

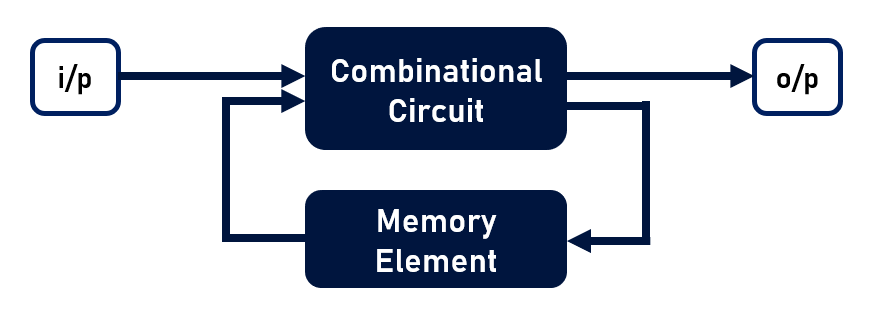
**Project**

**Report**

**Team Name : $514**

**Project Name : SAP-01**

**Sequential Logic Circuits**

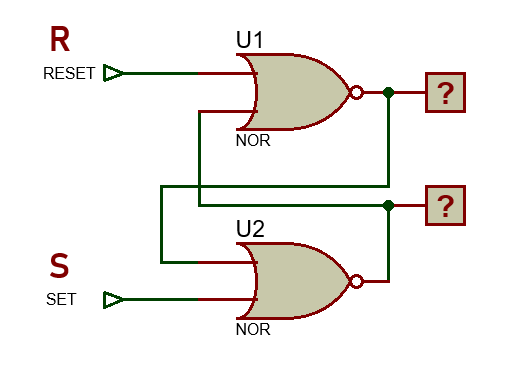
Unlike combinational circuits, in sequential circuits outputs vary based on input. Here is a general block diagram designed for sequential circuits,

**Latch**

A latch is an electronic logic circuit that has two inputs and one output. One of the inputs is called the SET input; the other is called the RESET input.

In Digital Electronics, a latch is a kind of circuit that has “Set” and “Reset” mode and we get a corresponding o/p based on whether it is a set or reset. Based on trigger of input units, circuits can be either ACTIVE HIGH or ACTIVE LOW. Generally, in Active high mode, Set refers to logic state 1 and Reset refers to Logic state 0. But when the circuit is active low, with logic state 0, we get the set mode and with logic state 1 we will get reset mode.

Q is our output for the latch. And or Q bar is the complement of the output Q.

**The basic S-R (Set-Reset) Latch**

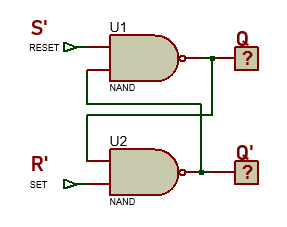
Active High Latch (w/ NOR Gate)

|  |  |  |  |
| --- | --- | --- | --- |
| **S** | **R** | **Q** | **Q’** |
| 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 |
| 1 | 1 | *invalid* | |

Here, S=1 means SET and R=1 means RESET.

And when S=0 and R=0, it will hold the previous memory. And we will get unpredictable results for when S=1 and R=1. So we label it’s output as invalid.

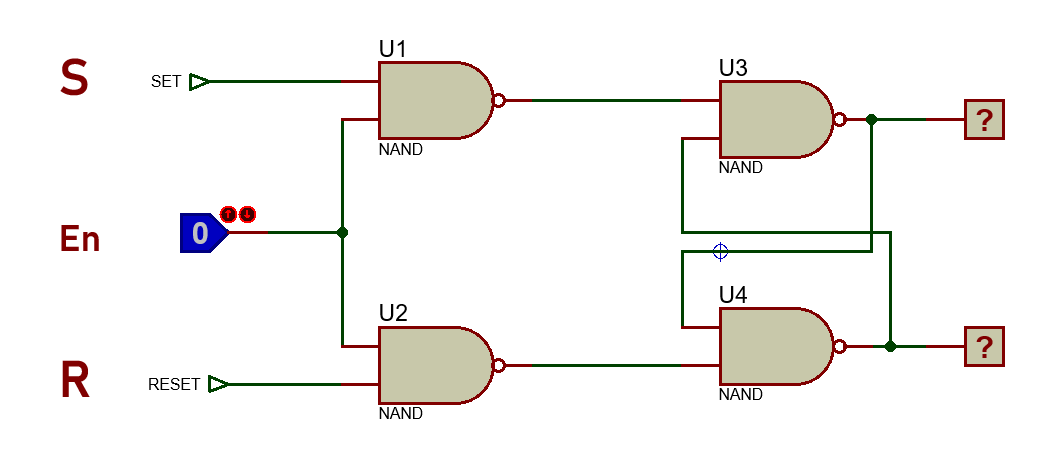
Active Low Latch (w/ NAND Gate)



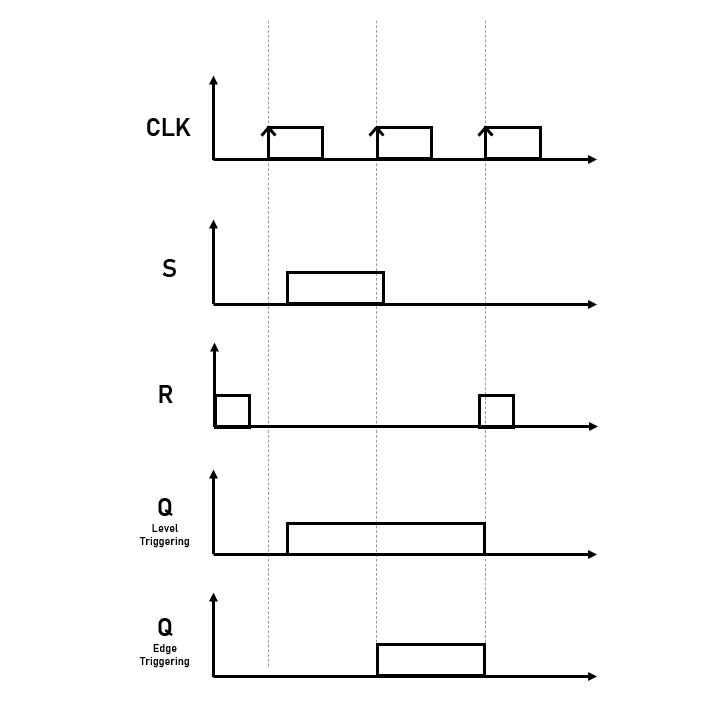
|  |  |  |  |
| --- | --- | --- | --- |
| **S’** | **R’** | **Q** | **Q’** |
| 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 |
| 0 | 0 | *invalid* | |

Here, S=0 means SET and R=0 means RESET.

And when S=1 and R=1, it will hold the previous memory. And we will get unpredictable results for when S=0 and R=0. So we label it’s output as invalid.

**Adding a Control Input to the S-R Latch**

Here we simply added another level in our previous latch. And the new input pin is the enable pin here. Which when turns 0, disables the new values for S and R pin. It only holds memory during that time. But when En=1, the circuit functions like the S-R latch.

**Clock Pulse**

Clock pulses are continuous, precisely spaced changes in voltage. And we use them in two ways.

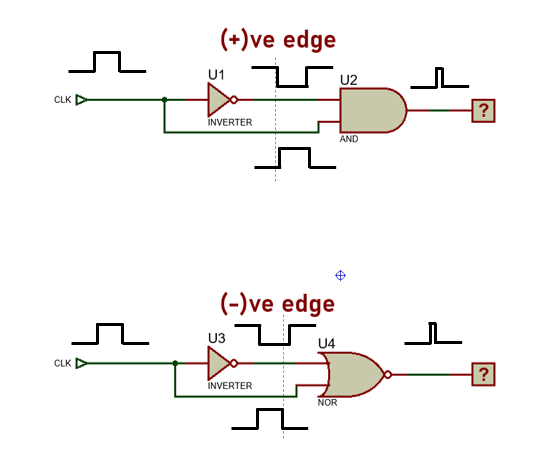
1. Level Triggering Clock Pulse
2. Edge Triggering Clock Pulse

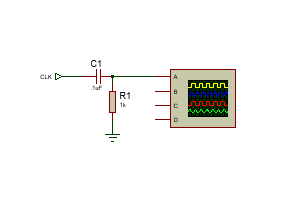
In edge triggering, we get outputs at the rising edge or falling edge.

In level triggering, we get outputs during the specific voltage state of the clock.

Added to that, it is important to mention that we are considering both the positive and negative edges of a clock pulse.

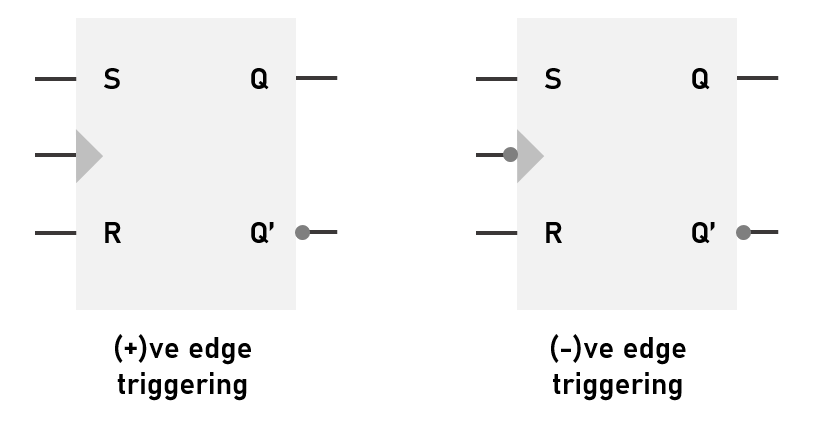
**Pulse Transition Detector**

This block will be able to find the rising and falling edges of a clock pulse. It is used in the edge triggering clock pulse modules. We require the following blocks for positive and negative edges respectively,

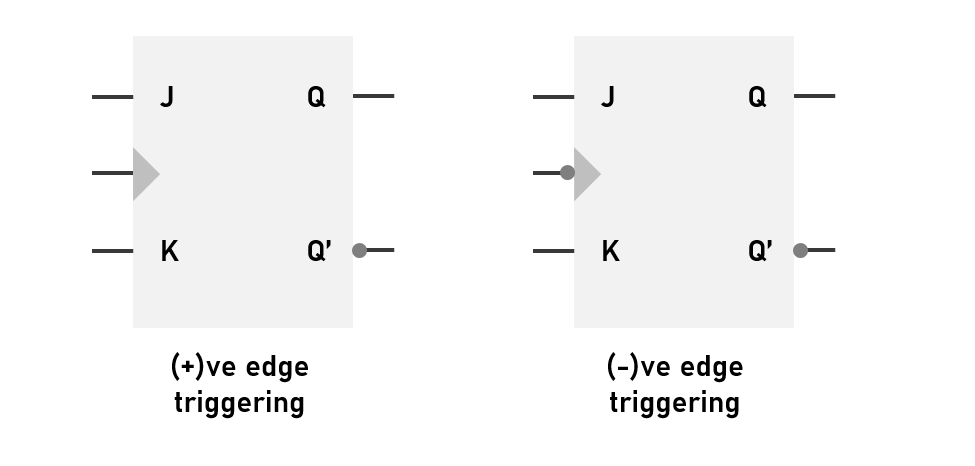
Here we had taken advantage of the delay through the NOT gate to get a specific trigger at the rising or falling edges.  
  
**Alternative**

We might also use a capacitor to get a trigger. Here at first when the pulse is turned on and it gets high, then the output current will flow in full. But at that time, the capacitor will start charging across the resistor, and as it does that we will get lesser current across the capacitor. Thus that will give us a spike or an edge. And we can control the transient time with

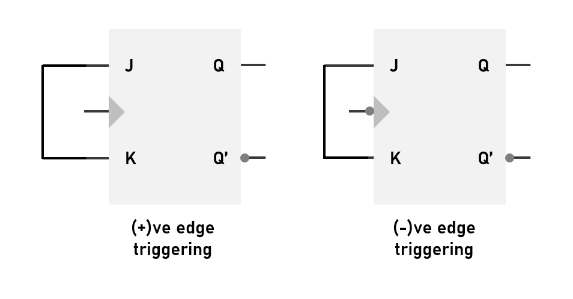
.

**S-R Flip Flop**

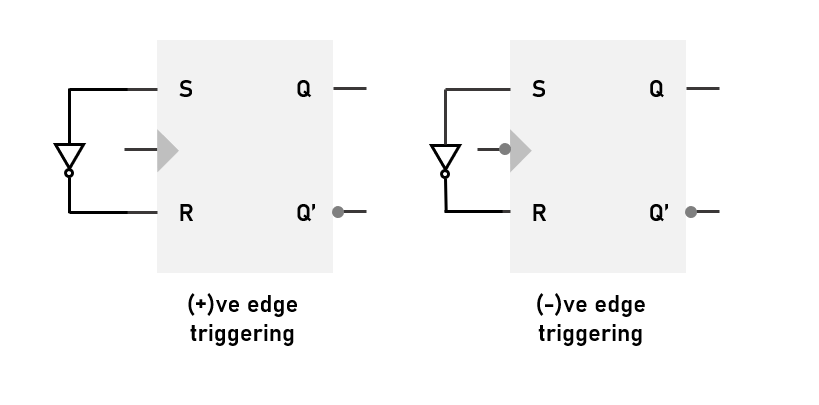
|  |  |  |  |
| --- | --- | --- | --- |
| **CP** | **S** | **R** | **Q** |
| 0 | x | x | Memory |
| ↑ | 0 | 0 | Memory |
| ↑ | 0 | 1 | Reset |
| ↑ | 1 | 0 | SET |
| ↑ | 1 | 1 | *invalid* |

**JK Flip Flop**

|  |  |  |  |
| --- | --- | --- | --- |
| **CP** | **S** | **R** | **Q** |
| 0 | X | X | Memory |
| ↑ | 0 | 0 | Memory |
| ↑ | 0 | 1 | Reset |
| ↑ | 1 | 0 | Set |
| ↑ | 1 | 1 | Toggle |

**D Flip Flop**

|  |  |  |
| --- | --- | --- |
| **CP** | **D** | **Q** |
| 0 | X | Memory |
| ↑ | 0 | 0 |
| ↑ | 1 | 1 |

****

**T Flip Flop**

|  |  |  |
| --- | --- | --- |
| **CP** | **T** | **Q** |
| 0 | X | Memory |
| ↑ | 0 | Memory |
| ↑ | 1 | Toggle |

**Preset & Clear inputs for Flip Flop**

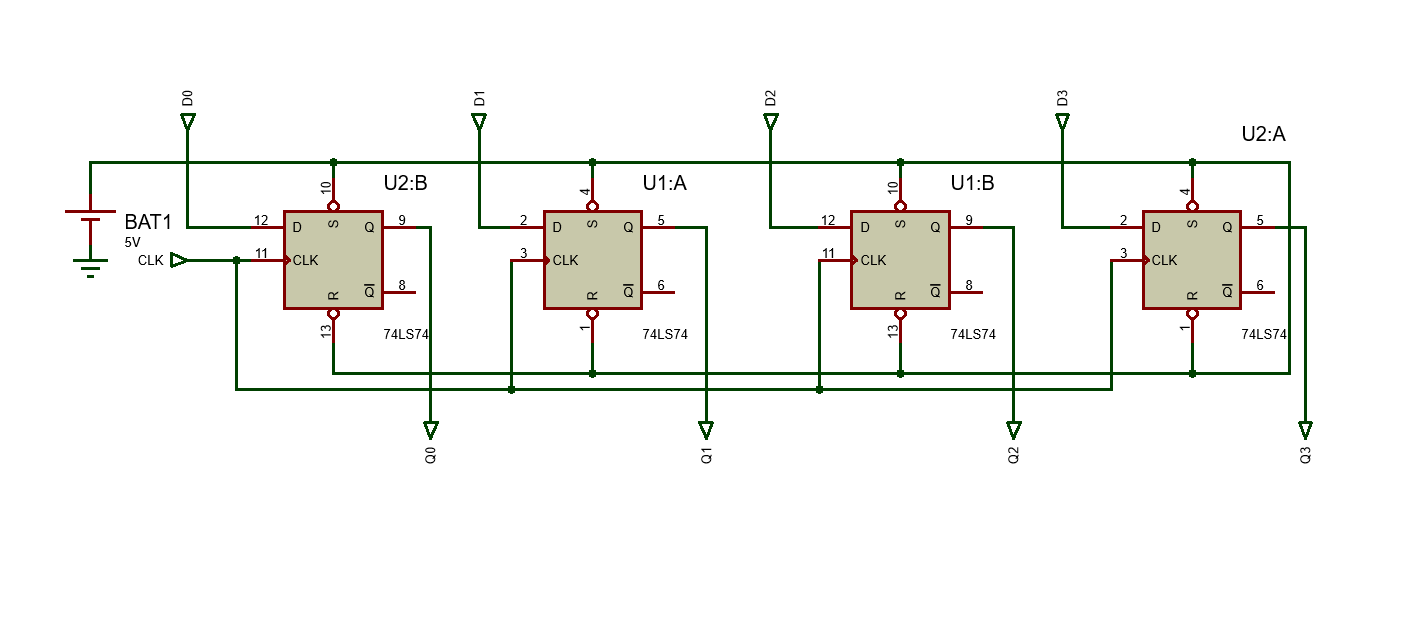
S & R and J & K inputs are called synchronous inputs. They are synchronous to the clock pulse. But the Preset and Clear inputs are not dependent on the clock pulse. Rather when preset is activated then the Flip Flop goes in SET mode and when Clean is activated then the Flip Flop goes in RESET mode. And it is to be noted that these two pins are active low pins.

**Register and bus architecture**

**Register**

A **register** is a collection of flip flops. A flip flop is used to store single bit digital data. For storing a large number of bits, the storage capacity is increased by grouping more than one flip flops.

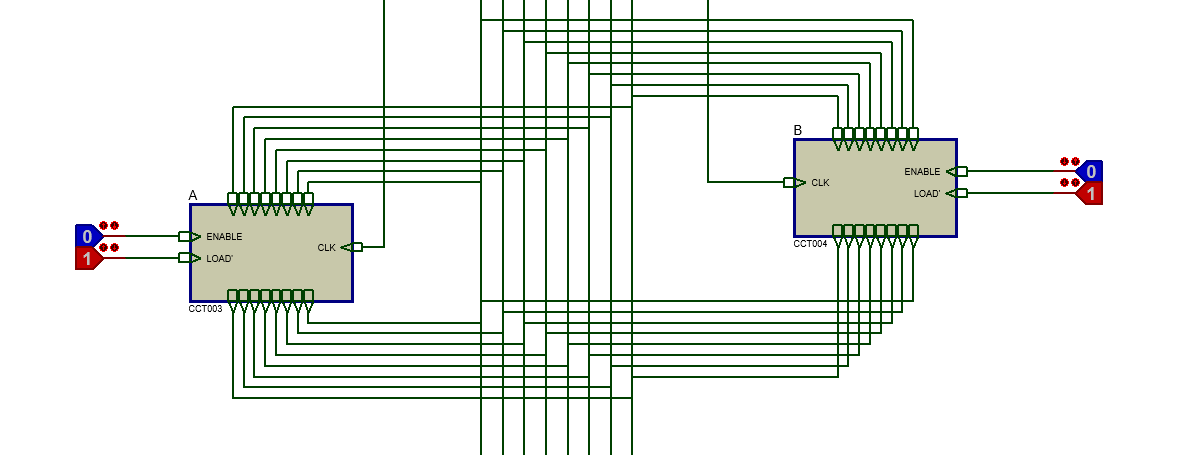
The register is used to perform different types of operations. For performing the operations, the CPU use these registers. The faded inputs to the system will store into the registers. The result returned by the system will store in the registers. There are the following operations which are performed by the registers.

A D Flip Flop is an example of a 1-bit data storage device. So, a single D flip flop can act like a one-bit register. Thus, if we want to store an n-bit word, we have to use an n-bit register containing n number of flip flops. There are various categories of registers. In the SAP-01 project, only the “Parallel In-Parallel Out” registers have been used.

The above figure is a diagram of a 4-bit parallel in-parallel out register designed using 4 D Flip Flops. Here each of the flip flops are connected to separate D inputs and four separate outputs get out from the flip flops. Thus, as the inputs and outputs are inserted and found at the exact same time, it is called parallel in parallel out register. And in each of these blocks the same clock pulse is fed. And it is either positive or negative edge triggered, depending on the design and need of the register. And the values from these registers can come or go from or into the bus on each edge trigger of the clock input. And here in this particular case data will go from D to Q on each positive edge trigger of the clock. And if we want to create an infrastructure which sends or receives some sort of data of particular bits, then we can make up our bus based on our need.

**Bus Architecture**

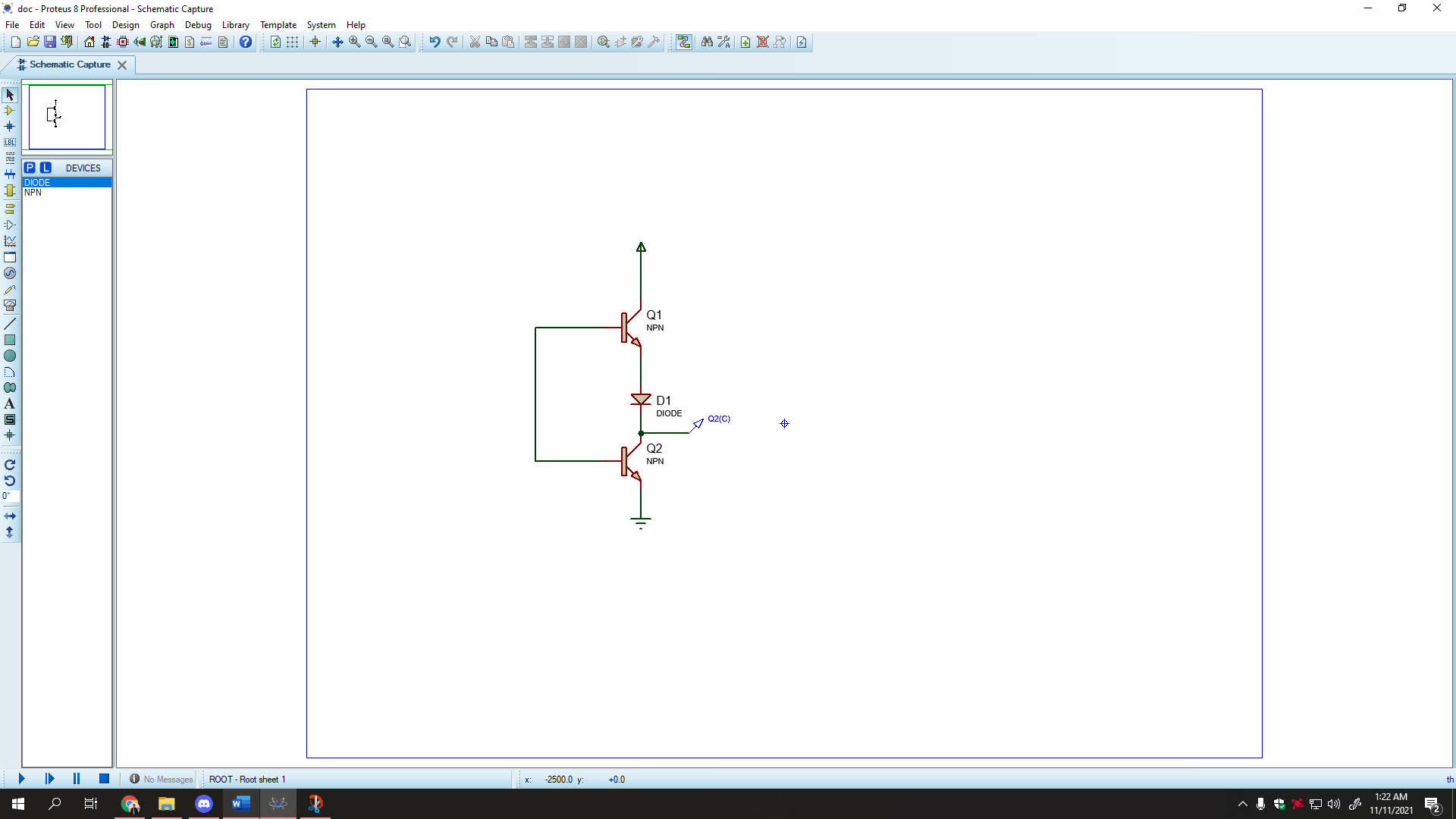
Bus is an abbreviation of omnibus.  In the English bus is the abbreviation of "for all" and here in the modern use, the "omnis" part is not in use anymore and the "bus" part is kept which originally referred to "for". Nowadays in computer science a bus is usually a group of signal lines connecting multiple devices.



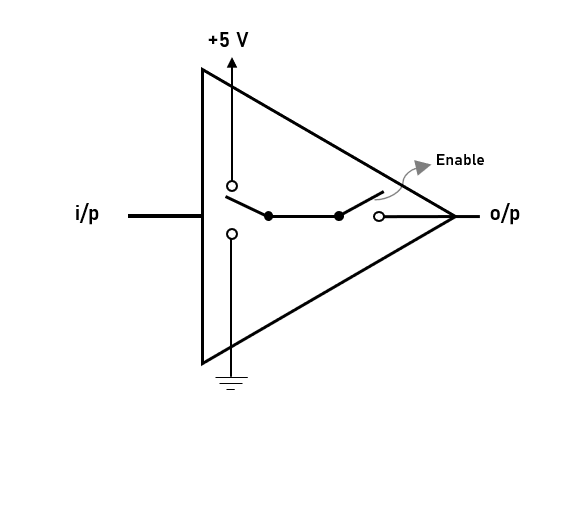
Here the above connection is a demonstration of the 8-bit bus. Here, each of the output and input connections of the modules are connected to the bus. But whether or not a module would input a value depends on the LOAD’ pin. And usually for the SAP project the load pins of the registers are active low. So when the input of the LOAD’ goes from 1 to 0, then the module will take inputs from the bus. And it will only give output when the ENABLE pin is activated. And usually ENABLE pins are active high. And if we try to send two data to the same bus at the same time, it would be Bus contention. In the proteus simulation we will get errors referring to logic contentions.

**Bus Contention**

So, when we are going to connect the bus pins, then we need to make sure that only one of the modules are outputting values at the same time. The error or the problem that happens when two modules try to send values to the bus at the same time is called Bus Contention. This is an undesirable state for the system.

We need to really understand what does a logic 1 and 0 mean in a SAP module. In simple words, Logic State 1 is a representation of the 5 V and on the other hand the Logic State 0 is a representation of the GND. And it can be better understood using the source and the sink analogy.

Here when the Q1 is turned ON, the 5 V coming in from the source directly goes through the transistor and through the output taken across the voltage probe. And then we could’ve said that the gate is sourcing current. And inversely when the Q2 transistor is turned on the current goes to the ground from the o/p. And then it’s kind of sinking the current. So, for the busses, the 0s indicate that the pin is going to sink current and the 1s are sourcing current.

**Tri state Buffers**

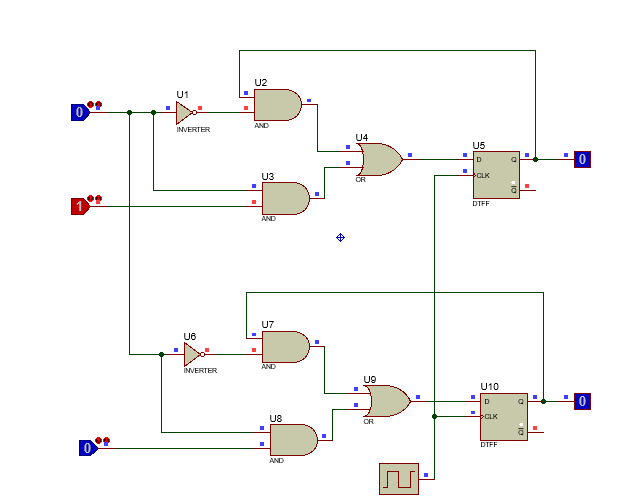
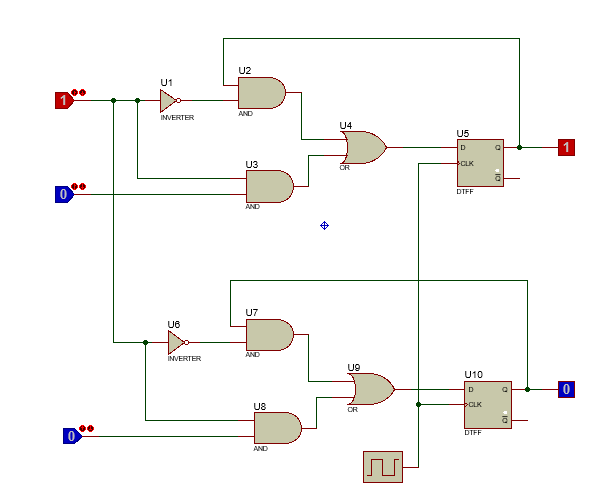
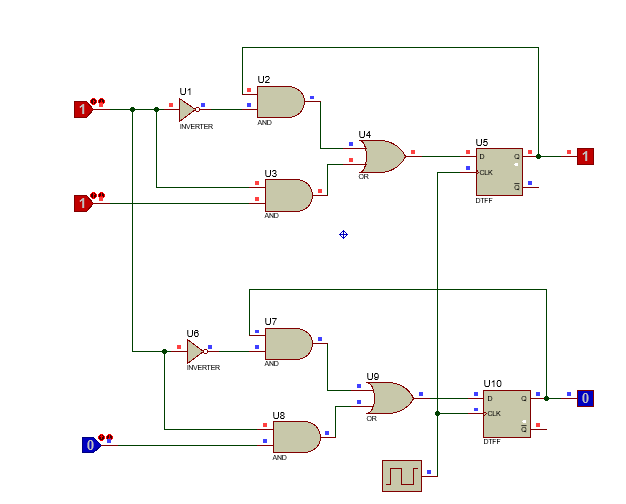
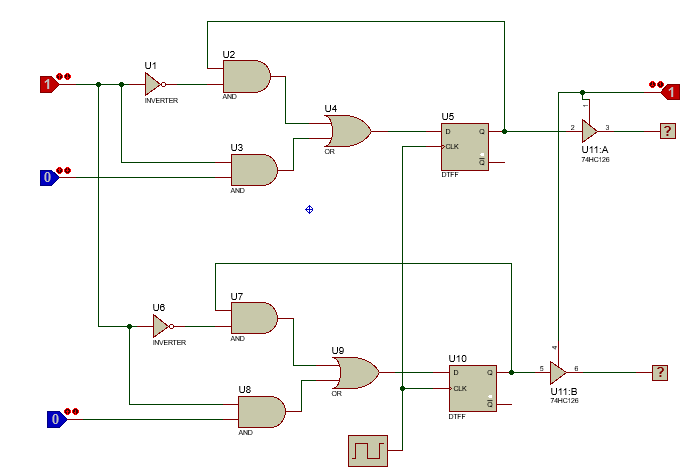
A tri-state buffer is a digital buffer circuit whose output can be electronically disconnected from the bus when required. Here these types of buffers are generally used in the SAP-01 project. Using these buffers, we can ignore the bus contention problem.

**Building an 8-bit register**

**Goal:** To load and output 8 bit data from/to the bus upon given instruction avoiding bus contention

Our initial thought would be to get 8 D flip flops and then upon each clock transition we would be able to input and output data but that would leave us with the door of facing bus contention.

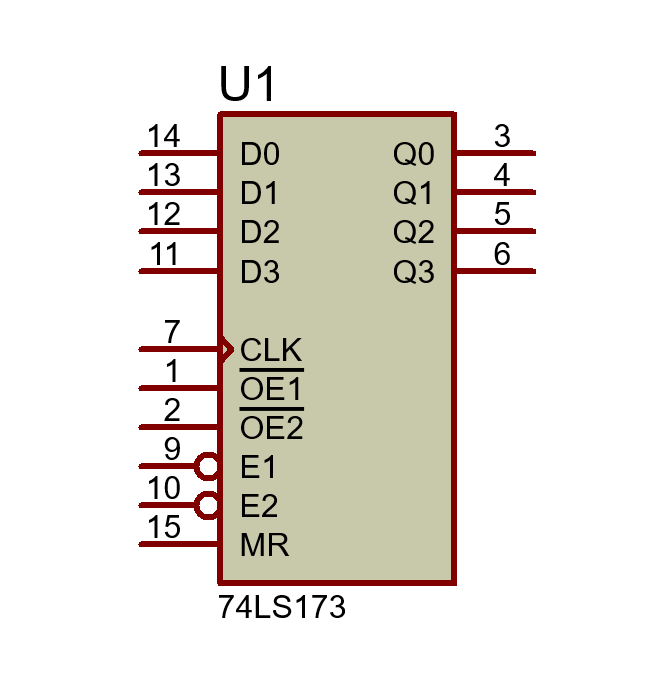
So, our first step around our initial thought is to somehow make a combinational circuit design which would load data upon given instruction to do so.

Here our combinational circuit before the D flip flops was integrating the Load pins so that when Load pin is enabled then we can get subsequent values present from the D inputs of the Flip Flops. And then when the load is turned off, we can see from the logic probes that the values are stored after the clock pulse transition. Thus, we understand that the circuit is now able to store values and we have thus designed our register. But our goal is yet to be fulfilled. We still need to electrically disconnect the outputs so that it doesn’t interfere with the values that is in the bus. And we must avoid bus contention in this case. Thus, we used the tri-state buffers at each of the outputs of the Flip Flops.

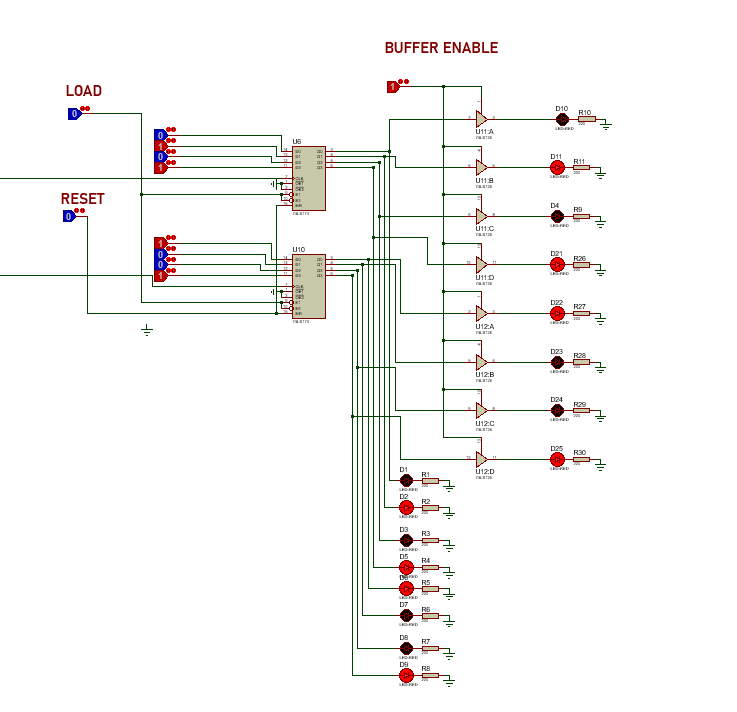
Now, that we know how the basic register is built, we can now move on to the ICs that were used in the registers of the SAP-01.

**74LS1731**

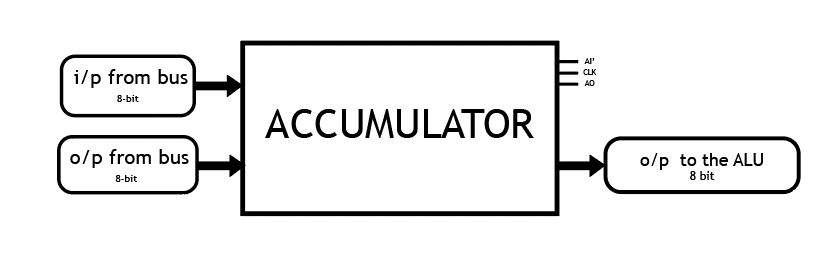
**PINS**



|  |  |  |  |
| --- | --- | --- | --- |
| **Pin No.** | **Name of Pin** | **Active Enable** | **Function** |
| 11,12,13,14 | D0-D3 | High | i/p pins |
| 3,4,5,6 | Q0-Q3 | High | o/p pins |
| 7 | CLK | + ve Edge | Clock Pulse |
| 1,2 | OE1’/OE2’ | Low | o/p enable pin |
| 9,10 | E1/E2 | High | Load pin |
| 15 | MR | High | Master Reset |

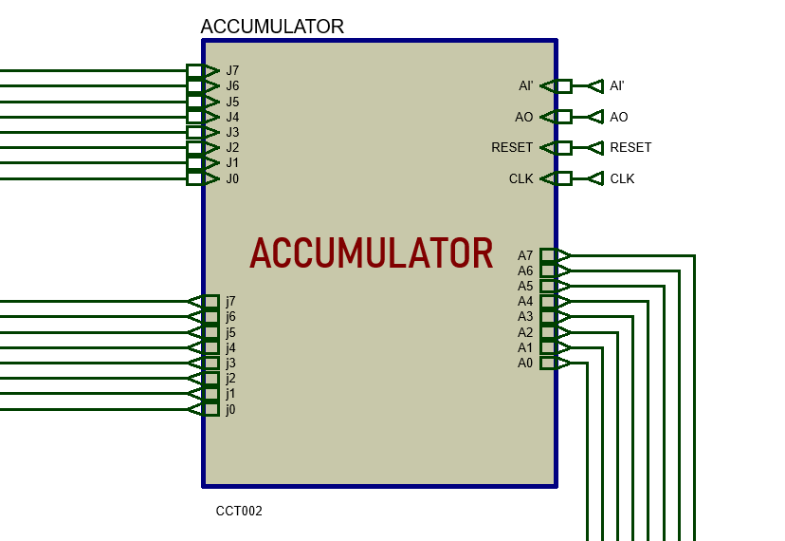
Here we used two 74LS173 IC which is basically used as an 8-bit register. Which has a load pin and reset pin which can be manipulated according to the user’s demand. And we had used 8 tri-state buffers with the outputs of the registers. Where we had grounded the 1 and 2 pins i.e. the enable pins of the register. And our approach is going to be limiting the output values from going to the bus by using 8 tri-state buffers for 8 outputs. We had connected LEDs as required with pull up resistors in order to see if our register is working.

**Accumulator**

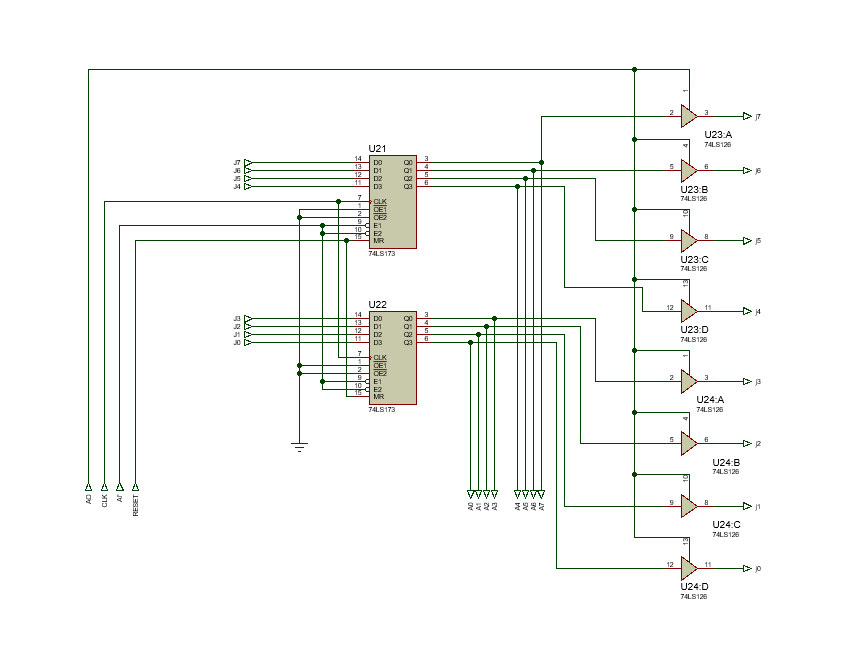
**Block Diagram**

**Specifications:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **IC** | **Input pins** | **O/P pins (bus)** | **O/P pins (ALU)** | **Control Pins** |
| **74LS173**  4-BIT D-TYPE REGISTERS | J7(MSB)-J0(LSB) | j7(MSB)-j0(LSB) | A7(MSB)-A0(LSB) | AI’, AO, CLK, RESET |

****

**Algorithm**

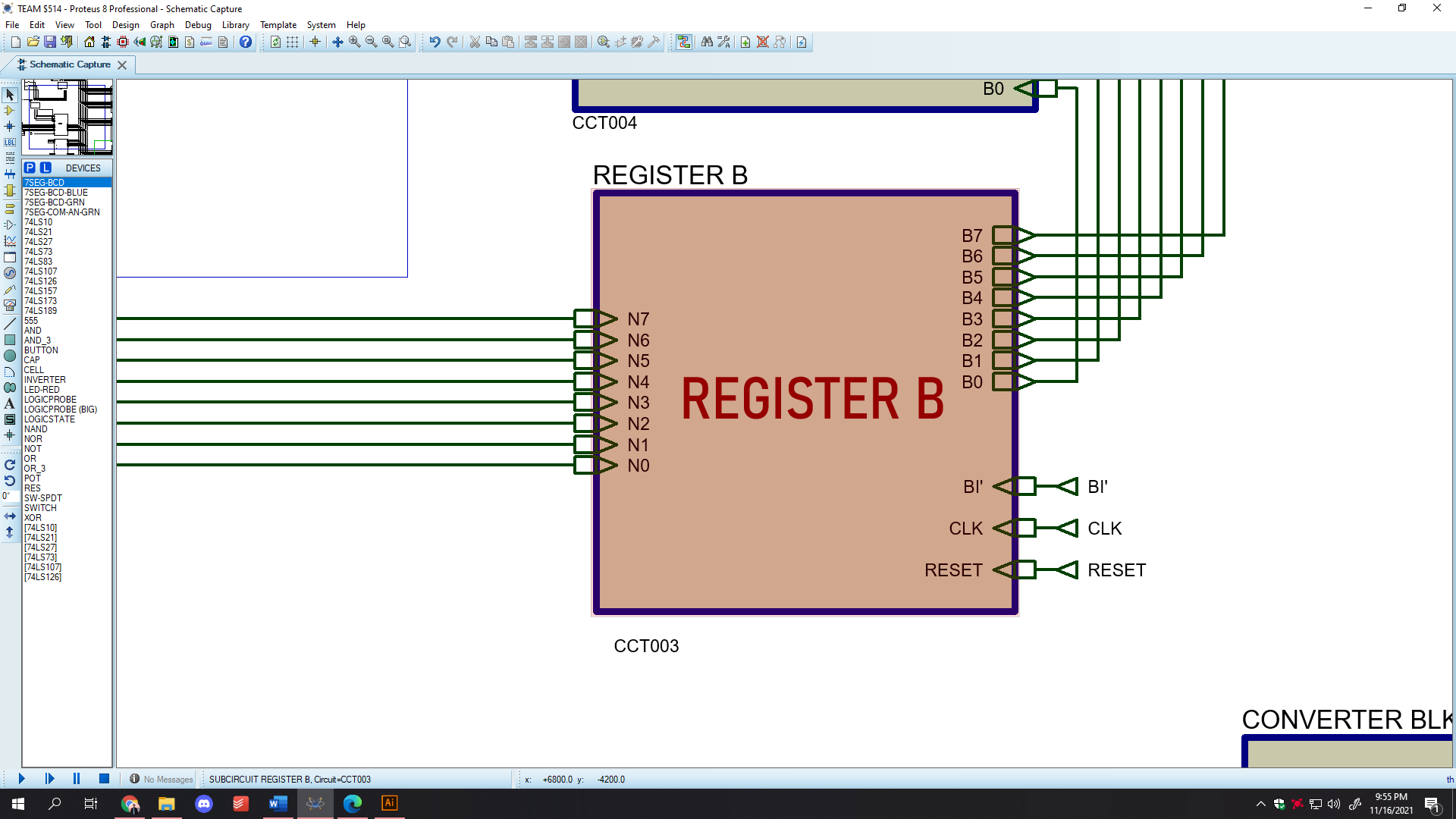
1. Here we have used two 74LS173 4-bit D type register.
2. We have used the 11,12,13,14 pins of the register as the inputs of the Accumulator.
3. And then the 7-no. pin is the Clock Pulse input of the registers. So, we short them together and then we take them as the inputs of the accumulator clock pulse coming in from our “Mode Switching”.
4. The no. 1 and 2 pins are active low. So, we make sure to keep them grounded so that the registers always store input value and gives output enabled.
5. 9 and 10 are the “Load” pins. Here, these pins are active low. So, when value transitions from 1 to 0, then the inputs pins are enabled to receive values from the bus.
6. 15 is the “RESET” pin. We short the 15 pins of both registers.
7. 3,4,5,6 are the output pins. And the direct values of these pins go to the ALU.
8. ****The values which come out of 3,4,5,6 goes through a tri-state buffer which has a singular input to enable/disable these values. So, this will help us, integrate the concept tri-state buffer in our accumulator to save us from Bus contention.

**Register B**

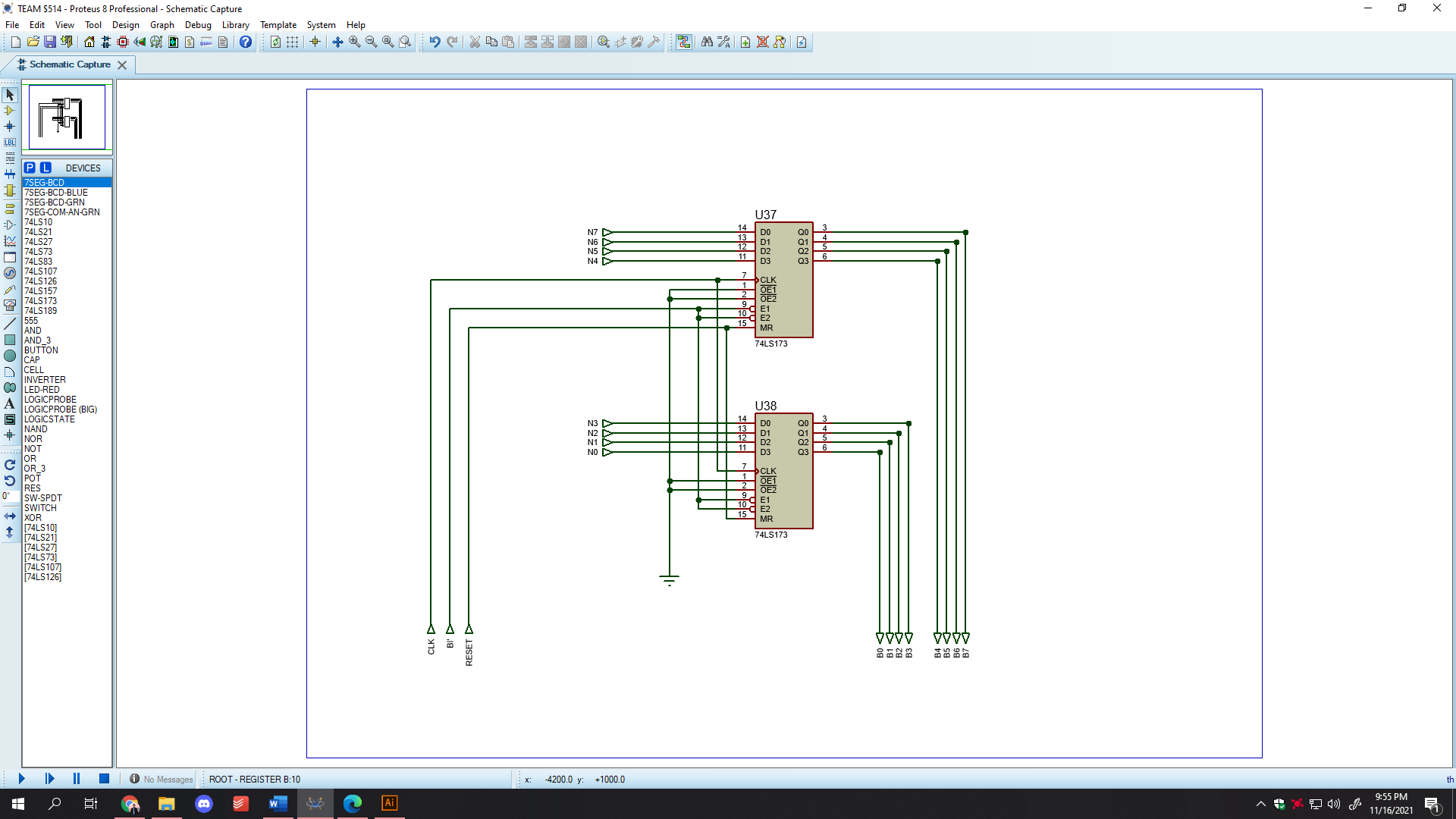
Block Diagram

**Specifications:**

|  |  |  |  |
| --- | --- | --- | --- |
| **IC** | **Input pins** | **O/P pins (ALU)** | **Control Pins** |
| **74LS173**  4-BIT D-TYPE REGISTERS | N7(MSB)-N0(LSB) | B7(MSB)-B0(LSB) | BI’, CLK, RESET |

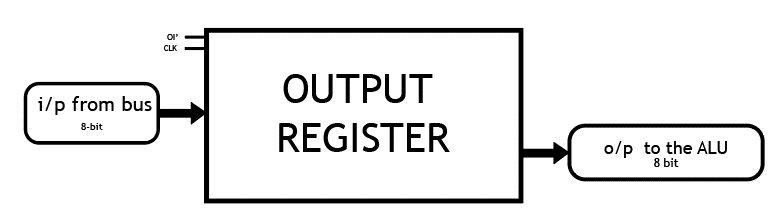
****

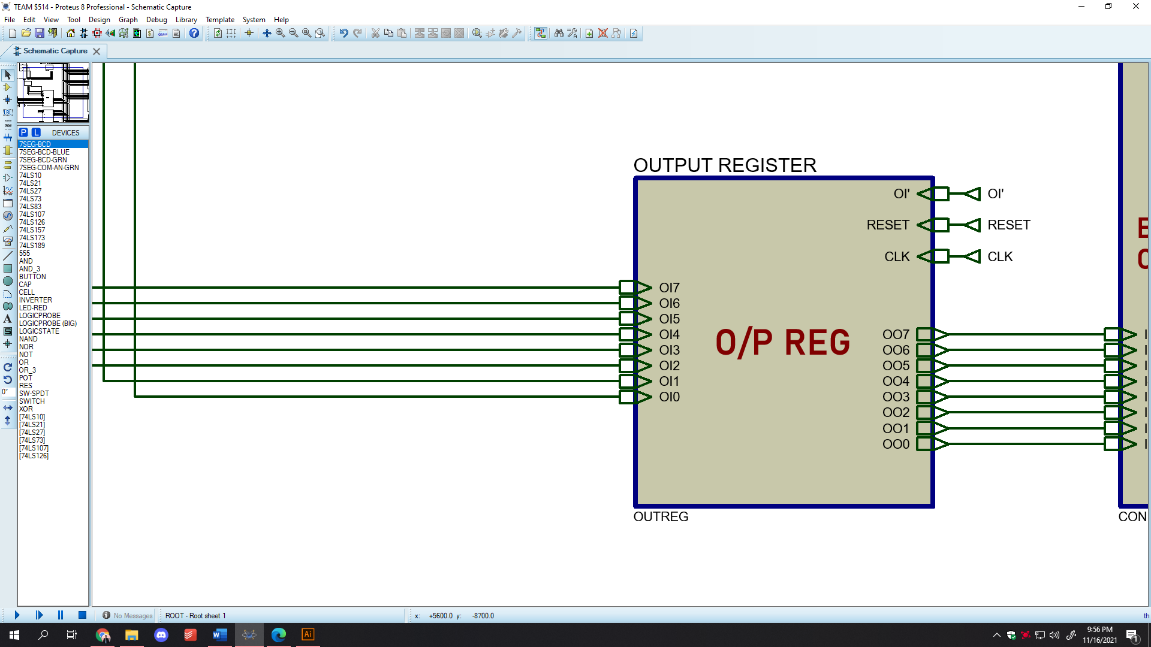
**Algorithm**

1. Here we have used two 74LS173 4-bit D type register.
2. We have used the 11,12,13,14 pins of the register as the inputs of the Register B.
3. And then the 7-no. pin is the Clock Pulse input of the registers. So, we short them together and then we take them as the inputs of the accumulator clock pulse coming in from our “Mode Switching”.
4. The no. 1 and 2 pins are active low. So, we make sure to keep them grounded so that the registers always store input value and gives output enabled.
5. 9 and 10 are the “Load” pins. Here, these pins are active low. So, when value transitions from 1 to 0, then the inputs pins are enabled to receive values from the bus.
6. 15 is the “RESET” pin. We short the 15 pins of both registers.
7. 3,4,5,6 are the output pins. And the direct values of these pins go to the ALU.

**Output Register**

**Block Diagram**

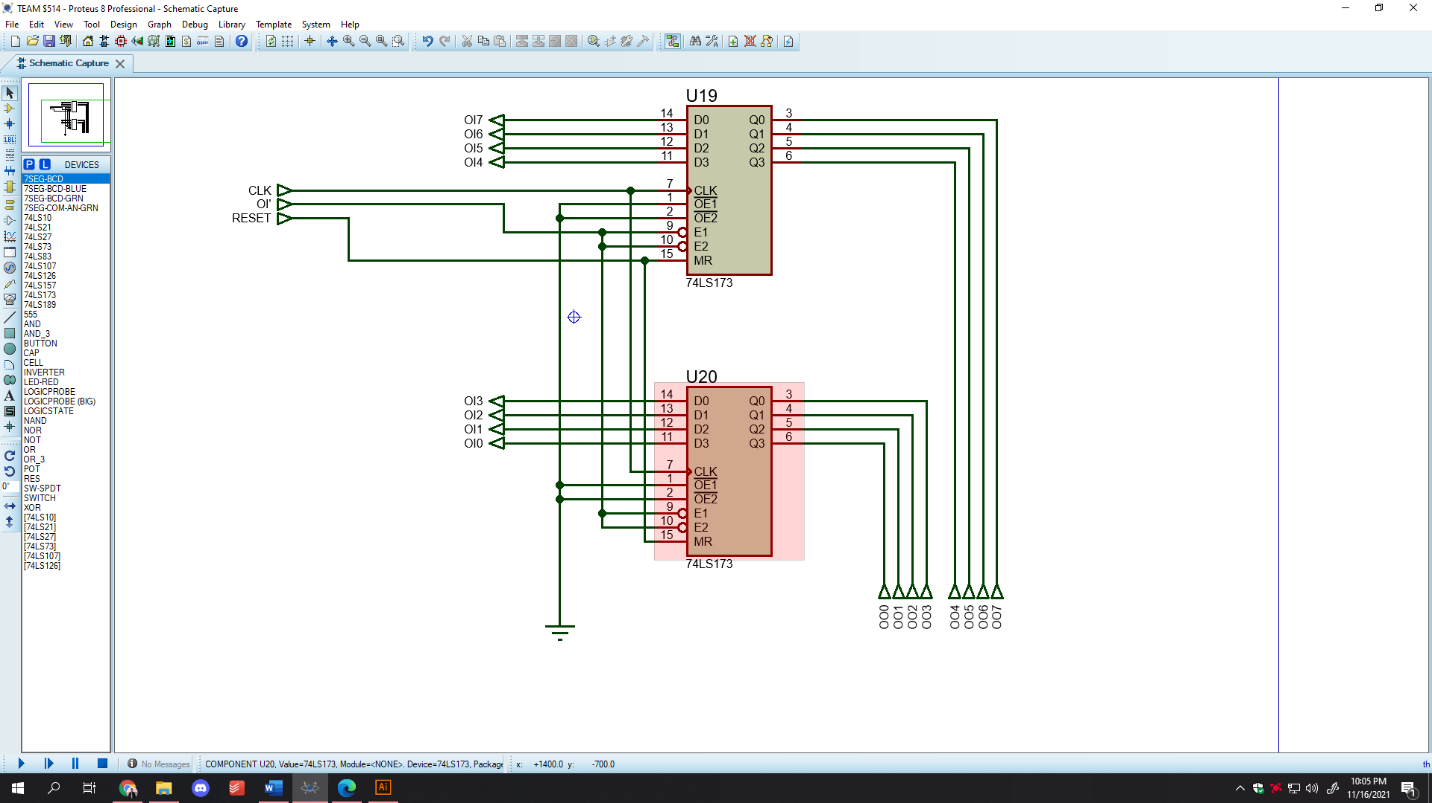
****

**

**Specifications:**

|  |  |  |  |
| --- | --- | --- | --- |
| **IC** | **Input pins** | **O/P pins (ALU)** | **Control Pins** |
| **74LS173**  4-BIT D-TYPE REGISTERS | OI7(MSB)-OI0(LSB) | OO7(MSB)-OO0(LSB) | OI’, CLK, RESET |

*Algorithm of output register is the same as Register B*

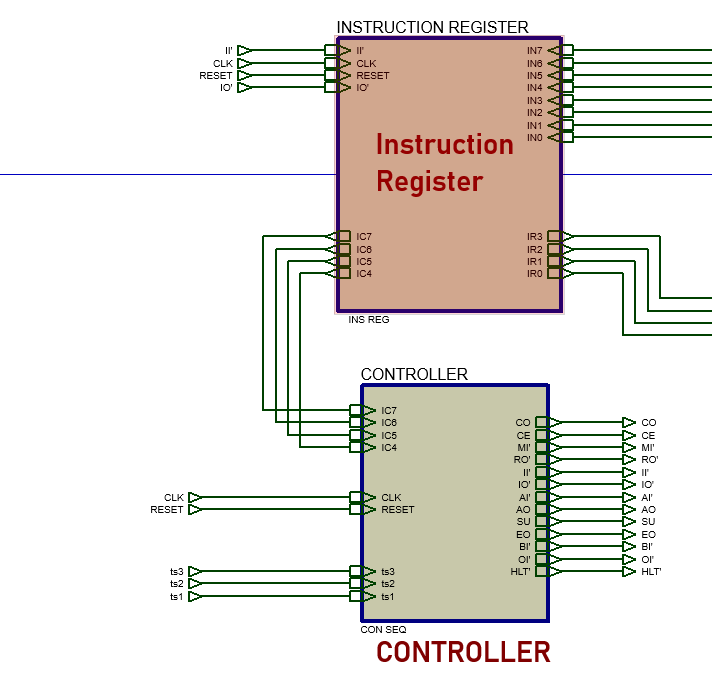


**Instruction Register**

**Block Diagram**

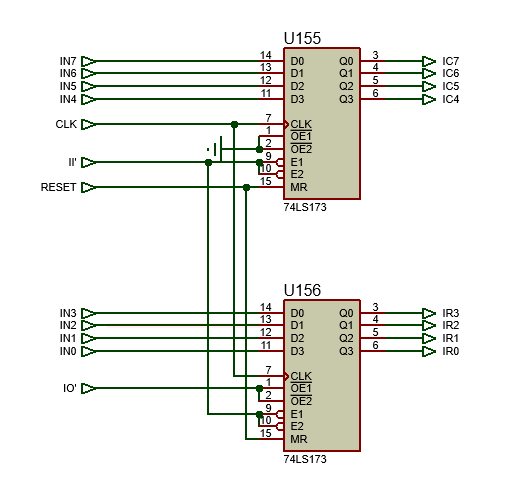
**Specifications:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **IC** | **Input pins** | **O/P pins (bus)** | **O/P pins (Sequencer)** | **Control Pins** |
| **74LS173**  4-BIT D-TYPE REGISTERS | IN7(MSB)-IN0(LSB) | IR3(MSB)-IR0(LSB) | IC7(MSB)-IC4 (LSB) | II’, IO’, CLK, RESET |

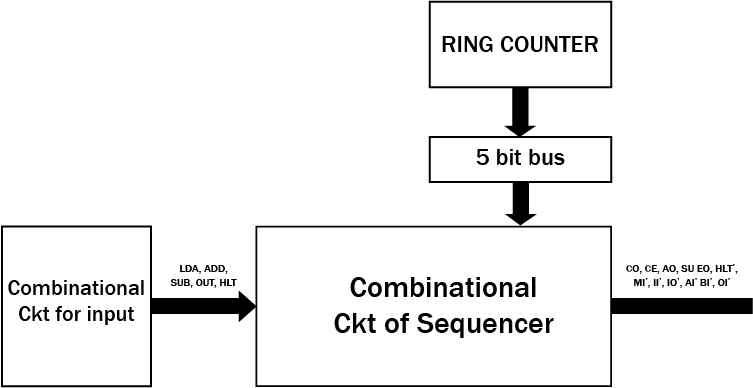
****

**Algorithm**

1. Here we have used two 74LS173 4-bit D type register.
2. We have used the 11,12,13,14 pins of the register as the inputs of the Instruction Register.
3. And then the 7-no. pin is the Clock Pulse input of the registers. So, we short them together and then we take them as the inputs of the accumulator clock pulse coming in from our “Mode Switching”.
4. The no. 1 and 2 pins are active low. Here, the register which will take the upper nibble will have the pins 1 and 2 grounded always. And the register which will take the lower nibble will have an enable option through these pins as IO’ input.
5. 9 and 10 are the “Load” pins. Here, these pins are active low. So, when value transitions from 1 to 0, then the inputs pins are enabled to receive values from the bus.
6. 15 is the “RESET” pin. We short the 15 pins of both registers.
7. 3,4,5,6 are the output pins. The register which will take the upper nibble will have the pins 1 and 2 grounded always. And the register which will take the lower nibble will have an enable option through these pins as IO’ input.



**Controller/Sequencer**

**Block Diagram**

**Purpose**

Sequence control refers to user actions and computer logic that initiate or stop sequence. Sequencer controls the SAP and the values that goes in and out of the bus at each transition state. Methods of sequence control require explicit attention in interface design, and many published guidelines deal with this topic.

**Specifications**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Input pins** | **Output pins** | | **Control Pins** | **Ring Counter** | **T State** |
| **Active low** | **Active**  **High** |
| IC7-IC4 | CO, CE, AO, SU EO | HLT’, MI’, II’, IO’, AI’ BI’, OI’ | CLK, RESET | Jump to “Ring Counter” | 5 |

**Programming SAP-1**

**Op-code and input combinational circuit design**

LDA 0000

ADD 0001

SUB 0010

OUT 0011

HLT 1111

So, our idea is to use create a combinational circuit where each significant bits are designed in accordance to whether it is 0 or 1. For example, to get LDA activated we need to make sure that when inputs pins get 0 then LDA=1. So, to ensure this, we get a 4 input AND gate and use the inverted connection from the input pins. So, during this only the AND gate responsible for the LDA activation will be ON and the rest will be OFF. But when we have 1 in the input pins such as in ADD we have 1 in the LSB. So, we connect it directly and then connect it to the AND gate responsible for the ADD activation. Thus, we connect all the other combinational circuits in this manner.

**Cycles and Instructions in SAP-1**

The address and data switches allow you to program SAP-1. Here the op code will go to the upper nibble and the lower nibble is responsible for the operand. For example,

|  |  |
| --- | --- |
| **Address** | **Instruction** |
| 0H | LDA FH |
| 1H | ADD EH |
| 2H | HLT |

1. Here at first convert each of these into binaries. For example, LDA will translate to 0000. And FH will translate to 1111. So LDA FH means 0000 1111.
2. Then 1H will translate to 0001 1110. And HLT to 1111 XXXX. And what these binary codes will translate to will be the core in understanding how the controller works.

There are generally **2 cycles** in our SAP-1.

1. Fetch Cycle
2. Execution Cycle

**Fetch Cycle**

Control unit generates the control words that fetch and execute each instruction and while this happens the computer will pass through various timing states(T STATES).

**Address State (T1)**

This is where the address of the program counter is transferred to the MAR. During this state CO and MI’ are activated.

**Increment and Memory State (T2)**

This will increase a value in the program counter. Simultaneously, a value will be transferred from RAM to the instruction register

**Execution Cycle**

**LDA**

If LDA 9H comes then Instruction Register will receive 0000 1001. And the 0000 will go as input to the sequencer and then the lower nibble will act as operand. And after the T3 cycle, the AI’ and RO’ will be activated. So the 1001 value will go to the MAR and act as address to the RAM. And RAM will subsequently output whatever value is in that specific address. And accumulator will take this value and temporarily store it.

**Summary:**

|  |  |
| --- | --- |
| **T3** | **IO’**  **MI’** |
| **T4** | **AI’**  **RO’** |

ADD, SUB and OUT will follow a similar route as the LDA. Only in their cases slight modifications are required as per the op-code.

**ADD**

|  |  |
| --- | --- |
| **T3** | **IO’**  **MI’** |
| **T4** | **BI’**  **RO’** |
| **T5** | **EO**  **AI’** |

**SUB**

|  |  |
| --- | --- |
| **T3** | **IO’**  **MI’** |
| **T4** | **BI’**  **RO’** |
| **T5** | **EO**  **AI’**  **SU** |

**OUT**

|  |  |
| --- | --- |
| **T3** | **OI’**  **AO** |

**HLT**

When the IR will get 1111 XXXX then HLT’ will get activated. It is done so that when 1111 is input in the sequencer then the NAND gate will give 0. And this 0 will go to the AND gate that is present in the Main CLOCK. And when 0 is an input in an AND gate, the output is automatically 0. Thus, we stop the clock and the entire SAP immediately.

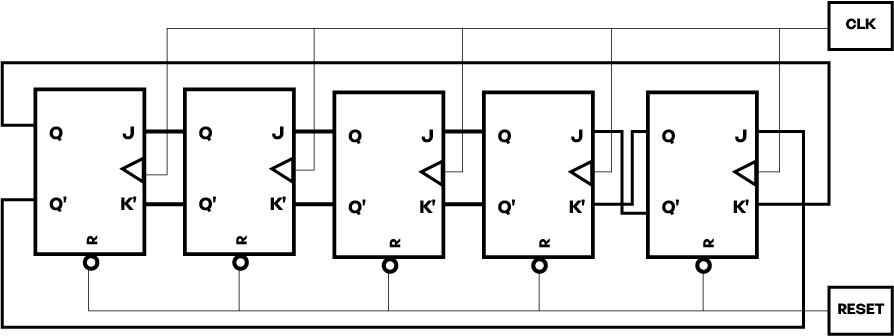
**Microinstructions**

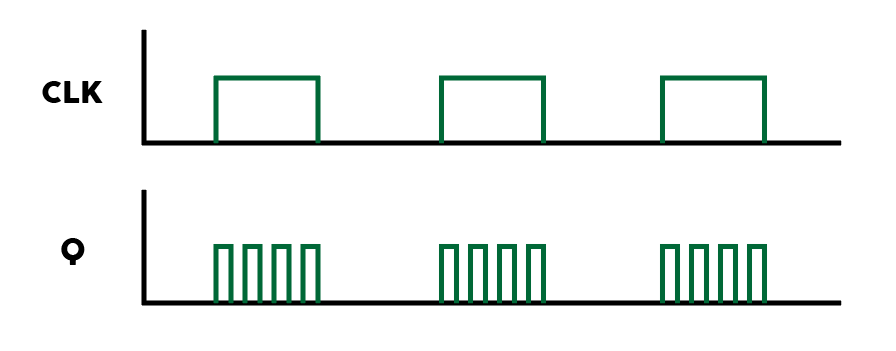
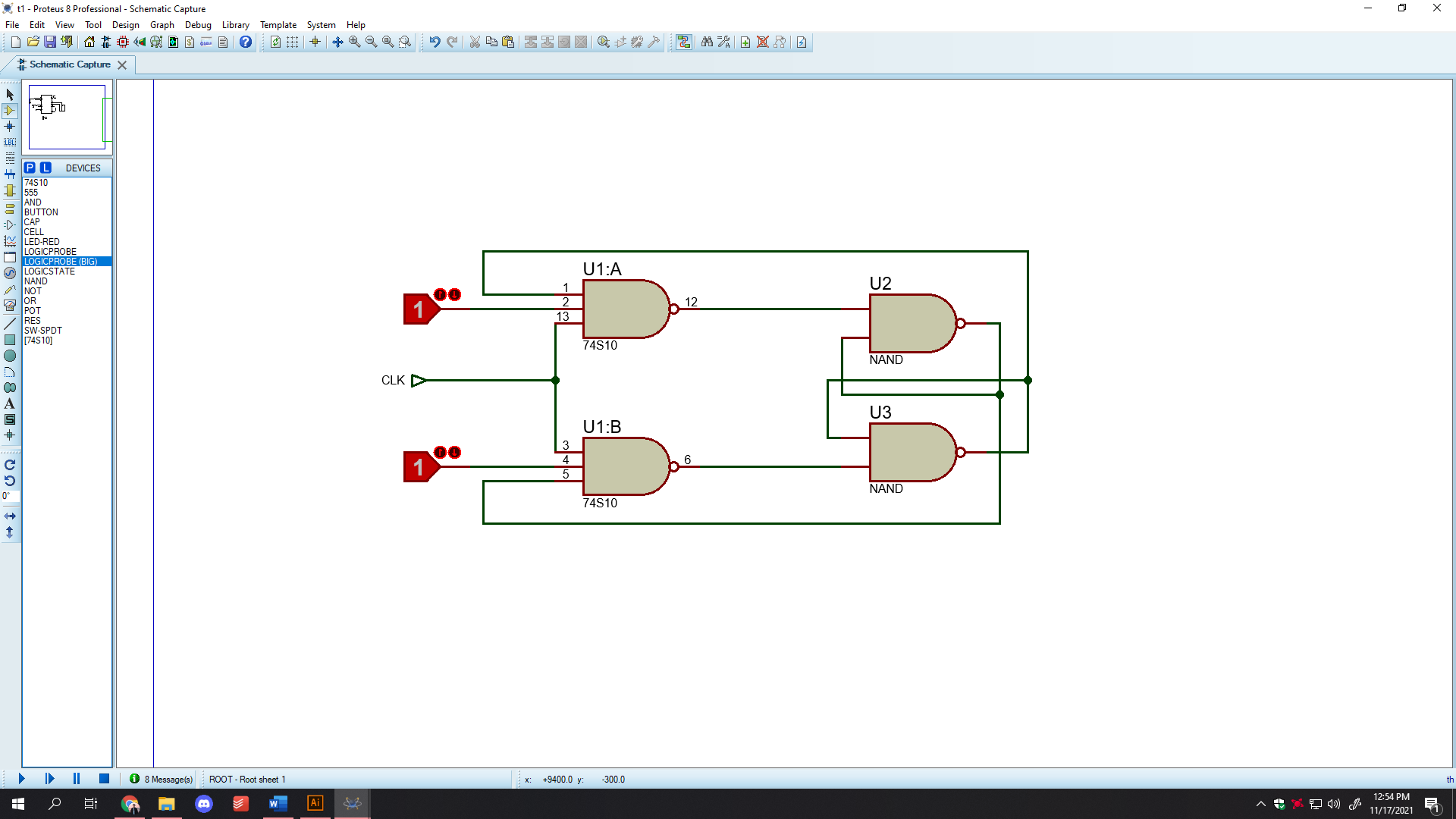
|  |  |  |
| --- | --- | --- |
| **Macro** | **State** | **OUTPUT STATE**  **CO CE MI’ RO’ II’ IO’ AI’ AO EO SU BI’ OI’** |
| LDA | T3 | 0 0 0 1 1 0 1 0 0 0 1 1 |
| T4 | 0 0 1 0 1 1 0 0 0 0 1 1 |
| ADD | T3 | 0 0 0 1 1 0 1 0 0 0 1 1 |
| T4 | 0 0 1 0 1 1 1 0 0 0 0 1 |
| T5 | 0 0 1 1 1 1 0 0 1 0 1 1 |
| SUB | T3 | 0 0 0 1 1 0 1 0 0 0 1 1 |
| T4 | 0 0 1 0 1 1 1 0 0 0 0 1 |
| T5 | 0 0 1 1 1 1 0 0 1 1 1 1 |
| OUT | T3 | 0 0 1 1 1 1 1 1 0 0 0 0 |

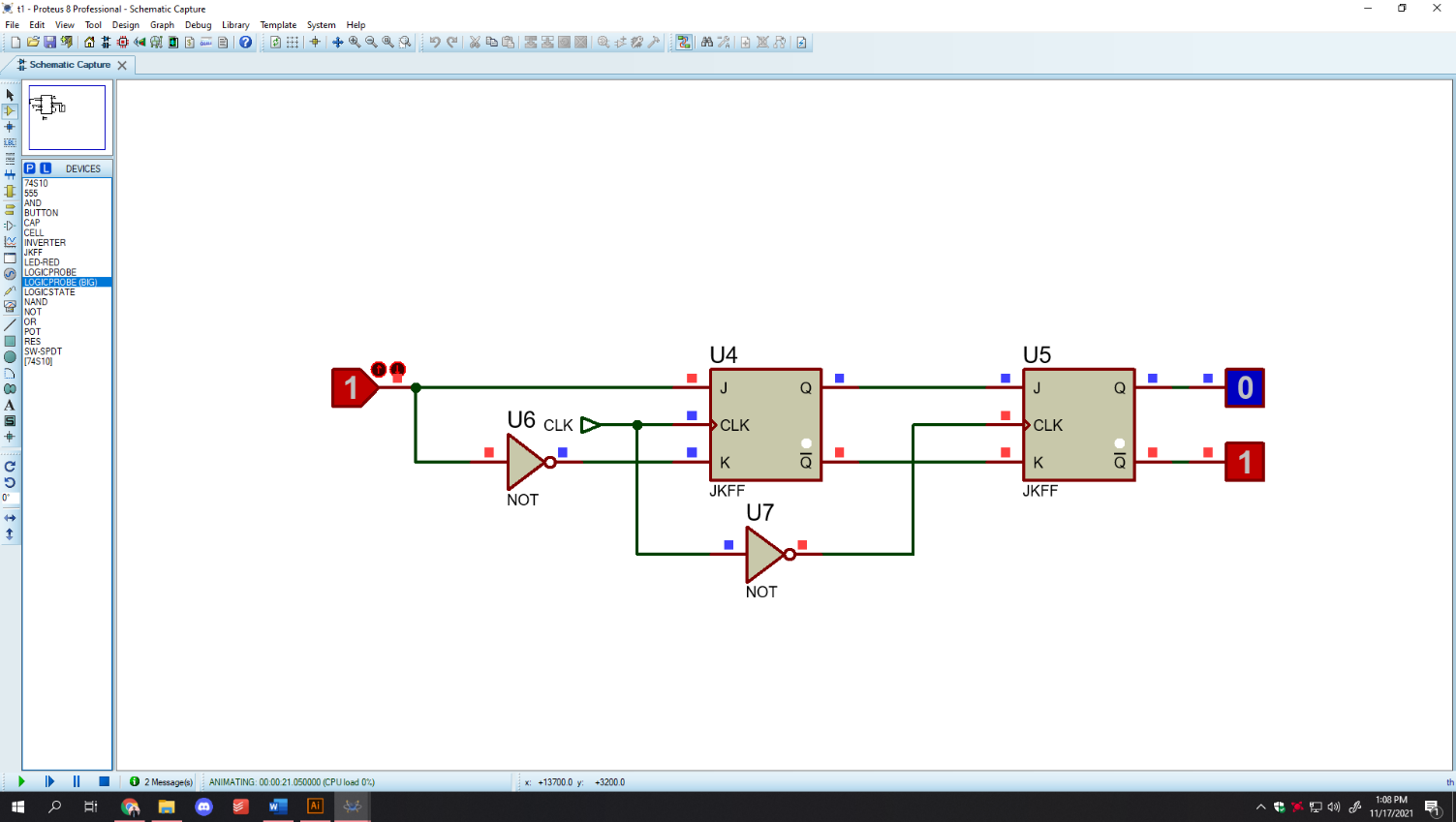
**Ring Counter**

Ring counter is a typical application of Shift resister. Ring counter is almost same as the shift counter. The only change is that the output of the last flip-flop is connected to the input of the first flip-flop in case of ring counter but in case of shift resister it is taken as output. Except this all the other things are same.

<https://www.geeksforgeeks.org/ring-counter-in-digital-logic/>

**Block Diagram**

******Master-Slave JK Flip Flop**

In the Level triggering mode of the clock pulse, we see that the value of Q keeps toggling during the clock pulse period. And this creates a phenomenon called “Race around Condition”. And now in order to go past this phenomenon, we need to understand the concept of Master Slave JK Flip Flop which will be critical in understanding the Ring Counter.

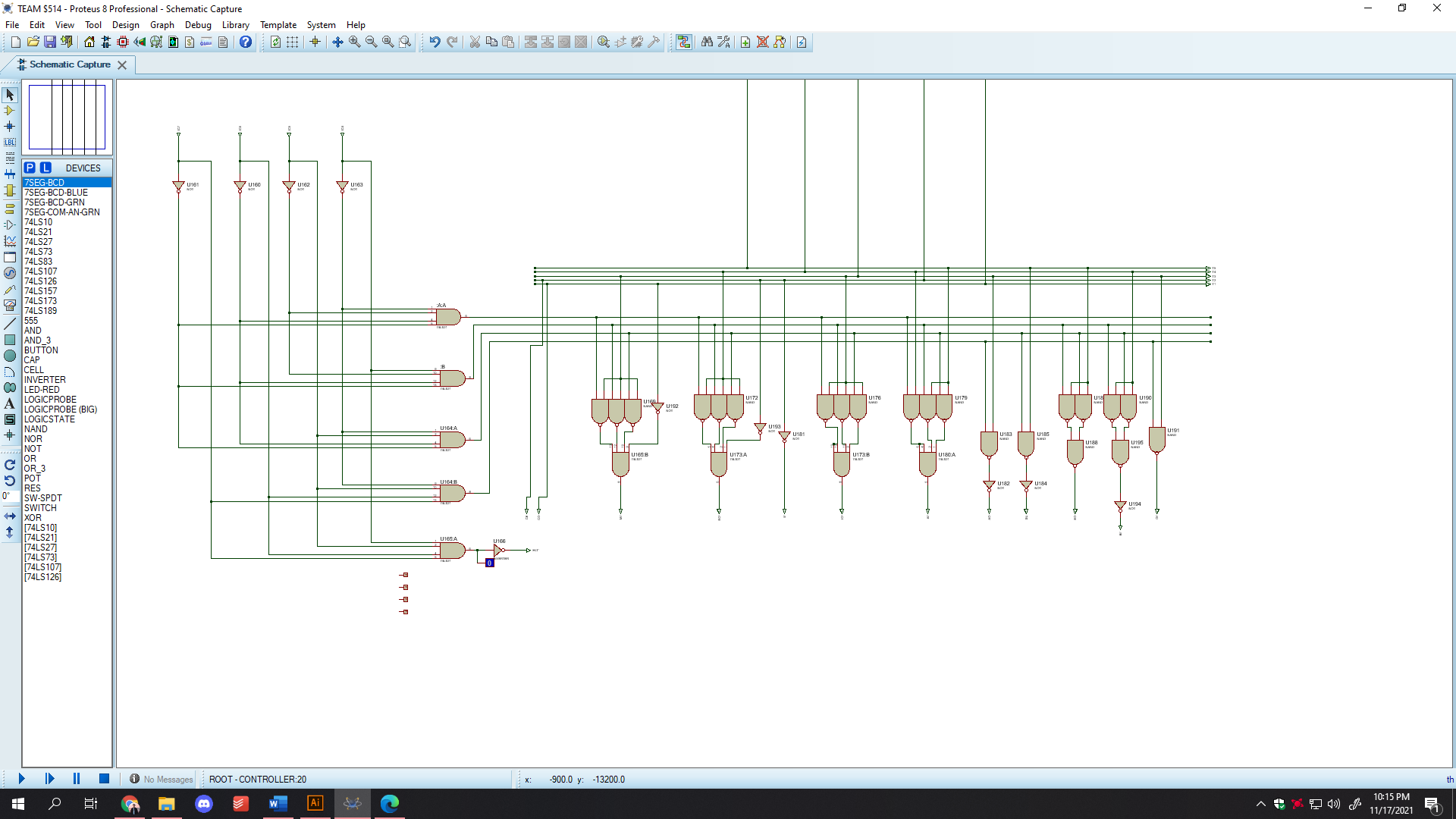
Here in the master slave JK flip flop we connect the master JK with the CLK and we invert it and connect it to the slave. So thus, here a particular value will be stored for one half of a clock pulse cycle. And then when the clock pulse hits again then the value will go from the output of master to the slave.

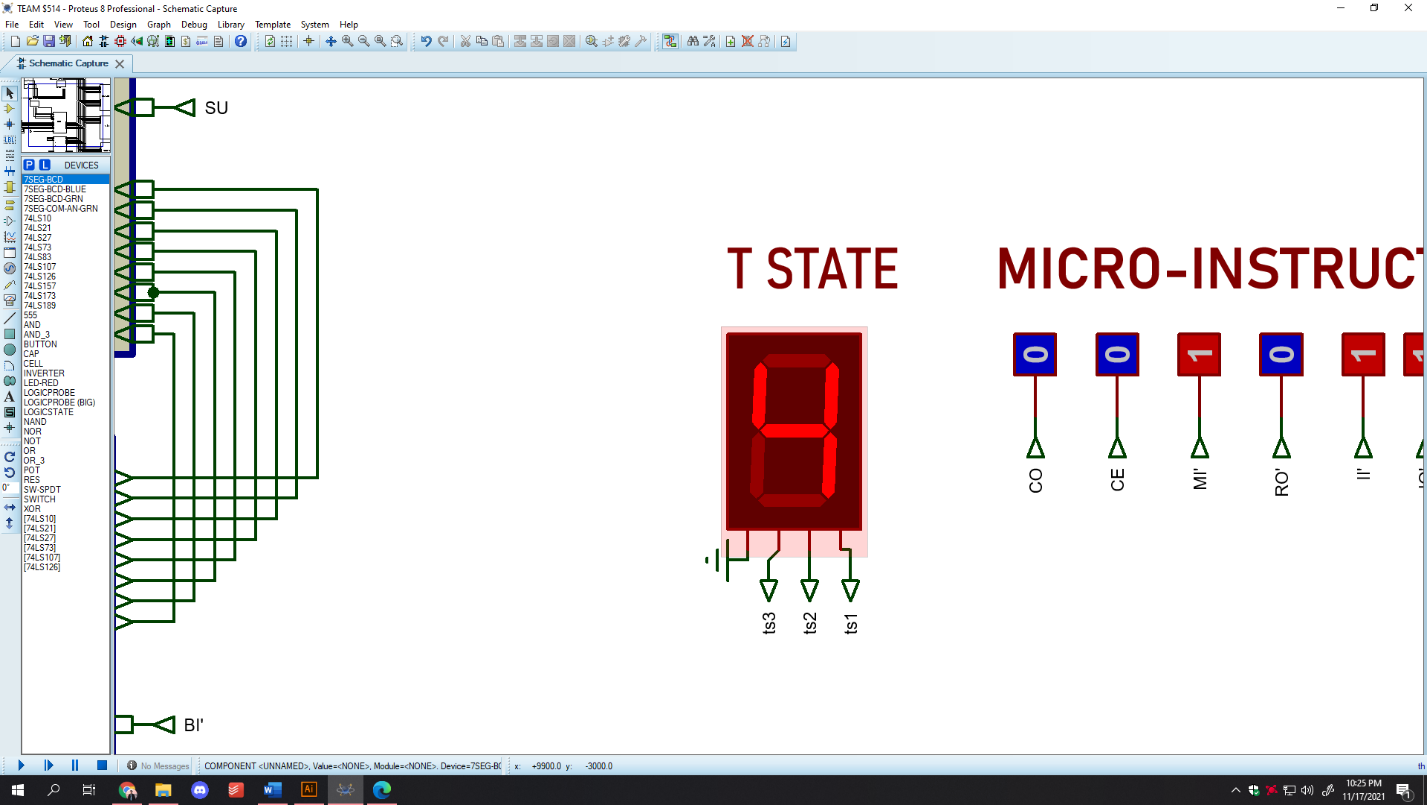
**Algorithm:**

1. We have 5 T states in our SAP. So, we will need 5 Flip flops.
2. As we start in the reset condition first. So, the complemented output of the rightmost flip flop will be our first output in the ring counter.
3. Because it is fed to the “J” input of the next flip flop, so in the next clock pulse, using the master slave JK flip flop theory, the value will go the flip flop that is right next to it in the left direction. It is to be kept in mind, that the shifting of the values is going from right to left in our Controller.
4. Then when we have clock pulse transition, then the value will shift to the Flip Flop on the left. And thus, we complete shifting the value 5 times through each of the 5 flip flops.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **RESET** | **CLK** | **Q0** | **Q1** | **Q2** | **Q3** | **Q4** |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| ↑ | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 |

**Algorithm**

****We must make combinational circuits so that we might get the required output. The combinational circuits are designed so that, when a specific T state is activated through the ring counter, then at that particular moment if either of the Macro is activated then it will give out certain output to the OUTPUT pins.

**Display of T-States**

Now, in our design we wanted to make a display to show the exact T state the SAP is in at a particular time, so that it is convenient for us to keep track. For this we will use the **7 seg BCD display**.

**T states (Table)**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **T1** | **T2** | **T3** | **T4** | **T5** | **t3** | **t2** | **t1** |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |

**Boolean Simplification**

**t3** = T1’.T2’.T3’.T4.T5’+ T1’.T2’.T3’.T4’.T5

= T1’.T2’.T3’ . (T4.T5’+ T4’.T5)

= (T1+T2+T3)’ . (T4⊕T5)

**t2** = T1’.T2.T3’.T4’.T5’+ T1’.T2’.T3.T4’.T5’

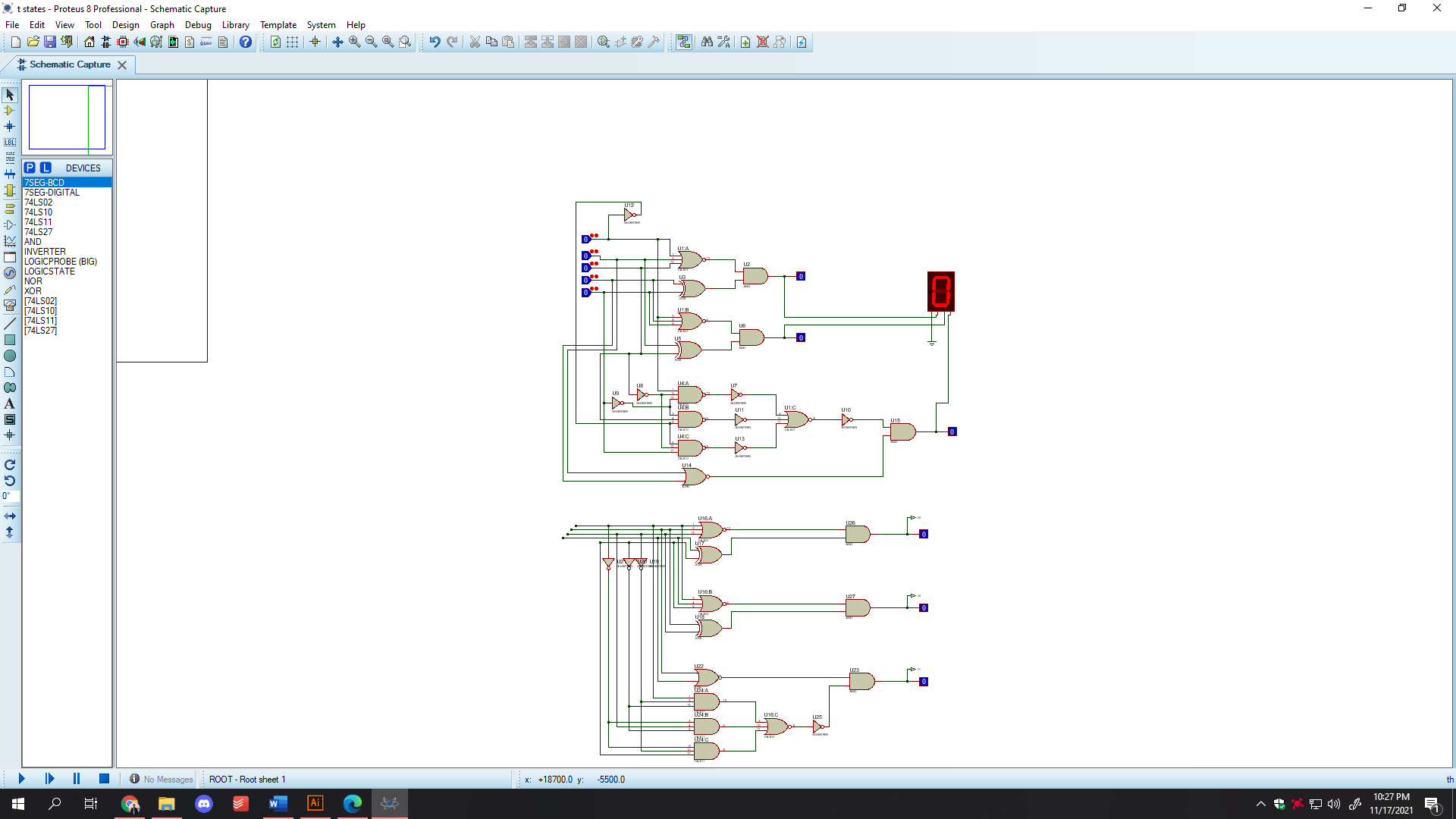
= T1’.T4’.T5’ . (T2.T3’+ T2’.T3)

= (T1+T4+T5)’ . (T2⊕T3)

**t1** = T1.T2’.T3’.T4’.T5’+ T1’.T2’.T3.T4’.T5’+ T1’.T2’.T3’.T4’.T5

=T2’.T4’ . (T1.T3’.T5’+ T1’.T3.T5’+ T1’.T3’.T5)

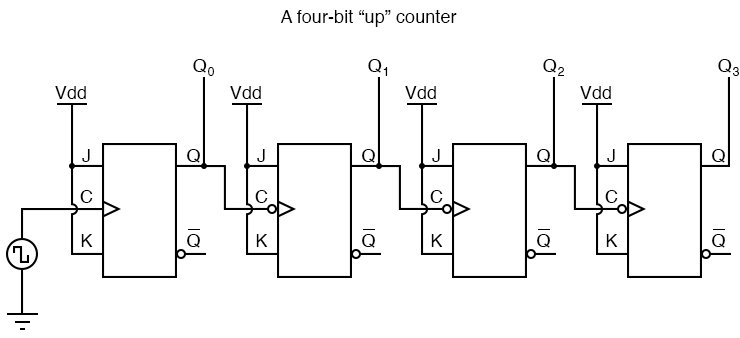
Thus we make a combinational circuit using the above Boolean calculations. And we connect t1, t2 and t3 to the display. And the 4th pin of the display is to be grounded as we won’t be using it anyways.

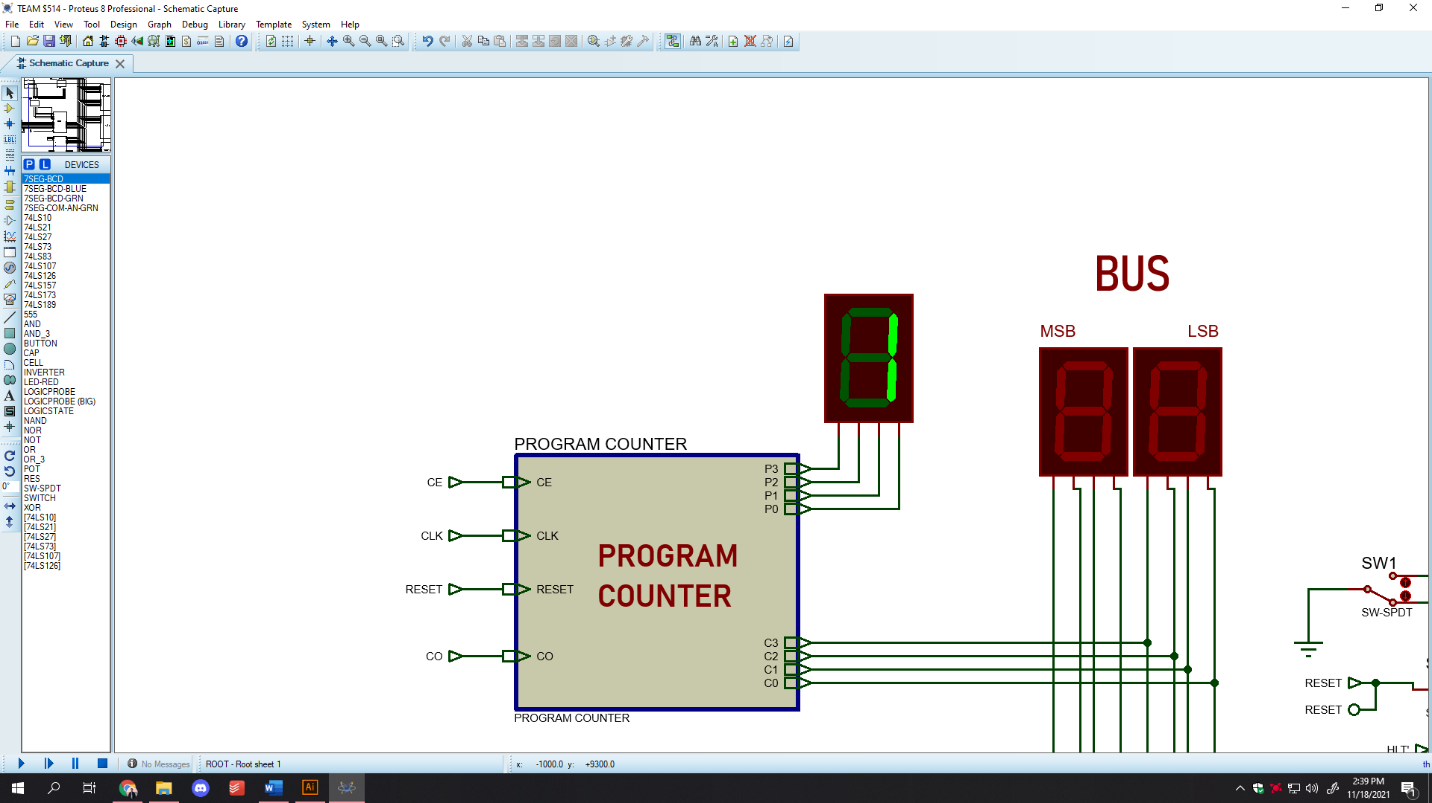


**Program Counter**

A program counter is a [register](https://whatis.techtarget.com/definition/register) in a computer [processor](https://whatis.techtarget.com/definition/processor) that contains the address (location) of the [instruction](https://whatis.techtarget.com/definition/instruction) being executed at the current time. As each instruction gets [fetched](https://searchsqlserver.techtarget.com/definition/fetch), the program counter increases its stored value by 1. After each instruction is fetched, the program counter points to the next instruction in the sequence. When the computer restarts or is reset, the program counter normally reverts to 0.

Source: <https://whatis.techtarget.com/definition/program-counter#:~:text=A%20program%20counter%20is%20a,its%20stored%20value%20by%201.&text=A%20register%20is%20one%20of,places%20that%20the%20processor%20uses>.

**Block Diagram**

Source: <https://www.allaboutcircuits.com/textbook/digital/chpt-11/asynchronouscounters/?fbclid=IwAR14WeMoOkQLTmvGUyqbfXwl30RC7hgRwpjpt1FTJx4o8FD-1uRPFMcDrks>

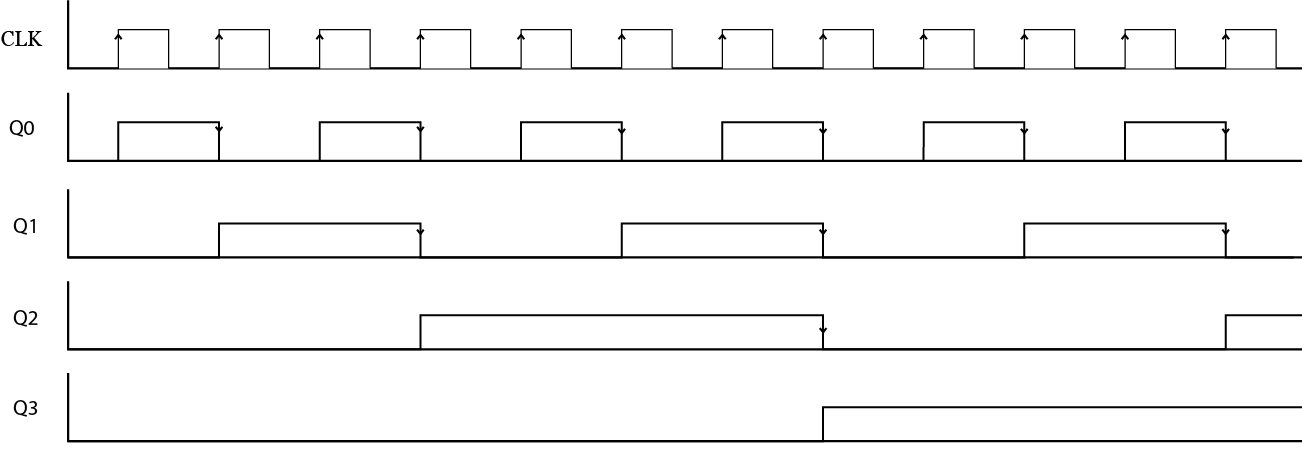
**Specifications**

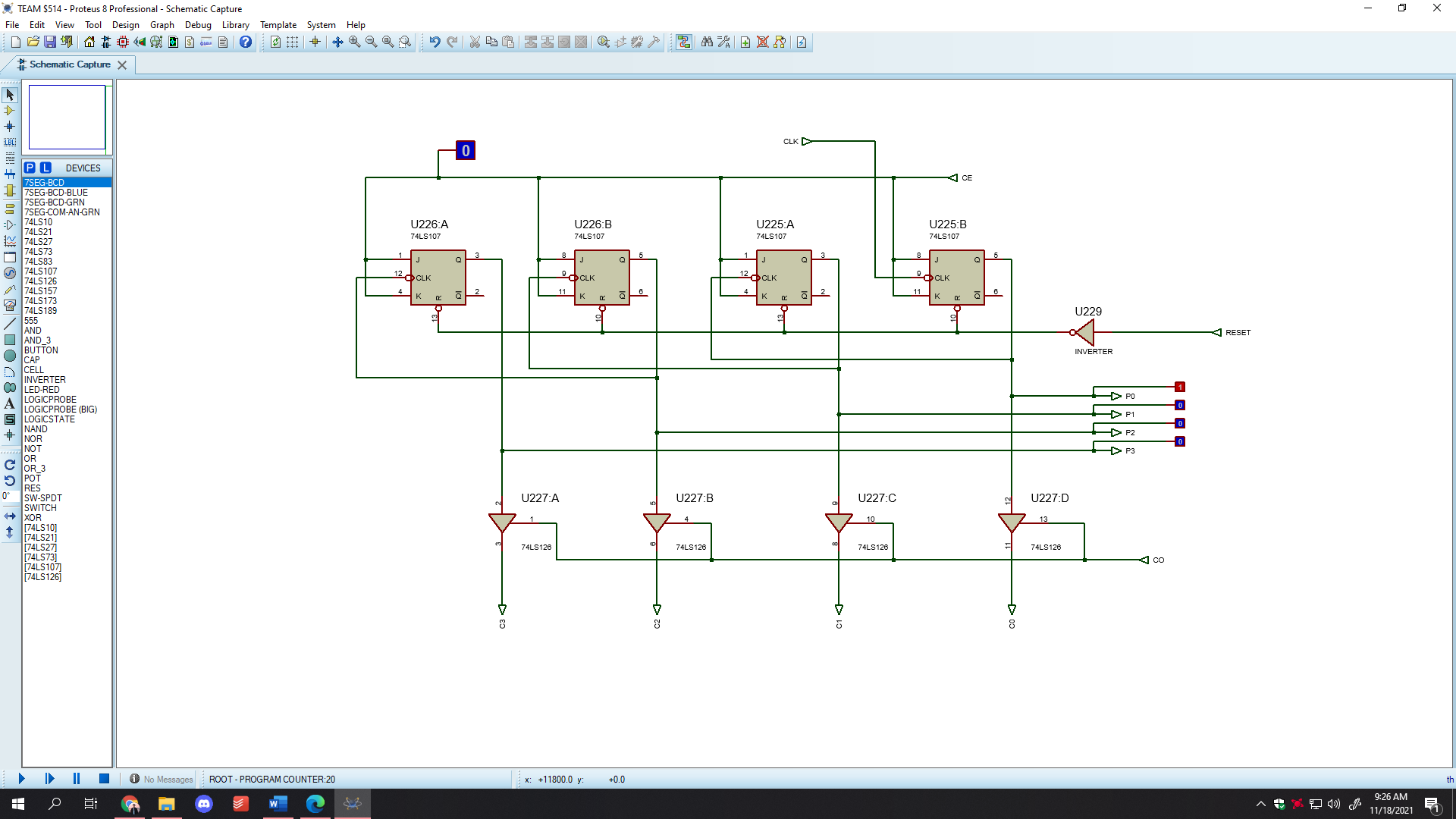
|  |  |
| --- | --- |
| **ICs used** | 74LS107 |
| **Input pins** | CE, CO, CLK, RESET |
| **Output pin (Buffer)** | C3(MSB)-C0(LSB) |
| **Output pin (Display)** | P3(MSB) -P0(LSB) |

|  |
| --- |
| 0000  0001  0010  0011  0100  0101  0110  0111  1000  1001  1010  1011  1100  1101  1110  1111 |

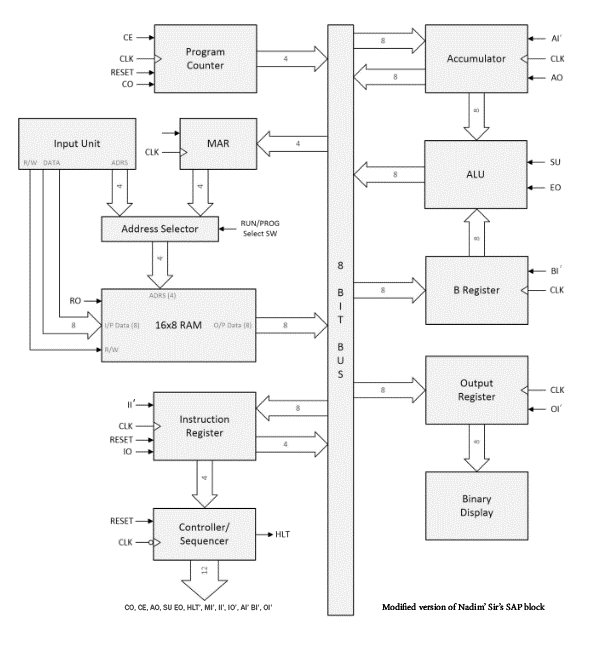
**Algorithm**

1. The rightmost flip flop has a positive edge triggered clock input, so it toggles with each rising edge of the clock signal.
2. Then we feed the output of this flip flop to the clock input of the next flip flop which is an ACTIVE LOW input.
3. Similarly, we connect the Q outputs to the clock input of the next flip flop
4. And then we short all the J and K input pins and connect it to CE input pins
5. Then the output of these flip flops is the output of the program counter
6. Then we add tri-state buffer to each of these output pins. And then the control pins of the buffer are the CO of the program counter.

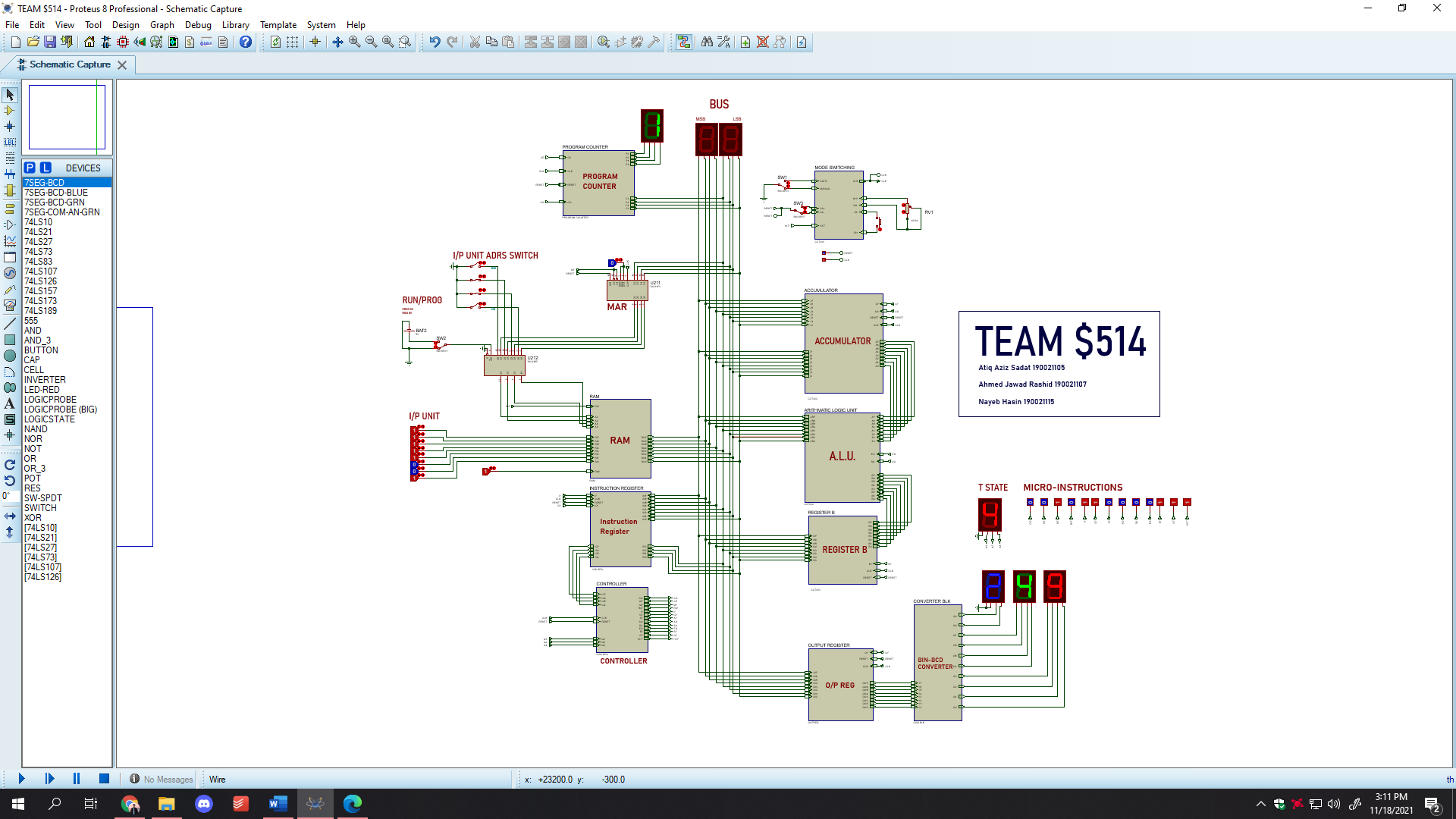
**Timing Diagram of Program Counter**

**Proteus Implementation**

**Complete SAP-1**

**Block Diagram**

**Circuit Implementation using Proteus 8.12**



**Problems faced**

**Clock**

1. **Clock Button and Potentiometer**: We wanted to create a sub-system for our clock module. But as it happened that the push button and the potentiometer was an internal component of the circuit. And there was a limitation in that regard. So we made terminals at particular points and using inputs to the “Mode Switching” module that we made.
2. **Astable mode error in Proteus**: Because there is no initial value to the clock spdt switch, so when we try to simulate the sap at any particular moment then error shows in Proteus.

**Accumulator**

1. When we ran accumulator separately then we couldn’t use the input and output pin names of upper case and lower case of the same alphabet. Otherwise, they were creating short circuit among themselves

**Sequencer**

1. We ran the sequencer using op codes that was created by us using required combinational circuits. Using those specific op codes, the sequencer was giving perfect values in isolation to the entire SAP. But when integrating Sequencer to the bus, we found logic contention issues in over 20 wires. And in that specific case we could only get to 4 T states. And we did not get outputs. And when we only programmed LDA we got output. So as we were debugging the issue we decided to revert back to another op code, which seemed to do the job at the time. And then we found out that the HLT’ was working as well, which was not functioning prior to changing the op codes.
2. We also faced difficulty in the creation of the combinational circuit of the display of T states. Because when we simulated it separately it was working but it didn’t give any result when we integrated it to the SAP. Then by changing some of the Boolean expressions we were able to execute our result.