

ECE 474a/574a Programming Project 2 – FSM+D Interface to Memory Module (50pts)

Due October 12, 2009 (Monday) at 8:00 pm

DELIVERABLES

- Text copy of the FSM+D circuit, specifically the FSM used to define the controller and the datapath components/connections
- Datapath module - behavioral model of each component within the datapath (registers, adders, comparators, etc.), structural connection of those components to create the datapath module
- Controller module - behavioral description of the FSM
- Count_Value module - structural connection of the controller and datapath
- Top_Circuit module - structural connection of Count_Value and Memory module
- Testbench - verify proper functionality of Top_Circuit

Additional ECE 574A submissions/functionality

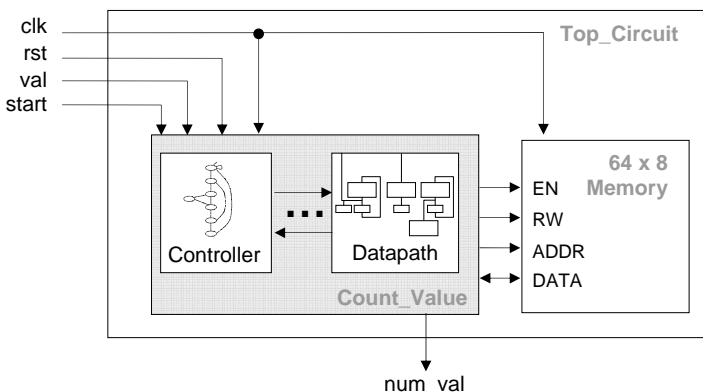
- Extended Count_Value functionality to support user-defined search space (starting and ending memory address)
- Increase the memory to 128x8 using the 64x8 memory modules provided (again the 64x8 modules should not be changed), and include any logic/controller required to interface to the larger memory
- After the search is completed write num_val to Mem[127]

PROJECT DESCRIPTION

In this project you will implement a circuit which reads all elements of a 64x8 memory and determines the numbers of entries within the memory equal to val, a user-specified 8-bit value. A 1-bit input start indicates when to begin the search, once the entire memory has been scanned a 7-bit output num_val indicates the number of entries found to be equal to the user specified value. num_val should only output the number of matching entries when the search is completely done, and continue to output this value until a new search is initiated.

A memory module is provided along with an input file specifying the contents of the memory (download from course page). You must interface to the memory without changing the provided memory interface. In developing a variety of systems you will often need to interface with code that others have previously developed or existing within a library. Module mem has inputs EN, RW, ADDR, and bidirectional bus DATA. DATA is a 8-bit wide set of data lines that can serve as input lines during a write operation, and output lines during a read operation. ADDR is a 6-bit input serving as the address lines during read and write operations. EN is a 1-bit control input that enables the memory for reading or writing. When EN=0, the memory is not active and will output a “Z” value on the DATA line. RW is a 1-bit control input that specifies whether the present operation should perform a read or a write. When RW=0, a write operation is performed storing the value located on the DATA bus at MEM[ADDR]. When RW=1, a read operation is performed outputting the contents of MEM[ADDR] on the DATA bus.

The Count_Value circuit interfaces to the memory module reading each entry within memory and determining if that entry is equal to val. The Count_Value circuit is constructed using a controller and datapath (an FSM + D implementation). The controller interfaces to the datapath (and memory module), orchestrating the components within the datapath as well as the interaction between the datapath and



```

module adder(...);
    // behavioral description of adder
endmodule

// other datapath components needed?

module dp(...);
    // instantiate and connect datapath components
endmodule

module cntrl(...);
    // define controller functionality
endmodule

module Count_Value(...);
    // instantiate and connect the controller
    // and datapath
endmodule

module Top_Circuit(...);
    // instantiate and connect the Count_Value
    // circuit to the memory module
endmodule

```

Figure 1: Block diagram and skeletal code for Top_Circuit. Note, the module declarations do not necessarily need to be in the same file.

memory. You should begin with a high-level state machine to describe the behavior of `Count_Value`, then using the RTL design method derive the corresponding FSM and datapath. For each component type in the datapath you will need to implement a separate Verilog module. The functionality of these components can be implemented behaviorally, and then connected structurally to create the datapath module. The final `Count_Value` circuit is implemented by structurally connecting the controller and datapath.

The `Top_Circuit` module implements the desired system by structurally connecting the `Count_Value` circuit described above to the memory module provided. You will need to provide a testbench to verify the functionality of `Top_Circuit`. You may also want to change the input file specifying the initial contents of the memory to test various scenarios.

ECE 574A ONLY

Update the `Count_Value` circuit to enable users to specify the starting and ending address searched within the memory. Additionally, update the `Top_Circuit` module to support a 128x8 memory module. You will need to utilize the existing 64x8 module provided to create the larger memory. You cannot change the existing 64x8 memory module, thus you may also need to incorporate additional logic or a controller within the memory to interface between the data received from the `Count_Value` circuit and the larger memory. Lastly, `MEM[127]` is not a valid data value. Rather this memory location stores the `num_value` determined by `Count_Value`. After the `Count_Value` circuit is finished searching specified memory locations, this circuit will output `num_value` as well as write the same value to `MEM[127]`. You may assume the end user is aware that `MEM[127]` is not valid and will limit `start_addr` and `end_addr`.

Figure 2: Block diagram of modified `Top_Circuit`

