications that need faster readback capture time and might only need to read a particular CLB register, a performance optimization of the readback capture could be to a single frame. If reading back a single frame, the <frame_address> and <frame_offset> fields in the `.ll` file would be used instead of the bit <offset> field.

Readback Capture - Tcl Script Usage

Vivado Design Suite 2015.3 includes a new option that performs a readback capture on all the frames in the device and saves the output to the `<filename>.rdbk` selected. To perform a readback capture, run this command:

```
readback_hw_device -capture -readback_file c:/<path>/<filename>.rdbk
```

For Vivado Design Suite versions prior to 2015.3 or to capture and readback a single frame, the `readback_capture.tcl` script is included as a reference with this application note and can be run from the Vivado Design Suite Tcl console. The board and cable should be connected as described in Demonstration Setup, the FPGA should have been programmed with the design, and the associated clocks stopped before the readback capture is performed. The `readback_capture.tcl` file should then be sourced in the Vivado Design Suite Tcl console as shown in Figure 7. For purposes of this example, the path used is `c:/XAPP1230/TclScript/readback_capture.tcl`.

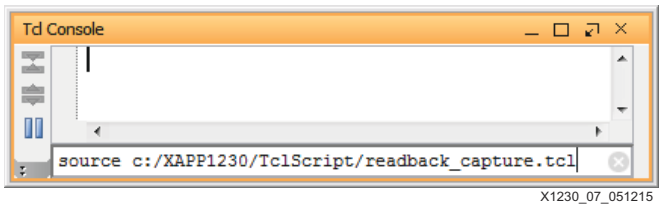

Figure 7: **Sourcing the `readback_capture.tcl` Script**

The Tcl script includes processes to initiate the readback sequence via JTAG and format the captured contents into an ASCII 32-bit format for user verification against their expected results in the `.ll` file. The readback sequence is initiated using the Vivado hardware manager JTAG commands. For more information on these and other commands, refer to the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 2].

After the `readback_capture.tcl` file is successfully sourced, the rdbk_jtag process should be run on the Vivado Design Suite Tcl console. The options for the rdbk_jtag process are:

```
rdbk_jtag <path to save readback capture data .rdbk> <total frame count>
<overwrite (1=yes, 0=no)>
>rdbk_jtag c:/XAPP1230/LED_Count.rdbk 32530 1
```

Table 6 contains the XCKU040 device information on the total frame count, 32530, used to read back the full device contents.

Table 6: **Example UltraScale Configuration Frame Count**

UltraScale FPGA	Total Configuration Frames	Words per Frame	Overhead Words (Pipeline Word Count: 1 Frame + 10 Words)
XCKU040	32530	123	133

Notes:
1. Refer to the *UltraScale Architecture Configuration User Guide* (UG570) [Ref 1] for other family members.

The Tcl script is split up into these key sections:

- Definitions: Provides command definitions.
- Readback capture processes: Includes setup_rdbk_cmd and rdbk_jtag processes that contain the readback sequence and configuration register command writes to set the CAPTURE bit.
- Data format conversion processes: Convert the JTAG serial data into ASCII 32-bit word formatted file. Processes called include setHexRevTbl, revHexData, and conv2hex.

Readback Capture Data Analysis

After running the `readback_capture.tcl` script and the processes described, an ASCII 32-bit word formatted `.rdbk` file is generated. The `.ll` file can be used as a bit map to determine the `.rdbk` values on desired design elements. In the reference counter design, there are eight registers that can be identified. This section describes how the `.ll` file is used.

Figure 8 shows an example readback capture representation of the CLB register from the reference design.

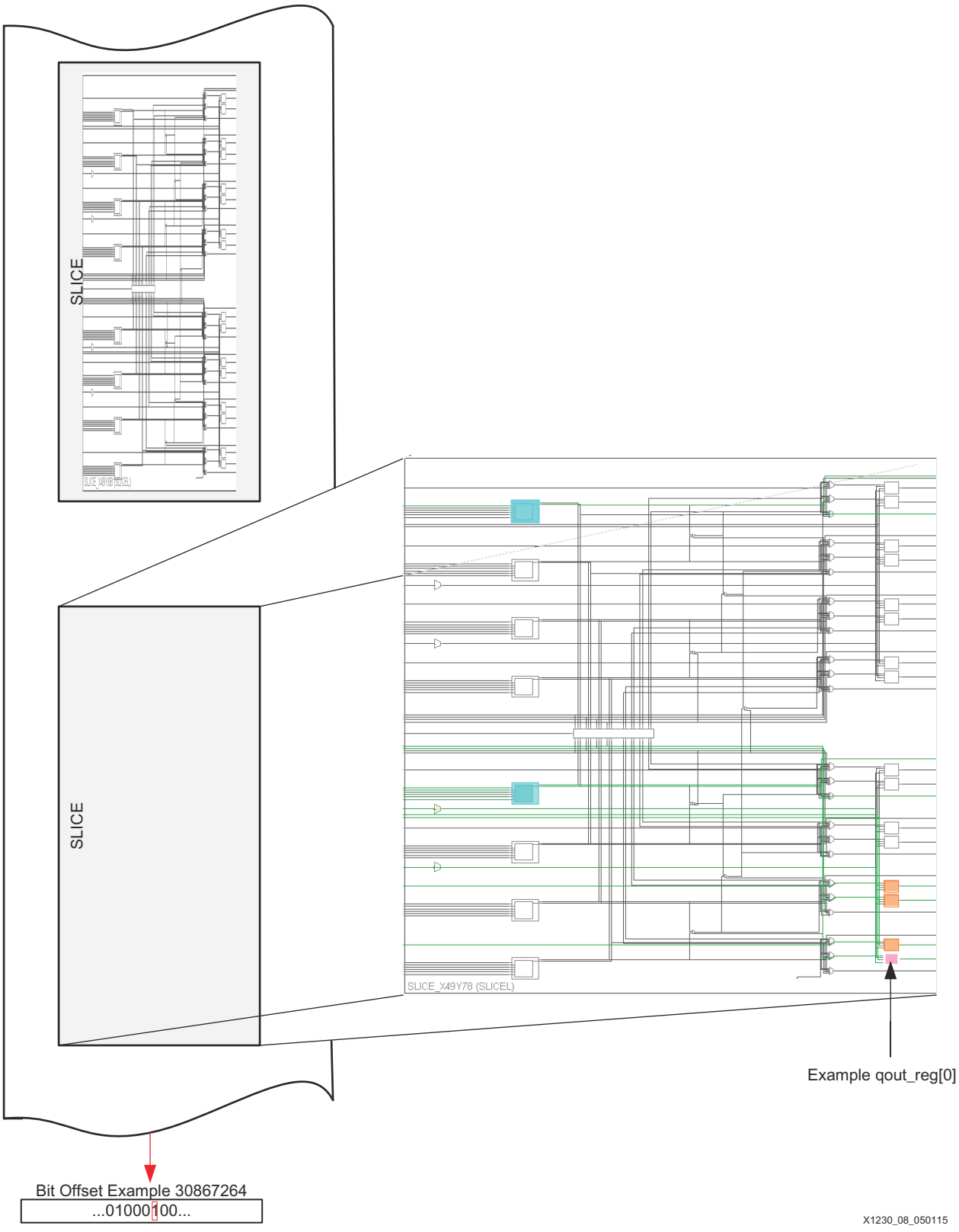

Figure 8: **Example Readback Capture Representation of CLB Register from Reference Design**

See Table 7 for the reference example `.ll` file contents. In the first register Q[0], a bit offset of 30867264 from the FAR `0x00000000` is used to calculate the CLB register bit location in the `.rdbk` file. The demonstration flow described in this application note is for a full device readback capture so bit offset is the only column required for the calculations.

Table 7: **CLB Register Summary from Reference Design `cntr.ll` Example**

Bit Offset	<frame_address>	<frame_offset>	<SLR name>	<SLR number>	Information
30867264	0x00023204	1152	SLR0	0	Block = SLICE_X49Y78 Latch = AQ Net = cntr/Q[0]
30867280	0x00023204	1168	SLR0	0	Block = SLICE_X49Y78 Latch = AQ2 Net = cntr/Q[1]
30867268	0x00023204	1156	SLR0	0	Block = SLICE_X49Y78 Latch = BQ Net = cntr/Q[2]
30867284	0x00023204	1172	SLR0	0	Block = SLICE_X49Y78 Latch = BQ2 Net = cntr/Q[3]
30868128	0x00023204	2016	SLR0	0	Block = SLICE_X49Y90 Latch = AQ Net = cntr/Q[4]
30868144	0x00023204	2032	SLR0	0	Block = SLICE_X49Y90 Latch = AQ2 Net = cntr/Q[5]
30868132	0x00023204	2020	SLR0	0	Block = SLICE_X49Y90 Latch = BQ Net = cntr/Q[6]
30868148	0x00023204	2036	SLR0	0	Block = SLICE_X49Y90 Latch = BQ2 Net = cntr/Q[7]

The Tcl script produces a `.rdbk` file that contains the readback capture data and pipeline/pad frames (133 words). The bit offset value does not include the pipeline overhead, so this must be accounted for when post-processing the logic location file.

The equation to determine the word line for the CLB register value is (int(bit offset/32) + pipeline word count + editor word) = word line. The editor word is 1 if the editor starts with word line 1, or 0 if the editor starts with word line 0. The word line offset is then calculated by the modulo of the bit offset: (bit offset mod 32) = word line offset. For example, the Q[0] CLB register bit is on word line (int(30867264/32) + 133 + 1) = 964736, with word line offset (30867264 mod 32) = 0 (Figure 9).

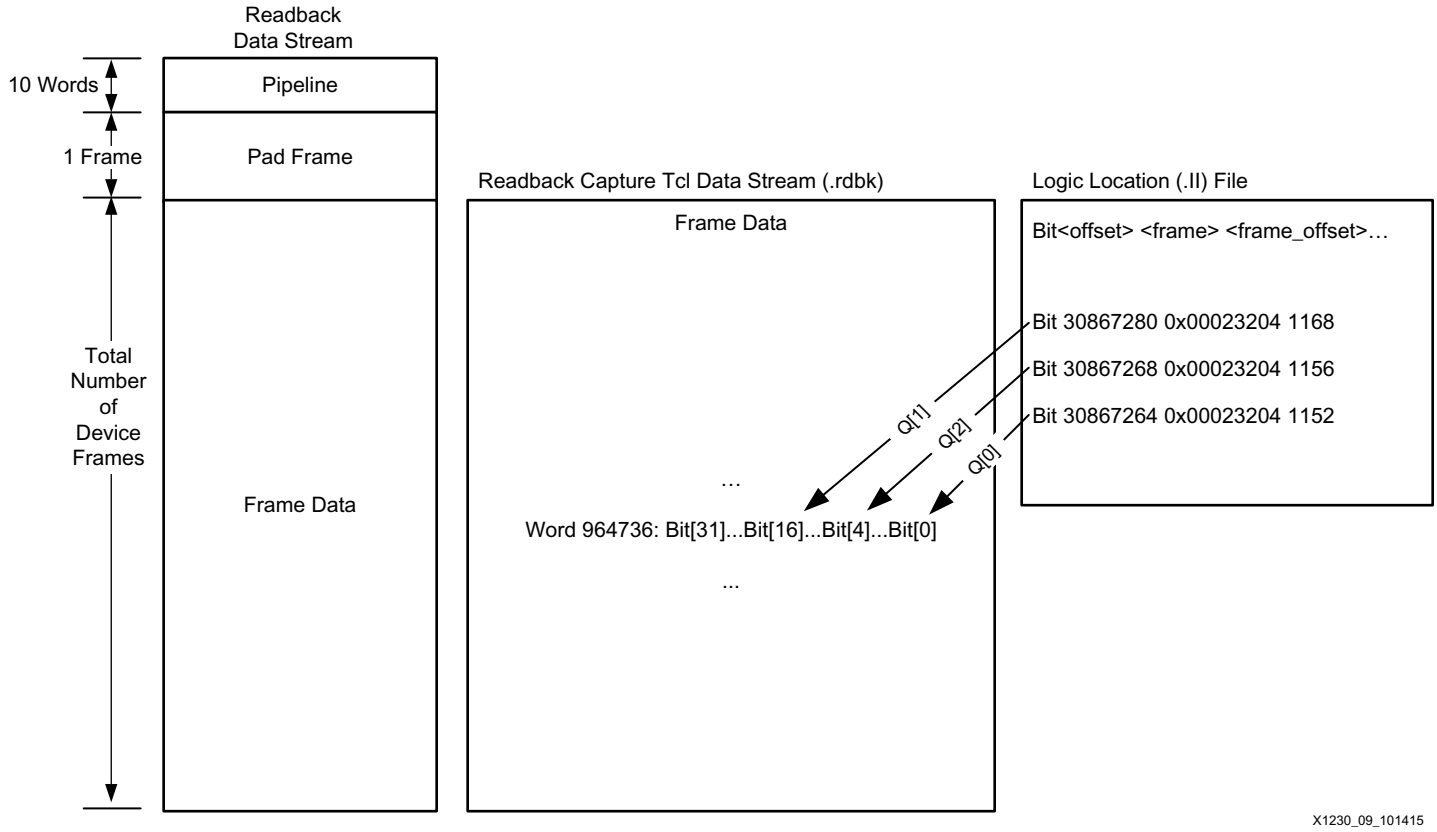

Figure 9: **Example of Using the `.ll` File to Identify Design Element Locations**

All of the CLB registers have an inversion when performing a readback capture. The CLB registers are inverted when captured, so a `0` should be seen in the readback capture file as a `1`. This does not exist for UltraScale FPGA block RAM, distributed RAM, or SRL captures.

In the example, a TCK value of 174 was entered which set the 8-bit register value and mirrored the value on LED[7:0] of `10101110`. When this value is used, the design value expected = `10101110`, and the readback value is inverted for the CLB registers as shown in Table 8.

Table 8: **Reference Design Example Readback Capture**

Reference Design Component	Bit Offset	Design Value Expected	CLB Register Readback Value Expected (Inversion)
Q[0]	Bit 0 on word line 964736	0	1
Q[1]	Bit 16 on word line 964736	1	0
Q[2]	Bit 4 on word line 964736	1	0
Q[3]	Bit 20 on word line 964736	1	0
Q[4]	Bit 0 on word line 964763	0	1
Q[5]	Bit 16 on word line 964763	1	0
Q[6]	Bit 4 on word line 964763	0	1
Q[7]	Bit 20 on word line 964763	1	0

Checklists

This section provides checklists that can be referred to in setting up your design.

Board Design/Schematic Checklist

1. Dependent on interface used, ensure connectivity as described in the *UltraScale Architecture Configuration User Guide* (UG570) [Ref 1] is followed.
2. Reference Tcl provided is for a single XCKU040 monolithic device. For devices built with SSI technology or multiple device JTAG chains, the JTAG instructions need to be modified.

Software Flow Checklist

1. Design entry: Ensure there is a method to freeze logic through readback capture (disable associated clocks).
2. Write_bitstream:
   a. Ensure a `.ll` file is generated.
   b. Avoid these bitstream settings/properties if performing readback capture:
      - JTAG disable.
      - Encryption.
   c. Only for readback capture through the SelectMAP interface, ensure the bitstream property for persist is enabled.
3. Vivado device programmer:
   a. Ensure the target design is successfully configured. For the reference design, ensure the KCU105 DONE and INIT LEDs are both lit.
   b. The UltraScale device status register can be a helpful source of debug information if the design is not loaded properly. Refer to the *UltraScale Architecture Configuration User Guide* (UG570) for more information.
4. TCL usage:
   a. Ensure the Tcl file is sourced before issuing the readback capture process command.
   b. Ensure the Tcl console is in the open_hw_target -jtag_mode 1 so that the JTAG commands can be received.
   c. If FPGA programming is needed to be repeated, ensure that the -jtag_mode is exited by running the Tcl command disconnect_hw_server.

Debug

The readback source Tcl script can be re-run to verify that the CLB register's states change. For example, an inverted pattern LED[7:0] = 01010001 can be loaded using these steps after the first readback capture with the reference design:

1. Reset the CLB register pattern by setting the KCU105 board SW12 RST signal to 0. Then issue these commands:

```
>scan_ir_hw_jtag 6 -tdi 23
>runtest_hw_jtag -tck 1
```

2. Next load the CLB register inverted pattern with these commands:

```
>scan_ir_hw_jtag 6 -tdi 23
>runtest_hw_jtag -tck 81
>scan_ir_hw_jtag 6 -tdi 3f
>rdbk_jtag c:/XAPP1230/LED_Count_inverted.rdbk 32530 1
```

The CLB register bit locations should be inverted from the LED_Count.rdbk file.

To verify the user flow and check the bit locations with the .ll file, you can create your design with the register values initialized to a known state. The write_bitstream readback_file should be selected and the INIT XDC constraint should be used for registers being captured (i.e., set_property INIT 1'b1 [get_cells FDRE_1]). You can use the .rbd file with specified INIT states against the .ll file to ensure the proper location is checked and any post-processing scripts you have are working as expected before performing the readback.

SelectMAP Interface

The readback capture can alternatively be performed through the SelectMAP or ICAP interface for applications that require faster readback performance. The SelectMAP or ICAP interface does not require the JTAG TAP command overhead so only select steps in Table 5 are applicable. The steps in Table 5 that are needed for the SelectMAP implementation are:

- Step 3: Packet data information that enters set the CAPTURE bit, sets the start frame address, and determines the number of words read.
- Step 5: Provides the readback bit count to be shifted out.
- Step 9: Gives the packet data to reset the CAPTURE bit back to default.

Conclusion

Readback capture provides the ability to read the current user state of internal CLB registers, block RAM, distributed RAM, and SRL contents to check for proper design functionality. Readback capture is a valuable feature for design debug hardware emulation or co-simulations where a large number of signals need to be observed. The feature provides easy access for observing the design state with little pre-planning and no added design logic resources. The readback capture flow is demonstrated on the KCU105 evaluation board and identifies CLB register user states.

Reference Design

You can download the reference design files for this application note from the Xilinx website.

Table 9 shows the reference design matrix.

Table 9: **Reference Design Matrix**

Parameter	Description
General	
Developer name	Stephanie Tapp
Target devices	Kintex UltraScale FPGA XCKU040
Source code provided	Yes
Source code format	Verilog
Design uses code and IP from existing Xilinx application note and reference designs or third party	N/A
Simulation	
Functional simulation performed	N/A
Timing simulation performed	N/A
Test bench used for functional and timing simulations	N/A
Test bench format	N/A
Simulator software/version used	N/A
SPICE/IBIS simulations	N/A
Implementation	
Synthesis software tools/versions used	Vivado Design Suite 2015.1
Implementation software tools/versions used	Vivado Design Suite 2015.1
Static timing analysis performed	N/A
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	KCU105 evaluation board

References

1. *UltraScale Architecture Configuration User Guide* (UG570)
2. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
3. *Vivado Design Suite Tcl Command Reference Guide* (UG835)
4. *Kintex UltraScale Architecture Data Sheet: DC and AC Switching Characteristics* (DS892)
5. *Virtex UltraScale Architecture Data Sheet: DC and AC Switching Characteristics* (DS893)
6. *KCU105 Evaluation Kit Quick Start Guide* (XTP391)
7. *KCU105 Board User Guide* (UG917)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
06/09/2015	1.0	Initial Xilinx release.
11/20/2015	1.1	Added Vivado Design Suite version to fifth paragraph in Introduction. Removed duplicate sentence from Programming FPGA via JTAG. Updated TDI for step 3 in Table 5. In Readback Capture - Tcl Script Usage, added description of readback capture option for Vivado Design Suite 2015.3 and updated rdbk_jtag process options. Updated word line equation after Table 7. Updated logic location file in Figure 9. Updated fourth command in step 2 of Debug.

Please Read: Important Legal Notices