Computer Structure Lab - Fall 2000
Dr. Guy Even

Handout #3: Built-In Self Monitoring

1 What is built-in monitoring?

Suppose you have designed your favorite design (i.e. a DLX processor), and simulation indicates that the design is correct. You upload the design to the FPGA and you let the design start running (i.e. the DLX runs a program that is stored in the memory). However, you do not see the results that you expected to see (assume that after a sufficient amount of time, you use the PC monitor program to read the RESA memory contents). What do you do then?

What we propose is not to be surprised by such an event, but to be prepared for it. Wouldn't it be useful to be able to run the DLX processor step by step and monitor (i.e. view) the values of registers and control signals? This sounds like standard debugging, and indeed, this is what we are after.

To enable such monitoring, our designs will include the hardware that will be responsible for reading various values in the DLX processor without changing the DLX behavior.

2 General Structure

Figure 1 depicts the general structure that we use for the built-in monitoring. The figure depicts only two components of the RESA: (a) The FPGA contains three modules: the application (e.g. the DLX processor), the Monitor Slave, and the I/O control logic. (b) The PC monitor program.

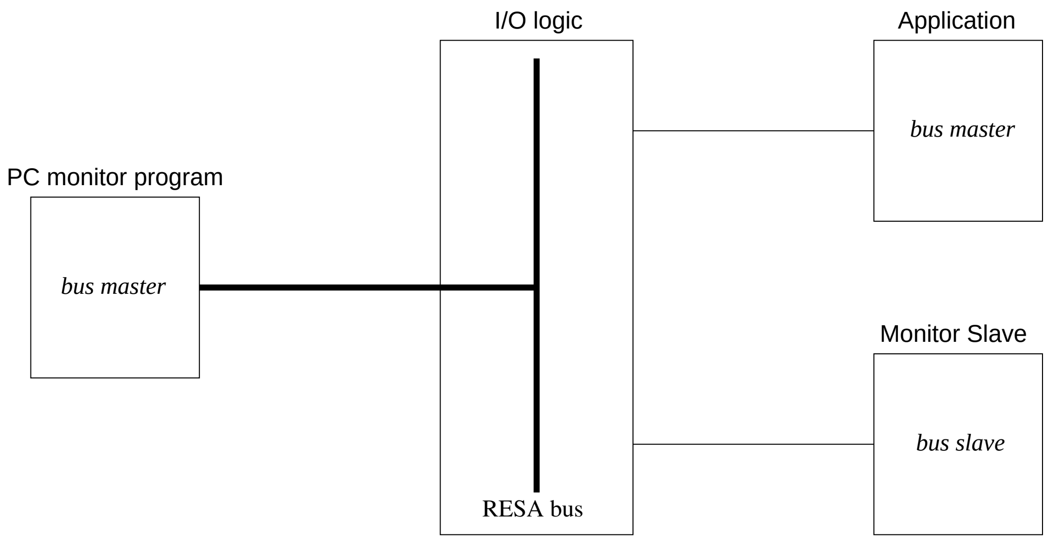

Figure 1: General structure of the monitor

The PC monitor program is a program that runs on the PC and behaves as a bus master that is controlled by you. Monitoring the application is achieved by initiating bus transactions that accesses a special slave device on the FPGA called the Monitor Slave. The Monitor Slave is a circuit that reads internal signals in the application, stores them, and sends them to the PC monitor program whenever requested to do so.

The FPGA is programmed so that it contains three components: the application, the Monitor Slave, and the I/O control logic. The I/O control logic gives the application and the Monitor Slave the abstraction that they are directly connected to the bus. The application is a bus master. The Monitor Slave is a bus slave, and is meant to participate only in bus transactions that are initiated by the PC monitor program.

In a normal execution (without monitoring), the PC monitor program does not initiate any bus transactions, and therefore, the Monitor Slave is idle.

In Assignment #2 you saw that implementing a Monitor Slave that can report current values of registers in the application is not a complicated task. It can be done as follows:

- Connect special wires from the registers in the application to a multiplexer in the Monitor Slave.
- The multiplexer selects an incoming signal based on the address in the read transaction.
- The selected signal is send as the data in the read transaction.

3 Monitoring Tasks

Before describing more complicated monitoring tasks, we need to describe the application. Our goal is to design a processor that executes DLX instructions. The control of the DLX starts each instruction execution in the "fetch" state, passes through a few other states and returns to the "fetch" state. We will have two modes: single-step and continuous. In single-step mode, each execution of an instruction waits until an appropriate signal arrives from the PC monitor program. In continuous mode, the execution of the next instruction is unconditional.

Suppose we wish to monitor signals and register values of the DLX. Consider the following cases:

1. If the DLX is running in continuous mode, we will get a "choppy motion picture". For example, we could try to monitor the value of the PC register (to know which instruction is executed). However, the read transaction might require more cycles than an instruction execution. That would mean that by the time the PC monitor program receives a value, it is no longer relevant. Moreover, consecutive reported values are likely to be non-consecutive values of the PC register. Such monitoring would provide only very partial information (e.g. is the value of the PC register changing over time?) but not full information (e.g. what is the sequence of executed instructions?).
2. If the DLX is running in single-step mode, we will be sampling signals only when the DLX is in the "fetch" state. This does not suffice. Consider, for example, the following situations:

- Monitoring the bus activity of the DLX. How can the Monitor Slave report signals related to bus transactions in which the application is a master? There seems to be a conflict since the Monitor Slave uses the bus to report values.
- Monitoring internal signals of the application over several consecutive cycles. The execution of an instruction in the DLX takes a few cycles (e.g. "fetch", "decode", "execute", "write-back"). Even if the DLX is running in single-step mode, a Monitor Slave that reports only current values is limited to reporting the signals during the "fetch" state.

Our conclusion is that reporting current values is not enough for debugging a design. The purpose of this handout is to design a Monitor Slave that can monitor signals from the application and store the sampled values during a few consecutive clock cycles. These stored values can be later reported to the PC monitor program.

4 The Logic Analyzer

The part of the Monitor Slave that stores past signals is called the Logic Analyzer. The Logic Analyzer contains a RAM in which the monitored signals are stored. These sampled values can be reported later to the PC monitor program. This implementation enables us to monitor signals during an instruction execution. (Reminder: An instruction execution is the interval of clock cycles between two consecutive entries to the "fetch" state.) If we want to monitor control signals and register values during the execution of a single instruction, we should be able to do the following:

1. Store the monitored signals cycle by cycle during the execution of an instruction.
2. After the instruction's execution is completed, be prepared to answer bus read transactions in which the PC monitor program asks about the sampled values.

The structure of the Logic Analyzer and its connections with the Monitor Slave and the application is depicted in Figure 2. The Logic Analyzer consists mainly of two parts:

1. The Logic Analyzer's RAM. This is a 32×32-bit RAM that stores the sampled values from the application. In each clock cycle, up to 32 signals (i.e. bits) can be stored.
2. The counter. The 5-bit counter generates the address into which sampled values are stored. In the beginning of an execution of an application's instruction, the counter is reset by the application. The counter's output value equals the number of cycles that have elapsed since the beginning of the last instruction. This way, the Logic Analyzer's RAM is filled "row by row" with the sampled values until the execution of the current instruction ends. The Monitor Slave can read the value of the counter and send it to the PC monitor program, so that the PC monitor program knows how many queries to make.

The Monitor Slave has two registers that are related to the Logic Analyzer.

1. The Command Register. This 32-bit register is divided into two fields.

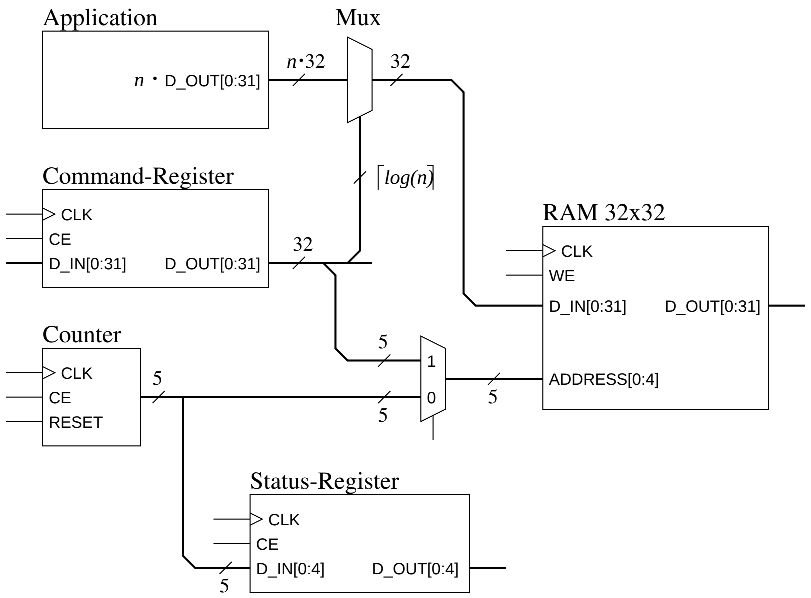

Figure 2: The structure of the Logic Analyzer and its connections with the Monitor Slave and the application.

(a) The bottom 5 bits are used to address the Logic Analyzer's RAM when the PC monitor program reads the sampled values (Recall that the address corresponds to the number of cycles that have elapsed since the beginning of the execution of the monitored instruction). Note that the PC monitor slave needs to be able to write values to the Command Register. This means that the Monitor Slave should also support write transactions.

(b) The top bits are used to select which 32-bit data set is routed from the application to the data-in input of the Logic Analyzer's RAM. Again, this bits are written by the PC monitor slave.

2. The Status Register latches the value of the Logic Analyzer's counter so that the Monitor Slave can report the number of rows that contain relevant data in the Logic Analyzer's RAM. Note that the output of Logic Analyzer's counter is registered. This means that one does not really need a new register to store the counter's output since it suffices to deactivate the counter enable input of the counter.

5 Pre-Lab Assignment

hand-in the pre-lab assignment in the beginning of the 5th lab meeting
Consider an application of a 5-bit counter which alternates between two modes: counting down from 20 to 0 (staying with 0 until the next step-enable signal) and counting down from 10 to 0 (staying with 0 until the next step-enable signal). A step-enable signals causes the counter to reset and start counting. Our goal is start sampling the state of counter starting with rising edge of the clock that occurs after the rising edge of the step enable signal. The sampling should stop when the counter is "stuck". The application generates a signal called

in_init. When in_init is high, the counter's value equals 0 (i.e. the counter is "stuck" and sampling by the Logic Analyzer should stop).

1. (5 pts) Which control signals should the application transmit to the Monitor Slave (including the Logic Analyzer)? Differentiate between signals that are monitored by the Logic Analyzer, and signals that aid the Monitor Slave and the Logic Analyzer in functioning properly.

2. (10 pts) Write the equations for the following signals:
   (a) Write enable (WE) signal of Logic Analyzer's RAM.
   (b) Count enable (CE) signal of the Logic Analyzer's counter.
   (c) Clock enable (CE) signal of the Status Register.
   (d) The reset signal of the Logic Analyzer's counter.
   (e) The Select signal of the mux that selects the address input of the Logic Analyzer's RAM.

3. (25 pts) Extend the Slave design from the previous handout to support (a) read transaction from the Status Register; (b) read transactions from the Logic Analyzer's RAM; (c) write transactions to the Command Register. Submit a hand-written design.

   Remark: a configuration file for the RESACTRL program will be posted on the lab's web-site. Use the same addresses that appear in that configuration file so that your design is compatible and to simplify comparison of the monitored wave-forms.

6 Lab Assignment

submit in the beginning of the 6th lab meeting.

1. Implement, configure, and run project "ex3" on the RESA. Monitor the states of the counter and verify that it functions properly.

2. (60 pts) Replace the Monitor Slave macro in project "ex3" with your own design. Submit printouts of your design, simulation wave forms, and wave forms monitored by the RESACTRL program.

   Warnings and reminders:

1. **This exercise requires using the RESA. Do not touch it! The RESA is connected to the PC and all the operations should be done via the PC control program (including reset).**

2. **Do not use drivers in your design!!! Conflicting drivers (due to design errors) can cause hardware damages.**

3. **Follow the following design rule: All registers should be clocked by the same clock. The registers differ in the clock enable signals.**

4. **You will need your Monitor Slave design for all your future lab assignments. Spend some extra time making sure it is well designed.**