Digital Design Hwk 6

Friday, November 29, 2019 1:07 PM

4.30 Convert the following two's complement binary numbers to decimal numbers:
(a) 11100000
(b) 01111111
(c) 11110000
(d) 11000000
(e) 11100000

a) -32 b) 127 c) -16 d) -64 e) -32

4.31 Convert the following 9-bit two's complement binary numbers to decimal numbers:
(a) 011111111
(b) 111111111
(c) 100000000
(d) 110000000
(e) 111111110

a) 255 b) -1 c) -256 d) -128 e) -2

(g) –2

4.34 Convert the following decimal numbers to 8-bit two's complement binary form:
(a) 6
(b) 26
(c) –8
(d) –30
(e) –60
(f) –90

a) 00000110
b) 00011010
c) 11111000
d) 11100010
e) 11000100
f) 10100110

4.35 Convert the following decimal numbers to 9-bit two's complement binary form:
(a) 1
(b) –1
(c) –256
(d) –255
(e) 255
(f) –8
(g) –128

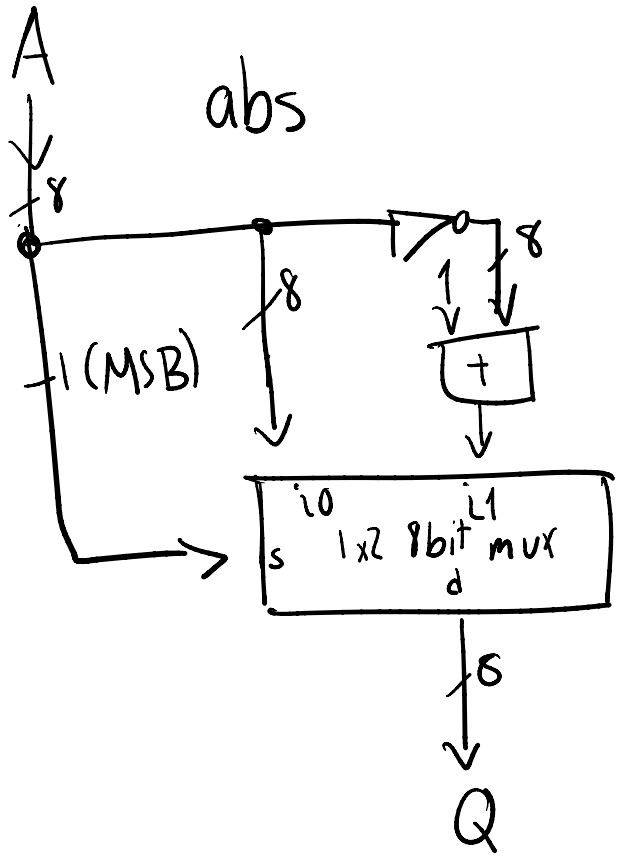
a) 000000001
b) 111111111
e) 011111111
f) 111111000

c) 1000000000

g) 1100000000

d) 1000000001

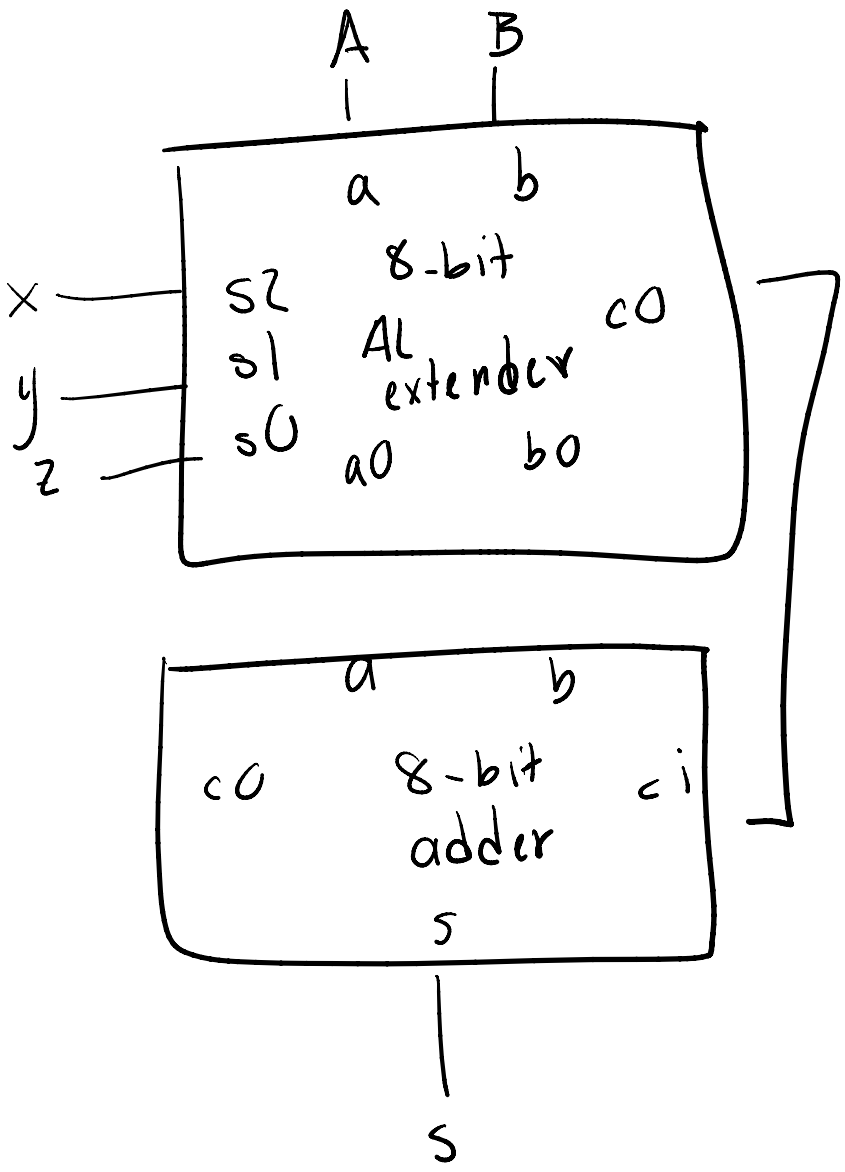
4.38 Create an absolute value component *abs* with an 8-bit input *A* that is a signed binary number, and an 8-bit output *Q* that is unsigned and that is the absolute value of *A*. So if the input is `00001111` (+15) then the output is also `00001111` (+15), but if the input is `11111111` (–1) then the output is `00000001` (+1).



4.40 Design an ALU with two 8-bit inputs *A* and *B*, and control inputs x, y, and z. The ALU should support the operations described in Table 4.3. Use an 8-bit adder and an arithmetic/logic extender. *(Component design problem.)*

TABLE 4.3 Desired ALU operations.

Inputs			Operation
x	y	z	
0	0	0	S = A – B
0	0	1	S = A + B
0	1	0	S = A * 8
0	1	1	S = A / 8
1	0	0	S = A NAND B (bitwise NAND)
1	0	1	S = A XOR B (bitwise XOR)
1	1	0	S = Reverse A (bit reversal)
1	1	1	S = NOT A (bitwise complement)

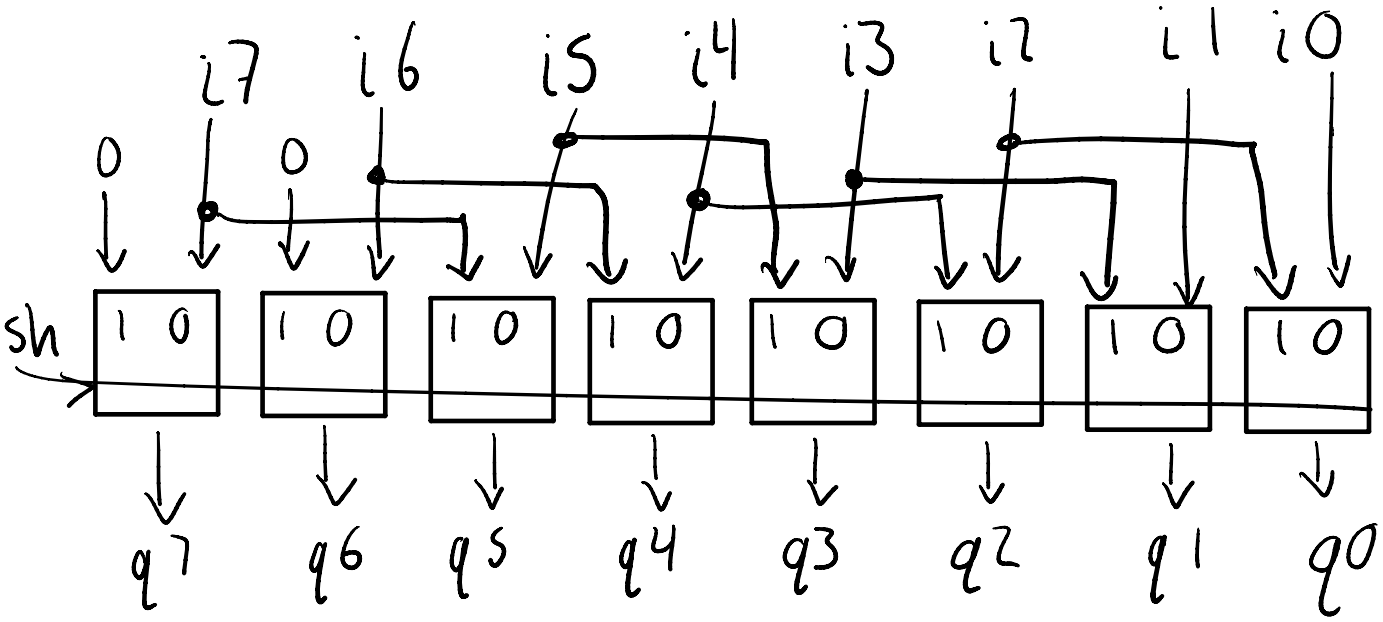


AL-extender

xyz	a0	b0	c0
000	a	b'	1
001	a	b	0
010	a<<3	0	0
011	a>>3	0	0
100	aNANDb	0	0

100	aNANDb	0	0
101	aXORb	0	0
110	a(reversed)	0	0
111	NOT a	0	0

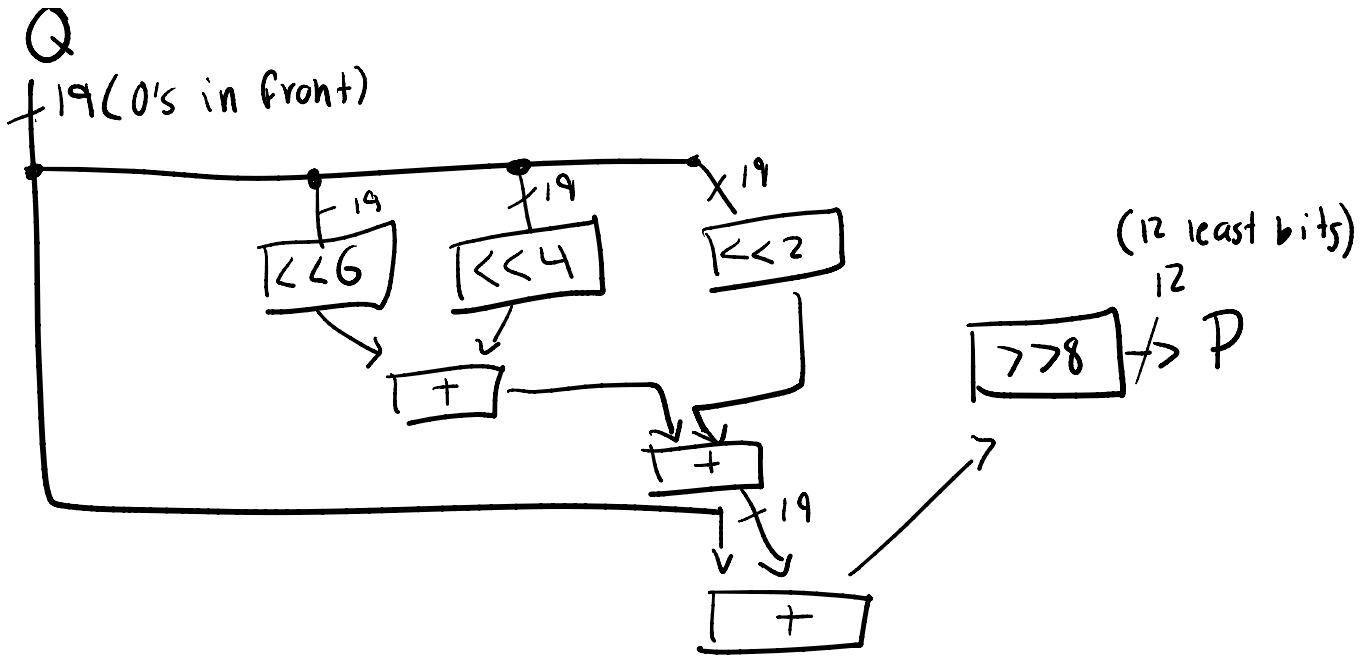
4.43 Design an 8-bit shifter that shifts its inputs two bits to the right (shifting in 0s) when the shifter's shift control input is 1. *(Component design problem.)*



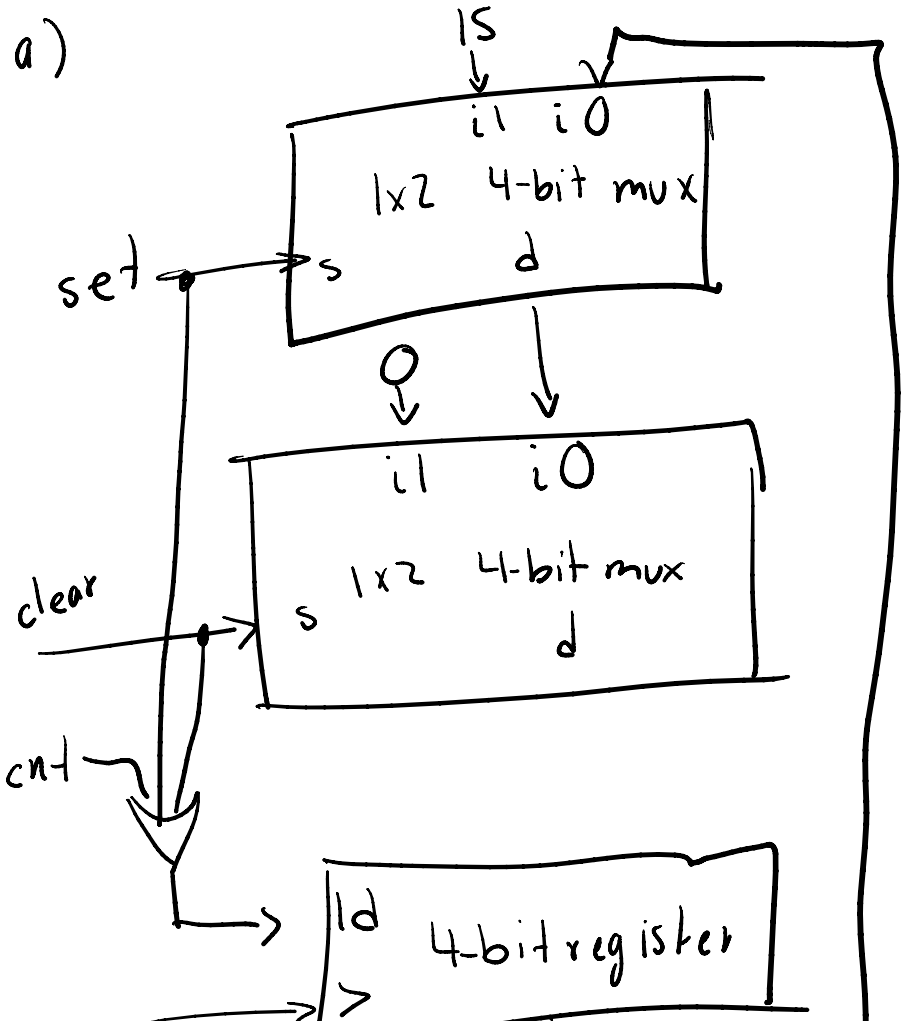
4.48 Use strength reduction to create a circuit that approximately computes $P=(1/3)*Q$ using only shifters and adders. Strive for accuracy to the hundredths place (0.33). P is a 12-bit output and Q is a 12-bit input. Use wider internal components and wires as necessary to prevent internal overflow.

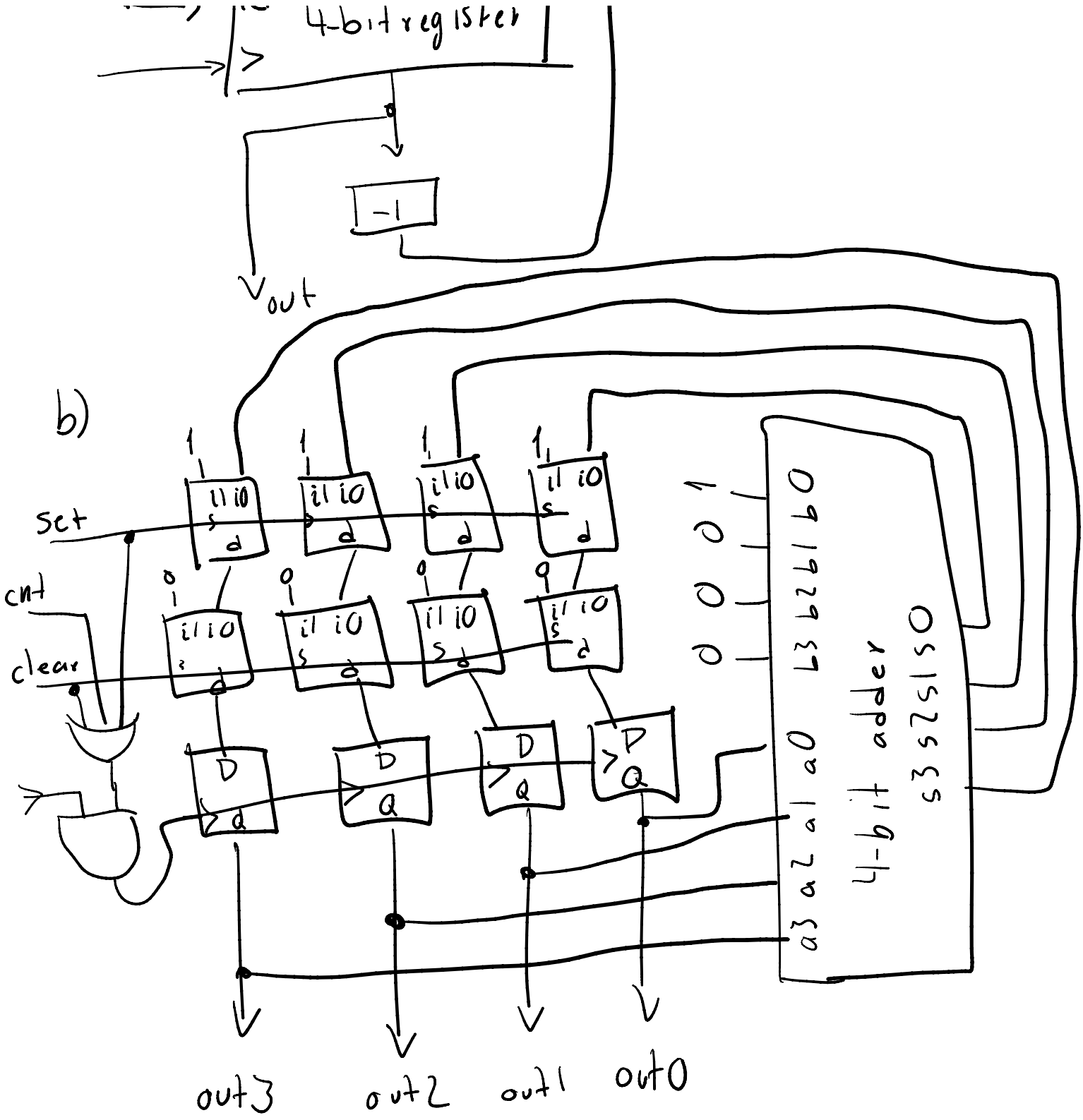
$$\frac{Q\cdot 64 + Q\cdot 16 + Q\cdot 4 + Q}{256} \rightarrow (Q<<6 + Q<<4 + Q<<2 + Q) >> 8$$

Q
1 sig bit in front)

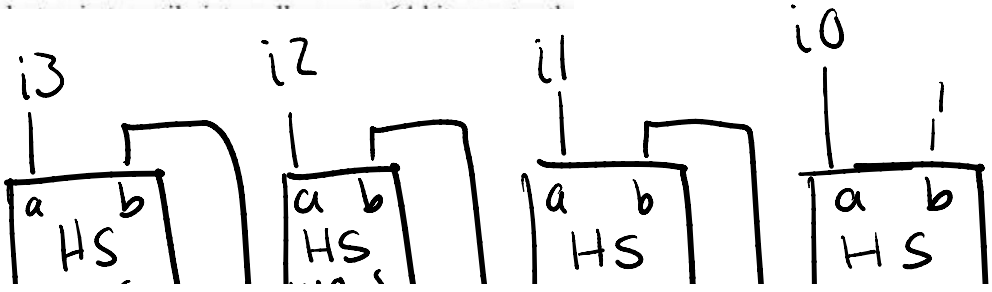


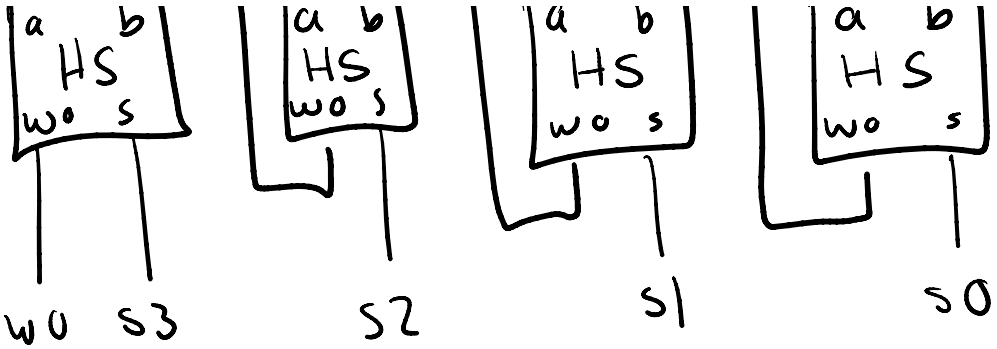
4.52 Design a 4-bit down-counter that has three control inputs: `cnt` enables counting up, `clear` synchronously resets the counter to all `0`s, and `set` synchronously sets the counter to all `1`s:
(a) using a parallel load register as a building block,
(b) using flip-flops and muxes by following the register design process of Section 4.2.
(Component design problem.)





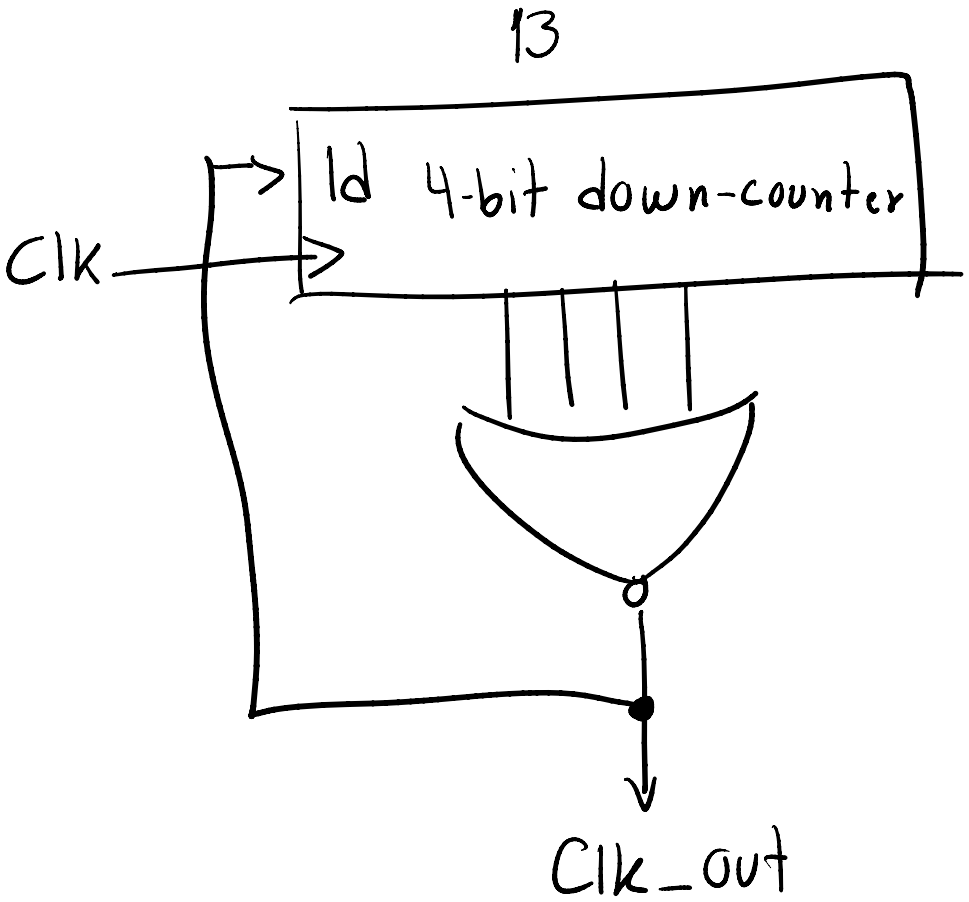
4.55 Design a circuit for a 4-bit decrementer. *(Component design probl*





(Component use problem.)

4.59 Create a clock divider that converts a 14 MHz clock into a 1 MHz clock. Use a down-counter with parallel load. Clearly indicate the width of the down-counter and the counter's load value. *(Component use problem.)*



4.64 A 4x4 register file's four registers initially each contain 0101.

(a) Show the input values necessary to read register 3 and to simultaneously write register 3 with the value 1110.

(b) With these values, show the register file's register values and output values before the next rising clock edge, and after the next rising clock edge.

a) W_data = 1110

b) Before rising edge:

W_data = 1110
W_addr = 11
W_en = 1
R_addr = 11
R_en = 1

R0: 0101
R1: 0101
R2: 0101
R3: 0101

R_data: 0101

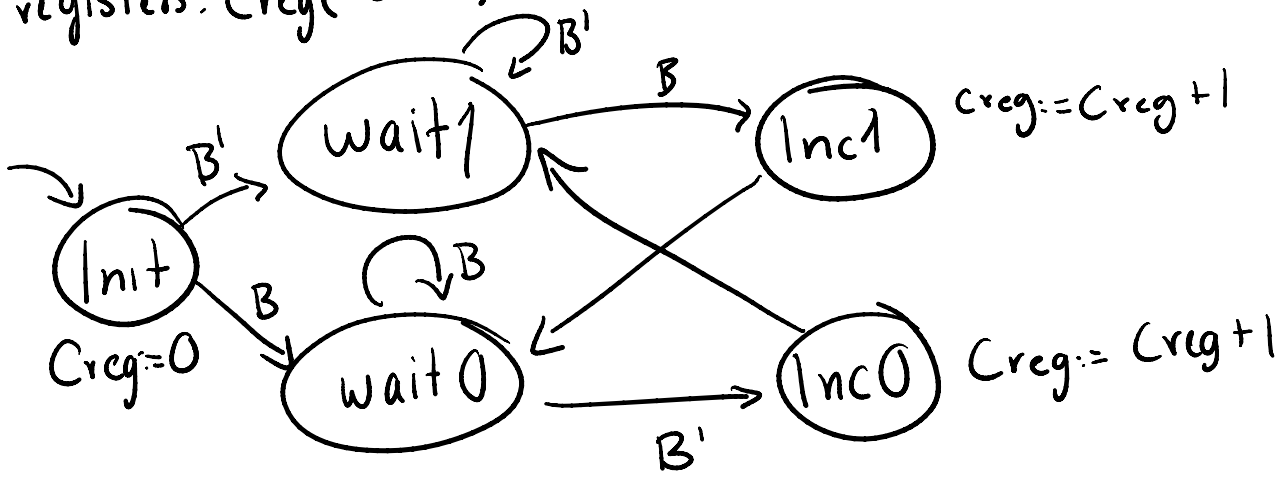
After rising edge:

R0: 0101
R1: 0101
R2: 0101
R3: 1110

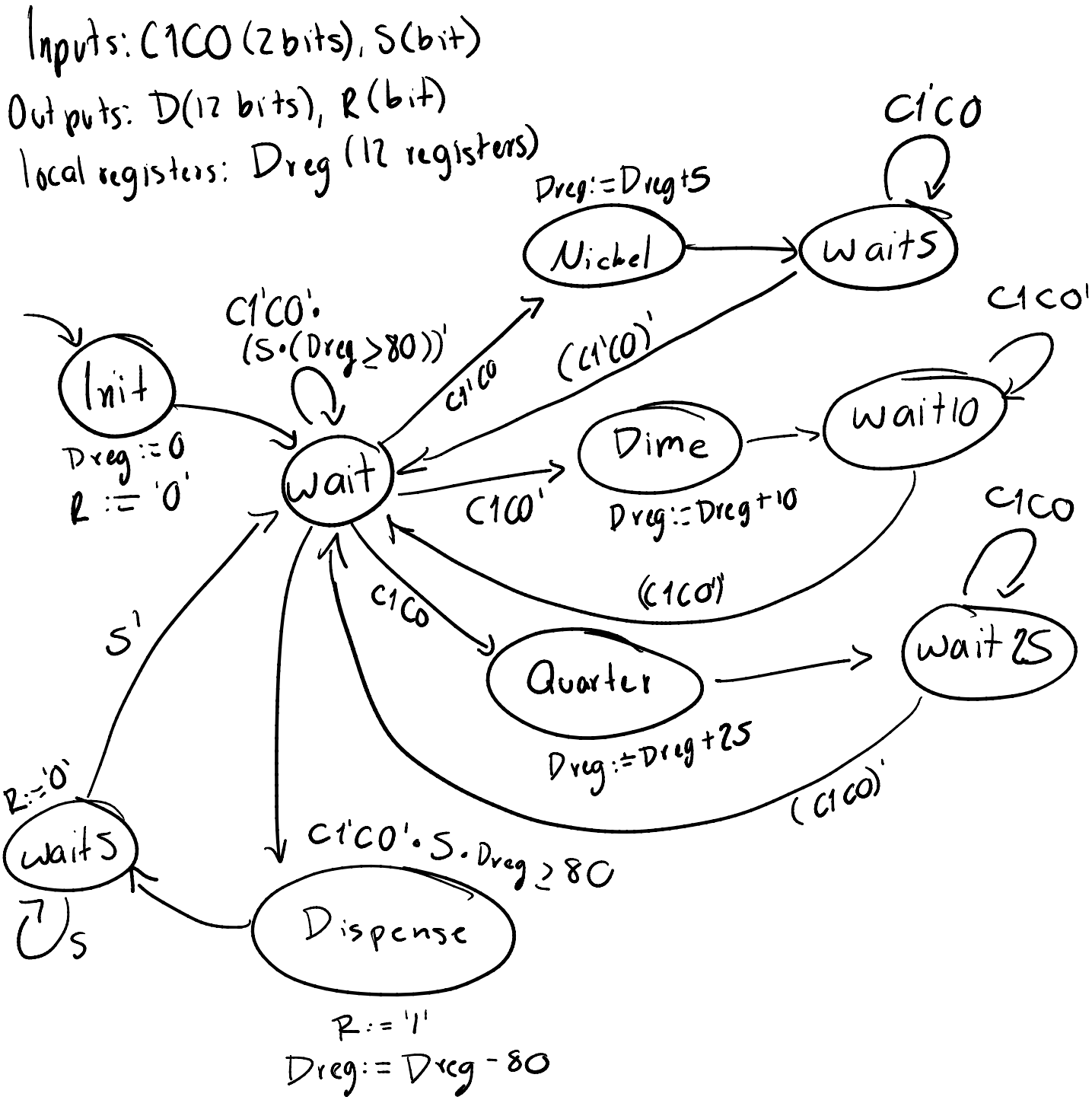
R_data: 1110

5.2 Capture the following system behavior as an HLSM. The system counts the number of events on a single-bit input B and always outputs that number unsigned on a 16-bit output *C*, which is initially 0. An event is a change from 0 to 1 or from 1 to 0. Assume the system count rolls over when the maximum value of *C* is reached.

Inputs: B(bit)
Outputs: C(16bit)
local registers: Creg(16bits)



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.



5.6 Create a high-level state machine for a simple data encryption/decryption device. If a single-bit input `b` is `1`, the device stores the data from a 32-bit signed input *I*, referring to this as an *offset* value. If `b` is `0` and another single-bit input `e` is `1`, then the device "encrypts" its input *I* by adding the stored offset value to *I*, and outputs this encrypted value over a 32-bit signed output *J*. If instead another single-bit input `d` is `1`, the device "decrypts" the data on *I* by subtracting the offset value before outputting the decrypted value over *J*. Be sure to explicitly handle all possible combinations of the three input bits.

Inputs: I(32 bits), b(bit), e(bit), d(bit)

Outputs: J(32 bits)

local registers: offset(32 bits), Jreg (32 bits)