Lab 3:

Project 3:: Sequential Lab

By. Melvin Evans
(ID#4692)

CSC 137
Professor Daryl Posnett
April 22, 2021

Problem Number One:

1. Implement a 5 bit CLA and then use this to create a 5 bit 2's complement adder/subtractor. You should have two five bit inputs, A and B. These inputs represent the data inputs for your adder. Include an input bit m that controls the mode, as discussed in lecture and in your book. You must implement overflow. Include an output Cout that represents the carry out of the adder. No masking needs to occur in this circuit. R represents the result.
    A. You may use any gates that you wish, i.e., you may choose and SOP or a POS approach, however, you must use basic gates AND/OR/NOT/XOR throughout.
    B. You may use multiplexors.
    C. Implement your 5-bit adder as a sub-circuit so that you can include it in a larger test circuit.
    D. Implement a separate test circuit sub-circuit that includes dip switches and led outputs so that you can easily test your adder/subtractor.

Solution:

Truth table for addition

A	B	C	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

K-Maps

Cout:

c\ab	00	01	11	10
0			1	
1		1	1	1

Sum:

c\ab	00	01	11	10
0		1		1
1	1		1	

Simplified Equations

Cout = ab + (a XOR b) * c

Sum = a XOR b XOR c

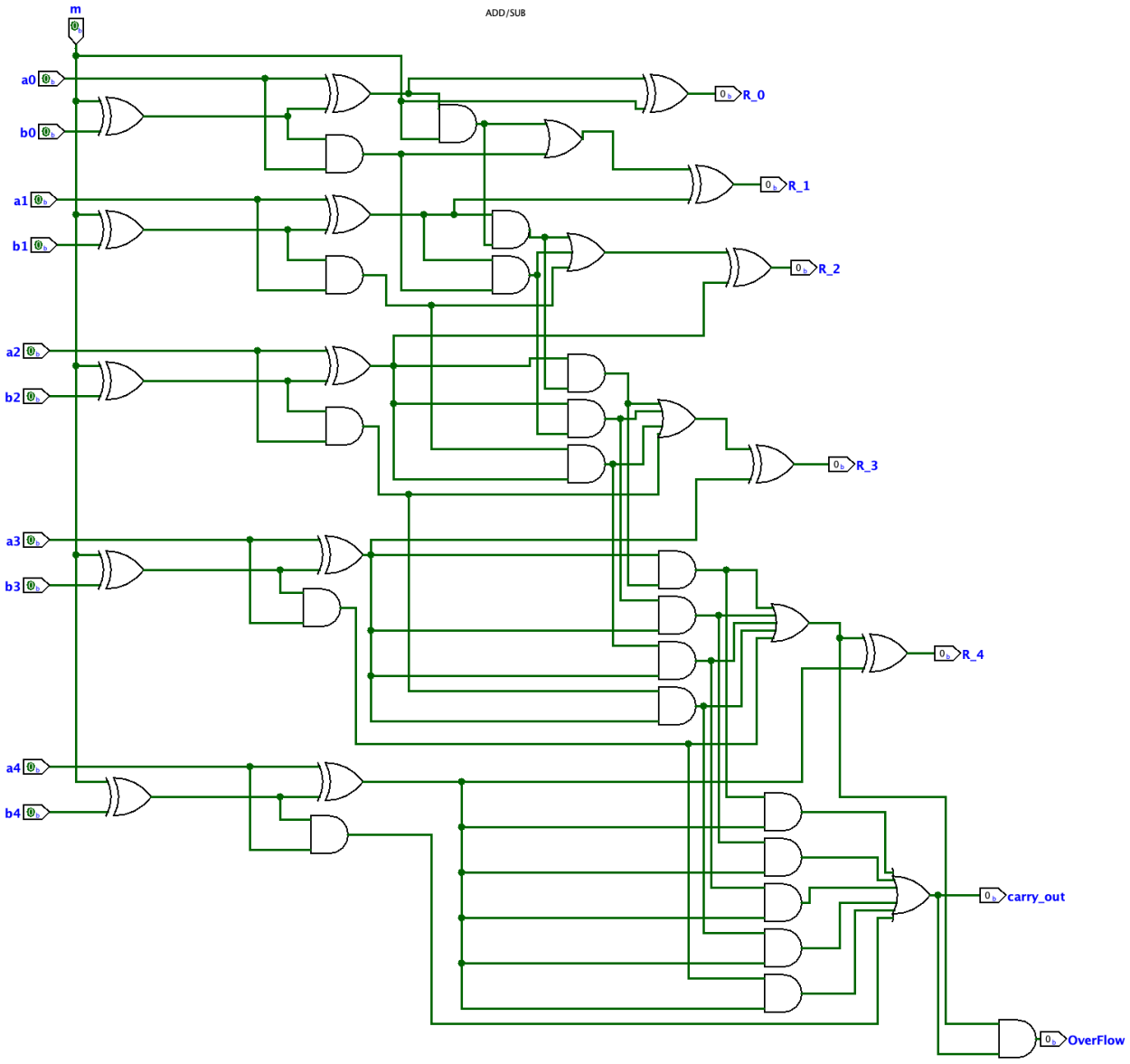
Subtraction

The only difference between subtraction and addition is that for subtraction you have to flip all of the bits of the second input and then add one. To allow this circuit to do addition and subtraction we will use to the m bit to represent that. Where when m=0 we will do addition, which follows the above truth table and when m=1 we will invert all bits and set the carry in to 1 as well. In order to accomplish this we will tie the carry in bit to the m input. Making it so that carry in will only be 1 when m=1. Along with that the circuit will run the b input through an XOR gate with m as the second input. This is because when m=0 you will still get returned the right output. While when m=1 the b input will be inverted which is what we want for subtraction. To show proof here is the truth table for this transaction.

B	m	Out
0	0	0
0	1	1
1	0	1
1	1	0

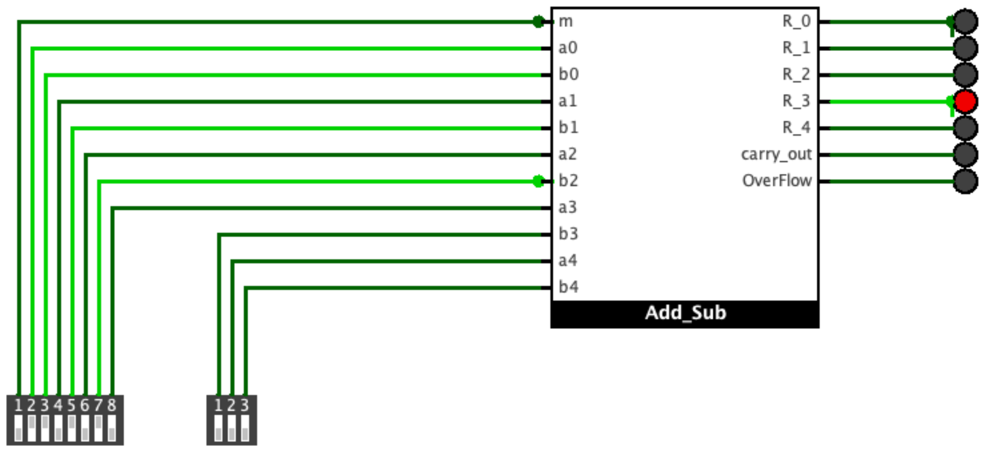
This is the same truth table as an XOR gate. With this and m being tied to the carry in there is now the ability to create the circuit which looks like this.

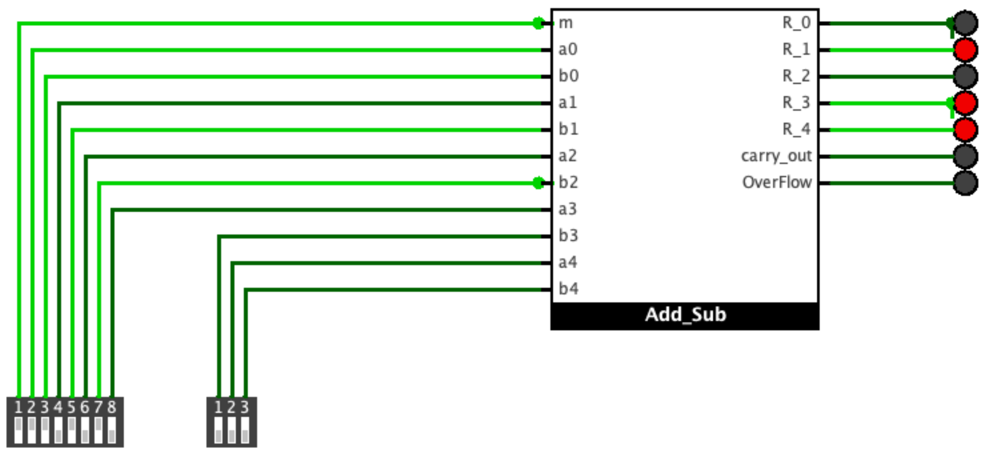
Add / Sub Circuit

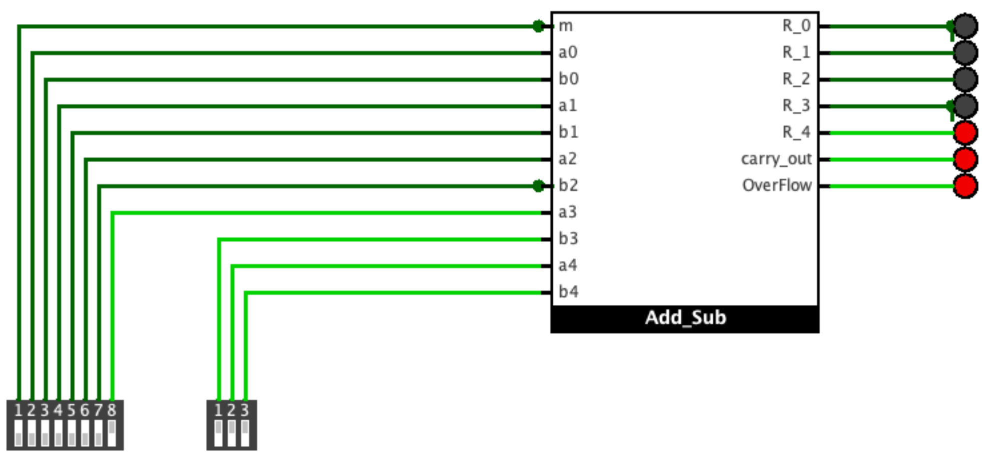


Testing

To test this circuit I used multiple test cases to not put too many here are three of them below which are 1+7, 1-7, and 24+24 in that order.


m
a0
b0
a1
b1
a2
b2
a3
b3
a4
b4
R_0
R_1
R_2
R_3
R_4
carry_out
OverFlow
Add_Sub
1 2 3 4 5 6 7 8
1 2 3


m
a0
b0
a1
b1
a2
b2
a3
b3
a4
b4
R_0
R_1
R_2
R_3
R_4
carry_out
OverFlow
Add_Sub
1 2 3 4 5 6 7 8
1 2 3


m
a0
b0
a1
b1
a2
b2
a3
b3
a4
b4
R_0
R_1
R_2
R_3
R_4
carry_out
OverFlow
Add_Sub
1 2 3 4 5 6 7 8
1 2 3

Problem Number Two:

2. Implement a 5 bit Arithmetic Logic Unit as described in the table below. You must use your CLA adder/subtractor from this lab. There are some special requirements for this ALU as follows.
   A. XOR and NOT must be implemented using an XOR gate only. Thus you must properly route data into the XOR gate to accomplish the NOT function. Note: This is just one block, you may not duplicate this block. Both opcode 6 and 6 from the table below will both run the data through this same block.
   B. You must use existing modules e.g., ADD/SUB, ROR, and ASL to create the MUL2 and MUL3 datapaths. You may not use more than one of each type in your ALU. This means that you need to carefully think about how to route data in your datapath. I suggest that you first think about how to build the datapath for each element separately, then think about how to optimize them. You may use other gates as necessary for control and you may use multiplexors as necessary for routing.
   C. You must generate a carry out and an overflow out. Overflow indicates arithmetic overflow. Carry simply indicates that the carry bit is set. One or both must be masked when appropriate: Overflow and carry are set on addition and subtraction as discussed in the book and in the lecture.
   D. Overflow is set on MUL2 and MUL3 whenever the result cannot be contained in the six bits formed by the 5 bit result register and the carry out. In this sense, overflow can be thought of as the seventh bit of these arithmetic operations. Both carry and overflow are masked (i.e., not set) for GRAY, ROR, and Logic operations You may not use any logic other than basic gates AND/OR/NOT/XOR and multiplexors.

OP	Code	Description
ADD	0	R = A+B (signed)
SUB	1	R=A-B (signed)
MUL2	2	R=2*A (unsigned)
MUL3	3	R=3*A (unsigned)
ASL	4	R=A shifted one bit left through carry, shift zero into low bit
ROR	5	R=A rotated right one bit (do not rotate through carry)
XOR	6	R=A XOR B
NOT	7	R = NOT A
AND	8	R = A AND B
OR	9	R = A OR B

Solution:

Op – Codes

Add and subtract are taken from the first problem. To do MUL 2 we could create another sub circuit for it but since we are also doing a bitwise shift left by 1, we can just use that as the MUL 2. Since when you bitwise shift any

number to the left by 1 you will always double the number. Therefore, we might as well reuse to lower the number of gates. Next is MUL 3. The difference between a MUL 2 and a MUL 3 is that MUL 3 is just MUL 2 + A. This is the case since multiplication is just repeated addition. So, since there is a one term difference between the two all we would need is to add one more A to the result of MUL 2. ASL and ROR are just bitwise shifts meaning we just have to move the input to the right output or carry bit to get the answer. No need for any gates. XOR, AND, OR are just three normal gates so there is nothing to say about those. Last code is NOT. Due to us having to only use an XOR gate to invert the number we can look at a truth table and show that all we would need to invert the number is to make sure one input is always one.

Truth table

X	A	out
0	0	0
0	1	1
1	0	1
1	1	0

As shown above when the first input X is 0 then A's output will be the same as what was fed into the XOR gate. While when X is 1 A's output will be inverted. Therefore, this proves that if we make the XOR gate always have an input as 1 and the other as A then the number would be inverted.

ALU Component: Map

This component will set the select inputs for all of the MUX's used inside of the ALU. Therefore, it includes an m, msk, and select inputs 1-6. With it having 4 inputs.

Truth Table

	F3	F2	F1	F0	M	msk	S3	S2	S1	S6	S5	S4
Add	0	0	0	0	0	0	0	0	0	0	0	0
Sub	0	0	0	1	1	0	0	0	1	0	0	0
Mul2	0	0	1	0	0	0	0	1	0	0	0	0
Mul3	0	0	1	1	0	0	0	1	1	0	0	0
Asl	0	1	0	0	0	0	1	0	0	0	0	0
Ror	0	1	0	1	0	0	1	0	1	0	0	0
Xor	0	1	1	0	0	1	D	D	D	0	0	1
Not	0	1	1	1	0	1	D	D	D	0	1	0
And	1	0	0	0	0	1	D	D	D	0	1	1
Or	1	0	0	1	0	1	D	D	D	1	0	0

Due to the ones after OR not mattering due to them being don't cares they were not included in this table to shorten the table. S1, S2, S3 are the select bits for the arithmetic and S4, S5, S6 are for selecting which value to output.

K-Maps

S3 = F2, S2 = F1, S1 = F0, S6 = (~F3*F2*F1*~F0), S4 = F0

S5:

F1,f0\ f3,f2	00	01	11	10
00				
01				
11		1	1	
10				

Simplified equation = f2*f1*f0

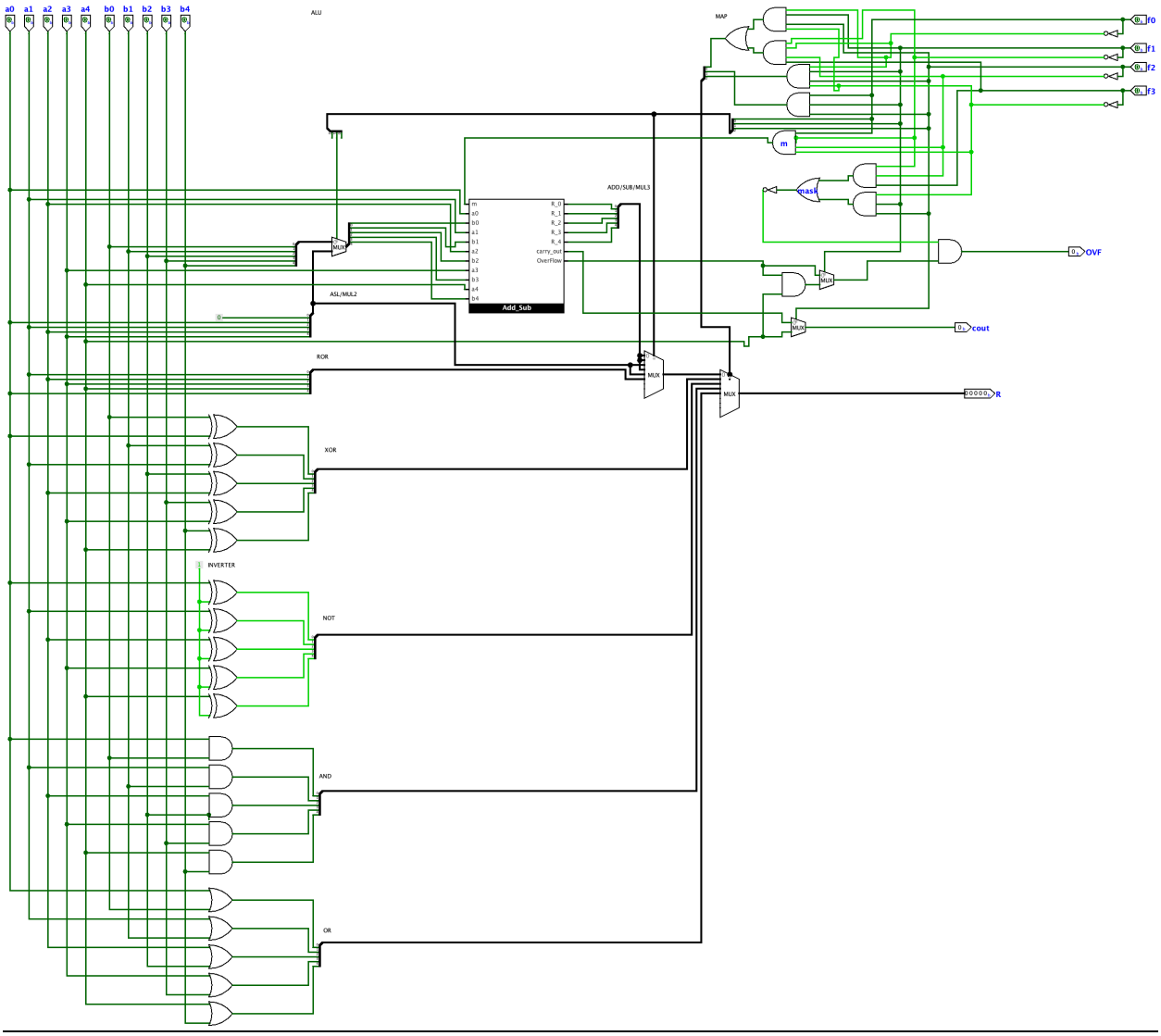
Msk:

F1,F0\F3,F2	00	01	11	10
00				1
01				1
11		1		
10		1		

Simplified equation = F3*~F2*~F1 + ~F3*F2*F1

ALU

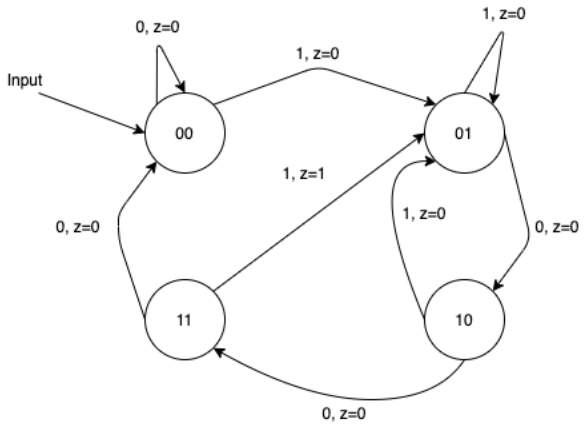
To achieve an ALU we would combine all of the parts of the op-codes and the map along with the mask in order to compose our circuit. The overall circuit including all these pieces and MUX's needed to control it look like the below circuit.


a0
a1
a2
a3
a4
b0
b1
b2
b3
b4
ALU
MAP
f0
f1
f2
f3
m
mask
ADD/SUB/MUL3
MUX
Add_Sub
m
a0
b0
a1
b1
a2
b2
a3
b3
a4
b4
R_0
R_1
R_2
R_3
R_4
carry_out
OverFlow
OVF
cout
ASL/MUL2
ROR
XOR
INVERTER
NOT
AND
OR
R

Problem Number Three

3. Problem 5.11 in your textbook. You must include your sequence diagram and schematics in your write-up.
    A. Design your OG and NSG circuits as sub-modules.
    B. Design your sequential circuit so that it includes your sub-modules. Your result should replicate the overall look of figure 5.12 in your book.
    C. You must also include complete schematics of your OG and NSG circuits in your write-up.
    D. You do not need to include a test circuit for this problem.
    E. Make sure you are as clear as possible in outlining all of your design steps.

Solution:

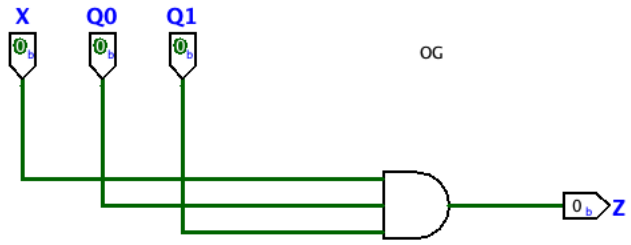


OG Truth Table

Q1	Q0	X	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Simplified equation: Q1*Q0*X

OG Circuit



NSG Truth Table

Q1	Q0	X	D1	D0
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	0	1

NSG K-Map

D1:

x\Q1,Q0	00	01	11	10
0		1		1
1				

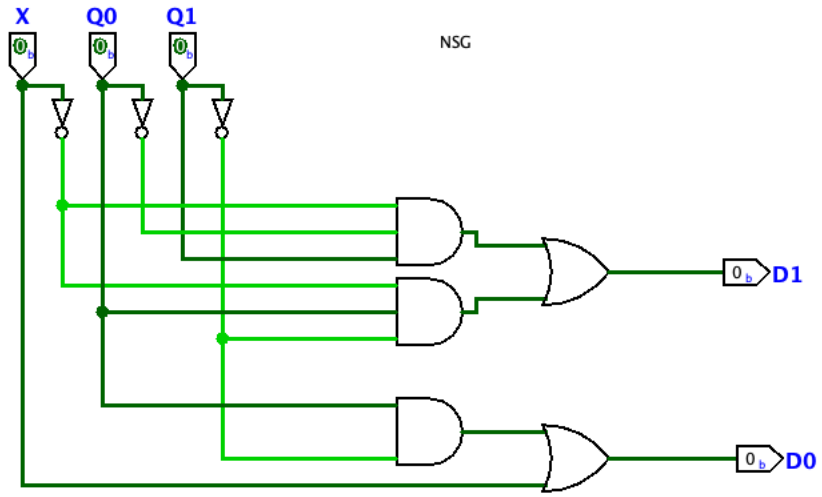
Simplified equation= ~Q1*Q0*~X + Q1*~Q0*~X

D0:

x\Q1,Q0	00	01	11	10
0				1
1	1	1	1	1

Simplified equation= Q1*~Q0 + X

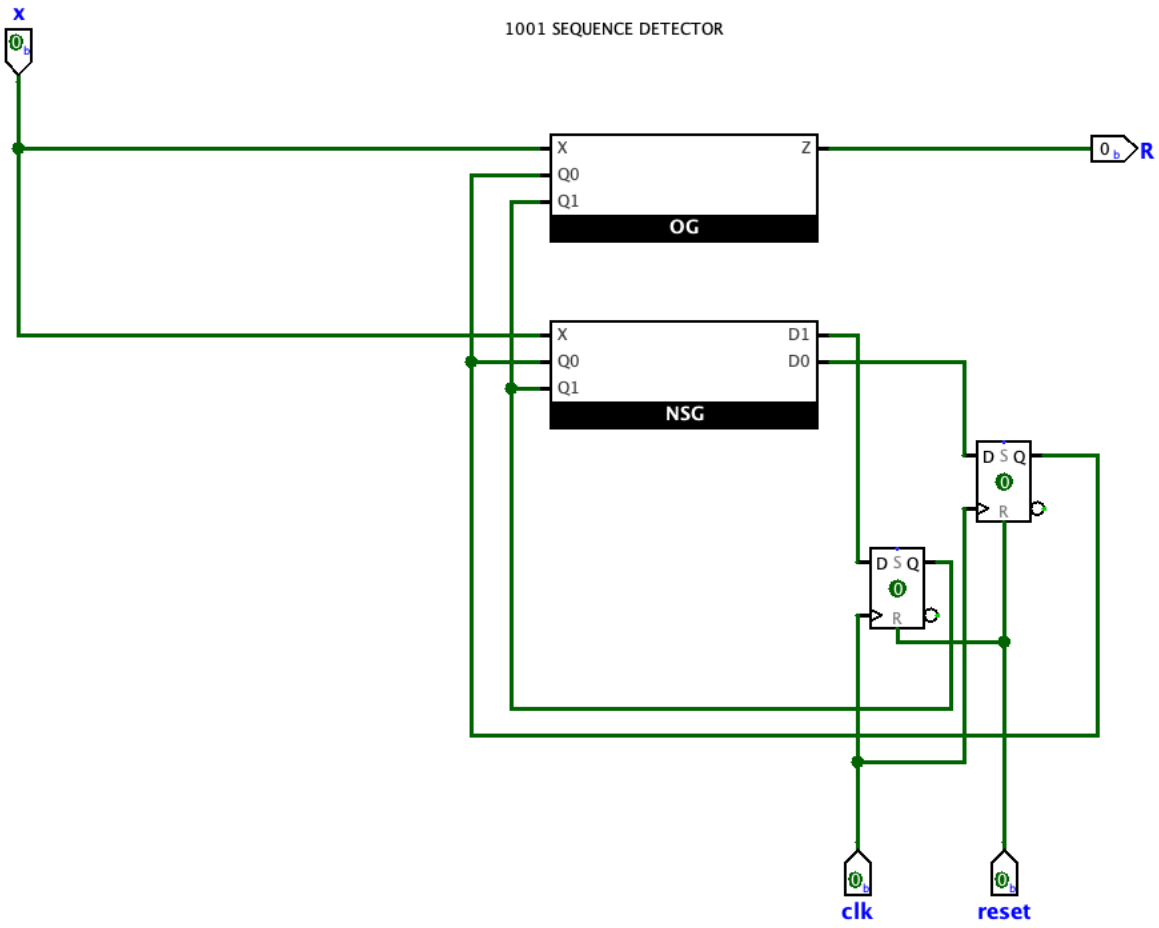
NSG Circuit



1001 Sequence Recognizer

For this design we can use the OG and the NSG from above combined with two D flip flops. There are only two flip flops being used as the maximum number of states can be shown in two bits. Due to the state labeled 3 being able to be shown in 2 bits. Then we just set both D flip flops to the same reset and clock to keep them synchronous. As shown below.

Sequence Recognizer Circuit

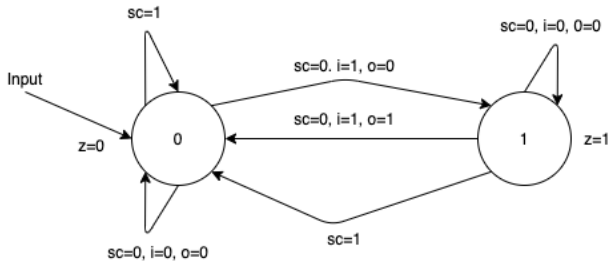


Problem Number Four

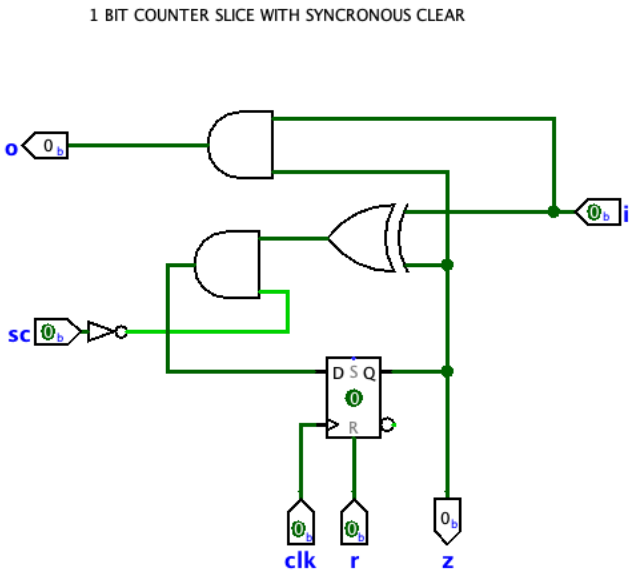
4. Problem 5.17 in your textbook. You must both design your counter and include details in your write-up as well as implement this in Logisim.
   A. Implement your counter as sub-circuit
   B. Implement a test circuit that operates your counter and displays the count output on LEDs

Solution

1 Bit Counter Finite State Diagram



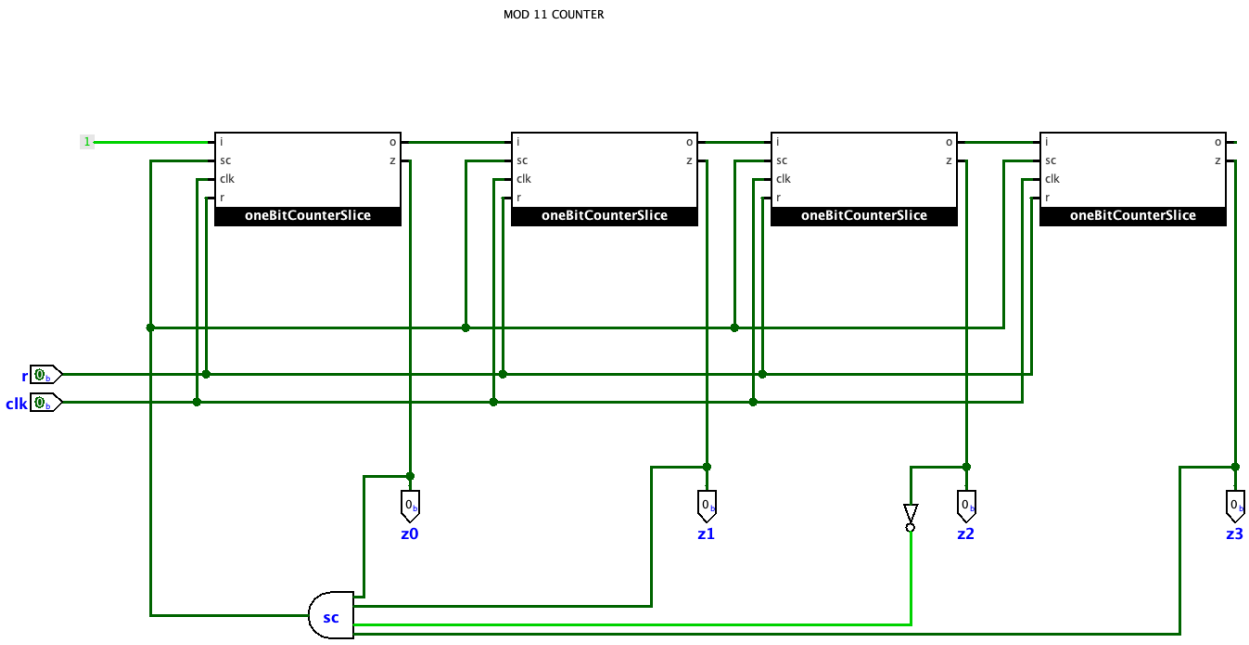
1 Bit Counter Circuit



11 Mod Counter

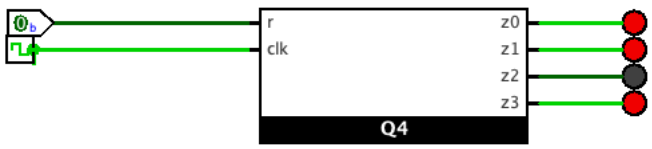
An 11-mod counter will display numbers counting from 0 – 11 in binary and then reset. Since it counts up to 11, we would need to use 4, 1-bit counter circuits done in serial. To know when to rest we will want to look at the binary version of 11 which is “1011”. This will be our synchronous clear which will end up being an AND gate of z3*~z2*z1*z0 so that once 11 is hit it clears all of them. To finish it off we just tie the rest signal so that the user can reset the circuit at any value and ad a clock to make it all synchronous.

11 Mod Counter Circuit



The 1 on the upper left is always active as it is needed to start the counter since the counter always goes up by one.

11 Mod Counter Test



Problem Number Five

5. Problem 6.17 in your textbook. For this problem you do not have to implement the design in Logisim. You also do not have to design the adder, multipler and divider blocks.
   A. Your goal with this assignment is to create a high level block diagram of the datapath that is complete. Identify the outputs of each arithmetic block and be sure to include the equation for the minimum clock period. Note, this is a generalized result. For example, you may specify the propagation delay of the multiplier as ΔMUL.
   B. Do both parts a and b with the same guidelines. That is, you are designing a high level datapath diagram and specifying the generalized equation for the minimum clock period in terms of the system modules.

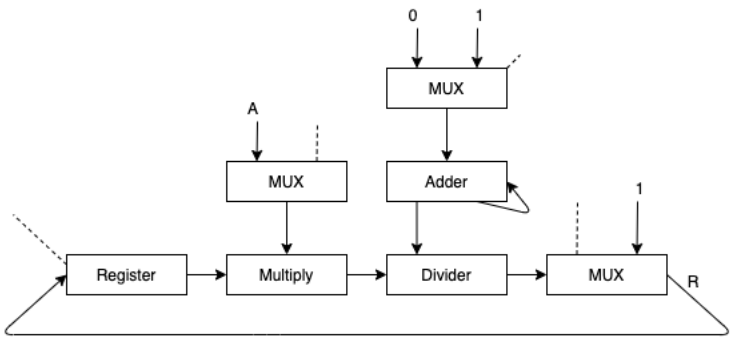
Solution

Part A:

Design

For this we can use an Adder, Divider, Multiplier, 3 MUX and a register. Starts at the Mux on the bottom right and loops around. The ordering of Multiply and divider were chosen this way due the mathematical information. Meaning that on the second rotation x/1 gets thrown into the multiply module which multiplies x/1 by x. This equals $x^2/1$ then this moves on to the divider which with the adder will divide it by 2 making it $x^2/2$. Then it pops out the result and the cycle continue.

Diagram



0 on the top MUX is there for the first addition as the value needs to output 1. So, 0+1 will help.

Equation

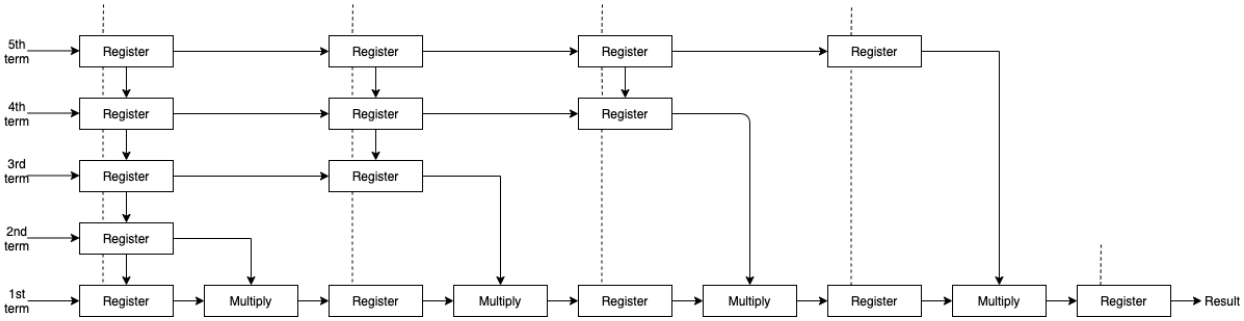
- $T \geq \Delta Add + \Delta Div + \Delta Mux + \Delta Mux + \Delta Mux + t_{st} + t_{cq} + t_{cs}$

Part B:

Design

In this design I assumed that all the terms were being fed into it are already divided. The reason being is that without that the divider would be done in serial and not in parallel which would defeat the purpose of doing pipeline. Overall, there are four stages in this equation as when you start the original value is already fed in. This is fed into the following stages of S = $register_{n-1}$ * $register_n$ where S is equal to stage. Creating a block diagram as shown below.

Diagram



Equation

- $T \geq \Delta Mul + t_{st} + t_{cq} + t_{cs}$