DesignCon 2010

Leveraging Simulation Tools in FPGA Lab Debug

Torrey Lewis, Synopsys
torrey.lewis@synopsys.com

Neil Songcuan, Synopsys
neil.songcuan@synopsys.com

Abstract

With increasing design complexity and limited simulation cycles, it becomes necessary to prototype designs. Lab debug tools have traditionally consisted of voltmeters, oscilloscopes and logic analyzers, rendering tools developed specifically for simulation (i.e. waveform viewer scripts, interface assertions, checkers and transaction monitors) unusable.

This paper documents how a new generation of FPGA and rapid prototyping debug tools can be used to leverage simulation tools for complex ASIC or SoC designs. Two categories of tools will be discussed: the first increases visibility into the design; the second replays transactions captured in the lab in a simulator, allowing for the use of a bus interface transaction monitor.

Author(s) Biography

Torrey Lewis has worked in FPGA Prototyping for 15 years, initially as an intern with Intel writing tools for RTL Replay, then with Radisys Corporation designing and prototyping an Intel compatible chipset. Now with Synopsys, Mr. Lewis leads the hardware validation team for the DesignWare PCI Express IP core. Mr. Lewis holds a B.S. degree in Computer Engineering from Portland State University.

Neil Songcuan is a Product Marketing Manager with Synopsys and is responsible for the Confirma rapid prototyping platform. He has 10 years of experience in the hardware-assisted verification industry, holding various marketing management positions with Synplicity, Mentor Graphics and IKOS Systems. Prior to this, Mr. Songcuan held customer marketing and application engineering roles with Altera Corporation. Mr. Songcuan holds a B.S. degree in Electrical Engineering from San Jose State University.

Introduction

Prototyping of complex RTL designs in an FPGA is becoming a necessity, rather than a luxury to chip designers today. The increasing complexity of SoCs also require these designs to be prototyped and debugged in an FPGA. With today's large high-speed FPGAs it is possible to prototype an entire chip design running at system speed prior to ASIC tapeout. Depending on product volumes it is possible that an FPGA may be the end product without any plans for an ASIC.

Once problems of clocking and reset are corrected on the prototyping board or ASIC design, serious testing and debug can happen. The beauty of the FPGA is that it can be reprogrammed without any cost other than time. This allows FPGA-based prototyping to happen in parallel to the final stages of simulation verification. Communication, or transactions, between the ASIC or SoC and other devices can now take place in real hardware at system speed. Real-world bus transactions conducted using the FPGA can bring up sequences that were not envisioned during simulation verification planning. For example, a software driver may setup the design in a different order of configuration transactions than what was run in simulation. This different order of transactions may expose an error in the design.

When an error occurs in the lab the engineer must determine the quickest way to perform the debug and determine if a functional issue has occurred. The engineer must also determine an easy way to communicate the design flaw to the development team. Communication can be further complicated with the possibility of multiple design blocks worked on by separate design and verification teams separated by thousands of miles, as well as IP from external vendors. Because of the scope and complexity, there may not be a single person with intimate knowledge of the entire design.

Traditional FPGA Lab Debug

Traditional debug involves the use of voltmeters, oscilloscopes and logic analyzers. While these tried and true methods have served us well for many years there are limitations.

Human Error and Equipment Failure

Human error can cause delays in lab debug. In assigning signal names to the logic analyzer it is possible to make a mistake and assign a signal to the wrong pod. Equipment failure can cause unexpected results. Logic analyzer cables can become damaged through repetitive use and lead the user to believe that a stuck-at-0 fault has occurred when in reality the signal is actually toggling correctly. Imagine spending hours in the lab debugging an issue only to realize that human error or equipment failure has just rendered those hours as a waste.

Unfamiliar Software

Frequently, oscilloscopes and logic analyzers come with their own set of software. This is software that RTL designers rarely see or use until presented with a problem originating from the lab. Determining the correct download location to retrieve the software from and the correct version of the software is time wasted. This slows down the communication and ownership of a

problem due to the engineer in the lab not being equipped to "speak the language" of the designer or verification engineer.

Budgets are Shrinking

Unfortunately, limited human resources and, especially as of late, limited budget for lab equipment make traditional FPGA debug and ASIC prototyping even more difficult. While your lab may be fully equipped a customer's lab may not be so fortunate. Imagine the frustration of asking a customer to use an oscilloscope to probe an FPGA pin only to be told that they do not have an oscilloscope.

While the use of voltmeters, oscilloscopes or logic analyzers may never be completely abandoned, new methods must be used in order to meet time-to-market demands, while keeping costs down and preserving the sanity of engineers.

FPGA Lab Debug Today

This paper explains the current usage of two categories; increasing observability and replaying lab transactions in simulation in debugging a complex ASIC or SoC design.

A fundamental problem with lab debug is that the engineer is removed from the FPGA itself. There may be layers of software that separate the stimulus that can be controlled and the actual input and output pins of the FPGA.

When an error occurs the first step is to determine the source of the problem. The problem could be in the software or in the hardware. It would be wise to ask the question: "Did the transaction I stimulated in software actually reach the FPGA?" If the transaction never reached the FPGA then there is a software issue or a hardware issue in the datapath before the FPGA. If the transaction did reach the FPGA then there could be an FPGA design issue.

Determining if a transaction is received by the FPGA could be difficult to determine depending on the board layout and FPGA package. The input and output pins of the FPGA may be inaccessible to directly probe using an oscilloscope. The board may have mictor connectors that can be connected to a Logic Analyzer, such as shown in Figure 1. The board shown in Figure 1 has 4 mictor connectors each supplying to 32 signals, for total access to 128 signals.

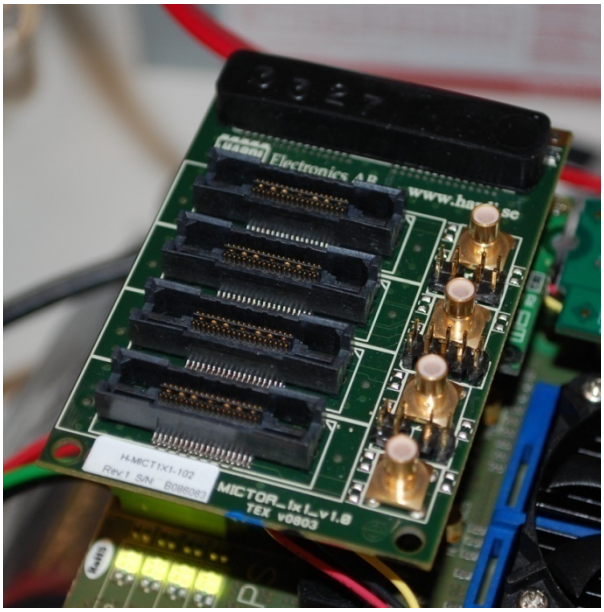


Figure 1. Logic Analyzer Mictor Connectors

If the mictor connectors are connected to FPGA general purpose I/O pins then internal signals can also be routed to these pins. For example, internal state machine and control signals could be routed to the mictor connectors for observation with the Logic Analyzer.

However, as designs increase in complexity the number of signals that can be routed to the available mictor connectors may become a limitation. 128 signals may not be enough to fully observe several internal data busses as well as controlling state machines. Trade-offs would need to be made to determine which signals route to the mictor connectors.

Increasing Observability

There is a better way. Using this first category of FPGA debug tools it is possible to increase the visibility of the internal signals of a design beyond the mictor connector limitation.

FPGA and prototyping debug tools such as Synopsys Identify, Xilinx ChipScope or Altera SignalTap add logic analyzer capability inside the FPGA. FPGA logic is used for triggering, and internal RAM Blocks are

used for data storage. These debug tools modify a typical design flow as shown in Figure

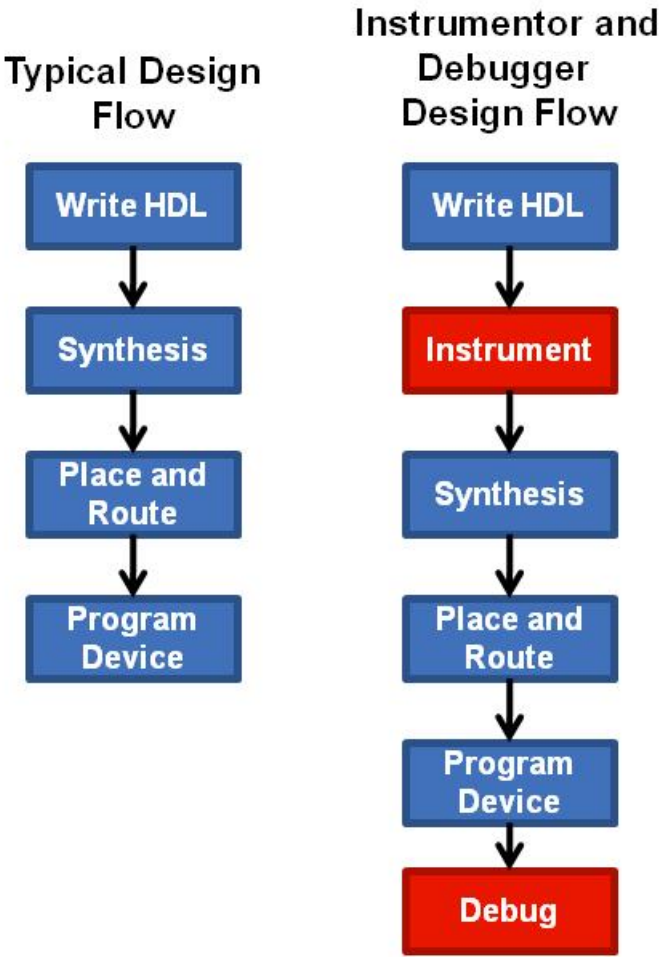


Figure 2.

In the instrument phase, the instrumentation tool is used to select which signals to observe. The RTL Instrumentor then automatically connects the internal signals to the RAM blocks and creates the necessary triggering logic. The design with the instrumented logic is then synthesized into the FPGA.

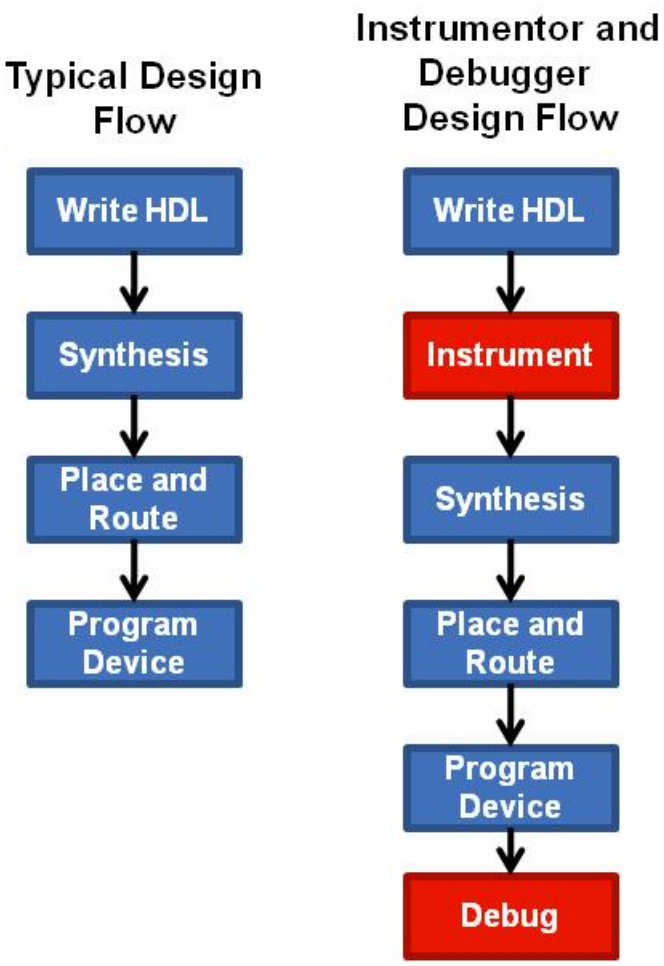


Figure 2. Design Flow When Adding Observability

After the FPGA is configured, the RTL debugger software (Synopsys Identify tool) communicates with the FPGA using a JTAG interface as shown in Figure 3. This software is used to set a trigger and capture the instrumented signals. The captured signals can then be viewed using a standard waveform viewer.

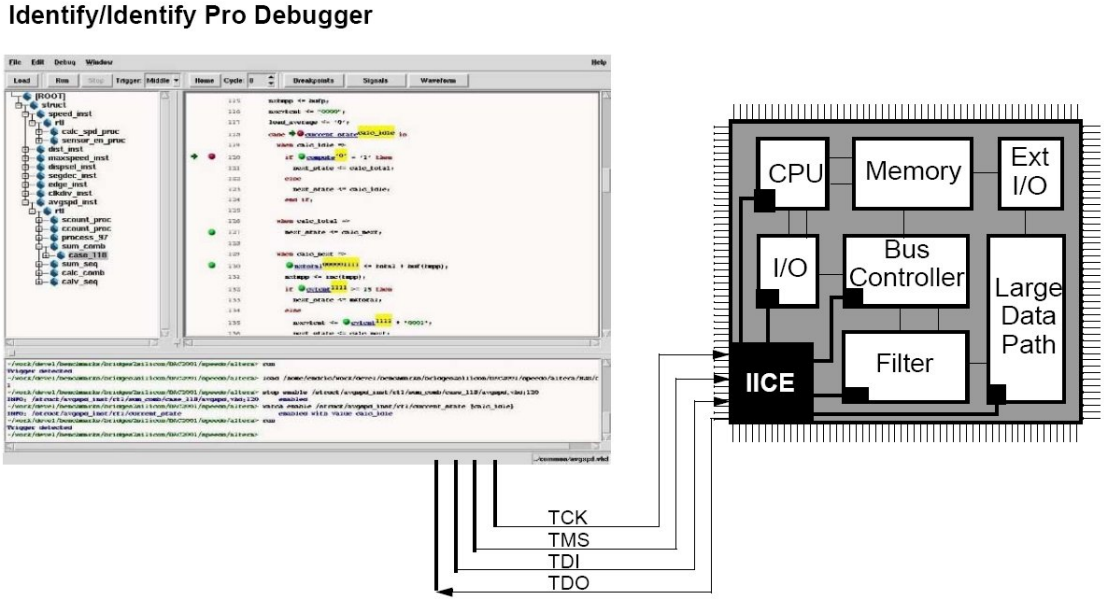


Figure 3. RTL Debugger Connection to FPGA via JTAG

The debug tools can output data in the popular .vcd (Verilog Change Dump) file format that is used by many waveform viewers. This means that the same waveform viewer used by the design and verification teams use can also be used when viewing waveforms generated from the lab.

Time can be saved when using the standard waveform viewer as well. Naturally, the signals in the FPGA are ones and zeros. For humans, this may not be as intuitive as using state machine names. For example, in the Synopsys DesignWare PCI Express design the Link Training and Status State Machine state of "L0" is encoded as 5'b10001. A User-Defined Radix in the waveform viewer can be used to automatically convert the 5'b10001 to the readable text of "L0". In fact, in most cases these user-defined radices have already been written by the design and verification teams and can now be re-used to aid debug.

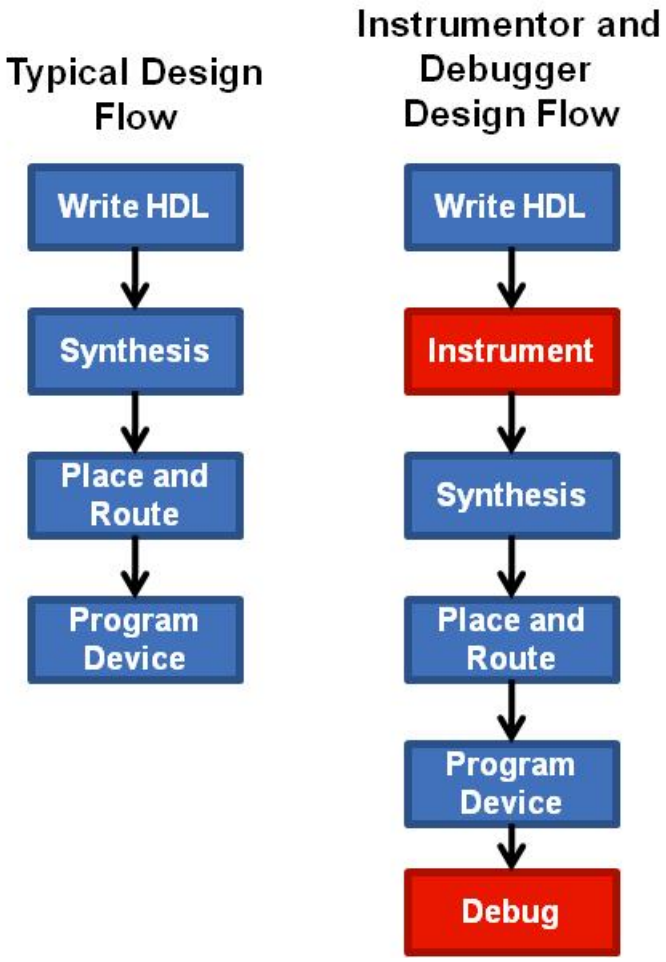
Also, using the standard waveform viewer of the design and verification teams eliminates any software installation issues regarding specific logic analyzer software. This removes the need for a design or verification engineer to install new software to effectively debug an issue originating in the lab. The person performing debug can now focus on finding the source of the problem, rather than learning how to use a new tool.

Using debug tools to increase observability can also save money. Utilizing the FPGA's internal RAM Blocks as data storage and logic as the triggering mechanism can reduce the need for logic analyzers.

One drawback to this type of FPGA and prototyping debug tools is that the signals must be defined during the instrumentation phase of the design flow. If more signals are needed, instrumentation and synthesis need to be rerun. For large designs, FPGA synthesis may be an overnight process.

Replaying Lab Transactions in Simulation

The second category of FPGA or prototyping debug tools addresses this very issue by providing full-visibility into the instrumented portion of the design. These tools automatically create a simulation environment using the stimulus captured in the lab. The created simulation environment consists of a testbench containing only the portion of the design to be replayed along with the input stimulus captured in the lab.



The design flow is similar to that shown in

Figure 2 (page 7), with the exception that the Debug phase is replaced by a simulation phase and debug is performed in the simulator. Figure 4 shows an example design prior to instrumentation for simulation replay. The `module_B` portion of the design is suspected to contain a bug. A tool such as Synopsys Identify Pro is used to instrument `module_B` for replay in simulation.

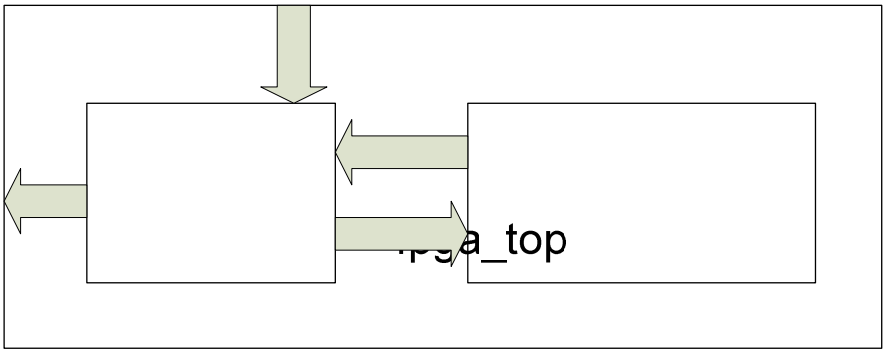


Figure 4. Original Example FPGA Design

In order to correctly replay lab transactions, two types of information are needed. The first is the initial condition of the module to be simulated. The second piece of information is the input stimulus to the module. In order to obtain both sets of information Identify Pro modifies the original design as shown in Figure 5.

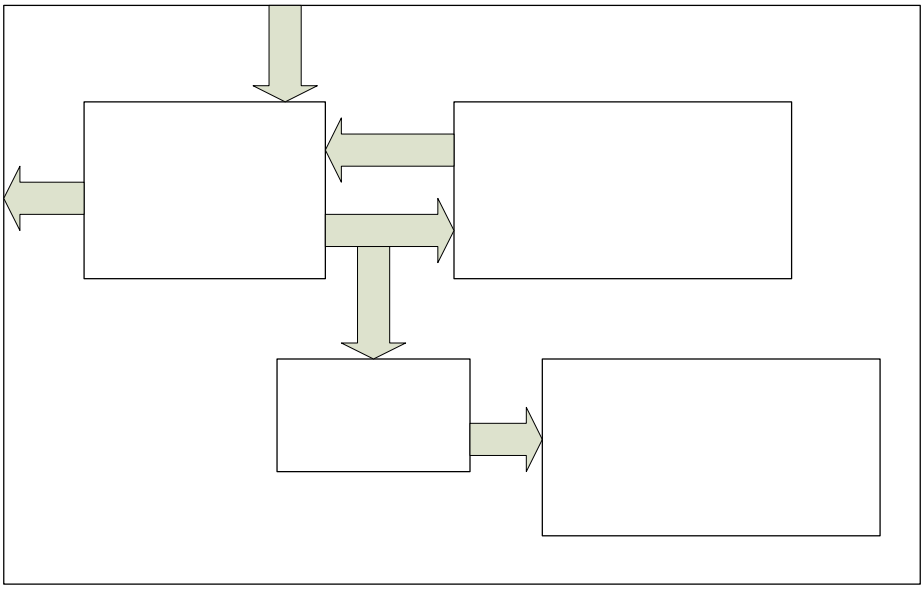


Figure 5. FPGA Design Instrumented for Simulation

Identify Pro replicates the instrumented module and inserts a FIFO buffer that stores the input stimulus to the instrumented block. The depth of the FIFO buffer determines how many cycles the `replicated_module_B` is behind `module_B`. When the trigger condition occurs in `module_B` the FIFO buffer is stopped. Nothing more is loaded into the FIFO buffer or unloaded into `replicated_module_B`. `replicated_module_B` now contains the initial condition of `module_B` and the FIFO buffer contains all of the stimulus that led to the trigger condition.

Identify Pro then creates a testbench containing the `replicated_module_B`, clock generation logic that reproduces the clock timing relationships and the input stimulus captured in the FIFO buffer. This is shown in Figure 6.

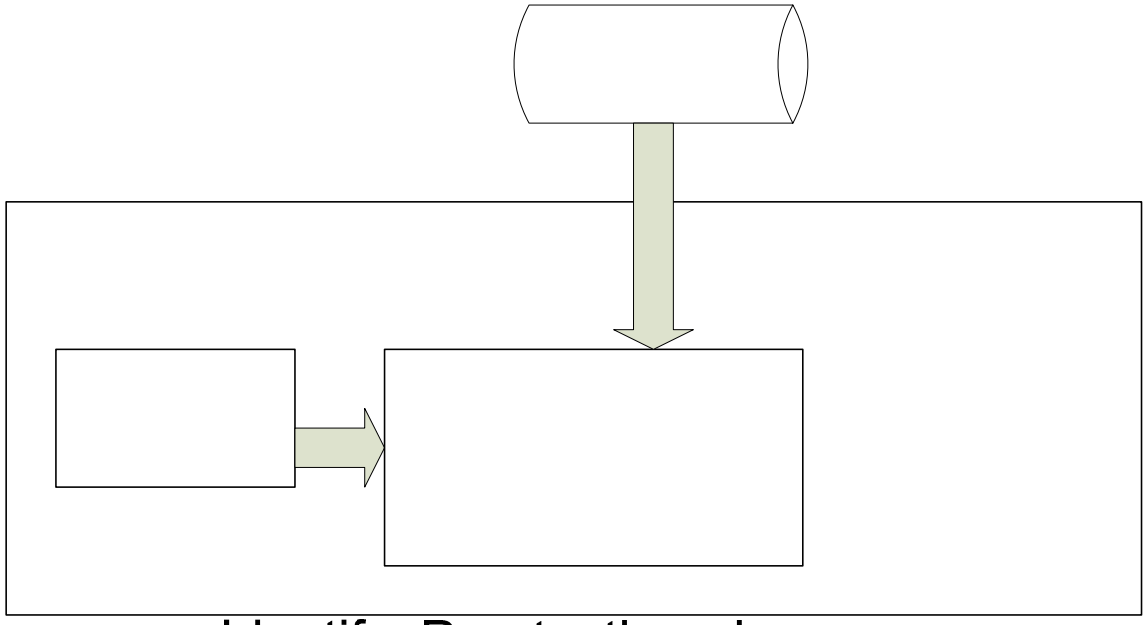


Figure 6: Identify Pro Simulation Testbench

Because the transactions are replayed in simulation the user will have full visibility at the RTL level of all of the signals in `replicated_module_B`. Again the standard waveform viewer can be used during debug in a simulation environment. Notice that all of the information about `replicated_module_B` is contained in the simulation produced waveform. There is no longer a need to rerun instrumentation and synthesis to observe more signals.

To take this idea further, assertions, protocol checkers and monitors could be added to the Identify Pro testbench manually by the user. Assertions, checkers and monitors are frequently written and used during the simulation stage of a project. Now, with the ability to replay lab captured stimulus in simulation, these tools can be employed again. Figure 7 shows an example of how these debug features could be added.

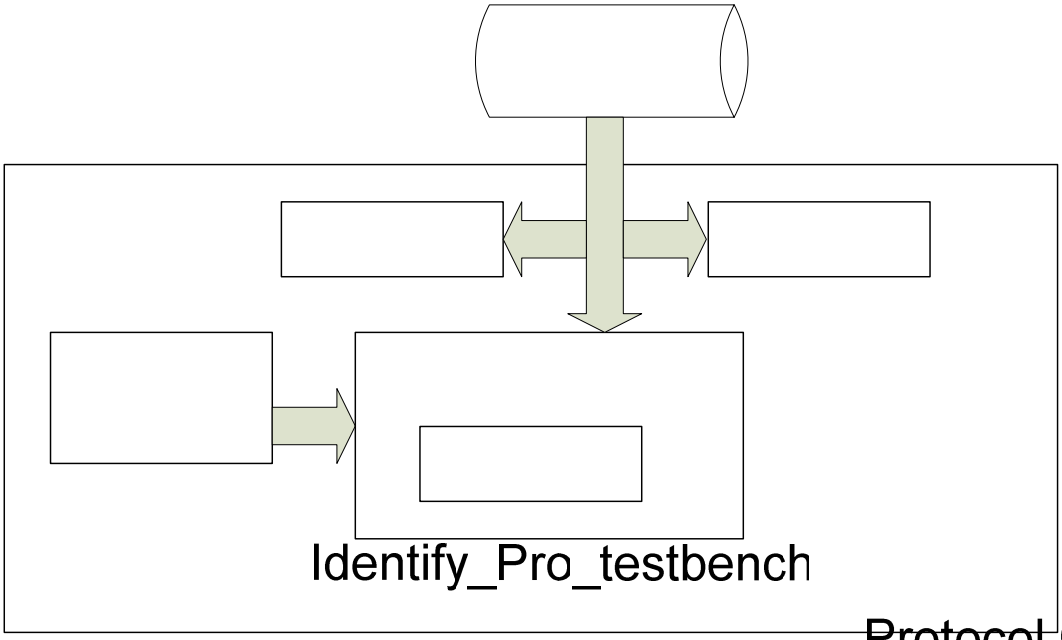


Figure 7. Identify Pro Testbench with Additional Debug Features

Bringing It All Together in a Real Design

Theory is great, but FPGA lab or prototyping debug is all about making it real. In order to provide concrete real-world examples of these concepts a design utilizing the Synopsys DesignWare SATA and PCI Express IP cores and Synopsys FPGA tools have been used. This particular design involved separate IP design teams for SATA and PCI Express separated by both distance and time. Additionally, the design interfaces to the Rocket I/O Transceivers in the FPGA, which is equivalent to third party IP.

Figure 8 shows the block diagram of a design used in this example. This design acts as a simple PCI Express to SATA bridge. The FPGA-based prototyping board plugs into the PCI Express slot of a PC motherboard and then bridges to a SATA DVD or HD drive.

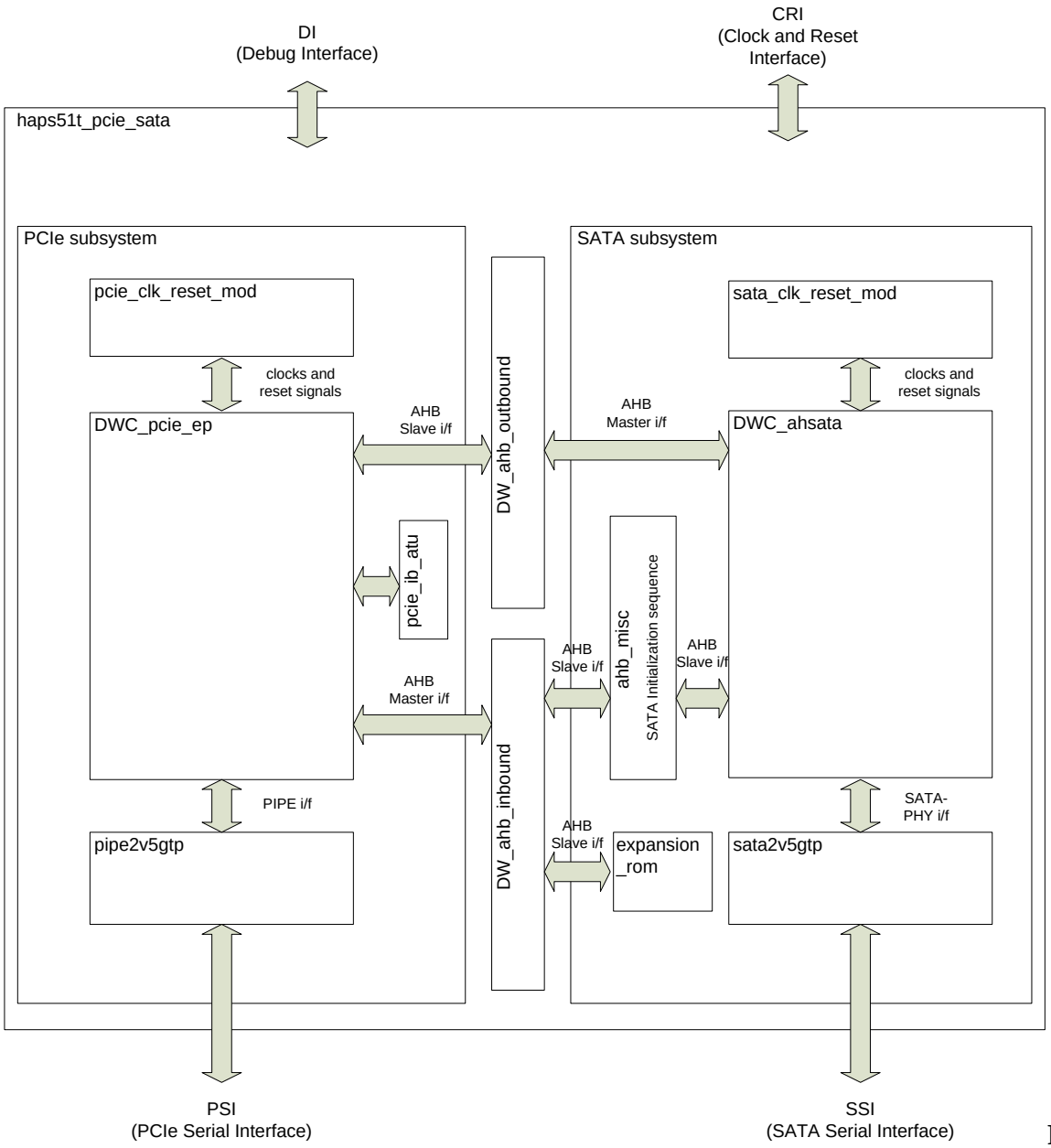


Figure 8. Example Design Block Diagram

Increasing Observability in the Real World

An FPGA or prototyping debug tool such as Synopsys Identify, Altera SignalTap, or Xilinx ChipScope could be used to instrument the signals that need to be observed. For increasing the observability into this design the Synopsys Identify tool is used. The following steps are taken:

1. Determine which signals to instrument.
2. Use the Instrumentation tool to instrument the signals.
3. Perform synthesis on the instrumented design.
4. Set up the Debugger tool to trigger on the error.
5. Use the standard waveform viewer to assess the situation.

Determining the Signals to Instrument

When working on any design there are always opportunities for bugs. From the outset of testing, it can be assumed that being able to observe each of the major interfaces in the design would speed determining which blocks are at fault. In the example design, the AHB busses as well as the ports of the DWC_pcie and DWC_ahsata IP blocks were identified even prior to initial synthesis. By using this divide and conquer strategy it will be easier to determine which design team to contact for additional support.

As mentioned earlier, the PCIe and SATA designs are independent IP blocks designed by separate design teams. Each design team is an expert on their portion of the design. The PCIe design team will know in advance which signals are important for debug of the PCIe subsystem. The SATA design team will know which signals are important for debug of the SATA subsystem. Each subsystem or IP core may even have dedicated debug ports that bring out the signals to the top-level. Major internal state machines can also be identified and instrumented in advance.

Using the Instrumentation Tool to Instrument the Signals

While the Identify Instrumentor TCL scripts may not already be written by the design or verification engineer, they can be written in a way that can be re-used for future FPGA designs using the same IP blocks. An example Identify Instrumentor TCL script is shown in Figure 9. Notice that the `rtl_path` parameter could be changed when used in a future FPGA design.

The first TCL procedure creates the Intelligent In-Circuit Emulator (IICE). The IICE acts as the triggering mechanism similar to that of a logic analyzer. The IICE can be configured to cross-trigger from other IICEs in the design and have a state machine associated with it to allow for complex trigger mechanisms.

The second TCL procedure shown in Figure 9, tells the Identify tool to instrument the specified signal, in this case the PCI Express Link Training and Status State Machine (LTSSM). As a result of this command, Identify will connect this signal to an internal FPGA RAM Block for data capture.

```
## create the IICE
proc DWC_pcie_create_iice_core_clk { \
          {rtl_path /u_DWC_pcie_ep} \
          {depth 4096} \
          {iice_name IICE_DWC_pcie_core_clk} \
          {crosstrigger 1} \
          {qualified_sampling 0} } {
  iice new ${iice_name}
  iice sampler -iice ${iice_name} -depth ${depth} \
     -qualified_sampling ${qualified_sampling} behavioral
  iice controller -iice ${iice_name} \
     -crosstrigger ${crosstrigger} statemachine
  iice clock -iice ${iice_name} ${rtl_path}/core_clk
}

## signals to sample and trigger
proc DWC_pcie_probe_ltssm { \
          {rtl_path /u_DWC_pcie_ep} \
          {iice_name IICE_DWC_pcie_core_clk} } {
  iice current ${iice_name}

  # xmlh_ltssm signals
  signals add -sample -trigger \
     ${rtl_path}/u_cx_pl/u_xmlh/u_xmlh_ltssm/ltssm
}
```

Figure 9. Identify Instrumentor TCL script

Perform Synthesis on the Instrumented Design

Once the design has been instrumented for observability it must be synthesized. Refer to the user manuals of your FPGA synthesis software for details.

Set Up the FPGA Debugger Tool to Trigger on the Error

In this example, we want to set the trigger condition to be when the DWC_pcie core receives a Configuration Read transaction. This event is signaled with a low to high transition on the `rcvd_cfg0rd_req` signal. Figure 10 through Figure 12 illustrate the setting of this trigger expression in Identify RTL Debugger.

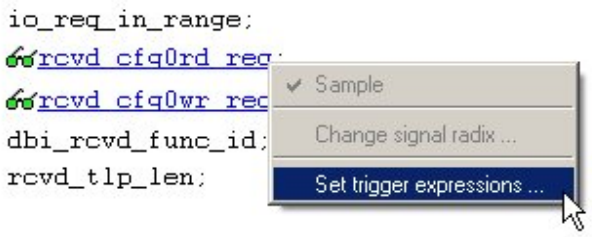


Figure 10. Setting the Trigger Expression

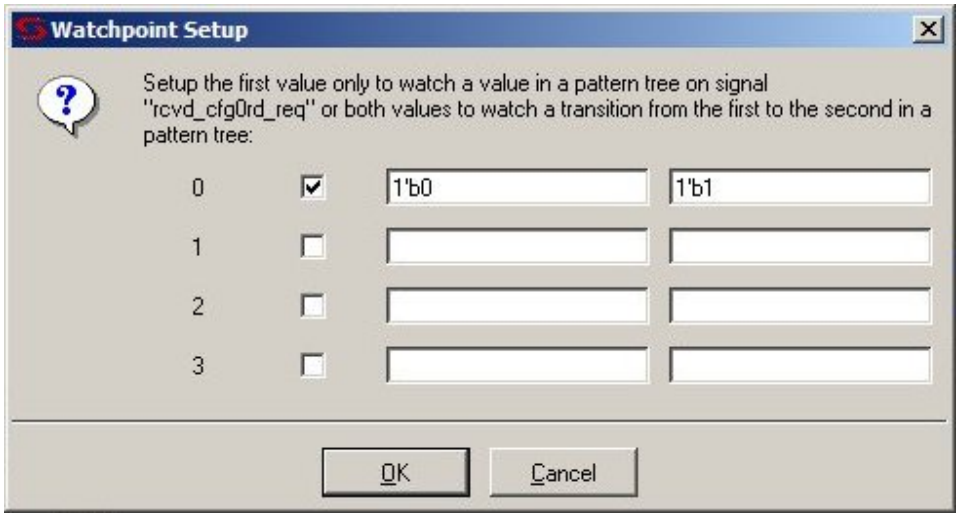


Figure 11. Low to High Transition

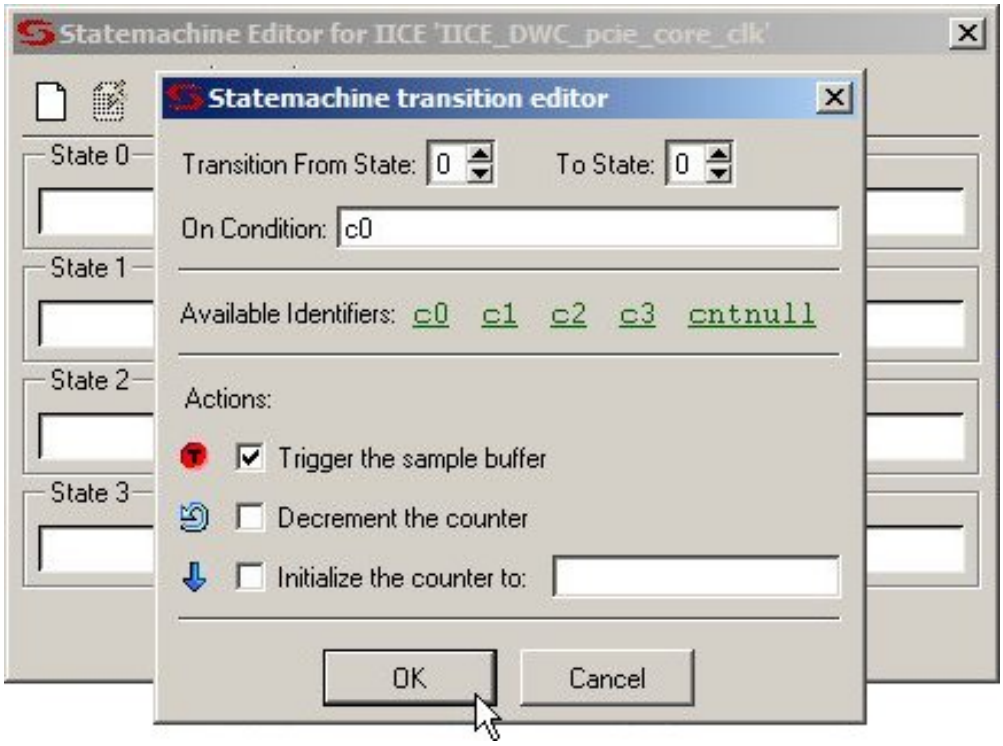


Figure 12. Configuring the IICE State Machine

Re-using Waveform Viewer and Scripts in the Lab

Since the output of Identify RTL Debugger is the standard .vcd file format, the standard waveform viewer of the design and verification teams can be re-used for lab debug. By using the same waveform viewer, user-defined radices and signal groupings makes it easy for a design or verification engineer to assist in lab debug. There are no new tools to use and the waveform viewer is one that they are comfortable with.

A portion of these waveform viewer scripts are shown in Figure 13 and Figure 14. Note that these scripts were developed independently by each design team.

```
# Global: User-defined Radices
gui_set_userradix -name DWC_pcie_LTSSM_RADIX { \
               { 5'h00 S_DETECT_QUIET   #ffffff #000000 } \
               { 5'h01 S_DETECT_ACT     #ffffff #000000 } \
               { 5'h02 S_POLL_ACTIVE    #ffffff #000000 } \
               { 5'h04 S_POLL_CONFIG    #ffffff #000000 } \
…
}
```

Figure 13. Example PCI Express Waveform Viewer Script

```
gui_set_userradix -name DWC_ahsata_tx_init_state_RADIX {  \
   { 4'd0  I_PHY_RESET       #ffffff #000000 } \
   { 4'd1  I_PHY_WAIT        #ffffff #000000 } \
   { 4'd2  I_COMRESET        #ffffff #000000 } \
   { 4'd3  I_WAIT_COMINIT    #ffffff #000000 } \
   { 4'd4  I_WAIT_NO_COMINIT #ffffff #000000 } \
   { 4'd5  I_COMWAKE         #ffffff #000000 } \
…
}
```

Figure 14 Example SATA Waveform Viewer Script

Figure 15 shows the result of using the waveform viewer scripts. Notice that the design and verification teams have already setup many user defined radices. This saves time in the lab debug process.

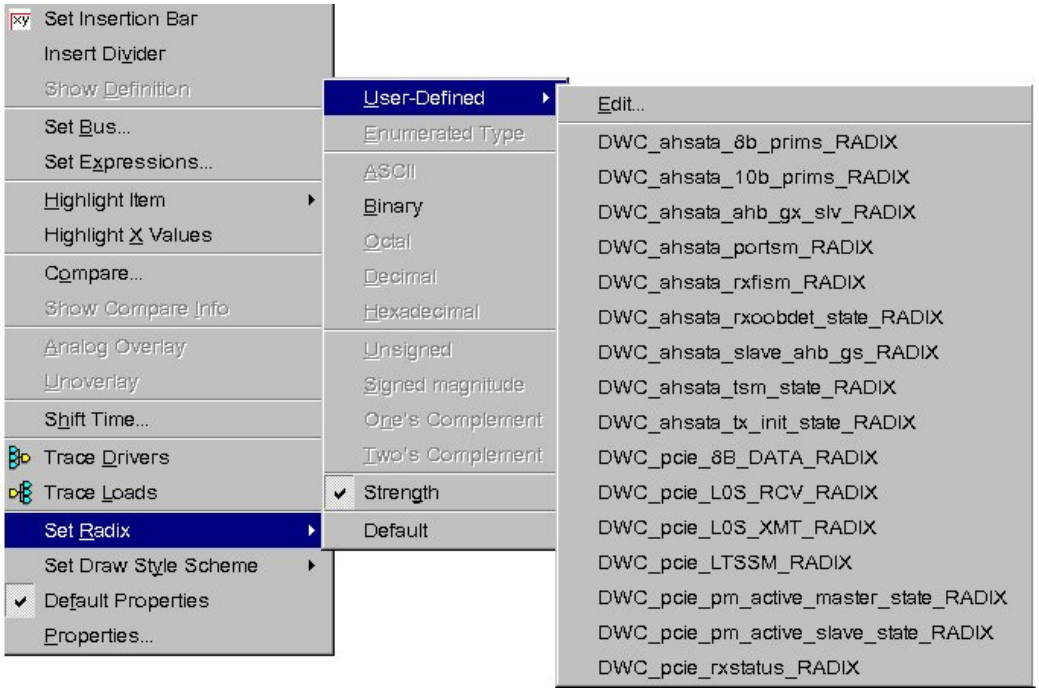


Figure 15. Waveform Viewer User-Defined Radices

For this design it was determined that the Link Training and Status State Machine in the PCIe subsystem is important in tracking the PCIe bus. Additionally, the SATA team determined that the Device Initialization State Machine in the SATA subsystem is important to track during FPGA bring-up in the lab.

Figure 16 shows the viewing of these two state machines during a lab debug session. Note, the state machine names and waveform viewer are identical to the simulation and design environment. There is no reason for the design or verification engineer to learn a new tool specific to a particular logic analyzer. All functional debug can be conducted in tools that the design and verification engineers are familiar with.

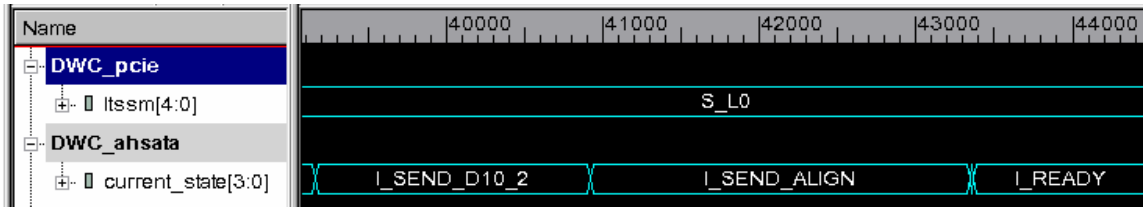


Figure 16. Waveform Viewer Common to Simulation and Lab Debug

Replaying Lab Transactions in Simulation in the Real World

As mentioned before, during a debug session it may be impossible to know all of the signals of interest. For example, a first iteration may involve many high-level signals to determine which RTL blocks may be the source of the problem. In communicating with the design team the bug

is narrowed to a smaller portion of logic, and more signals from that portion of the design may need to be observed.

In this example, a portion of the DWC_pcie IP core was instrumented with Synopsys Identify Pro. Identify Pro was then used to trigger on an incoming Configuration Read transaction as indicated by the low to high transition of `rcvd_cfg0rd_req` signal as mentioned before. The set of transactions was then replayed in simulation.

Figure 17 shows the result of the simulation. Notice that the previous input stimulus, not shown in the figure, actually caused the `rcvd_cfg0rd_req` signal to transition. Also notice how the `lbc_proc_state` machine responds to this signal. Both of these transitions were never captured in the FIFO buffer added by Identify Pro, but are the result of replaying the input stimulus to the replicated block.

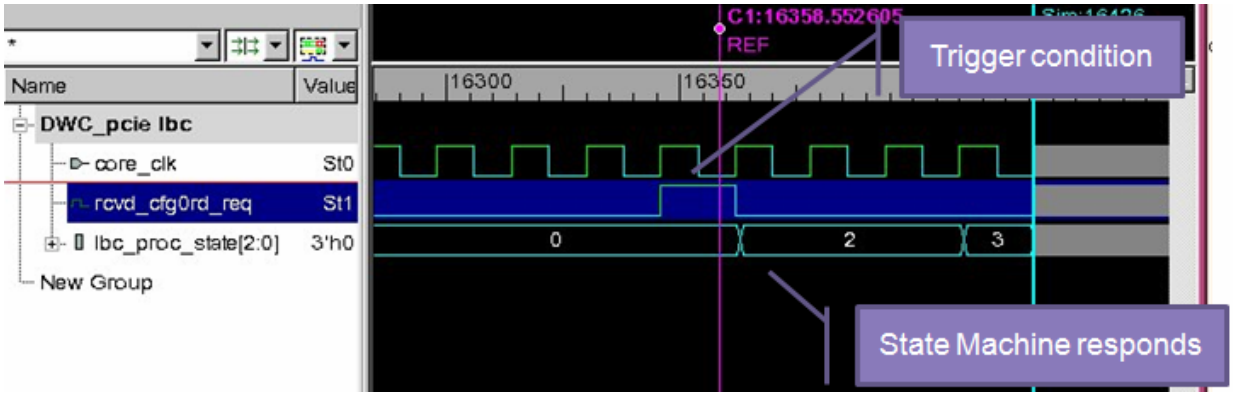
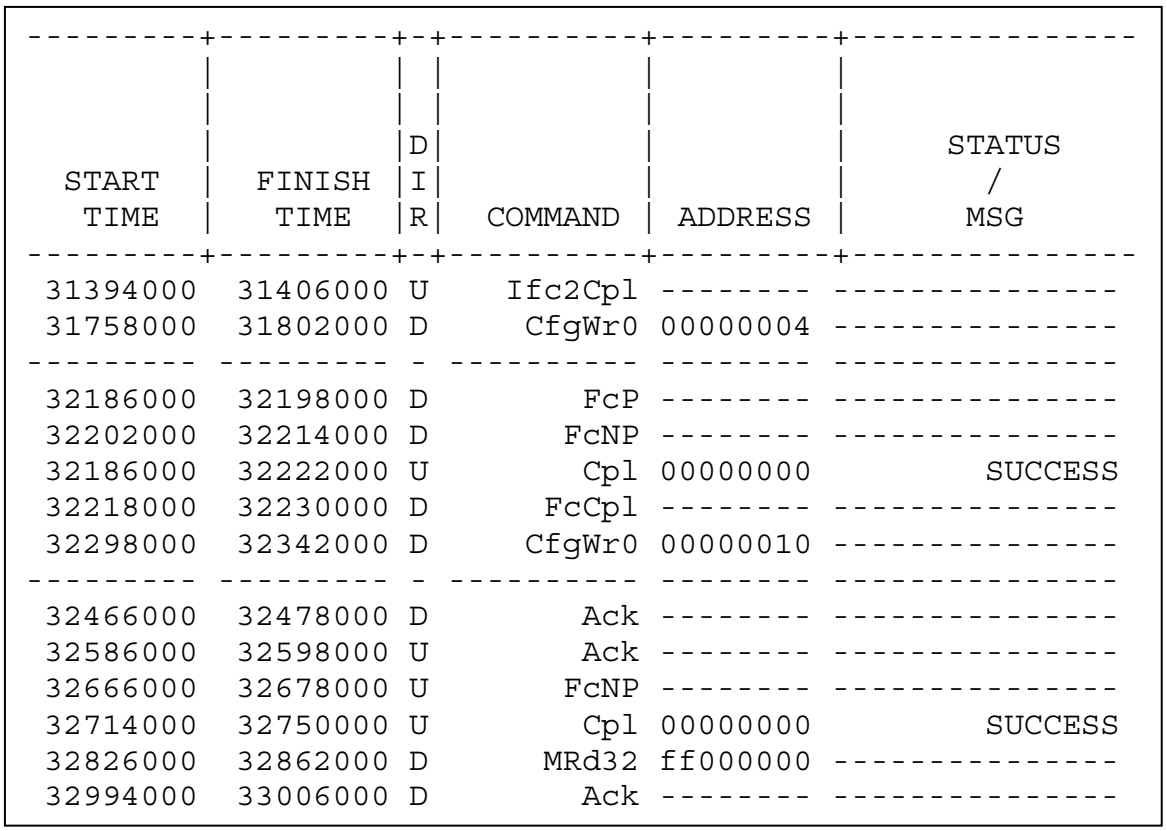


Figure 17. Replaying Lab Captured Transactions

To extend this idea further, a PCIe transaction monitor could be added to the Identify Pro created testbench to track transactions in simulation. Transactions coming into the DWC_pcie module are 8-bit codes that make up the transactions. With the use of the PCIe transaction monitor, the transactions could be presented in a more human readable format, such as shown in Figure 18**Error! Reference source not found.**.



START TIME	FINISH TIME	DIR	COMMAND	ADDRESS	STATUS / MSG
31394000	31406000	U	Ifc2Cpl	--------	----------
31758000	31802000	D	CfgWr0	00000004	----------
32186000	32198000	D	FcP	--------	----------
32202000	32214000	D	FcNP	--------	----------
32186000	32222000	U	Cpl	00000000	SUCCESS
32218000	32230000	D	FcCpl	--------	----------
32298000	32342000	D	CfgWr0	00000010	----------
32466000	32478000	D	Ack	--------	----------
32586000	32598000	U	Ack	--------	----------
32666000	32678000	U	FcNP	--------	----------
32714000	32750000	U	Cpl	00000000	SUCCESS
32826000	32862000	D	MRd32	ff000000	----------
32994000	33006000	D	Ack	--------	----------

Figure 18. Example Output of PCIe Transaction Monitor

It should be noted that the initial conditions of the PCIe transaction monitor must also be set. The section of transactions captured in the FIFO buffer may not contain vital information such as the number of PCIe lanes negotiated between the components on either side of the PCIe link. This would have happened in the physical hardware much earlier in time.

Conclusion

In this paper we have explored ways that existing simulation tools can enhance and speed-up FPGA Lab and prototyping debug. Through the use of a common waveform viewer environment, engineers are equipped to take advantage of already developed waveform viewer scripts. Design and verification engineers are also not required to learn how to use a new piece of software specific to a particular logic analyzer.

Taking it to the next level, we have also explored ways that transactions captured in the lab can be simulated, thus providing full visibility into the design as well as taking advantage of debug tools developed for simulation such as transaction monitors.