Shyavan Sridhar, 400388748

December 6, 2022

ENGPHYS 2E04 Design Project – Sequential Logic

Introduction

The final project for this course involved cycling through a nine-digit student number (400388748) on a seven-segment LED display. This was accomplished by applying the methods learned throughout the course for sequential logic design. The objective of this report is to develop three designs for the logic to obtain this end result, choose one to proceed with, and validate the design's functionality. This is accomplished through analytically using K-Mapping alongside logic gates and J-!K flip-flops to produce the desired logic by finding expressions for the inputs of the flip-flops, and simulating the logic using the components available through the application NI Multisim 14.3 to confirm the analytically designed logic works as intended. Moreover, a physical portion was also completed, building and presenting a single circuit design as a group to further demonstrate knowledge gained through the course, however that is not discussed as part of this report as it does not apply to the individual portion presented here.

Analytical Solution

To begin, the student number must first be converted into binary to identify all desired states within the stable loop that would cycle through the sequence of digits continuously. In addition, in order to use sequential logic design methods to cycle through the number, each state must also be unique in order to ensure the correct transitions are made at the correct times to form the number. As such, extra counter bits must be added alongside the digits of the number to represent any repeated digits. The number converted to binary with counter values for any repeated digits is presented in *Table 1*. Note that the "X"s in the counters represent "don't care" or irrelevant states as either the number is not repeated, or does not require that bit in the counter due to only repeating a set number of times, less than the number of unique states able to be represented by the bits used for the counter. Adding these cases also produces a more robust implementation, making it far more likely that the logic falls into the desired stable loop on its own, without requiring the pre-setting of a desired state. Furthermore, as the student number contains digits as high in value as eight, four bits must be used to represent the digits in the number alone, while the counter will require an additional two bits as the number contains digits that repeat as many as three times.

Decimal	Binary	Counter
4	0100	X0
0	0000	X0
0	0000	X1
3	0011	XX
8	1000	00
8	1000	X1
7	0111	XX
4	0100	X1
8	1000	10

Table 1 – Digits formatted in Decimal and Binary alongside Counter Values for Repetitions

Having converted the student number into binary representations with counter values added, the next step would be to create an excitation table for the J-!K flip-flops to be used in the logic design such that a state transition table can then be made and then K-Maps for each flip-flop's input derived from that. This table can be derived from the J-!K truth table found within the course topic notes and given below for reference as *Table 2* by identifying what inputs of J and !K produce a specific transition between two states.

J_n	$!K_n$	Q
0	1	q (hold)
0	0	0 (reset)
1	1	1 (set)
1	0	!q (toggle)

Table 2 – J-!K Flip-Flop Truth Table

Throughout, this report, a lowercase letter will denote the current state, while a capital letter will denote the next state proceeding it. In addition, "Q" will be used for any values making up the digit as part of the student number as well as denoting any generic value in both the truth and excitation tables for the J-!K flip-flop specifically, while "C" will be used for any values making up the counter portion. These will be numbered with a subscript according to their position when read from left to right in the sequence that makes up the digit, as will be their corresponding flip-flops. The excitation table developed as such is shown below as *Table 3*.

q	Q	J	!K
0	0	0	X
0	1	1	X
1	0	X	0
1	1	X	1
0	X	X	X
1	X	X	X
X	0	0	0
X	1	1	1

Table 3 – J-!K Flip-Flop Excitation Table

Now that both the excitation table and the binary representations of each state are ready, a state transition table for the entire stable loop can be made to represent exactly what the J and !K values must be for each flip flop in order to transition through each state in the loop successfully. This is shown as *Table 4* and is colour-coded such that each flip-flop has a unique colour in order to better identify any associations between values.

q_1	q_2	q_3	q_4	c_1	c_2	J_{Q1}	$!K_{Q1}$	J_{Q2}	$!K_{Q2}$	J_{Q3}	$!K_{Q3}$	J_{Q4}	$!K_{Q4}$	J_{C1}	$!K_{C1}$	J_{C2}	$!K_{C2}$	Q_1	Q_2	Q_3	Q_4	C_1	C_2
0	1	0	0	X	0	0	X	X	0	0	X	0	X	X	X	0	X	0	0	0	0	X	0
0	0	0	0	X	0	0	X	0	X	0	X	0	X	X	X	1	X	0	0	0	0	X	1
0	0	0	0	X	1	0	X	0	X	1	X	1	X	X	X	X	X	0	0	1	1	X	X
0	0	1	1	X	X	1	X	0	X	X	0	X	0	0	0	0	0	1	0	0	0	0	0
1	0	0	0	0	0	X	1	0	X	0	X	0	X	X	X	1	X	1	0	0	0	X	1
1	0	0	0	X	1	X	0	1	X	1	X	1	X	X	X	X	X	0	1	1	1	X	X
0	1	1	1	X	X	0	X	X	1	X	0	X	0	X	X	1	1	0	1	0	0	X	1
0	1	0	0	X	1	1	X	X	0	0	X	0	X	1	1	X	0	1	0	0	0	1	0
1	0	0	0	1	0	X	0	1	X	0	X	0	X	X	X	0	X	0	0	1	0	0	X

Table 4 – State Transition Table for Student Number Stable Loop

From studying this table carefully, some obvious simplifications arise that can make the K-Mapping process simpler. Firstly, Q_3 is equivalent to Q_4 at every state throughout the loop and as such, there is no need for the flip-flop corresponding to Q_4. The value produced by the flip-flop corresponding to Q_3 can simply be taken as the output for both bits and thus only K-Maps for Q_3 need be computed. Next, it is also evident that $!K_{Q3}$ can be simplified to always being tied low or set to 0 as it only contains "0"s and irrelevant cases across all states (as a result of the previous simplification this also applies to $!K_{Q4}$, however as the flip-flop for Q_4 is not required to be included, $!K_{Q4}$ is also subsumed into $!K_{Q3}$). As such, this value can be tied directly to ground without computing a K-Map for it. Lastly, J_{C1} is also equivalent to $!K_{C1}$ and as such, only one K-Map need be computed, with the logic for both values being the same, allowing both inputs to be tied together. In this fashion, only eight K-Maps are to be computed out of the initial twelve that would have been necessary for the six flip-flops making up this finite state machine (FSM). The above simplifications also allow the K-Maps that are to be computed to be composed from five variables rather than six as would be required otherwise, simplifying the computation process further. While this could possibly be further reduced by choosing the irrelevant cases such that they line up with other values and result in more simplifications, as stated earlier, including more of these cases builds a more robust implementation that will be far more likely to enter the desired loop directly without any unforeseen issues.

After applying these simplifications and removing the requisite columns, a modified state transition table can be used to begin computing the K-Maps to develop the required logic. This modified table is given as *Table 5* and retains the same colour-coding scheme as the one above for the headers to still denote the values corresponding to each other. However, it utilizes a different colour-coding scheme for the values that will be present in the computed K-Maps in order to better identify the positioning of each value within the K-Maps and make the computation process simpler and less likely to involve any mishaps.

q_1	q_2	q_3	c_1	c_2	J_{Q1}	$!K_{Q1}$	J_{Q2}	$!K_{Q2}$	J_{Q3}	J_{C1}	J_{C2}	$!K_{C2}$	Q_1	Q_2	Q_3	C_1	C_2
0	1	0	X	0	0	X	X	0	0	X	0	X	0	0	0	X	0
0	0	0	X	0	0	X	0	X	0	X	1	X	0	0	0	X	1
0	0	0	X	1	0	X	0	X	1	X	X	X	0	0	1	X	X
0	0	1	X	X	1	X	0	X	X	0	0	0	1	0	0	0	0
1	0	0	0	0	X	1	0	X	0	X	1	X	1	0	0	X	1
1	0	0	X	1	X	0	1	X	1	X	X	X	0	1	1	X	X
0	1	1	X	X	0	X	X	1	X	X	1	1	0	1	0	X	1
0	1	0	X	1	1	X	X	0	0	1	X	0	1	0	0	1	0
1	0	0	1	0	X	0	1	X	0	X	0	X	0	0	1	0	X

Table 5 – Modified State Transition Table for Student Number Stable Loop with Colour-Code for K-Maps

With the modified state transition table made above, the eight K-Maps can now be computed to find suitable Boolean logic expressions to achieve the desired outcome. These K-Maps are colour-coded with each entry matching the respective state row in *Table 5* to make it easier to understand. In addition, any white entries are all marked as irrelevant cases as they would represent values outside the scope of the state transition table and thus, outside the range of the student number looking to be implemented in this logic. Sum of products (SOP) expressions are identified by boxing portions of the K-Map containing "1"s and "X"s as large as possible with areas as powers of two and product of sums (POS) expressions are identified in the same manner albeit boxing portions containing "0"s and "X"s. For clarity, these are further identified as SOP boxes use either a blue, green, or purple outline while POS boxes use either a red or dark red outline, varying the colour to maximize contrast against the already coloured entries in the K-Maps and in order to separate different boxes constituting different terms in the final expressions with different colours. Furthermore, any boxes with a dashed outline are boxes that wrap around or continue to other parts of the K-Map such as around the edges or through the "stack". Boxes with solid outlines denote the opposite, or that they are not continued elsewhere and rather just remain within the confines of a single region bounded by a single outline. Expressions will be derived for three implementations to identify the most optimal one based on the number of logic gates and the number of ICs required to construct each one. These three implementations will be the SOP implementation using AND, OR, and NOT gates, the SOP implementation using NAND gates, and the POS implementation using AND, OR, and NOT gates. These will be further analysed to see if there is any advantage to taking different implementations for each K-Map. These eight K-Maps are presented below as *Tables 6 – 13* and are captioned with their corresponding values.

$C_1C_2\backslash Q_1Q_2Q_3$	000	001	011	010	100	101	111	110
00	0	1	0	0	X	X	X	X
01	0	1	0	1	X	X	X	X
11	0	1	0	1	X	X	X	X
10	0	1	0	0	X	X	X	X

Table 6 – K-Map for J_{Q1}

$C_1C_2\backslash Q_1Q_2Q_3$	000	001	011	010	100	101	111	110
00	X	X	X	X	1	X	X	X
01	X	X	X	X	0	X	X	X
11	X	X	X	X	0	X	X	X
10	X	X	X	X	0	X	X	X

Table 7 – K-Map for $!K_{Q1}$

$C_1C_2\backslash Q_1Q_2Q_3$	000	001	011	010	100	101	111	110
00	0	0	X	X	0	X	X	X
01	0	0	X	X	1	X	X	X
11	0	0	X	X	1	X	X	X
10	0	0	X	X	1	X	X	X

Table 8 – K-Map for J_{Q2}

$C_1C_2\backslash Q_1Q_2Q_3$	000	001	011	010	100	101	111	110
00	X	X	1	0	X	X	X	X
01	X	X	1	0	X	X	X	X
11	X	X	1	0	X	X	X	X
10	X	X	1	0	X	X	X	X

Table 9 – K-Map for $!K_{Q2}$

$C_1C_2\backslash Q_1Q_2Q_3$	000	001	011	010	100	101	111	110
00	0	X	X	0	0	X	X	X
01	1	X	X	0	1	X	X	X
11	1	X	X	0	1	X	X	X
10	0	X	X	0	0	X	X	X

Table 10 – K-Map for J_{Q3}

C_1C_2\$Q_1Q_2Q_3$	000	001	011	010	100	101	111	110
00	X	0	X	X	X	X	X	X
01	X	0	X	1	X	X	X	X
11	X	0	X	1	X	X	X	X
10	X	0	X	X	X	X	X	X

Table 11 – K-Map for J_{C1}

C_1C_2\$Q_1Q_2Q_3$	000	001	011	010	100	101	111	110
00	1	0	1	0	1	X	X	X
01	X	0	1	X	X	X	X	X
11	X	0	1	X	X	X	X	X
10	1	0	1	0	0	X	X	X

Table 12 – K-Map for J_{C2}

C_1C_2\$Q_1Q_2Q_3$	000	001	011	010	100	101	111	110
00	X	0	1	X	X	X	X	X
01	X	0	1	0	X	X	X	X
11	X	0	1	0	X	X	X	X
10	X	0	1	X	X	X	X	X

Table 13 – K-Map for !K_{C2}

The three implementations stated above can now be produced by generating their respective expressions from the K-Maps. Each implementation's Boolean expressions are presented below and simplified if possible. The minimum number of gates required is then calculated from the simplified expressions and listed alongside them. These are listed in *Equations 1 – 5*.

$$J_{Q1} = \overline{Q_2}Q_3 + Q_2\overline{Q_3}C_2$$
$$\overline{K_{Q1}} = \overline{C_1}\,\overline{C_2}$$
$$J_{Q2} = Q_1C_1 + Q_1C_2$$
$$\overline{K_{Q2}} = Q_3$$
$$J_{Q3} = \overline{Q_2}C_2$$
$$\overline{K_{Q3}} = 0$$
$$J_{C1} = Q_2$$
$$\overline{K_{C1}} = J_{C1}$$
$$J_{C2} = \overline{Q_1}\,\overline{Q_2}\,\overline{Q_3} + Q_2Q_3 + Q_1\overline{C_1}$$
$$\overline{K_{C2}} = Q_2Q_3$$

Equation 1 – Original SOP Implementation Boolean Expressions using AND, OR, and NOT Operations

$$J_{Q1} = \overline{Q_2}Q_3 + Q_2\overline{Q_3}C_2 - 3\text{ AND, 1 OR}$$
$$\overline{K_{Q1}} = \overline{C_1}\,\overline{C_2} - 1\text{ AND}$$
$$J_{Q2} = Q_1(C_1 + C_2) - 1\text{ AND, 1 OR}$$
$$\overline{K_{Q2}} = Q_3 - 0\text{ Gates}$$
$$J_{Q3} = \overline{Q_2}C_2 - 1\text{ AND}$$
$$\overline{K_{Q3}} = 0 - 0\text{ Gates}$$
$$J_{C1} = Q_2 - 0\text{ Gates}$$
$$\overline{K_{C1}} = J_{C1} - 0\text{ Gates}$$
$$J_{C2} = \overline{Q_1}\,\overline{Q_2}\,\overline{Q_3} + \overline{K_{C2}} + Q_1\overline{C_1} - 3\text{ AND, 2 OR}$$
$$\overline{K_{C2}} = Q_2Q_3 - 1\text{ AND}$$

Equation 2 – Simplified SOP Implementation Boolean Expressions using AND, OR, and NOT Operations

$$J_{Q1} = \overline{\overline{\overline{\overline{Q_2}}Q_3}\,\overline{Q_2\overline{\overline{Q_3}}C_2}} - 4\text{ NAND}$$
$$\overline{K_{Q1}} = \overline{\overline{\overline{\overline{C_1}}\,\overline{\overline{C_2}}}} - 2\text{ NAND}$$
$$J_{Q2} = \overline{\overline{Q_1C_1}\,\overline{Q_1C_2}} - 3\text{ NAND}$$
$$\overline{K_{Q2}} = Q_3 - 0\text{ Gates}$$
$$J_{Q3} = \overline{\overline{\overline{Q_2}C_2}} - 2\text{ NAND}$$
$$\overline{K_{Q3}} = 0 - 0\text{ Gates}$$
$$J_{C1} = Q_2 - 0\text{ Gates}$$
$$\overline{K_{C1}} = J_{C1} - 0\text{ Gates}$$
$$J_{C2} = \overline{\overline{\overline{\overline{\overline{Q_1}}\,\overline{\overline{Q_2}}\,\overline{Q_3}}}\,\overline{K_{C2}}\,\overline{Q_1\overline{\overline{C_1}}}} - 5\text{ NAND}$$
$$\overline{K_{C2}} = \overline{\overline{Q_2Q_3}} - 2\text{ NAND}$$

Equation 3 – Simplified SOP Implementation Boolean Expressions using NAND Operations

$$J_{Q1} = (Q_2 + Q_3)(Q_3 + C_2)(\overline{Q_2} + \overline{Q_3})$$
$$\overline{K_{Q1}} = (\overline{C_1})(\overline{C_2})$$
$$J_{Q2} = (Q_1)(C_1 + C_2)$$
$$\overline{K_{Q2}} = (Q_3)$$
$$J_{Q3} = (\overline{Q_2})(C_2)$$
$$\overline{K_{Q3}} = 0$$
$$J_{C1} = (Q_2)$$
$$\overline{K_{C1}} = J_{C1}$$
$$J_{C2} = (Q_2 + \overline{Q_3})(\overline{Q_2} + Q_3)(\overline{Q_1} + \overline{C_1})$$
$$\overline{K_{C2}} = (Q_2)(Q_3)$$

Equation 4 – Original POS Implementation Boolean Expressions using AND, OR, and NOT Operations

$$J_{Q1} = (Q_2 + Q_3)(Q_3 + C_2)(\overline{Q_2} + \overline{Q_3}) - 2 \text{ AND, } 3 \text{ OR}$$
$$\overline{K_{Q1}} = \overline{C_1}\,\overline{C_2} - 1 \text{ AND}$$
$$J_{Q2} = Q_1(C_1 + C_2) - 1 \text{ AND, } 1 \text{ OR}$$
$$\overline{K_{Q2}} = Q_3 - 0 \text{ Gates}$$
$$J_{Q3} = \overline{Q_2}C_2 - 1 \text{ AND}$$
$$\overline{K_{Q3}} = 0 - 0 \text{ Gates}$$
$$J_{C1} = Q_2 - 0 \text{ Gates}$$
$$\overline{K_{C1}} = J_{C1} - 0 \text{ Gates}$$
$$J_{C2} = (Q_2 + \overline{Q_3})(\overline{Q_2} + Q_3)(\overline{Q_1} + \overline{C_1}) - 2 \text{ AND, } 3 \text{ OR}$$
$$\overline{K_{C2}} = Q_2Q_3 - 1 \text{ AND}$$

Equation 5 – Simplified POS Implementation Boolean Expressions using AND, OR, and NOT Operations

From the above expressions it is evident that the SOP NAND implementation uses a total of eighteen gates, requiring five ICs. This is greater than either of the other two implementations and as such, would be less efficient to build in this case. Both the POS and SOP implementations utilizing AND, OR, and NOT gates will require a total of four ICs, the SOP implementation requiring ten AND and four OR gates, while the POS implementation requires eight AND and seven OR gates, for a total of fourteen and fifteen gates respectively. From here, it is clear that the SOP implementation uses the least gates even though it requires the same number of chips as the POS implementation. Further analyzing the expressions to see if there is any benefit in using different expressions for different values, it is shown that for all expressions, the simplified SOP implementation using AND, OR, and NOT gates either utilizes less or an equal amount of gates to either the SOP NAND or POS implementations. Although NAND gates are considered to be more efficient in terms of transistor usage and space utilization, due to the fact that all ICs used take up the same amount of space, all having four gates on a single package, the space utilization advantage in this case is nullified. In addition, the transistor usage benefit is not as valuable here as simplicity in the number of connections made in the circuit would be more advantageous since using NAND gates would require four more than the alternative, even if the increase in the number of gates was made up for by the reduction in transistor usage. Thus, the remainder of this report will proceed with the construction of the circuit using the SOP implementation with AND, OR, and NOT gates for the purposes of simplicity and efficiency it provides for this particular scenario.

Multisim Simulation

After generating the Boolean expressions to produce the desired logic analytically and choosing the implementation to proceed with, the logic was then implemented within Multisim. Shown below in *Figure 1* is the SOP implementation done using AND, OR, and NOT gates as described above analytically.

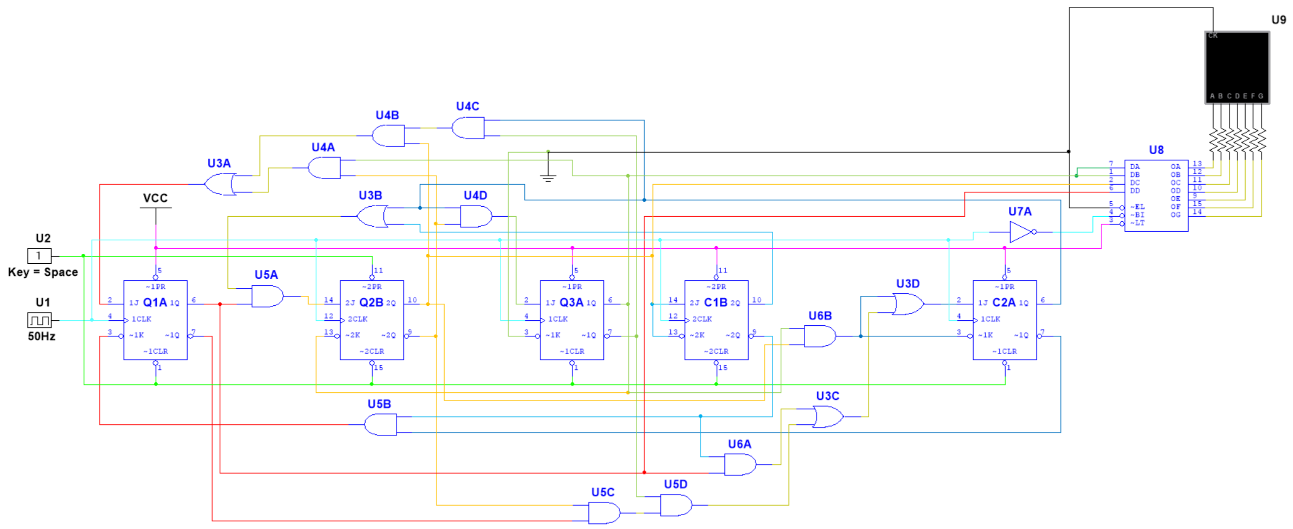

Figure 1 – SOP Circuit Implementation using AND, OR, and NOT Gates in Multisim

As shown in *Figure 1*, The circuit is implemented with five J-!K flip-flops ($Q_1 – Q_3$, C_1, and C_2) all labeled, ten AND gates ($U_4 – U_6$), four OR gates (U_3), a digital clock (U_1), and seven-segment LED display with common cathode (U_9). In addition, a NOT gate (U_7) was also used in connecting the clock signal to the !BL line of the Binary-Coded-Decimal-to-Seven-Segment-Display (BCD-7SD) decoder IC (U_8) as this was found to prevent flickering of the display by blanking it when the flip-flops were transitioning states rather than when they were displaying. The digital clock was set to 50 Hz and a duty cycle of 20% as this was found to flash the numbers at a reasonable pace with sufficient blanking between each digit and keep flickering of the display to a minimum. Seven 470 Ω resistors were also placed in series between each leg of the display and the outputs of the decoder IC in order to ensure the LEDs current draw was limited and no other components of the circuit were damaged. A net for ground is also in place as well as one for V_{CC} at 5 V which was found to be sufficient for all ICs in the circuit. An interactive digital constant (U_2) was also bound to the "Space" key and used to allow for easy pre-setting of the desired initial state or the first digit of the student number, ensuring that the circuit always is able to be set into the desired stable loop even if it starts elsewhere in another stable loop that was unforeseen. This was done by connecting this constant to the !PR pin of Q_2 as well as the !CLR pins of all the flip-flops and connecting the remaining flip-flops' !PR pins to V_{CC}. When the constant is set to 0 or pulled low, this then forces the circuit into the desired initial state for the number four by only setting Q_2

high. Then, the constant can be set back to 1 or high once more which will allow the circuit to continue into the remainder of the stable loop from that point.

The colour-coding scheme for the circuit was made to match that of *Table 4* used in the analytical for all possible connections with regards to the flip-flops, however, some extraneous connections required additional colours. These extra colours are as follows. Any intermediary connections between various logic gates or between the decoder IC and the display were coloured gold to signify that they were separate from the other previously established connections. In addition, all connections to V_{CC} were coloured magenta as the conventional colour of red was already used for one of the flip-flops and all connections to ground were coloured the conventional colour of black. Any connections to the clock were coloured cyan as well as the one intermediary connection between the NOT gate and the decoder IC to make this clearer, rather than colouring it gold similar to the other intermediary connections. Lastly, any connections to the interactive digital constant were made a neon green to differentiate them as part of the pre-setting function. For the cases where a single wire net would meet at two different places requiring different and conflicting colour codes, a main colour would be used for majority of the net, with the conflicting segment made to be the other colour up until it intersects the remainder of the net. One example of this is the net for ground where majority of the net is black, however one connection is a light green to signify that it is part of Q_3, this is the segment that is connected to $!K_{Q3}$ which was determined earlier to be always tied low. All ICs used and their part numbers are listed in *Table 14* for quick reference.

IC	Part Number
NOT Gate	74HC04N_6V
AND Gate	74HC08N_6V
OR Gate	74HC32N_6V
J-!K Flip-Flop	74HC109N_6V
BCD-7SD	4511BP_5V

Table 14 – List of ICs Used and Corresponding Part Numbers

Running the circuit simulation and connecting the logic analyzer to the outputs of all flip-flops and tying the output of Q_3 to another pin of the analyzer as Q_4 produces the full timing diagram for each bit and each state in the stable loop cycling through the student number. Shown below in *Figure 2* is the colour-coded and annotated timing diagram. The colour-coding scheme is the same as outlined previously for Multisim and each bit is also labeled as according to the scheme used throughout this report. In addition, each state is written in binary in white text below the analyzer's outputs and a white underline denotes one full loop of the student number before repeating and beginning again. The timing diagram starts at the first digit in the student number and ends on the next loop at the same state it started at, showcasing how it not only continues through the student number, but then also proceeds to cycle endlessly though it.

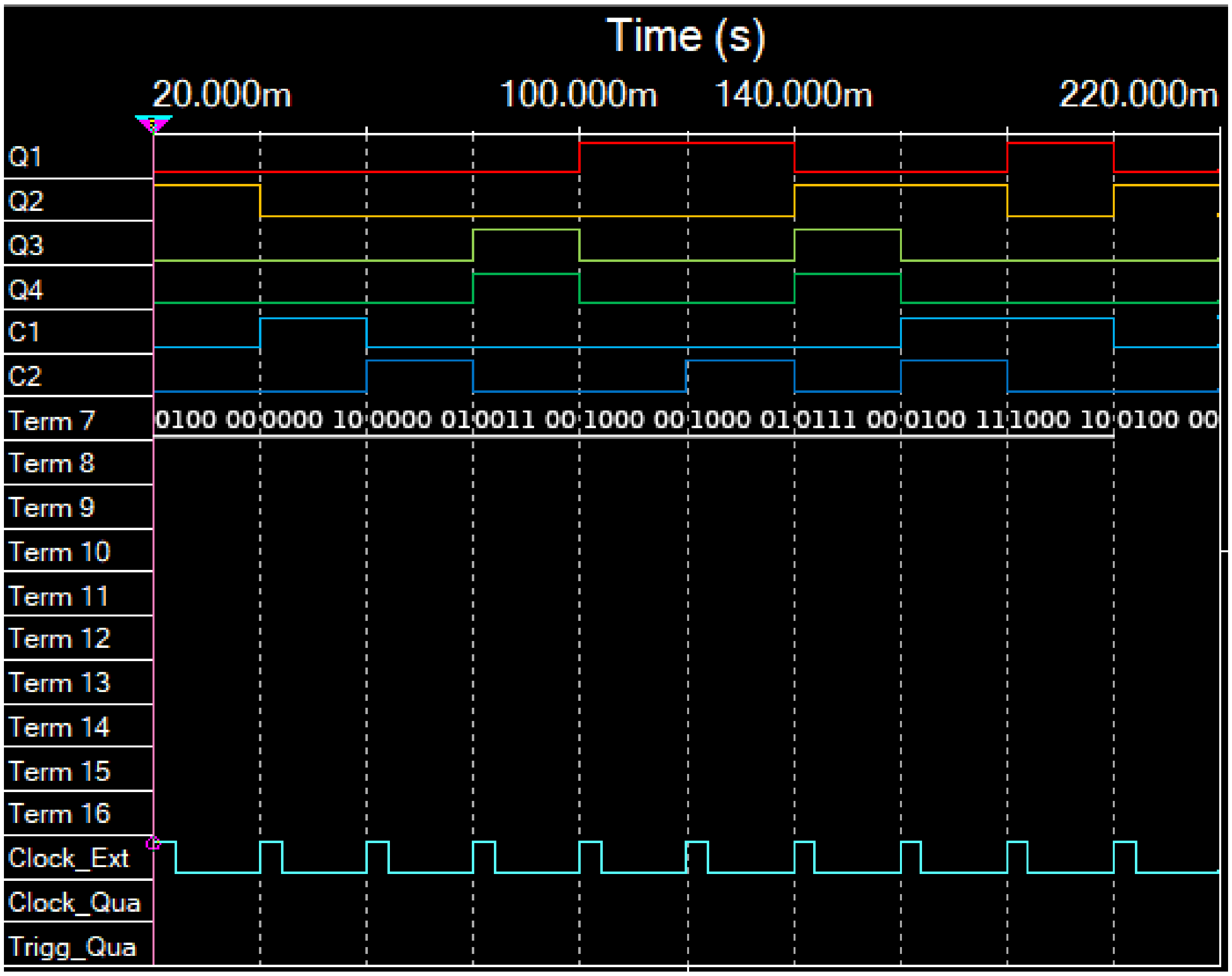

Figure 2 – Annotated Timing Diagram produced via the Multisim Logic Analyzer

Referring back to *Table 1* for the student number in decimal and how digit is represented in binary, it is evident that the timing diagram produced through Multisim lines up perfectly with what was desired. As such, it can be concluded that the logic derived analytically is correct in addition to the fact that the circuit constructed through Multisim was also done properly to produce the desired outcome. The FSM behaves correctly and when set to begin at a state within the stable loop constituting the student number (400388748), it continuously follows that sequence.

Shyavan Sridhar, 400388748

December 6, 2022

Discussion

As evidenced by the Multisim simulation, the logic derived analytically for this project is entirely valid and was done correctly. This final project and the course as a whole have taught many valuable lessons and have many useful applications for future courses or in the workspace, such as in the many digital electronic devices used everyday involving countless sequential logic operations preformed by a processor several million times per second. Furthermore, the skills refined through this such as K-Mapping can be used to construct a circuit capable of solving problems logically without the need for programming a microcontroller, allowing for the ease in construction of embedded systems that need to be portable or run on as little power as possible such as those found in smart fabrics, wearable sensors, and household electronics projects. Moreover debugging and FSM such as this and ensuring tidy circuit-diagrams with colour-coded wire nets are also important skills to learn when working on far more complex electronics projects that could get out of hand very quickly otherwise. Hence, there are numerous applications for what has been learned through this experience.