INF2270, building a simple memory control unit

P. Häfliger

March 16, 2010

Abstract

In this exercise you will have a closer look at a part of the control unit in an implementation of the von Neumann model.

A memory controller can be seen in the von Neumann model as part of the control unit. In the simple computer model presented in the lecture, the memory controller would be responsible to control the MBR and MAR. It will perform its task, quite similarly to the ALU, as commanded by the main part of the CU by means of an opcode. Thus, the main CU has a standard interface to the RAM through the memory controller and need not know the details of the RAM behind it.

Memory controllers were in fact often integrated circuits separate from the CPU to be more flexible with respect to address space: To handle a different address space size one might have to exchange the memory controller but not the CPU. The most modern designs, however have the memory controller integrated on the CPU mainly for speed sake.

The Task

Implement and simulate a small finite state machine (FSM) that acts as a memory controller: It receives input control signals ('CUopcode(1:0)' in the ISE project) from the 'main' CU telling it to either load a value from the RAM or store a value into the RAM. Our implementation is somewhat primitive, in so far as there is only a single data bus that connects the main CU's registers with the memory controller. This restricts the access time, since the memory address register (MAR) and the memory buffer register (MBR) cannot be accessed in parallel (but makes a nicer exercise resulting in a FSM with a few more states). The data bus connecting the memory controller with the registers of the main CU will be called 'CUregbus(7:0)' in the ISE project. Furthermore, the memory controller is connected with an address and a data bus to an SRAM and needs to generate the right sequence of control signals for accessing it using the MAR and MBR. Dependent on a the opcode input (CUopcode) it will generate either a sequence of signals to load or to write memory content or do nothing. This CUopcode could be directly 2 bits of the machine code instruction (but could also be derived by combinational logic from the machine code).

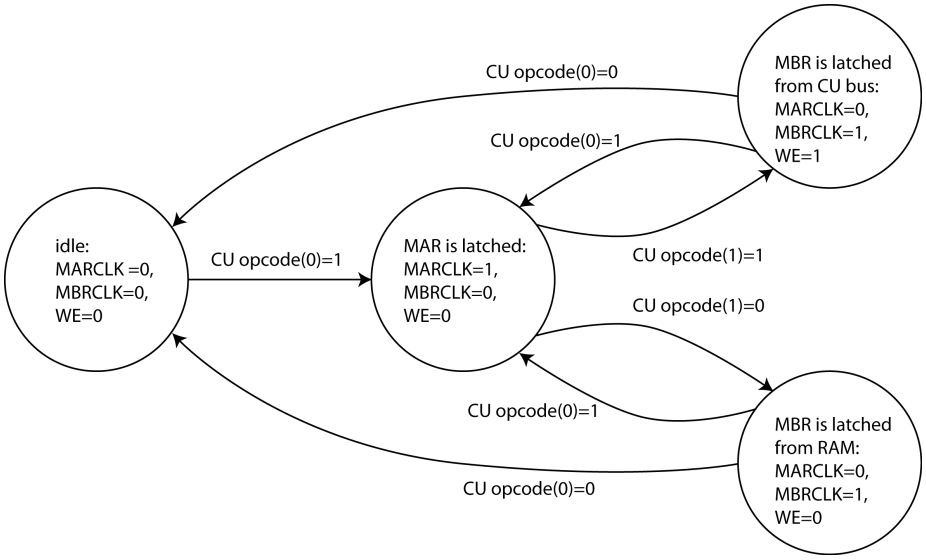

Figure 1: The finite state machine that defined the memory control unit.

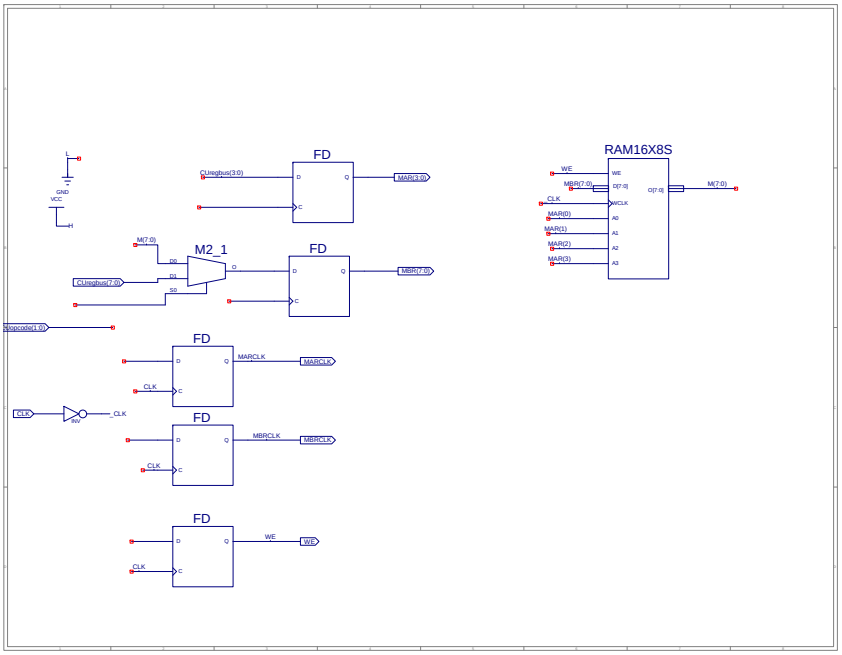

Figure 2: The ISE skeleton project to be completed with combinational logic.

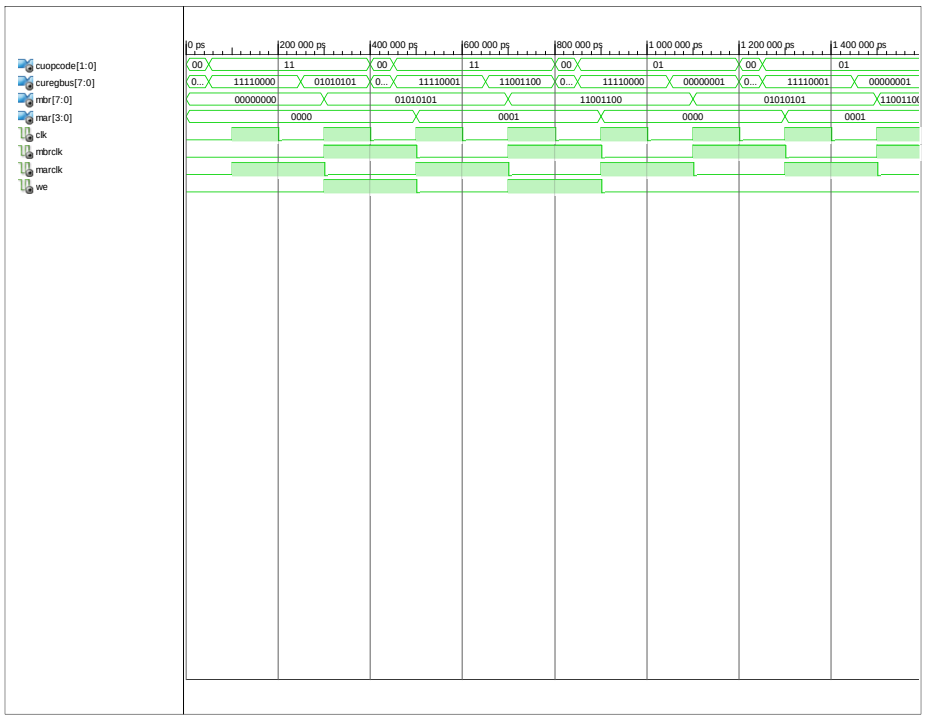

Figure 3: Target behaviour.

In ISE there are a number of predefined RAM cells. For this exercise the cell 'RAM16x8S' is used which is a simple RAM with a separate data input and output port (8 bit word length). The memory cell that is addressed is, thus, always visible on the output port. The address space is 4 bits. A control signal 'write enable' (WE) controls, if new data is written into the memory. The actual writing is triggered with the rising edge of the input signal 'WCLK'. If you want detailed information on the RAM, including a characteristic table, you may select the symbol tab in ISE and the cell 'RAM16x8S' symbol from library 'Memory' and click the button 'symbol info' (just above the tabs 'design' 'files' 'libraries' 'symbols' in the left column).

Furthermore there are two more 'high level' symbols that you will use for this task: a D-flipflop (symbol name: 'FD') from the library 'Flip_Flop' and a multiplexer (symbol name: 'M2_1') from library 'mux'.

The task of the state machine is as follows:

If the CUopcode=='00' :
Do nothing

If the CUopcode=='01' :
Load the MAR from the 'CUregbus' (1st clock cycle)
Load the MBR from the SRAM at the address stored in the MAR (2nd clock cycle)

If the CUopcode=='11' :
Load the MAR from the 'CUregbus' (1st clock cycle)
Load the MBR from the 'CUregbus' (rising edge of the 2nd clock cycle) and write the content of the MBR into the RAM at the address stored in the MAR (falling edge of the 2nd clock cycle).

To repeat this using RTL:

If the CUopcode=='00' :

If the CUopcode=='01' :
[MAR] $\leftarrow$ [CUregbus]
[MBR] $\leftarrow$ [M([MAR])]

If the CUopcode=='11' :
[MAR] $\leftarrow$ [CUregbus]
[MBR] $\leftarrow$ [CUregbus] ; [M([MAR])] $\leftarrow$ [MBR]

The FSM has four states and could be implemented using only a 2-bit state variable. However, it's conceptually easier to use the necessary control signals to control the RAM as state variables, such that no decoding will be necessary. Figure 1 shows the FSM that is one possible implementation for the memory control unit. The main CU will provide the control signal CUopcode (which will be held constant during two consecutive rising edges of the clock)and make

sure that the right registers are connected to the CUregbus at the right time during the two clock cycles it takes the memory controller to do its job.

In order to test your circuit you may design it with the ISE design suite (see the mandatory exercise oblig1.pdf for an introduction to the tool). For your convenience, a skeleton setup for your design can be found under

`~inf2270/programmer/memcontrol`

. Figure 2 shows the cell top.sch of the ISE project. You will need to derive the following FSM output signals and FSM state transition signals with combinational logic from the inputs and the state variables 'MARCLK', 'MBRCLK' and 'WE' (Note that there might be output control signals that is derived from an input signal, i.e. from the CUopcode, making this a Mealy FSM rather than a Moore FSM):

1. State transition signals
   (a) The inputs to the D-flipflops that hold the state variables 'MARCLK', 'MBRCLK', and 'WE'. Hint: The FSM in figure 1. Note, that you cannot only use the transition conditions on the arrows but you'll have to take into account also the starting state of those arrows. For example the input to a D-flipflop holding a state variable needs to be something like: this variable (that is high in state B) goes high if the state is A AND the transition condition A $\rightarrow$ B is met.
2. FSM outputs/control signals
   (a) The clock signals for the MAR and MBR (since the MBR and the MAR are composed of parallel D-flipflops, the clock signal also appears as a 8-bit bus. You will need to label it accordingly with 8 times your clock signal, e.g. myclk,myclk,myclk,myclk,myclk,myclk,myclk,myclk). Hint: two of the state variables have quite suggestive names, so this should be straight forward.
   (b) The control input for the bus multiplexer that routes either the CUregbus or the memory output to the MBR input.

The simulation will provide signals from the CU to:

1. write the number 10101010 into the memory at address 0000
2. write 11001100 at address 0001
3. read the memory at address 0000
4. read the memory at address 0001

Figure 3 shows the signals if the task is solved correctly.
Good luck!