Verilog HW#4: Sequential Controller Design

Chun-Jen Tsai
National Chiao Tung University
6/1/2020

Verilog HW#4

- Goal: Design a synchronous circuit with a finite state machine that reads an 8 elements 8-bit signed integer array from its input port, computes the maximal sum of all its subsequences, and output the maximal sum through its output port.

- Deadline: 6/17, 23:55pm. You must upload your Verilog module to the E3 website by the deadline.

Maximal Subsequence Sum Example

- For example, if the input sequence is

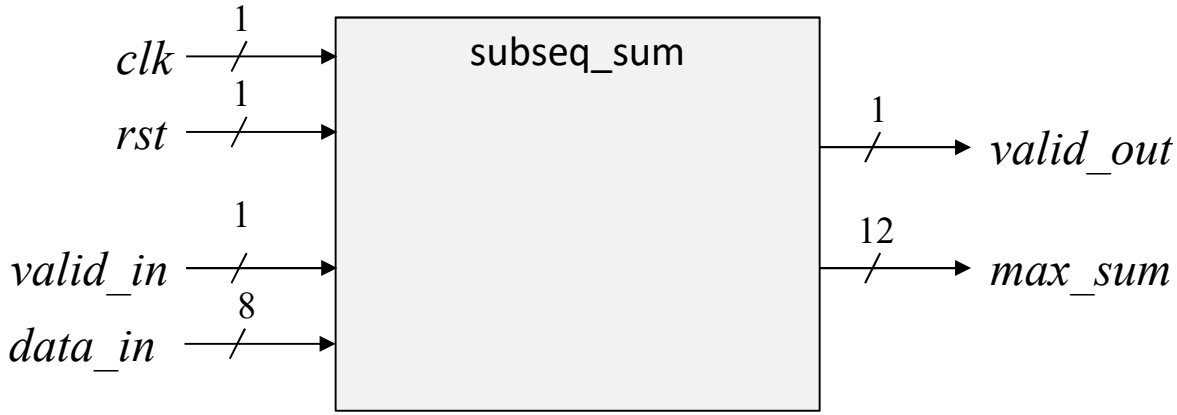
$$\{ -7, 1, -3, 2, -1, 1, 3, -5 \},$$

the maximal-sum subsequence sum should be the sum of $\{ 2, -1, 1, 3 \}$ which equals 5.

- Note that you can assume that at least one of the number in the input array will be positive, so the output is always a positive number.

Block Diagram of the Module

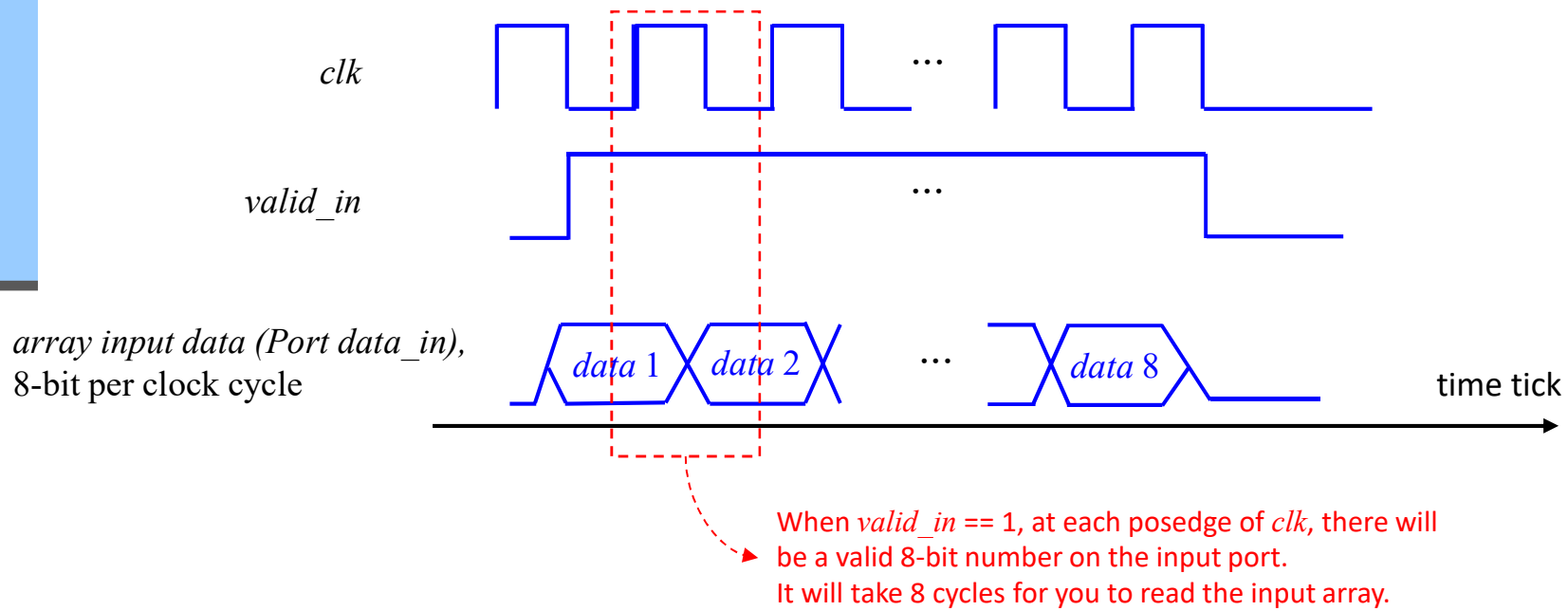
- Your module should be called subseq_sum, as follows:



  - *clk* is the 100MHz reference clock.
  - *rst* is the reset signal for the circuit.
  - *data_in* is the input ports for two 1×8 vectors.
  - *sum* is the output port for the maximal sum value.
  - *valid_in* and *valid_out* are the same as in HW#03.

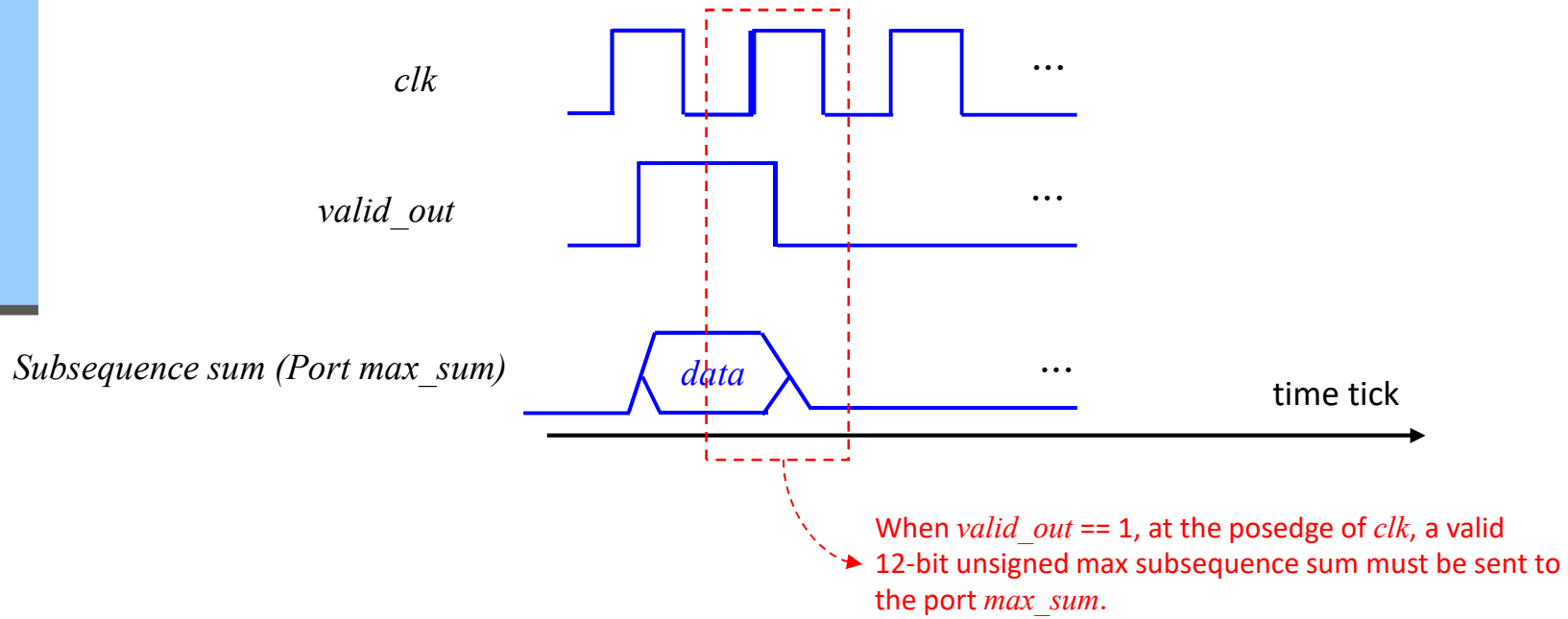
Input of the array

- The input interface is like that in HW#03, except you only have 8 signed 8-bit numbers to read
- The input must be read into a register array in your circuit for further processing



Output of the Max Sum

- After computing the maximal subsequence sum, your circuit should send the result to the port *max_sum*.
  - Your must raise the *valid_out* signal to inform the testbench the transmission of the output value:



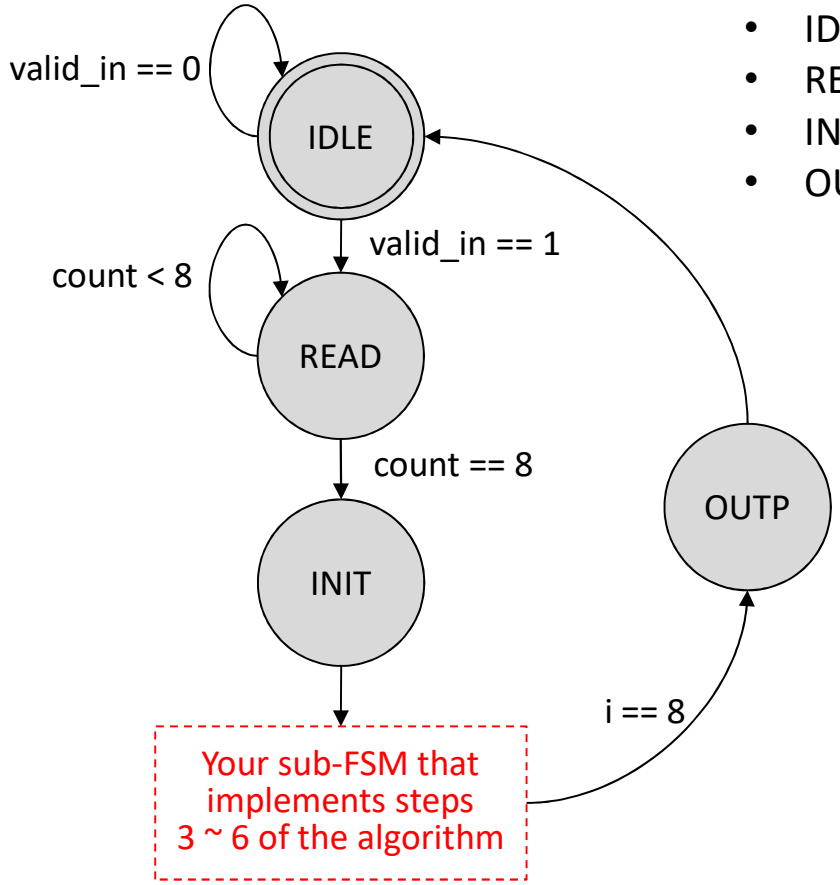
The Algorithm

- You must implement the following algorithm using a finite-state machine:

```
1. Read 8 numbers from input, and store them in n[].
2. Set i ← 0, partial_sum ← 0, max_sum ← 0.
3. partial_sum ← partial_sum + n[i].
4. if (partial_sum < 0) then partial_sum ← 0.
5. if (max_sum < partial_sum) max_sum ← partial_sum.
6. if (i < 8) then i ← i + 1, go to step 3.
7. Output the value max_sum.
```

Sample FSM

- A template of the FSM is as follows:



- IDLE: waiting for new input data
- READ: read input data into a register array
- INIT: i ← 0, partial_sum ← 0, max_sum ← 0
- OUTP: send max_sum to the output port

Requirements for Verilog HW#4 (1/2)

- The module you designed must be declared as follows:

```
module subseq_sum(input clk,
                  input rst,
                  input valid_in,
                  input signed [7:0] data_in,
                  output valid_out,
                  output unsigned [11:0] max_sum);

  /* Implement your design here. */
endmodule
```

- Your circuit must be able to perform subsequence sum repeatedly every time *valid_in* is set.
  - Note: the sample testbench TAs provided on E3 only tests your circuit once, you must try repeated testing by yourself!

Requirements for Verilog HW#4 (2/2)

- You can assume that once *valid_in* is set, it is guaranteed to last for 8 clock cycles.
- You can assume that at least one of the number in the input array will be positive.
- You shall not use for-loops in your module. All iterative operations shall be done through the FSM.
- You can design your own FSM freely, you do not have to use the template on page 8.