Digital Design Hwk 7

Saturday, November 30, 2019 1:28 PM

5.7 Create a datapath for the HLSM in Figure 5.98.

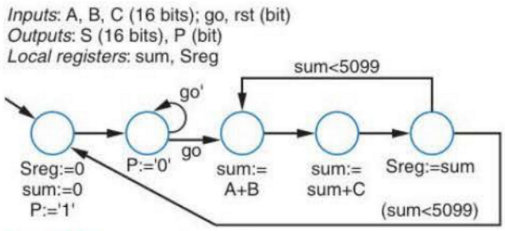
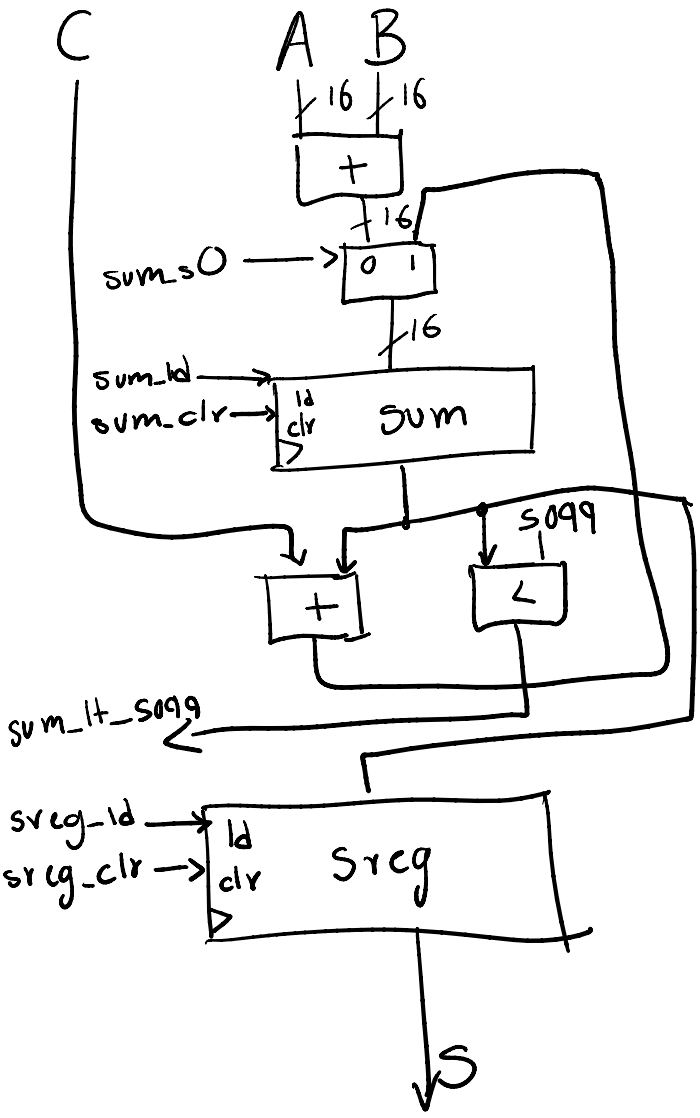


Figure 5.98 Sample high-level state machine.



HLSM in Figure 5.63.

5.9 For the HLSM in Figure 5.14, complete the RTL design process:
(a) Create a datapath.
(b) Connect the datapath to a controller.
(c) Derive the controller's FSM.

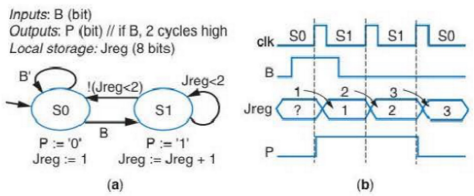
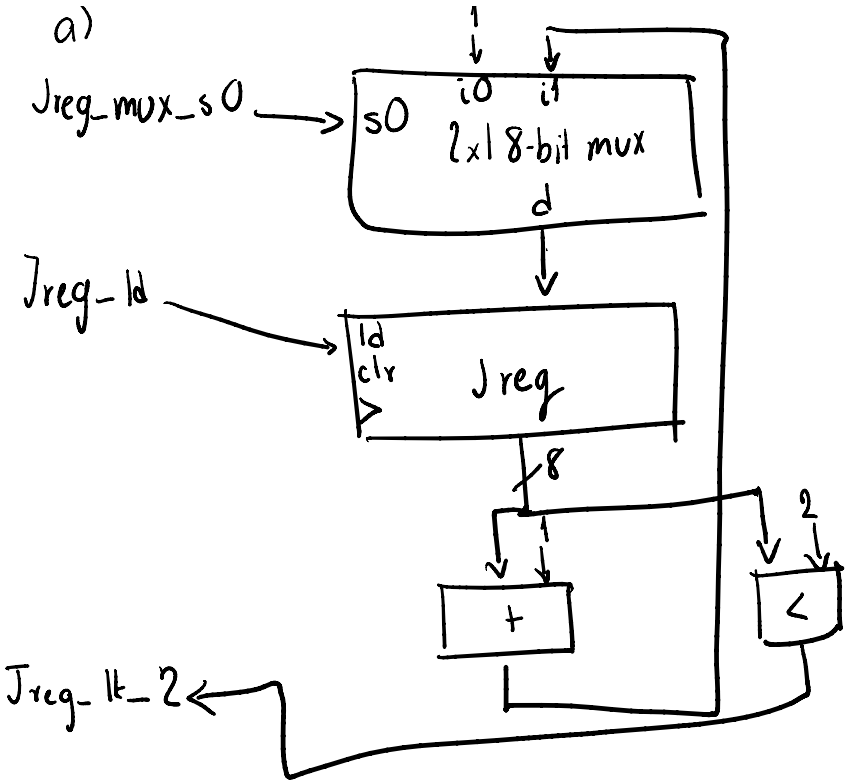
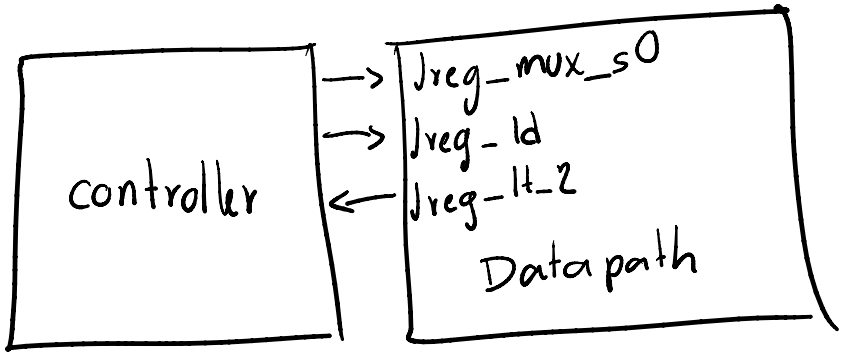


Figure 5.14 HLSM clocked storage update behavior: (a) HLSM with storage *Jreg*, (b) *S0*'s setting *Jreg* to *1* doesn't occur until the next clock edge, which is also when *S1* is entered. *S1*'s setting of *Jreg*:=*Jreg*+1 will set *Jreg* to 2, but not until the next edge, meaning *Jreg*<2 is false for the first *S1* state.

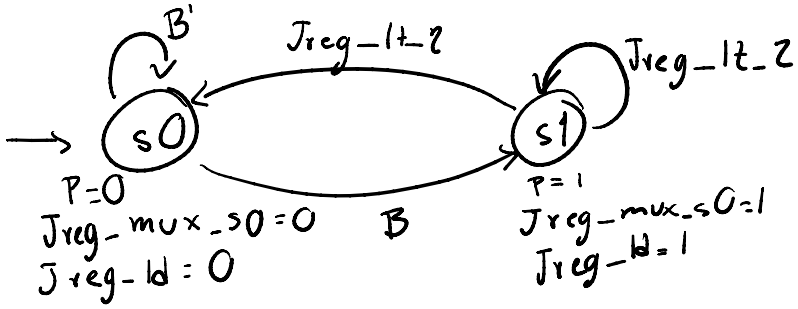
a)



b)



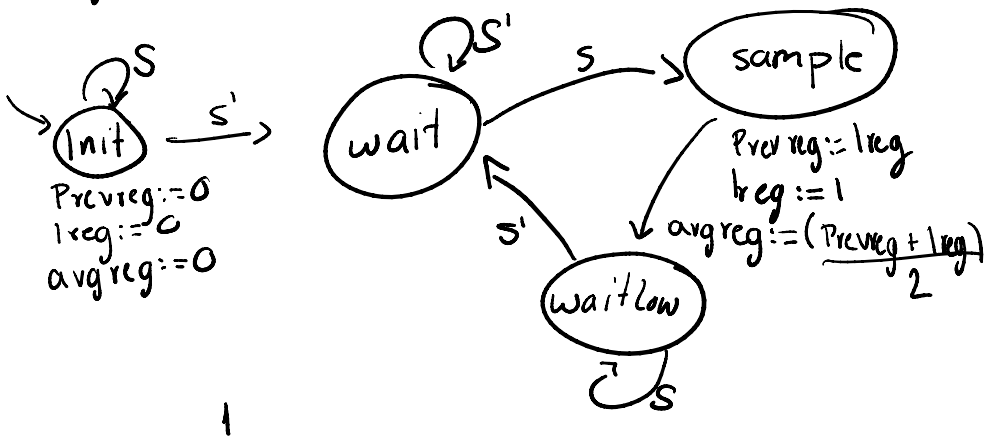
c) Inputs: B, Jreg_lt_2
Outputs: P, Jreg_mux_s0, Jreg_ld

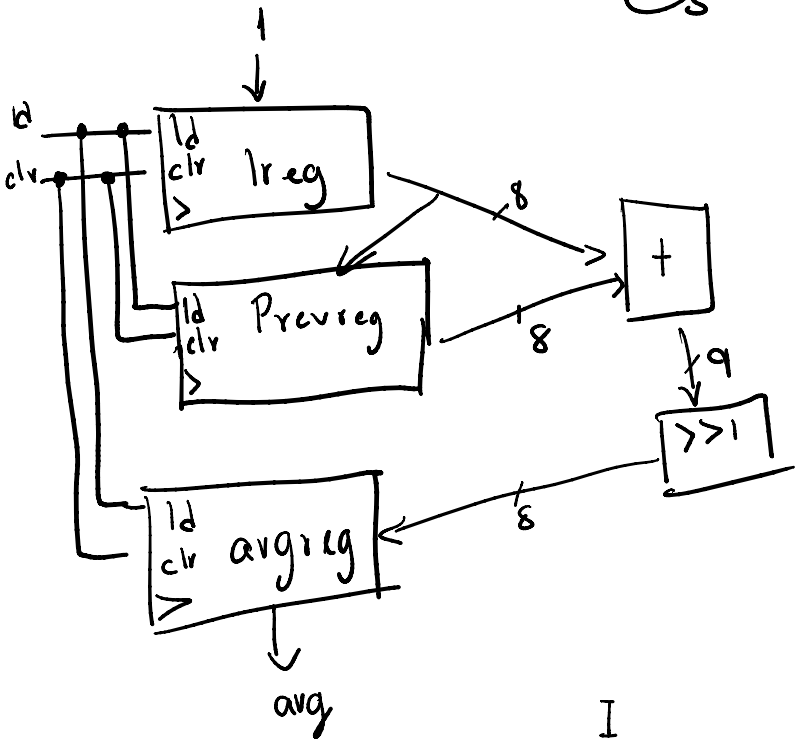


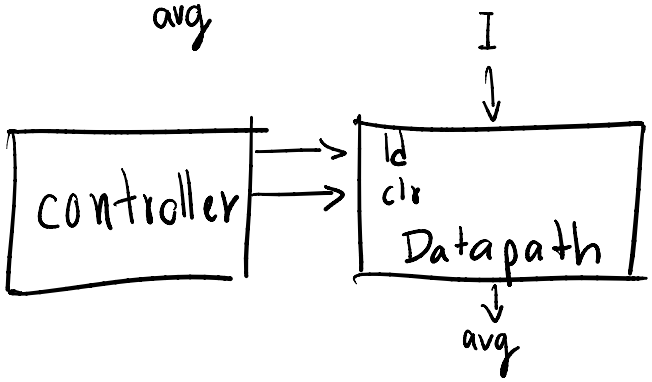
5.13 Use the RTL design process to design a system that outputs the average of the most recent two data input samples. The system has an 8-bit unsigned data input *I*, and an 8-bit unsigned output *avg*. The data input is sampled when a single-bit input S changes from 0 to 1. Choose internal bitwidths that prevent overflow.

Inputs: I(8bits), S(bit)

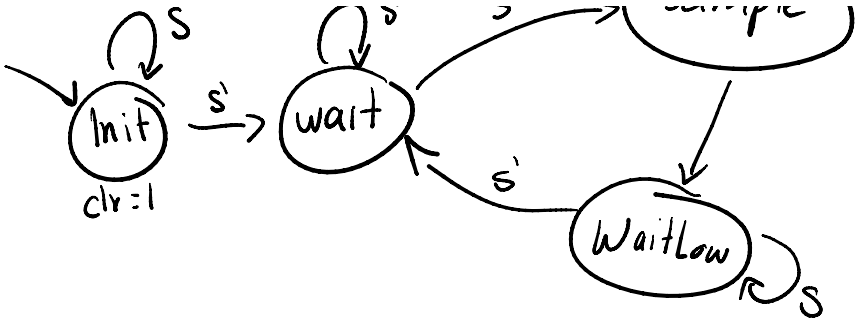
Inputs: I(8bits), S(bit)
Outputs: avg(8bits)
local registers: Prevreg(8bits), Ireg(8bits), avg reg (8bits)





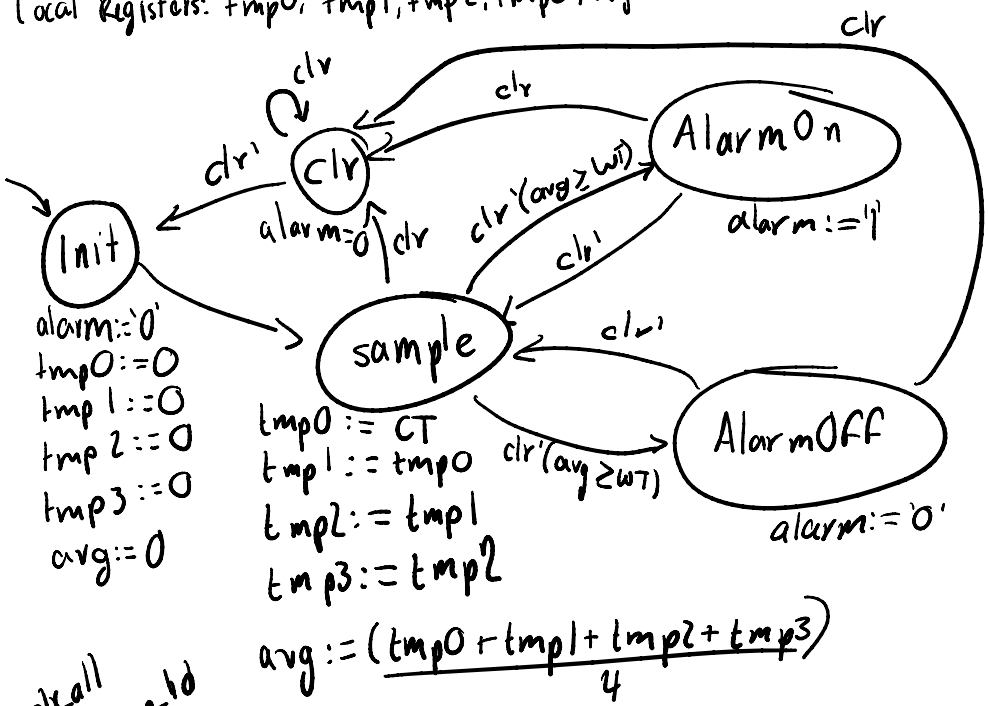


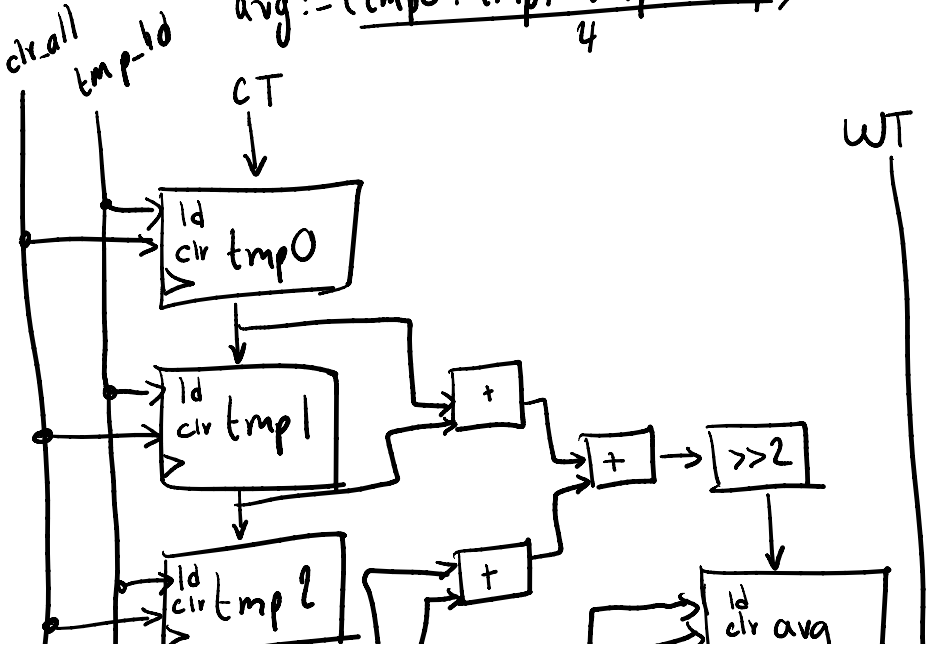
Inputs: S
Outputs: ld, clr

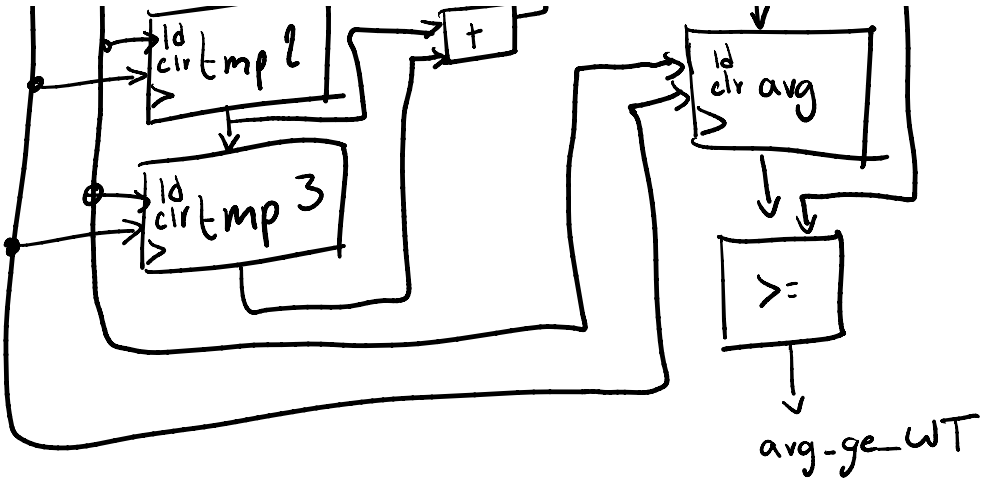


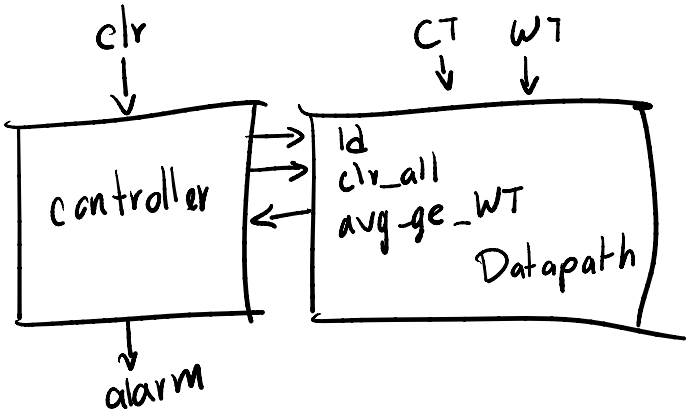
5.14 Use the RTL design process to create an alarm system that sets a single-bit output `alarm` to `1` when the average temperature of four consecutive samples meets or exceeds a user-defined threshold value. A 32-bit unsigned input *CT* indicates the current temperature, and a 32-bit unsigned input *WT* indicates the warning threshhold. Samples should be taken every few clock cycles. A single-bit input `clr` when `1` disables the alarm and the sampling process. Start by capturing the desired system behavior as an HLSM, and then convert to a controller/datapath.

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

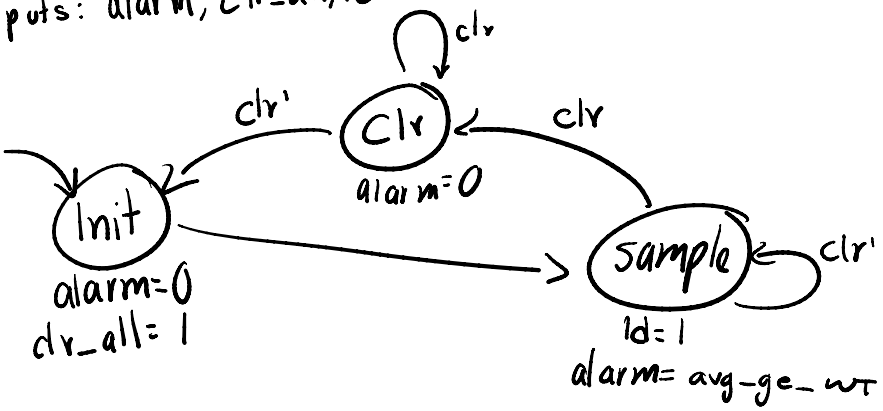






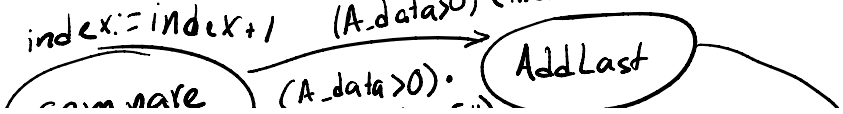


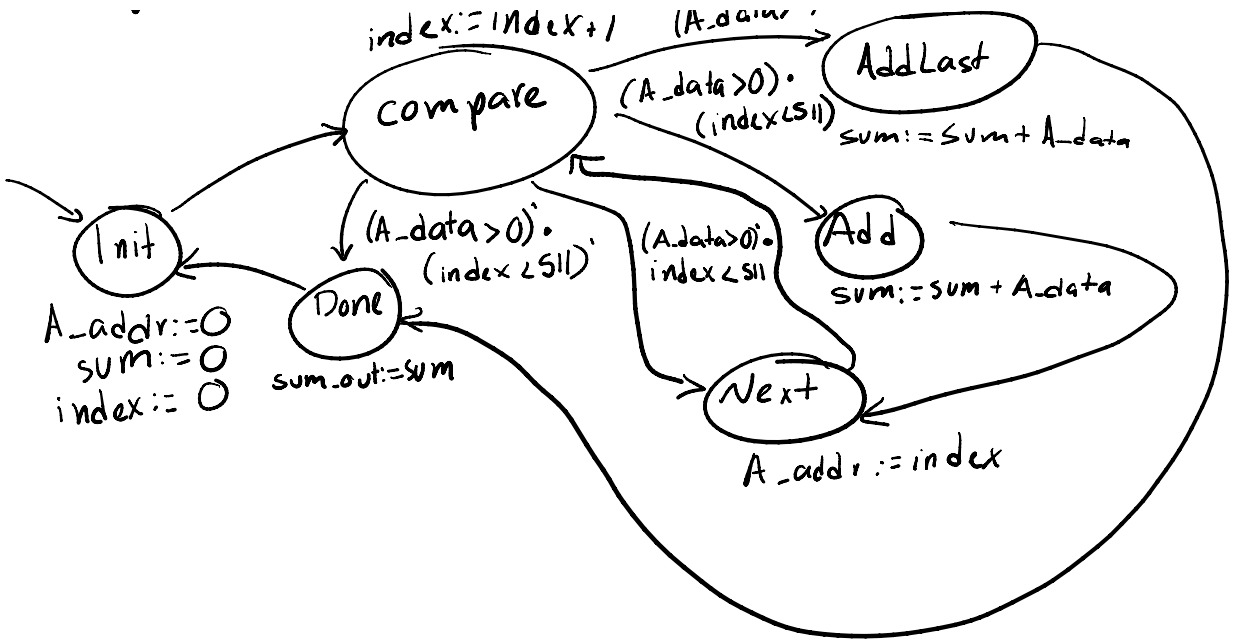
Inputs: clr, avg_lt_WT
Outputs: alarm, clr_all, ld

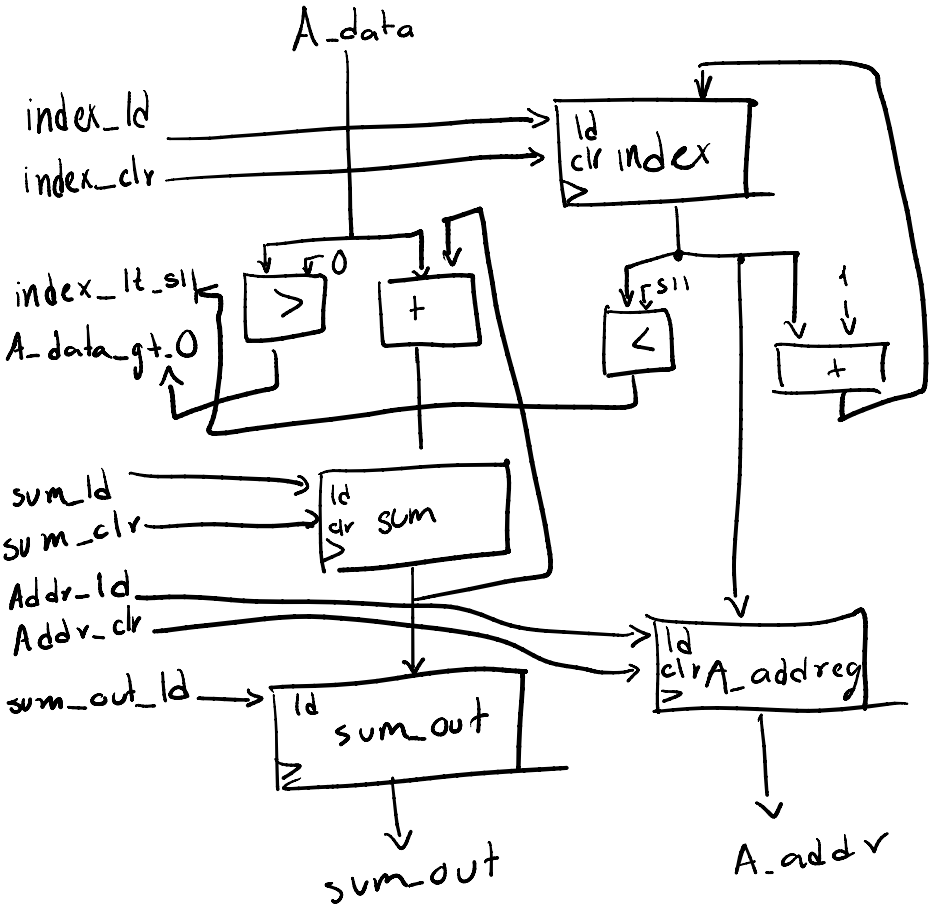


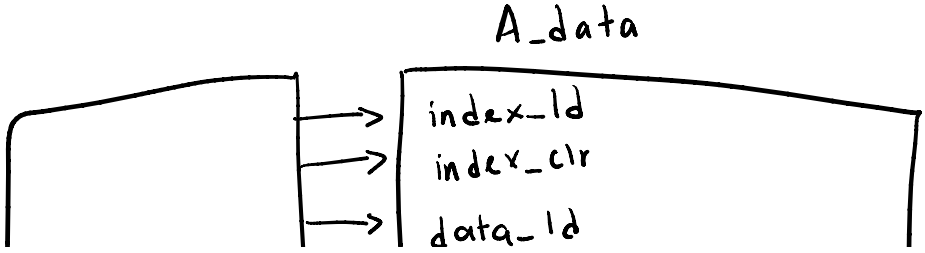
5.17 Design a system that repeatedly computes and outputs the sum of all *positive* numbers within a 512-word register file *A* consisting of 32-bit signed numbers.

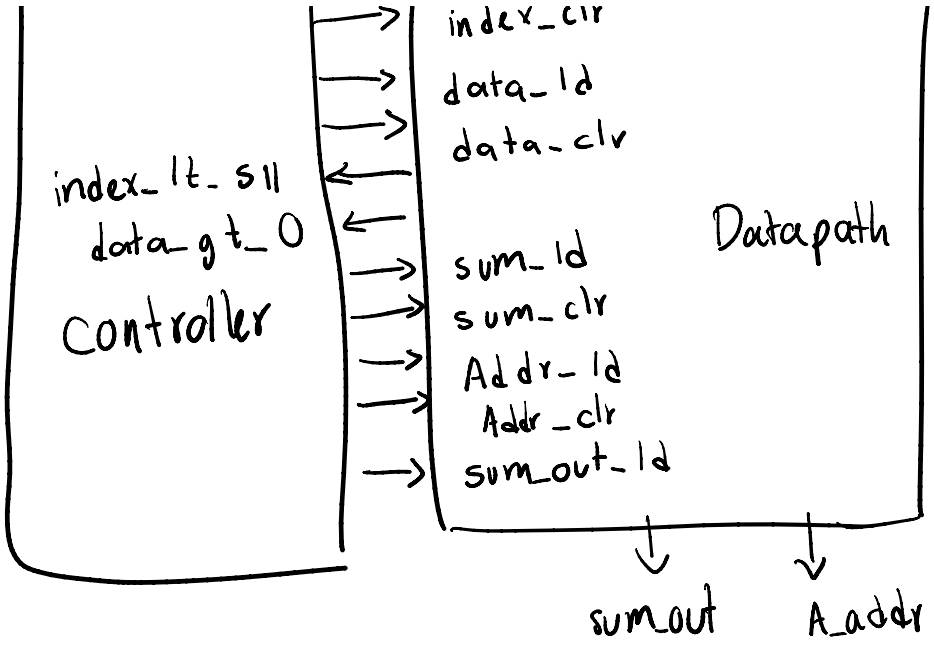
Inputs: A_data(32 bits)
Outputs: A_addr(9 bits), sum_out(32 bits)
Local Registers: sum(32 bits), index(9 bits)

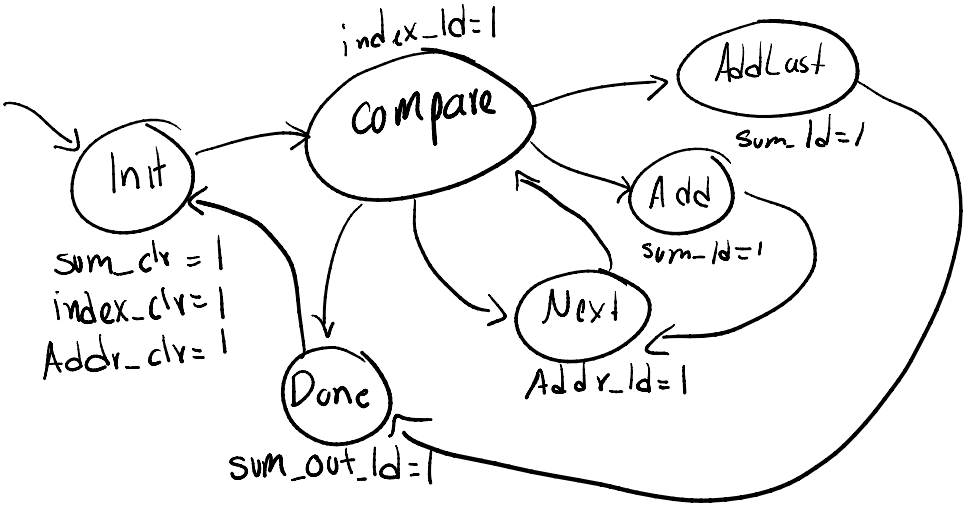



index:=index+1
compare
(A_data>0)·
(index<511)
AddLast
sum:=sum+A_data
Init
A_addr:=0
sum:=0
index:=0
(A_data>0)'·
(index<511)'
Done
sum_out:=sum
(A_data>0)·
index<511
Add
sum:=sum+A_data
Next
A_addr:=index


A_data
index_ld
index_clr
ld
clr index
index_lt_511
A_data_gt_0
>
0
+
<
511
1
+
sum_ld
sum_clr
ld
clr sum
Addr_ld
Addr_clr
ld
clr A_addreg
sum_out_ld
ld
sum_out
sum_out
A_addr

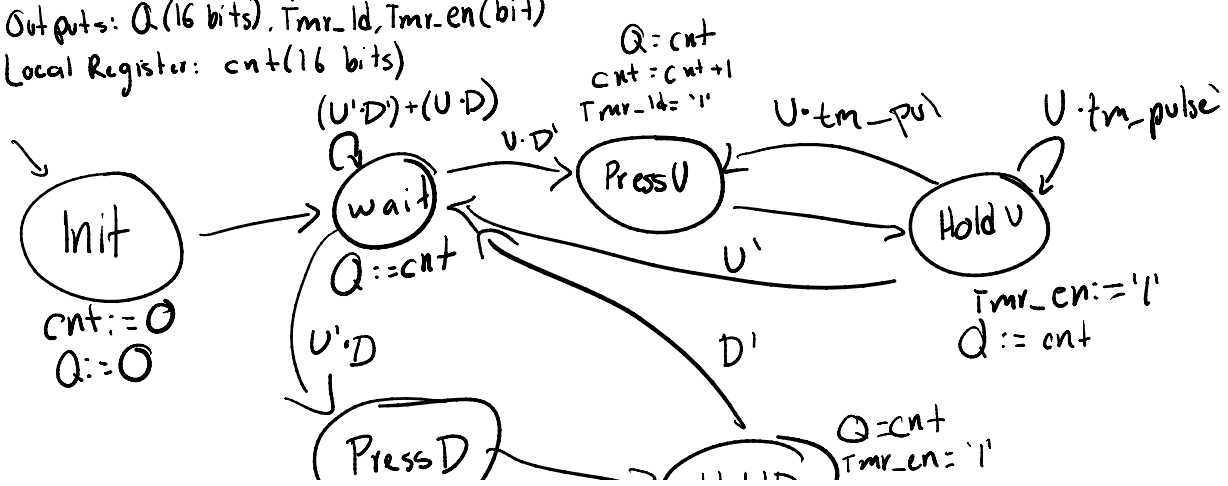

A_data
index_ld
index_clr
data_ld

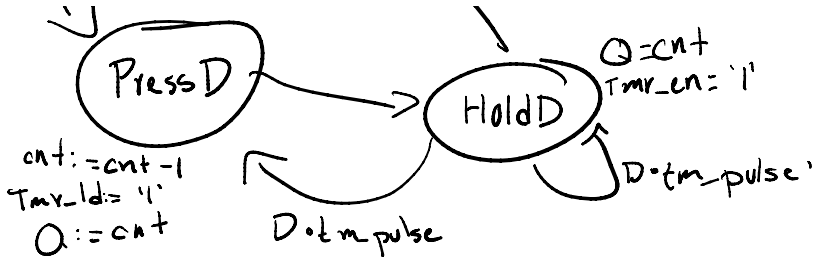


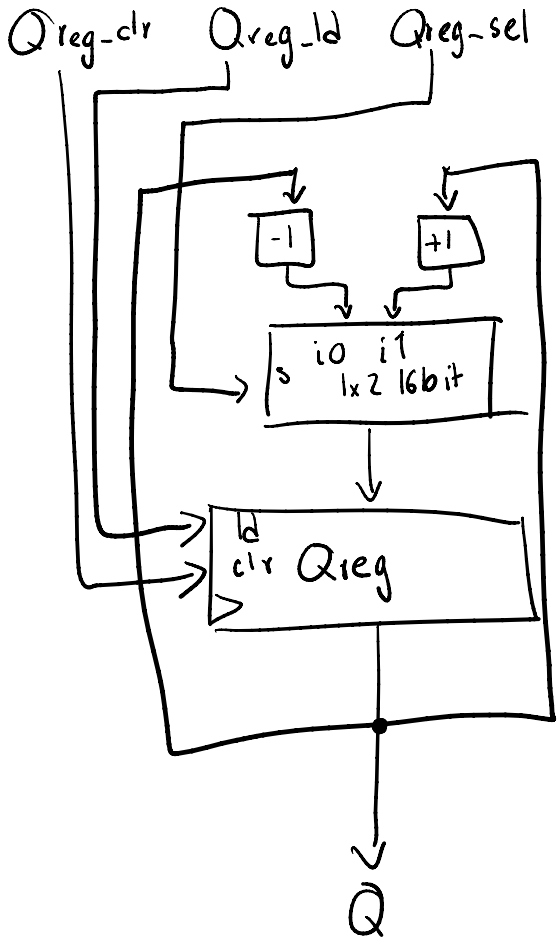


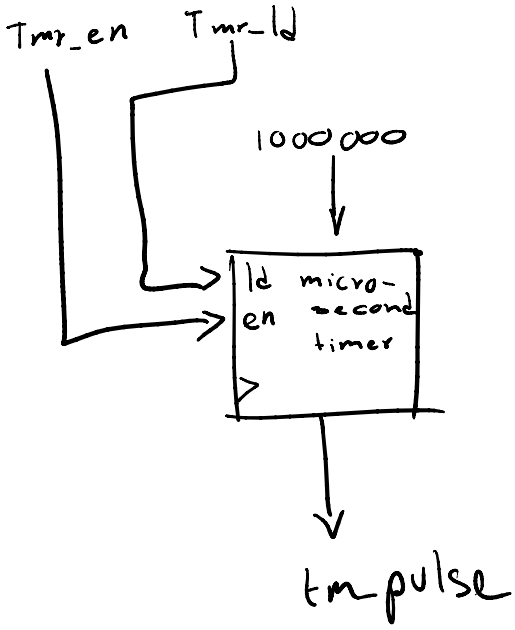
6.19 Using a timer, design a system with single-bit inputs U and D corresponding to two buttons, and a 16-bit output Q which is initially 0. Pressing the button for U causes Q to increment, while D causes a decrement; pressing both buttons causes Q to stay the same. If a single button is held down, Q 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.

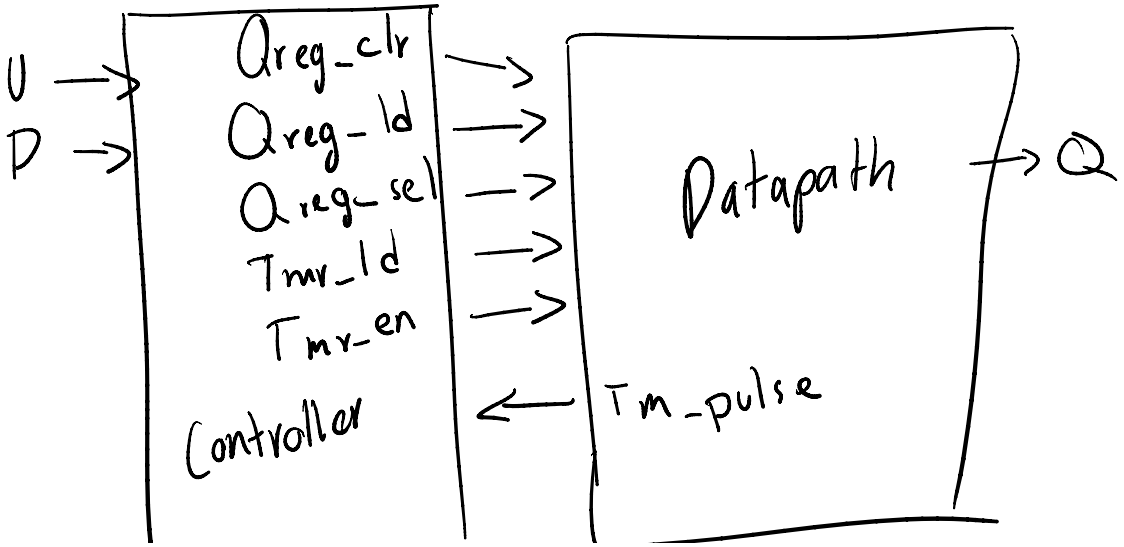
Inputs: U, D, tm_pulse(bit)
Outputs: Q(16 bits), Tmr_ld, Tmr_en(bit)
Local Register: cnt(16 bits)

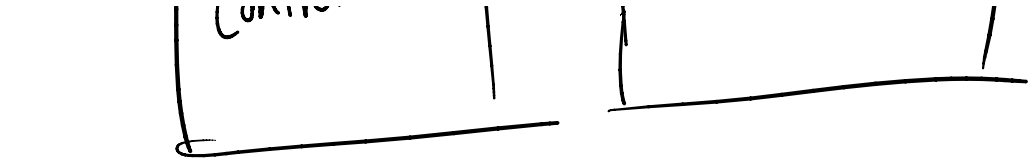



PressD
HoldD
Q:=cnt
Tmr_en:='1'
D•tm_pulse'
cnt:=cnt-1
Tmr_ld:='1'
Q:=cnt
D•tm_pulse


Qreg_clr
Qreg_ld
Qreg_sel
-1
+1
s i0 i1
1x2 16bit
ld
clr Qreg
Q

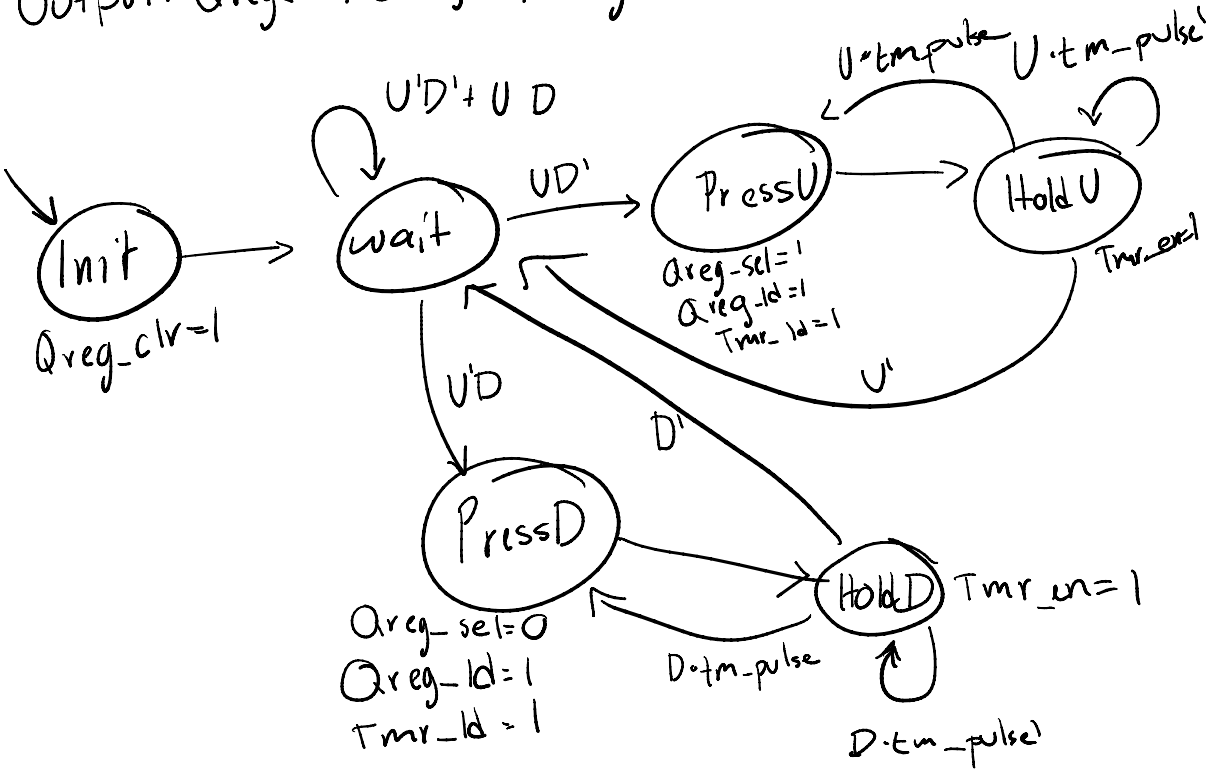

Tmr_en
Tmr_ld
1000000
ld micro-
en second
timer
tm_pulse


U
D
Qreg_clr
Qreg_ld
Qreg_sel
Tmr_ld
Tmr_en
Controller
Tm_pulse
Datapath
Q



Input: U, D, tm_pulse

Output: Qreg_clr, Qreg_ld, Qreg_sel, Tmr_ld, Tmr_en



5.23 Assuming an inverter has a delay of 1 ns, all other gates have a delay of 2 ns, and wires have a delay of 1 ns, determine the critical path for the 3x8 decoder of Figure 2.62.

The critical path of the decoder is on the inverted inputs to outputs

wire + inverter + wire + AND + wire

1 + 1 + 1 + 2 + 1 = 6

5.37 Calculate the approximate number of SRAM bit storage cells that will fit on an IC with a capacity of 10 million transistors.

10 million transistors / 6 transistors / SRAM

1,666,666 SRAM bit storage cells

5.42 Summarize the main differences between EEPROM and flash memories.

EEPROM permits erasing one word at a time
flash memory permits erasing any at a time

5.45 Create an FSM describing the queue controller of Figure 5.79. Pay careful attention to correctly setting the `full` and `empty` outputs.

Inputs: wr, rd, reset, eq
Outputs: rear_clr, rear_inc, front_clr, front_inc, rf_wr, rf_wd, full, empty

ad and write ports.

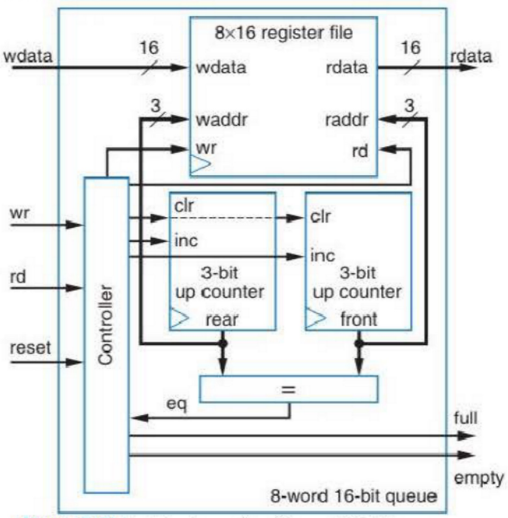
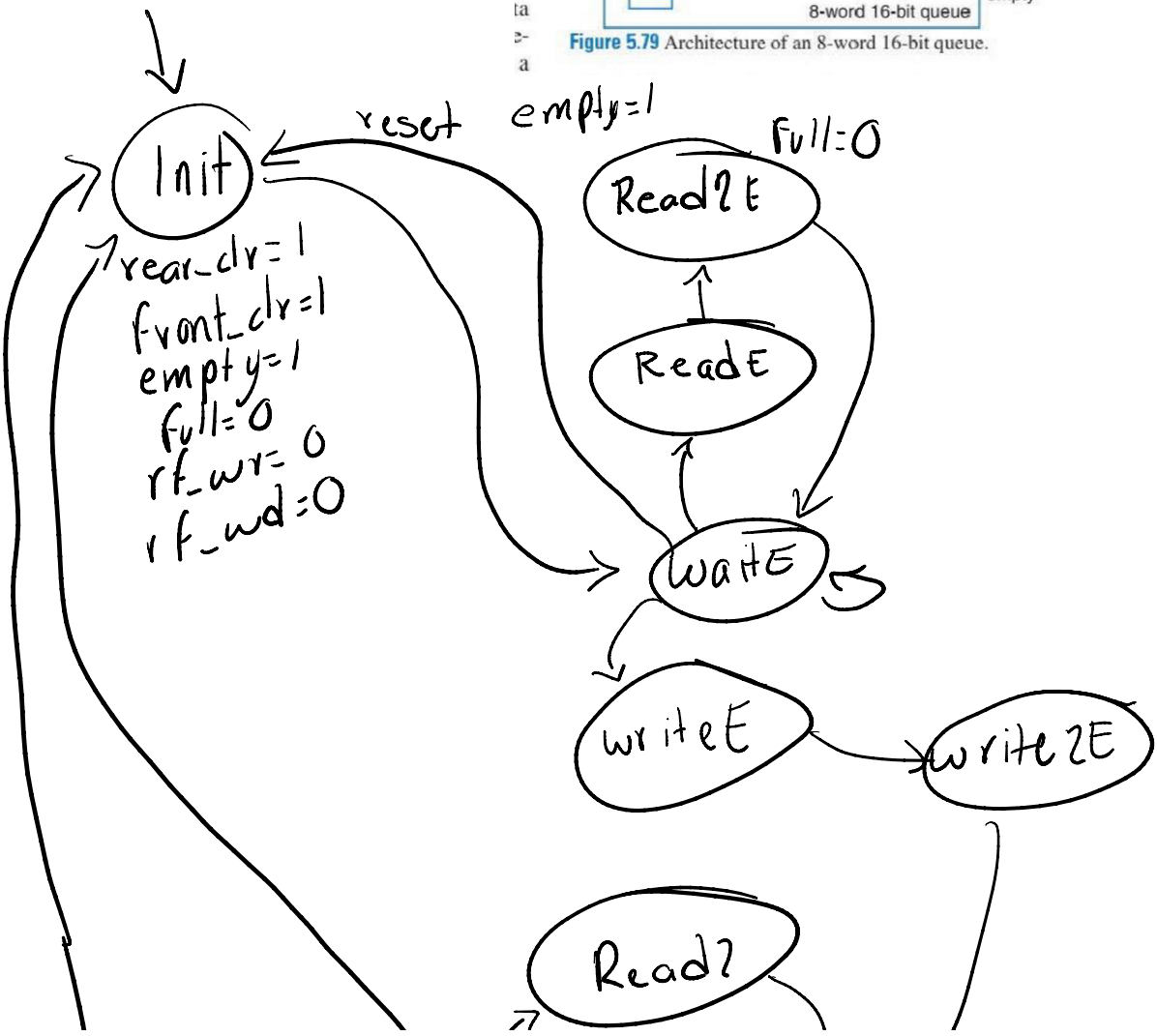
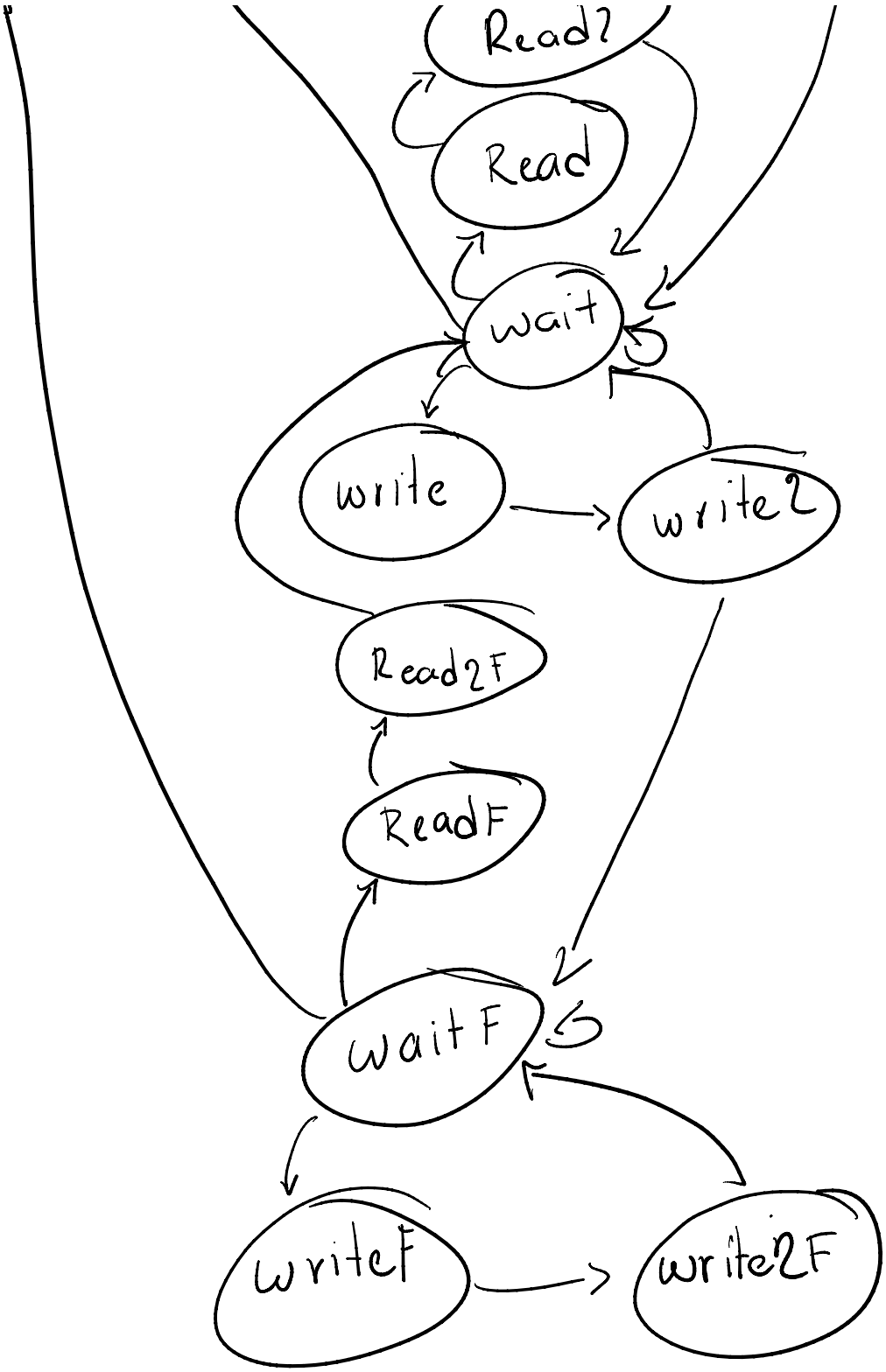
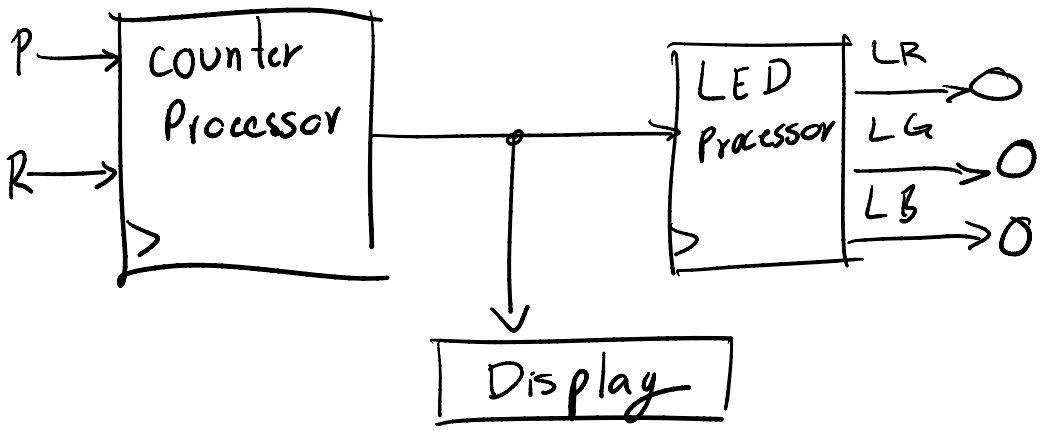


Figure 5.79 Architecture of an 8-word 16-bit queue.

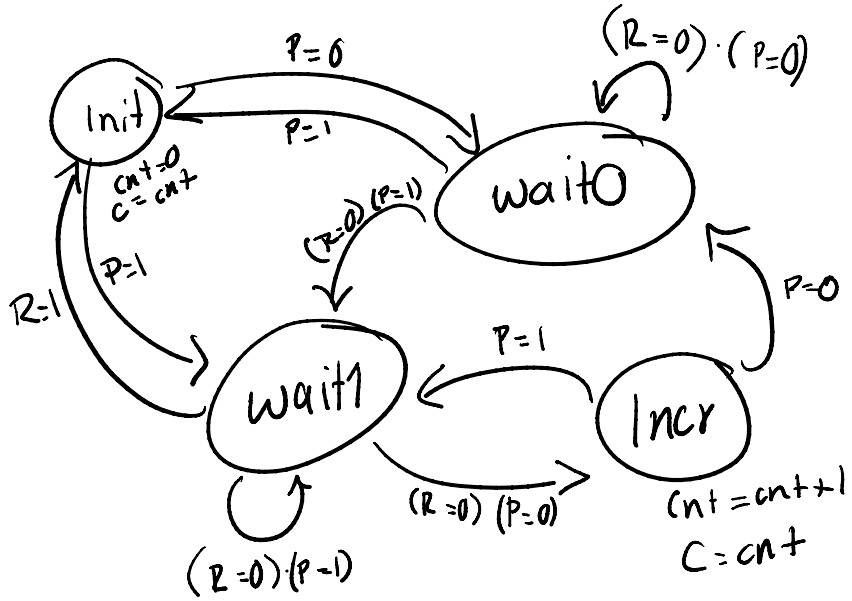




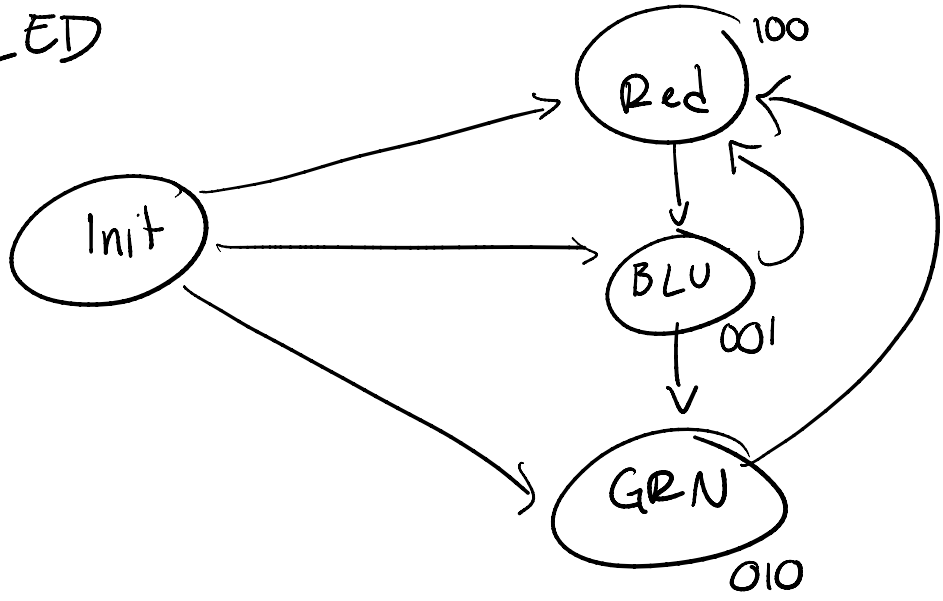
5.47 A system *S* counts people that enter a store, incrementing the count value when a single-bit input `P` changes from `1` to `0`. The value is reset when `R` is `1`. The value is output on a 16-bit output *C*, which connects to a display. Furthermore, the system has a lighting system to indicate the approximate count value to the store manager, turning on a red LED (`LR=1`) for 0 to 99, else a blue LED (`LB=1`) for 100 to 199, else a green LED (`LG=1`) for 200 and above. Draw a block diagram of the system and its peripheral components, using *two* processors for the system *S*. Show the HLSM for each processor.


P
R
counter
Processor
LED
Processor
LR
LG
LB
0
0
Display

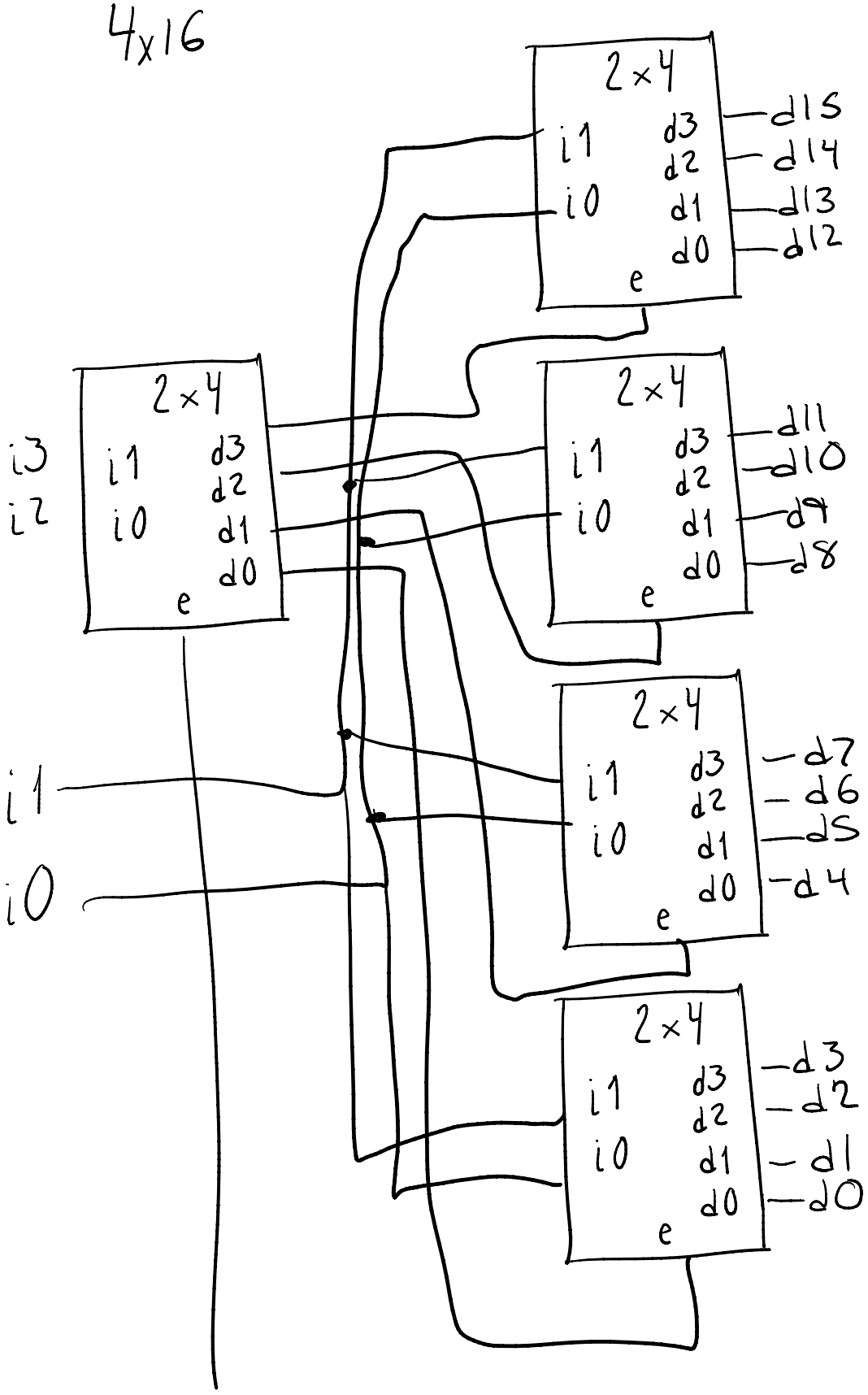
counter


Init
wait0
wait1
Incr
P=0
P=1
(R=0)·(P=0)
cnt=0
C=cnt
(R=0)(P=1)
R=1
P=1
P=1
P=0
(R=0)(P=0)
(R=0)(P=1)
cnt=cnt+1
C=cnt

LED

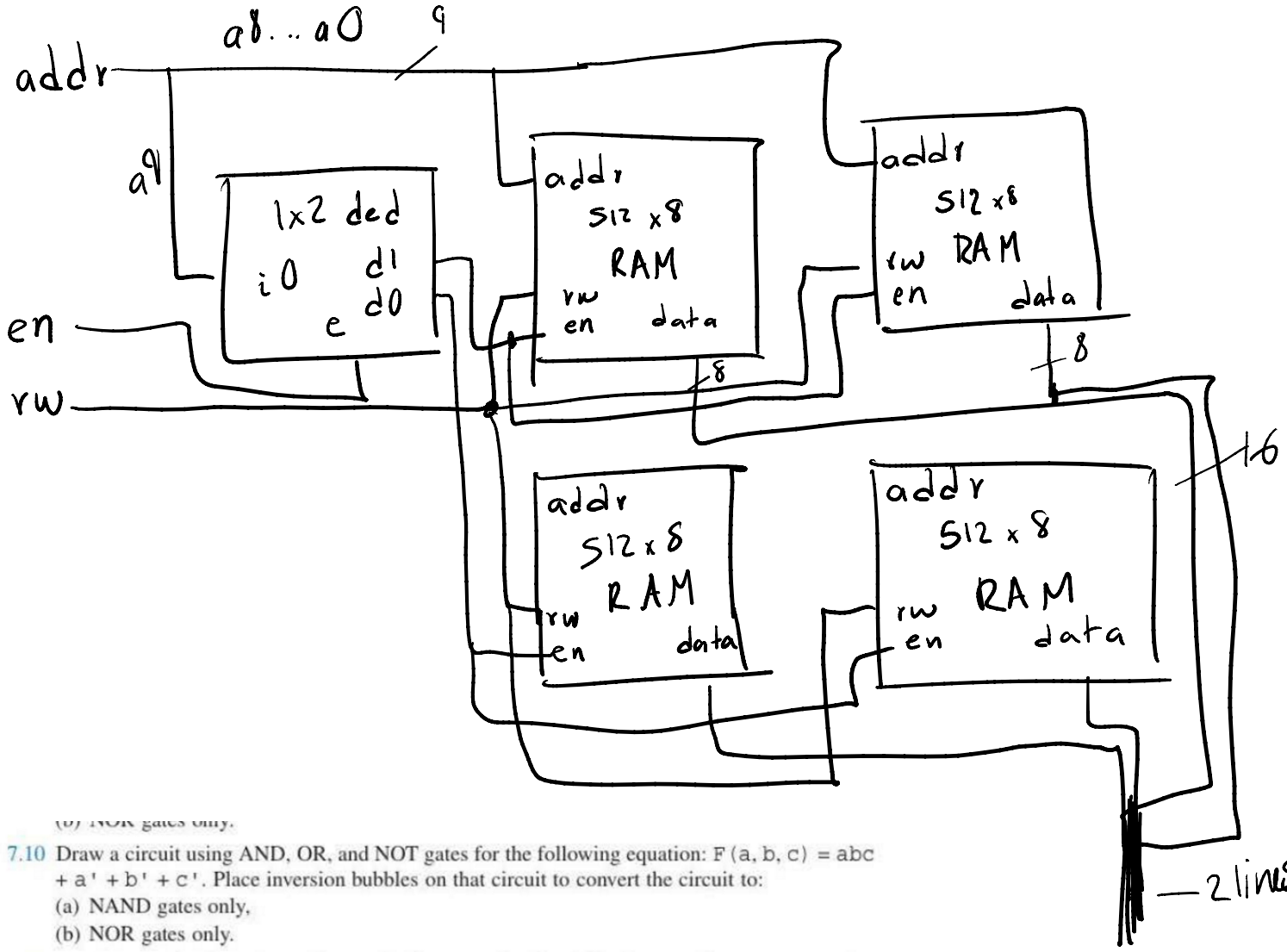

Init
Red
100
BLU
001
GRN
010

5.52 Compose a 4x16 decoder with enable from 2x4 decoders with enable.



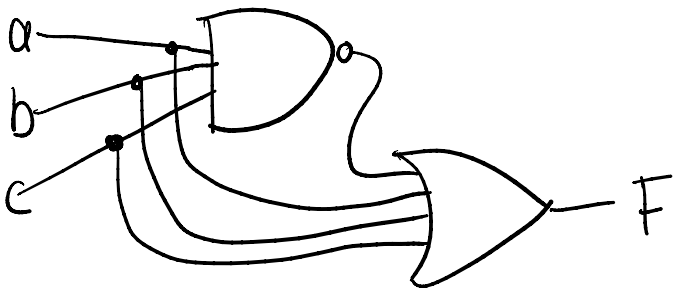
e

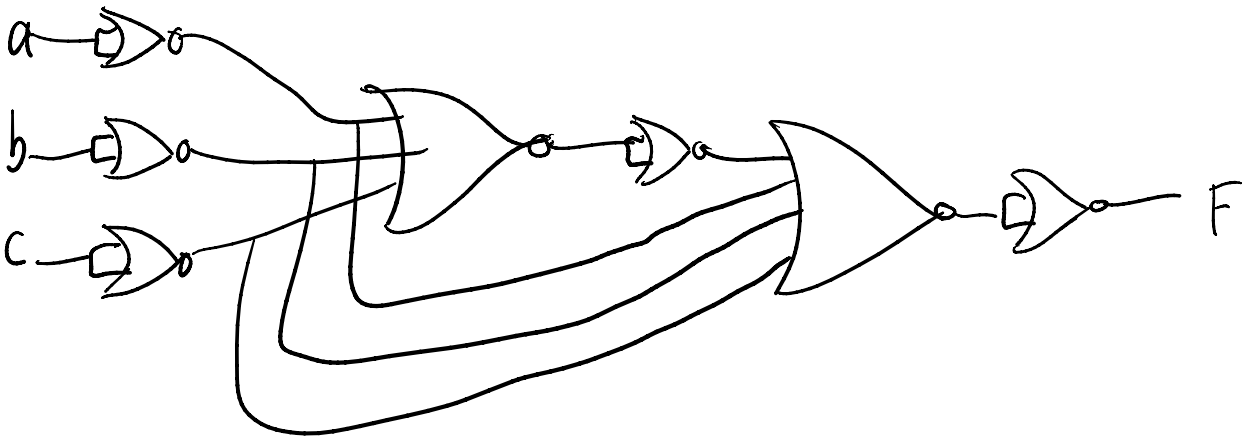
5.57 Compose a 1024x16 RAM using only 512x8 RAMs.



(b) NOR gates only.

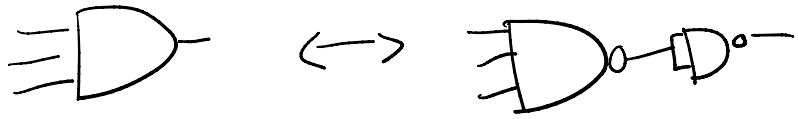
7.10 Draw a circuit using AND, OR, and NOT gates for the following equation: `F(a, b, c) = abc + a' + b' + c'`. Place inversion bubbles on that circuit to convert the circuit to:
(a) NAND gates only,
(b) NOR gates only.



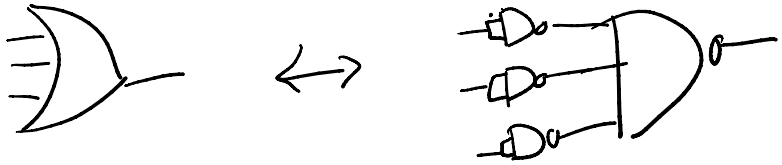


7.14 Show how to convert the following gates into circuits having only 3-input NAND gates:
(a) A 3-input AND gate.
(b) A 3-input OR gate.
(c) A NOT gate.

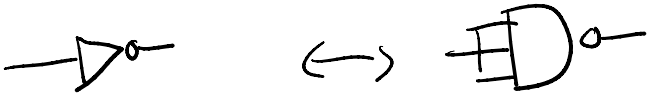
a)



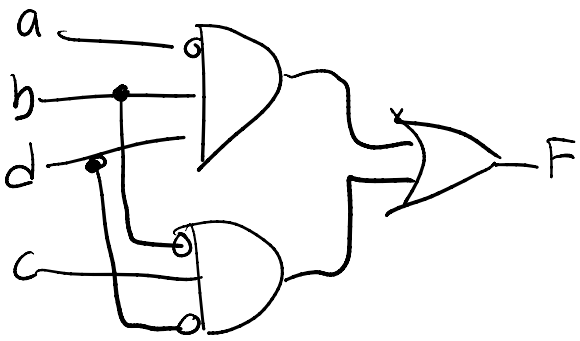
b)



c)



20 Show how to implement on two 3-input 2-output lookup tables the following function: `F(a, b, c, d) = a'bd + b'cd'`. Assume the two lookup tables are connected in the manner shown in Figure 7.47. You may not need to use every lookup table output.



r wires).

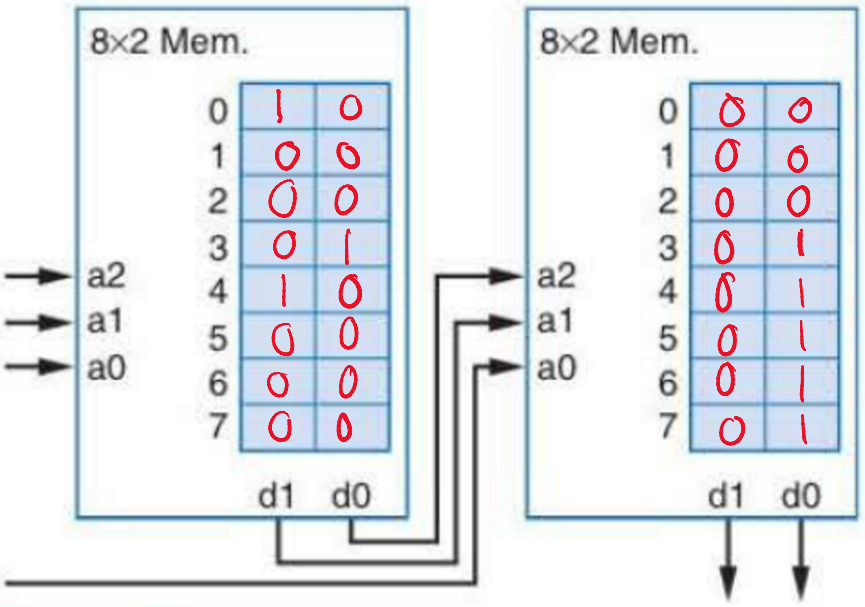
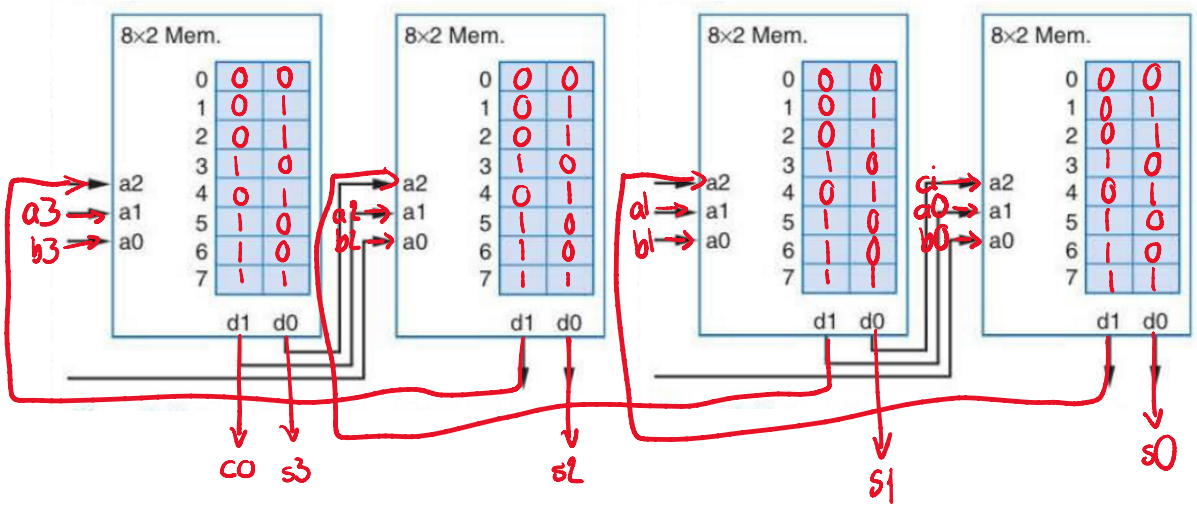


Figure 7.47 Two 3-input 2-output lookup tables implemented using 8x2 memory.

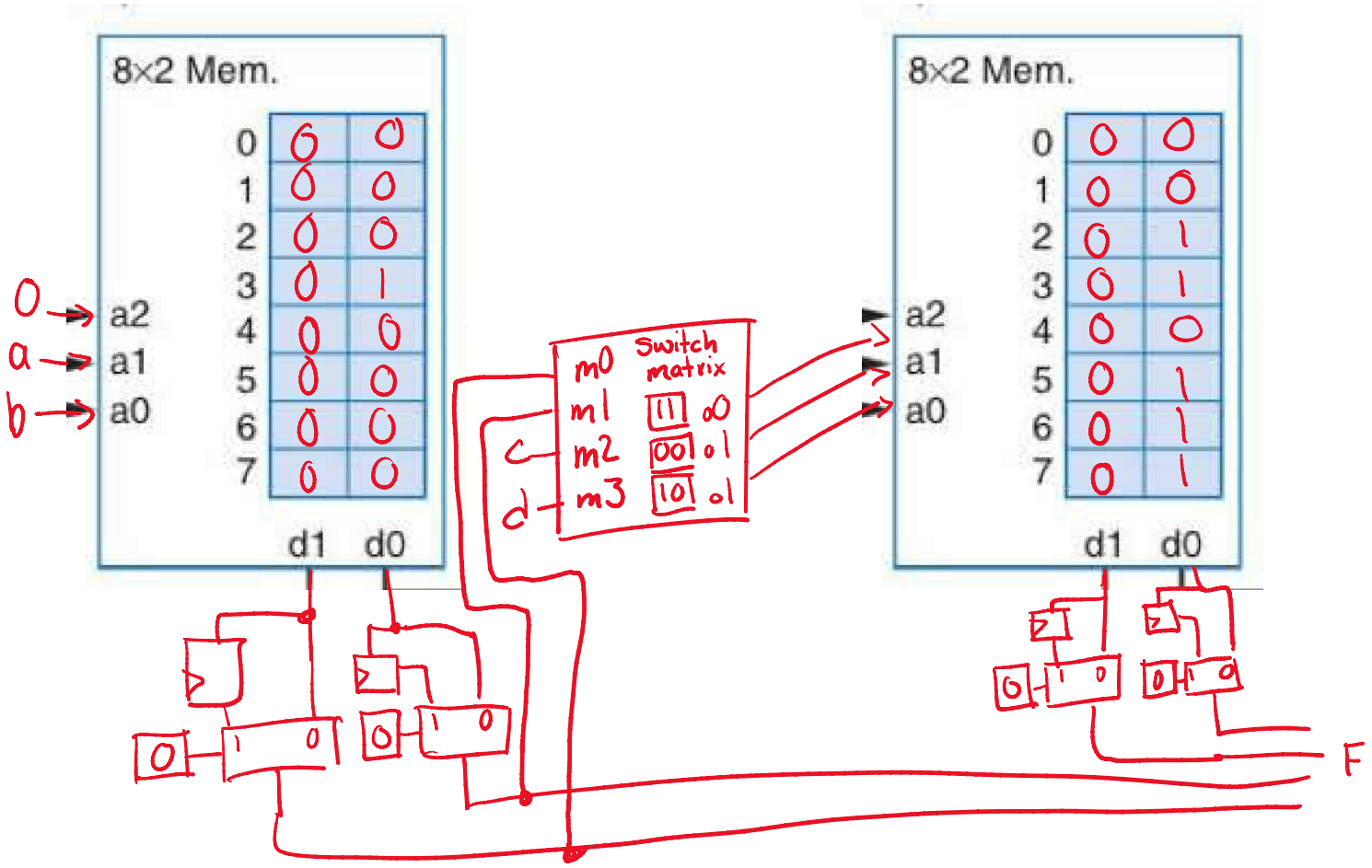
tables and custom connections among the lookup tables.

7.24 Show how to implement a 4-bit carry-ripple adder using any number of 3-input 2-input lookup tables and custom connections among the lookup tables. Hint: map one full-adder to each lookup table.



among the lookup tables.

7.27 Show the bit file necessary to program the FPGA fabric in Figure 7.31 to implement the function `F(a, b, c, d) = ab + cd`, where `a`, `b`, `c`, and `d` are external inputs.


8×2 Mem.
0
1
2
3
4
5
6
7
a2
a1
a0
d1
d0
m0
m1
m2
m3
Switch matrix
c
d
11
00
001
10
01
01
8×2 Mem.
a2
a1
a0
d1
d0
F