Computer Structure Lab - Fall 1998
Dr. Guy Even

1 Handout #2: an application with built-in monitoring (v 1.0)

The purpose of this handout is to describe how to use the FPGA to design a very simple design (i.e. a counter). We will also include some non-trivial hardware to enable built-in monitoring.

The steps of this assignment are:

1. Designing a counter with a Monitor Slave (including simulation).
2. Uploading the design to the FPGA chip.
3. Using the monitor program to test the design.

2 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 a PC 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.

3 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 CPU board which has three parts: the application (e.g. the DLX processor), the Monitor Slave, and the I/O control logic. (b) The PC monitor program.

The PC monitor program is a program that accesses (via bus transactions) the Monitor Slave, and asks it to read data from the application. This data is sent back to the PC monitor program, and is displayed so that the user can view these values.

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 bus master, which means that it is the master in all the bus transactions that it participates in. The Monitor Slave is a bus slave, and participates only in 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.

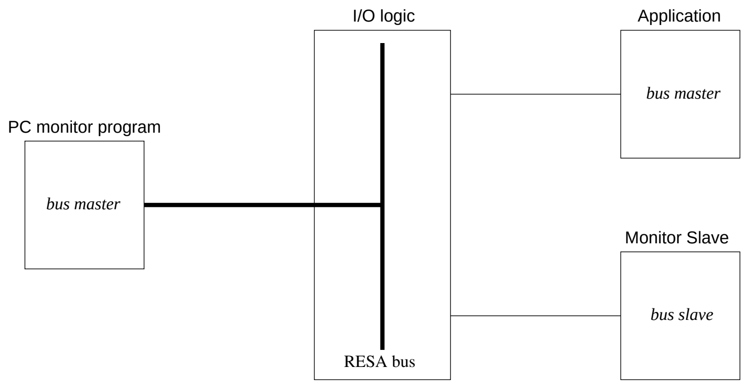

Figure 1: General structure of the monitor

The PC monitor program enables the user to monitor the execution of the application. That means that bus transactions are initiated by the PC monitor program. In these transactions, the PC monitor program requests data from the Monitor Slave. The Monitor Slave stores values that it reads off from the applications, such as: control signals and register values. These stored values are sent back to the PC monitor program.

An interesting question that should be raised at this point is: How can the PC monitor be used to monitor 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. Obviously, we must have an ability to monitor the application's bus activity.

4 The PC Control Program

The PC control program is a suite of programs which is used to diagnose and setup the RESA. We describe the parts of the PC control program which will be used by you.

1. Configure RESA-CPU board. This is how the FPGA is programmed. There are two options. In the first option, the design is read from the EPROM which is placed on the processor board. In the second option, the design is read from a file located in the PC.
2. Access RESA-Memory board. Data can be read and written to the main memory of the RESA. This is useful mainly to read blocks of memory after an application on the RESA completes its execution.
3. Run and Debug. The part has a few roles.
   (a) Upload application programs (these are programs stored in the RESA's main memory that the application runs) and application data to the main memory.

(b) Set single-step mode or continuous mode. If the application functions properly, then in single-step mode, the application executes only one "instruction" and waits for the monitor program to allow it to execute the next "instruction".

(c) Run the PC monitor program. The PC monitor program enables one to request data from the Monitor Slave. All data transferred between the Monitor Slave and the PC monitor program is given addresses (which are invented during the design of the Monitor Slave). For example, in a DLX design, one may decide that the address of the PC-register is 128, the address of IR register is 129, etc. When the PC monitor program wishes to view the value of the PC register, it initiates a read transaction in which it requests the Monitor Slave to send the value whose address is 128.

The user sets the values to be monitored using a configuration file. In the configuration file, the user writes the addresses of the requested data. To make things readable, labels are given to the addresses, so that the PC monitor program can display the PC-register value using the label "PC" rather than the address "128". The Monitor PC program has an interactive mode in which the requested signals can be viewed. A sample configuration file is appended to this handout (you may ignore now the statements there about the "Logic Analyzer").

It is important to note that the PC monitor program behaves as a bus master and can initiate bus transactions (both read and write). This means that the user can use the PC monitor program to access data not only via the Monitor Slave (which is part of the FPGA design) but also to access data in the memory board. Since the bus protocol is address based, access to different components is achieved by the header of the address. The address is divided into fields: the leftmost field is used to decide which board is accessed, and so on.

5 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. Figure 2 depicts the I/O control logic. The RESA bus is drawn within the I/O control logic since both the application and the Monitor Slave access the RESA bus via the I/O control logic. The I/O control logic is divided into two parts: one which serves the application and the other serves the Monitor Slave.

We start by describing the outputs and inputs of the I/O control logic that serve the application. The output signals to the application are:

1. `clk`: This is the clock signal of the application.
2. `step_en`: This signal notifies the application that it should run in single-step mode. If the application is a counter, then the `step_en` signal is a simple clock enable signal. If the application is a DLX processor, then the `step_en` signal tells the DLX control whether a new instruction should be executed or not.
3. `reset`: The reset signal resets the application.
4. `ack`: The acknowledge signal is sent by the slave in a bus transaction. This signal is active low. The design entry tool is not capable of handling signal names starting

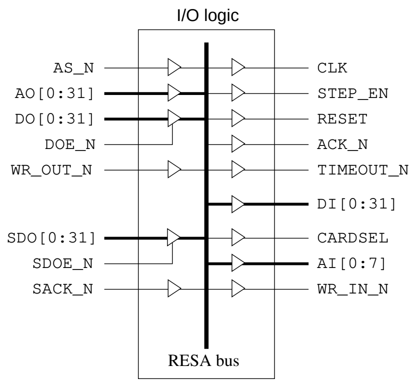

Figure 2: The I/O control logic

with the symbols "/" and "!". Thus naming the active low acknowledge signal `/ack` is not possible. Therefore we denote the polarity of active low signals by adding the appendix "_n" to the signals names.

5. `timeout`: The timeout signal is generated by the bus controller to indicate that a bus transaction failed. This signal is active low. Whenever a bus transaction starts, the bus controller starts counting. After a certain amount of time (128 cycles), if the bus transaction is not over, then the bus controller terminates it by generating an `ack` and by generating a `timeout`. The `ack` signal causes the transaction to end, and the `timeout` signal tells the master that the transaction failed. A situation where such an intervention by the bus controller is necessary, is when the master tries to access an address which does not correspond to any device on the bus. When such an access occurs, no device will respond, and unless the bus controller kills the transaction, the transaction will never end.
6. `DI`$[0:31]$: This is the data-in bus of the application and the monitor (they share this bus).

The input signals from the application are:

1. `as`: This is the address-strobe signal generated by the application. This signal is active low.
2. `AO`$[0:31]$: This is the address-out bus generated by the application.
3. `DO`$[0:31]$: This is the data-out bus generated by the application.
4. `DOE`: This signal is the data output enable signal. It means that when this signal is low, the application wishes to write the `DO` signal to the RESA data-bus.
5. `wr_out`: This is the `wr` signal generated by the application. This signal is active low.

We now describe the outputs and inputs of the I/O control logic that serve the Monitor Slave. The output signals to the Monitor Slave are:

1. `clk`: This is the clock signal of the monitor (this signal is shared with the application).
2. `DI`$[0:31]$: This is the data-in bus of the application and the monitor (they share this bus).
3. `cardsel`: This signal indicates that the slave of the current transaction is on the CPU board. This signal is computed by the I/O control logic from the upper bits of the address in the RESA bus.
4. `AI`$[0:7]$: This is the address-in bus of the Monitor Slave. We use the convention that the address space of the Monitor Slave is restricted to addresses with `AI`$[7]=1$.
5. `wr_in`: This is the `wr` signal that is input to the Monitor Slave. This signal is active low.

The input signals from the Monitor Slave are:

1. `SDO`$[0:31]$: This is the data-out bus generated by the Monitor Slave.
2. `SDOE`: This signal is the data output enable signal. It means that when this signal is low, the Monitor Slave wishes to write the `SDO` signal to the RESA data-bus.
3. `sack`: This is the `ack` signal generated by the Monitor Slave. This signal is active low.

6 The Monitor Slave

Suppose the application is a 32-bit binary counter. The Monitor Slave for a counter design is a very simple device. There is one address that selects the counter output. When the PC monitor program wishes to read the counter's value, it initiates a read bus transaction with the address of the counter's output. The Monitor Slave receives this request, and routes the counter's output to the `SDO`-bus. The Monitor Slave then acknowledges that the requested data has been sent.

Note that the PC monitor program can read the counter's value both in single-step mode and in continuous mode. What would you expect these readings to be in both modes?

The Monitor Slave for a more complex design, such as the DLX processor, is more complex, and will be dealt with in future handouts.

7 Homework

You will be examined orally on Tuesday 22.12.98. Written homework should be handed in by Wednesday 23.12.98

1. Read about Xilinx FPGA in the data book. Available in the APPLinx CDROM: databook.pdf

   (a) What is a CLB? what is an IOB? How are the signals routed in an FPGA?
   (b) Show how a single CLB (depicted in Figure 2 page 4-10 (page 128)) can be configured to behave as:
      i. a register without using the function generator in the CLB;
      ii. a combinational circuit with one output (Y) and 4 inputs using only one function generator;
      iii. a registered signal (XQ) with 4 inputs. (A registered signal is a finite state machine whose state depends only on the previous input values); and
      iv. two independent combinational circuits each with 4 inputs.
      v. one combinatorial circuit with 9 inputs.
   (c) Show how a single IOB (depicted in Figure 16 page 4-22 (page 140)) can be configured to behave as:
      i. a combinatorial input signal;
      ii. a registered input signal; and
      iii. a tristate (combinatorial) output with an output signal and an output enable signal;
   (d) Prove or refute: It possible for every Boolean function $f:\{0,1\}^8 \rightarrow \{0,1\}$ to configure a single CLB so that the CLB computes f.
   (e) Prove or refute: It possible to use IOBs to implement a wired-OR signal.
   (f) What are the differences between the different types of lines (wires)? See Figure 26 on page 4-30 (page 148).

2. Design using ABLE a 32-bit binary counter and simulate the counter. The counter should have the following inputs and outputs:
   (a) `clk`: the clock;
   (b) `ce`: the clock enable, when high the clock is enabled, when low the clock is ignored;
   (c) `reset`: resets the counter value to zero;
   (d) `cnt`$[0:31]$: the output counter value.

   The functionality of the counter is as follows.

   (a) A clock pulse P is enabled if the clock enable signal is high during the duration of P.
   (b) A reset signal is caught in time t if: (a) `reset`$(t)=1$; (b) `ce`$(t)=1$; and (c) the clock undergoes a rising transition in time t.
   (c) Let $\delta(t_0,t_1)$ denote the number of rising edges of enabled clock pulses that occurred in the interval $(t_0,t_1]$.
   (d) Consider a time t. The time $t_{\texttt{reset}} \leq t$ is defined as the last time in which a reset was caught.
   (e) The binary number represented by `cnt`$[0:31]$ in time t equals $\delta(t_{\texttt{reset}},t) \bmod 2^{32}$.

(f) Bonus: think about the following point: how can you make the counter faster? Take into account that this is not a general adder.

3. Design a circuit that consists of the library component of the I/O control logic, the 32-bit binary counter, and the Monitor Slave. The task of designing consists of: design entry, simulation (for you - do not hand in simulations), and implementation. When you start the project (with the Xilinx software) specify the type of FPGA you will be using as: XC 4025 HQ240 speedgrade 2.

   Prepare a configuration file for the PC monitor program and execute your design with the PC monitor program.

Configuration file for the PC monitor program

```
// Example of a configuration file for the PC monitor program
// Description file *EXAMPLE* for IDLX

// CPU Board addresses
c PC        0x80
c IR        0x81
c DO        0x82
c AO        0x83
c AR        0x84
c ANALYZER  0x85
c Command   0x86
c Status    0x87

// Memory addresses
m CNT     0x80011a
m Var_X   0x800056
m Var_Y   0x800059

// Waveform bits
- specifying the control signals

w AS     0x07
w ACK    0x00

w ADD    0x03
w AluF0  0x04
w AluF1  0x05

w dcode  0x0b
w delse  0x0c
w DIsel  0x0d
w IRce   0x0f

- specifying the internal DATABUS

w DINT10 0x10
w DINT11 0x11
w DINT12 0x12
w DINT13 0x13
w DINT14 0x14
w DINT15 0x15
w DINT16 0x16
w DINT17 0x17
w DINT18 0x18
```

Remarks:

1. Order of declarations only effects the order in which the values are displayed.

2. (a) Addresses in lines beginning with a "c" are given address headers that correspond to the CPU board.

   (b) Addresses in lines beginning with an "m" are given address headers that correspond to the memory board.

   (c) Addresses in lines beginning with a "w" refer to the local RAM of the Logic Analyzer. This is a 32×32-bit RAM, and the address refers to a column in this RAM (i.e. the ith bit in each word).

   (d) Lines beginning with a letter not in $\{c, m, w, C, M, W\}$ are treated as comments (note that the first letter is not case sensitive).

3. Reserved labels are (not case sensitive): Analyzer, Command, Status. The meaning of these reserved labels is:

   (a) Analyzer specifies the address of the Logic-Analyzer (which is an FPGA-Slave). The Logic Analyzer contains a 32×32-bit RAM. We assume that the PC monitor program does not attempt to write to this RAM (only read transaction will be made with the Logic Analyzer by the PC monitor program).

   (b) Command specifies the address of a register in the Logic Analyzer. This register has two fields: the lower 5 bits serve as an address of a row in the Logic Analyzer's RAM. The upper bits of the Command Register may be used to select which data is routed to the data-in input of the Logic Analyzer's RAM.

   (c) Status specifies the address of a register in the Logic Analyzer that holds the number of rows of the Logic Analyzer's RAM that have been filled with fresh data. Loosely speaking, in single-step mode, the application executes "instructions" one by one. The execution of an instruction may take several clock cycles during which data is routed to the Logic Analyzer's RAM, filling it word by word as the clock cycles proceed. The Status Register will hold the number of cycles that have elapsed, so that the PC monitor program can know which rows of the Logic Analyzer's RAM hold relevant data.

4. Illegal lines will simply be ignored without reporting any errors!

5. Labels have a maximum length: c-type labels and m-type labels: 12 characters; w-type labels: 5 characters.

6. Valid address ranges are:

   - c-type: 0x80 - 0xff
   - m-type: 0x800000 - 0x8fffff
   - w-type: 0x00 - 0x1f

   All addresses MUST be specified in hexadecimal!