LA-UR- 09-00200

Approved for public release;
distribution is unlimited.

Title:	In-situ FPGA Debug Driven by On-Board Microcontroller
Author(s):	Zachary K. Baker, CCS-1
Intended for:	IEEE Symposium on Field-Programmable Custom Computing Machines April 6-8, 2009 Napa, CA

Form 836 (7/06)

In-situ FPGA Debug Driven by On-Board Microcontroller

Zachary K. Baker
Los Alamos National Laboratory
Los Alamos, NM 87545
Email: {zbaker}@lanl.gov

Abstract

Often we are faced with the situation that the behavior of a circuit changes in an unpredictable way when chassis cover is attached or the system is not easily accessible. For instance, in a deployed environment, such as space, hardware can malfunction in unpredictable ways. What can a designer do to ascertain the cause of the problem? Register interrogations only go so far, and sometimes the problem being debugged is register transactions themselves, or the problem lies in FPGA programming. This work provides a solution to this; namely, the ability to drive a JTAG chain via an on-board microcontroller and use a simple clone of the Xilinx Chipscope core without a Xilinx JTAG cable or any external interfaces required. We have demonstrated the functionality of the prototype system using a Xilinx Spartan 3E FPGA and a Microchip PIC18f2550 microcontroller. This paper will discuss the implementation details as well as present case studies describing how the tools have aided satellite hardware development.

1 Introduction

Visibility in a design is a priceless commodity during debug. Providing that visibility outside the normal channels of a register system and application behavior can be the difference between unexplained failures and mission success. At Los Alamos National Lab, aggressive launch schedules force engineers to solve problems quickly, thus requiring unique tools. One of the key tools we have recently found to be invaluable is a suite a JTAG tools to provide in-situ debug.

We provide a variety of tools to the debug engineer to provide an increased amount of debug control and visibility than would otherwise be available. This is possible because we can read and write an arbitrary address space. This allows us to set the system to a particular state, set parameters, trigger actions, as well as provide programmed input/output (PIO) to allow for access to a secondary, indirect address space. This is particularly useful for debugging large data-driven applications. For instance, through PIO, we provide access to the entire seven banks of QDR SRAM in our system from the linux command line. Because it is meant for interactive debug, the slow JTAG debug connection is not a problem – the system runs more than fast enough for human use.

In order to accomplish this, we implement a JTAG controller in an embedded controller. In our case, we assume this is a SPARC implemented in a RTAX 1000 such that we have a radiation-tolerant processor providing reliable operation. The controller implements software routines to run the JTAG I/O pins. The software is based on Xilinx Application Note #058, which demonstrates how to implement a SVF file player (Serial Vector Format), which can be used to query ID codes or program the FPGA. This functionality is extended to support the extended user-mode JTAG commands required to support the debug core. Because the set of extended commands is built within the normal TAP controller architecture, it is a minimal extension to the original source (see Figure 1).

2 Implementation Details

The internal debug core resides in user logic. Thus, the FPGA must be programmed for the internal debug to function, however, IDCODE and FPGA program-

ming can be accomplished through the JTAG controller without the internal code in place. The debug core is based on the Xilinx BSCAN component, which gives user logic access to the boundary scan. The initial code implemented this functionality was based on the Gnat application note, but extended significantly to include read/write register access as well as the more elaborate triggering and snapshot system required for the internal digital logic analyzer. The Gnat application note includes firmware examples that proved to be unreliable in practice. This code was extended to improve its reliability using error correction among other techniques.

In all recent Xilinx Spartan and Virtex devices, there are at least two BSCAN components. We use the second component in the chain, such that the first is available for other debug tools. For instance, Xilinx Chipscope uses first component, thus, we can simultaneously use both Chipscope and our tools. Because the Xilinx core is not as flexible in regards to its ability to both write and read user registers, it is sensible to instantiate both cores in a system to cover difficult debugging chores.

Our debug core is based on a simple triggering system, which includes a trigger pattern, a trigger mask, and an arm bit. The snapshot system is based on a Block-RAM along with a double-registered input. This allows for captures that should not negatively affect the timing behavior of a design. After the trigger and trigger mask is set, the arm bit is set. The JTAG controller must then poll the done bit to determine when to download the capture. Download is accomplished through simple PIO on the snapshot buffer.

Figure 3 shows an example capture of a counter attached to the inputs of the debug core. The trigger was set to x"00000001" and the mask was set to x"FFFFFFFF", which means that all of the input bits must match the input trigger pattern. The output display mechanism is determined entirely by the software implementation downstream of the JTAG controller; in our case, the output data is dumped to an Octave display package.

3 Integrating Microcontroller with FPGA Device

The schematic of our microcontroller implementation follows. The controller attachment from the microcontroller is achieved through the use of either open-collector outputs or high-Z outputs. This is required so that the normal behavior of the Xilinx JTAG controller is not compromised when attached (this is also addressed in the FPGA programming, see below). If the microcontroller is not capable of switching off its outputs (to a high-Z state), then an external driver with an enable must be used. In Figure 2, the connections of the JTAG controller to the JTAG pins of the FPGA slave is illustrated. TMS and TDI have internal pull-ups in Xilinx FPGAs. However, TCLK does not and must be driven high. Thus, in either the open-collector or high-Z option is used, TCLK must be arranged so that it can be cut off to allow an external controller to drive the chain. Alternatively, if it proves impossible to prevent the microcontroller from driving the chain, a resistor of sufficient size can be inserted serially between the controller and the JTAG socket, such that the external controller can safely overpower the embedded controller. Of course, this will limit the current drive capability of the microcontroller and thus limit the frequency at which TCLK can be driven, but will not impact the overall functionality of the system, as the JTAG chain can be driven at DC clock rates.

The logic analyzer core is based on the register functionality handled directly from the BSCAN primitive. This can be used for generalized register reads and writes in addition to the logic analyzer. We allocated a block of register addresses for replicating access to the control system normally handled through the rad-hard flight computer.

This is highly useful in a debug scenario as it can be used independently of the normal communication control channel. This allows in-situ system monitoring while normal activities proceed.

4 Case Studies

We have already have some success using the tools for debug. In particular, the simple register interface has proven to be an irreplaceable tool in solving difficult debug problems. We will present three case stud-

ies wherein the tools have proven useful.

Case 1. In our first debug challenge, we were experiencing intermittent register write failures. It was rare enough to disregard any obvious problem, and, at 30 MHz and easily meeting the timing constraints it was fairly unlikely the problem was in timing. We have seen some interesting problems where the language construct used to describe the array of std_logic vectors has resulted in measureably different behavior in certain conditions, and thus were not certain which of the gate arrays were to blame. In this case, we were able to connect to the system via JTAG, then write a set of test registers and read them back. This was working fine, so we moved to reading and writing in the other permutations in combination with the failing normal register system. This allows us to determine that the problem lay solely in the data connection between the two chips. The problem was solved through the replament of the grid array interposer, which have a limited number of cycles before they begin to fail.

Case 2. In our second problematic debug challenge, we struggled to explain the behavior of an application that worked fine when the FPGA was programmed via JTAG but certain application functions failed when programmed via the select MAP interface. The select MAP programming was successful, but it was clear something was being corrupted in the programming sequence. At this point, PROMs for both the FPGA bitfiles as well as the software PROM had been burned, toward an early deadline. This was important because it forced us to try to minimize the changes to either PROM. The situation was made even more intriguing by the fact that any attempt to dump state for debugging would cause the failure to disappear.

However, by connecting to the system through the JTAG interface, we were able to dump the registers and determine that one of the registers was corrupted between programming and the application being armed. We then forced an application break and dumped the memory location that was the source of the register write. The memory was corrupted as well. Eventually, we determined that the decompression routine was corrupting a single byte outside of its allocated space, which eventually was written to the FPGA. Because the JTAG interface provided access to FPGA application space, we were able to track down a software corruption on the other end of the PCI bus.

Case 3. In our development of a airborne persistent surveillance platform, we wished to develop multiple hardware components in parallel, namely, a high speed serial connection between a microprocessor and the FPGA, and the QDR SRAM interfaces. Without the microprocessor connection, we had no way to communicate with the FPGA to test the QDR interfaces. By connecting to the system via JTAG, we were able to move test images in and out of the QDRs without use of the normal flight communication path. While the JTAG connection is much slower, it allowed us to make progress where it otherwise be impossible.

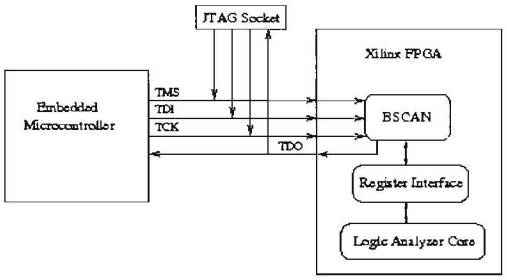

Figure 1. High-level FPGA connection to JTAG controller

Figure 3. Example capture from embedded logic analyzer using microcontroller-implemented JTAG controller

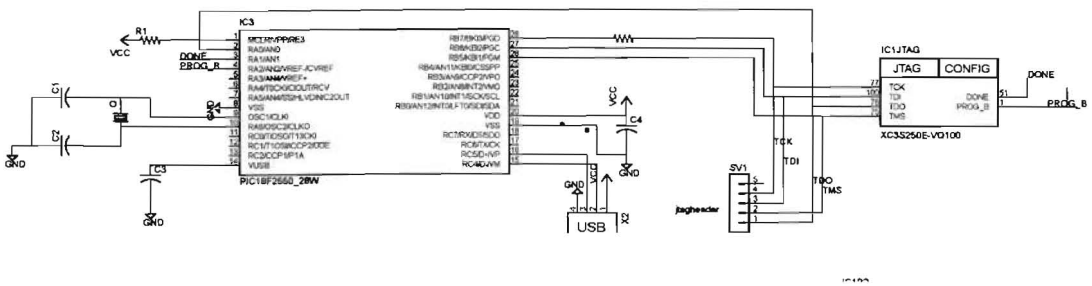

Figure 2. Schematic for use of PIC 18f2550 microcontroller with a general JTAG chain

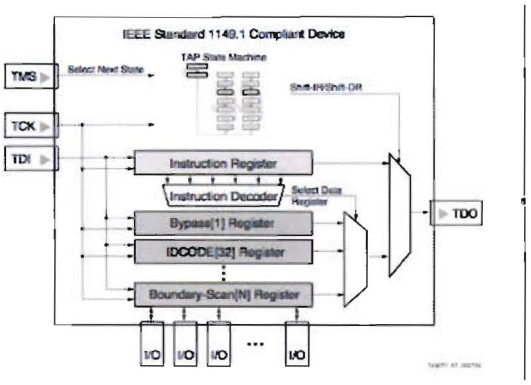

Figure 4. Xilinx internal boundary scan state machine for Virtex 4 [2]

5. Conclusions

In this paper, we present a system for in-situ debug using an on-board microcontroller acting as the JTAG controller. The system provides an approach for debugging difficult problems in remote environments. The system includes a variety of useful functionality, including:

- JTAG programming without an external programmer
- Readable/writeable address space
- Debug channel is separate from normal communication channels
- Linux command-line control over FPGA user fabric

Acknowledgements

We would like to acknowledge Joshua Monson's work on the JTAG user fabric.

References

[1] Using the JTAG Interface as a General-Purpose Communication Port, 2005. http://www.xilinx.com/support/documentation/user_guides/ug071.pdf.

[2] Xilinx UG071 Virtex-4 Configuration Guide, 2006. http://www.xilinx.com/support/documentation/user_guides/ug071.pdf.

[3] PIC18F2455/2550/4455/4550 Product Family, 2007. http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en010280.

[4] Xilinx In-System Programming Using an Embedded Microcontroller, 2007. http://www.xilinx.com/support/documentation/application_notes/xapp058.pdf.