LAB 4: A MIDI Interface

ELEC2607 - LOE, Mon 2:35pm – 5:25 pm

Monday, April 6th, 2020

Youssef Ibrahim
Abdalrahman Shaheen

1.0 INTRODUCTION

The purpose of this lab is to design a Musical Instrument Design Interface circuit (MIDI) which can be used between a digital keyboard and a synthesizer or between a keyboard and a PC running a music composition program. The MIDI is a serial interface hence the bits come in one after the other. The circuit is assembled to have a 10-bit input and 8-bit parallel output. The keyboard sends 10-bits in series containing 8-bits called data type and 2-bits called control bits (START and STOP). Those 10-bits from the keyboard is then passed through a Serial-to-Parallel Receiver (SER2PAR) which converts the 10-bits in series to 8-bits in parallel (D7, D6, D5, D4, D3, D2, D1, D0) which is then sent to an instrument such as an mixer or synthesizer. The circuit was designed and simulated using a software called Xilinx ISE and ModelSim. The following report shows the Specifications in part 2.0, Design in part 3.0, Implementation and Testing in part 4.0, and the Conclusion in part 5.0 of the telephone switch circuit.

2.0 SPECIFICATIONS

The MIDI circuit was designed using Xilinx software hence there was not any restrictions on the number of gates and flipflops. ModelSim was used to generate waveforms which represented the MIDI behavior. The MIDI interface receives 10-bit series input from a keyboard and outputs 8-bits in parallel to an instrument. The MIDI interface consists of seven subcircuits:

1) Last4Samp: captures a sample on each active clock edge and then holds the last 4 samples.
2) FindStart: checks the samples for three zeroes out of four samples then it sends GotStart when this is found.
3) Majority: outputs the majority value of last 3 samples.
4) CountClear: checks that GotStart=1which means that it has found a valid START bit.
5) SampCount: starts at a count of zero then start counting halfway through the START bit when CLRSAMPCOUNT goes down.
6) BitCount: keeps track of bits in the MIDI interface.
7) Ser2Par: responds to each GRAB8 pulse then transfers the data bit values from MAJ to seven parallel data outputs D_0, D_1, …, D_7.

After designing the mentioned subcircuits, the where all then put together to create the MIDI interface.

3.0 DESIGN

The block diagram as follows in Figure 1. Each sub-circuit was built with the total design Goals put in mind. Which means that the circuit converts an input of 10-bit information from keyboard (SerIn/SigIn) to 8-bit data which is stable and recognizable by the synthesizer. The design of each of the sub-circuits is provided in Figure 3.0 below.

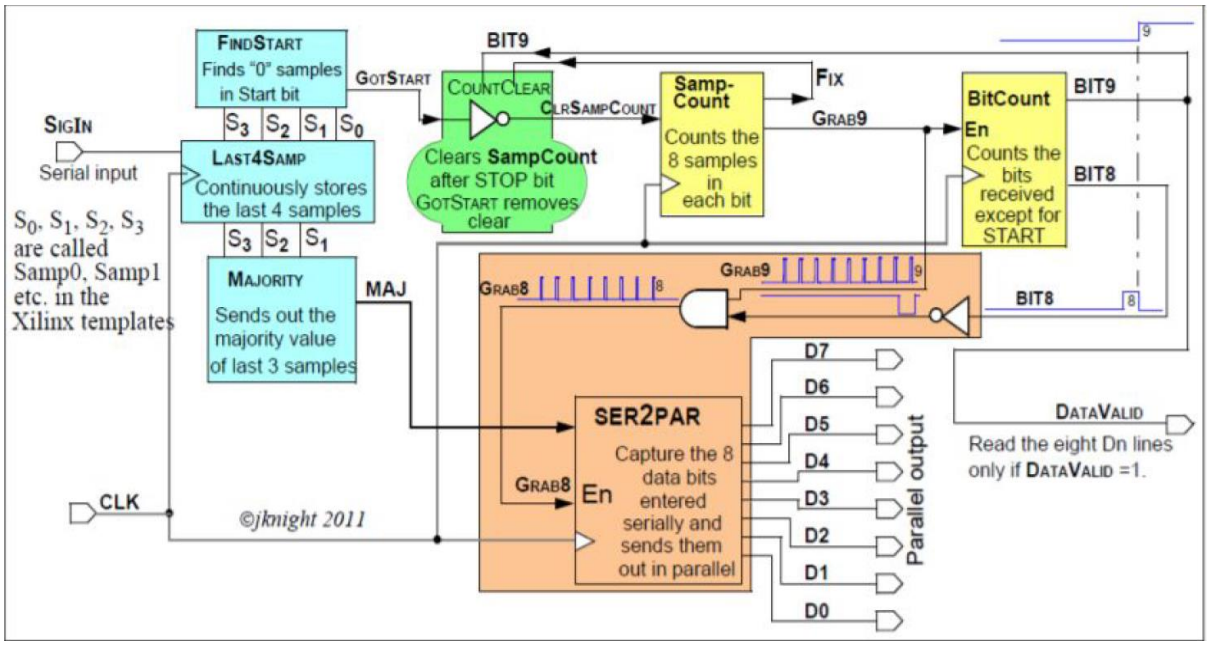

Figure 3.0: MIDI Interface Block Diagram [Lab 4 – p5, Modified]

3.1 Last4Samp:

The Last4Samp inputs are 10-bit SigIn input, a clock, and a reset. The Last4Samp's objection is to gather the input and a capture sample on every active CLK edge. After that, it holds the last 4 samples, and output them in S_0, S_1, S_2, and then S_3. This means it is done with a shift register design. A shift register job is to take a sample on every active CLK edge and store it on the leftmost D-flip flop. In the next cycle, the new sample is again stored to the left, and the sample before it is then pushed towards the right. By the end, the old sample will change to be on the rightmost storage. It is necessary to add inverters to our shift register, both

initially after SerIn input, also before each S(t) output. We do this part to accomplish avoiding errors when the reset signal initially clear all of the flip flops. On the FindStart, it usually conflicts by thinking it found the start bit and keep going on, which would give incorrect result. But we can avoid this by using an initial inverter. And with this inverter, shows both before and after each flip flop result and it will be the same. The schematic is displayed below in Figure 3.1.

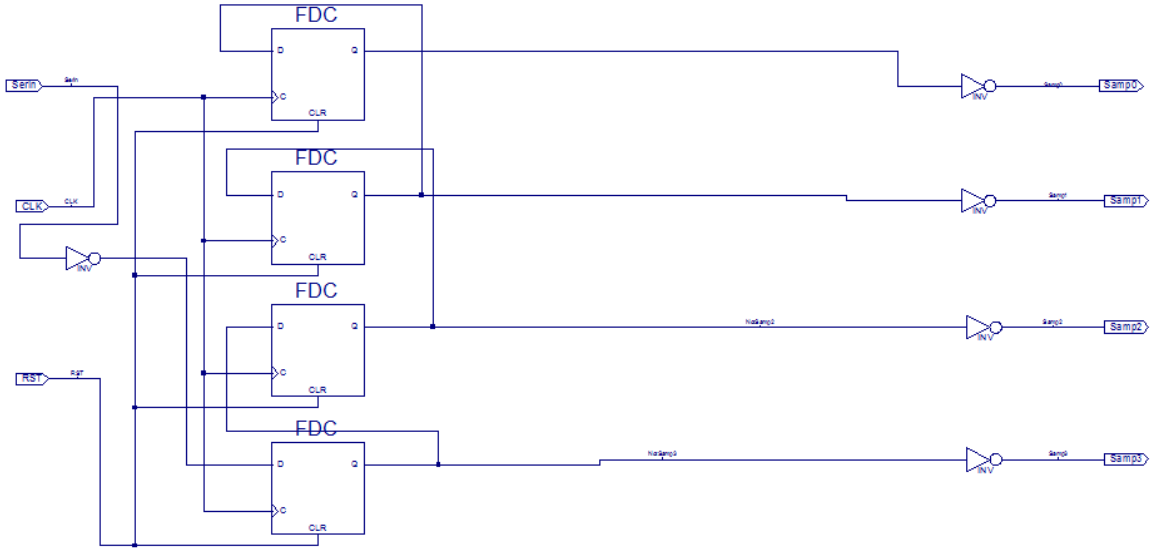

Figure 3.1: Last4Samp Schematic

3.2 Majority:

The objective behind Majority circuit is to take the majority value that can be either 1 or 0.

Which is taken from the last three samples.

Samp1	Samp2	Samp3	MAJ
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1

Table 3.0: Majority Truth Table

$MAJ = S_2S_3 + S_1S_3 + S_1S_2 + S_1S_2S_3$

$MAJ = S_2S_3 + S_1S_3 + S_1S_2(1+ S_3)$

$MAJ = S_2S_3 + S_1S_2 + S_1S_3$

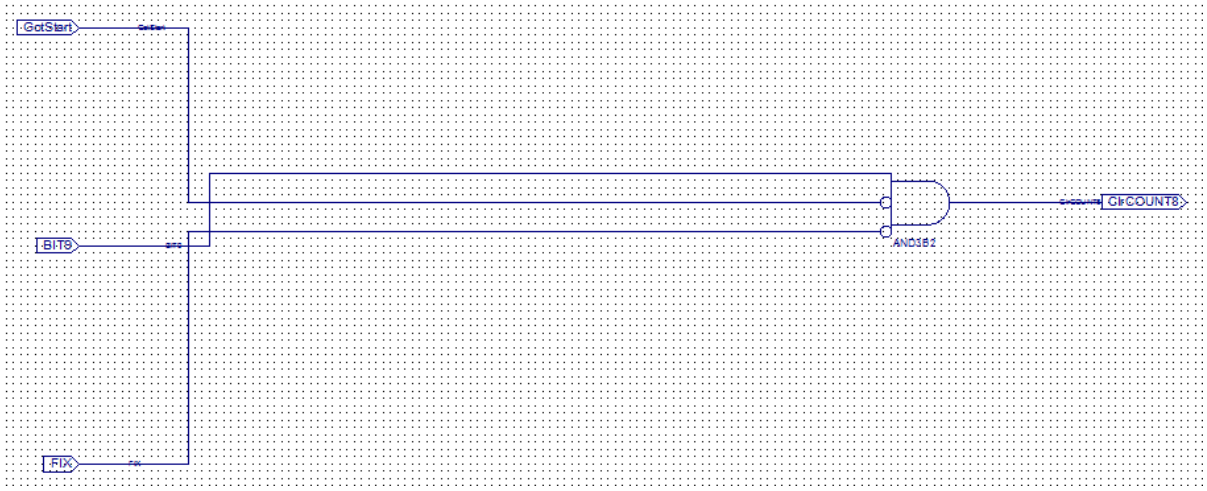

Figure 3.2: Majority Schematic

3.3 FindStart:

FindStart circuit used 4 samples (S_3, S_2, S_1), and S_0 from the Last4Samp to clarify

there is 3 zeros out of 4. If this is correct, the FindStart will output high GotStart. This can be

shown in the truth table below.

S3S2 / S1S0	00	01	11	10
00	1	1	0	1
01	1	0	0	0
11	0	0	0	0
10	1	0	0	0

Table 3.1: FindStart Truth Table

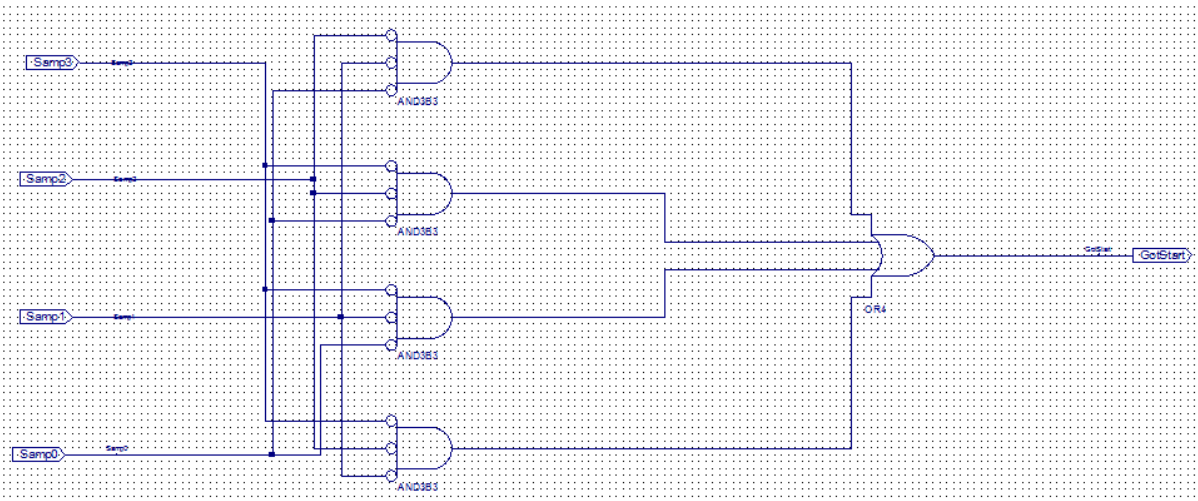

Figure 3.3: FindStart Schematic

3.4 The SampCount:

The purpose of the SampCount is to count from 0 to 7. Then we recount again, once every clock cycle. The inputs are ClrSampCount, CLK, and RST. When ClrSampCount is considered high, the count will be sent to 0 on the next clock edge. When the count reaches 7, a GRAB pulse will come from the output. The design of the SampCount was made using D flip flops. The design can be determined from the equations coming from the k map for the truth table. The table for SampCount is shown below in Table 3.2.

Count	State			Next State			D input		
	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
1	0	0	1	0	1	0	0	1	0
2	0	1	0	0	1	1	0	1	1
3	0	1	1	1	0	0	1	0	0
4	1	0	0	1	0	1	1	0	1
5	1	0	1	1	1	0	1	1	0
6	1	1	0	1	1	1	1	1	1
7	1	1	1	0	0	0	0	0	0

Table 3.2: SampCount Truth Table

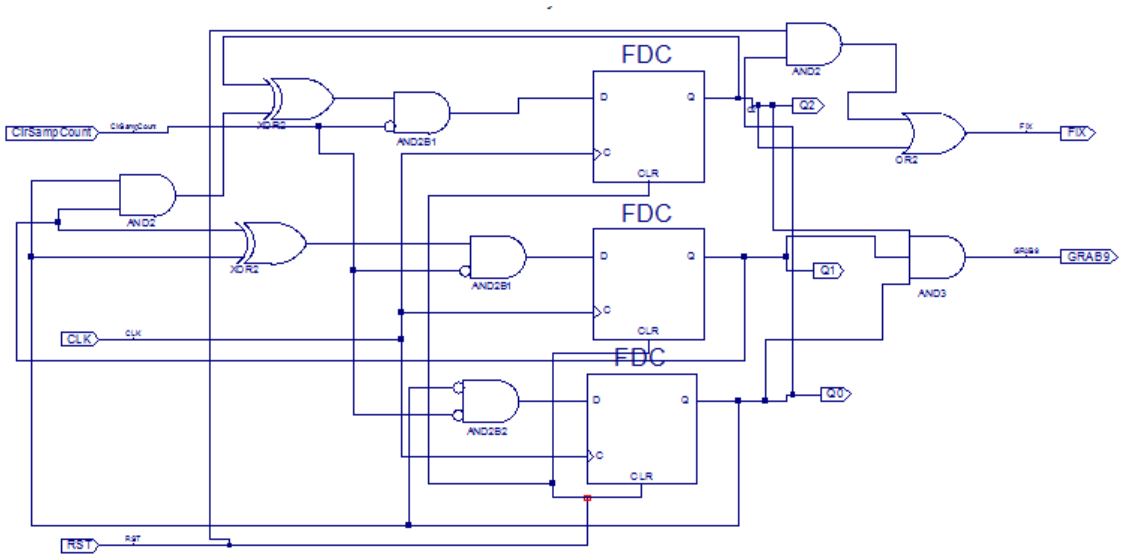

Figure 3.4: SampCount Schematic

Figure 3.4 above shows the SampCount schematic.

3.5 BitCount:

The idea of having the BitCount is to count the 8 data bits, and the stop bit. There would be 4 flip flops required to use. These D flip flops but with an enabled. This enable will be set by using the GRAB pulse as the input. The other inputs are clock and reset. The circuit has two outputs BIT8, BIT9, and the Q output from each flip will be stored. The BIT8 counts

from 0-7 and can go high in the 8th count. Therefore, the BIT9 counts from 0-8, and go high in the 9th count.

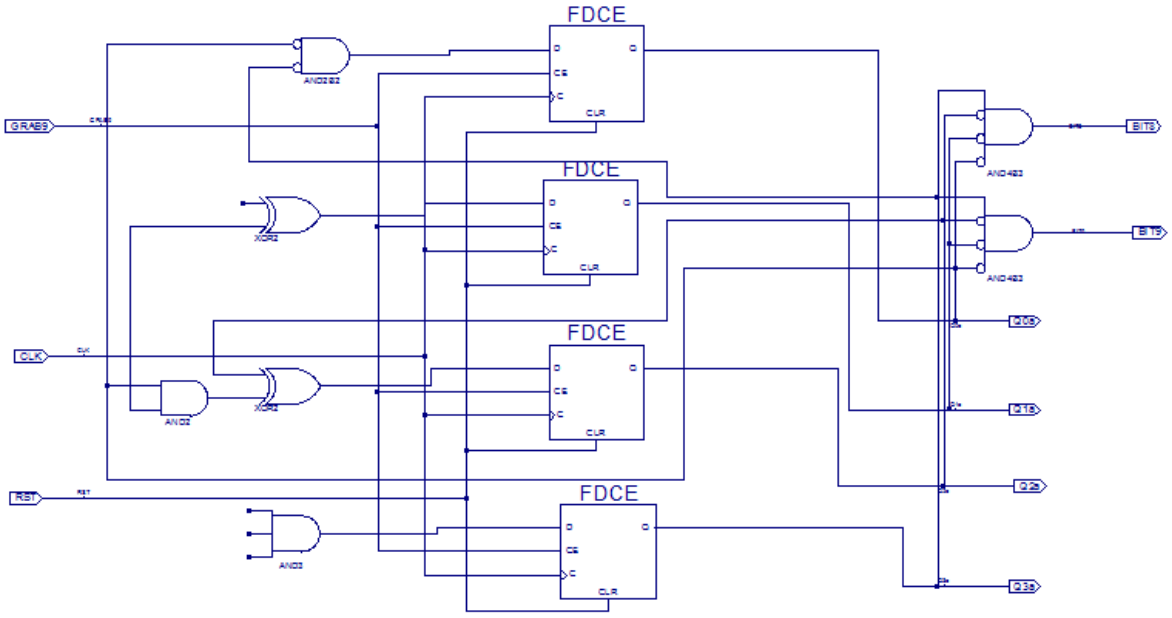

Figure 3.5: BitCount Schematic

Figure 3.5 above shows the BitCount schematic.

3.6 CountClear:

The purpose of having a CountClear is to make sure SampCount perform the counting correctly. At first, it checks if the GotStart found the start bit, which means that (GotStart = 1). Next check will be CountClear input that has to consider BIT9. If BIT9 has a high value, that means the last bit is a stop bit. The last input will be FIX to make sure we do not have starting errors. The output is going to be ClrSampCount, which we want to be low so SampCount can start counting correctly.

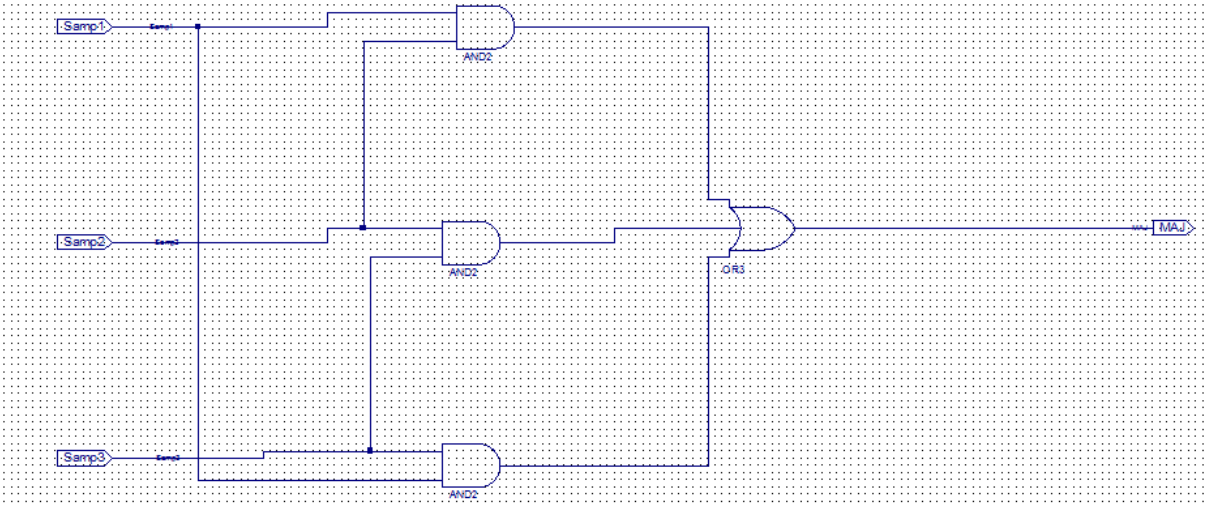

Figure 3.7: CountClear Schematic

Figure 3.7 above shows the CountClear schematic.

3.7 Ser2Par

Ser2Par purpose in this lab is to convert the data entered to parallel to allow synthesizer to read them. When GRAB8 is high, 8 serial bits will be converted to 8 parallel bits. This can be obtained by using 8 enabled D flip flops. The schematic of Ser2Par is shown below in Figure 3.8.

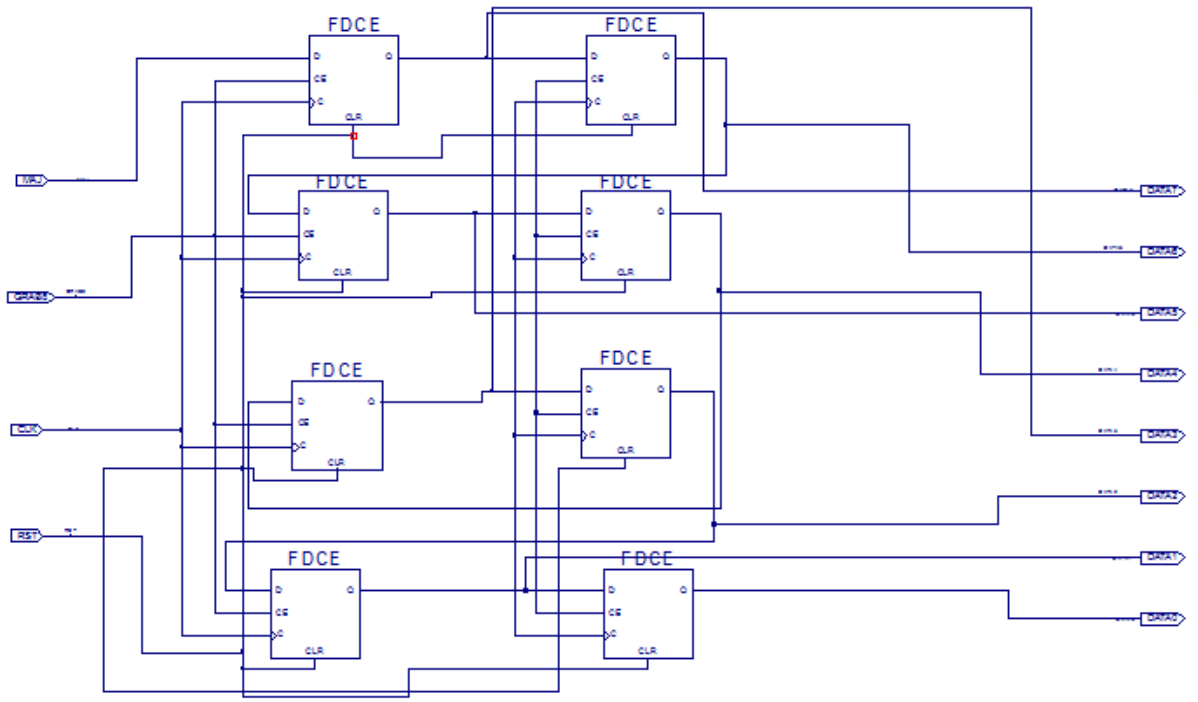

Figure 3.8: Ser2Par Schematic

3.8 The Whole MIDI

The whole MIDI interface schematic is shown below in Figure 3.9.

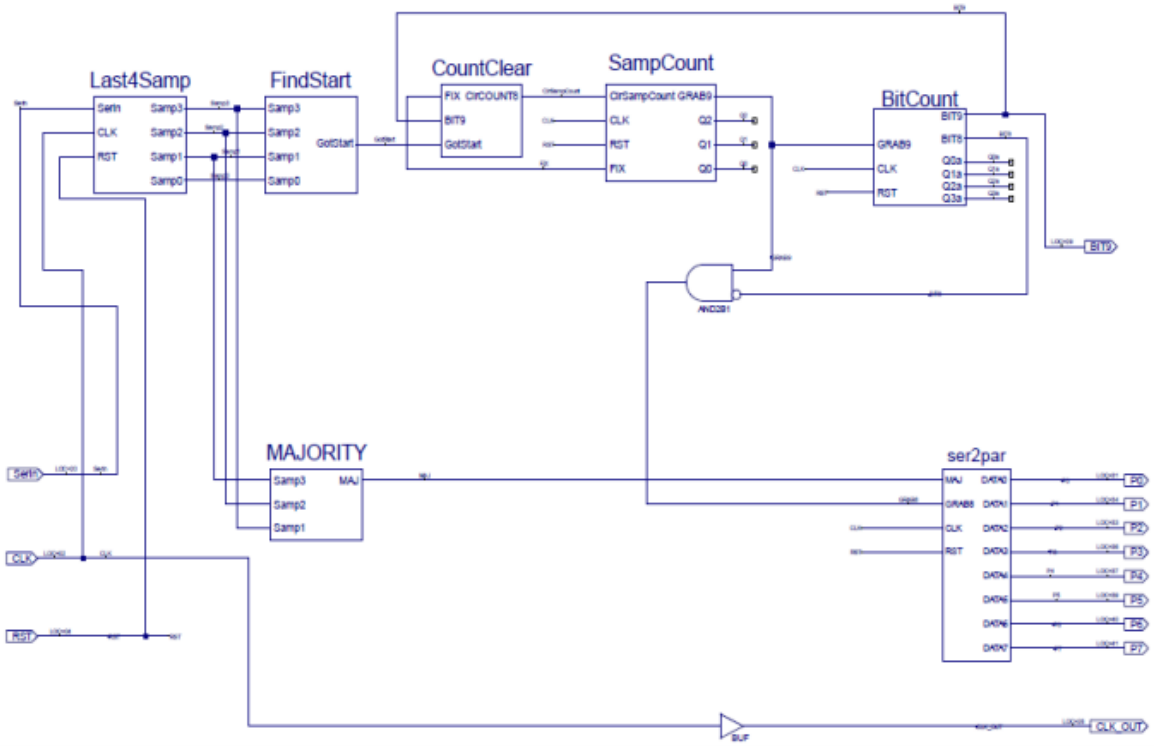

Figure 3.9: The Whole MIDI

4.0 TESTING AND IMPLEMENTATION

The MIDI circuit that was constructed consists of 7 main blocks. These blocks were built using Xilinx ISE software and were implemented on ModelSim to run simulation. In this lab, we ran each block individually, we start by debugging the circuit and check for errors in the Xilinx design constructed by comparing it to our Prelab. After checking for errors, the Circuits were run on Modelsim. The waveforms of each constructed circuit is shown below.

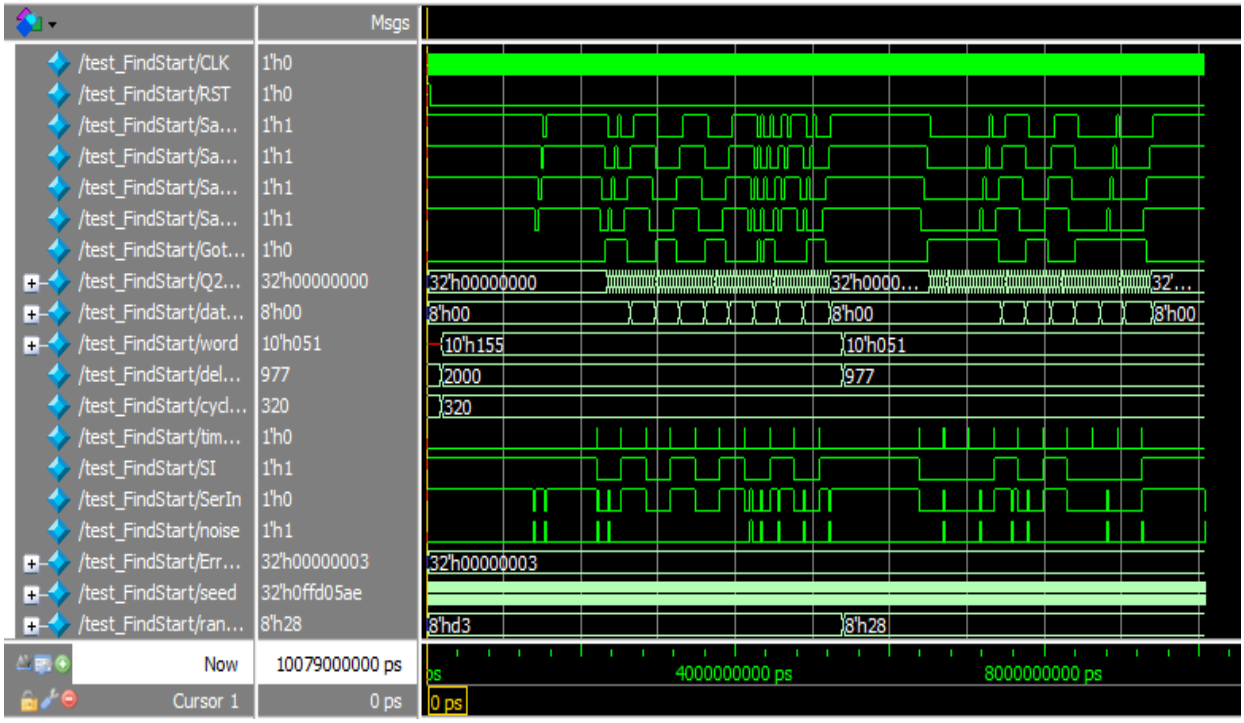

Figure 3.10: FindStart Simulation Waveform

Figure 3.10 above shows the waveform outputted for the “Find start” which is identical to the implemented Circuits.

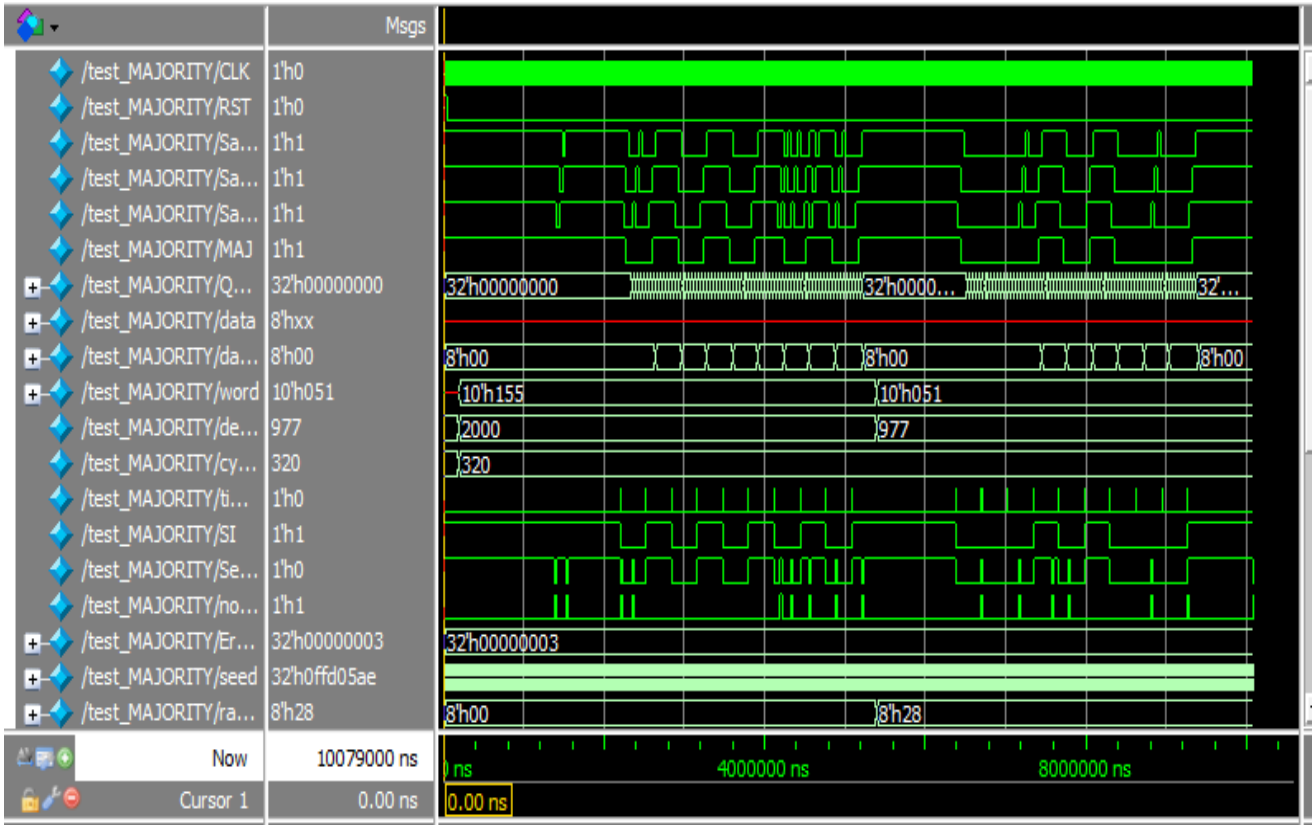

Figure 3.11: Majority Simulation Waveform

Figure 3.11 above shows the majority output which is identical to the implemented circuit.

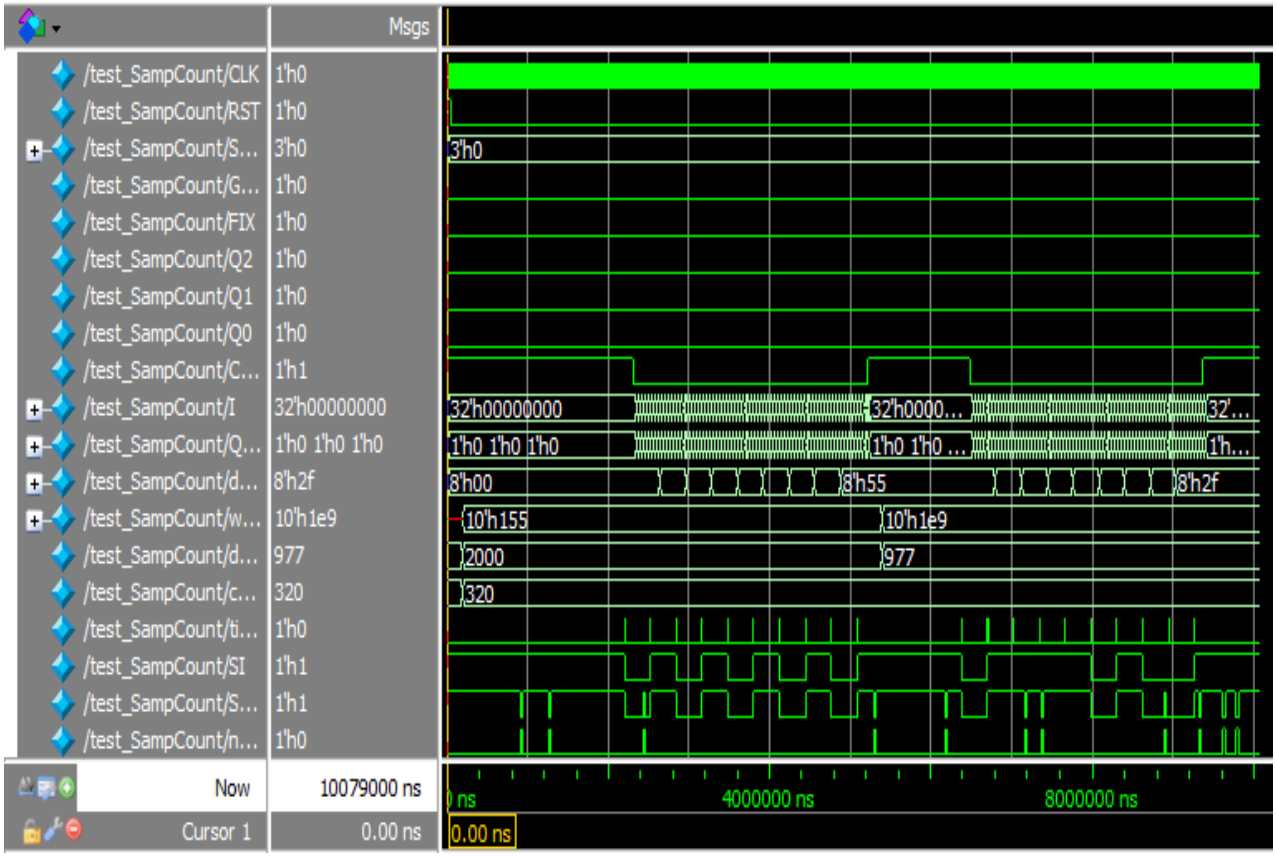

Figure 3.12: SampCount Simulation Waveform

Figure 3.12 above shows the SampCount output which is identical to the implemented circuit.

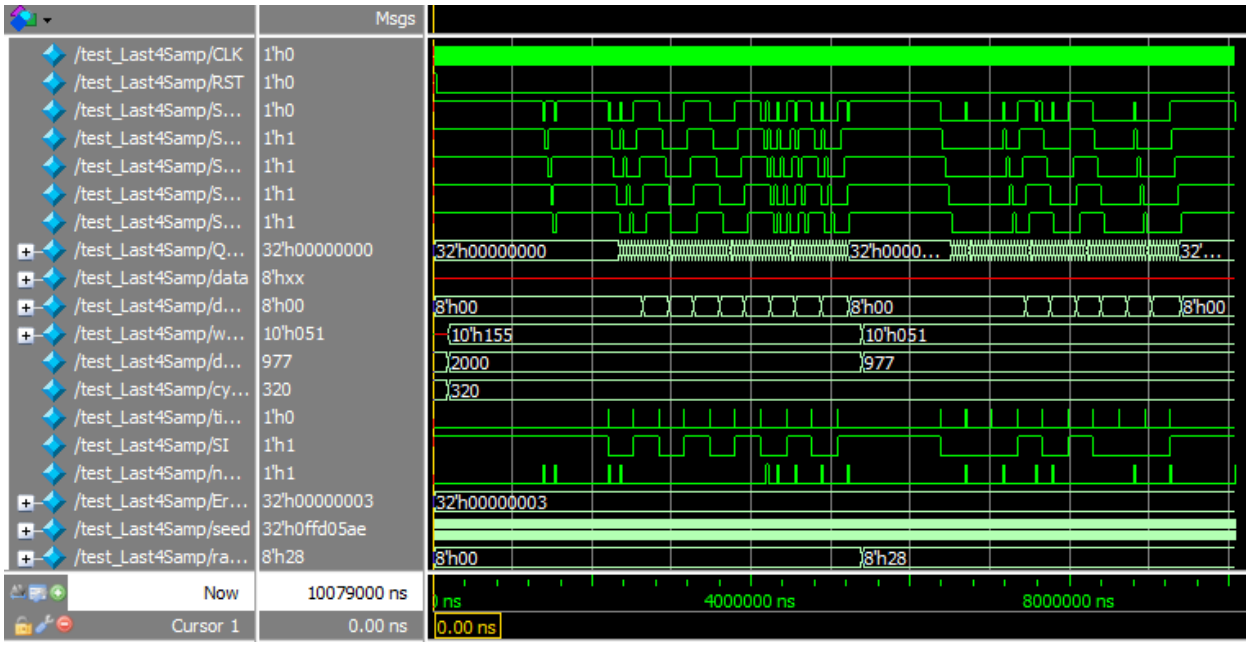

Figure 3.13: Last4Samp Simulation Waveform

Figure 3.13 above shows the output for Last4Samp corresponded to samp0, samp1, samp2 and samp3 of the Implemented circuit.

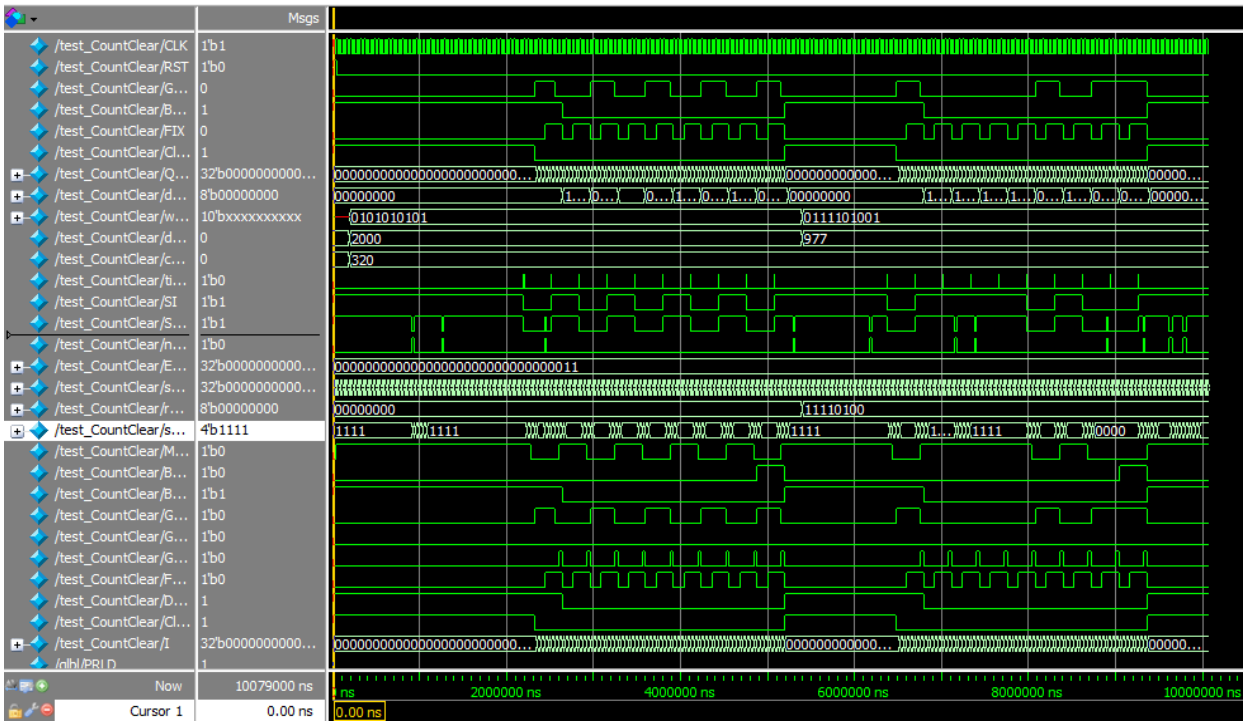

Figure 3.14: CountClear Simulation Waveform

Figure 3.14 above shows the output of the CountClear circuit.

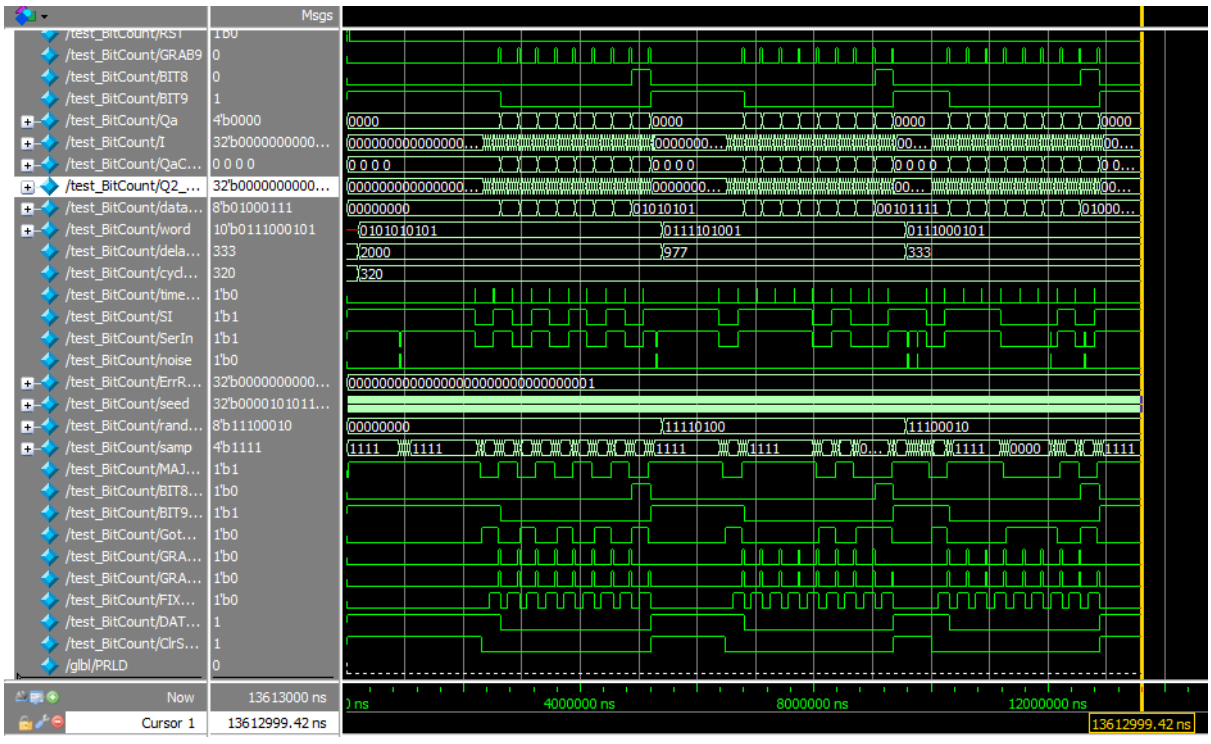

Figure 3.15: BitCount Simulation Waveform

Figure 3.15 above shows the output of BitCount circuit.

Figure 3.16: Ser2Par Simulation Waveform

Figure 3.16 above shows the Ser2Par simulation waveform.

5.0 CONCLUSION

In conclusion and based on the results obtained in this lab. The main objectives on this lab were successfully achieved. The test constructed on the CPLD matched the serial inputs to their corresponding output values. Our prelab was a good guide. Some issues were faced while working on Xilinx program since the prelab was long and exhausting to follow the design of each circuit. Looking on the full design we built. This design is advanced and can be seen in a real-life application. MIDI interface allows devices to interact with each other and also communicate with one another using the MIDI messages. It can be seen as a step gate for further complicated designs. applicable in our generation for creating electronical circuits that would develop the way music/ film/media work. Finally, we usually manage to finish the lab barely on time, but this lab was divided to 2 parts which helped us to discuss and further understand the lab and we took all the time we needed to finish.