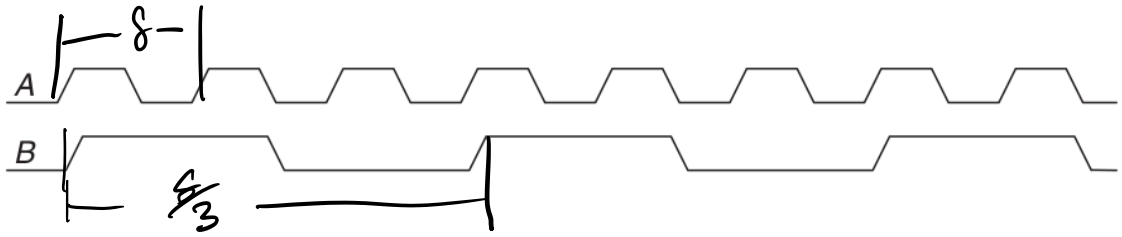
Name:

Homework 4

1. Sequential Logic Practice

a) Flip Flops and Gates

Question 3.8 Given signal *A*, shown in Figure 3.77, design a circuit that produces signal *B*.



You can assume that your flip flops have been properly reset.

Clock divide by 3

$\delta_{out} = \delta_{CLK} \times \frac{p}{2^N}$ w/ an N bit counter that adds p on each cycle

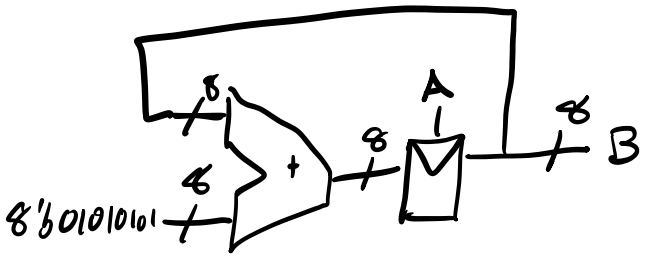
$\frac{\delta}{3} = \delta \cdot \frac{p}{2^N}$ $\frac{p}{2^N} = \frac{1}{3}$ $p = \frac{2^N}{3}$ 8 bit → 256

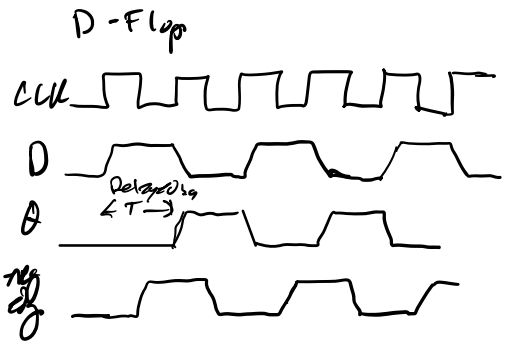
$\delta_{out} = \delta.33203$ N=8 p=85 $p = \frac{256}{3} = 85.33 \rightarrow 85$

16 bit → 65536

$\delta_{out} = \delta.33332$ N=16 p=21,845 $p = \frac{65536}{3} = 21845.33 \rightarrow 21,845$

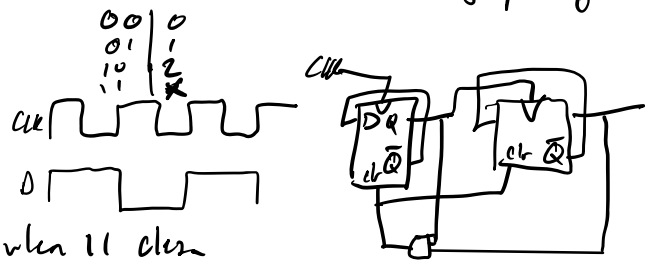
N = 8 p = 85 = 7'b1010101



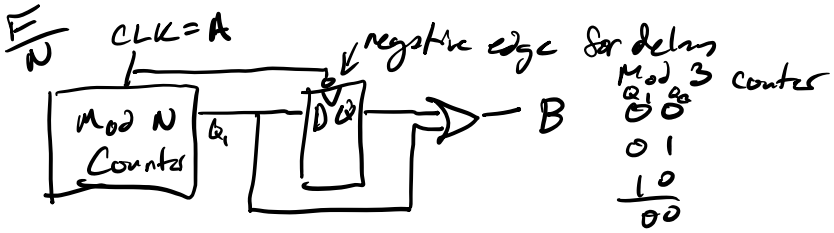


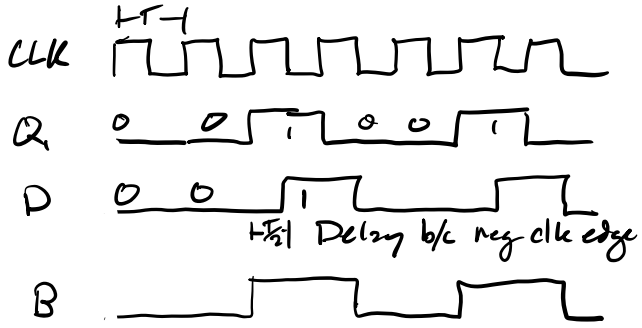
Mod N counter Asynchronous

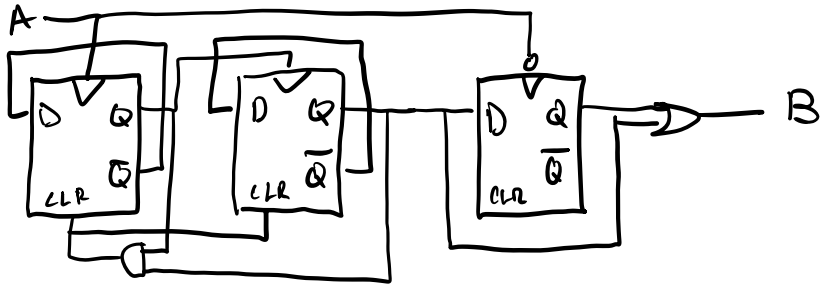
Mod 3 → 0 → 1 → 2 → 00, 01, 10 → 2 flops req



F/N







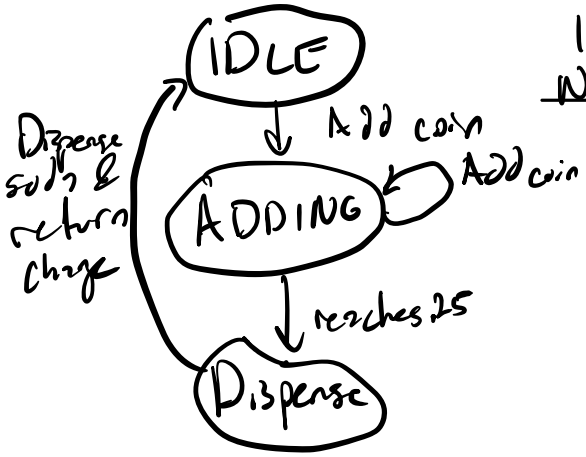
Still confused by this

b) FSMs

Exercise 3.26 You have been enlisted to design a soda machine dispenser for your department lounge. Sodas are partially subsidized by the student chapter of the IEEE, so they cost only 25 cents. The machine accepts nickels, dimes, and quarters. When enough coins have been inserted, it dispenses the soda and returns any necessary change. Design an FSM controller for the soda machine. The FSM inputs are *Nickel*, *Dime*, and *Quarter*, indicating which coin was inserted. Assume that exactly one coin is inserted on each cycle. The outputs are *Dispense*, *ReturnNickel*, *ReturnDime*, and *ReturnTwoDimes*. When the FSM reaches 25 cents, it asserts *Dispense* and the necessary *Return* outputs required to deliver the appropriate change. Then, it should be ready to start accepting coins for another soda.

.05, .10, .25

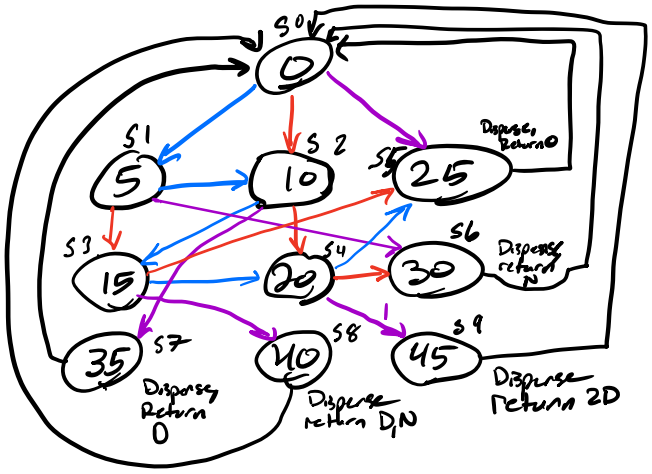
Draw the state diagram, then implement the FSM as a schematic (gates, muxes, adders, flops).
Bonus: make a verilog module from your schematic!



Inputs			Dispense	Return		
Nickel	Dime	Quarter		N	D	2D
0	0	1	1	0	0	0
0	1	0	0			
0	1	1	1	0	1	0
1	0	0	0			
1	0	1				

.05 .10 .15 .20 .25 + states

N = Nickel
D = Dime
Q = Quarter



S1 → S2 S3 S6
S2 → S3 S4 S7
S3 → S4 S5 S8
S4 → S5 S6 S9
S5 → S0
S6 → S0
⋮

Next State Logic

State	Input N D Q	Next State
0 S0 = 0 0 0 0	0 0 1	S5 = 0101
	0 1 0	S2 = 0010
	1 0 0	S1 = 0001
5 S1 = 0 0 0 1	0 0 1	S6 = 0110
	0 1 0	S3 = 0011
	1 0 0	S2 = 0010
10 S2 = 0 0 1 0	0 0 1	S7 = 0111
	0 1 0	S4 = 0100
	1 0 0	S3 = 0011
15 S3 = 0 0 1 1	0 0 1	S8 = 1000
	0 1 0	S5 = 0101
	1 0 0	S4 = 0100
20 S4 = 0 1 0 0	0 0 1	S9 = 1001
	0 1 0	S6 = 0110
	1 0 0	S5 = 0101
25 S5 = 0 1 0 1	X X X	S0 = 0000
30 S6 = 0 1 1 0	X X X	S0
35 S7 = 0 1 1 1	X X X	S0
40 S8 = 1 0 0 0	X X X	S0
45 S9 = 1 0 0 1	X X X	S0

Output Logic

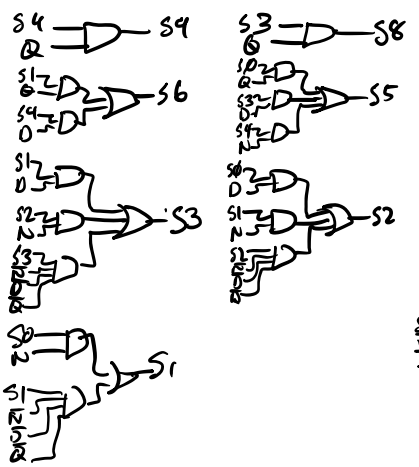
State	Dispense	N	D	Q
S5	1	0	0	0
S6	1	1	0	0
S7	1	0	1	0
S8	1	1	0	0
S9	1	0	1	0

Next State

$S0 = S5 + S6 + S7 + S8 + S9 + S0\bar{N}\bar{D}\bar{Q}$
$S1 = S0N + S1\bar{N}\bar{D}\bar{Q}$
$S2 = S0D + S1N + S2\bar{N}\bar{D}\bar{Q}$
$S3 = S1D + S2N + S3\bar{N}\bar{D}\bar{Q}$
$S4 = S2D + S3N + S4\bar{N}\bar{D}\bar{Q}$
$S5 = S0Q + S3D + S4N$
$S6 = S1Q + S4D$
$S7 = S2Q$
$S8 = S3Q$
$S9 = S4Q$

Output

$D_{1:3} = S5 + S6 + S7 + S8 + S9$
$N = S6 + S8$
$D = S7 + S8$ $D_L = S9$



$\overline{S0} = S5 + S6 + S7 + S8 + S9 + S0\bar{N}\bar{D}\bar{Q}$

$S1 = S0N + S1\bar{N}\bar{D}\bar{Q}$

$S2 = S0D + S1N + S2\bar{N}\bar{D}\bar{Q}$

$S3 = S1D + S2N + S3\bar{N}\bar{D}\bar{Q}$

$S4 = S2D + S3N + S4\bar{N}\bar{D}\bar{Q}$

$S5 = S0Q + S3D + S4N$

$S6 = S1Q + S4D$

$S7 = S2Q$

$S8 = S3Q$

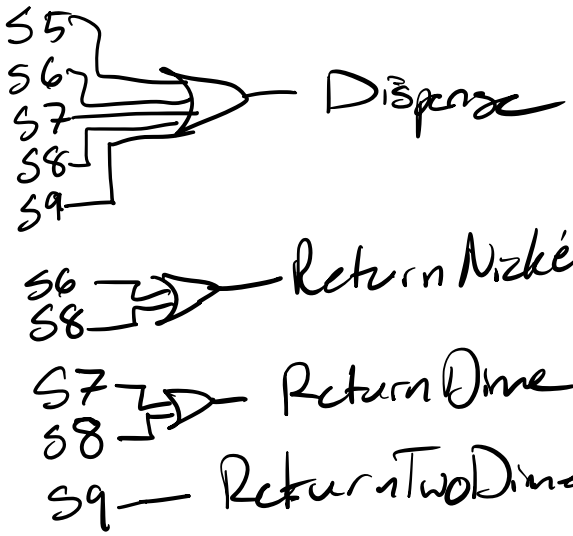
$S9 = S4Q$

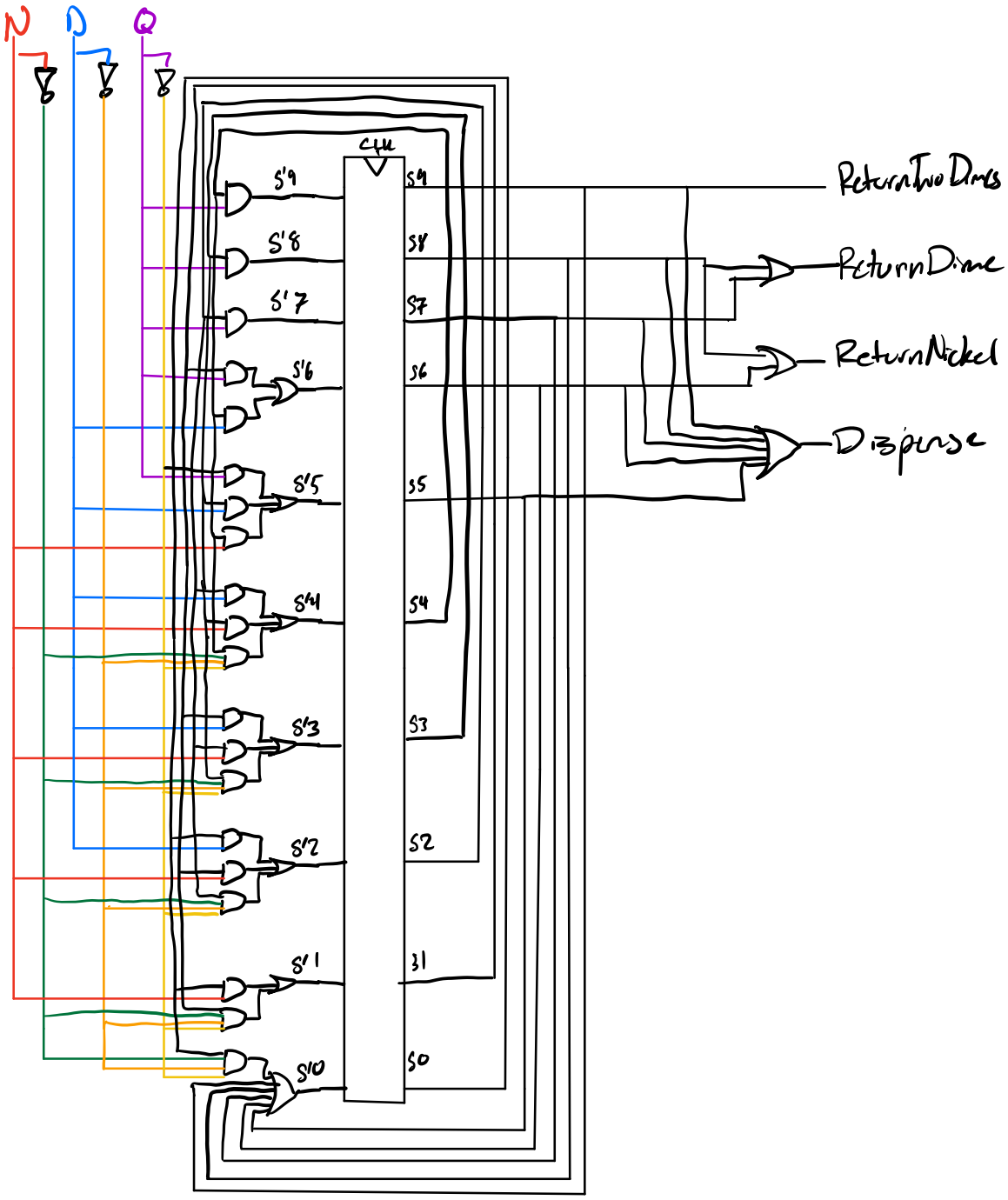
Output

$D_3 = S5 + S6 + S7 + S8 + S9$

$N = S6 + S8$

$D = S7 + S8 \quad D_2 = S9$




N
D
Q
CLK
S'9
S'8
S'7
S'6
S'5
S'4
S'3
S'2
S'1
S'10
S9
S8
S7
S6
S5
S4
S3
S2
S1
S0
ReturnTwoDimes
ReturnDime
ReturnNickel
Dispense

c) Reasoning about Metastability

Exercise 3.38 You are walking down the hallway when you run into your lab partner walking in the other direction. The two of you first step one way and are still in each other's way. Then, you both step the other way and are still in each other's way. Then you both wait a bit, hoping the other person will step aside. You can model this situation as a metastable point and apply the same theory that has been applied to synchronizers and flip-flops. Suppose you create a mathematical model for yourself and your lab partner. You start the unfortunate encounter in the metastable state. The probability that you remain in this state after t seconds is $e^{-\frac{t}{\tau}}$. τ indicates your response rate; today, your brain has been blurred by lack of sleep and has $\tau = 20$ seconds.

(a) How long will it be until you have 99% certainty that you will have resolved from metastability (i.e., figured out how to pass one another)?

(b) You are not only sleepy, but also ravenously hungry. In fact, you will starve to death if you don't get going to the cafeteria within 3 minutes. What is the probability that your lab partner will have to drag you to the morgue?

(a) Prob not in state: $P = 1 - e^{-t/20}$

$.99 = 1 - e^{-t/20}$ $\quad e^{-t/20} = .01$ $\quad \ln e^{-t/20} = \ln .01$

$\frac{-t}{20} = \ln .01$ $\quad t = -20(\ln .01) = \boxed{92.103\text{s}}$

(b) Prob still in metastable state $P = e^{-t/20}$ $\quad t = 180\text{s}$

$P = e^{-\frac{180}{20}} = \boxed{0.0001234}$ $\quad \boxed{0.01234\%}$

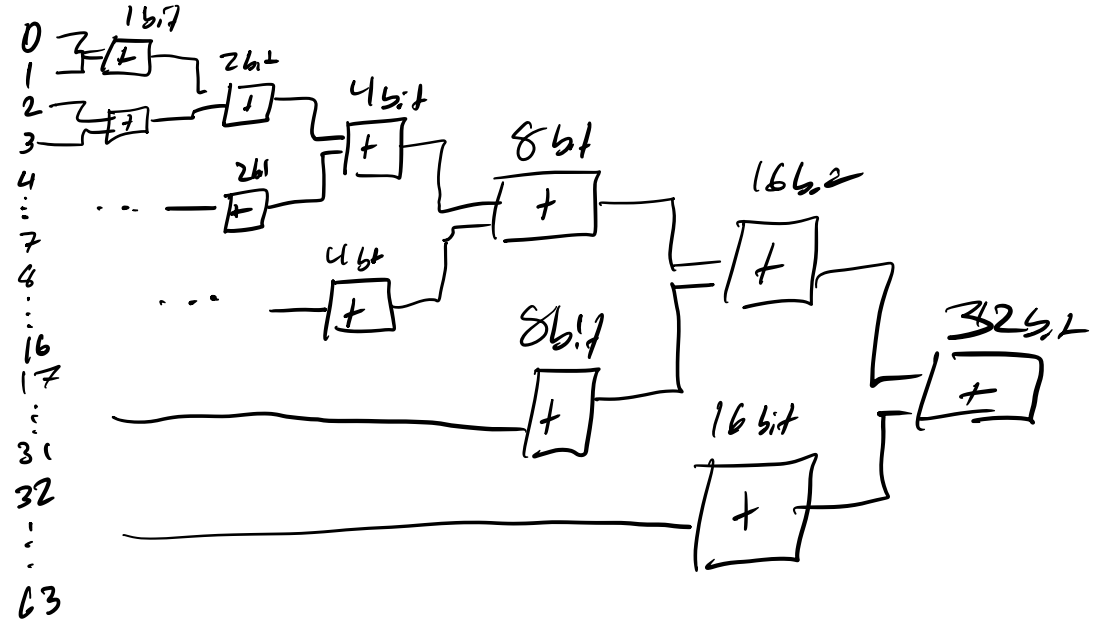
2. Combination Logic Review + HDL Practice: ALU

An arithmetic logic unit is a critical building block for a CPU. It does a variety of mathematical operations on two inputs (addition, subtraction, comparison, etc.). The next few problem sets will have us build up the different units inside it. For this problem set we are going to build a 32 bit wide 32:1 MUX and a 32 bit wide adder. Assemble a zip file with the following:

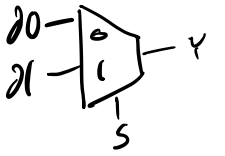
- a mux32.sv file
- an add32.sv file
- all submodule files that you used to make the above
- testbenches that give you confidence that you implemented them correctly
- a Makefile that runs your tests
- a README.md file with
    - A description of how you implemented the modules (you can include pictures or reference notes in your homework pdf)
    - A description of how you tested the mux32.
    - How to run your tests

You may use any of the basic gates for this task (and, or, xor, nor, nand, not, mux2). This cannot have any sequential logic. See the examples directory in the git repository for how to approach this!

Bonus: there's enough code in the examples to make a ripple carry adder, but that's the slowest way to do this. Instead, implement a faster adder, like Carry Look Ahead!

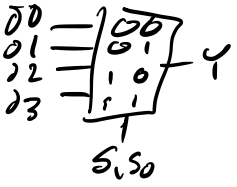


2:1 mux



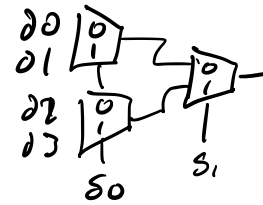
input			
S	d0	d1	Y
1			d1
0			d0

if s=1, then y=d1
if s=0, then y=d0



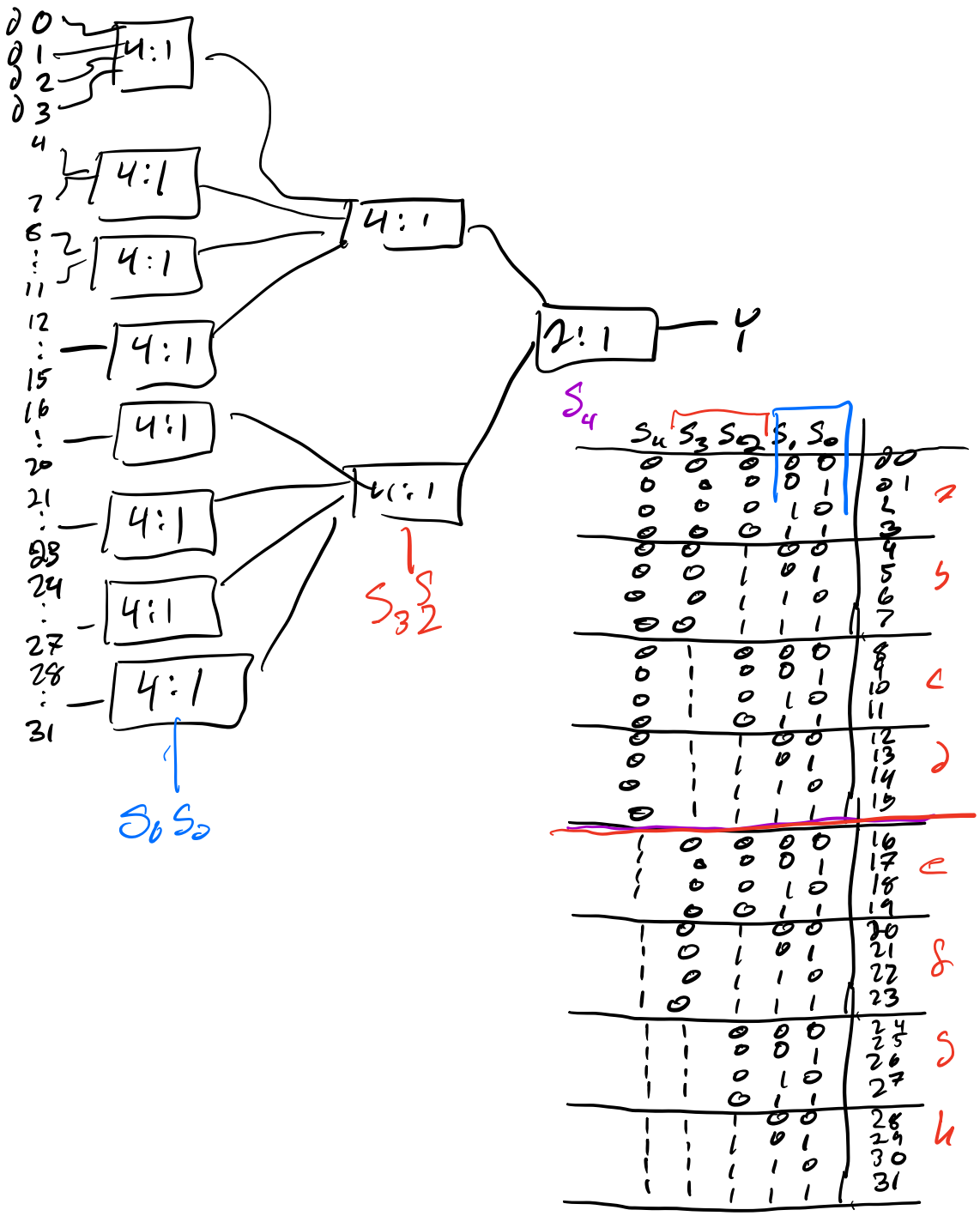
S_1	S_0	Y
0	0	d0
0	1	d1
1	0	d2
1	1	d3

S_1 ? (S_0 ? d3 : d2)
: (S_0 ? d1 : d0)



mux2(d0, d1, s[0], z)
mux2(d2, d3, s[0], w)
mux2(z, w, s[1], y)

d0
d1
d2
d3
d4
d5
d6
d7
d8


0
1
2
3
4
7
6
5
11
12
15
16
20
21
23
24
27
28
31
4:1
4:1
4:1
4:1
4:1
4:1
4:1
4:1
4:1
2:1
Y
S_4
S_3 S_2
S_0 S_1
S_4 S_3 S_2 S_1 S_0
7
5
4
2
e
8
b
d

Metadata

How long did the reading take you? 6

How long did the written part take you? 2

How long did the Verilog part take you? 3