

Verilog HW#4: Sorting Numbers in Memory



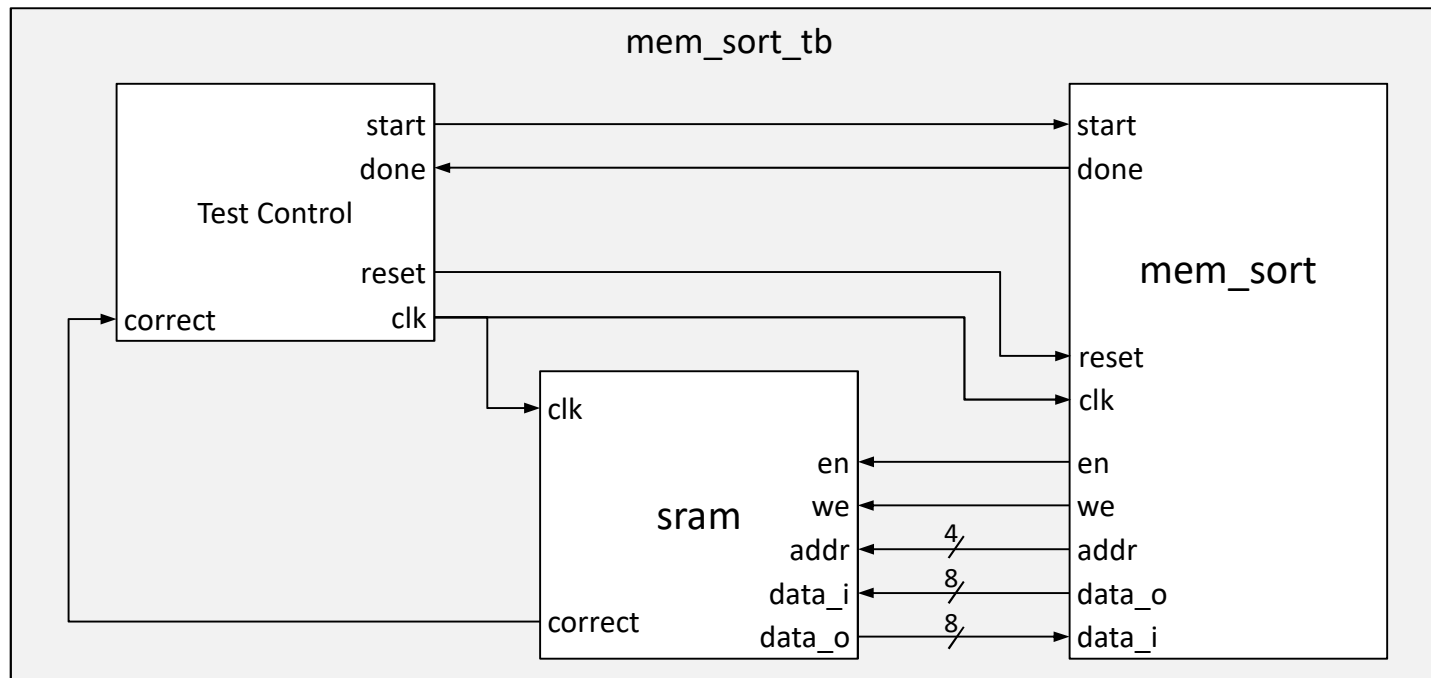
Chun-Jen Tsai
National Chiao Tung University
5/21/2021

Verilog HW#4

- ❑ Goal: Design a synchronous sequential circuit that reads sixteen 8-bit unsigned integers from a memory device, sorts the numbers in descending order, and writes back the sorted numbers to the memory device.
- ❑ Deadline: 6/7, 23:55pm. You must upload your Verilog modules to the E3 website by the deadline.

System Block Diagram

- ❑ The top-level system block diagram[†] are as follows:



- *start* is the 1-bit signal that triggers the sorting operation
- *done* is the 1-bit signal that signals the completion of sorting

[†]A sample testbench that includes a simple version with the `test_control` and the `sram` modules will be given to you.

Memory Module (1/2)

- ❑ The memory module `sram.v` will be provided.
You **SHALL NOT** modify its I/O ports:

```
module sram #(parameter DATA_WIDTH = 8, ADDR_WIDTH = 4)
  (input  clk,
   input  en, // Enable the memory device.
   input  we, // we == 1 for write operation.
   input  [ADDR_WIDTH-1 : 0] addr,
   input  [DATA_WIDTH-1 : 0] data_i,
   output reg [DATA_WIDTH-1 : 0] data_o,
   output correct);
```

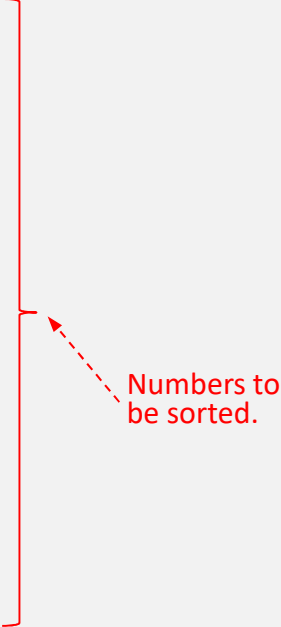
- *clk* is the system clock.
- *en* enables the memory device.
- *we* triggers the write operation. Reading is always active.
- *addr* is the address of the cell to be read/written.
- *data_i/data_o* are the input/output ports for data.
- *correct* is 1 if the memory cells are sorted in descending order.

Memory Module (2/2)

- ❑ The memory module has 16 8-bit memory cells, the SRAM block is initialized using the code on the right:
- ❑ Your circuit `sort.v` shall read the numbers from the memory cells, sort them in descending order, and then write them back to the memory cells.

```
// Declaration of the memory cells
reg [DATA_WIDTH-1:0] RAM[RAM_SIZE-1:0];

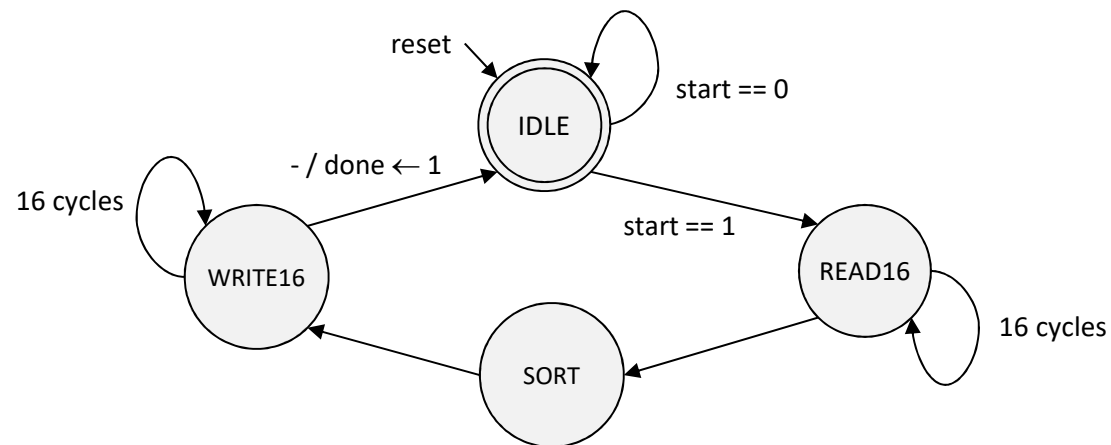
// -----
// RAM cell initialization
// -----
initial begin
    RAM[ 0] = 8'd34;
    RAM[ 1] = 8'd215;
    RAM[ 2] = 8'd122;
    RAM[ 3] = 8'd17;
    RAM[ 4] = 8'd77;
    RAM[ 5] = 8'd67;
    RAM[ 6] = 8'd63;
    RAM[ 7] = 8'd194;
    RAM[ 8] = 8'd139;
    RAM[ 9] = 8'd24;
    RAM[10] = 8'd71;
    RAM[11] = 8'd244;
    RAM[12] = 8'd246;
    RAM[13] = 8'd40;
    RAM[14] = 8'd247;
    RAM[15] = 8'd66;
end
```



Numbers to be sorted.

About the FSM in the `sort` Module

- ❑ For this HW, you can freely design your own FSM
 - For example, you can design an FSM to implement the C-style bubble sort algorithm literally
- ❑ Alternatively, a quick-and-dirty algorithm is as follows:
 1. Read 16 numbers from SRAM to a register array (16 cycles)
 2. Sort the 16 numbers using the technique in HW#2 (1 cycle)
 3. Write the 16 sorted numbers back to SRAM (16 cycles)



Requirements for Verilog HW#4 (1/2)

- ❑ The module you designed must be declared as follows:

```
module mem_sort#(parameter DATA_WIDTH = 8, ADDR_WIDTH = 4)
    (input  clk,
     input  reset,
     input  start,
     output done,
     output en,
     output we,
     output [ADDR_WIDTH-1 : 0] addr,
     input  [DATA_WIDTH-1 : 0] data_i,
     output [DATA_WIDTH-1 : 0] data_o);
    /* Implement your design here. */
endmodule
```

- ❑ You should only upload this module to E3.

Requirements for Verilog HW#4 (2/2)

- ❑ The reset signal should set the circuit state to IDLE and clear all internal registers to zero
- ❑ The circuit can be invoked repeatedly
 - If the *start* signal is activated again before the previous computation is finished, it shall be ignored