EEEN202 Design Exercise: Mars Rover Rotary Encoder

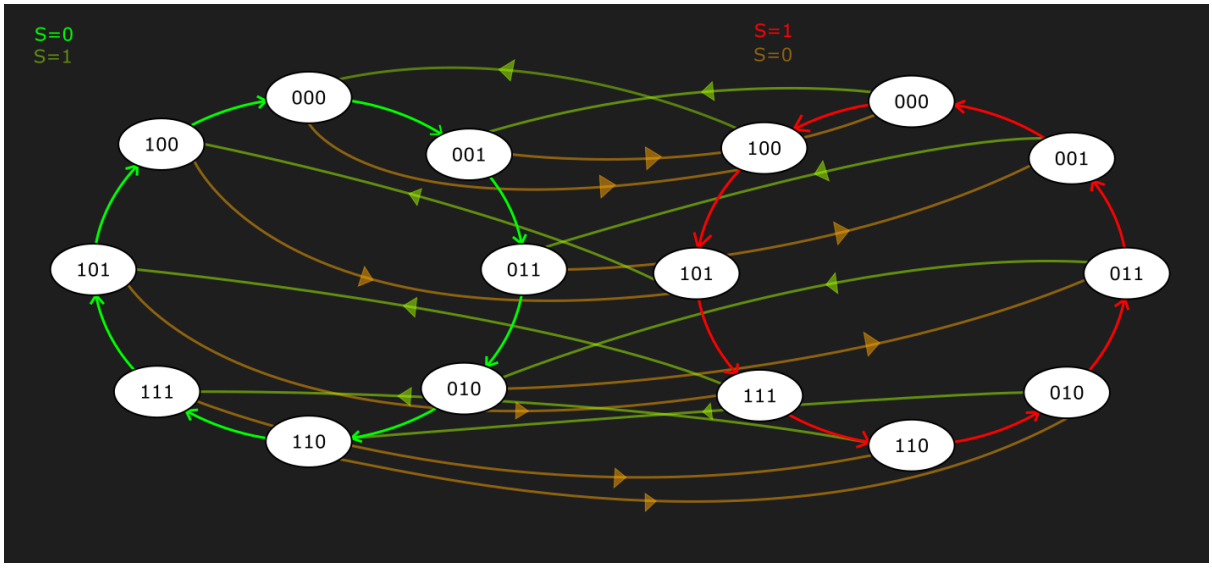
Task – design a measurement and control system for the motor driven by a rotary encoder. The rotations of the wheel are encoded in gray code, which is to be converted into binary and further processed to be used for system control.

The entire system is broken down into 5 subsystems:

1. Gray code generator
   - Gray code counter; used to simulate a motor rotating the wheel. Contains a control bit to change it from an up to down counter
2. Gray code to binary converter
   - Converting to binary for processing
3. Odometer
   - Counter used to measure the distance that the simulated wheel has travelled – 1m = 1 full rotation of the wheel. After it reaches 50m, it should output a signal to let the system know the rover should stop moving to let the batteries recharge
4. Motor control
   - Circuit for controlling the motors. Uses the signal from the odometer to turn off the motors when it reaches 50. After the batteries have recharged, the control system should reset the motors and odometer so it can begin a new cycle
5. Odometer display
   - Circuit for turning the binary numbers into a readable format i.e., seven segment displays driver circuit

Task 1 – Subsystem A: Gray Code counter

To simulate the rotation of the wheel of the Mars rover, a 3-bit gray code generator is to be implemented. A control bit S is also to be implemented that changes the counter from an up to down counter. There is no requirement for it be reset when the bit is flipped, so I am working under the assumption that it continues from where it left when the bit is flipped. Here is the transition diagram for such a system.



The design calls for DD flip flops to be used for the counter; the following is the excitation table for the transitions using them and then the K-Map and MUX simplifications:

Excitation Table for gray code counter

Switch	Present State Q(n)			Next State Q(n+1)			D FF states needed		
S	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	D_2	D_1	D_0
0	0	0	0	0	0	1	0	0	1
0	0	0	1	0	1	1	0	1	1
0	0	1	1	0	1	0	0	1	0
0	0	1	0	1	1	0	1	1	0
0	1	1	0	1	1	1	1	1	1
0	1	1	1	1	0	1	1	0	1
0	1	0	1	1	0	0	1	0	0
0	1	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	1	0	0
1	0	0	1	0	0	0	0	0	0
1	0	1	1	0	0	1	0	0	1
1	0	1	0	0	1	1	0	1	1
1	1	1	0	0	1	0	0	1	0
1	1	1	1	1	1	0	1	1	0
1	1	0	1	1	1	1	1	1	1
1	1	0	0	1	0	1	1	0	1

K Maps and SOP statements for each Input + MUX Simplification

NOTE: the shading in the tables used to indicate when the bit S is 0 or 1. Colour is used when the output is 1. This is helpful when simplifying MUXs and will be seen throughout the design process.

S	Q_2	Q_1	Q_0	D_2
0	0	0	0	0
0	0	0	1	0
0	0	1	1	0
0	0	1	0	1
0	1	1	0	1
0	1	1	1	1
0	1	0	1	1
0	1	0	0	0
1	0	0	0	1
1	0	0	1	0
1	0	1	1	0
1	0	1	0	0
1	1	1	0	0
1	1	1	1	1
1	1	0	1	1
1	1	0	0	1

D_2	$\bar{S}.\overline{Q_0}$	$\bar{S}.Q_0$	$S.Q_0$	$S.\overline{Q_0}$
$\overline{Q_2}.\overline{Q_1}$	0	0	0	1
$\overline{Q_2}.Q_1$	1	0	0	0
$Q_2.Q_1$	1	1	1	0
$Q_2.\overline{Q_1}$	0	1	1	1

$$D_2 = Q_2.Q_0 + \bar{S}.Q_1.\overline{Q_0} + S.\overline{Q_1}.\overline{Q_0}$$

S	Q_2	Q_1	Q_0	D_2
0	0	0	0	0
0	0	0	1	0
0	0	1	1	0
0	0	1	0	1
0	1	1	0	1
0	1	1	1	1
0	1	0	1	1
0	1	0	0	0

S	Q_2	Q_1	Q_0	D_2	MUX IN
1	0	0	0	1	S
1	0	0	1	0	0
1	0	1	0	0	$\bar{S}$
1	0	1	1	0	0
1	1	0	0	1	S
1	1	0	1	1	1
1	1	1	0	0	$\bar{S}$
1	1	1	1	1	1

S	Q_2	Q_1	Q_0	D_1
0	0	0	0	0
0	0	0	1	1
0	0	1	1	1
0	0	1	0	1
0	1	1	0	1
0	1	1	1	0
0	1	0	1	0
0	1	0	0	0
1	0	0	0	0
1	0	0	1	0
1	0	1	1	0
1	0	1	0	1
1	1	1	0	1
1	1	1	1	1
1	1	0	1	1
1	1	0	0	0

D_1	$\bar{S}.\overline{Q_0}$	$\bar{S}.Q_0$	$S.Q_0$	$S.\overline{Q_0}$
$\overline{Q_2}.\overline{Q_1}$	0	1	0	0
$\overline{Q_2}.Q_1$	1	1	0	1
$Q_2.Q_1$	1	0	1	1
$Q_2.\overline{Q_1}$	0	0	1	0

$$D_1 = Q_1.\overline{Q_0} + \bar{S}.\overline{Q_2}.Q_0 + S.Q_2.Q_0$$

S	Q_2	Q_1	Q_0	D_1
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	$\bar{S}$
1	0	1	0	1
1	0	1	1	$\bar{S}$
1	1	0	0	0
1	1	0	1	S
1	1	1	0	1
1	1	1	1	S

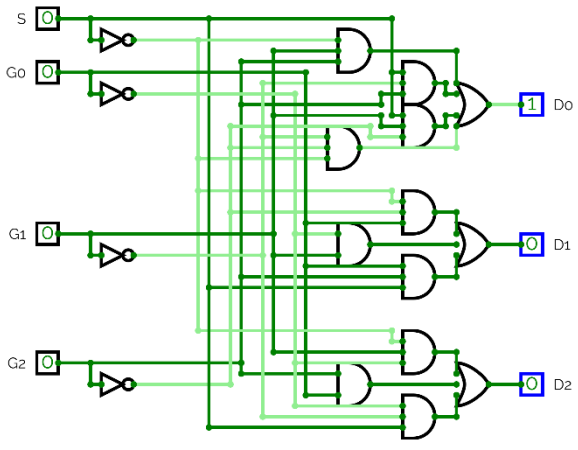
S	Q_2	Q_1	Q_0	D_1
0	0	0	0	1
0	0	0	1	1
0	0	1	1	0
0	0	1	0	0
0	1	1	0	1
0	1	1	1	1
0	1	0	1	0
0	1	0	0	0
1	0	0	0	0
1	0	0	1	0
1	0	1	1	1
1	0	1	0	1
1	1	1	0	0
1	1	1	1	0
1	1	0	1	1
1	1	0	0	1

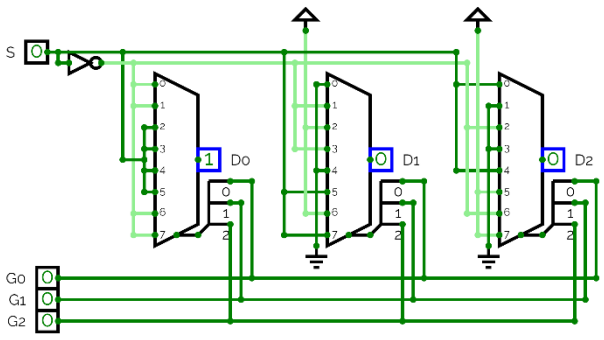
D_1	$\bar{S}.\overline{Q_0}$	$\bar{S}.Q_0$	$S.Q_0$	$S.\overline{Q_0}$
$\overline{Q_2}.\overline{Q_1}$	1	1	0	0
$\overline{Q_2}.Q_1$	0	0	1	1
$Q_2.Q_1$	1	1	0	0
$Q_2.\overline{Q_1}$	0	0	1	1

$$D_0 = \bar{S}.\overline{Q_2}.\overline{Q_1} + \bar{S}.Q_2.Q_1 + S.\overline{Q_2}.Q_1 + S.Q_2.\overline{Q_1}$$

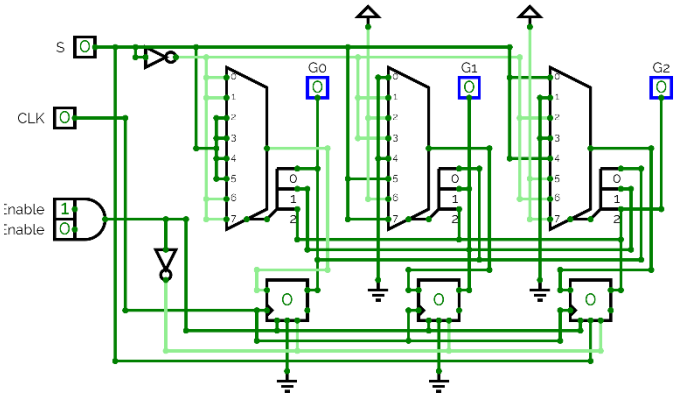
S	Q_2	Q_1	Q_0	D_1
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	$\bar{S}$
1	0	0	1	$\bar{S}$
1	0	1	0	S
1	0	1	1	S
1	1	0	0	S
1	1	0	1	S
1	1	1	0	$\bar{S}$
1	1	1	1	$\bar{S}$

Result of the following tables





Each circuit was tested by inputting the gray code by hand; both worked perfectly and showed the required inputs for the next transition. The result of the K-Map is messier and more complicated than the MUX implementation. Implementing in a clean manner was difficult and mistakes occurred frequently when trying to rout lines in a clean manner. In comparison, the lines for the MUXs are much simpler. There is the potential for even more simplification to a 4:1 MUX, but for now, going from a 16:1 to 8:1 MUXs is fine for testing and prototyping. Here is the resulting Gray Code counter using D FFs:



The ENABLES of all FFs are bound to an AND gate with two inputs: a master enable switch and an input for motor control for later. The RESET is also bound the same line but through a NOT – this resets all of the gates when the counter is disabled and prevents it from starting from anything other than the start of the sequence

Speaking of the start of sequences, the PRESET of each FF is a little different. G0 and G1 are pulled to ground/zero as normal, but the PRESET of G2 is S. This edit was done so when S = 1, the counter starts at 100 instead of 000; instead of starting from 000 and going to 100, it properly counts down to 000, cutting out the extra transition. This probably will not have much bearing on the system as a whole – it only occurs after the counter has reset once and the wheel turning 1/8 extra is most likely not going to be critical – but it is a painless fix that adds consistency.

Task 2 – Subsystem B: Gray code to Binary

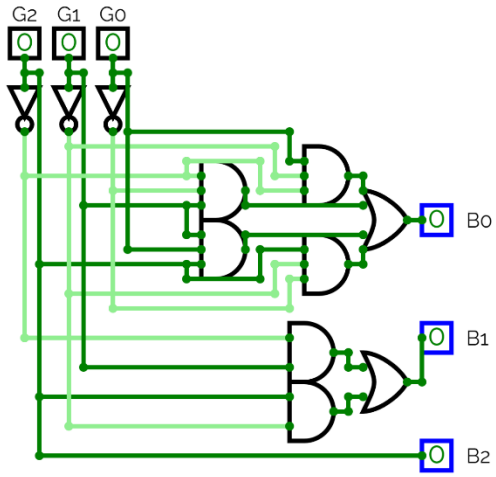
Now that the Gray Code generator is finished, we need to turn it into binary so it can be used elsewhere. The conversion table is below and the K-Map simplifications follows:

G_2	G_1	G_0		B_2	B_1	B_0
0	0	0		0	0	0
0	0	1		0	0	1
0	1	1		0	1	0
0	1	0		0	1	1
1	1	0		1	0	0
1	1	1		1	0	1
1	0	1		1	1	0
1	0	0		1	1	1

G_2	G_1	G_0	B_2		B_2	$\overline{G_0}$	G_0
0	0	0	0		$\overline{G_2}.\overline{G_1}$	0	0
0	0	1	0		$\overline{G_2}.G_1$	0	0
0	1	1	0		$G_2.G_1$	1	1
0	1	0	0		$G_2.\overline{G_1}$	1	1
1	1	0	1		$B_2 = G_2$		
1	1	1	1				
1	0	1	1				
1	0	0	1				

G_2	G_1	G_0	B_1		B_1	$\overline{G_0}$	G_0
0	0	0	0		$\overline{G_2}.\overline{G_1}$	0	0
0	0	1	0		$\overline{G_2}.G_1$	1	1
0	1	1	1		$G_2.G_1$	0	0
0	1	0	1		$G_2.\overline{G_1}$	1	1
1	1	0	0		$B_1 = \overline{G_2}.G_1 + G_2.\overline{G_1}$		
1	1	1	0				
1	0	1	1				
1	0	0	1				

G_2	G_1	G_0	B_0		B_0	$\overline{G_0}$	G_0
0	0	0	0		$\overline{G_2}.\overline{G_1}$	0	1
0	0	1	1		$\overline{G_2}.G_1$	1	0
0	1	1	0		$G_2.G_1$	0	1
0	1	0	1		$G_2.\overline{G_1}$	1	0
1	1	0	0		$B_0 = \overline{G_2}.\overline{G_1}.G_0 + \overline{G_2}.G_1.\overline{G_0} + G_2.G_1.G_0 + G_2.\overline{G_1}.\overline{G_0}$		
1	1	1	1				
1	0	1	0				
1	0	0	1				



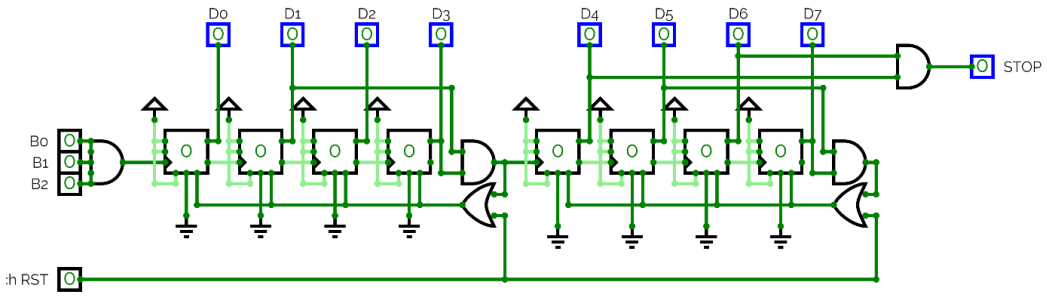
These give the following circuit – not insanely complicated, but somewhat messy. It takes the inputs of the gray code generator and outputs them in normal binary as described. B0 is the main problem in terms of complexity since it does not simplify at all. To solve this issue, I can implement it using a MUX and try and simplify it that way. However, this will do for the rest of the system for now; I may revisit this later if I have the time.

Regardless, the circuit works as intended after testing it with the gray code counter

Task 3 – Subsystem C: Odometer

Ideally, the odometer can tell when a full rotation occurs even while bit S is flipped multiple times. But to simplify the problem, we can ignore the cases when the rover reverses, or S = 1. The rover also travels 1m every time the wheel makes a full rotation. Using this, we can make the simplification that whenever the output of the gray code to binary decoder reaches 111, the rover has moved by one meter. Basically, to increment the odometer, I can set the inputs of the decoder to an AND gate. This is a great simplification and means it counts to only 7 for the first rotation before transitioning states and may need to be addressed. But for now, it will work fine.

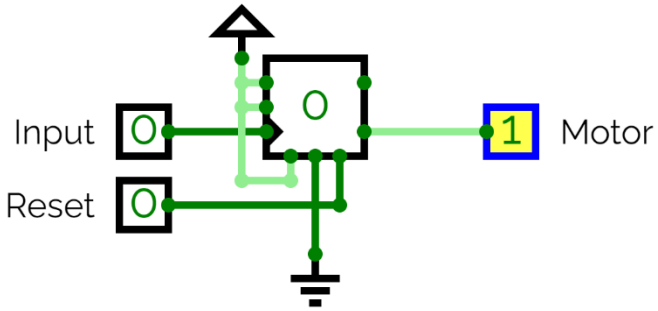
Another issue is the counter type. If I were to do binary, it would be easier to make, but since I need to account for 50 states, creating a single MOD 50 counter for binary would not be practical; while MOD counters are relatively simple, I need to output to a 7-segment display, which would be a mess to create. Therefore, I am going to use two BCD ripple counters instead. This greatly simplifies the amount of work needed. Here is the implementation of said BCD counters:



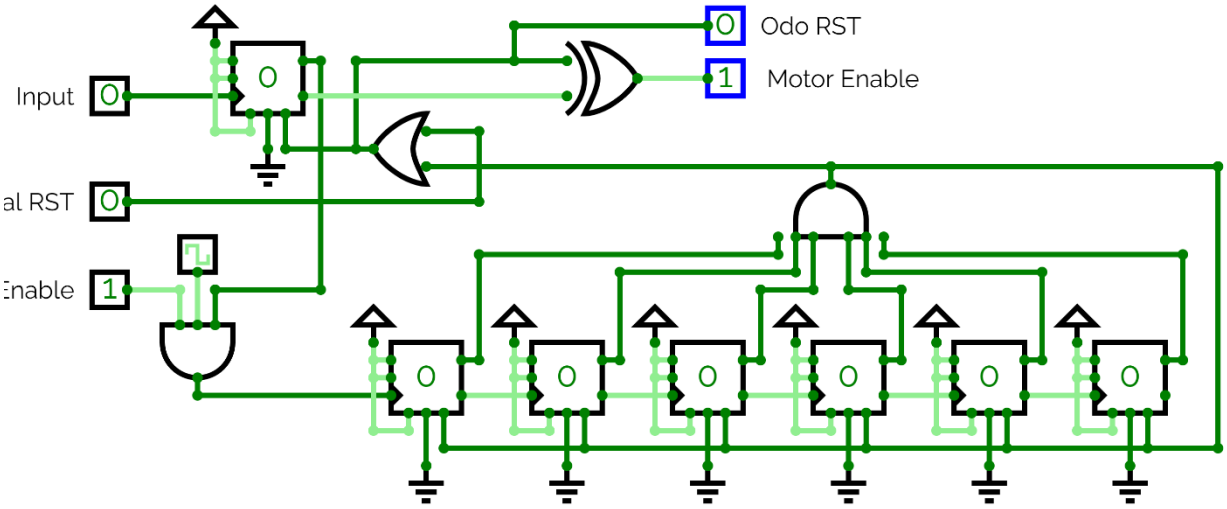
The counters are asynchronous and use JK FFs with both inputs tied to HI (toggle mode). The counters automatically reset on 1010, or 10 but also have a manual reset needed for the motor control. The stop signal is tied to the second BCD counter; it outputs HI when it reaches 5. The advantage of this is that this circuit is incredibly simple to implement and expand to higher orders of magnitude; the disadvantage is propagation delay, which if high enough, will cause errors in the accuracy of the odometer. Alone I do not see any issues, but I will have to test it fully integrated to see if any errors do pop up.

Task 4 – Subsystem D: Motor Control

This part of the system is meant to take the stop signal from the odometer and turn off the rover motors to let the batteries recharge. Incredibly simple, to the point that the circuit was just a JK FF set on toggle that turned off the gray code generator until toggle again for a while.



The stop signal triggers the FF – connected to the enable of the gray code generator with $\bar{Q}$ – and stops the system until reset is triggered. Dead simple and works fine. Later, on I expanded it a little:



The final motor control is still just the JK FF but with a few more additions. The main addition was a counter that automatically resets the FF. This is to simulate the rover recharging and means I do not have to manually reset the system every time. The clock can be disabled or enabled and the option to manually reset

is still there. The size is also arbitrary; I can expand the counter as much as I want, but I chose 6 bits as the limit because of size.

The more important addition is the Odometer reset I had forgot to implement. Previously the odometer reset automatically, but as the brief say that the odometer should still display the count until the whole system resets, I removed that and put the reset here. This time the motor is connected via an XOR gate – this prevents the Motor enable and the Odometer reset outputs being both on at the same time, which would turn on the Gray code generator but still lock the odometer and desyncing them.

Task 5 – Subsystem E: Odometer display

The final part is displaying the odometer readings in a format which can be read easily i.e., using seven segment displays. Thankfully because I already used BCD counters for my odometers, the entire process is streamlined as I only need to convert from BCD to seven segment values.

BCD				Seven Segment						
B_3	B_2	B_1	B_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1

Again, the following is the conversion table and the resulting K-Map plus MUX simplification if suitable. The shading in the MUX simplification have meanings as well:

- Green means one of the inputs is being assigned to a MUX input
- Yellow and a darker yellow mean that the highlighted parts are the same
- Red means the variable is being discarded/left unused

B_3	B_2	B_1	B_0	a		$\overline{B_1}.\overline{B_0}$	$\overline{B_1}.B_0$	$B_1.B_0$	$B_1.\overline{B_0}$
0	0	0	0	1	$\overline{B_3}.\overline{B_2}$	1	0	1	1
0	0	0	1	0	$\overline{B_3}.B_2$	0	1	1	1
0	0	1	0	1	$B_3.B_2$	X (1)	X (1)	X (1)	X (1)
0	0	1	1	1	$B_3.\overline{B_2}$	1	1	X (1)	X (1)
0	1	0	0	0	$a = B_3 + B_1 + B_2.B_0 + \overline{B_3}.\overline{B_2}.\overline{B_0}$				
0	1	0	1	1					
0	1	1	0	1					
0	1	1	1	1					
1	0	0	0	1					
1	0	0	1	1					

MUX approach

B_3	B_2	B_1	B_0	a	Input	B_3	B_2	B_1	B_0	a
0	0	0	0	1	1	0	0	0	0	1
0	0	0	1	0	B_3	0	0	0	1	B_3
0	0	1	0	1	1	0	0	1	0	1
0	0	1	1	1	1	0	0	1	1	1
0	1	0	0	0	0	0	1	0	0	0
0	1	0	1	1	1	0	1	0	1	1
0	1	1	0	1	1	0	1	1	0	1
0	1	1	1	1	1	0	1	1	1	1
1	0	0	0	1		1	0	0	0	1
1	0	0	1	1		1	0	0	1	B_3

B_3	B_2	B_1	B_0	b		$\overline{B_1}.\overline{B_0}$	$\overline{B_1}.B_0$	$B_1.B_0$	$B_1.\overline{B_0}$

0	0	0	0	1	$\overline{B_3}.\overline{B_2}$	1	1	1	1
0	0	0	1	1	$\overline{B_3}.B_2$	1	0	1	0
0	0	1	0	1	$B_3.B_2$	X (1)	X (0)	X (1)	X (0)
0	0	1	1	1	$B_3.\overline{B_2}$	1	1	X (1)	X (1)
0	1	0	0	1	$b = \overline{B_2} + \overline{B_1}.\overline{B_0} + B_1.B_0$				
0	1	0	1	0					
0	1	1	0	0					
0	1	1	1	1					
1	0	0	0	1					
1	0	0	1	1					

MUX approach

B_3	B_2	B_1	B_0	b	Input	B_3	B_2	B_1	B_0	b
0	0	0	0	1	1	0	0	0	0	1
0	0	0	1	1	1	0	0	0	1	1
0	0	1	0	1	1	0	0	1	0	1
0	0	1	1	1	1	0	0	1	1	1
0	1	0	0	1	1	0	1	0	0	1
0	1	0	1	0	0	0	1	0	1	0
0	1	1	0	0	0	0	1	1	0	0
0	1	1	1	1	1	0	1	1	1	1
1	0	0	0	1		1	0	0	0	1
1	0	0	1	1		1	0	0	1	1

B_3	B_2	B_1	B_0	c		$\overline{B_1}.\overline{B_0}$	$\overline{B_1}.B_0$	$B_1.B_0$	$B_1.\overline{B_0}$
0	0	0	0	1	$\overline{B_3}.\overline{B_2}$	1	1	1	0
0	0	0	1	1	$\overline{B_3}.B_2$	1	1	1	1
0	0	1	0	0	$B_3.B_2$	X (1)	X (1)	X (1)	X (1)
0	0	1	1	1	$B_3.\overline{B_2}$	1	1	X (1)	X (0)
0	1	0	0	1	$c = B_0 + \overline{B_1} + B_2$				
0	1	0	1	1					
0	1	1	0	1					
0	1	1	1	1					
1	0	0	0	1					
1	0	0	1	1					

No MUX needed

B_3	B_2	B_1	B_0	d		$\overline{B_1}.\overline{B_0}$	$\overline{B_1}.B_0$	$B_1.B_0$	$B_1.\overline{B_0}$
0	0	0	0	1	$\overline{B_3}.\overline{B_2}$	1	0	1	1
0	0	0	1	0	$\overline{B_3}.B_2$	0	1	0	1
0	0	1	0	1	$B_3.B_2$	X (0)	X (1)	X (0)	X (1)
0	0	1	1	1	$B_3.\overline{B_2}$	1	0	X (1)	X (1)
0	1	0	0	0	$d = \overline{B_2}.\overline{B_1}.\overline{B_0} + B_2.\overline{B_1}.B_0 + \overline{B_2}.B_1 + B_1.\overline{B_0}$				
0	1	0	1	1					
0	1	1	0	1					
0	1	1	1	0					
1	0	0	0	1					
1	0	0	1	0					

MUX approach

B_3	B_2	B_1	B_0	d	Input	B_3	B_2	B_1	B_0	d
0	0	0	0	1	1	0	0	0	0	1
0	0	0	1	0	0	0	0	0	1	0
0	0	1	0	1	1	0	0	1	0	1

0	0	1	1	1	1	0	0	1	1	1
0	1	0	0	0	0	0	1	0	0	0
0	1	0	1	1	1	0	1	0	1	1
0	1	1	0	1	1	0	1	1	0	1
0	1	1	1	0	0	0	1	1	1	0
1	0	0	0	1		1	0	0	0	1
1	0	0	1	0		1	0	0	1	0

B_3	B_2	B_1	B_0	e		$\overline{B_1}.\overline{B_0}$	$\overline{B_1}.B_0$	$B_1.B_0$	$B_1.\overline{B_0}$
0	0	0	0	1	$\overline{B_3}.\overline{B_2}$	1	0	0	1
0	0	0	1	0	$\overline{B_3}.B_2$	0	0	0	1
0	0	1	0	1	$B_3.B_2$	X (0)	X (0)	X (0)	X (1)
0	0	1	1	0	$B_3.\overline{B_2}$	1	0	X (0)	X (1)
0	1	0	0	0	$e = B_1.\overline{B_0} + \overline{B_2}.\overline{B_1}.\overline{B_0}$				
0	1	0	1	0					
0	1	1	0	1					
0	1	1	1	0					
1	0	0	0	1					
1	0	0	1	0					

MUX approach

B_3	B_2	B_1	B_0	e	Input	B_3	B_2	B_1	B_0	e
0	0	0	0	1	1	0	0	0	0	1
0	0	0	1	0	0	0	0	0	1	0
0	0	1	0	1	1	0	0	1	0	1
0	0	1	1	0	0	0	0	1	1	0
0	1	0	0	0	0	0	1	0	0	0
0	1	0	1	0	0	0	1	0	1	0
0	1	1	0	1	1	0	1	1	0	1
0	1	1	1	0	0	0	1	1	1	0
1	0	0	0	1		1	0	0	0	1
1	0	0	1	0		1	0	0	1	0

B_3	B_2	B_1	B_0	f		$\overline{B_1}.\overline{B_0}$	$\overline{B_1}.B_0$	$B_1.B_0$	$B_1.\overline{B_0}$
0	0	0	0	1	$\overline{B_3}.\overline{B_2}$	1	0	0	0
0	0	0	1	0	$\overline{B_3}.B_2$	1	1	0	1
0	0	1	0	0	$B_3.B_2$	X (1)	X (1)	X (1)	X (1)
0	0	1	1	0	$B_3.\overline{B_2}$	1	1	X (1)	X (1)
0	1	0	0	1	$f = B_3 + \overline{B_1}.\overline{B_0} + B_2.\overline{B_1} + B_2.\overline{B_0}$				
0	1	0	1	1					
0	1	1	0	1					
0	1	1	1	0					
1	0	0	0	1					
1	0	0	1	1					

MUX approach

B_3	B_2	B_1	B_0	f	Input	B_3	B_2	B_1	B_0	f
0	0	0	0	1	1	0	0	0	0	1
0	0	0	1	0	B_3	0	0	0	1	B_3
0	0	1	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	1	1	0
0	1	0	0	1	1	0	1	0	0	1
0	1	0	1	1	1	0	1	0	1	1

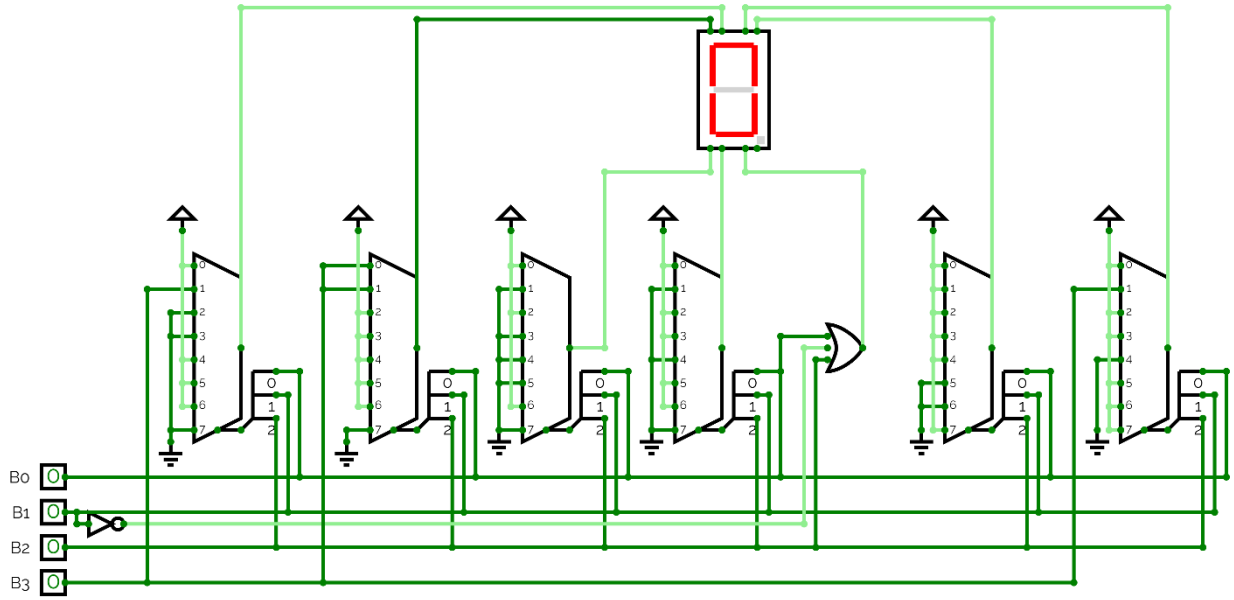
0	1	1	0	1	1	0	1	1	0	1	
0	1	1	1	0	0	0	1	1	1	0	
1	0	0	0	1		1	0	0	0	1	
1	0	0	1	1		1	0	0	1	B_3	

B_3	B_2	B_1	B_0	g		$\overline{B_1}.\overline{B_0}$	$\overline{B_1}.B_0$	$B_1.B_0$	$B_1.\overline{B_0}$
0	0	0	0	0	$\overline{B_3}.\overline{B_2}$	0	0	1	1
0	0	0	1	0	$\overline{B_3}.B_2$	1	1	0	1
0	0	1	0	1	$B_3.B_2$	X (1)	X (1)	X (1)	X (1)
0	0	1	1	1	$B_3.\overline{B_2}$	1	1	X (1	X (1)
0	1	0	0	1	$g = B_3 + B_2.\overline{B_1} + B_1.\overline{B_2} + B_1.\overline{B_0}$				
0	1	0	1	1					
0	1	1	0	1					
0	1	1	1	0					
1	0	0	0	1					
1	0	0	1	1					

MUX approach

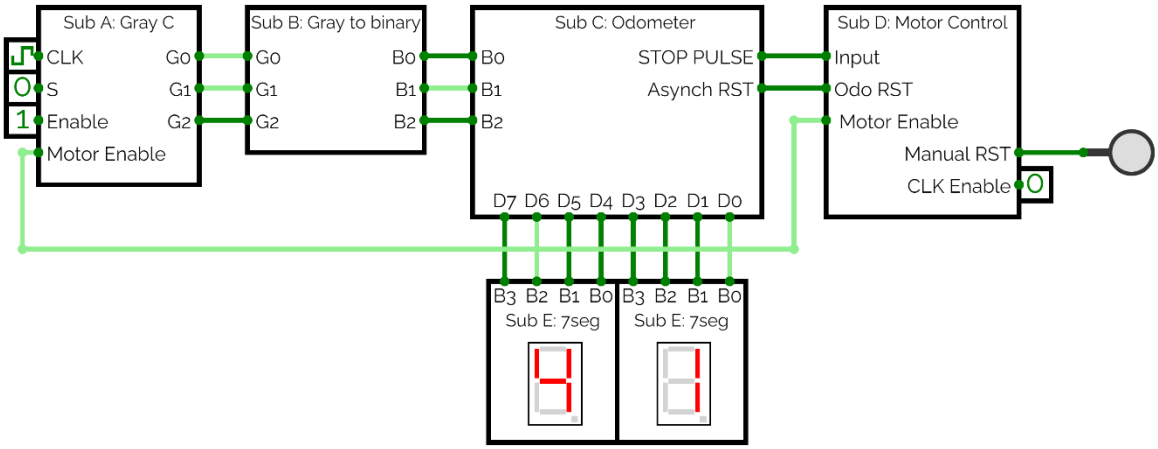
B_3	B_2	B_1	B_0	g	Input	B_3	B_2	B_1	B_0	g
0	0	0	0	0	B_3	0	0	0	0	B_3
0	0	0	1	0	B_3	0	0	0	1	B_3
0	0	1	0	1	1	0	0	1	0	1
0	0	1	1	1	1	0	0	1	1	1
0	1	0	0	1	1	0	1	0	0	1
0	1	0	1	1	1	0	1	0	1	1
0	1	1	0	1	1	0	1	1	0	1
0	1	1	1	0	0	0	1	1	1	0
1	0	0	0	1		1	0	0	0	B_3
1	0	0	1	1		1	0	0	1	B_3

The result of which is this:



Very large, but not too bad as far as wiring goes, especially since an output was simplified to just an OR. You may notice this only has four inputs; this is because the odometer is already split into two BCD counters, meaning I can just copy this circuit to display both values.

Task 6: System integration and testing



Here is the first iteration of the entire system integrated. Some work was done in the layouts of the subcircuits to minimise the amount of wire lines needed, though there is still one obnoxiously long wire needed for the motor enable. Every single component was tested and its operation verified as soon as they were completed, meaning there was very little trouble shooting I had to do afterwards. The Gray code generator works as expected; the decoder translates it into binary; the odometer successfully counts to 50 and outputs the stop signal correctly; the motor control immediately stops the motors and automatically resets the system when enabled; the seven segments display the correct numbers. In other words, a success.

Improvements made afterwards

Subsystem C: Odometer

As with any first iteration of anything, there is room for improvements. While trialling the whole system, the display stutters while counting – speeding and slowing at a consistent rate. In other words, the propagation delay is high enough to be noticed. While I believe the delay is not substantial enough to cause major errors (the motors stop on time), delay in a circuit meant for controlling a system should not be present and the odometer should be as accurate to the current state of the machine. So, I am replacing it with a synchronous version using D FFs, like the Gray code generator. The following transition diagram and MUX simplification is the design process for a single BCD counter:

Present State				Next State				States needed			
B_3	B_2	B_1	B_0	B_3	B_2	B_1	B_0	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	1	0	0	1	1
0	0	1	1	0	1	0	0	0	1	0	0
0	1	0	0	0	1	0	1	0	1	0	1
0	1	0	1	0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	1	0	1	1	1
0	1	1	1	1	0	0	0	1	0	0	0
1	0	0	0	1	0	0	1	1	0	0	1
1	0	0	1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0

1	1	1	1		0	0	0	0		0	0	0	0

NOTE: the shading is the same as the previous MUX simplification, with the added blue shading used for the second step.

B_3	B_2	B_1	B_0	D_3		B_2	B_1	B_0	IN
0	0	0	0	0		0	0	0	B_3
0	0	0	1	0		0	0	1	0
0	0	1	0	0		0	1	0	0
0	0	1	1	0		0	1	1	0
0	1	0	0	0		1	0	0	0
0	1	0	1	0		1	0	1	0
0	1	1	0	0		1	1	0	0
0	1	1	1	1		1	1	1	$\overline{B_3}$
1	0	0	0	1					
1	0	0	1	0		B_1	B_0	IN	
1	0	1	0	0		0	0	$B_3.\overline{B_2}$	
1	0	1	1	0		0	1	0	
1	1	0	0	0		1	0	0	
1	1	0	1	0		1	1	$\overline{B_3}.B_2$	
1	1	1	0	0					
1	1	1	1	0					

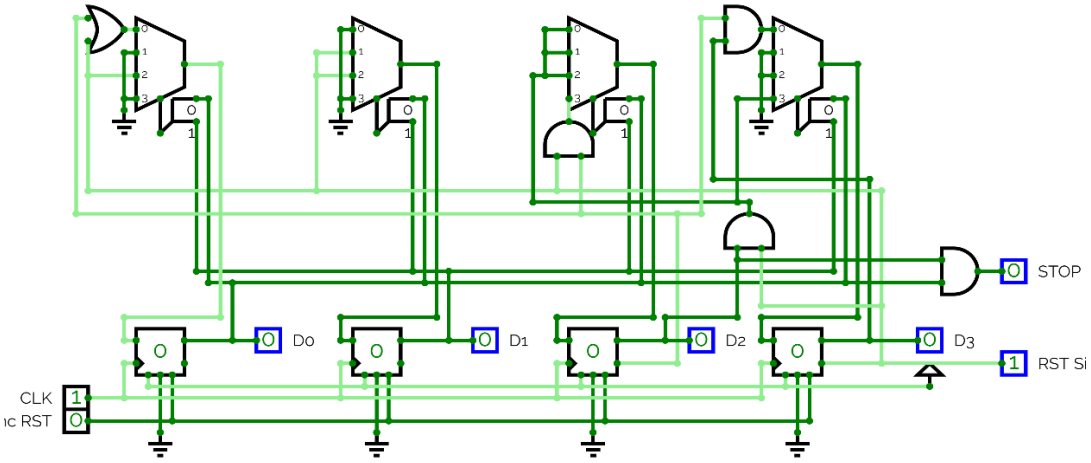
B_3	B_2	B_1	B_0	D_2		B_2	B_1	B_0	IN
0	0	0	0	0		0	0	0	0
0	0	0	1	0		0	0	1	0
0	0	1	0	0		0	1	0	0
0	0	1	1	1		0	1	1	$\overline{B_3}$
0	1	0	0	1		1	0	0	$\overline{B_3}$
0	1	0	1	1		1	0	1	$\overline{B_3}$
0	1	1	0	1		1	1	0	$\overline{B_3}$
0	1	1	1	0		1	1	1	0
1	0	0	0	0					
1	0	0	1	0		B_1	B_0	MUX IN	
1	0	1	0	0		0	0	$\overline{B_3}.B_2$	
1	0	1	1	0		0	1	$\overline{B_3}.B_2$	
1	1	0	0	0		1	0	$\overline{B_3}.B_2$	
1	1	0	1	0		1	1	$\overline{B_3}.\overline{B_2}$	
1	1	1	0	0					
1	1	1	1	0					

B_3	B_2	B_1	B_0	D_1		B_2	B_1	B_0	IN
0	0	0	0	0		0	0	0	0
0	0	0	1	1		0	0	1	$\overline{B_3}$
0	0	1	0	1		0	1	0	$\overline{B_3}$
0	0	1	1	0		0	1	1	0
0	1	0	0	0		1	0	0	0
0	1	0	1	1		1	0	1	$\overline{B_3}$
0	1	1	0	1		1	1	0	$\overline{B_3}$
0	1	1	1	0		1	1	1	0
1	0	0	0	0					
1	0	0	1	0		B_1	B_0	MUX IN	
1	0	1	0	0		0	0	0	
1	0	1	1	0		0	1	$\overline{B_3}$	

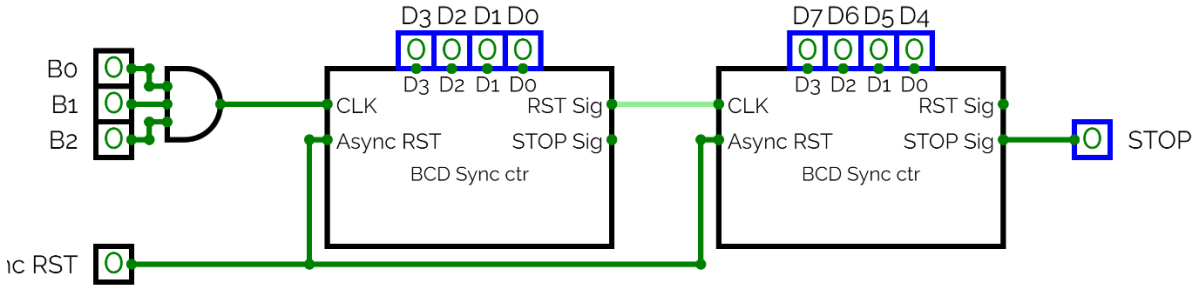
1	1	0	0	0		1	0	$\overline{B_3}$	
1	1	0	1	0		1	1	0	
1	1	1	0	0					
1	1	1	1	0					

B_3	B_2	B_1	B_0	D_0		B_2	B_1	B_0	IN
0	0	0	0	1		0	0	0	1
0	0	0	1	0		0	0	1	0
0	0	1	0	1		0	1	0	$\overline{B_3}$
0	0	1	1	0		0	1	1	0
0	1	0	0	1		1	0	0	$\overline{B_3}$
0	1	0	1	0		1	0	1	0
0	1	1	0	1		1	1	0	$\overline{B_3}$
0	1	1	1	0		1	1	1	0
1	0	0	0	1					
1	0	0	1	0		B_1	B_0	MUX IN	
1	0	1	0	0		0	0	$\overline{B_3} + \overline{B_2} = \overline{B_3}.\overline{B_2}$	
1	0	1	1	0		0	1	0	
1	1	0	0	0		1	0	$\overline{B_3}$	
1	1	0	1	0		1	1	0	
1	1	1	0	0					
1	1	1	1	0					

Here is the resulting Synchronous BCD counter:



And here is what it would look like implemented in a similar fashion the previous BCD counter:



The counter has similar inputs to the first odometer, but the difference is the RST signal is always on. Since the counter always resets after 9, there is no need for an AND gate tied to 1010 that resets the FF. But this means I have no easy way of sending a pulse to trigger the next counter. To solve this, the RST signal is tied to $\bar{Q}$ of the last FF. As the FFs are all rising edge CLK inputs, I can take advantage of the most significant bit only resetting at the 9. More clearly, the RST signal turns off when the counter reaches 8 but turns back on after the counter resets back to 0, triggering the next BCD counter. But how does this compare to the previous odometer – is it in anyway better?

Short answer is no; long answer is maybe. When ran with the previous odometer side by side, checking the seven segment displays, they both exhibited the same stuttering even though my assumption was that the stuttering was caused by propagation delay. The real cause may have been revealed when I slowed down the timing of the Circuitverse clock and watched the stuttering disappear. It turns out the cause was probably the program not being able to handle so many elements changing at the same time or my internet being unable to render it properly. Whatever the case, it was not due to propagation delay.

This was also my firs instance of trying to further simplify the MUXs down to 4:1. The process resulted in some improvements, but the resulting circuit is not as clean and I still needed some additional logic gates afterwards. Therefore, I will not simplify the other MUXs as they are fine as is. Though if given additional time, I may revisit it in case the results do make clean circuits

Unaddressed Issues/Potential improvements

While there was very little, if any, issues encountered after integration, there are still a few potential improvements to address:

- Adding a circuit or improving existing ones to deal with the issue of tracking the rover moving forwards and backwards: this was deliberately ignored for the moment to simplify the whole problem but is a limitation that still exists and will need to be address in the future. One way this can be done is by removing the Gray code decoder all together and counting bit changes until 8 changes have been made and incrementing the odometer
- Simplifying all the other MUXs: again, not needed but could yield good results if done properly
- Changing Subsystem B from gates to MUXs
- Rearranging the layouts to tidy up wiring.