JOINT INSTITUTE
交大密西根学院

Ve270 Introduction to Logic Design

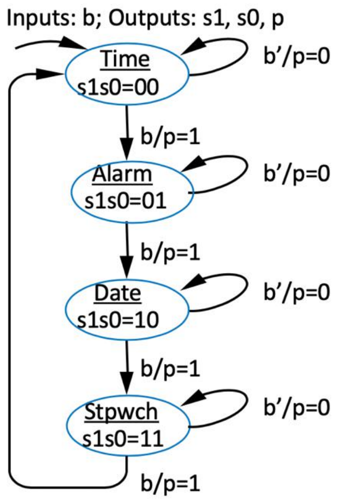
Homework 8

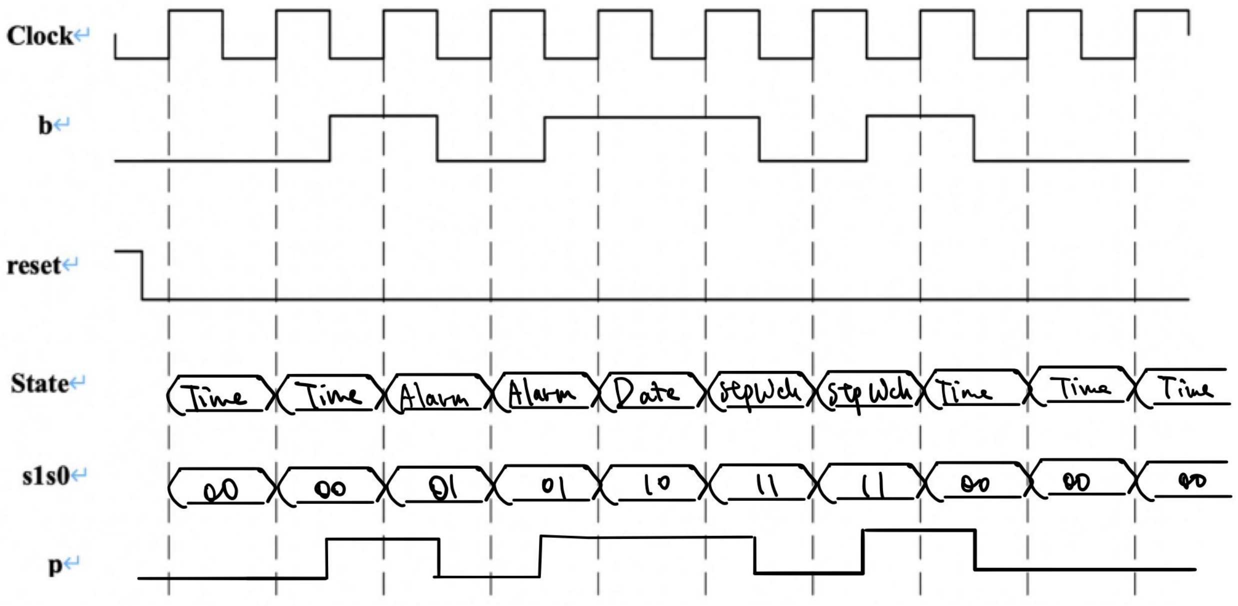
Assigned: July 14, 2020

Due: July 21, 2020, 2:00pm.

A pop quiz will be given on July 23.

1. Given a finite state machine described as the following state diagram, complete the timing diagrams of states and outputs s1s0 and p according to the given inputs b and reset. Ignore delays. (20 Points)





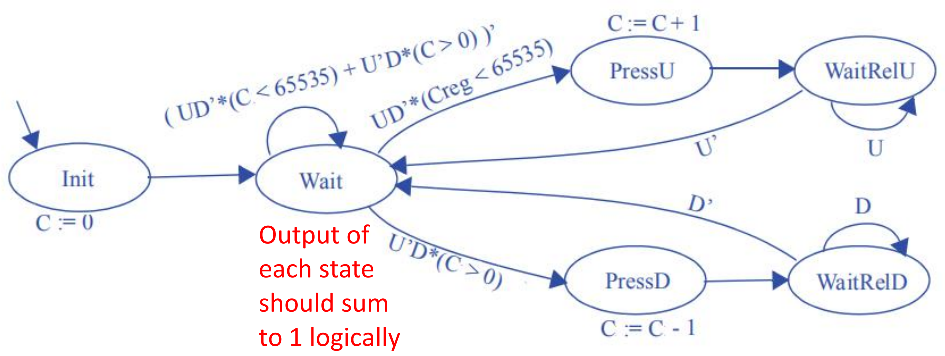
p is asynchronous

2. Problem 5.3 (15 points)

5.3 Capture the following system behavior as an HLSM. The system has two single-bit inputs U and D each coming from a button, and a 16-bit output C, which is initially 0. For each press of U, the system increments C. For each press of D, the system decrements C. If both buttons are pressed, the system does not change C. The system does not roll over; it goes no higher than than the largest C and no lower than C=0. A press is detected as a change from 0 to 1; the duration of that 1 does not matter.

Inputs and outputs should be declared

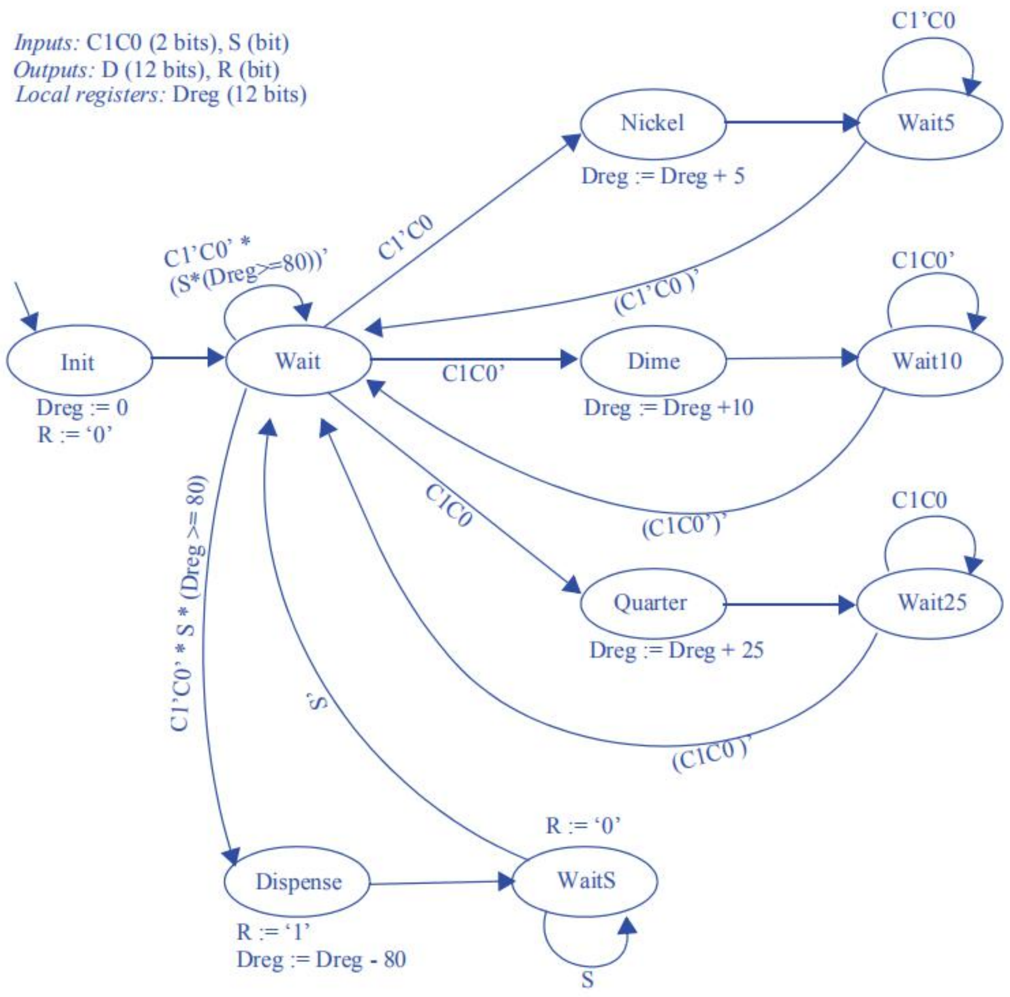
Inputs: U (bit), D (bit)
Outputs: C (16 bits)



Output of each state should sum to 1 logically

3. Problem 5.4 (20 points)

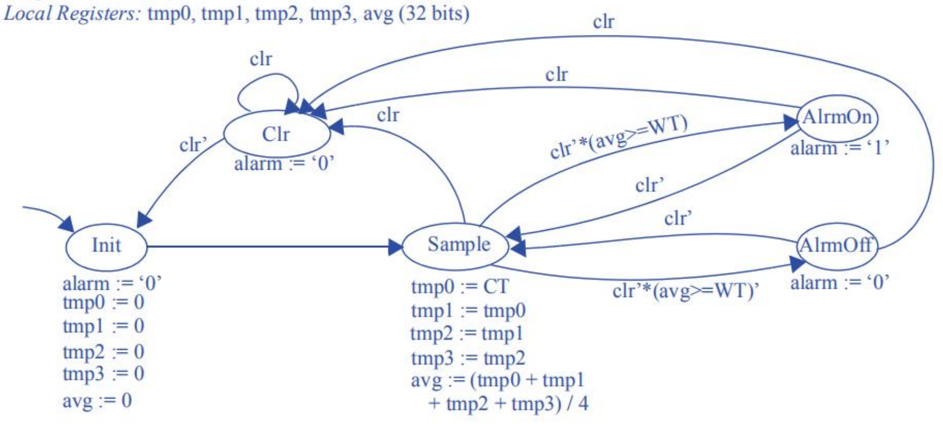
5.4 Capture the following system behavior as an HLSM. A soda machine dispenser system has a 2-bit control input `C1 C0` indicating the value of a deposited coin. `C1C0` = `00` means no coin, `01` means nickel (5 cents), `10` means dime (10 cents), and `11` means quarter (25 cents); when a coin is deposited, the input changes to indicate the value of the coin (for possibly more than one clock cycle) and then changes back to `00`. A soda costs 80 cents. The system displays the deposited amount on a 12-bit output D. The system has a single-bit input `S` coming from a button. If the deposited amount is less than the cost of a soda, `S` is ignored. Otherwise, if the button is pressed, the system releases a single soda by setting a single-bit output `R` to `1` for exactly one clock cycle, and the system deducts the soda cost from the deposited amount.



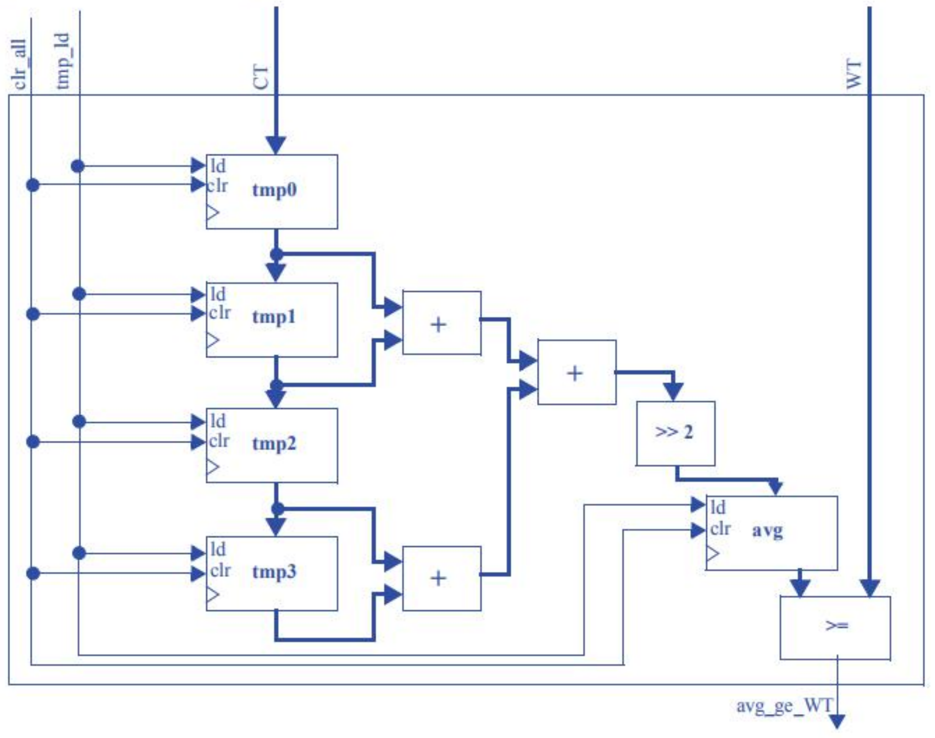
4. Problem 5.14. (25 points)

Step 1 - Capture a high-level state machine

Inputs: CT, WT (32 bits); clr (bit)
Outputs: alarm (bit)
Local Registers: tmp0, tmp1, tmp2, tmp3, avg (32 bits)

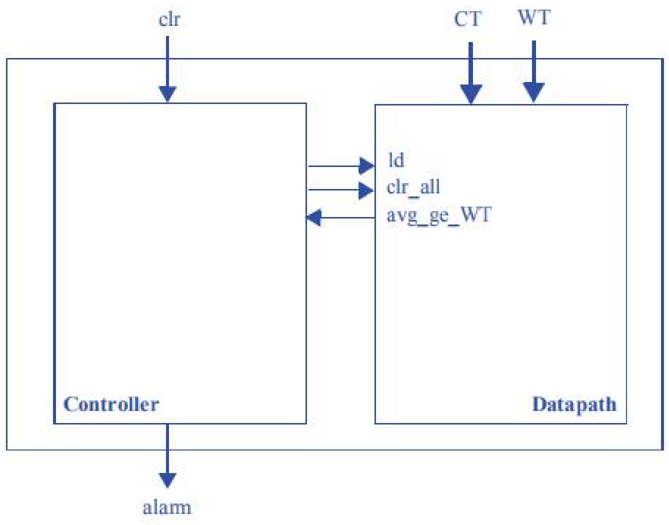


Step 2A - Create a datapath



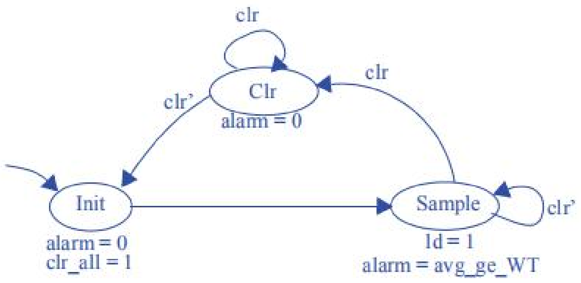
Note: A solution more consistent with the chapter's methdology would use a separate clear and ld signal for each register. In this particular example, a single clr and a single load line happens to work.

Step 2B- Connect the datapath to a controller

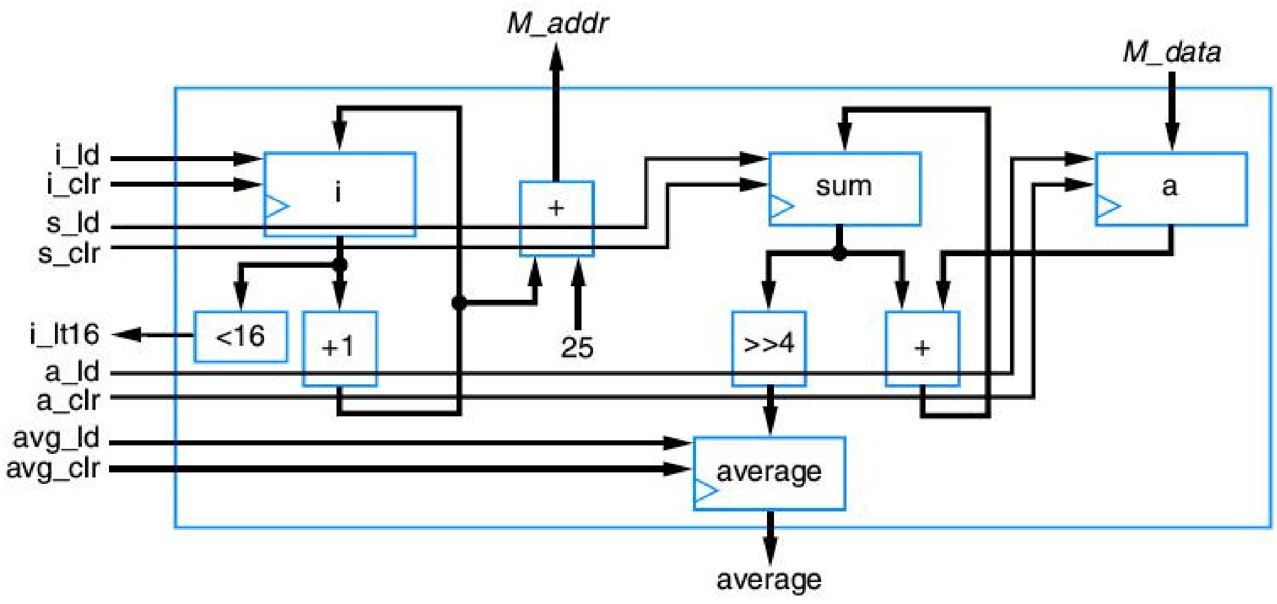


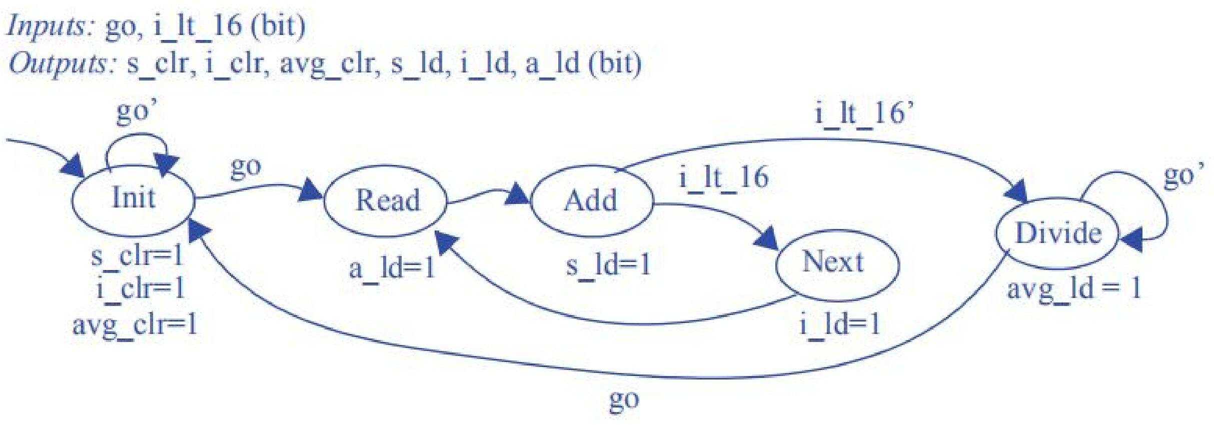
Step 2C - Derive the controller's FSM

Inputs: clr, avg_lt_WT
Outputs: alarm, clr_all, ld



5. (20 points) Create an FSM that interfaces with the datapath in following figure. The FSM should use the datapath to compute the average value of the 16 32-bit elements of an array A. Array A is stored in a memory, with the first element at address 26, then second at address 27, and so on. Assume that putting a new value onto the address lines *M_addr* causes the memory to almost immediately output the read data on the *M_data* lines. Ignore overflow issues.


M_addr
M_data
i_ld
i_clr
s_ld
s_clr
i
+
sum
a
i_lt16
<16
+1
25
>>4
+
a_ld
a_clr
avg_ld
avg_clr
average
average


Inputs: go, i_lt_16 (bit)
Outputs: s_clr, i_clr, avg_clr, s_ld, i_ld, a_ld (bit)
go'
Init
go
Read
Add
i_lt_16'
i_lt_16
Next
Divide
go'
s_clr=1
i_clr=1
avg_clr=1
a_ld=1
s_ld=1
i_ld=1
avg_ld = 1
go