EE 435 : Verilog Digital Systems Modeling

Project 4

Due Date: Wednesday April 8, 2020

1. In the first part of the project you are to model a RAM and a simple memory controller in Verilog. The depth of the RAM is 256. Each dataline in RAM is 32 bits (4 bytes) and the memory controller reads and writes all 4 bytes at the same time.

   Your memory controller sits between the CPU and the RAM. It receives read and write requests from CPU, and does a complete handshake with it. Then it sends the read and write requests to the RAM. Once the data is read from or written to the RAM, it signals the completion of the operation along with data (in case of a memory read) with a READY signal to CPU.

   The interface and Spec of the synchronous memory is as follows:

   Module name : **`RAM`**
   Data width : **`parameter DWIDTH = 32`**
   Address width : **`parameter AWIDTH = $clog2(MEMDEPTH)`**

   Input Ports (RAM):

   `Clk`
   `rdEn`
   `wrEn`
   `Addr`

   Bidirectional Ports:

   `Data`

   Note: **`rdEn`** and **`WrEn`** signals are active high, and are never asserted at the same time. The **`Addr`** is the address to read from or write to memory, and **`Data`** is the data to read from or write to memory. The Data port should have Z values whenever rdEn signal is deasserted.

   The memory controller has the following interface and Spec:

   Module name : **`MemControl`**
   Data width : **`parameter DWIDTH = 32`**
   Address width : **`parameter AWIDTH = $clog2(MEMDEPTH)`**

   Input Ports:

```
Clk
reset
Addr_in (16 bits) // from CPU
RW                //  from CPU
Valid             //  from CPU
```

Output Ports:

```
rdEn            // to RAM
wrEn            // to RAM
Addr            // to RAM
Ready           // to CPU
```

Bidirectional Ports:

```
Data_in         // from & to CPU
Data            // from & to RAM
```

Communication with CPU:

During a **write** operation, CPU places the memory address to write to on **`Addr_in`** and the data to be written on **`Data_in`** ports, then **`RW`** is de-asserted (**`RW = 0`**) and **`Valid`** is asserted (**`Valid`** = 1). The memory controller deasserts **`Ready`**, performs any address calculations and communicates with RAM through appropriate signals write the data to the memory. It takes 3 cycles for the write to be completed. So, after 4 cycles the operation is complete, so the memory controller asserts **`Ready`** to indicate to CPU that the write operation is done. At this time CPU deasserts **`Valid`**, then it can initiate a new read/write from/to the memory.

During a **read** operation, CPU places the memory address to read from on **`Addr_in`** port, then **`RW`** is asserted (**`RW = 1`**) and **`Valid`** is asserted (**`Valid`** = 1). The memory controller deasserts **`Ready`**, performs any address calculations and communicates with RAM through appropriate signals to read data from the memory. It takes 3 cycles for the read to be completed. So, after 4 cycles, the memory controller asserts **`Ready`** to indicate to CPU that the read is complete and data is ready on **`Data_in`** for CPU to read in case of a read. At this time CPU deasserts **`Valid`**, then it can initiate a new read/write from/to the memory.

Communication with RAM:

During a **write** operation, the memory controller reads the memory address to write to and the data to be written from the CPU interface and places them on **`Addr`** and **`Data`** ports, then it asserts **`rdEn`**, and keeps **`wrEn`** de-asserted (**`rdEn = 1`** and **`wrEn = 0`**). The write operation takes 3 cycles to complete. Hence the memory controller waits 4 cycles before asserting **`Ready`** to indicate this to CPU. The memory controller needs to keeps track of write delays to ensure correct operation of the design.

During a **read** operation, the memory controller reads the memory address to read from rom CPU interface and places it on **`Addr`** port, then it asserts **`wrEn`**, and keeps **`rdEn`** de-asserted (**`wrEn = 1`** and **`rdEn = 0`**). The read operation takes 3 cycles to complete. Hence the memory controller waits 4 cycles before reading the data from the RAM interface (**`Data`**) and writing it to the CPU interface (**`Data_in`**), then asserting **`Ready`** to indicate to CPU that data is ready. The memory controller needs to keeps track of read delays to ensure correct operation of the design.

You are to write a Verilog model for implementing the RAM and the Memory Controller (40%), then write the testbench that acts as the CPU. To test your memory controller, you need to write several values to the same or different addresses, then read from these addresses (20%) in different order. You need to submit any block diagrams, state transition diagrams and explanations in your report. Your Verilog code should include a header file with all the necessary information and be well commented. You need to provide waveforms that show your design works correctly. (10%)

2. You are to expand the design of your ALU. For this phase of the project, your ALU must be able to perform more arithmetic and logic operations of the β microprocessor including the multiplication and division. (10%).

3. You are to complete the design of the data-path of the CPU, buy adding a 32-bit Instruction register (IR), a 32-bit Memory Data Register (MDR) and a 10-bit Memory Address Register (MAR), then making interconnections to complete the design of the data-path. (20%)

   The IR holds the instruction to be decoded with the fields for Opcode, up to 2 source operands, one destination operand, or a small constant.

   The two registers MDR and MAR provide the interface to the memory controller.