ECE 385

Fall 2020

Experiment #3

Logic Processor

Alex Wen (acwen2), Manav Agrawal (manava3)

Section ABF

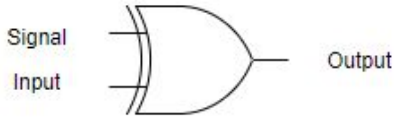
Xinbo Wu

Introduction

This lab delves into the understanding and implementation of a bit-serial logic operation processor through the use of CMOS logic. The purpose of this processor is such that it can perform basic arithmetic logic operations including AND, OR, XOR, NAND, NOR, and XNOR from 2 4-bit designated registers, which in turn can store either the input data or the output of the arithmetic logic unit as designated by the routing selection.

Pre-Lab Inquiries

If there is a two input one-output circuit and one signal needs to be optionally inverted then a XOR gate can be used.



Input	Signal	Output
0	0	0
0	1	1
1	0	1
1	1	0

Figure 1: Optionally Inverting Signal through XOR Gate

If we see the signal determines if the output is equal to another input or not. If the signal is 0, this means that the input and the output are the same. If it is one that means that inverted input will be equal to the output. If seen, it can optionally invert the signal.

The modular design is very helpful as it helps to test each functional block of the circuit before the error gets too large and hard to fix. This also makes processing it faster. It also means that it is independent of others so the control unit won't affect how the ALU works. So they can be tested independently. This increases work efficiency and we do not have to fix or redo the entire circuit again and again. This experiment will be very modularized. Therefore the logic serial processor will be built on the same principles.

Logic Processor Operation, Implementation, and Block Diagram

The combinational logic circuit is designed to implement a 4-bit serial operation processor; its objective is to determine different operation functions (Figure 2). There are 3 partitions to this processor: 1 to store the values, 1 to perform the operational computation, and 1 to restore selected outputs back into the registers. The input switches are D3-D0, F2-F0, R1, R0, LDA, LDB, EXECUTE.

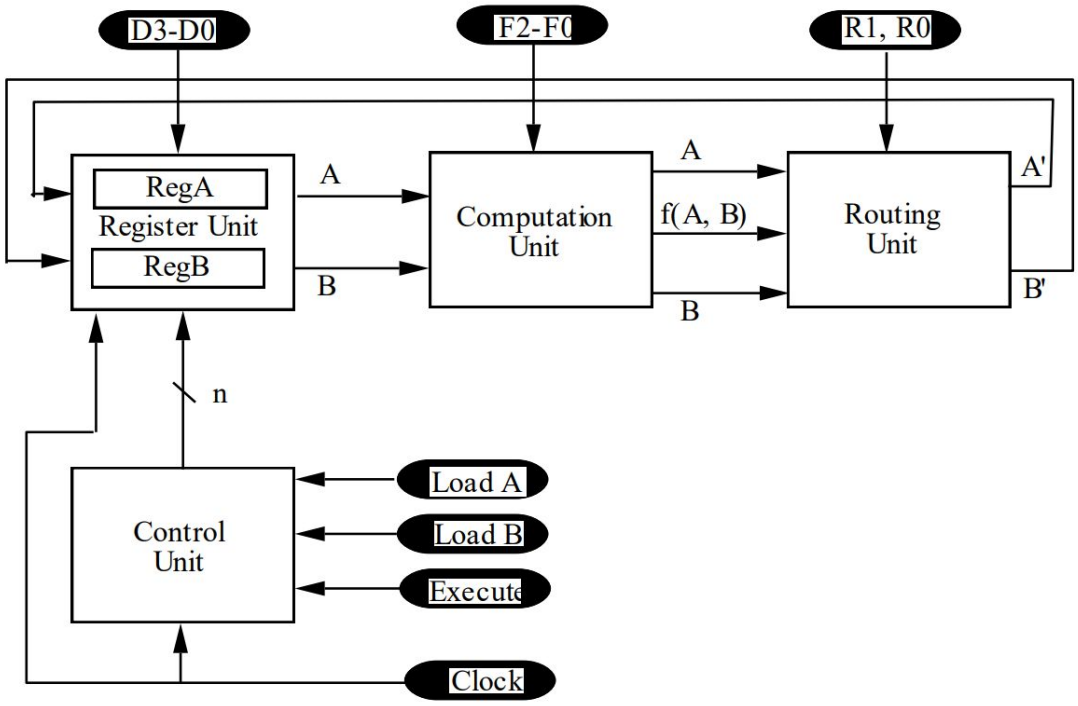


Figure 2: Block diagram of the Logic Processor

This block diagram will be modified as the design of the schematic is discussed.

(1) 2 4-bit data is needed through the use of D3-D0 switches that will input the 2 words into the registers. LDA and LDB switches from the control unit will allow for the input D3-D0 to be actually stored in the shift registers, from which the EXECUTE switch will send the output (2 words stored in the registers) to the computation unit. The values of the 4-bit words are also displayed on the DIP LEDs - 4 LEDs for each register.The data inputs are designed to be definite; in other words, once EXECUTE is set to high, the data will be processed to the computation unit even if EXECUTE is set back to low in the middle of the clock cycle. During this time, the functions determined by the other logic units are not considered, so inputs F2-F0, R1, R0 are not considered.

(2) In the computation unit, the unit obtains 2 4-bit inputs and implements a variety of logic that determines the Function output. The 6 operational logic taken into consideration by the processors are AND, OR, XOR, NAND, NOR, XNOR, along with 0 and 1-values. What operation the computation unit derives is dependent on the functional selection bits that will direct the operation to implement said functions (Table 1). As such, only F2-F0 switches are considered while the rest are not. These switches are fed into 4:1 Multiplexer 2-selection bits while A and B from the shift registers are fed into the input. However, since there are only 2 selection bits available, F1-F0 are fed in for the function

selection bits while F2 can be reorganized by acting as input into an XOR gate along with the output of the multiplexer.

(3) The routing unit takes the functional output from the computation along with values in register A and B, and logically selects which of the values should be stored back into both registers. The objective of the routing unit consists of 4 selections, therefore R1 and R0 switches are on high while the rest are not taken into account. As such, the design of the unit consists of a 4:1 multiplexer in which the inputs are values A, B, or the functional output of A and B (designated as f(A,B) yielding 2 outputs leading to respective registers (Table 1), which is demonstrated through DIP LEDs.

To note, the DIP LEDs are designed to show the contents of the registers A and B (shift registers). The LEDs will only be set to high either when data is loaded into the registers or the output of the routing unit is transmitted back into the registers.

Function Selection Inputs			Computation Unit Output
F2	F1	F0	f(A, B)
0	0	0	A AND B
0	0	1	A OR B
0	1	0	A XOR B
0	1	1	1111
1	0	0	A NAND B
1	0	1	A NOR B
1	1	0	A XNOR B
1	1	1	0000

Routing Selection		Router Output	
R1	R0	A*	B*
0	0	A	B
0	1	A	F
1	0	F	B
1	1	B	A

Table 1: Function and Routing Selections Truth Tables

This implementation was done with both switches and FPGA. The switches handled the selection for both the computation unit and the routing unit; each switch is powered and connected to pull-down resistors to ground to prevent any potential lag that may short circuit the circuit. The FPGA handled the 3.3V supply power and ground, the data and control switches and the clock cycle.

As the logic units in the bit-serial processor must maintain its data as the data goes through computation and routing along with some storage, a 1kHz clock cycle is supplied by the FPGA. This will directly transmit the output to the LEDs. During the troubleshooting process, the FPGA clock cycle is set to 1 Hz; this enables for a step-by-step analysis of the circuit function to locate and fix any errors. When the clock cycle is set to 1 Hz, it will take up to 4 clock cycles for the registers to maintain the data.

State Machine Diagram

The Logic serial processor requires a state machine diagram. So if brainstormed, there are 6 states, 1 is the start/reset state, the other 4 states will be the shifting state and the sixth state will be the halt state. The middle four states can be implemented by a counter. So a 3 state moore machine can be created.

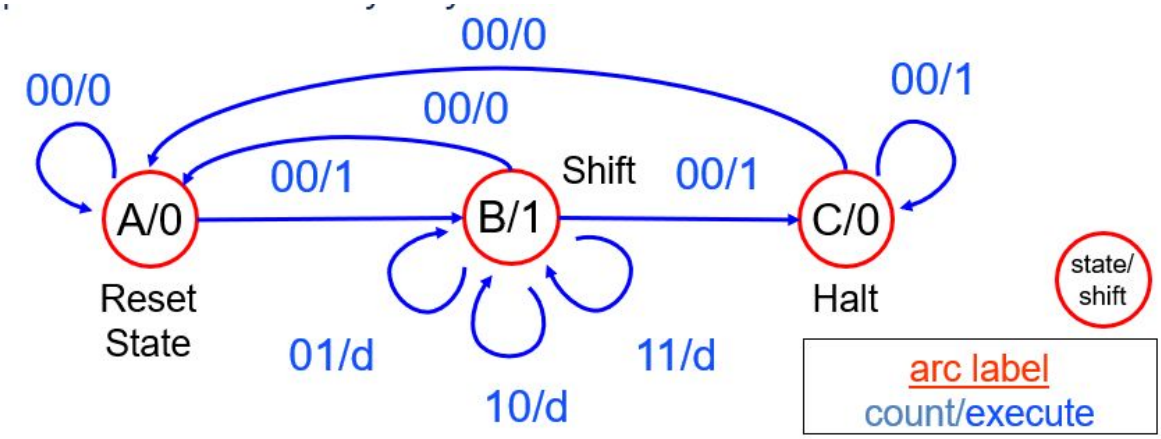


Figure 3: 3 State Moore Machine (Prof. Cheng)

The figure 3 shows the 3 state moore machine. The first state is a start or a reset state. This state is when the register will not shift. When the execute is active high, it goes to the shifting stage for 4 cycles which is controlled by the counter. Thirdly, if the execute is still active high, it moves to the halt state so it does not continue shifting. But if the execute is low, it goes to reset state. Now if it is observed, the transition from shift to reset and halt to reset state is the same. The only difference between halt state and shift state is that the halt state does not have the output Shift as 1. However this can be changed if this is made as a Mealy machine. The second state and third state can be combined with the inputs execute and the counter outputs 00. So as soon it reaches 0 and if the execute is still 1 in the shift state, the shift output can be 0. Hence, this can be transformed in the below Mealy machine which makes it 2 states or only 1 bit encoding. Originally the Moore machine was 3 states or 2 binary flip-flops.

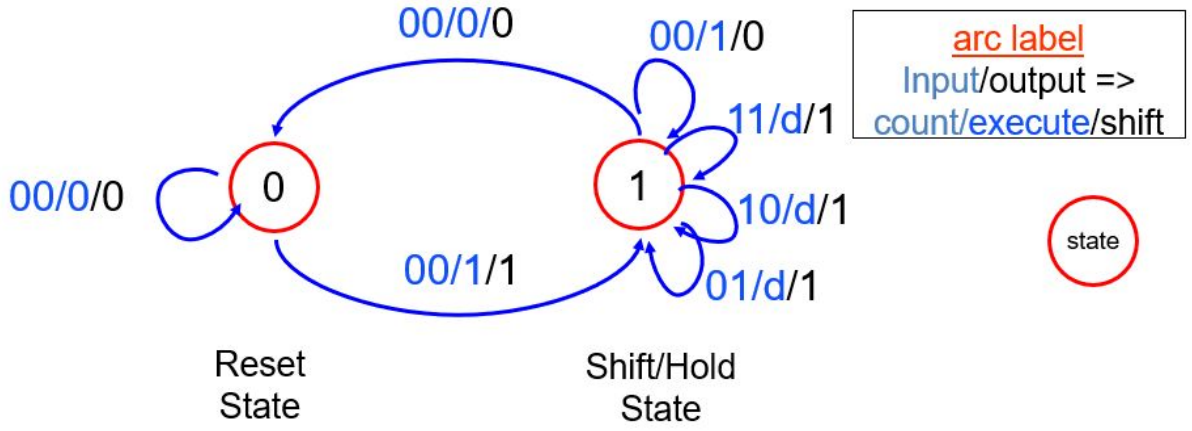


Figure 4: 2 State Mealy Machine

This shows how it is reduced to 2 states. The output is written outside the state in black number. (Refer to the legend for better understanding) Now if seen, the output does not depend upon states but counter inputs and the execute switch. This means that now only 1 D Flip Flop needs to be used. This Mealy state machine was used in the experiment to create this logic serial processor.

Design Process and Schematic

Since the block diagram has been explained and the state machine diagram has been constructed, now the design process can begin. Firstly, the control unit which will dictate whether the register should shift to the ALU, load in new inputs or to keep it constant. The control unit will implement the state diagram. As this experiment will implement the mealy diagram as above, the following truth table was created. The register should shift or keep its values is determined by whether execute switch is on/off, what state it is on and what values the counters are on? The register should only shift for 4 cycles hence it is necessary to check whether the counter is done for 4 cycles as seen in the above state diagram.

Table 2: Control Unit K-Map

E	Q	C1	C0	S	Q+	C1+	C0+
0	0	0	0	0	0	0	0
0	0	0	1	d	d	d	D
0	0	1	0	d	d	d	D

0	0	1	1	d	d	d	d
0	1	0	0	0	0	0	0
0	1	0	1	1	1	1	0
0	1	1	0	1	1	1	1
0	1	1	1	1	1	0	0
1	0	0	0	1	1	0	1
1	0	0	1	d	d	d	d
1	0	1	0	d	d	d	d
1	0	1	1	d	d	d	d
1	1	0	0	0	1	0	0
1	1	0	1	1	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	1	1	0	0

From this truth table, we see the C1+ and C0+ can be implemented by a counter. What needs to be designed from this is the next-state logic and the output of the state diagram is whether to shift the register or not to. Hence, we can create 2 4x4 K-maps for Q+ and S.

Q+	C1C0 = 00	C1C0 = 01	C1C0 = 11	C1C0 = 10
EQ = 00	0	X	X	X
EQ = 01	0	1	1	1
EQ = 11	1	1	1	1
EQ = 10	1	X	X	X

Table 3: Q+ K-Map

As seen from the Q+ K-map, there are 3 eight element loops which can be used. The expression is Q+ = E + C1 + C0. So this can be fed into the D Flip-Flop for the next-state. Now the second k-map for the output shift (for register) will be created.

S	C1C0 = 00	C1C0 = 01	C1C0 = 11	C1C0 = 10
EQ = 00	0	X	X	X
EQ = 01	0	1	1	1
EQ = 11	0	1	1	1
EQ = 10	1	X	X	X

Table 4: S for Register K-Map

In this Shift output k-map, there are 2 eight element loops and 1 four element loop which will be used as a way to use as a signal to determine whether the register should shift or hold. The expression is Q+ = EQ' + C1 + C0. Moreover, one more design choice needs to be made. The counter that will be used for this experiment is 74161. Since there are three modes we want, RESET, COUNT and NO CHANGE. For no change, pin 7 or pin 10 needs to be turned to active low. For it to increment, the pin 7 or 10 needs to be turned to active high. So the pins 7 and 10 can be connected with the signal S which makes it easy. The pin 1 is connected to a reset switch for debugging purposes so that it can reset the counter. It can be connected to the next state for automatic reset.

The last small part about the control logic is for the shift register to determine when to utilize the shift signal or the load A/B signal. If it is seen, the 74194 shift register modes are controlled by pins 9(S0) and 10(S1). When both pins are active low, the shift register just remains constant. When both of them are active high, it loads in data values at the rising clock edge. When S1 is active low and S0 is active high, it shifts the data.

S1	S0	Mode	Load A/B	Shift
0	0	Constant	0	0
0	1	Shift Right	0	1
1	1	Load in	1	0

Table 5: Shift Register Logic Unit

S1 is only 1 when LOAD is active high hence S1 is connected to each of its load switch. For S0, there is some more logic. S0 is when either Load is 1 or Shift is 1 so the Shift and Load needs to be put into an OR gate and then to pin 9 which is S0 of 74194 chip datasheet.

All the expressions and the workings have been deduced for the control logic, let's work it out for the register, computation and routing unit also.
The register unit will have 2 74194 registers which will primarily be shifting right, loading or keeping it constant. It will have values of A and B and there is not much there. The computation unit will compute the basic 8 arithmetic functions of a single bit. From the set of tables below, it

can be seen that there are F2, F1, F0 however, we have to implement 8 functions. The first thought will be to use a 8:1 Multiplexer but upon a closer examination, when F2 is 1, the functions are just inverted. So a 4:1 Multiplexer can be used and then the output of the multiplexer can be XOR with F2 which will invert when F2 is 1. As seen from table 1, the routing unit is just a 4:1 multiplexer with select signals R1, R0 which will be switches.

Now the schematic of the entire circuit can be designed. Firstly, the control unit can be implemented. So let's implement the control unit's first part, the D-flip flop with combinational logic.

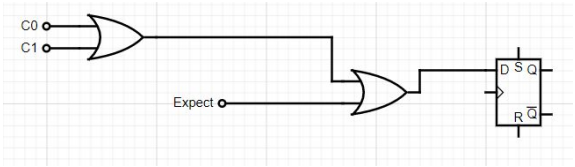


Figure 5: Basic D Flip-Flop Logic with AND/OR Gate

Now this image shows how the combinational logic expression Q+ = E + C1 + C0 will be fed as next state logic to the D Flip Flop. But since the CMOS chips that the kit has does not contain a OR gate, a NOR Gate will be used. It is even better if we use a 3 INPUT NOR 7427 as this makes wiring easier and more efficient. The above image is a basic representation, this will be implemented better in the fritzing. The basic Control Unit with only AND/OR Gates is shown below. There is a counter and a D Flip-Flop.

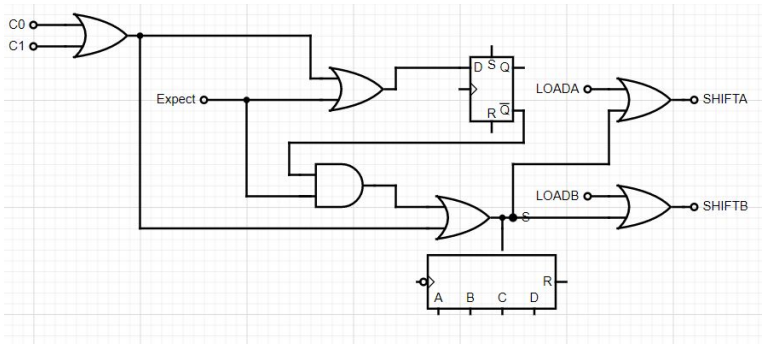


Figure 6: Basic Control Unit with AND/OR Gate

This figure 6 shows the implementation of the basic control unit with AND/OR gates where the D Flip-Flop and counter are more BLACK boxes. The OR can be converted to NOR and a hex inverter in front of it. The AND can be converted to 2 NAND or 1 NAND and 1 NOT gate. The Fritzing will use NOR/NAND Gates to implement the logic. This can be represented as a functional block and converted to fritzing as below.

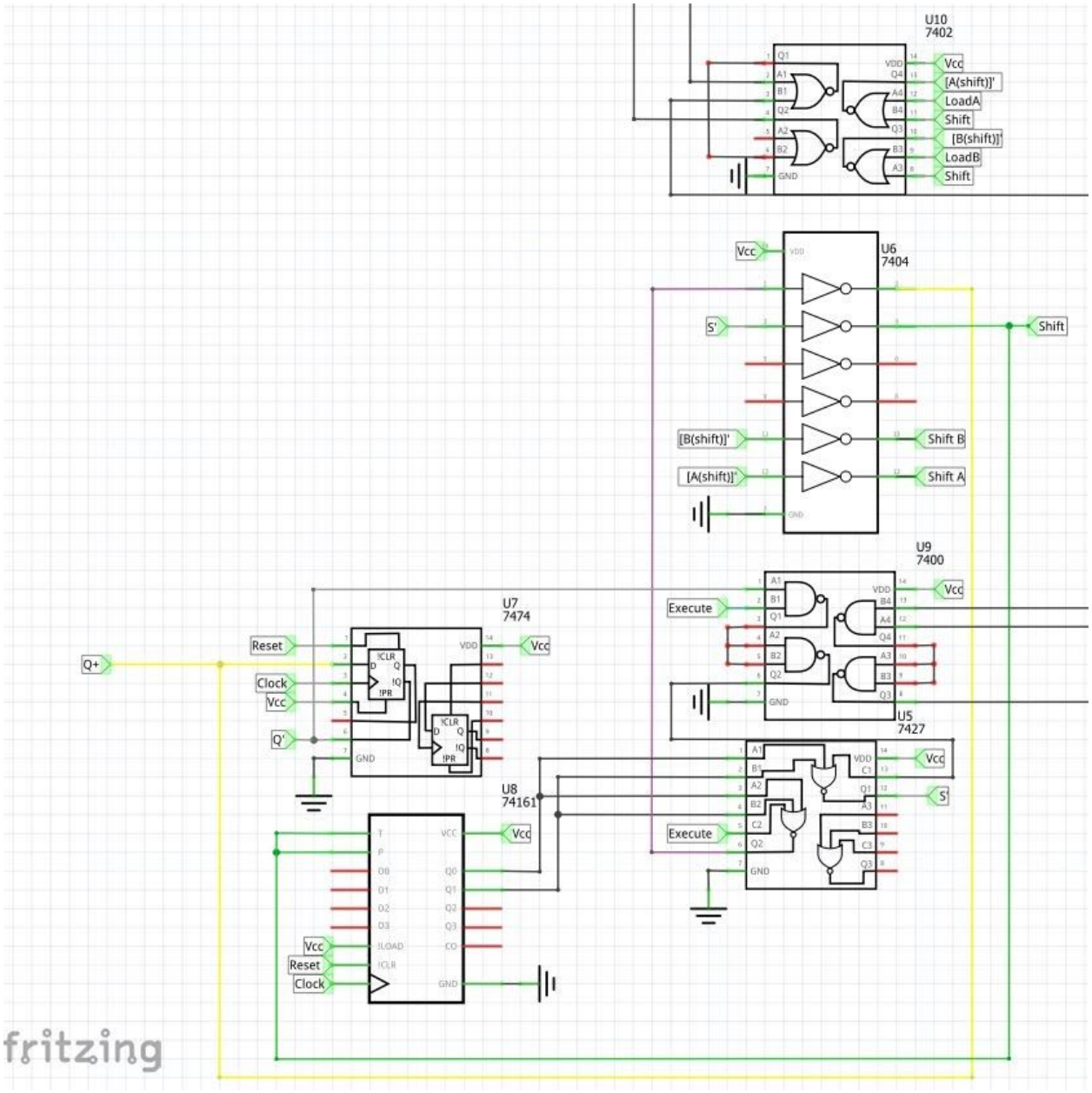


Figure 7: Control Unit of the Experiment (Partial Fritzing Schematic)

This clearly shows how the experiment implemented the control logic. The yellow wire is the next state logic wire. The green wire shows the shift signal. This is the detailed schematic of the control unit in Figure 7. Now the other parts of the circuit can be implemented. The register unit is simple. The Shift A and B signal from the inverter will be fed onto the register. The register will be configured as discussed during the k-maps and logic designing. Now onto the creation of the ALU or the computation unit. Since the lab kit only has 4:1 multiplexer, it was decided to use XOR to optionally invert the signal using F2 as the second input. This can be designed onto the logic gates to get a better understanding.

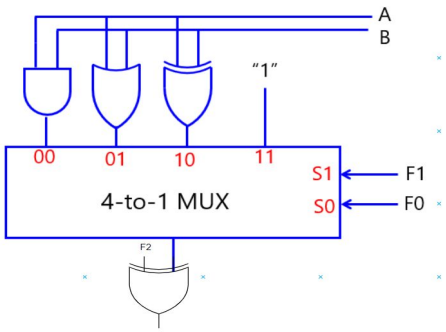


Figure 8: ALU with F0, F1, and F2

Since we don't have AND/OR, NAND/NOR will be used. This is going to be the new circuit.

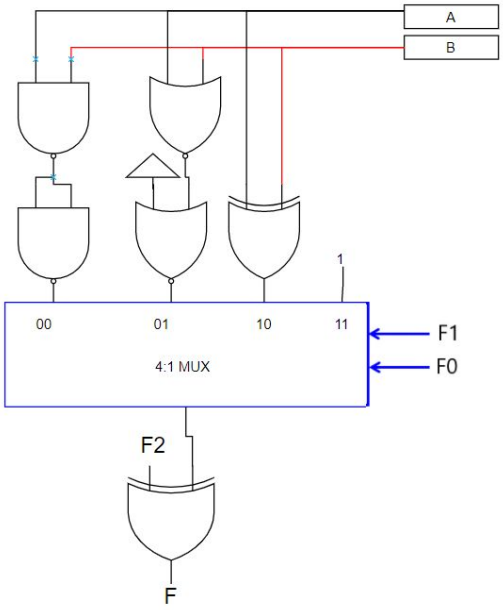


Figure 9: ALU with NAND/NOR with F0, F1, F2

These are all the logic gates used. For the routing unit, it is just a single implementation of multiplexer. So if the register unit, ALU unit and routing unit is combined, the other part of the circuit will be in the following figure. This will create a better block diagram.

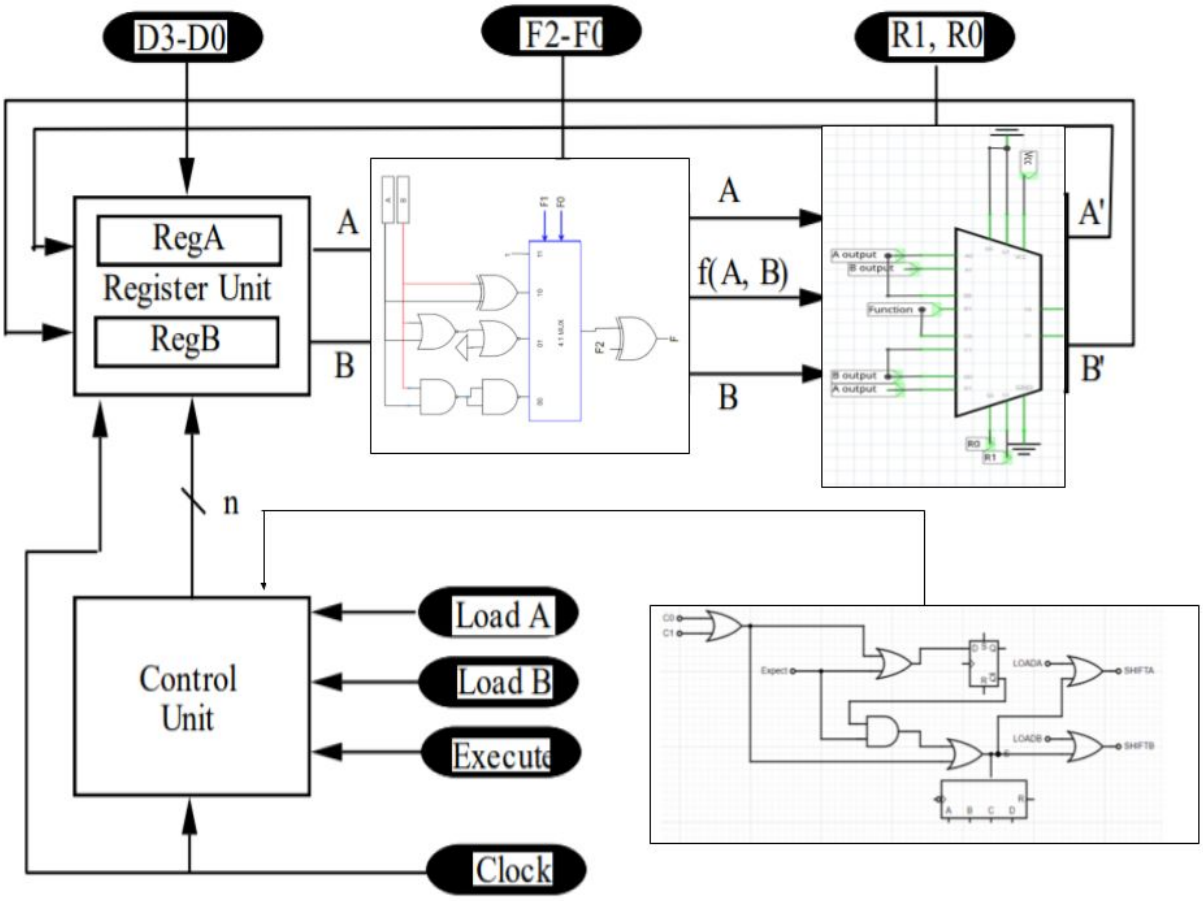


Figure 10: Modified Block Diagram

This is the modified block diagram which shows each and every block filled with logic and its respective multiplexers. It is not the exact design but it gives a better overview of the entire circuit which will be implemented. The control unit consists of the circuit which was shown above and the functional block was already created in fritzing for a modular design. Now the register, ALU and routing will have a functional design which will be seen in the figure 11 which will implement the top row as seen by the modified block diagram.

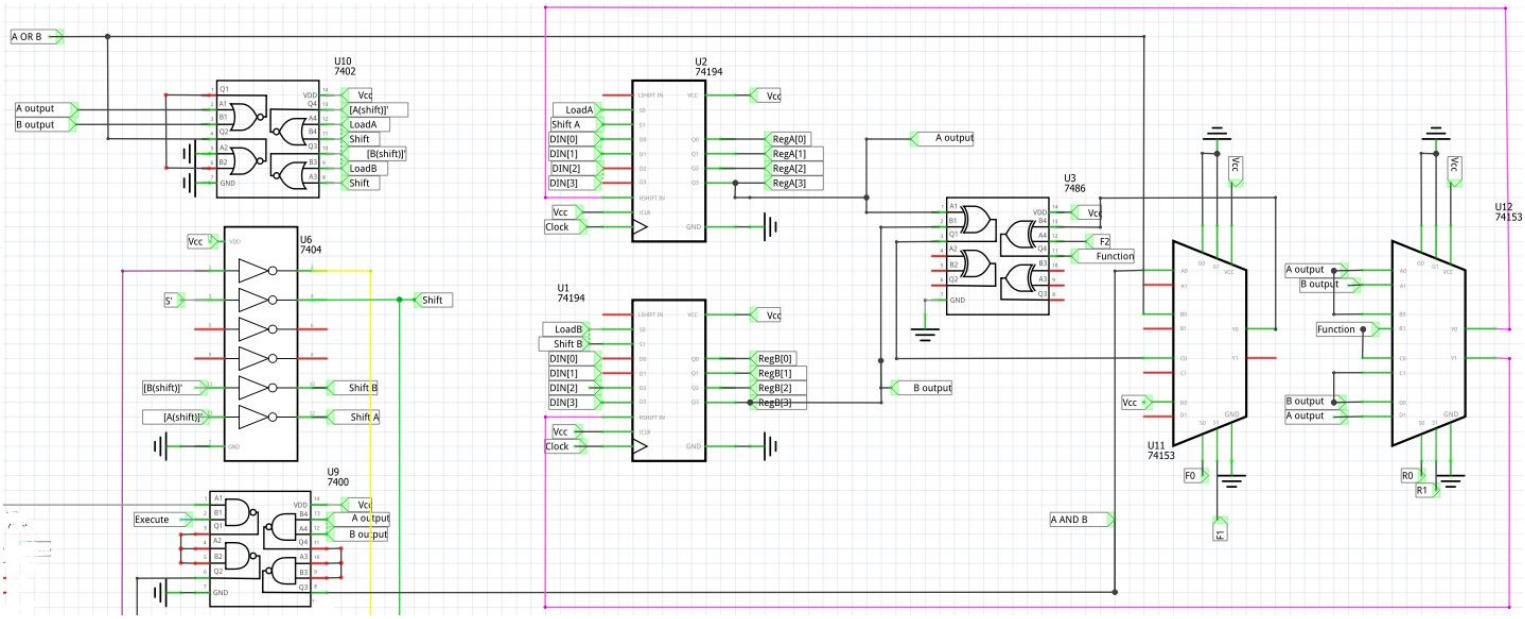


Figure 11: The other part of the Fritzing Schematic (Reg Unit, ALU, Routing)

All of this can be compiled to a singular fritzing schematic which is what will be designed in the breadboard. Below is the full fritzing schematic.

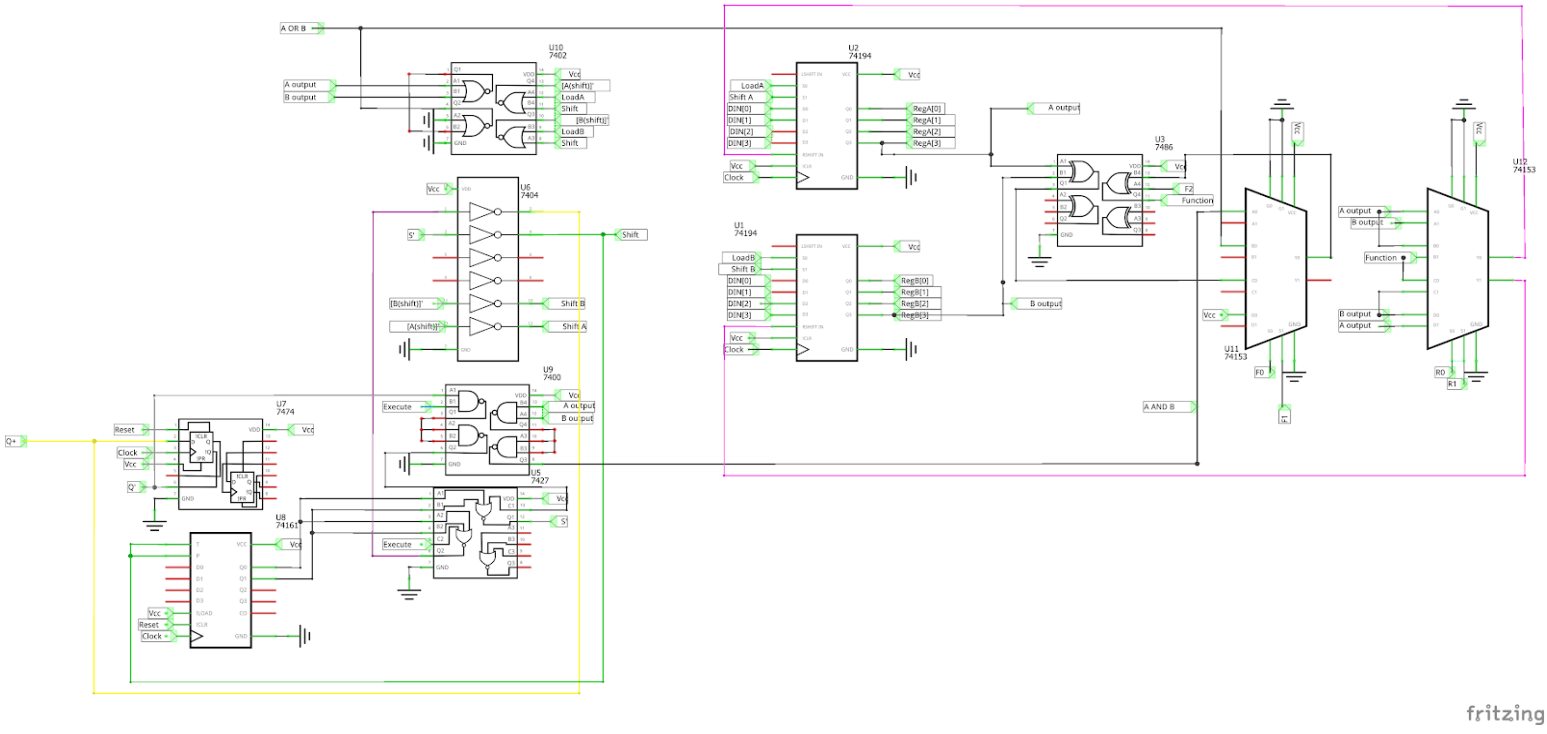


Figure 12: The Full Fritzing Schematic Showcasing the Logic Serial Processor

This was assembled into a breadboard which is as follows below.

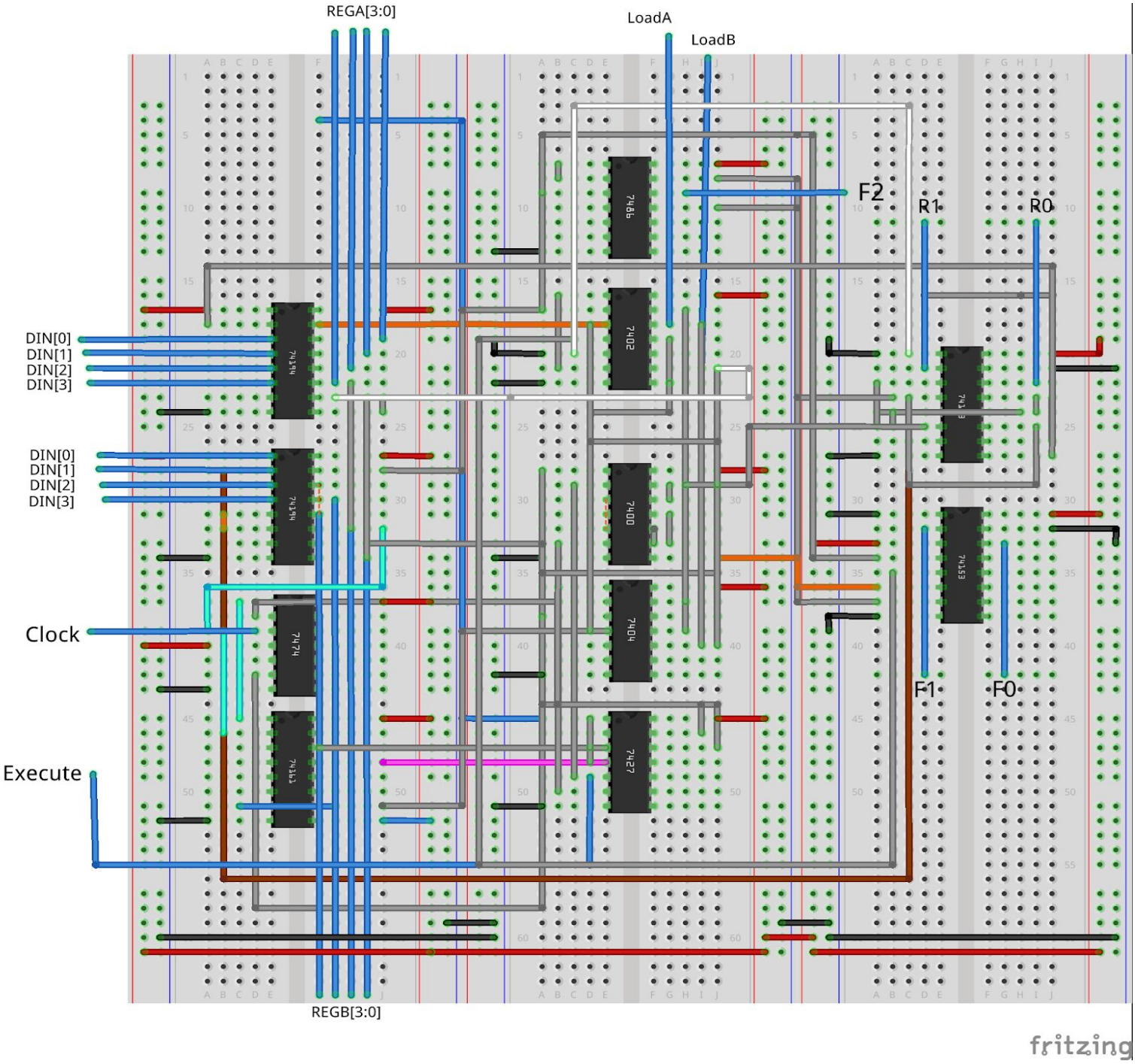


Figure 13: Fritzing Breadboard View of the Circuit

This was a similar way of how the breadboard was assembled. The blue wires are what went to the FPGA, switches and LEDs. The CMOS chips were arranged in the best way possible.

Unfortunately, before taking a picture the circuit was disassembled by the TA. In the appendix, there is a photo which just shows the remains of what the breadboard looked like.

There were multiple design choices which were considered with itching having its different tradeoffs. Firstly, like the previous experiment, it was pondered upon to decide between 74194 vs 74195. Since 74194 has a single right shift pin and the modes matched well with the logic, 74194 was opted. The 74161 was the best counter since our entire circuit was clocked and its modes were also easily connected to logic. The main priority was to lose as least CMOS chips as possible so it makes it very simple and easy. 7400 (2-input NAND), 7402(2-input NOR), 7427(3-input NOR) and 7404(Hex Inverter) were the 4 chips which were used in the best way possible. Inverter chip was useful because there were a significant amount of outputs which needed to be inverted and using a separate NAND/NOR wastes the advantage of having a NOR/NAND gate. In addition, the main 8 switches were put on FPGA to easily demonstrate the functionality.

Debug Process

The lab had multiple components so it is easier to debug block by block. So since the circuit was not working properly, the first unit that was started with debugging was the control unit. The LED 9 and 10 were connected to the counter values to observe if it works well. If it increments and holds 00. This functionality was not working which meant that our problem was either with configuring the counter or the logic for it. So the pin 7 and 10 were started to debug. The logic was fine but since the counter wasn't working, another logic was tried where the states and the third bit of the counter is used. But the result was the same. Over looking at the wires of the counter, it was figured out that the connections for switch 6 and 7 were flipped and switched. So the connections of switch 6 and 7 were rerouted. This fixed the counter. Just to make sure that the control unit worked, the D Flip Flops were again inspected to see if it outputs the correct state. It was working fine and finally the shift signal was also working. After the control unit, the circuit was not working and the LEDs suggested that there was something wrong with the register unit. While checking the connections, it was discovered that the right shift and left shift was confused. The shift register (when right shift enabled), goes QA to Qb to Qc to Qd. So the bit that the operation needed to be conducted on is pin 12 not pin 15 so those connections were again rerouted. While this was being checked, it was discovered that some of the switches that were assigned for function and routing were loose so those were fixed. After fixing the register unit and the switches, and a couple of rerouting some pin assignments, the logic serial processor started to work.

Post-Lab Inquiries

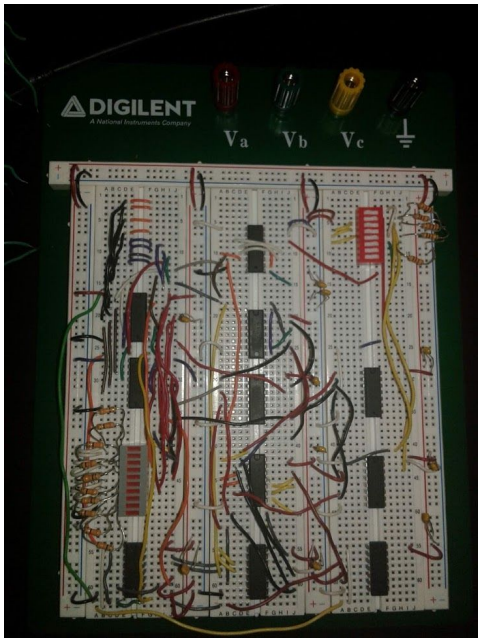
As seen above, a modular design helps to isolate where the error arises from. It was clear how the wiring faults were resolved. The CMOS logic can be ever expanding and without a proper schematic or guide, it can take an exponential runtime to fix any errors. Now the main question that arises is the tradeoff of choosing a Mealy Machine vs a Moore Machine. How does choosing Mealy benefitted this lab and how will it be harder? Firstly, using a Mealy machine decreased the number of states as you did not need the Halt state. It could be fixed by using the input as a way to determine the output. Now this saves the number of D Flip-Flops used or essentially the memory that a device needs. On larger processors, a Mealy machine helps to save a number of bits in a memory otherwise it becomes troublesome. The disadvantage is that now the output also depends upon the input. Any glitches in the input will affect the output. In Moore, the output was just determined by its state so if that is correct, the functionality is fine. Only the glitches in D Flip-flop affected Moore machine. But now, in a Mealy machine, there can be input glitches from the D flip flop or the switches which act as the inputs. In a faster clock cycle, it can be hard as the output may be more unreliable. A Moore machine is also simpler to implement whereas in Mealy you need more logic after the state with the inputs so in a sense it may increase the logic gates.

Conclusion

The logic serial processor was implemented in four parts: control unit, register unit ALU/computation and routing unit. This experiment used a 2 state Mealy machine as shown in the above diagram. The breadboard successfully implemented the schematic as shown in the fritzing. Through modular design, the experiment was carefully assembled and the debugging of circuits was made easier. In addition, in ECE 120, the Mealy machine was not formally taught however in this class, the Mealy machine was taught and implemented which made a huge learning opportunity for future labs and experiments. This logic processor is a very basic representation of how a CPU might do computations. Nowadays advanced SRAM is used and parallel processing is used. However, this is a great step towards computer architecture. This lab also encompassed decision-making of chips and state diagram and this makes for a better engineer as it is an engineer's job to optimize by making important decisions.

There is a google drive link with images.
https://drive.google.com/drive/folders/1EmxGR0canqdEwE5urSF3MVcZ8RNdAqCL?usp=sharing

Appendix



1) **The partial breadboard (Photo taken after disassembled as asked by TA)**