Computer Structure Lab - Fall 2000
Dr. Guy Even

Handout #2: A Simple Slave Device

In this assignment you will design a simple slave device. You will run your design on the RESA. We overview the structure of the RESA , the I/O CONTROL LOGIC, and the slave design.

1 RESA Architecture

The RESA Computer was designed for teaching how to design a CPU. Students can use the RESA as a concrete platform for designing a CPU and testing their design by running programs on it. The CPU of the RESA is based on a chip called an FPGA (Field Programmable Gate Array) that can be configured or "programmed" with regular signals without removing it from a board or disconnecting it.

Figure 1 depicts the parts of the RESA Computer. The RESA Computer consists of:

CPU board. The CPU board contains an FPGA which can be programmed to function as a CPU of the RESA .

Memory board. The memory board contains two types of memories: a read-only memory and a read-write memory. The type of memory that is accessed depends on the address. A read-only (non-volatile) memory which is based on EPROM chips (erasable programmable ROM) is used for storing a fixed program (the test program) which can be executed by the CPU after reset or power-up. A read-write memory based on DRAM (dynamic RAM) chips serves as the main memory of the CPU.

Backplane bus. All the cards are connected to the backplane bus. Every card has logic for controlling the communication via the backplane bus. The backplane bus can support upto 4 masters and 4 slaves, which means that upto 4 memory boards and 4 CPU boards can communicate via the the backplane bus.

Backplane bus controller. The backplane bus controller coordinates the communication over the bus. Due to engineering considerations, the connection between the RESA and the PC computer is implemented by a connection from an ISA special purpose card placed in one of the PC slots to the bus controller card.

2 The RESACTRL Program

The RESASTRL program is a suite of programs which is used to diagnose and setup the RESA . We describe the parts of the RESACTRL 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.

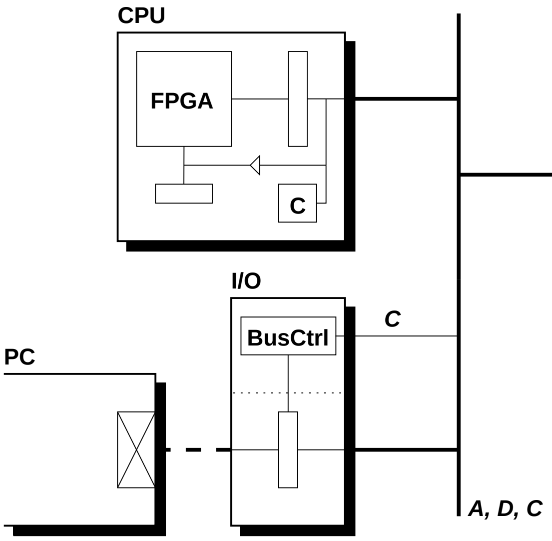

Figure 1: The RESA cards and buses

2. Access RESA Memory board. Data can be read and written to the memory board of the RESA. This is useful mainly to read blocks of memory after an application (e.g. a CPU) on the RESA completes its execution.

3. Run and Debug. The part has a few roles.

   (a) Upload programs. We often refer to the bus master as an application, for example, a CPU. Uploading a program means storing a program in the application's language in the memory board.

   (b) Set single-step mode or continuous mode. The purpose of this feature is to enable you to run your applications in single-step mode. This means that the application executes only one "instruction" and waits for the monitor program to allow it to execute the next "instruction". Between instructions, you can monitor various values in the memory and application. Usage of this feature requires that the application wait for a "start" signal before executing a new "instruction".

   (c) Run the PC monitor program. The PC monitor program enables one to initiate read and write transactions. This is a very powerful tool, as you will see later in the lab when we use a concept called "build-in monitoring". In this assignment you will be using the PC monitor program to initiate read transactions from the slave device that you are designing. This way you can see if your slave is functioning properly.

       The PC monitor program reads a configuration file (not to be confused with the HEX file used for configuring the FPGA). The configuration file of the PC monitoring program contains a list of physical addresses and labels attached to these addresses. You can use the PC monitor program to initiate read transactions that read the values of the signals and registers corresponding to these labels. A sample configuration file is attached to this handout (you may ignore in this assignment all the references to the Logic Analyzer).

3 The I/O control logic

In the previous assignment you designed a bus interface between the RESA bus and a bus master device. There is a risk in connecting devices designed by students to the bus, the reason being that design errors might cause drivers to be in conflict (e.g. two drivers connected to the same bus, one trying to pull the signal up, and the other trying to pull it down). The solution is to provide you with a bus interface module, called the I/O control logic. You will not be able to configure the FPGA unless your design uses this interface.

The I/O control logic serves as the bus interface for one bus master device and one bus slave device. 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 design environment does not enable you to access the RESA bus directly or even the FPGA's I/O pins. Hence, the "outer world" is hidden from you and the I/O control logic encapsulates the "outer world". The I/O control logic is divided into two parts: one which serves the bus master device and the other serves the bus slave device.

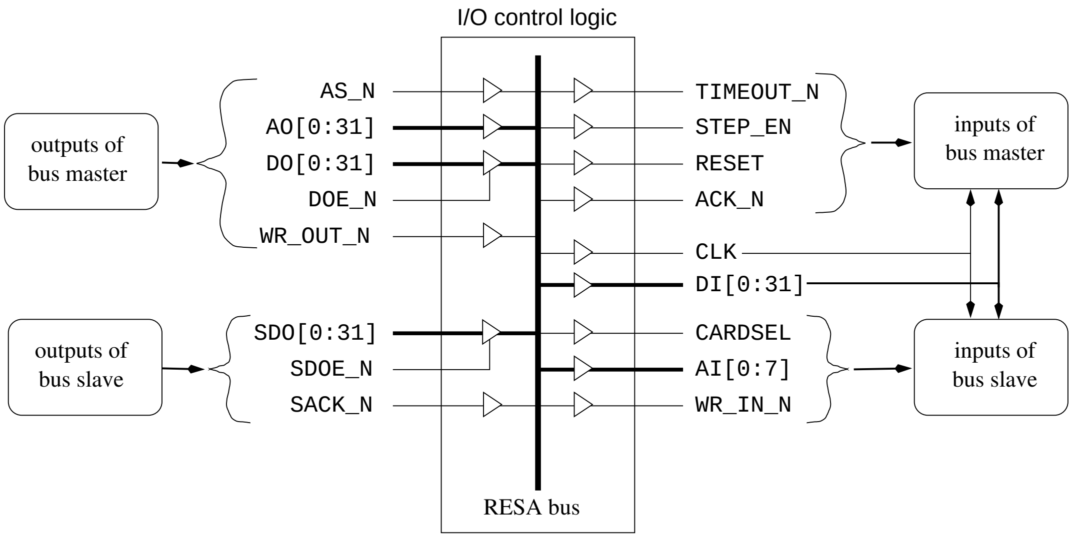

Figure 2: Connections between the I/O control logic and the master and slave devices on the FPGA

We start by describing the outputs and inputs of the I/O control logic that serve the bus master device. The following signals are fed to the bus master device:

1. `CLK`: clock.

2. `STEP_EN`: This signal notifies the bus master (e.g. a CPU) that it should run in single-step mode. If the bus master is a counter, then the `STEP_EN` signal is a simple clock enable signal. If the bus master is a DLX processor, then the `step_en` signal tells the DLX control whether a new instruction should be executed or not.

3. `RESET`: A reset signal.

4. ACK: The acknowledge signal is sent by the slave in a bus transaction. This signal is active low.[1]

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 (why is that undesirable?).

6. DI$[0:31]$: This is the data-in bus of the master and slave devices (both the bus master and the bus slave share these signals).

The following signals are fed to the I/O control logic by the bus master device:

1. AS: This is the address-strobe signal. This signal is active low.

2. AO$[0:31]$: This is the address-out bus (no sharing with the slave device).

3. DO$[0:31]$: This is the data-out bus (no sharing with the slave device).

4. DOE: This signal is the data output enable signal. It means that when this signal is low, the bus master application wishes to write the DO$[0:31]$ signals to the RESA data-bus.

5. WR_OUT: This is the WR signal generated by the bus master. This signal is active low.

We now describe the outputs and inputs of the I/O control logic that serve the bus slave device. The following signals are fed to the bus slave device:

1. CLK: clock.

2. DI$[0:31]$: This is the data-in bus (shared with the bus master device)

3. CARDSEL: This signal indicates that the slave of the current transaction is on the CPU board. Not all 32 bits of the address bus are fed to the slave device. The CARDSEL signal is active when the value of the address bus corresponds to the address of the CPU board. [2]. The CARDSEL 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 slave device. We will later introduce a special type of slave device called the Monitor Slave. We use the convention that the address space of the Monitor Slave is restricted to addresses with AI$[7] = 1$.

[1]The design entry tool is not capable of handling signal names starting with the symbols "/" and "!", which makes /ack an illegal name. Therefore we denote the polarity of active low signals by adding the suffix "_N" to the signals names.

[2]Note that this means that from the point of view of the I/O control logic there is only one slave device on the FPGA. You should be able to think of ways to partially bypass this restriction.

5. WR_IN: This is the WR signal that is input to the slave device. This signal is active low.

The input signals from the slave device are:

1. SDO$[0:31]$: Data-out bus.

2. SDOE: The data output enable signal. It means that when the SDOE_N signal is low, the slave device wishes to write the SDO$[0:31]$ signals to the RESA data-bus.

3. SACK: This is the ACK signal generated by the slave device. This signal is active low.

4 Simple Master and Slave Devices

In this assignment you will design a trivial master device and a simple slave device. The slave device is connected to nets of the master device by "private wires" (i.e. the private wires are not part of the bus). The values monitored by the slave device are reported in read transactions that the slave participates in. We will be using an extension of this mechanism to design applications (i.e. a DLX) with "built-in monitoring".

The master device we use for this assignment is a 32-bit binary counter. This master device does not initiate any bus transactions, and is therefore, a degenerated master device.

The slave device in this assignment reads the value output by the counter. This value is reported by the slave. The address space of the slave device consists of a single address, namely, the address that selects the counter's 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 slave device receives this request, and routes the counter's output to the SDO-bus. The slave device then acknowledges that the requested data has been sent, and the read transaction is completed.

5 Homework - Submit in the beginning of the 4th lab meeting.

Guideline for the whole lab: do not use tri-state buffers (i.e. drivers) at all in your designs!!! The justification for this is that a conflict of drivers may cause electrical damages to the circuit. Use MUXs instead of buffers.

When you start the project (with the Xilinx software) specify the type of FPGA you will be using as: XC 4025 HQ240 speedgrade 2.

1. Read about Xilinx FPGA in the data book. Available in the APPLinx CDROM: databook.pdf (Chapter 6).

   (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 VHDL 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) $\texttt{reset}(t) = 1$; (b) $\texttt{ce}(t) = 1$; 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 $\texttt{cnt}[0:31]$ 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 (i.e. shorten the propagation delay of the counter)? Take into account that this is not a general adder.

3. Design the slave device. The slave is connected to the $\texttt{cnt}[0:31]$ output of the counter.

4. Build a library component consisting of: the I/O control logic, the 32-bit binary counter, and the slave device.

5. Prepare a configuration file for the PC monitor program. Run your design on the RESA with the PC monitor program.

6. Submit your lab report one week later. Print your designs, configuration file, and simulations.

6 A Sample Configuration File

```
// 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 labels that correspond to the CPU board.

   (b) Addresses in lines beginning with an "m" are given labels 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 transactions 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!