JOINT INSTITUTE
交大密西根学院

Ve270 Introduction to Logic Design

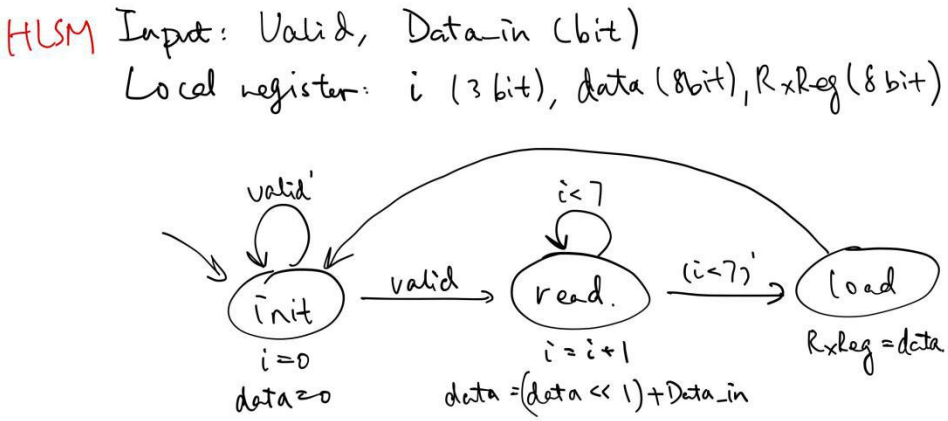
Homework 9

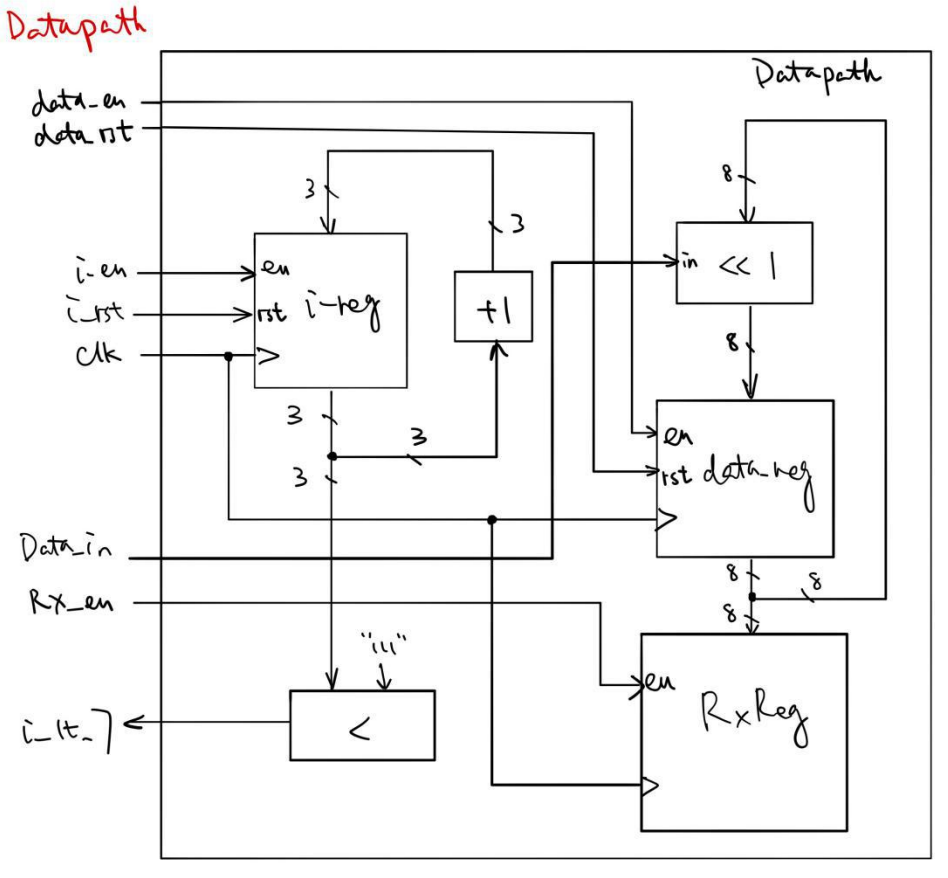
Assigned: July 21, 2020

Due: July 28, 2020, 2:00pm.

A pop quiz will be given on July 30.

1. Design a circuit called ***Receiver*** that receives two single bit signals, **Valid** and **Data_in**, from another device called ***Transmitter***. The **Valid** signal sent from the ***Transmitter*** will be a 1-clock cycle pulse. After the ***Receiver*** receives the **Valid** pulse, it will start receiving 8 bits through port **Data_in**, bit by bit. After the 8 bits of data is received, they should be copied into an 8-bit register called **RxReg**. (25 points)

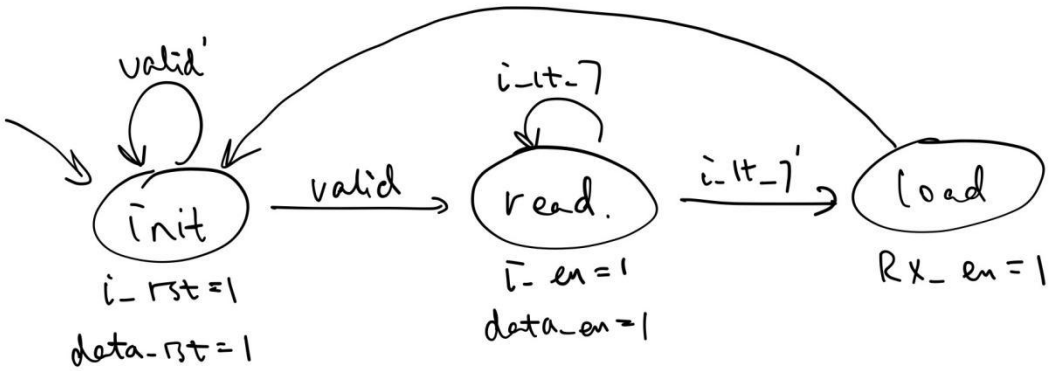




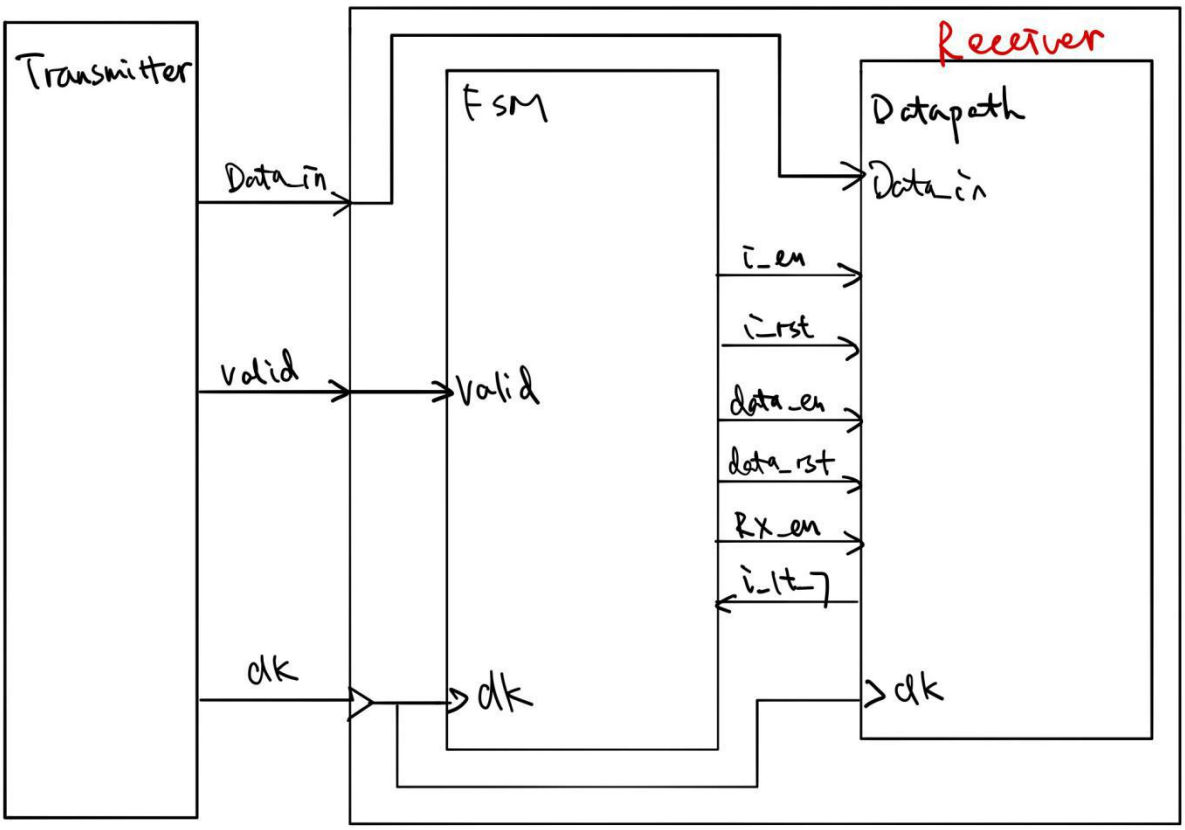
FSM.

Input: Valid, i_lt_7 (bit)

Output: i_en, i_rst, data_en, data_rst, Rx_en (bit).



Implement the circuit of FSM. (omitted here).

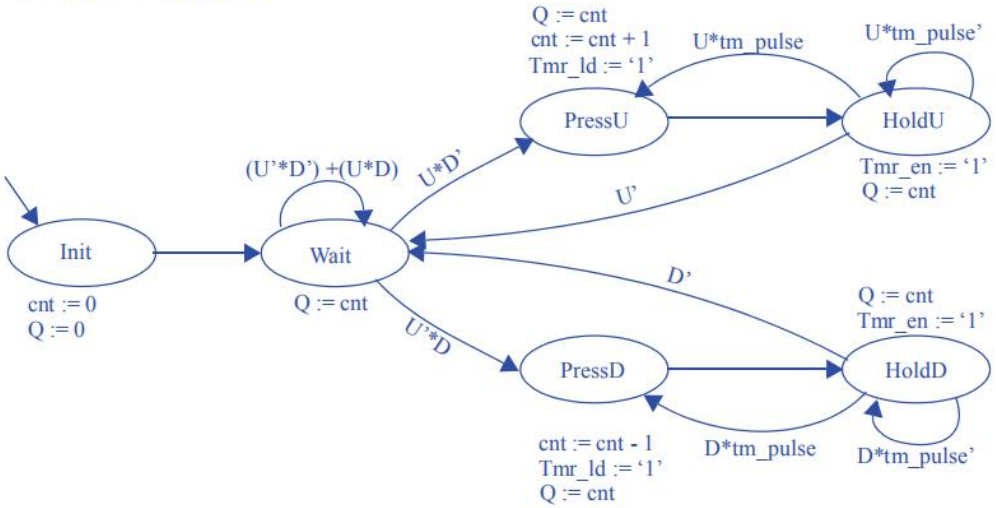


2. Problem 5.19 (25 points)

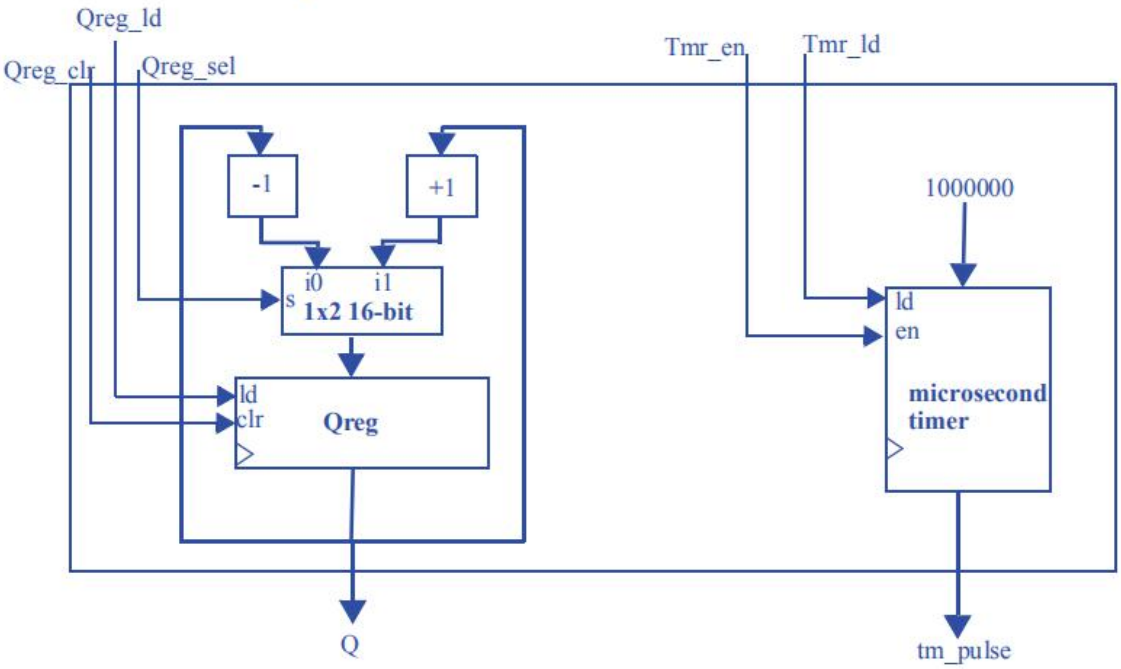
5.19 Using a timer, design a system with single-bit inputs `U` and `D` corresponding to two buttons, and a 16-bit output Q which is initially 0. Pressing the button for `U` causes Q to increment, while `D` causes a decrement; pressing both buttons causes Q to stay the same. If a single button is held down, Q should then continue to increment or decrement at a rate of once per second as long as the button is held. Assume the buttons are already debounced. Assume Q simply rolls over if its upper or lower value is reached.

Step 1 - Capture a high-level state machine

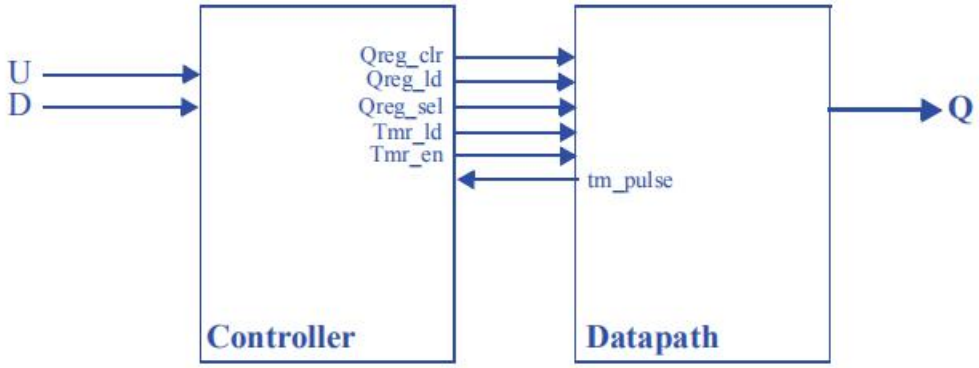
Inputs: U, D, tm_pulse (bit)
Outputs: Q (16 bits), Tmr_ld, Tmr_en (bit)
Local Registers: cnt (16 bits)



Step 2A - Create a datapath

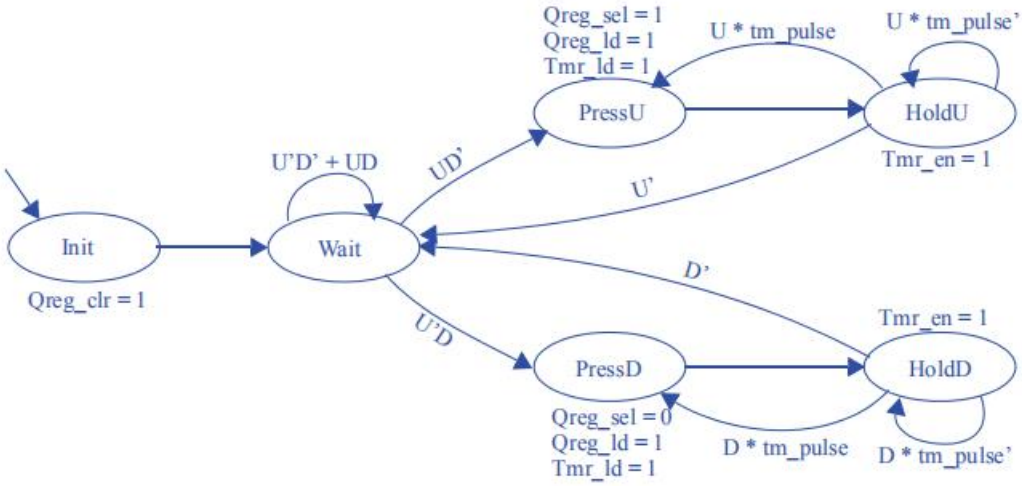


Step 2B - Connect the datapath to a controller



Step 2C - Derive the controller's FSM

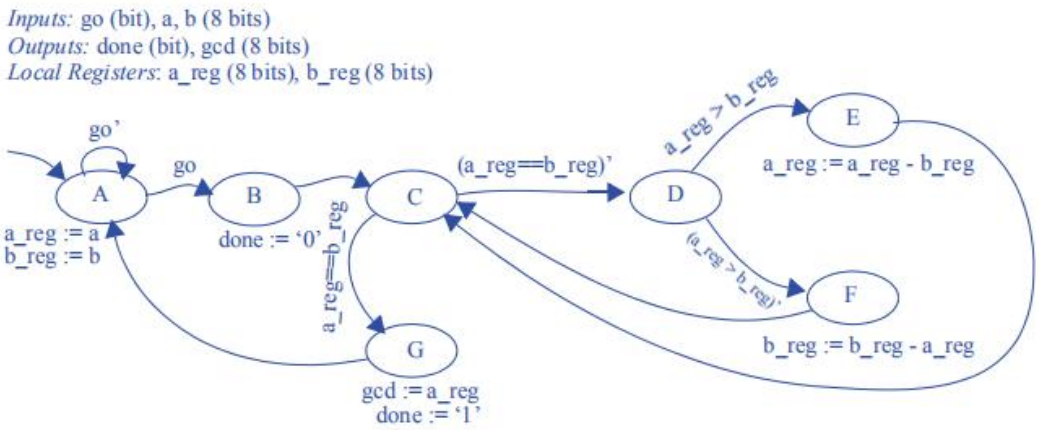
Inputs: U, D, tm_pulse
Outputs: Qreg_clr, Qreg_ld, Qregsel, Tmr_ld, Tmr_en



3. Problem 5.27 (15 points)

5.27 Convert the following C-like code, which calculates the greatest common divisor (GCD) of the two 8-bit numbers *a* and *b*, into a high-level state machine.

```
Inputs: byte a, byte b, bit go
Outputs: byte gcd, bit done
GCD:
while(1) {
  while(!go);
  done = 0;
  while ( a != b ) {
    if( a > b ) {
      a = a - b;
    }
    else {
      b = b - a;
    }
  }
  gcd = a;
  done = 1;
}
```



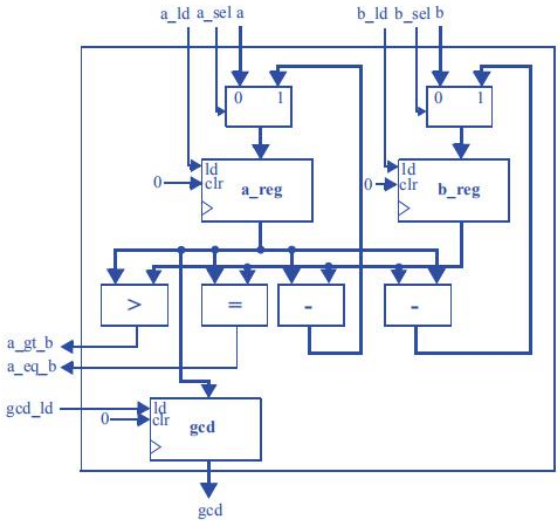
4. Problem 5.28 (15 points)

5.28 Use the RTL design process to convert the high-level state machine you created in Exercise 5.27 to a controller and a datapath. Design the datapath to structure, but design the controller to the point of an FSM only.

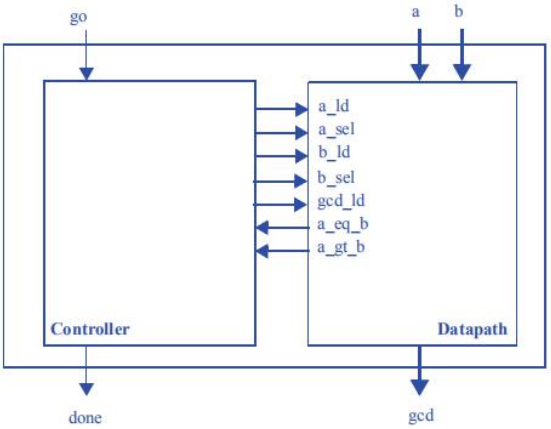
Step 1 - Capture a high-level state machine

The high-level state machine was developed in Exercise 5.27.

Step 2 - Create a datapath

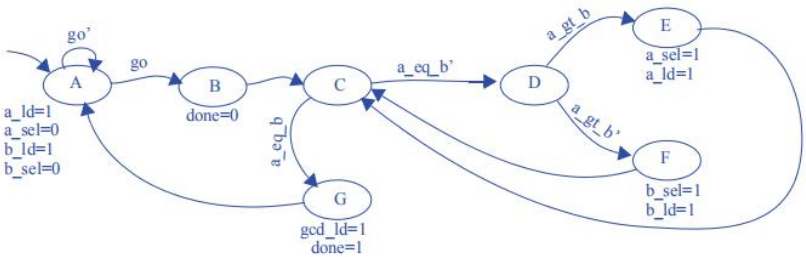


Step 3 - Connect the datapath to a controller



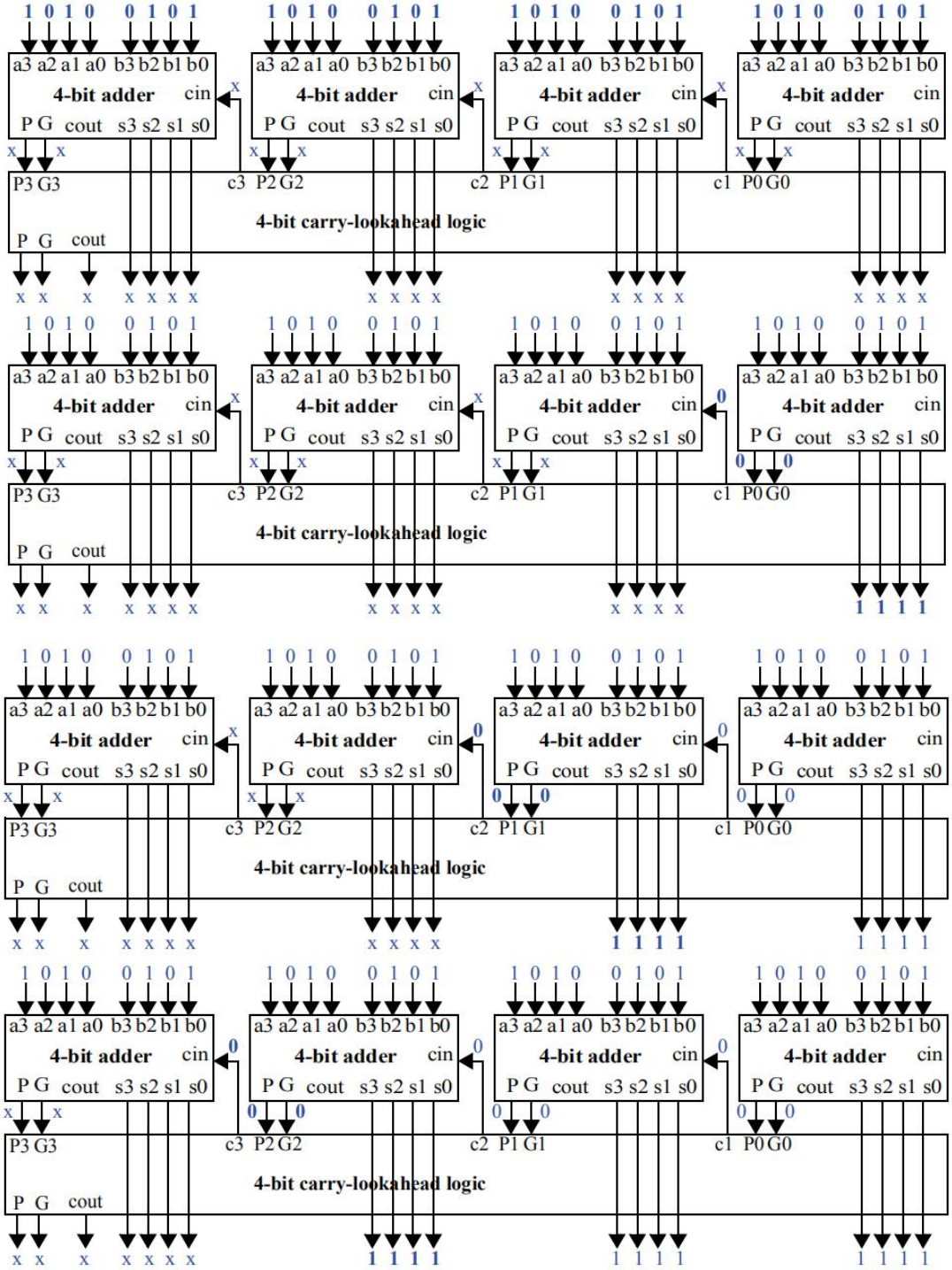
Step 4 - Derive the controller's FSM

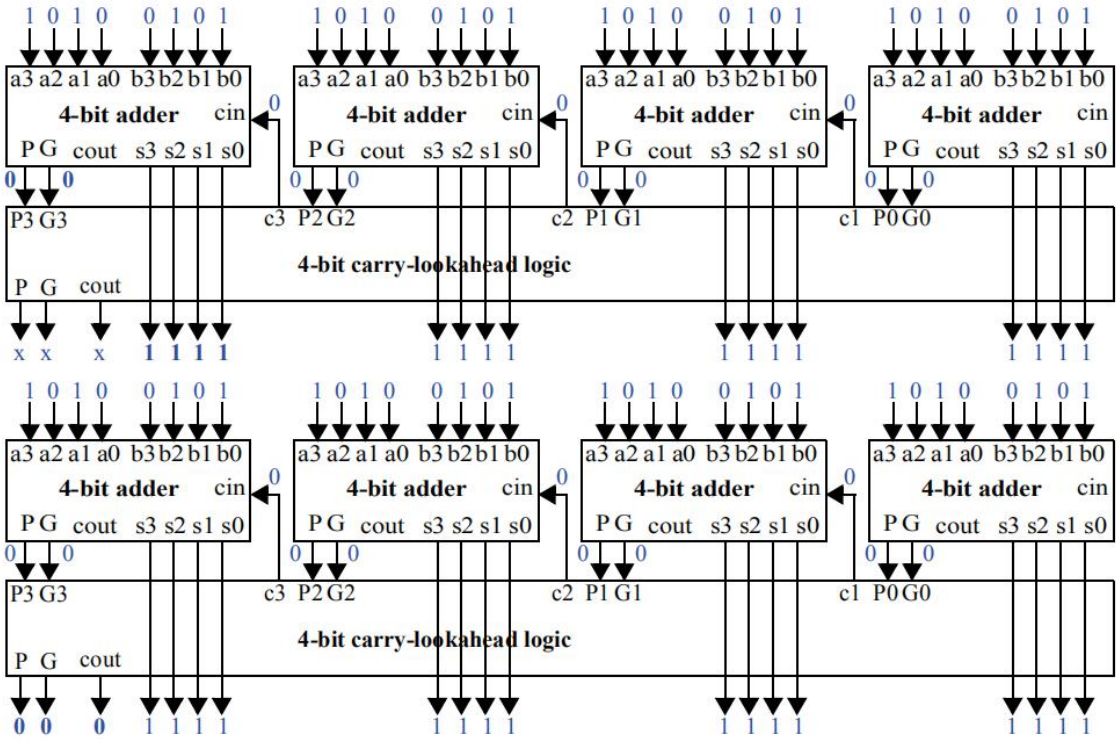
Inputs: go, done, a_gt_b, a_eq_b (bit)
Outputs: done, a_ld, a_sel, b_ld, b_sel, gcd_ld (bit)



5. Problem 6.27 (10 points)

6.27) Trace the execution of the 16-bit carry-lookahead adder built from 4-bit adders as shown in Figure 6.60 when a = 43690 and b = 21845. Do not trace internal behavior of the individual 4-bit carry-lookahead adders..




4-bit adder
4-bit carry-lookahead logic
4-bit adder
4-bit carry-lookahead logic

6. Following circuit is a carry-ripple style incrementor. Redesign the circuit as a carry-lookahead structure using the same Half Adders (HA). (10 points)