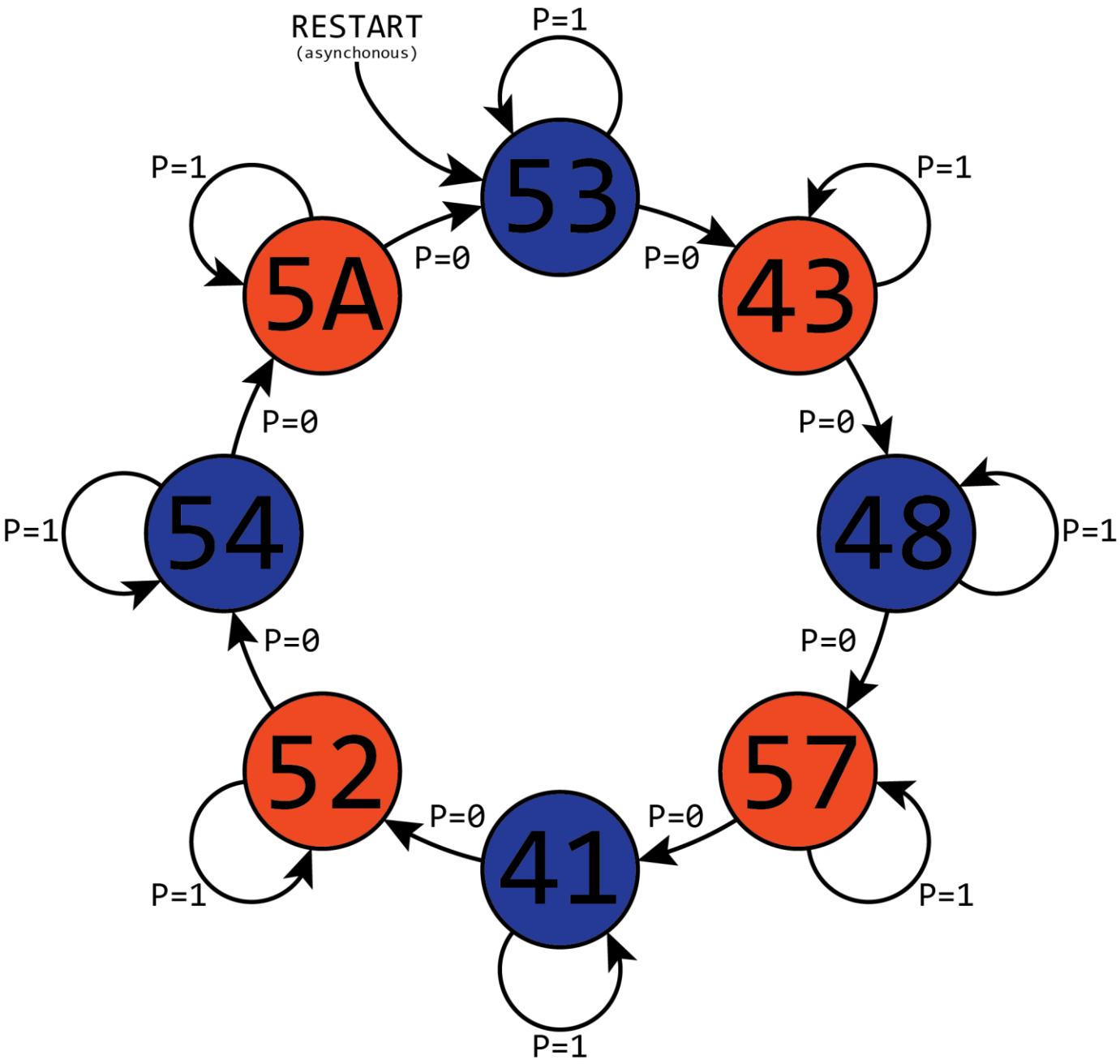
University of Florida
Electrical & Computer Engineering Dept.
EEL3701C: Digital Logic and Computer Systems

Created by Blake Shaffer
Date Created: 28/6/2019

OVER THE TOP COUNTER EXAMPLE



University of Florida
Electrical & Computer Engineering Dept.
EEL3701C: Digital Logic and Computer Systems

Created by Blake Shaffer
Date Created: 28/6/2019

Statement of the problem:

Referring to the diagram above, construct a counter that counts through the hexadecimal sequence as specified above. The counter must have an active low asynchronous RESET switch that returns it to the starting number specified. The counter must also have an active high pause switch P that, when true, holds the counter at whichever value it is currently at. All of the outputs used to display the numbers must be active high. You must use a JKFF for the MSB, a DFF for the LSB, and TFFs for any other bits you may need for the count sequence. Use an SPDT switch with the debounce circuit constructed in Quartus for the CLK signal and an SPST switch for the other two required inputs. Use an LED DIP package to display the hexadecimal values as outputs.

Breaking the problem down:

The counter designed in this example will count the sequence (in hexadecimal):

53, 43, 48, 57, 41, 52, 54, 5A (then repeats)…

This counter will have an *asynchronous* *active low* reset called "RESET" that will immediately return the count to the first number (53). This counter will also have an *active high* pause feature "P" that, when true, will pause the sequence at whichever count value it was last at. These two switches will be made by means of an accompanying switch circuit using an SPST switch. This counter will make use of a JKFF for the MSB (most significant bit), a DFF for the LSB (least significant bit), and a TFF for any other bits that may be needed. All of the output bits will be *active high* and will be constructed with an LED circuit using an LED DIP package.

Though this is a *counter* example, it does provide some knowledge on how to construct ASMs (arithmetic state machines). If you are unfamiliar with this, don't worry about it for now, it will be covered in future lectures. If you do, then this may help with your understanding of them as we will be defining "states" for each number.

Created by Blake Shaffer
Date Created: 28/6/2019

In the following pages, a solution to this counter will be laid out and an accompanying Quartus Archive File with this design and simulation will be provided.

This counter is MUCH BIGGER than anything you will ever have to do for labs, exams, or HW. The purpose of this demonstration is to show you that all counters can be constructed exactly the same way, no matter how big, which flip flops are chosen, or what the output values need to be. If you can design and construct this counter all by yourself, then you can tackle any counter that this class may throw at you.

Good luck on your labs, exams, and this counter! Without further ado, let’s get started.

University of Florida
Electrical & Computer Engineering Dept.
EEL3701C: Digital Logic and Computer Systems

Created by Blake Shaffer
Date Created: 28/6/2019

Part 1: Next State Truth Table

The FIRST step of ANY counter design is the construction of the Next State Truth Table. I CANNOT stress enough how important this table is for the counter. Once you have this made, then the whole rest of the counter will fall into place. Double and triple check your table after you have finished making it. If it ends up being wrong, then it's much easier to fix right when you make it than later on when you have to redo much more work.

Generally, you should put all your inputs on the left side and all your outputs on the right. One problem that we must first figure out is how many flips flops we'll need.

Given 2^n states (aka numbers you want to display), you will need n flip flops. What does this mean? Let's say you have 2 FFs, then the maximum number of states you can have with these 2 FFs is 2^2 or 4 states. This is the MAXIMUM number of states (numbers we want to display) that we can handle. Similarly, if you had 4 FFs, you could make a counter that has 2^4 = 16 states. But what if you have a number that's not a power of 2, say 5? Well, if you wanted to make a counter that has 5 different numbers (aka states), then you couldn't use just 2 FFs because that can only handle up to 4 states, so you must use 3 FFs since doing so can handle up to 8 = 2^3 states.

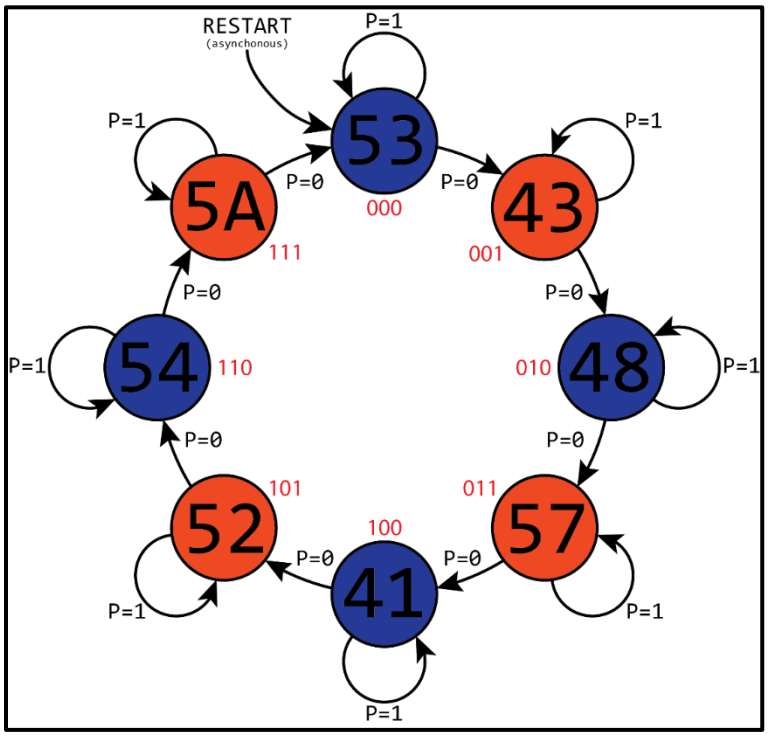
Back to the problem. Since we have 8 different numbers in our sequence, we have 8 different states we'll need. The smallest number of FFs we could use is thus 2^3 = 8, so 3. Therefore, we know that we will need just 3 FFs.

Since the problem specified the kind of FFs we must use, we will use a JKFF for the high bit, a TFF for the middle bit, and a DFF for the low

bit. Though also note that you *could* use ANY FF if you wanted, this is just what we will be doing in this problem.

All of our hex numbers have 2 hex characters in them. Since a single hex character contains 4 bits, then we will need 8 bits to represent the output sequence for each number. I will denote the output bits using Y's and since we have 8 bits, we need 8 lines for outputs. So, the Next State Truth Table must have columns for the output lines $Y_7 \ldots Y_0$.

When it's time to make the table, group the inputs together on the left side and count through the sequence of possible input values in COUNTING ORDER. This will make it MUCH easier for you later on to debug as well as construct the K-maps that will be required for the various outputs. On the diagram given above, it is helpful to write the state numbers next to the number you want to output for that state. This sounds confusing but it is very simple. Look at the copy of the counter below, now with the states labeled in red:



Created by Blake Shaffer
Date Created: 28/6/2019

So here I called the output number “53” state 000, the output number “43” state 001 and so on. You can technically use any number you want for the given states, but it is usually much easier to just use the numbers 000, 001, 010 and so on. Now that the states have been given values, we know how many and which FFs to use, as well as how many outputs we need, it’s time to make the table!

Next State Truth Table

P	Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	J_2K_2	T_1	D_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	1	0 X	0	1	0	1	0	1	0	0	1	1
0	0	0	1	0	1	0	0 X	1	0	0	1	0	0	0	0	1	1
0	0	1	0	0	1	1	0 X	0	1	0	1	0	0	1	0	0	0
0	0	1	1	1	0	0	1 X	1	0	0	1	0	1	0	1	1	1
0	1	0	0	1	0	1	X 0	0	1	0	1	0	0	0	0	0	1
0	1	0	1	1	1	0	X 0	1	0	0	1	0	1	0	0	1	0
0	1	1	0	1	1	1	X 0	0	1	0	1	0	1	0	1	0	0
0	1	1	1	0	0	0	X 1	1	0	0	1	0	1	1	0	1	0
1	0	0	0	0	0	0	0 X	0	0	0	1	0	1	0	0	1	1
1	0	0	1	0	0	1	0 X	0	1	0	1	0	0	0	0	1	1
1	0	1	0	0	1	0	0 X	0	0	0	1	0	0	1	0	0	0
1	0	1	1	0	1	1	0 X	0	1	0	1	0	1	0	1	1	1
1	1	0	0	1	0	0	X 0	0	0	0	1	0	0	0	0	0	1
1	1	0	1	1	0	1	X 0	0	1	0	1	0	1	0	0	1	0
1	1	1	0	1	1	0	X 0	0	0	0	1	0	1	0	1	0	0
1	1	1	1	1	1	1	X 0	0	1	0	1	0	1	1	0	1	0

The $Q_2Q_1Q_0$ represent the current state and $Q_2^+Q_1^+Q_0^+$ represent the next state. The way you construct these tables is to first make the 4 columns on the left with all the inputs and just make the counting order sequence, then go through each row and determine what the next state must be. For the first row, since P=0, we are not pausing and thus state 000 will always go to state 001. Then row 2, P=0, so we are

not pausing and state 001 will always go to state 010, and so on for the rest of the rows. When you get to the rows with P=1, the next state will simply be the current state since we want to *pause* the sequence. Therefore, you can just copy and paste the values over to the next state columns since they will be exactly the same numbers.

Now we need to fill in the values for the various FF columns. The only way to do this is to know the excitation tables for the various FFs which I will now show below:

DFF Excitation Table		
Q	Q+	D
0	0	0
0	1	1
1	0	0
1	1	1

TFF Excitation Table		
Q	Q+	T
0	0	0
0	1	1
1	0	1
1	1	0

JKFF Excitation Table			
Q	Q+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Using these tables, we can fill out the columns for all the different FFs by looking at each state and then filling in the corresponding FF column value based on what the Q bits are. DFFs are the easiest since we can literally just copy and paste the Q_0^+ to the D_0 column.

TFFs are also easy. The way to remember them is the fact that T stands for “toggle” meaning it toggles one state to the next. If you look at the excitation table, when the Q value changes from Q to Q^+, i.e. went from 0 to 1 or 1 to 0, then the T column has a 1, i.e. it toggled. If you go from Q to Q^+ and it stayed as 0 or stayed as 1, i.e. it DID NOT toggle, then the T column just has a 0.

The JKFF table is a bit trickier and may just be memorized. If you would like to gain more intuition on it, it is helpful to search up

how the table is derived. It is not required for this class to know *exactly* how the inner workings of the FFs work but doing so may give you a better understanding as to why these tables are what they are. I encourage you to do some self-study on the topic. Start by searching up some YouTube videos. (That's how I learned it!)

Now that all the FF columns are filled in, all that is left is to fill in the columns for the Y output bits.

Referring back to the diagram that we annotated earlier with the state numbers in red, we can now correspond the specific states to the hex number we want to output. So, for state 000, we want to output hex 53 which is 0101 0011 in binary. Therefore, in the columns labeled $Y_7..Y_0$, we put in 01010011. We do the same procedure for every state until the whole rest of the table is filled out.

NOTE: The Q^+'s are NEVER used as inputs to a FF or any of the equations we will soon make. The ONLY purpose of including the Q^+'s (the next state columns) in the table is to be able to fill in the columns for the FFs. Once again, this is their ONLY purpose. Once you have the columns for the various FFs filled out, you can pretty much ignore them, except if you need them for debugging of course.

Part 2: The Combinational Logic

Next, we need to make all the equations for the FF inputs and the Y outputs. I will assume you know how to construct K-maps so I will not go into detail about how you get simplified equations using them, but I will include the filled out K-maps that I used to solve this problem for you to reference.

The best way to save time in counter problems when making equations is to try to avoid having to use K-maps and only use them if absolutely

necessary. For example, if you look at the columns for Y_7, Y_6, and Y_5, you can see that they are all just 1's or all just 0's. Meaning that the equations for them are the easiest possible. So $Y_7=0$, $Y_6=1$, and $Y_5=0$.

You can also avoid K-maps with the J and K columns as well. By scanning the columns, you can see that there is only a single 1 in each, meaning that if we just try and find the SOP equation for each, we will then have the MSOP equation since it is just a single term! This is why JKFFs are so powerful, they have MANY don't cares (X's), that allow us to choose what we want them to be. Here we made them all 0's since we wanted the single "1" value to make our MSOP equations. If this is confusing, try to make the equations for J and K using SOP and you'll see why it's simple.

The rest are not super easy to see just from looking at the various input columns so K-maps will have to be constructed for those. A general rule of thumb for making your counter equations is to first check and see if it is easy to find the various equations required just by sight and reasoning so you can avoid having to make the K-maps if possible.

Below, I have provided the filled out K-maps as well as the various simplified equations for all the different FF inputs and Y outputs.

Note I did not include a K-map for the "T" bit for the TFF. This is because I was able to find it from the Next State Truth Table without having to make an equation. Can you figure out how? (Hint: Look at the equation for T and try to see how it relates to the columns of the table!)

University of Florida
Electrical & Computer Engineering Dept.
EEL3701C: Digital Logic and Computer Systems

Created by Blake Shaffer
Date Created: 28/6/2019

K-Maps for Next State Truth Table Equations:

D

PQ_2 \ Q_1Q_0	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	0	1	1	0
10	0	1	1	0

Y_4

PQ_2 \ Q_1Q_0	00	01	11	10
00	1	0	1	0
01	0	1	1	1
11	0	1	1	1
10	1	0	1	0

Y_3

PQ_2 \ Q_1Q_0	00	01	11	10
00	0	0	0	1
01	0	0	1	0
11	0	0	1	0
10	0	0	0	1

Y_2

PQ_2 \ Q_1Q_0	00	01	11	10
00	0	0	1	0
01	0	0	0	1
11	0	0	0	1
10	0	0	1	0

Y_1

PQ_2 \ Q_1Q_0	00	01	11	10
00	1	1	1	0
01	0	1	1	0
11	0	1	1	0
10	1	1	1	0

Y_0

PQ_2 \ Q_1Q_0	00	01	11	10
00	1	1	1	0
01	1	0	0	0
11	1	0	0	0
10	1	1	1	0

$$J = \bar{P}\overline{Q_2}Q_1Q_0 \qquad K = \bar{P}Q_2Q_1Q_0 \qquad T = Q_0\bar{P} \qquad D = PQ_0 + \bar{P}\,\overline{Q_0}$$

$$Y_7 = 0 \quad Y_6 = 1 \quad Y_5 = 0 \qquad Y_4 = \overline{Q_2}\,\overline{Q_1}\,\overline{Q_0} + Q_2Q_0 + Q_2Q_1 + Q_1Q_0$$

$$Y_3 = Q_2Q_1Q_0 + \overline{Q_2}Q_1\overline{Q_0} \qquad Y_2 = \overline{Q_2}Q_1Q_0 + Q_2Q_1\overline{Q_0}$$

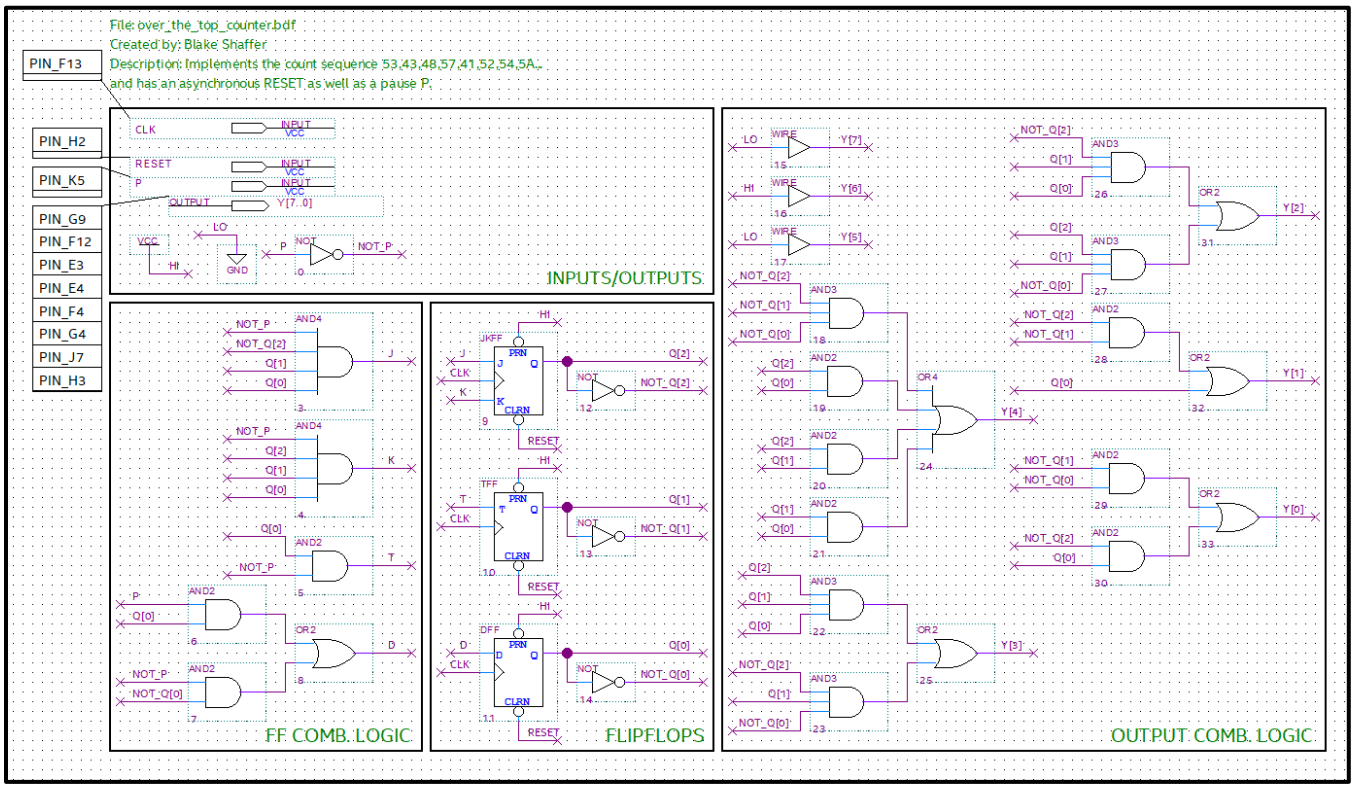
$$Y_1 = \overline{Q_2}\,\overline{Q_1} + Q_0 \qquad Y_0 = \overline{Q_1}\,\overline{Q_0} + \overline{Q_2}\,Q_0$$

Created by Blake Shaffer
Date Created: 28/6/2019

Part 3 Building it in Quartus:

Now that you have your NSTT done and all your equations made, all the hard parts are done. Pat yourself on the back. The rest is just building it in Quartus, simulating to make sure it works, and finally building in on the breadboard.

A screenshot of the Quartus .bdf I made for this design is shown below. Once again, you have access to the archive file of this project so you can mess around with it and simulate it yourself if you'd like.



This bdf has the required flip flops in the middle, the combinational logic for all the flip flops on the left side and the combinational logic for all the outputs on the right.

Note that I haven't included an SR-Latch. This should be included for the debounce switch since the problem specified to construct the

debounce circuit inside of Quartus rather than using an external chip. So if you do this problem, you will need to make one yourself.

Nothing too complicated is happening in this bdf (compared to the rest of what we've done at least!). You simply just have to implement all the equations you made, create you CLK signal input with your debounce circuit and then create the appropriate outputs and assign pin numbers.

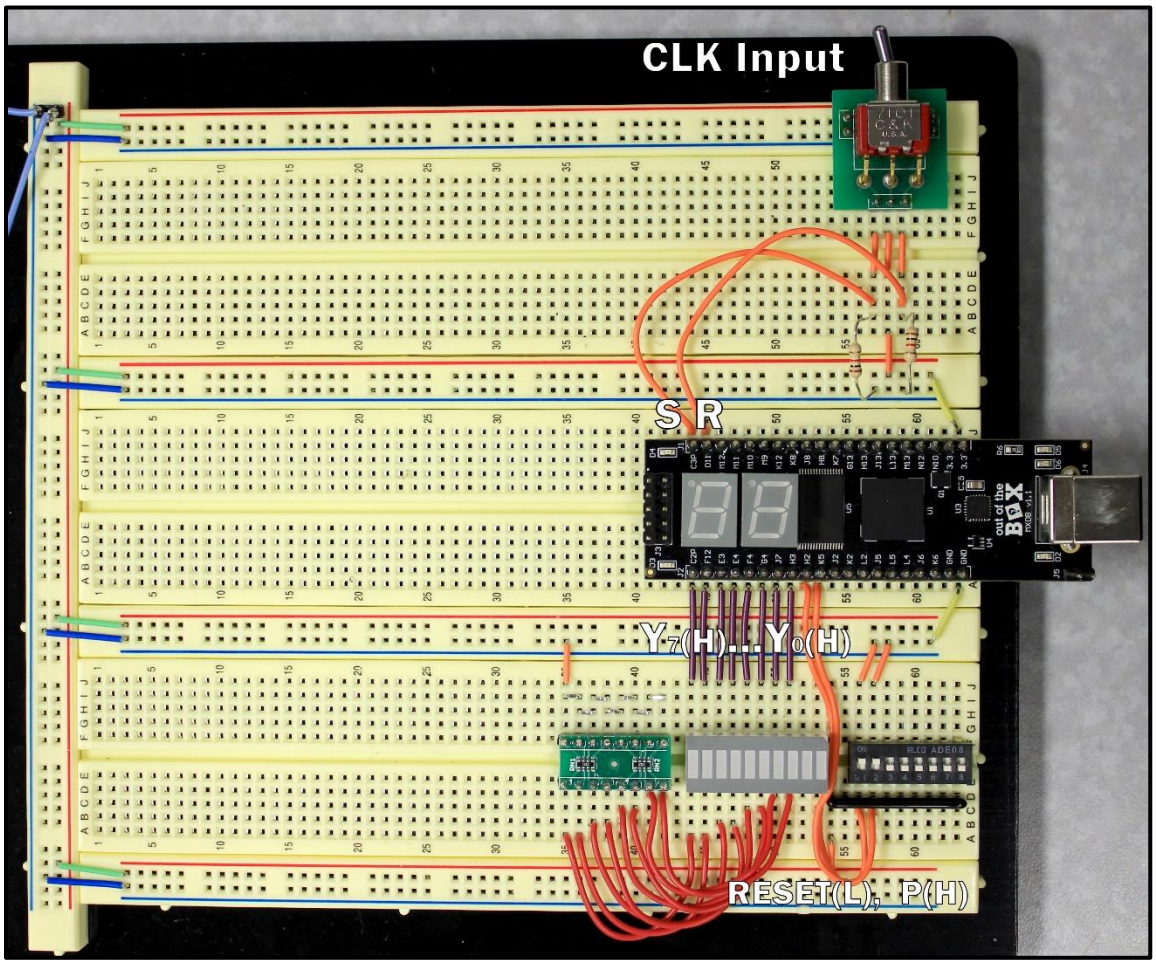
If you would like to see the simulation output of this circuit, I encourage you to download the .qar file and simulate it yourself. A *Waveform.vwf* file is already included for you to load. See if you notice anything special about the output of the count sequence to see why I chose these specific numbers.

Created by Blake Shaffer
Date Created: 28/6/2019

Part 4: Constructing it on the Breadboard

All that's left to do is program the file onto the PLD and construct the appropriate Switch and LED circuits to display the numbers. Since the Y outputs are active high, we must make active high LED circuits.

Included below is a picture of the finished build. The 8 leftmost LEDs are the ones used as outputs. The far left switch on the SPST is the active low switch used for RESET and the next switch over is used as the active high pause P.



After going through the sequence a few times and testing the various inputs, you can determine if the circuit works as specified or not. Once you get it working, congrats!

Created by Blake Shaffer
Date Created: 28/6/2019

Again, this is an OVER THE TOP counter design. In this class, you will never have to construct something like this for any exam or lab exercise. HOWEVER, in doing this whole thing by yourself from scratch, you will gain insight into building ANY counter circuit you may en*counter* in this class. Doing this will give you the skill to implement any of the 3 types of flipflops we use, work on using K-maps, creating and programming Quartus files, building switch and LED circuits, and (unless you're lucky) practice debugging which will inevitably make you learn even more.

I hope that this document was instructive on the process of designing and building a counter circuit and that you feel more confident in your ability to do something similar. Good luck with the rest of your time in Digital Logic and onward!