Expt. No:	11	Registers and Counters
Date:	7/11/2020	

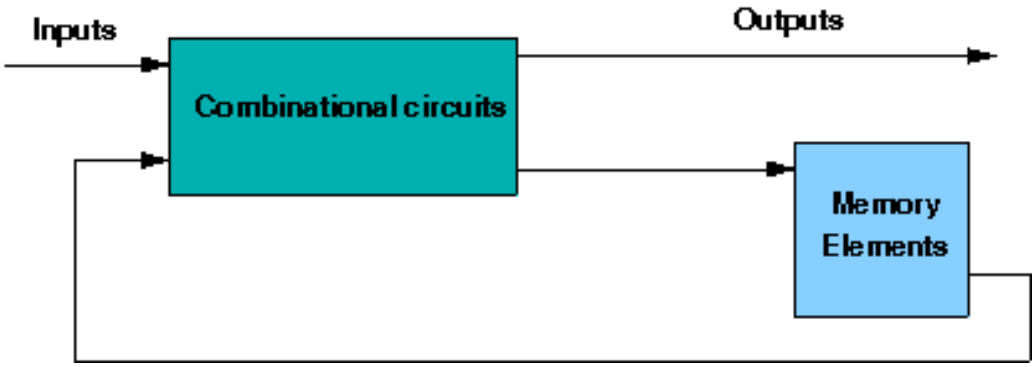
AIM: To study, design and implement 3-Bit up Counter, Mod-7 Counter, 4-Bit Shift Right Register, 4 – Bit Shift Left Register.

SOFTWARE TOOLS / OTHER REQUIREMENTS:

1. Multisim Simulator/Circuit Simulator

THEORY:

In a sequential circuit the present output is determined by both the present input and the past output.
In order to receive the past output some kind of memory element can be used. The memory element commonly used in the sequential circuits are time-delay devices. The block diagram of the sequential circuit-



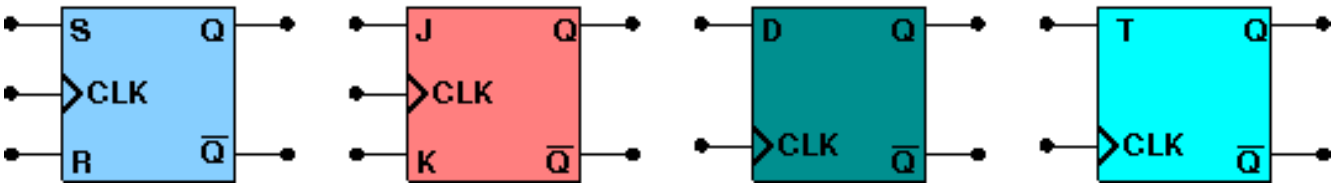
A circuit with flip-flops is considered a *sequential circuit* even in the absence of combinational logic.

Circuits that include flip-flops are usually classified by the function they perform.
Two such circuits are registers and counters:

1. **Register** is a group of flip-flops. Its basic function is to hold information within a digital system so as to make it available to the logic units during the computing process.
2. **Counter** is essentially a register that goes through a predetermined sequence of states.

There are various different kind of flip-flops. Some of the common flip-flops are: R-S flip-flop, D flip-flop, J-K flip-flop, T flip-flop.

The block diagram of different flip-flops are shown here –



• RS flip-flop if r is high then reset state occurs and when s=1 set state.
The both cannot be high simultaneously. This input combination is avoided.

• JK flip-flop if J and K are both low then no change occurs.
If J and K are both high at the clock edge then the output will toggle from one state to the other.

• D flip-flop the d flip-flop tracks the input, making transitions with match those of the input d. It is used as data store.

• T flip-flop or "Toggle" flip-flop changes its output on each clock edge,

I.) REGISTERS:

Flip flops can be used to store *a single bit of binary data* (1 or 0). However, in order to store multiple bits of data, we need multiple flip flops. N flip flops are to be connected in an order to store N bits of data.

A **Register** is a device which is used to store such information.
It is a group of flip flops connected in series used to store multiple bits of data.

The information stored within these registers can be transferred with the help of **shift registers**.

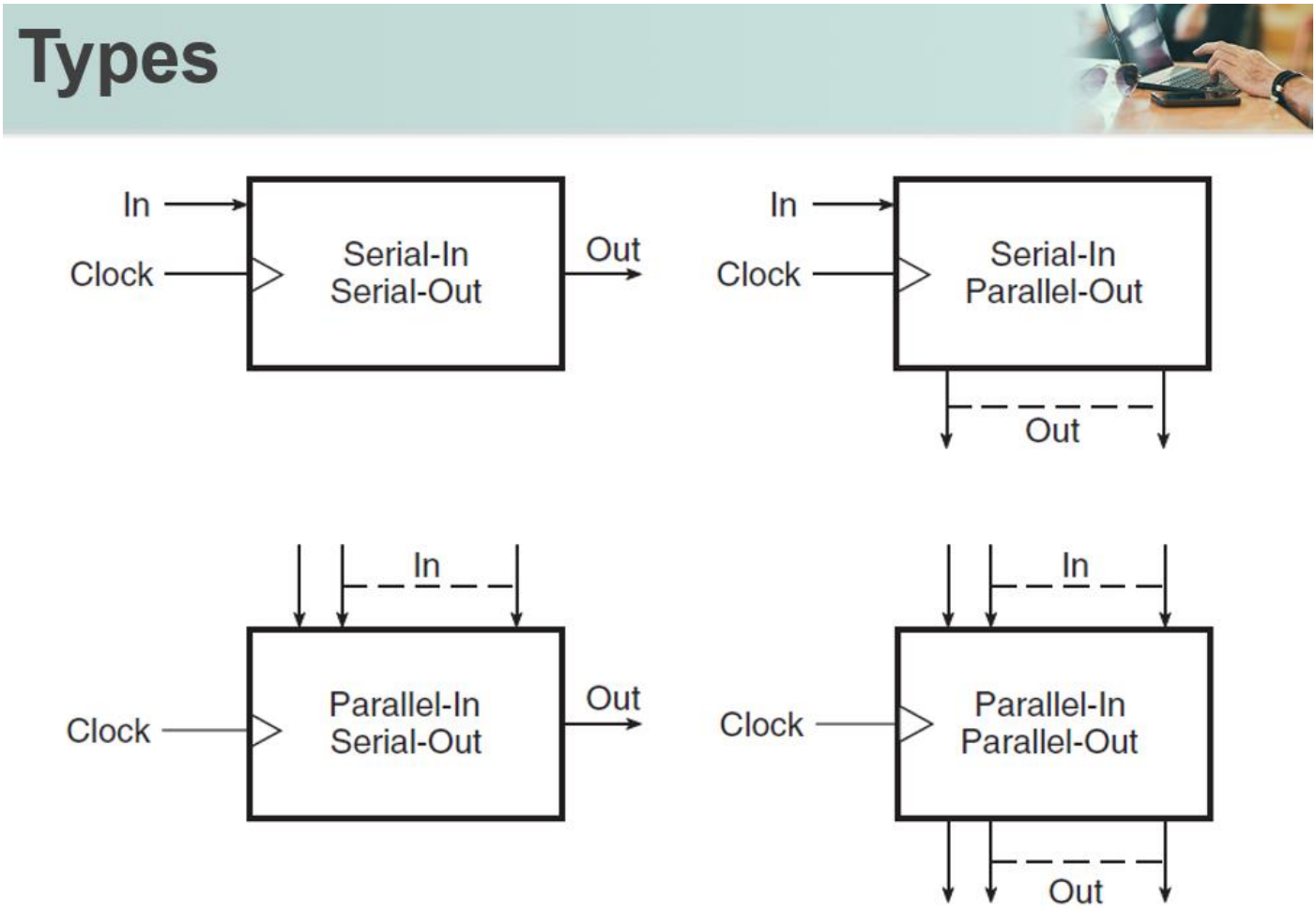
Shift Register is a group of flip flops used to store multiple bits of data. The bits stored in such registers can be made to move within the registers and in/out of the registers by applying clock pulses.
An **n-bit** shift register can be formed by connecting **n** flip-flops where each flip flop stores a single bit of data.

The registers which will shift the bits to left are called "*Shift left registers*".
The registers which will shift the bits to right are called "*Shift right registers*".

Shift registers are basically of **4** types.

These are:

1. Serial In Serial Out shift register [SISO]
2. Serial In parallel Out shift register [SIPO]
3. Parallel In Serial Out shift register [PISO]
4. Parallel In parallel Out shift register [PIPO]



A.) Serial-In Serial-Out Shift Register (SISO)

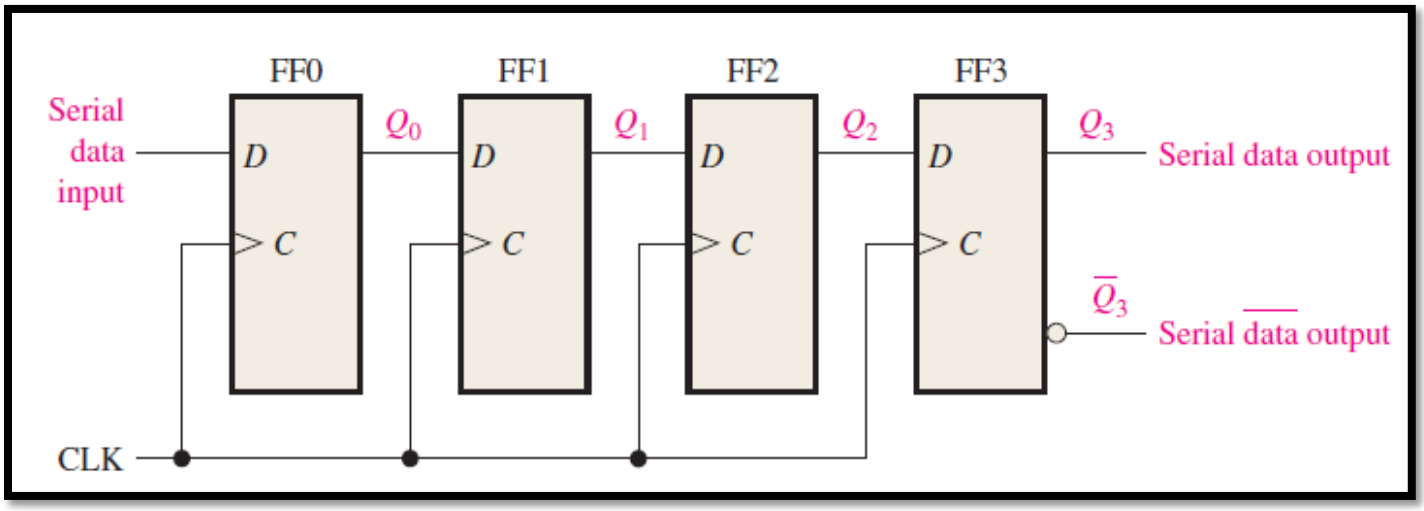
The shift register, which allows serial input (one bit after the other through a single data line) and produces a serial output is known as *Serial-In Serial-Out shift register*.

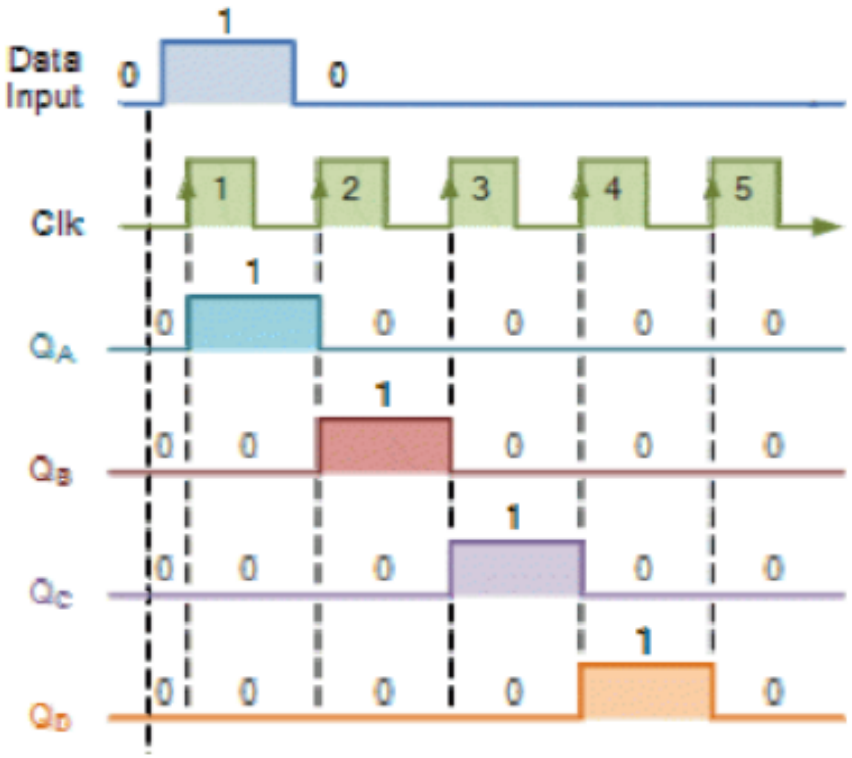
Since there is only **one** output, the data leaves the shift register one bit at a time in a serial pattern, thus the name Serial-In Serial-Out Shift Register.

The logic circuit given below shows a serial-in serial-out shift register.

The circuit consists of *four D* flip-flops which are connected in a serial manner.

All these flip-flops are synchronous with each other since the same clock signal is applied to each flip flop.





The timing diagram of data shift through a 4-bit SISO shift register

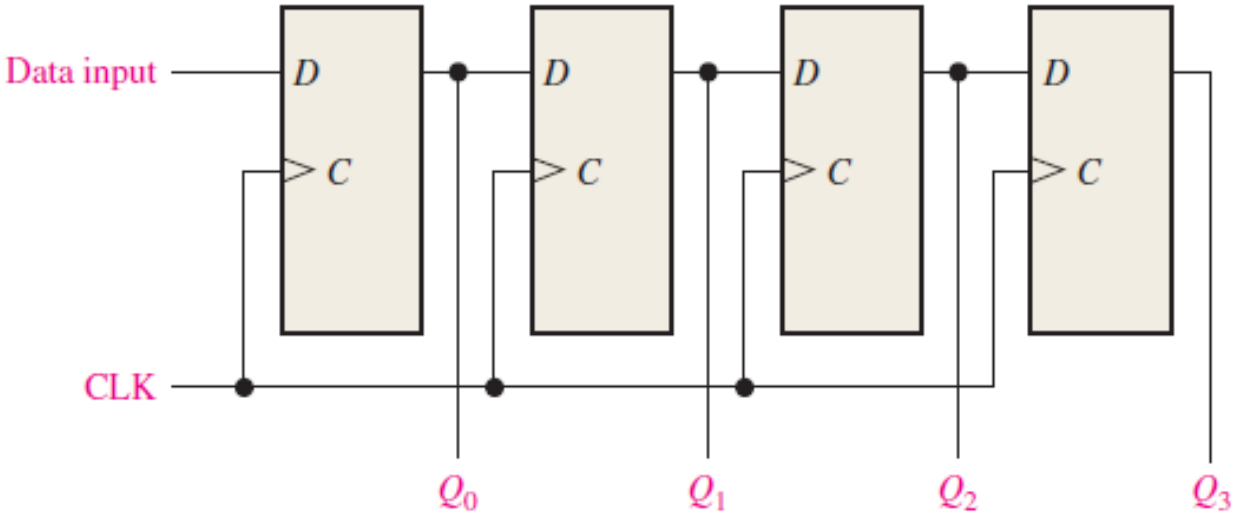
B.) Serial-In Parallel-Out shift Register (SIPO)

The shift register, which allows serial input (one bit after the other through a single data line) and produces a parallel output is known as *Serial-In Parallel-Out* shift register.

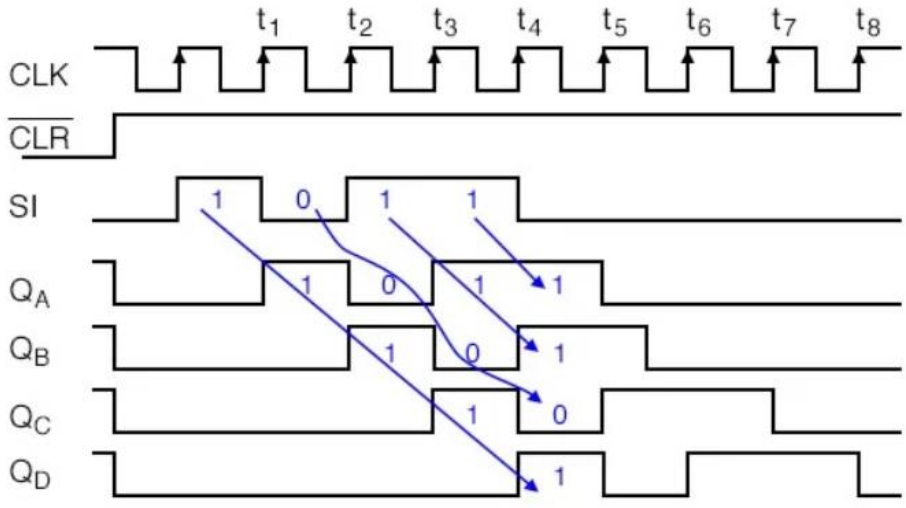
The logic circuit given below shows a serial-in-parallel-out shift register. The circuit consists of four D flip-flops which are connected.

The clear (CLR) signal is connected in addition to the clock signal to all the 4 flip flops in order to RESET them.

The output of the first flip flop is connected to the input of the next flip flop and so on. All these flip-flops are synchronous with each other since the same clock signal is applied to each flip flop.



SERIAL IN PARALLER OUT [S.I.P.O.]



Serial-in/ parallel-out shift register waveforms

C.) Parallel-In Serial-Out Shift Register (PISO)

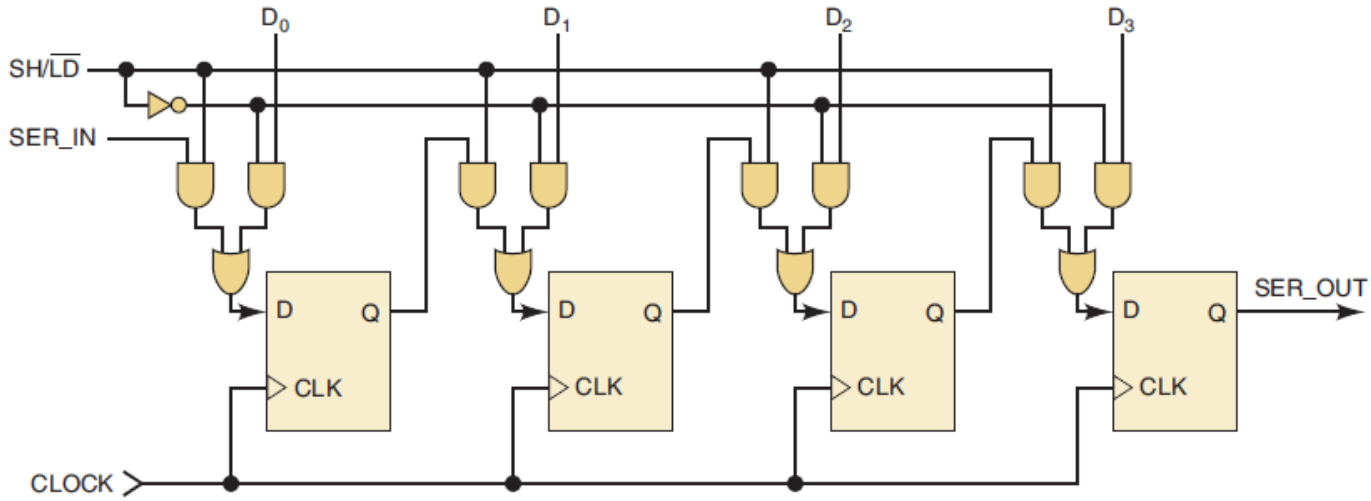
The shift register, which allows parallel input (data is given separately to each flip flop and in a simultaneous manner) and produces a serial output is known as *Parallel-In Serial-Out* shift register.

The logic circuit given below shows a parallel-in-serial-out shift register.

The circuit consists of four D flip-flops which are connected.

The clock input is *directly* connected to all the flip flops but the input data is connected individually to each flip flop through a multiplexer at the input of every flip flop.

The output of the previous flip flop and parallel data input are connected to the input of the MUX and the output of MUX is connected to the next flip flop. All these flip-flops are synchronous with each other since the same clock signal is applied to each flip flop.



PARALLEL IN SERIAL OUT REGISTER

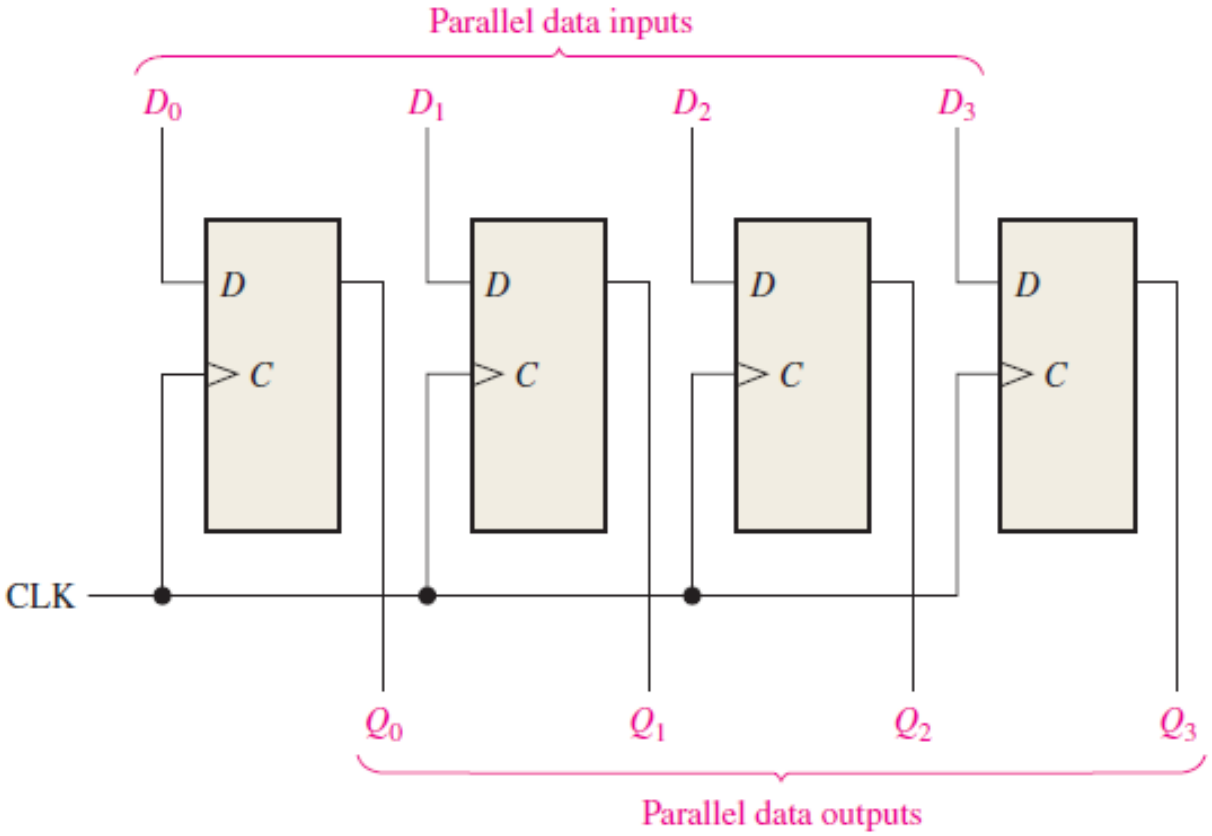
D.) Parallel-In Parallel-Out Shift Register (PIPO)

The shift register, which allows parallel input (data is given separately to each flip flop and in a simultaneous manner) and also produces a parallel output is known as *Parallel-In parallel-Out shift* register.

The logic circuit given below shows a parallel-in-parallel-out shift register.

The circuit consists of four D flip-flops which are connected. The clear (CLR) signal and clock signals are connected to all the 4 flip flops.

In this type of register, there are no interconnections between the individual flip-flops since no serial shifting of the data is required. Data is given as input separately for each flip flop and in the same way, output also collected individually from each flip flop.



COUNTER:

Counter is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock signal.

Counters are used in digital electronics for counting purpose, they can count specific event happening in the circuit. For example, in *UP counter* a counter increases count for every rising edge of clock. Not only counting, a counter can follow the certain sequence based on our design like any random sequence 0, 1, 3, 2….They can also be designed with the help of flip flops.

Modulus:

To determine the number of flip-flops requires to build a counter having a given modulus, identify the smallest integer m that is either equal to or greater than the desired modulus and is also equal to an integral power of 2.
For instance, if the desired modulus is 10, which is the case in a decode counter, the smallest integer greater than or equal to 10 and which is also an integral power of 2 is 16. The number of flip-flops in this case would be 4, as 16= 2^4.

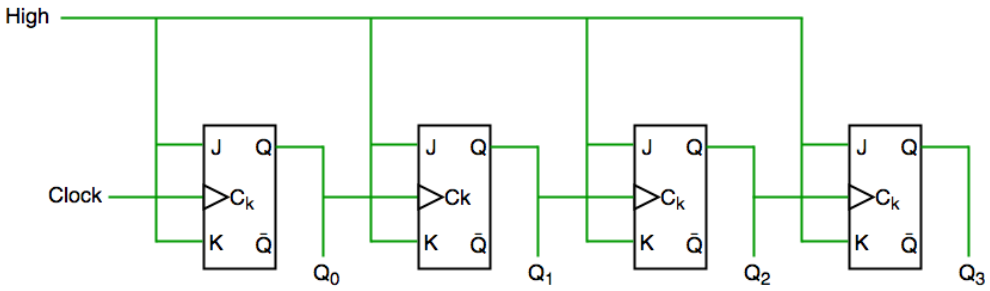
$$\text{Modulus} \geq 2^N$$

Counters are broadly divided into two categories

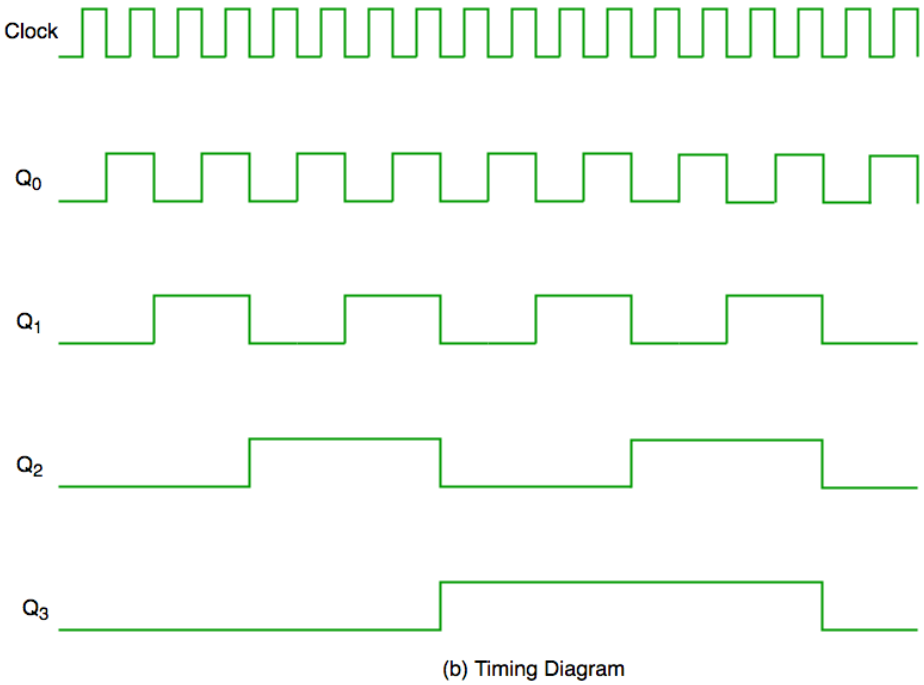
1. Asynchronous counter
2. Synchronous counter

1. **Asynchronous Counter**

In asynchronous counter we don't use universal clock, only first flip flop is driven by main clock and the clock input of rest of the following flip flop is driven by output of previous flip flops. We can understand it by following diagram-



(a) Asynchronous counter



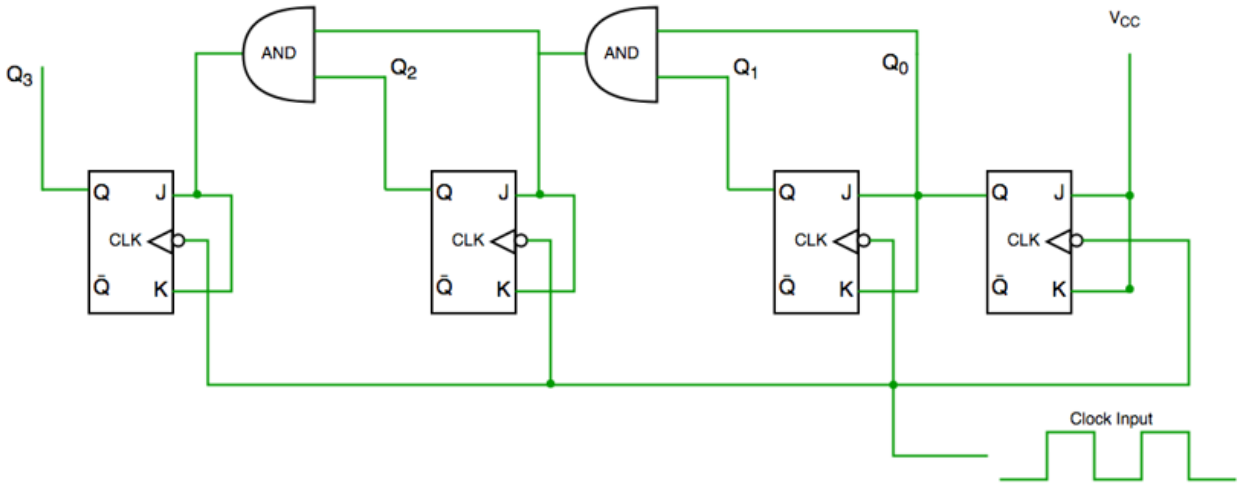
(b) Timing Diagram

ASYNCHRONOUS COUNTER AND ITS TIMING DIAGRAM

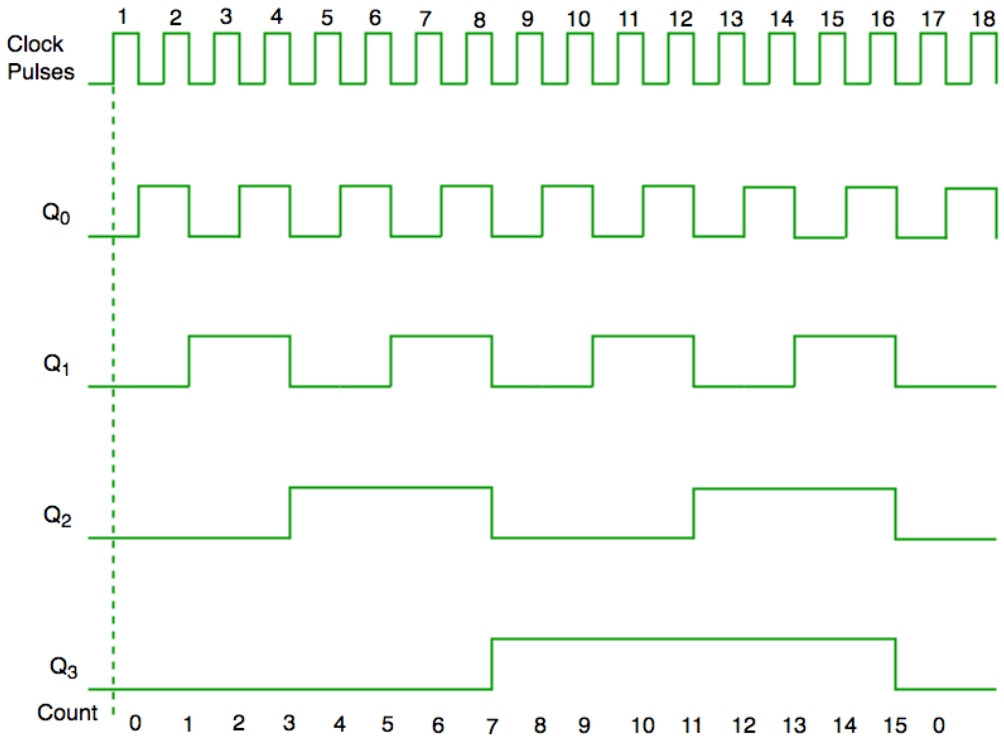
It is evident from timing diagram that Q0 is changing as soon as the rising edge of clock pulse is encountered, Q1 is changing when rising edge of Q0 is encountered (because Q0 is like clock pulse for second flip flop) and so on. In this way ripples are generated through Q0, Q1, Q2, Q3 hence it is also called **Ripple counter.**

2. Synchronous Counter

Unlike the asynchronous counter, synchronous counter has one global clock which drives each flip flop so output changes in parallel. The one advantage of synchronous counter over asynchronous counter is, it *can operate on higher frequency* than asynchronous counter as it does not have cumulative delay because of same clock is given to each flip flop.



SYNCHRONOUS COUNTER

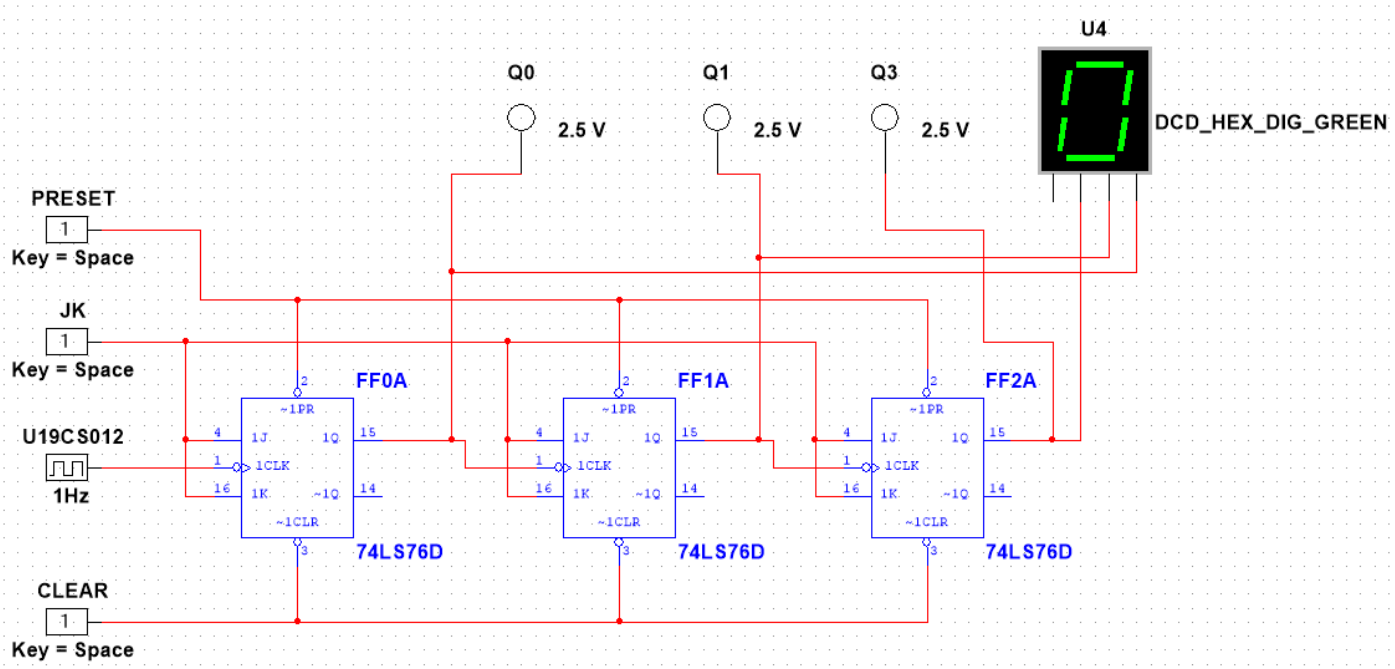


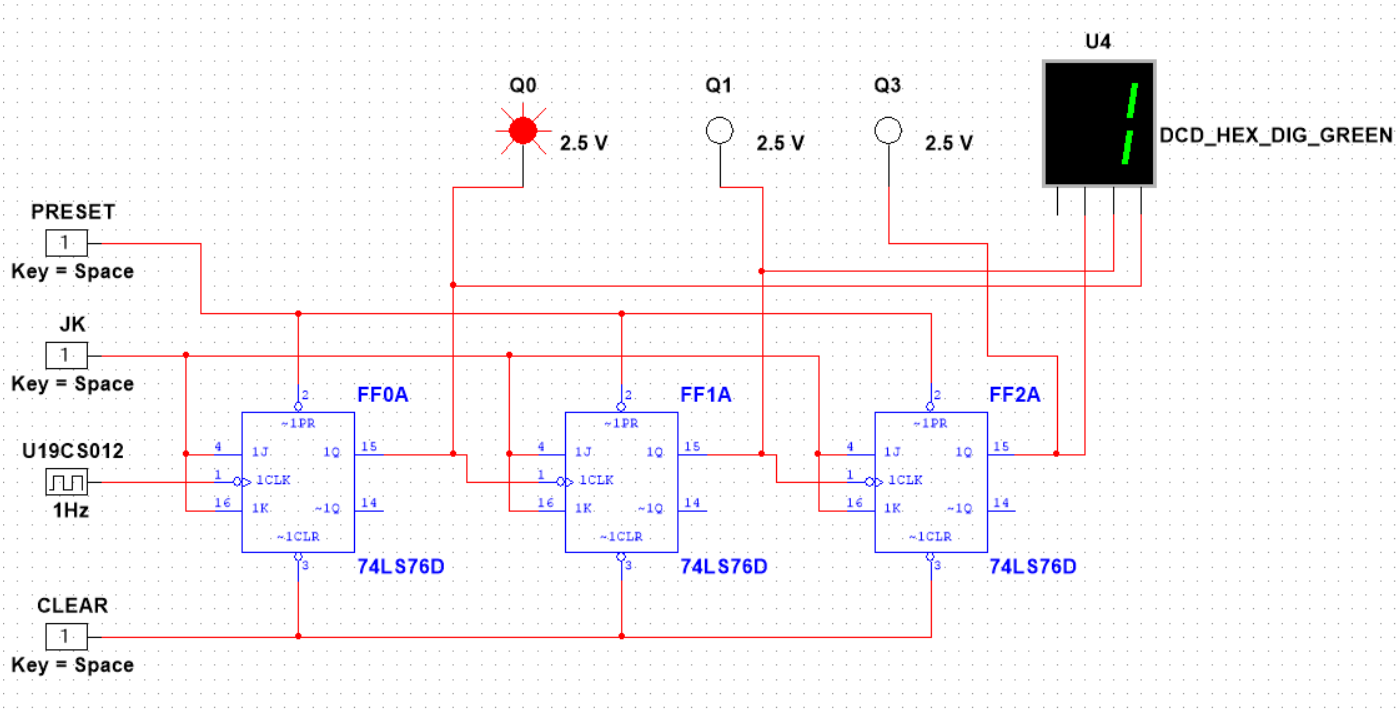
TIMING DIAGRAM

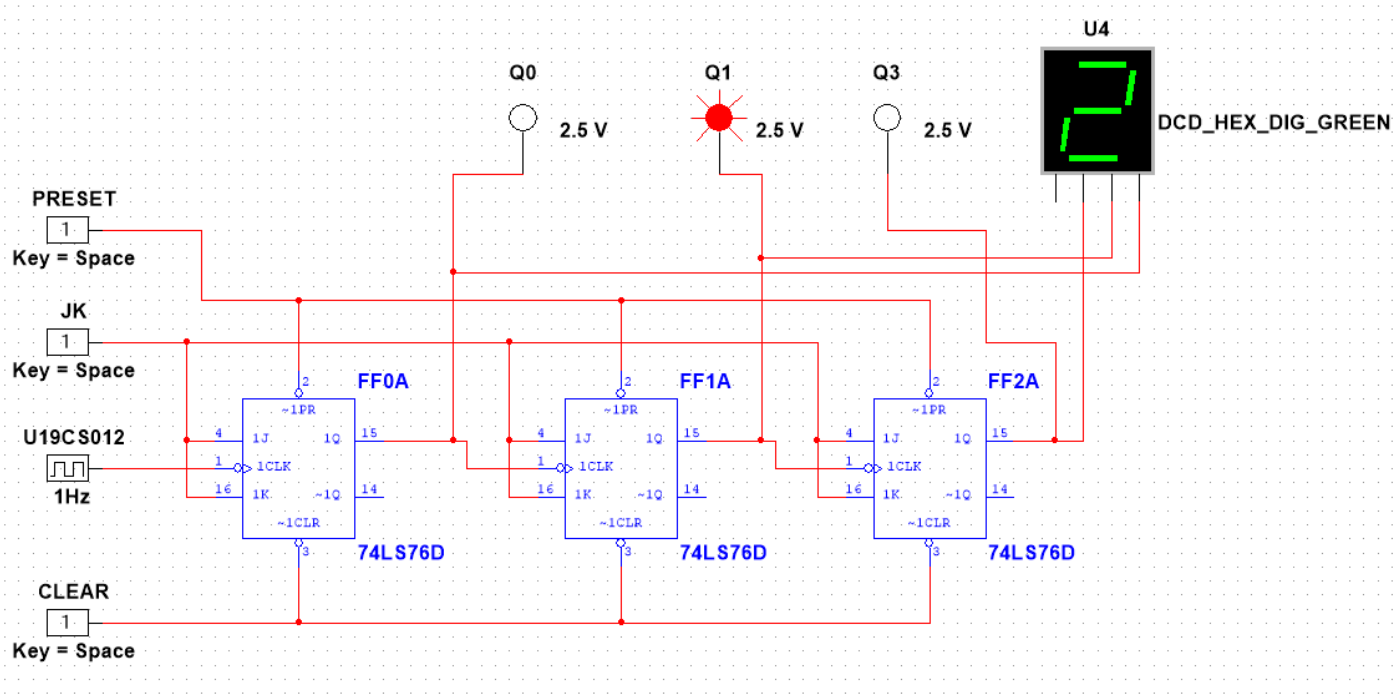
From circuit diagram we see that Q0 bit gives response to each falling edge of clock while Q1 is dependent on Q0, Q2 is dependent on Q1 and Q0, Q3 is dependent on Q2, Q1 and Q0.

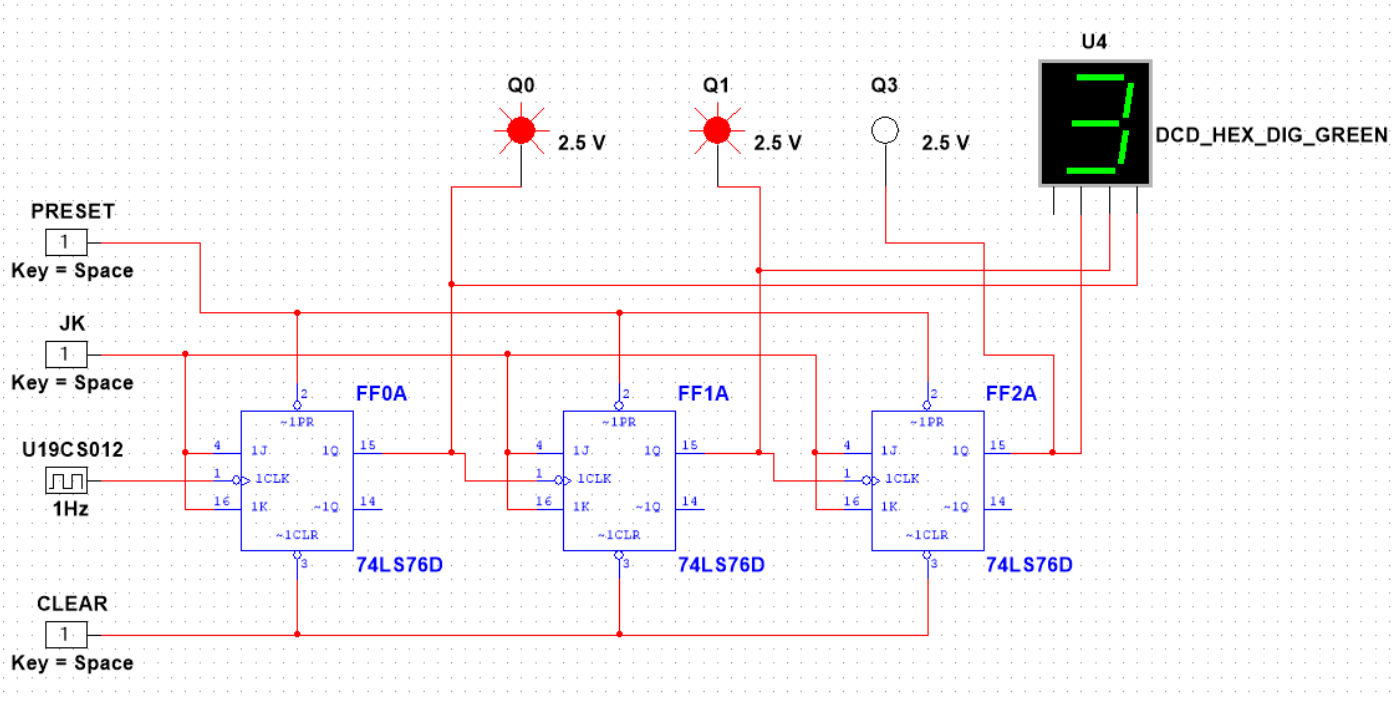
SIMULATION SCREENSHOTS

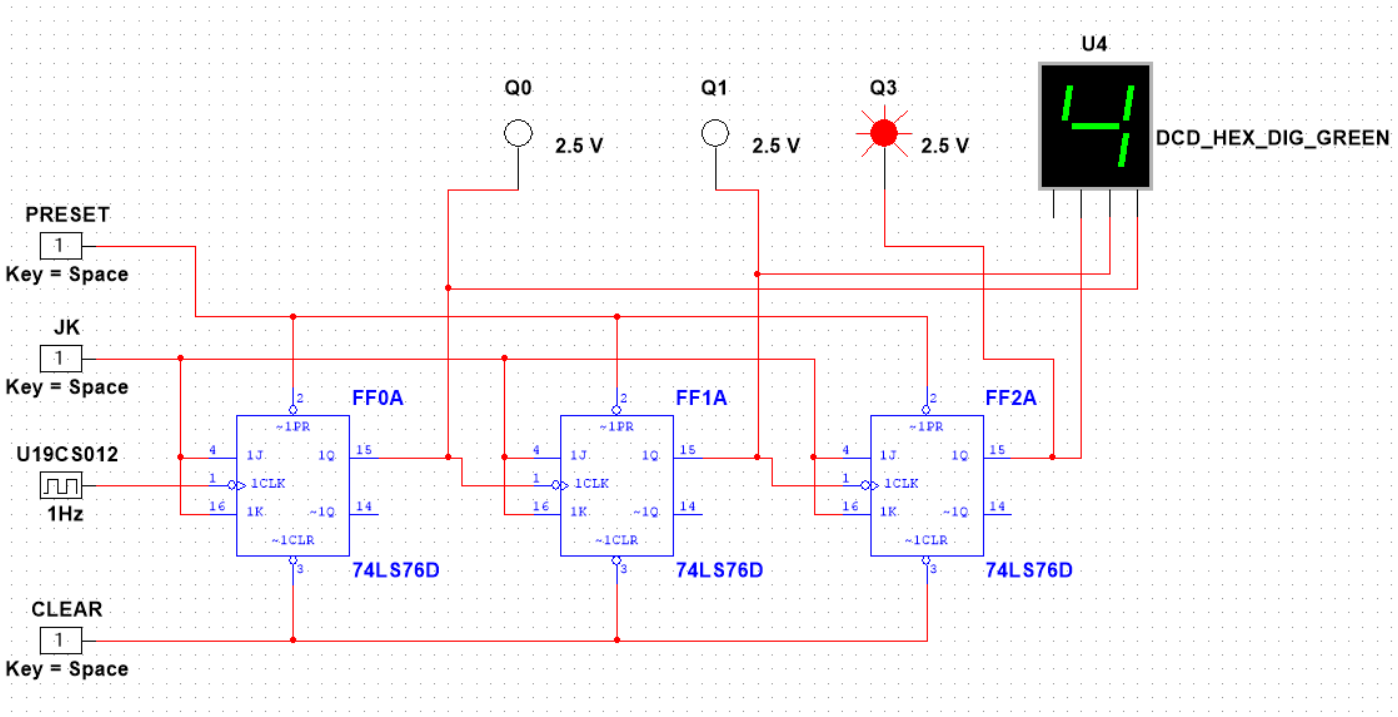
3-BIT UP FULL MODULUS COUNTER

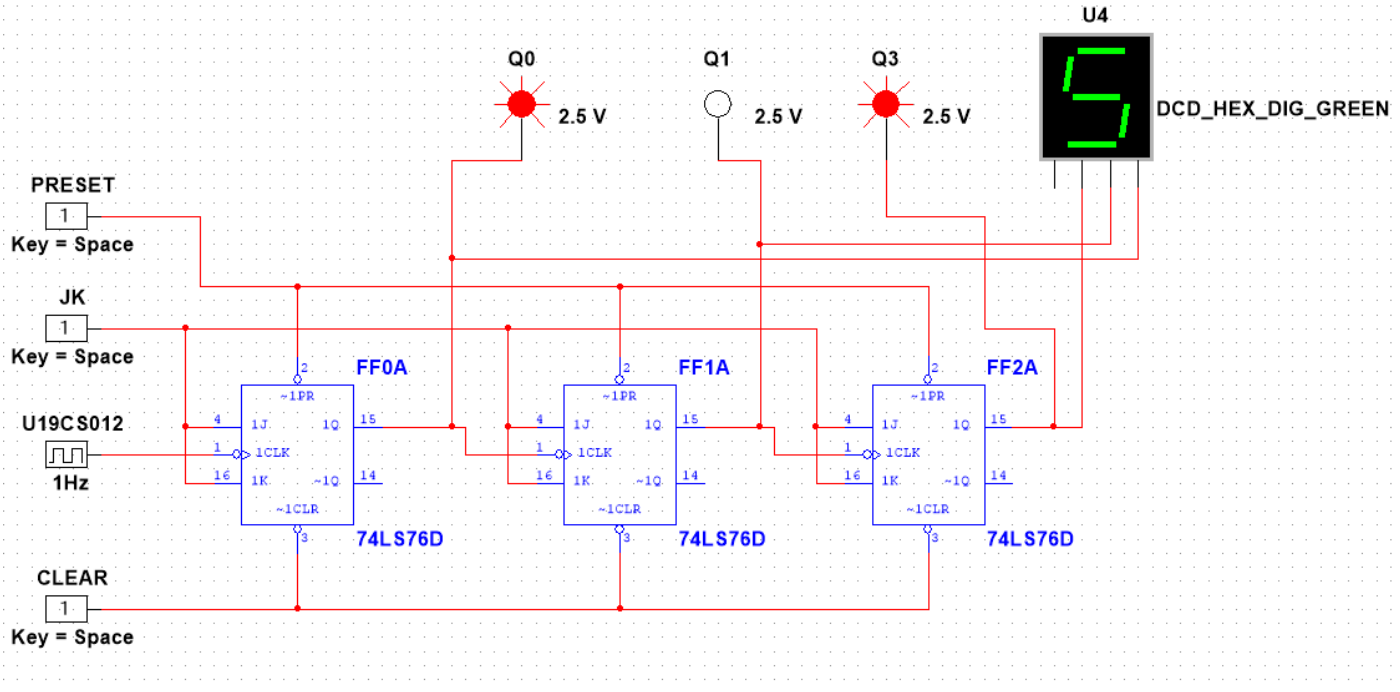


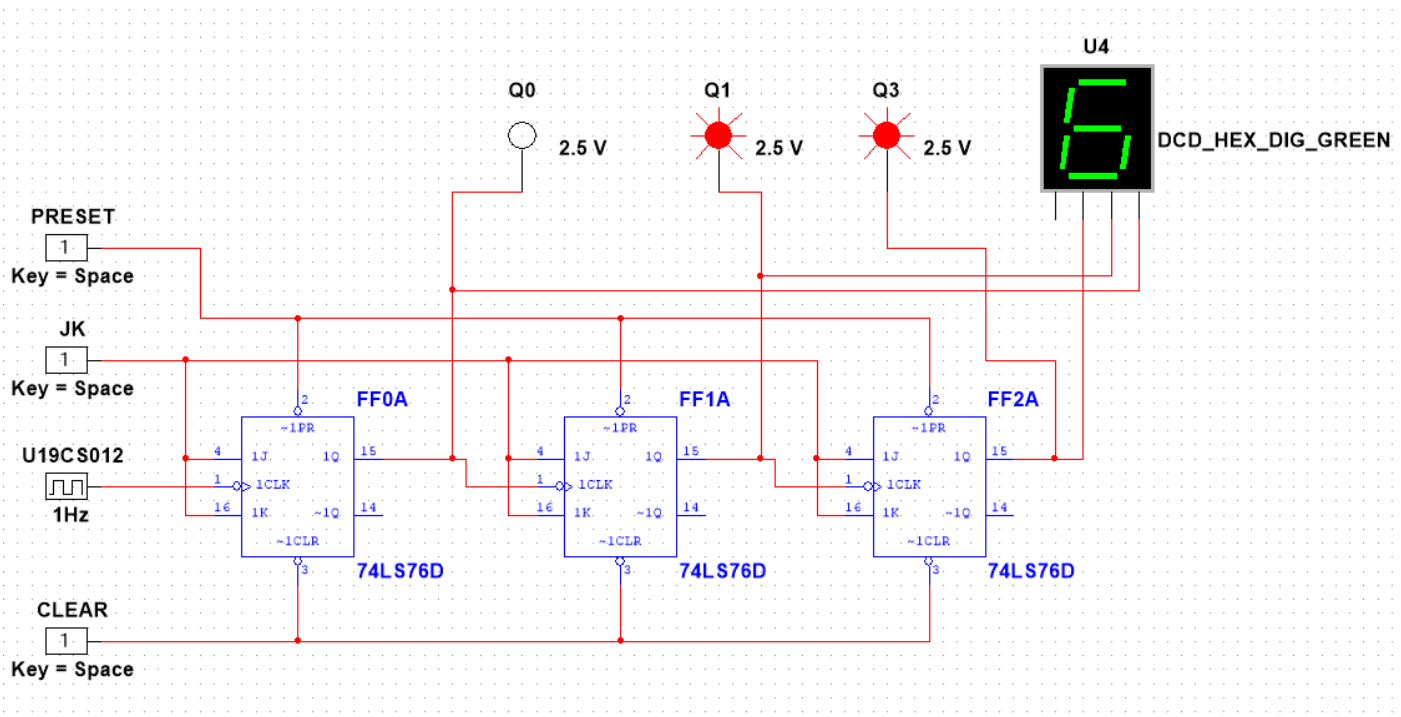


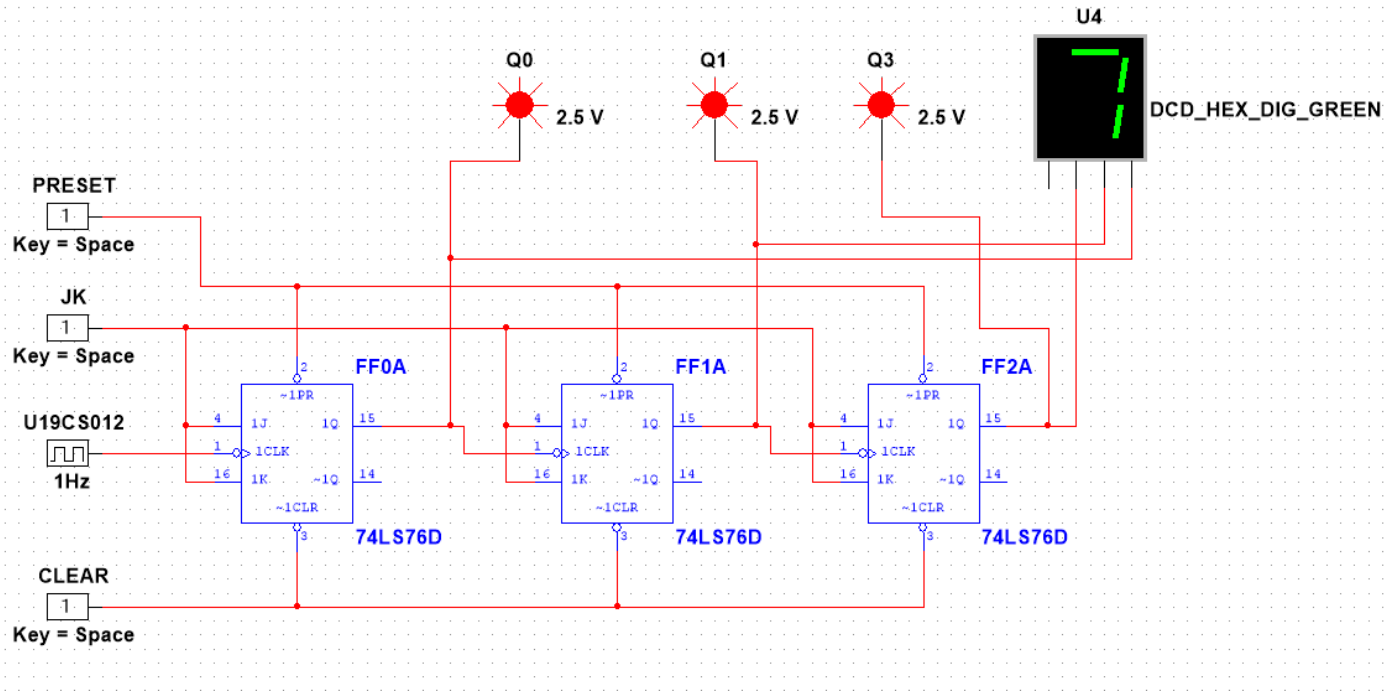

U4
Q0
Q1
Q3
2.5 V
2.5 V
2.5 V
DCD_HEX_DIG_GREEN
PRESET
1
Key = Space
JK
1
Key = Space
U19CS012
1Hz
CLEAR
1
Key = Space
FF0A
FF1A
FF2A
74LS76D
74LS76D
74LS76D

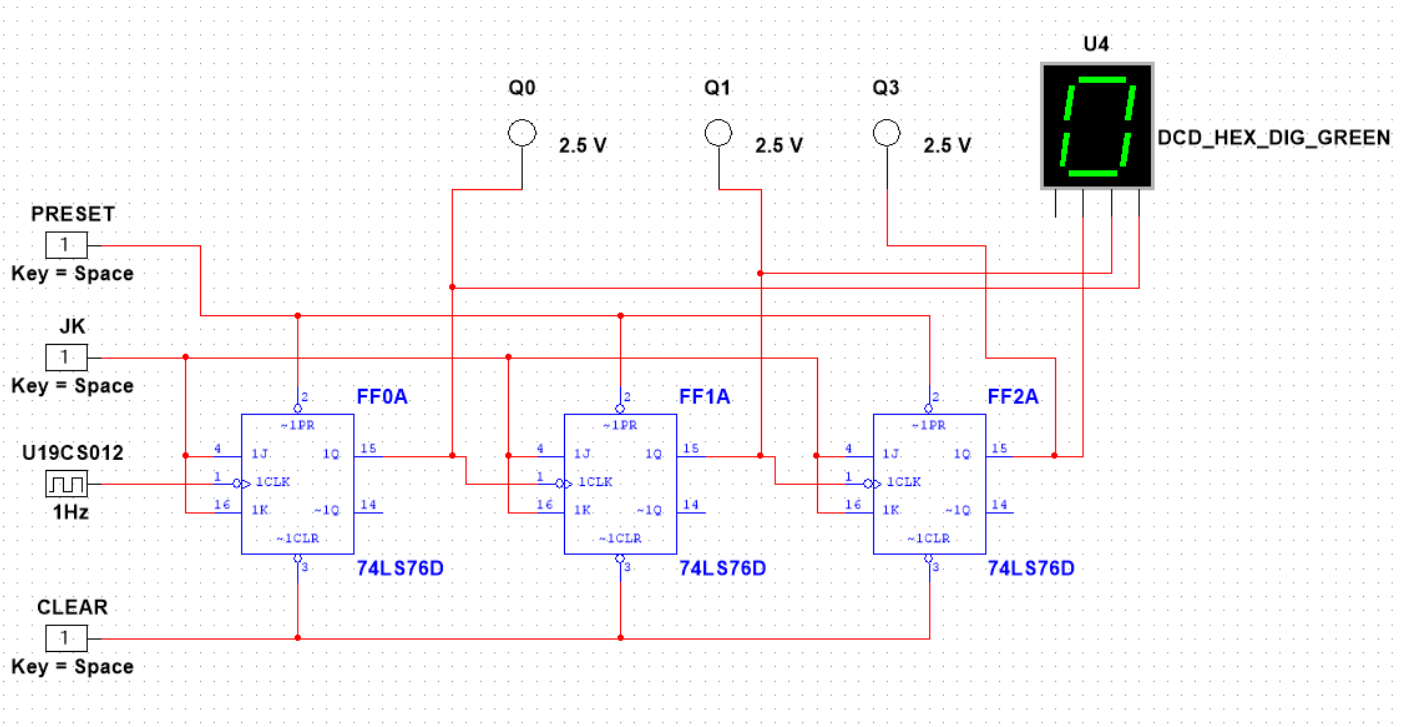

U4
Q0
Q1
Q3
2.5 V
2.5 V
2.5 V
DCD_HEX_DIG_GREEN
PRESET
1
Key = Space
JK
1
Key = Space
U19CS012
1Hz
CLEAR
1
Key = Space
FF0A
FF1A
FF2A
74LS76D
74LS76D
74LS76D


U4
Q0
Q1
Q3
2.5 V
2.5 V
2.5 V
DCD_HEX_DIG_GREEN
PRESET
Key = Space
JK
Key = Space
U19CS012
1Hz
CLEAR
Key = Space
FF0A
FF1A
FF2A
74LS76D
74LS76D
74LS76D


U4
Q0
Q1
Q3
2.5 V
2.5 V
2.5 V
DCD_HEX_DIG_GREEN
PRESET
Key = Space
JK
Key = Space
U19CS012
1Hz
CLEAR
Key = Space
FF0A
FF1A
FF2A
74LS76D
74LS76D
74LS76D

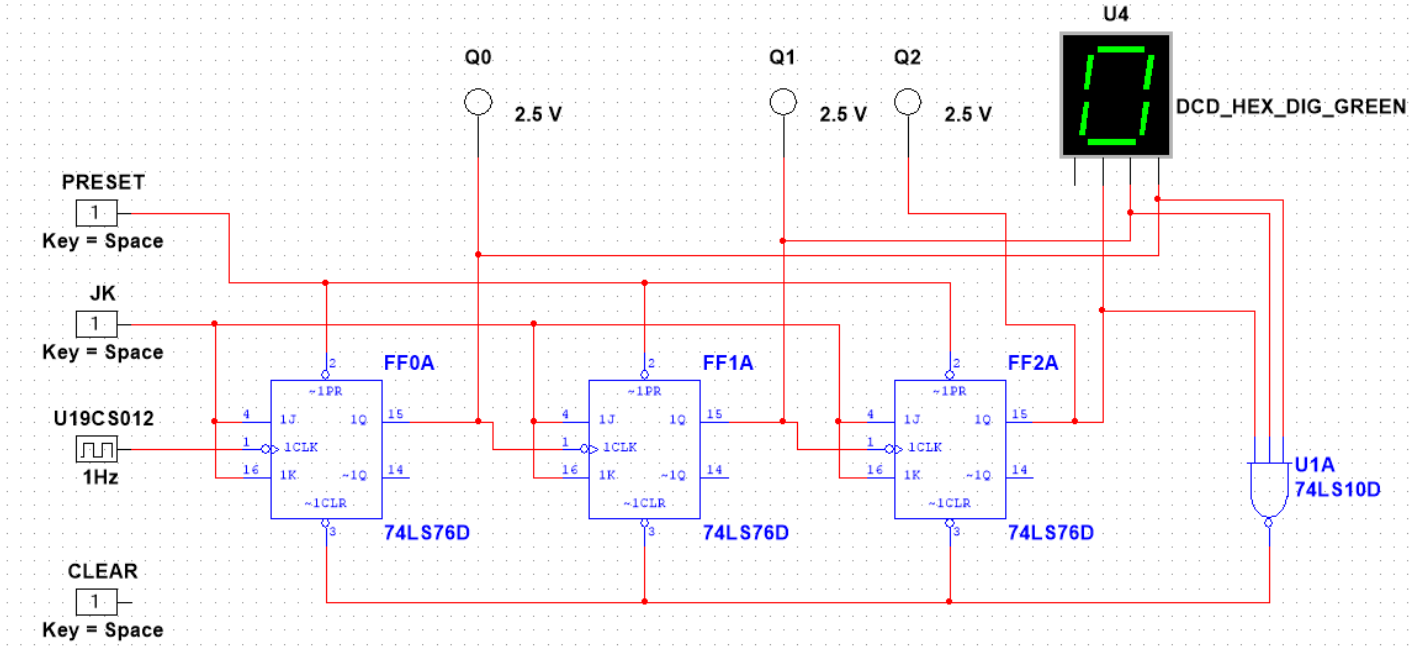

U4
Q0
Q1
Q3
2.5 V
2.5 V
2.5 V
DCD_HEX_DIG_GREEN
PRESET
1
Key = Space
JK
1
Key = Space
U19CS012
1Hz
CLEAR
1
Key = Space
FF0A
FF1A
FF2A
74LS76D
74LS76D
74LS76D

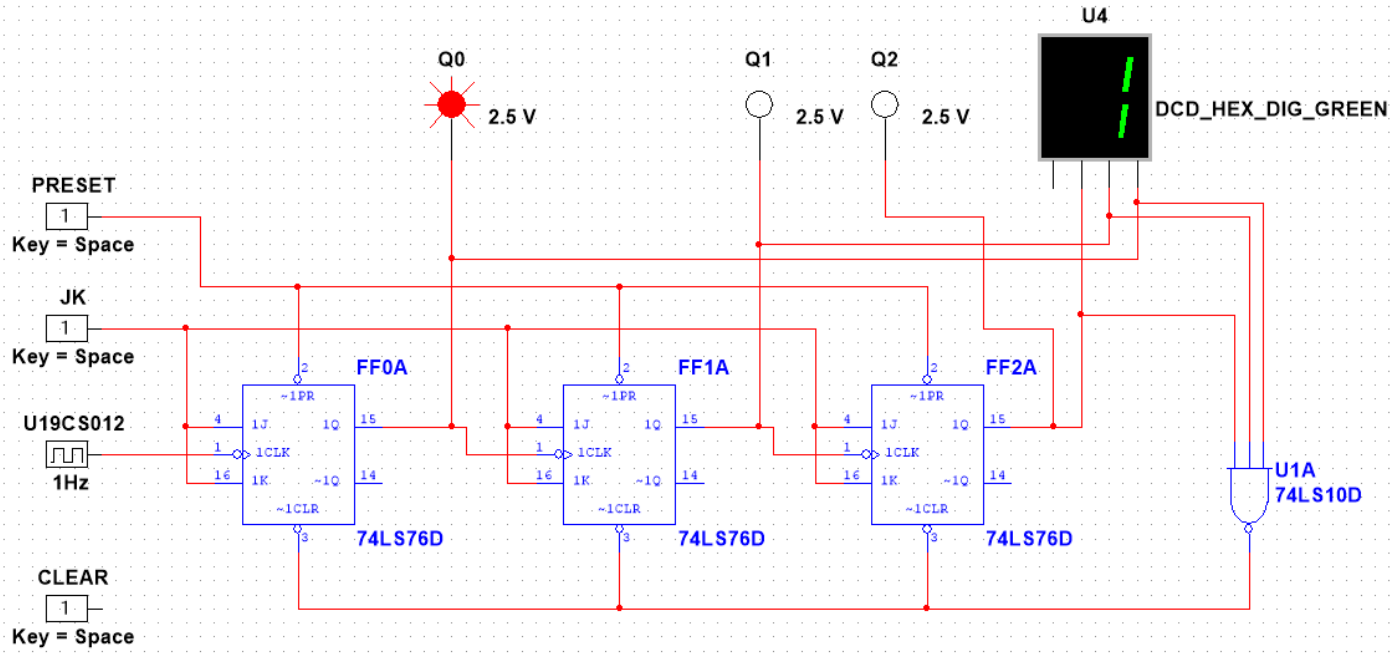

U4
Q0
Q1
Q3
2.5 V
2.5 V
2.5 V
DCD_HEX_DIG_GREEN
PRESET
1
Key = Space
JK
1
Key = Space
U19CS012
1Hz
CLEAR
1
Key = Space
FF0A
FF1A
FF2A
74LS76D
74LS76D
74LS76D

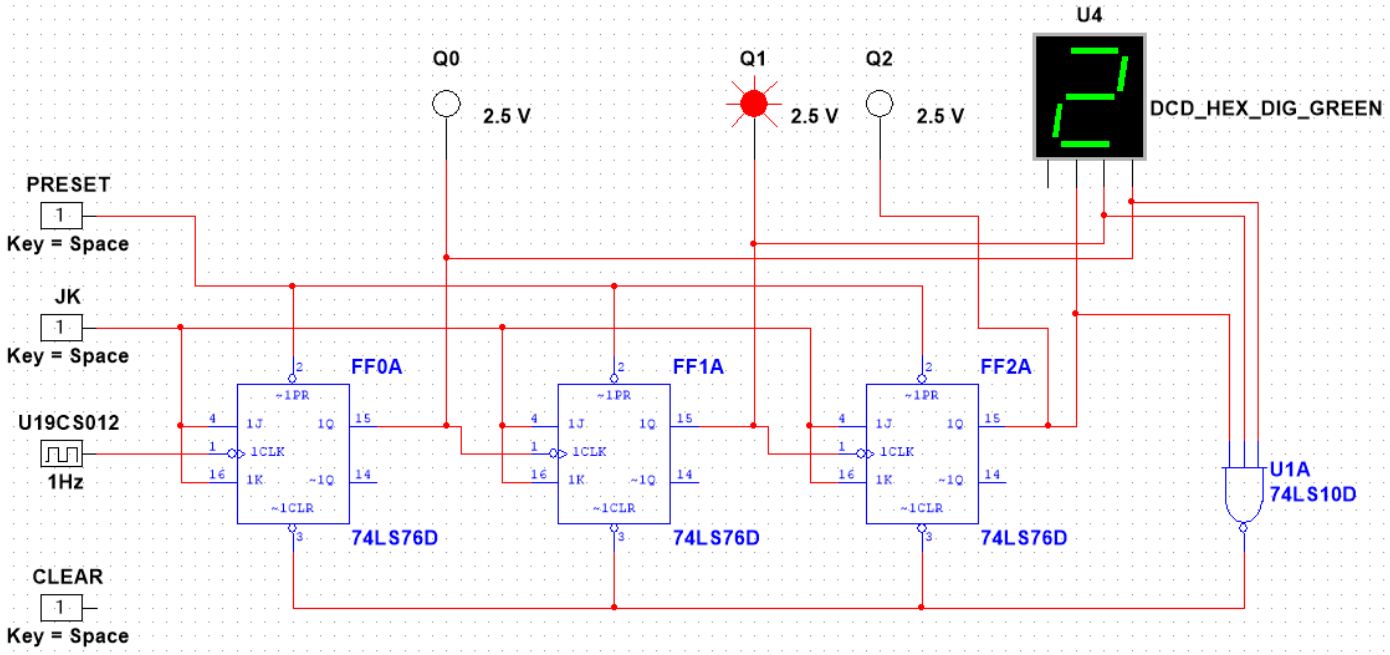


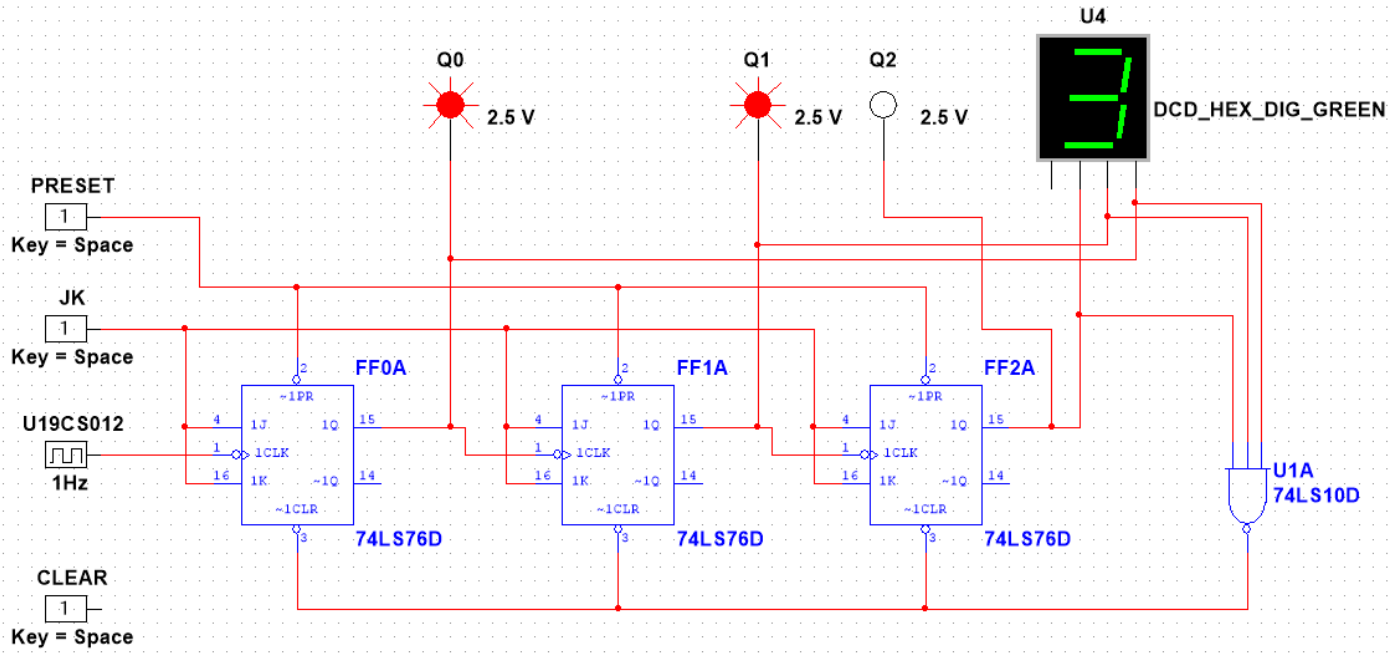
After this the Cycle Repeats Itself [0 →1→2→3→4→5→6→7→0]

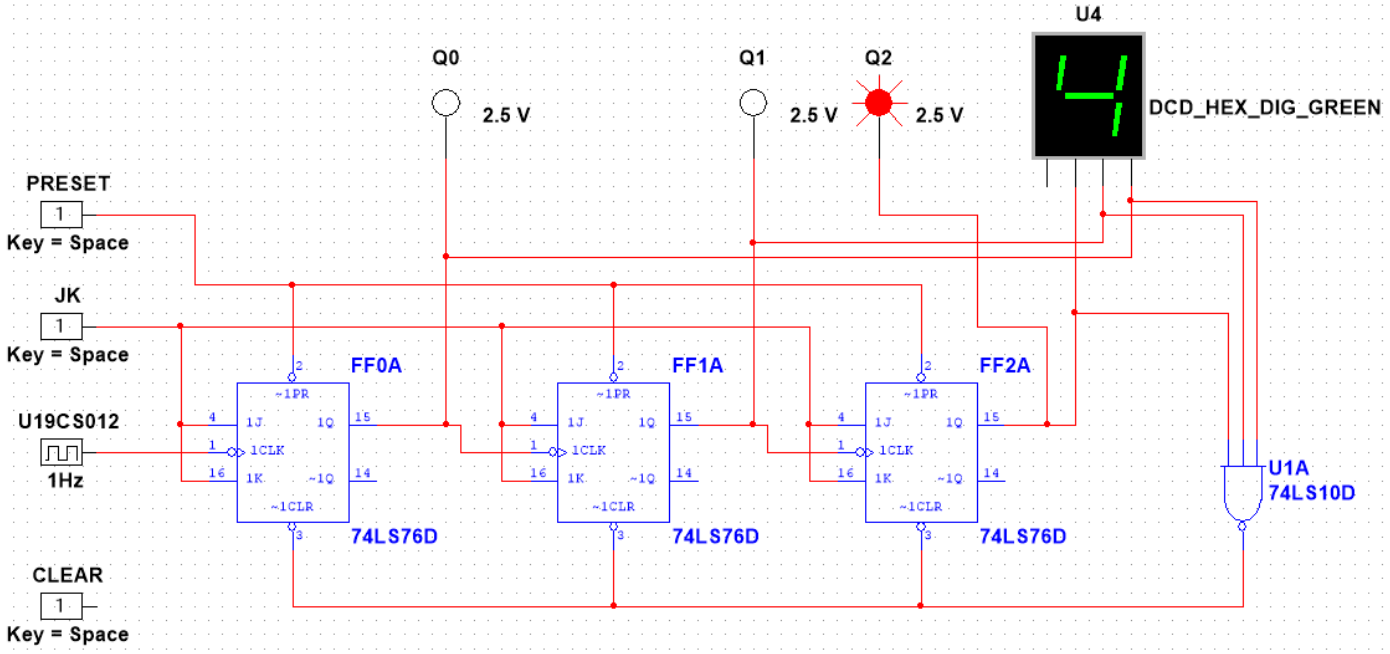
MOD-7 COUNTER

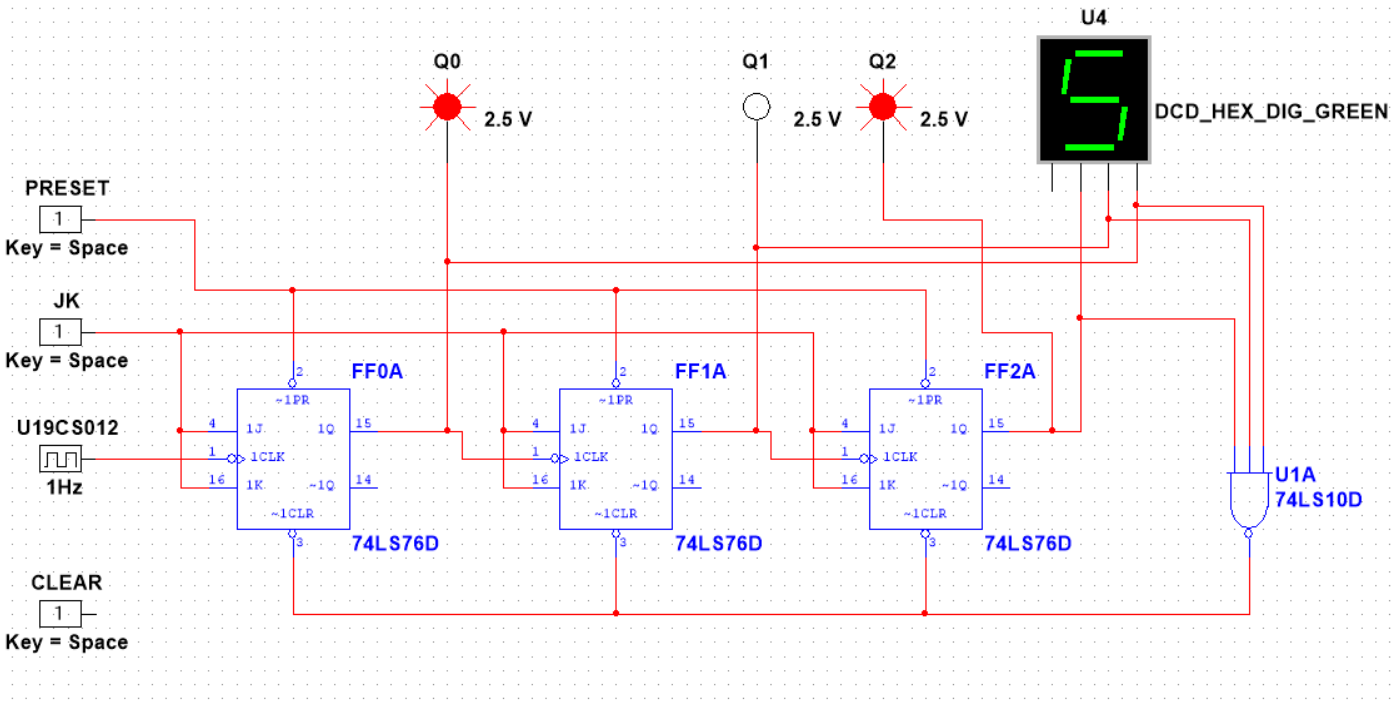


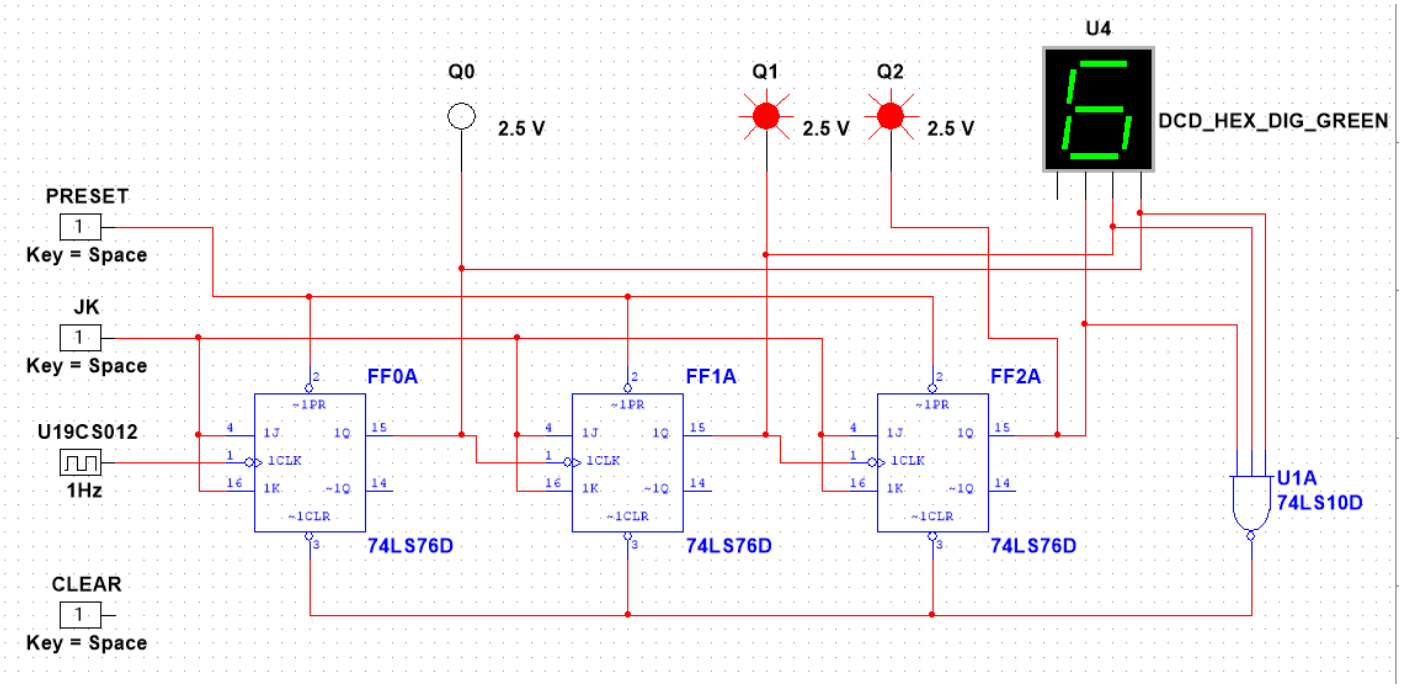

U4
Q0
Q1
Q2
2.5 V
2.5 V
2.5 V
DCD_HEX_DIG_GREEN
PRESET
Key = Space
JK
Key = Space
U19CS012
1Hz
CLEAR
Key = Space
FF0A
FF1A
FF2A
74LS76D
U1A
74LS10D

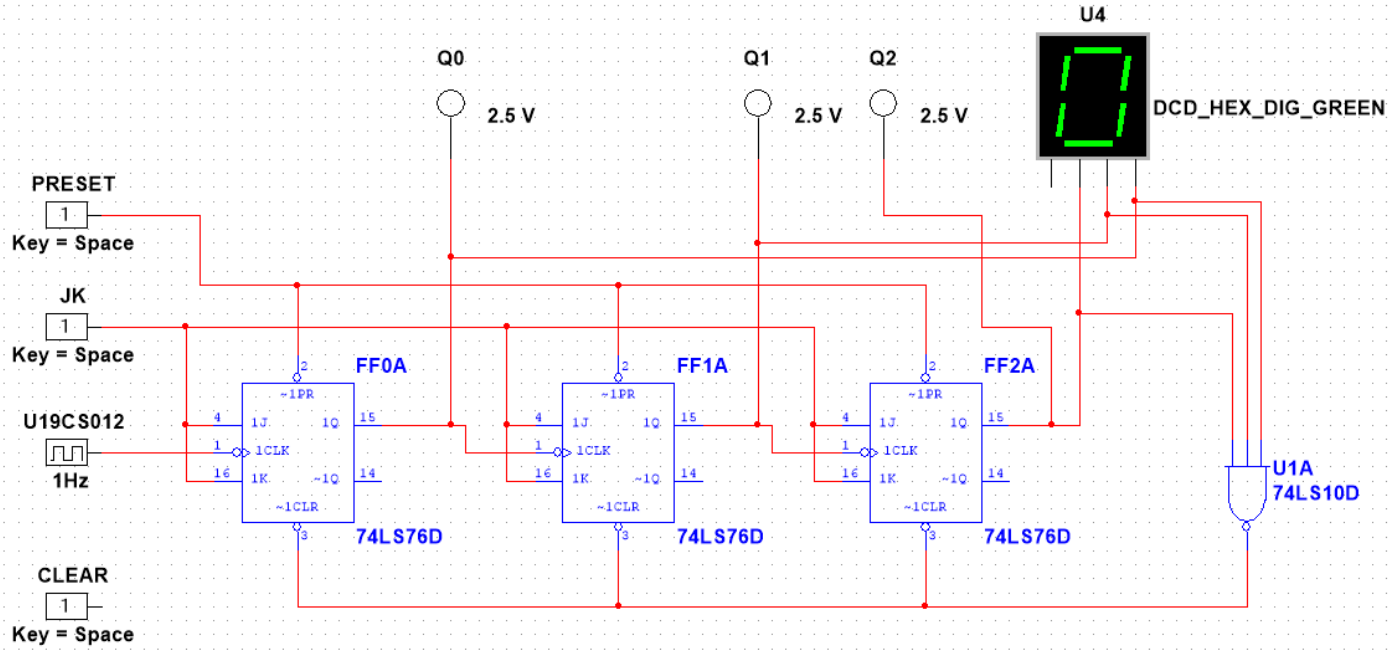

U4
Q0
Q1
Q2
2.5 V
2.5 V
2.5 V
DCD_HEX_DIG_GREEN
PRESET
Key = Space
JK
Key = Space
U19CS012
1Hz
CLEAR
Key = Space
FF0A
FF1A
FF2A
74LS76D
U1A
74LS10D


U4
Q0
2.5 V
Q1
2.5 V
Q2
2.5 V
DCD_HEX_DIG_GREEN
PRESET
Key = Space
JK
Key = Space
U19CS012
1Hz
CLEAR
Key = Space
FF0A
FF1A
FF2A
74LS76D
U1A
74LS10D


U4
Q0
2.5 V
Q1
2.5 V
Q2
2.5 V
DCD_HEX_DIG_GREEN
PRESET
Key = Space
JK
Key = Space
U19CS012
1Hz
CLEAR
Key = Space
FF0A
FF1A
FF2A
74LS76D
U1A
74LS10D

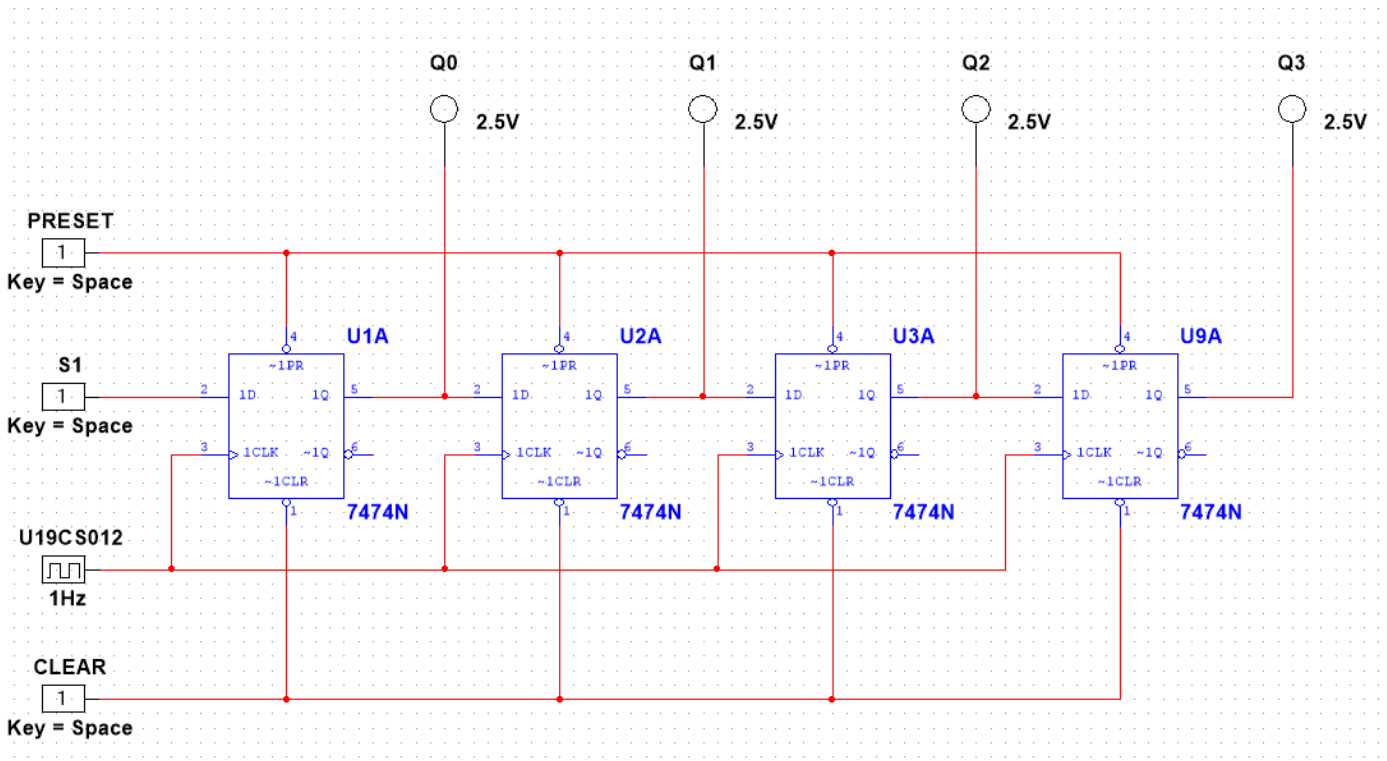

U4
Q0
Q1
Q2
2.5 V
2.5 V
2.5 V
DCD_HEX_DIG_GREEN
PRESET
1
Key = Space
JK
1
Key = Space
U19CS012
1Hz
CLEAR
1
Key = Space
FF0A
FF1A
FF2A
74LS76D
74LS76D
74LS76D
U1A
74LS10D

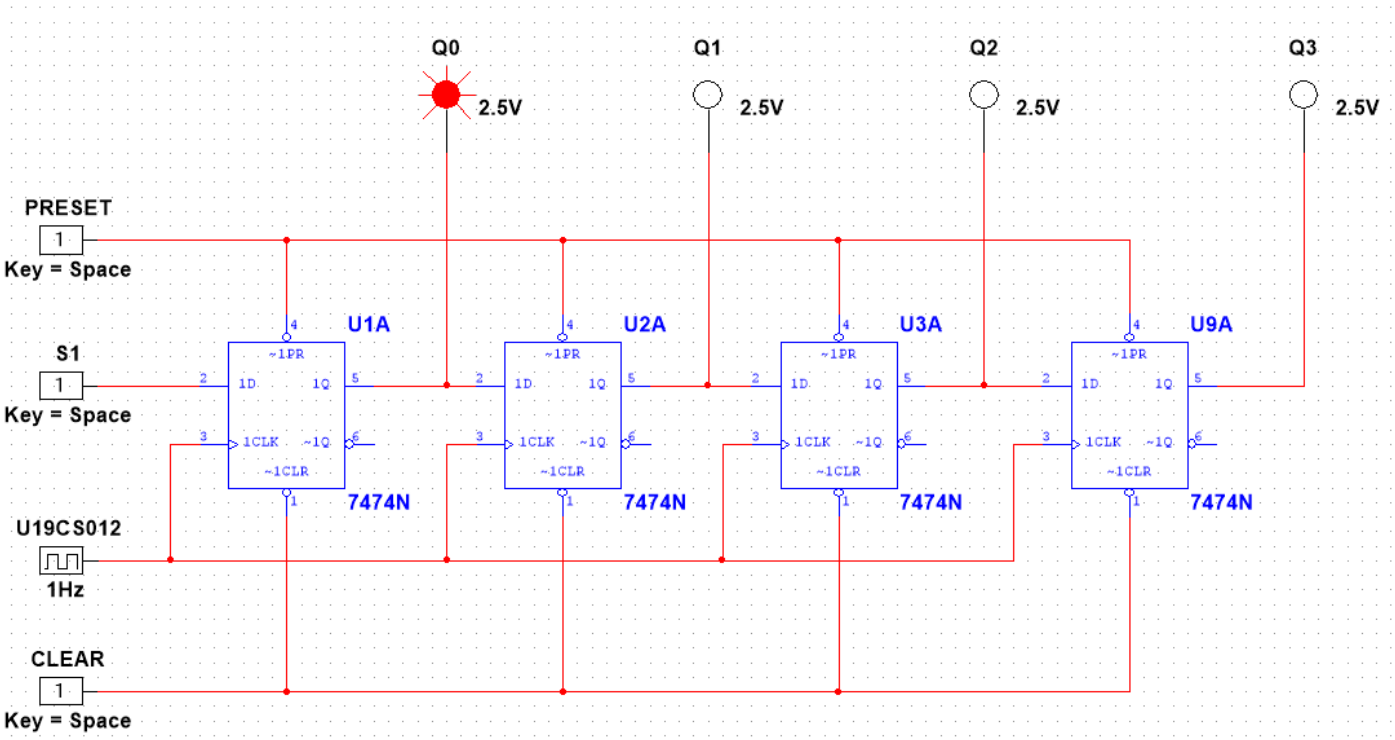

U4
Q0
Q1
Q2
2.5 V
2.5 V
2.5 V
DCD_HEX_DIG_GREEN
PRESET
1
Key = Space
JK
1
Key = Space
U19CS012
1Hz
CLEAR
1
Key = Space
FF0A
FF1A
FF2A
74LS76D
74LS76D
74LS76D
U1A
74LS10D

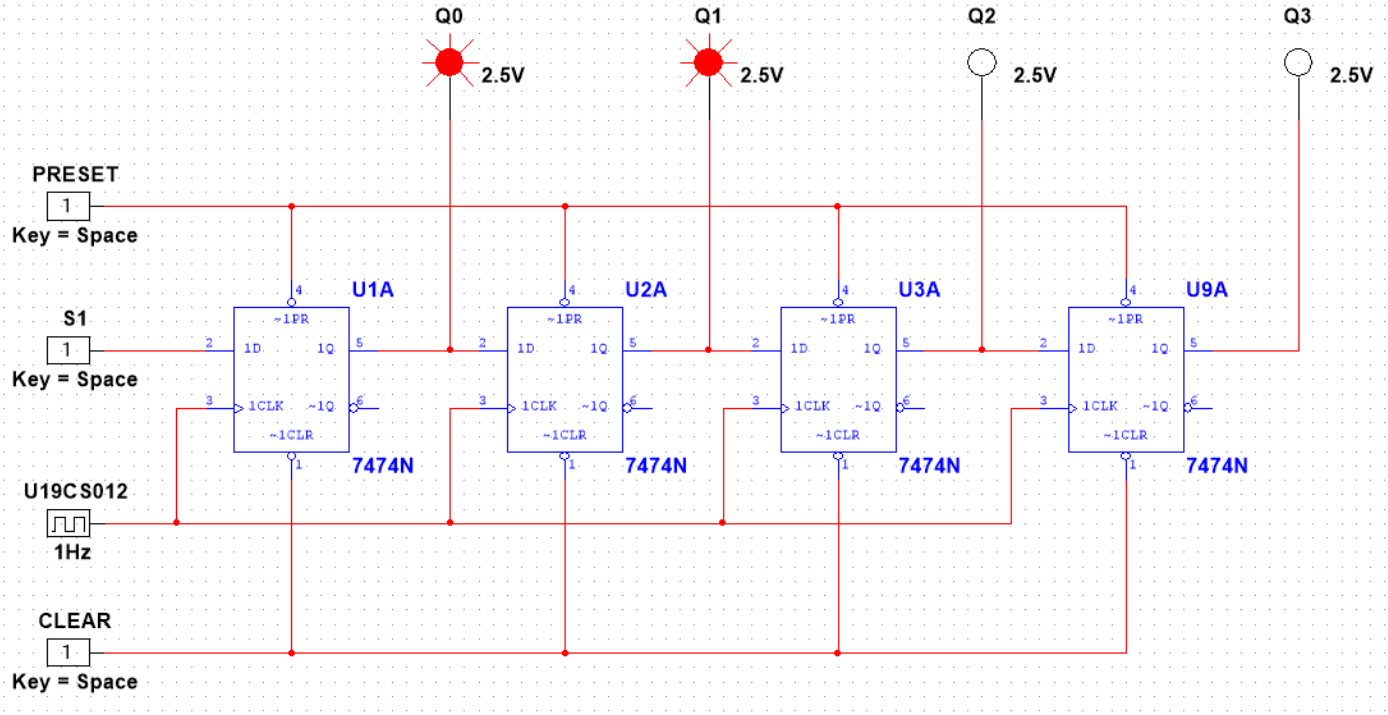


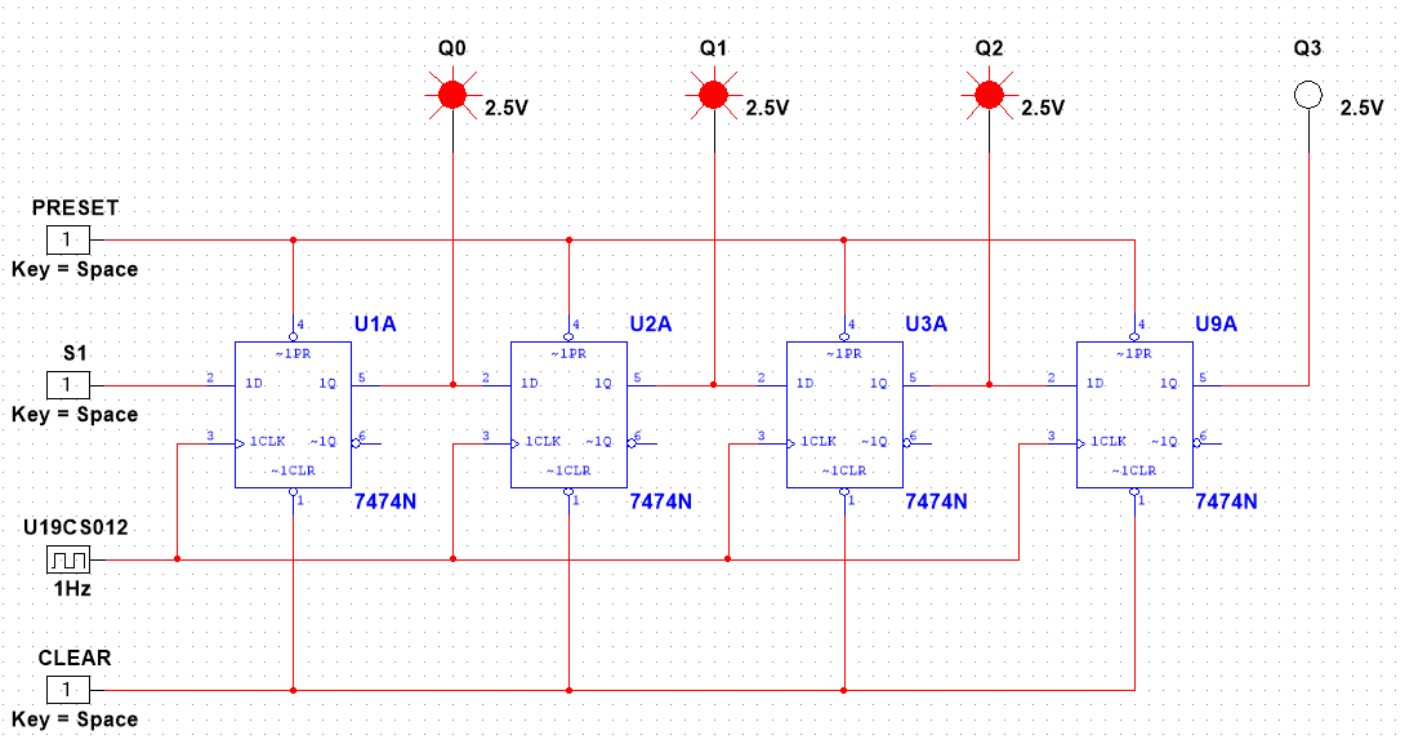
After this the Cycle Repeats Itself [0→1→2→3→4→5→6→0]

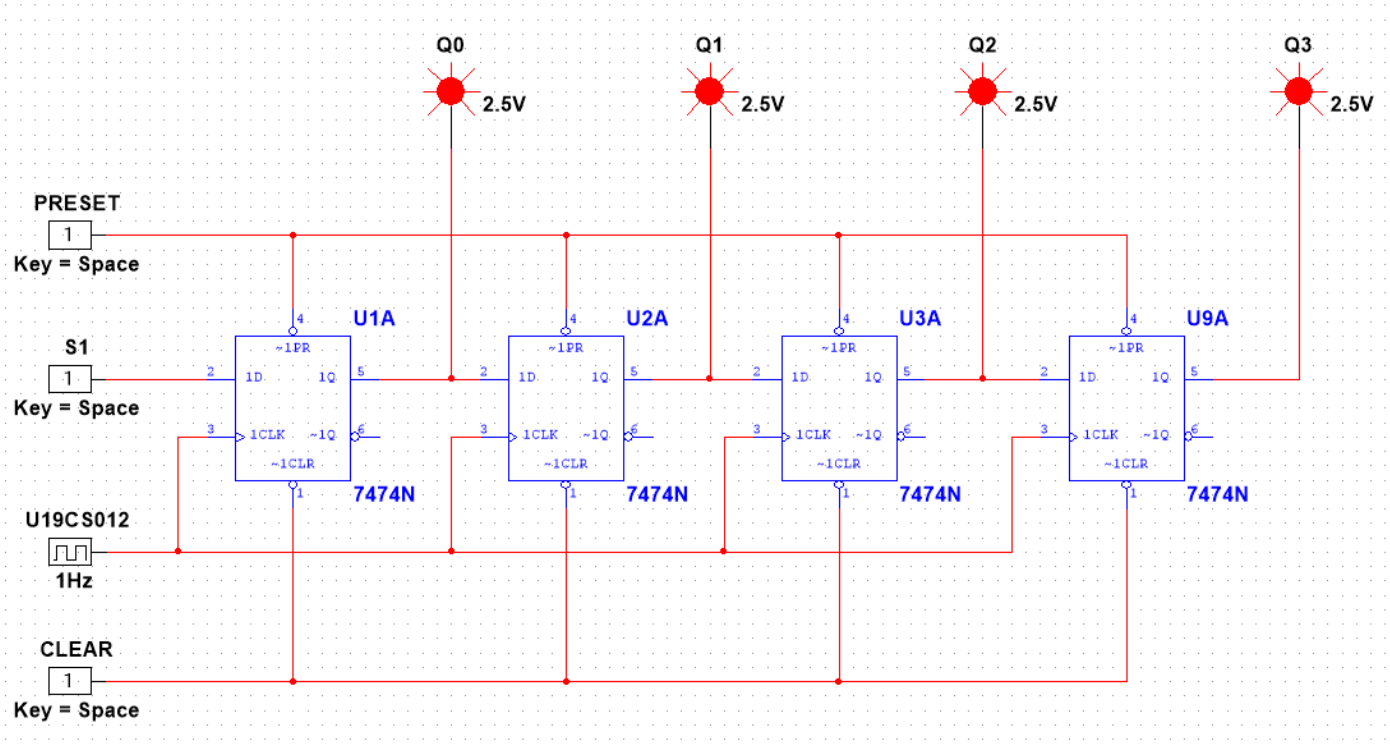
SHIFT RIGHT REGISTER



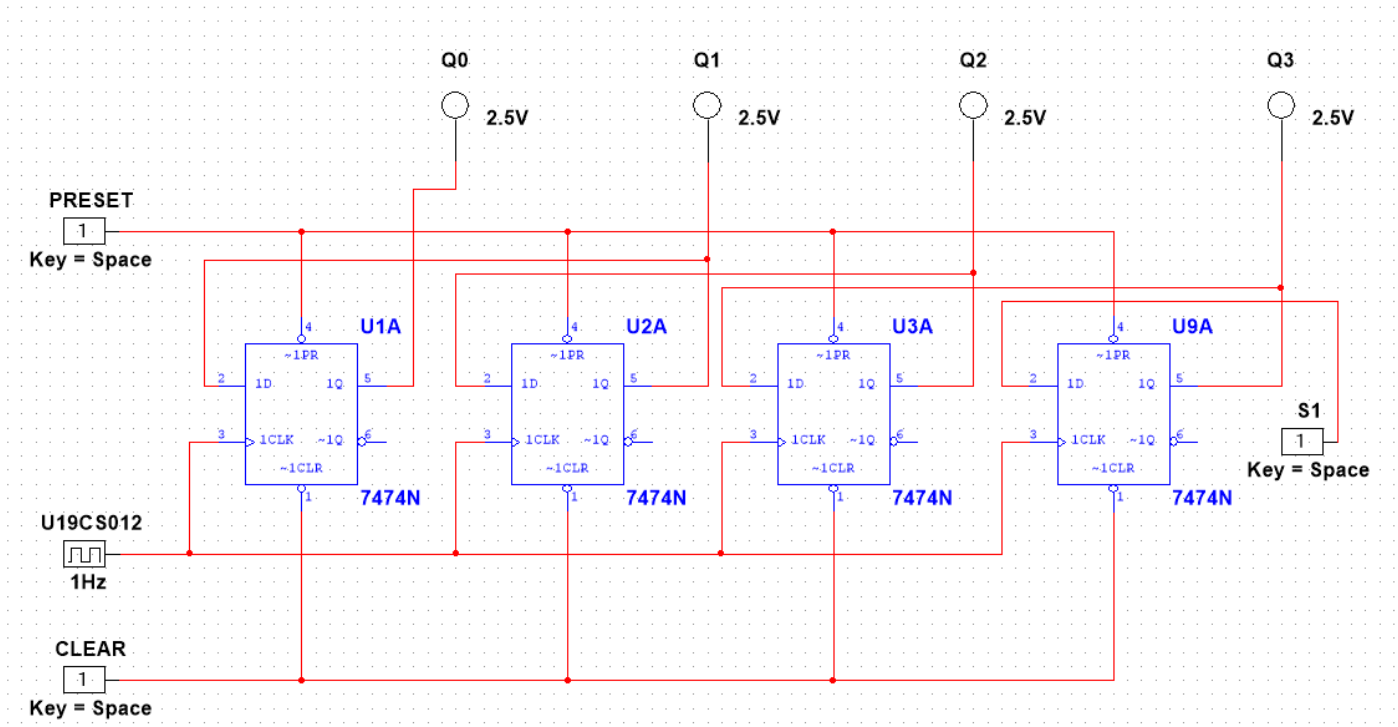

Q0
Q1
Q2
Q3
2.5V
2.5V
2.5V
2.5V
PRESET
1
Key = Space
S1
1
Key = Space
U19CS012
1Hz
CLEAR
1
Key = Space
U1A
U2A
U3A
U9A
7474N
7474N
7474N
7474N

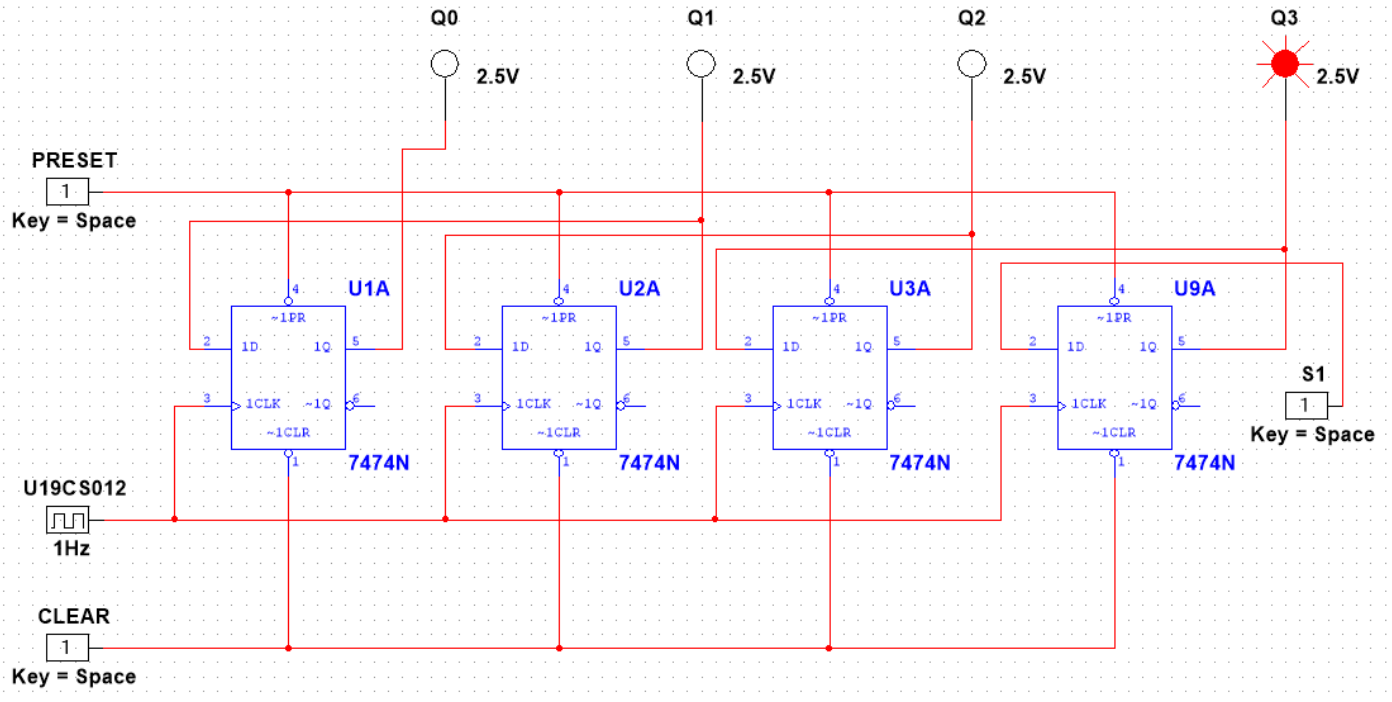

Q0
Q1
Q2
Q3
2.5V
2.5V
2.5V
2.5V
PRESET
1
Key = Space
S1
1
Key = Space
U19CS012
1Hz
CLEAR
1
Key = Space
U1A
U2A
U3A
U9A
7474N
7474N
7474N
7474N

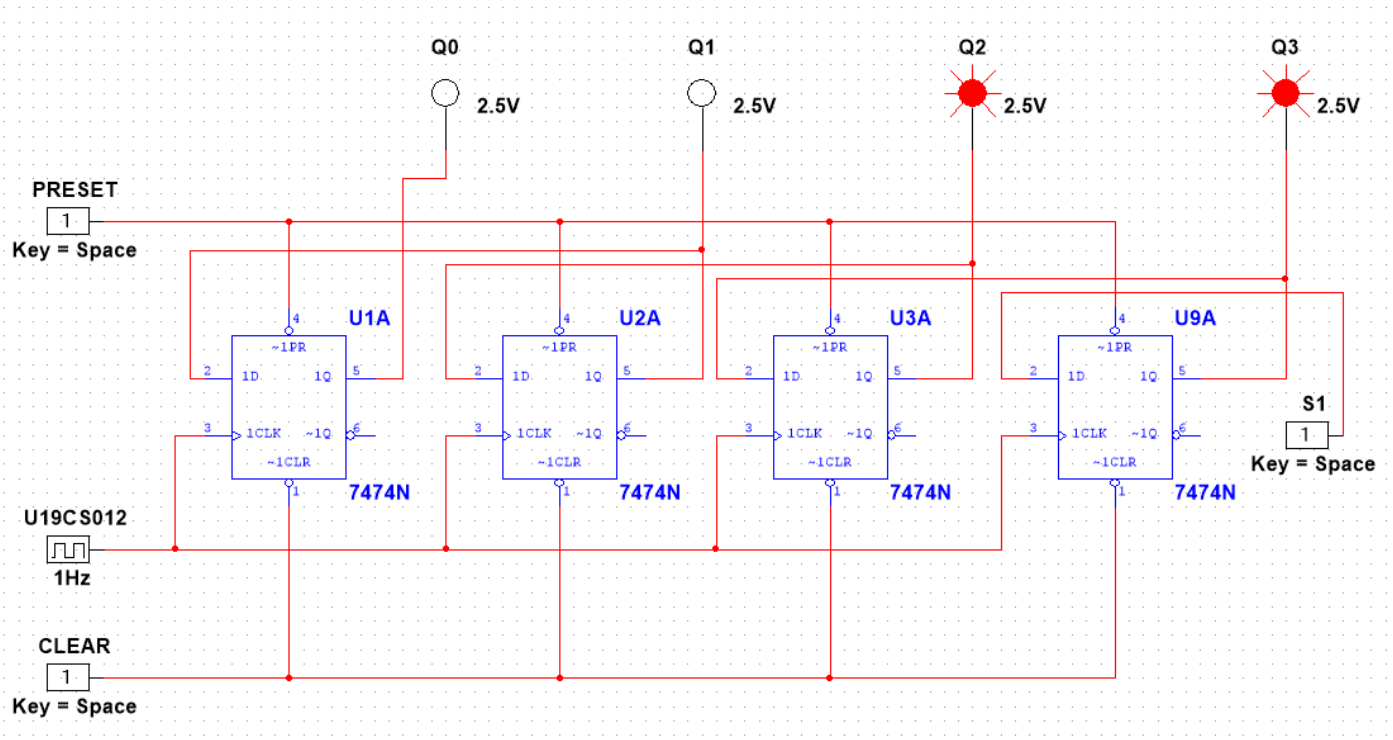

Q0
Q1
Q2
Q3
2.5V
2.5V
2.5V
2.5V
PRESET
1
Key = Space
S1
1
Key = Space
U19CS012
1Hz
CLEAR
1
Key = Space
U1A
U2A
U3A
U9A
~1PR
1D
1Q
1CLK
~1Q
~1CLR
7474N

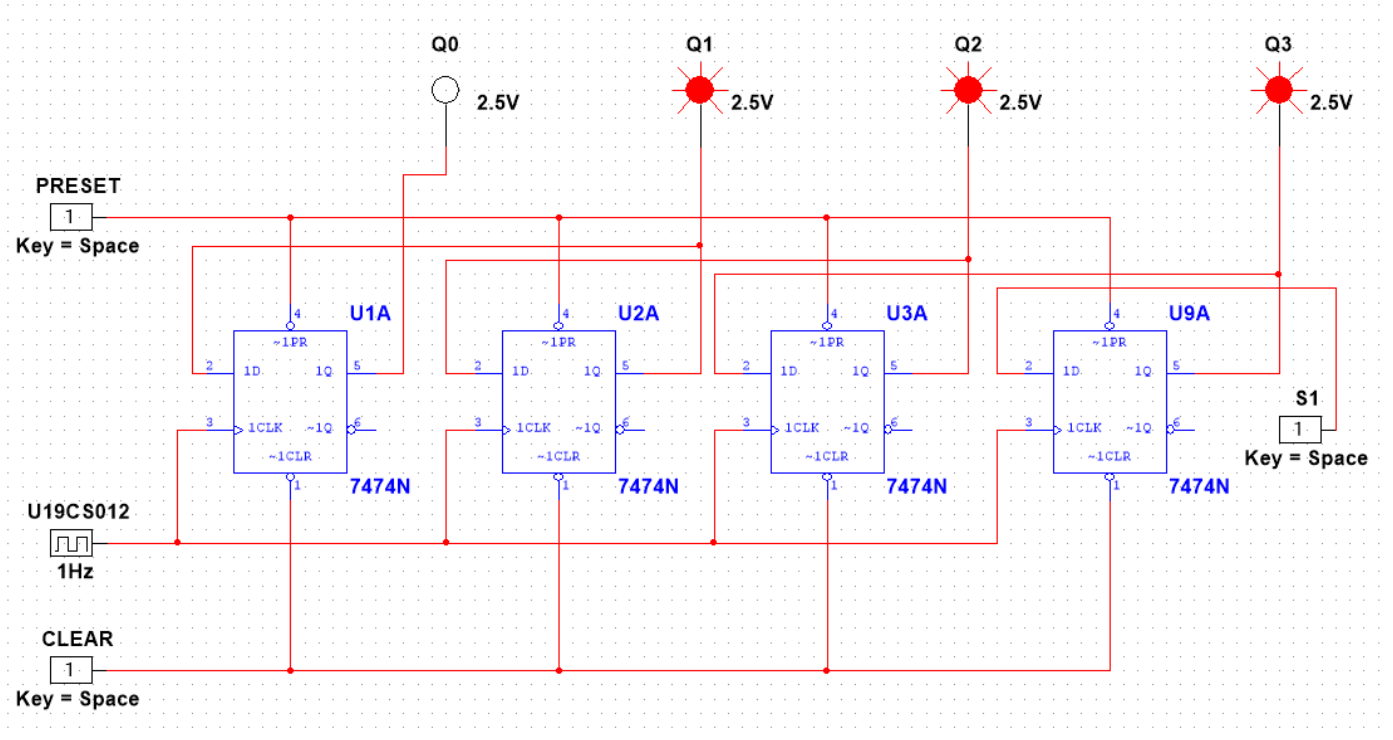

Q0
Q1
Q2
Q3
2.5V
2.5V
2.5V
2.5V
PRESET
1
Key = Space
S1
1
Key = Space
U19CS012
1Hz
CLEAR
1
Key = Space
U1A
U2A
U3A
U9A
~1PR
1D
1Q
1CLK
~1Q
~1CLR
7474N

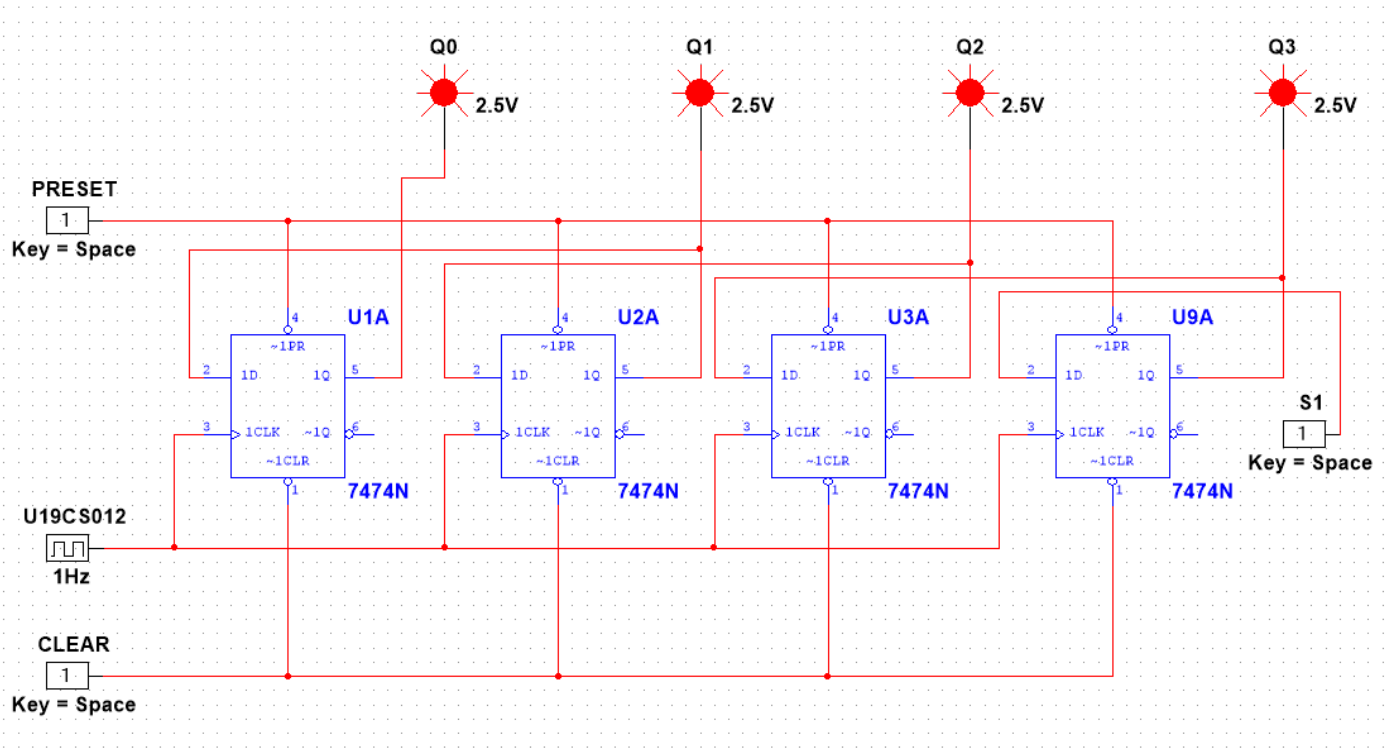
SHIFT LEFT REGISTER






Q0
2.5V
Q1
2.5V
Q2
2.5V
Q3
2.5V
PRESET
1
Key = Space
U1A
U2A
U3A
U9A
7474N
S1
1
Key = Space
U19CS012
1Hz
CLEAR
1
Key = Space


Q0
2.5V
Q1
2.5V
Q2
2.5V
Q3
2.5V
PRESET
1
Key = Space
U1A
U2A
U3A
U9A
7474N
S1
1
Key = Space
U19CS012
1Hz
CLEAR
1
Key = Space



CONCLUSIONS

1.) In this Experiment, We have studied about Registers and Counters like 3-Bit Full Modulus and 7-Modulus Circuit and also implemented it successfully on Multisim.

2.) We **Verified** the Theoretical Knowledge of Shift Registers [Left and Right Shift] by *Observing the Bit Movement in Shift Registers*.

3.) We also Implemented 3-Bit Up Counter, Mod-7 Counter, 4-Bit Shift Right Register and 4-Bit Shift Left Register using Multisim and Observed its Working.

ASSIGNMENT-11

U19CS012

1.) Design and Implement *MOD-12 Counter* in Multisim using JK Flip-Flops.

A.) Solution:

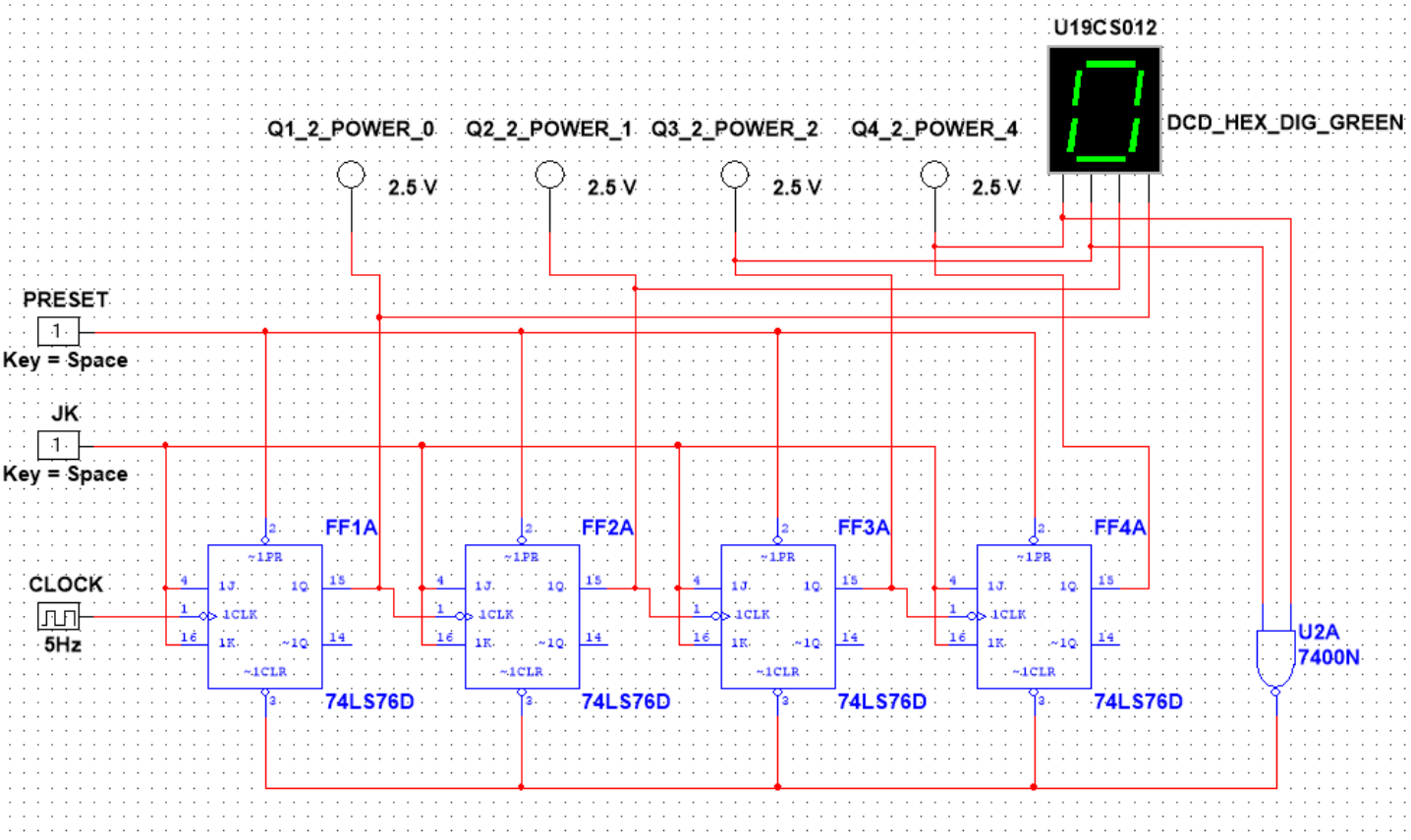
A MOD-12 Counter would be having **12** Valid States from 0 [0000] to 11 [1011].

We will make 4-Bit Full Counter [0-15] and Try to Reduce Valid States to **12**[0-11] using Additional NAND Gate that will take *Clear All Bits* as the Value in Counter turns out to be 12 [1100] i.e. When *First Two Bits Become One*, we will Clear all Bits.

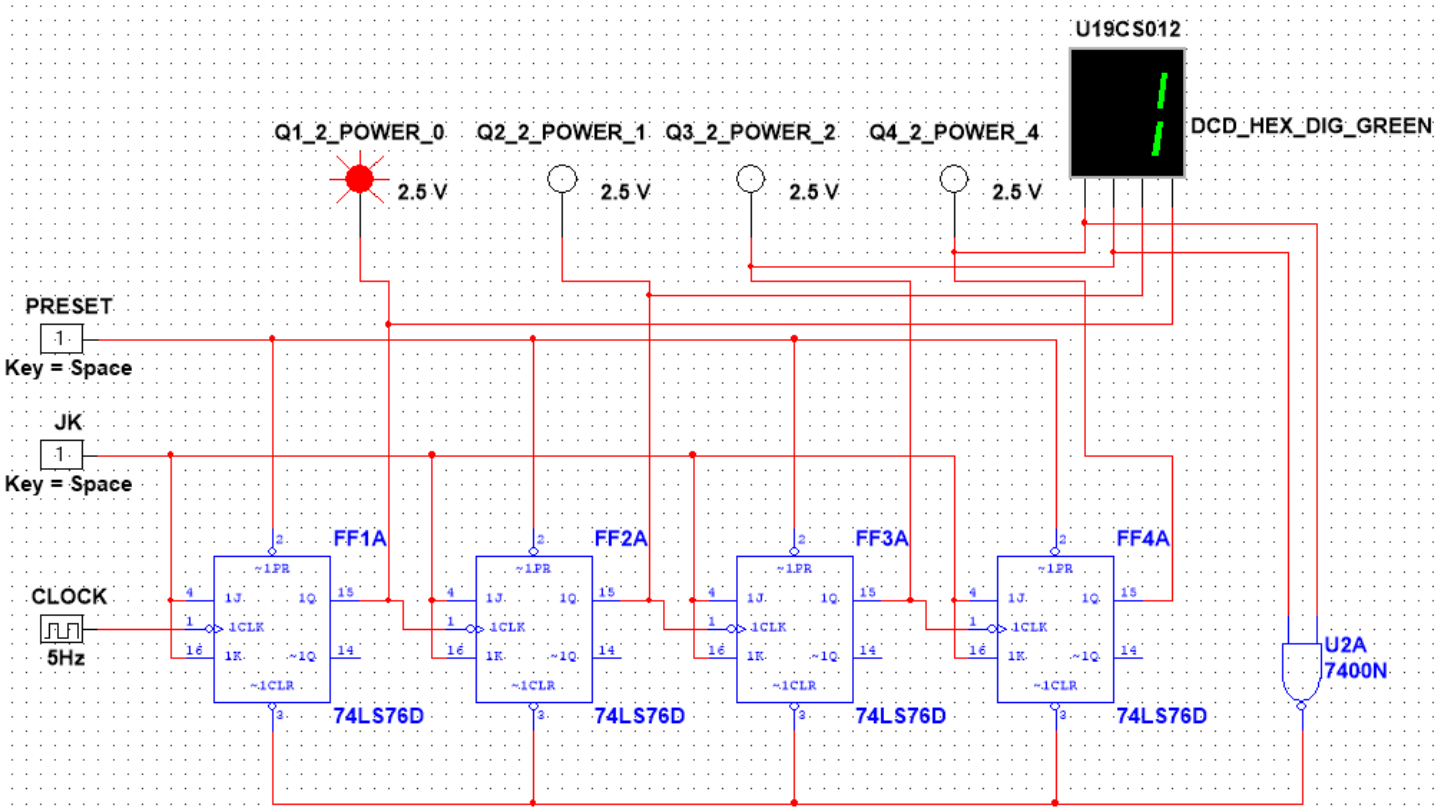
The Concept is Similar to MOD-7 Counter Implemented Earlier in Practical 11.

B.) Implementation:

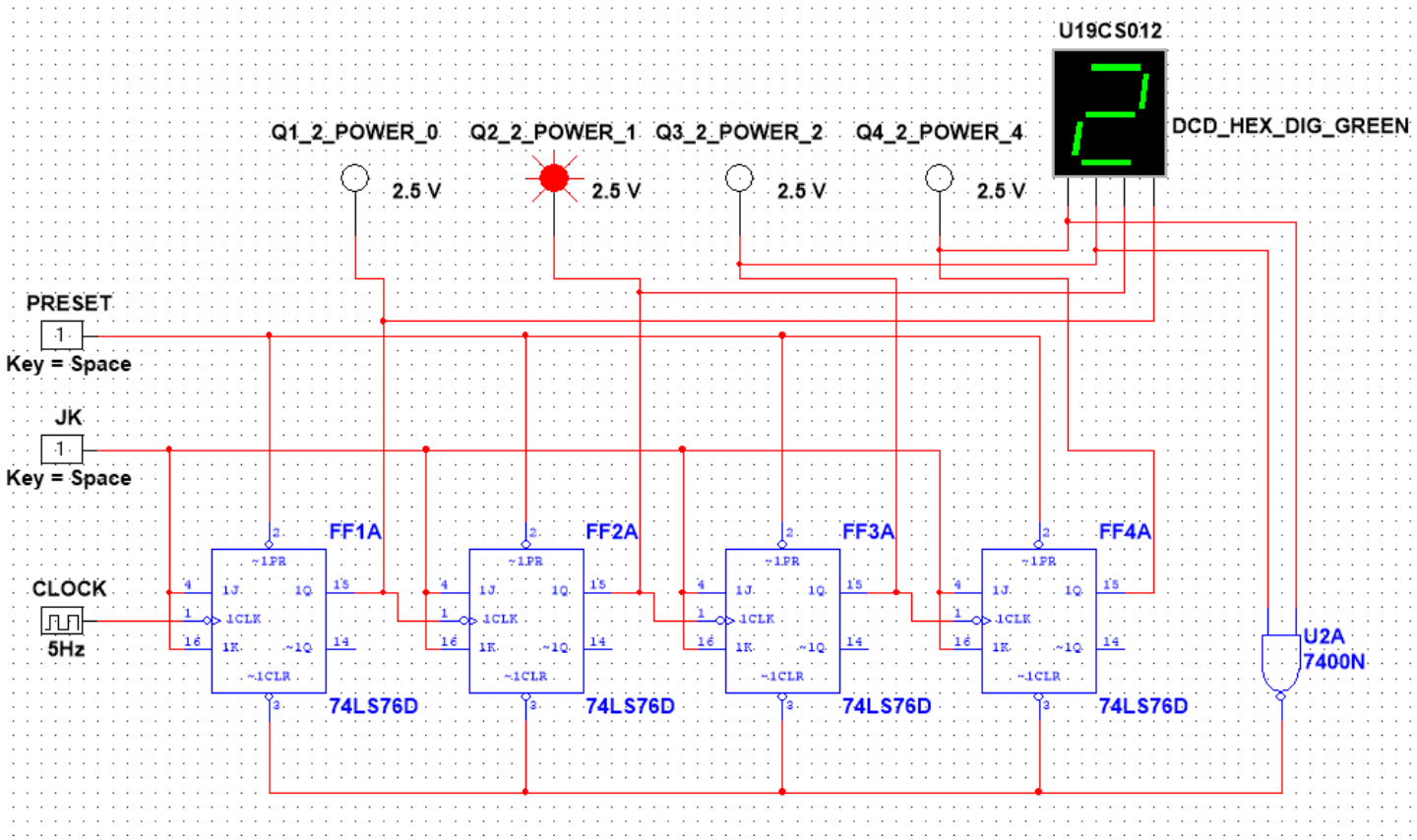
1.) Valid State: 0 [0000]



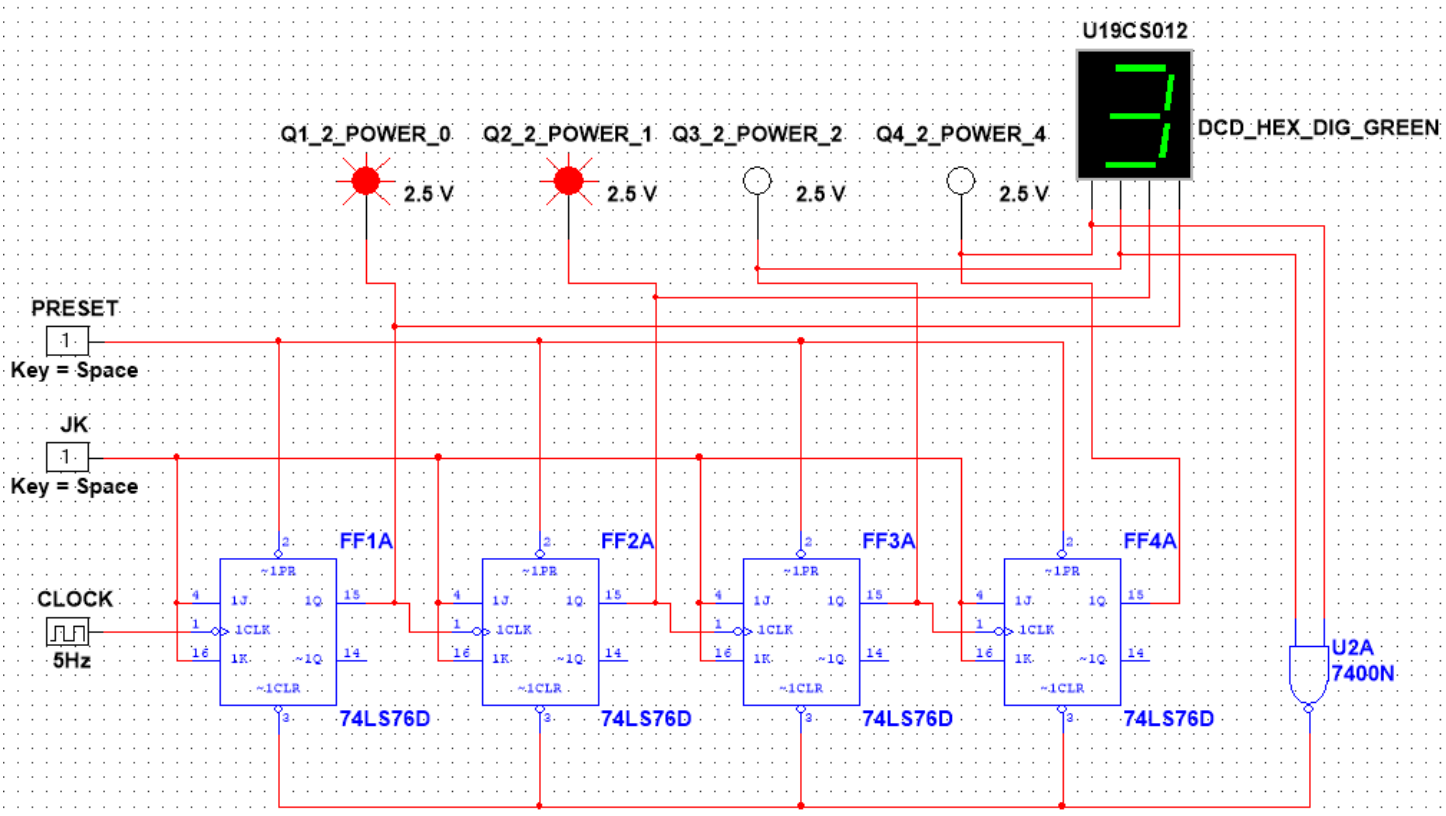
2.) Valid State: 1 [0001]



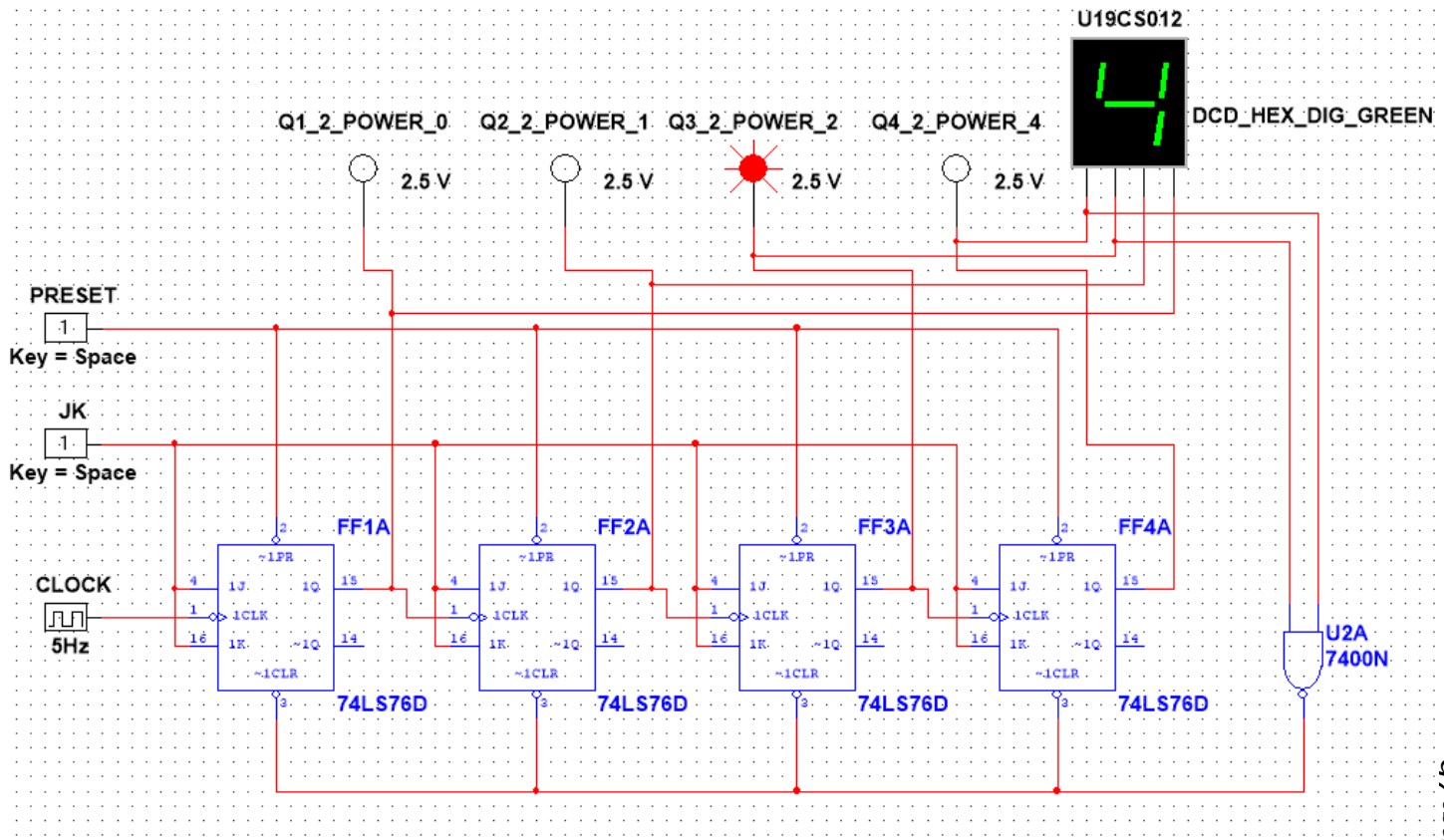
3.) Valid State: 2 [0010]



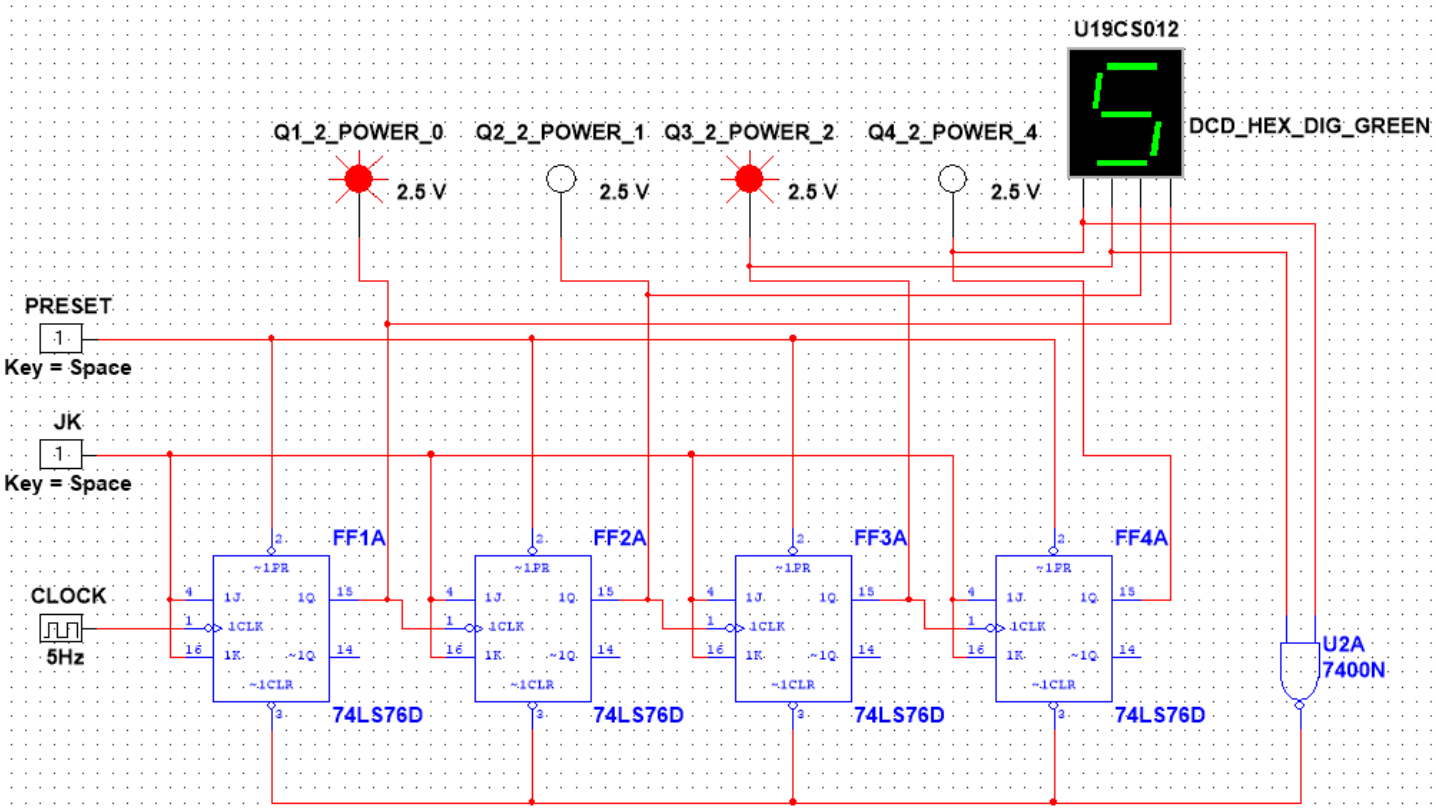
4.) Valid State: 3 [0011]



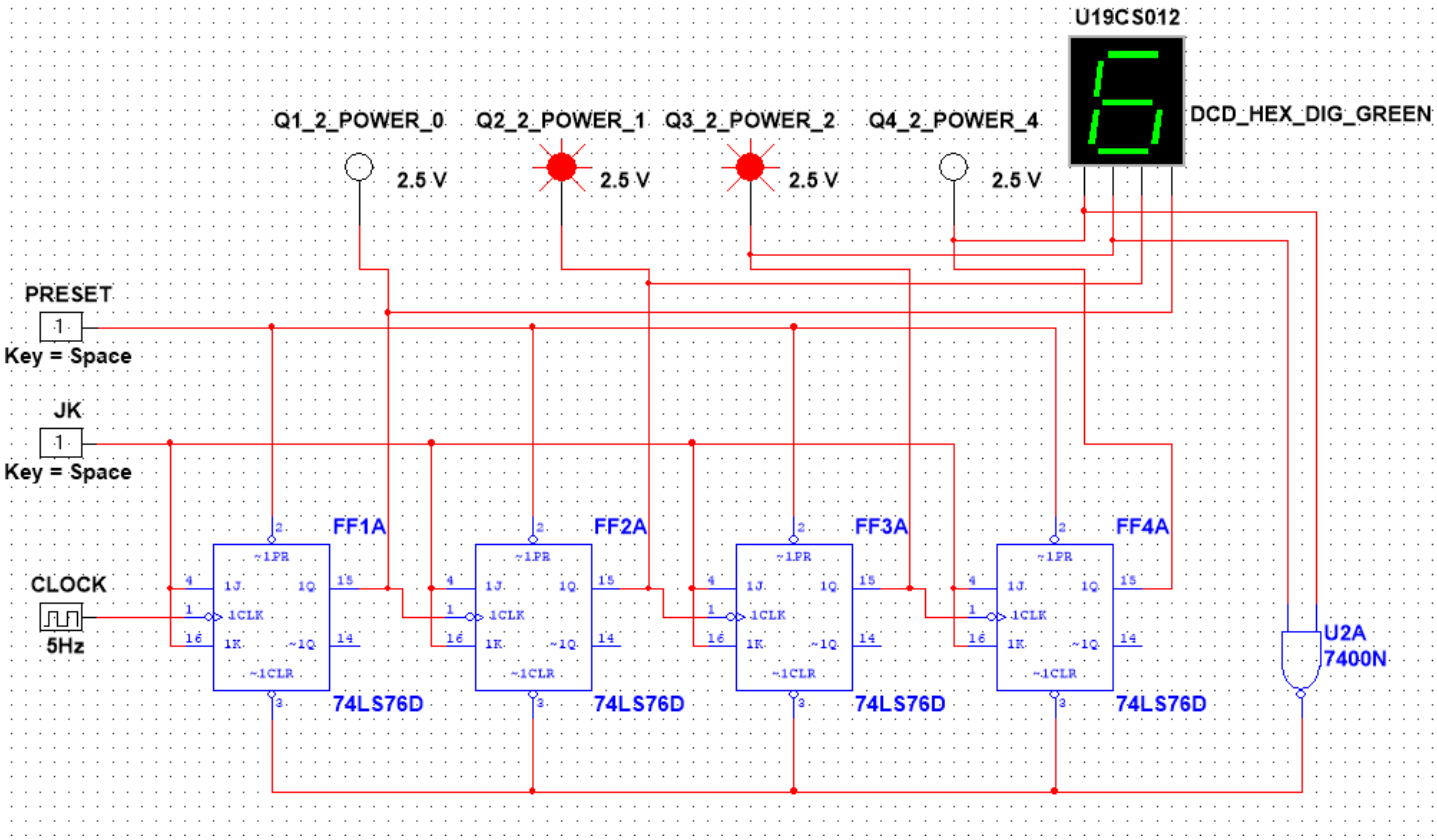
5.) Valid State: 4 [0100]



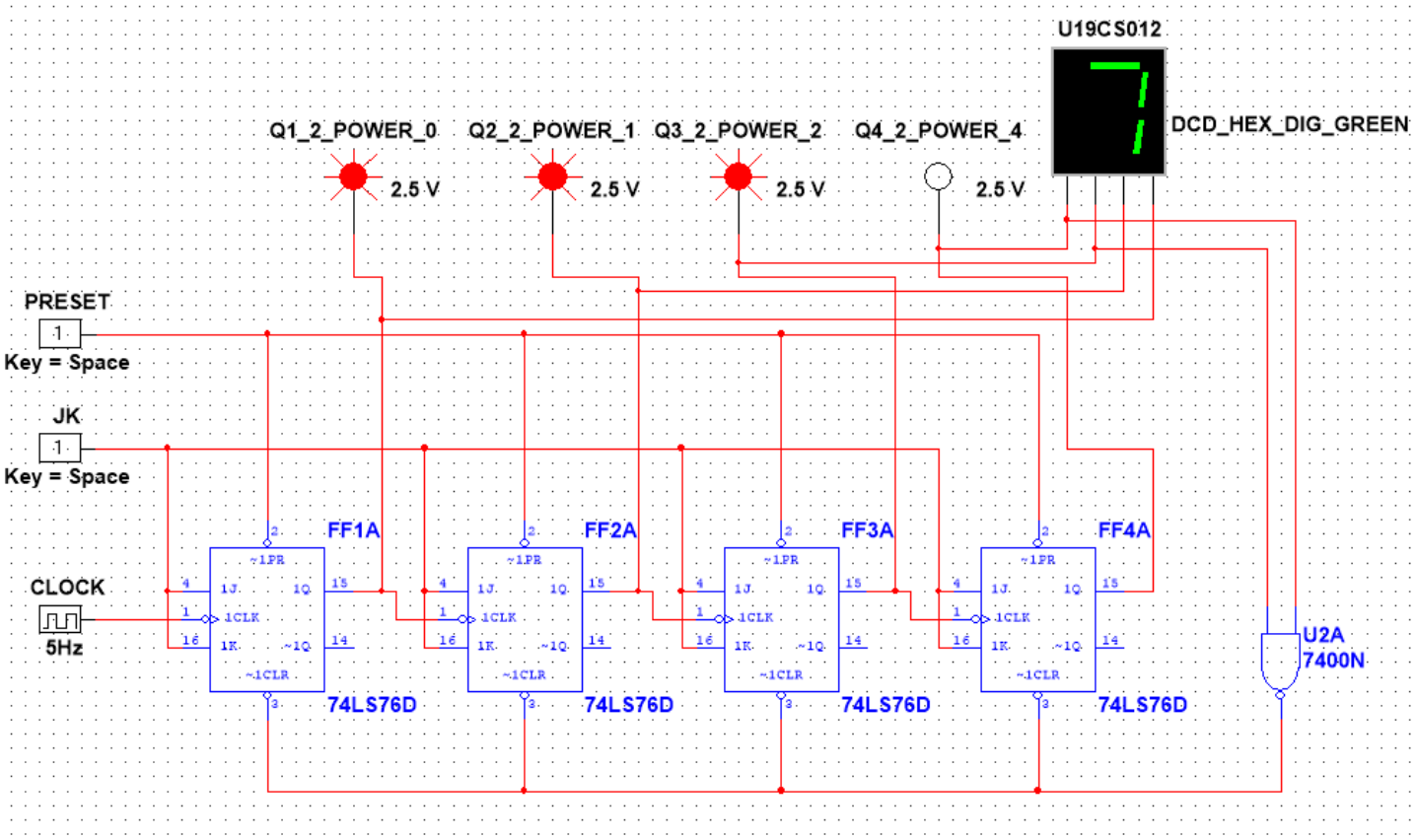
6.) Valid State: 5 [0101]



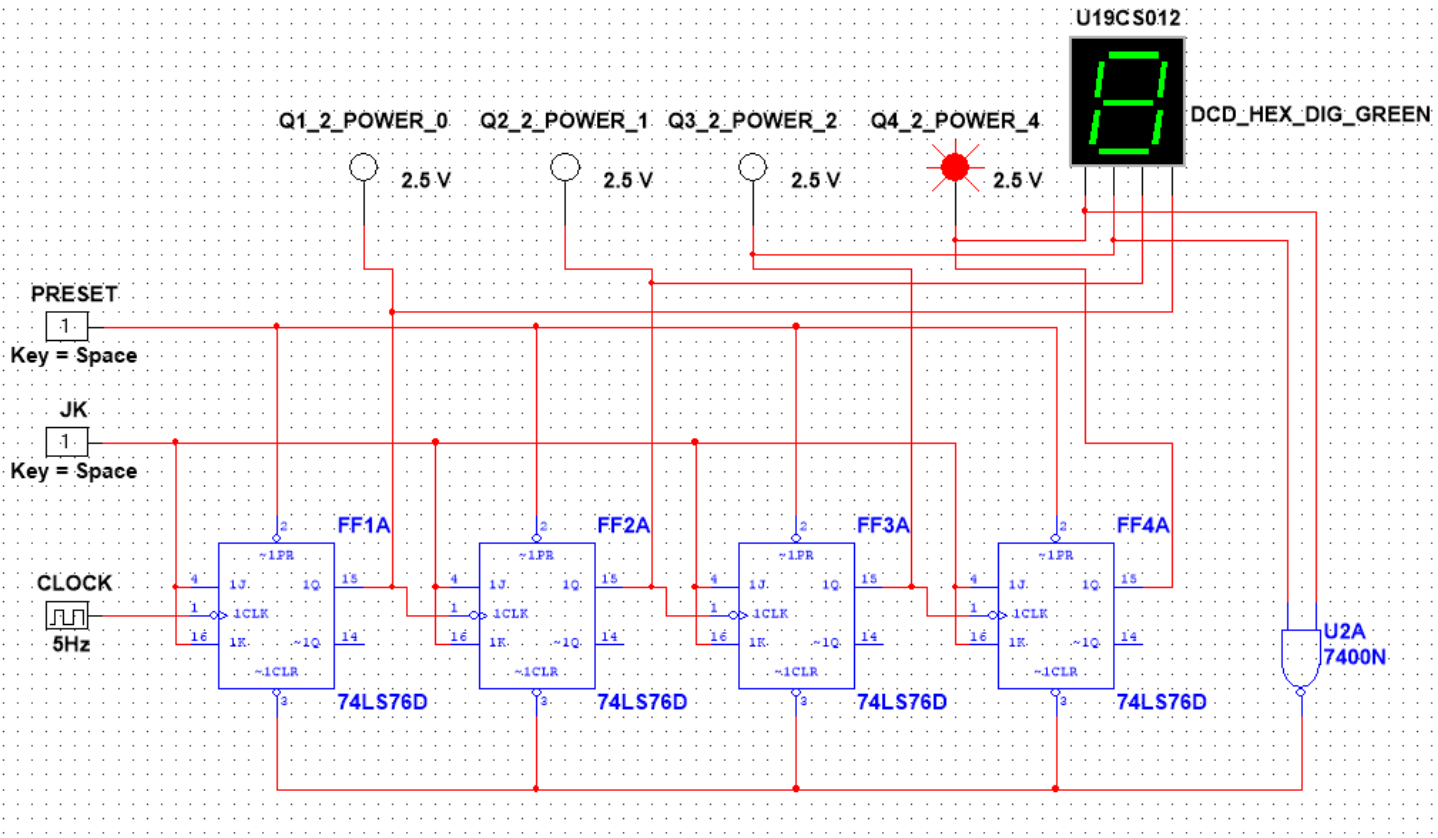
7.) Valid State: 6 [0110]



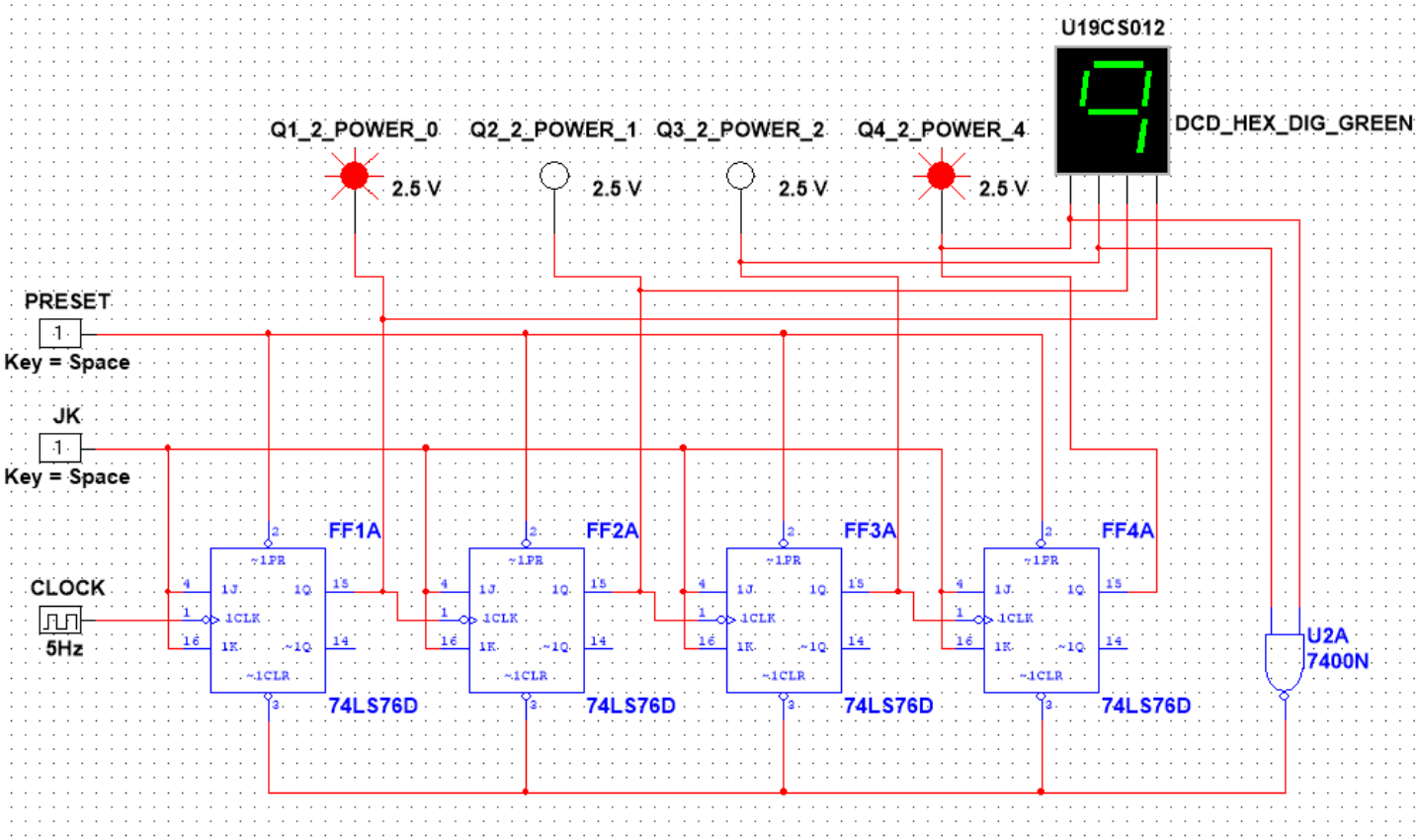
8.) Valid State: 7 [0111]



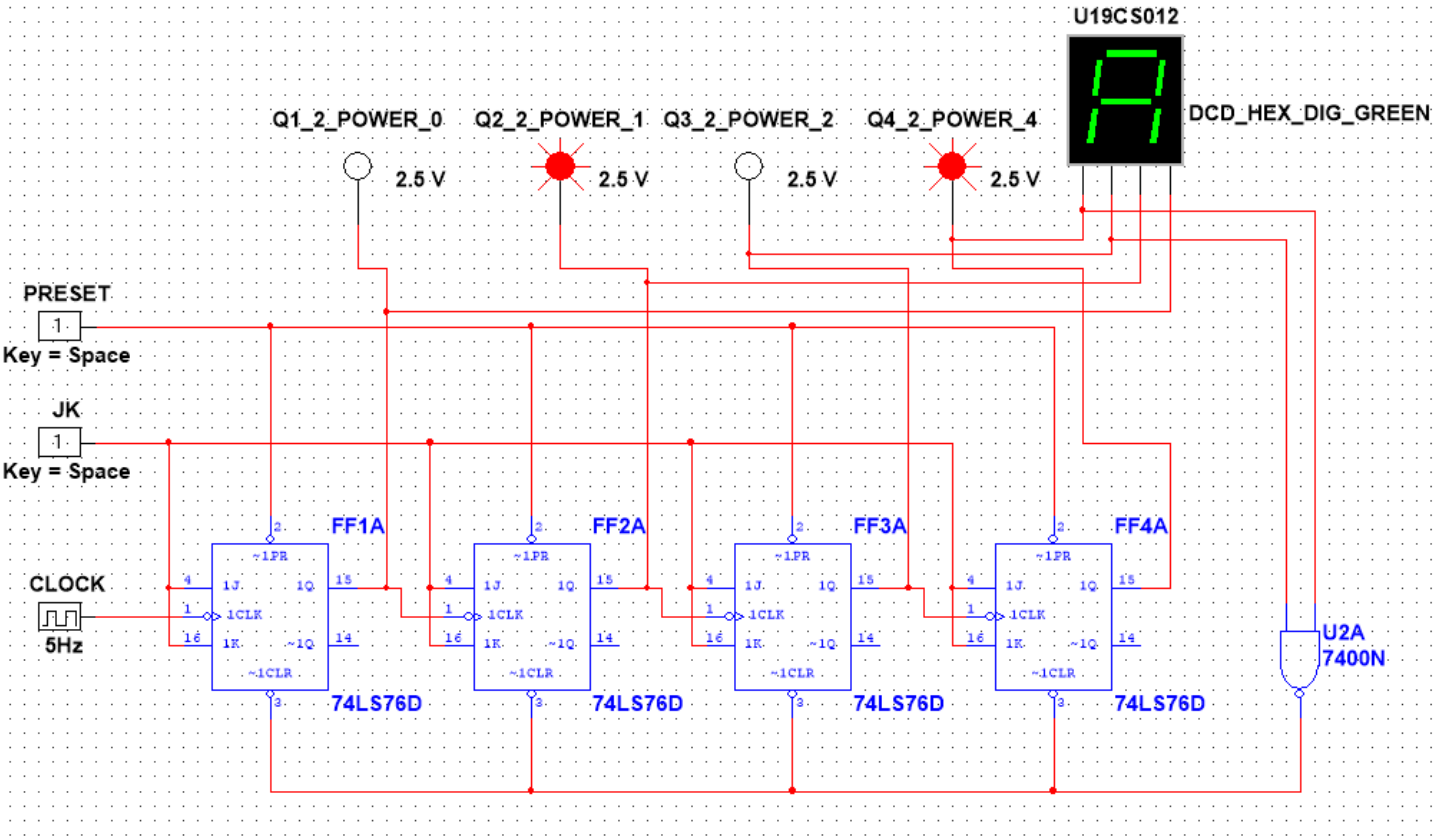
9.) Valid State: 8 [1000]



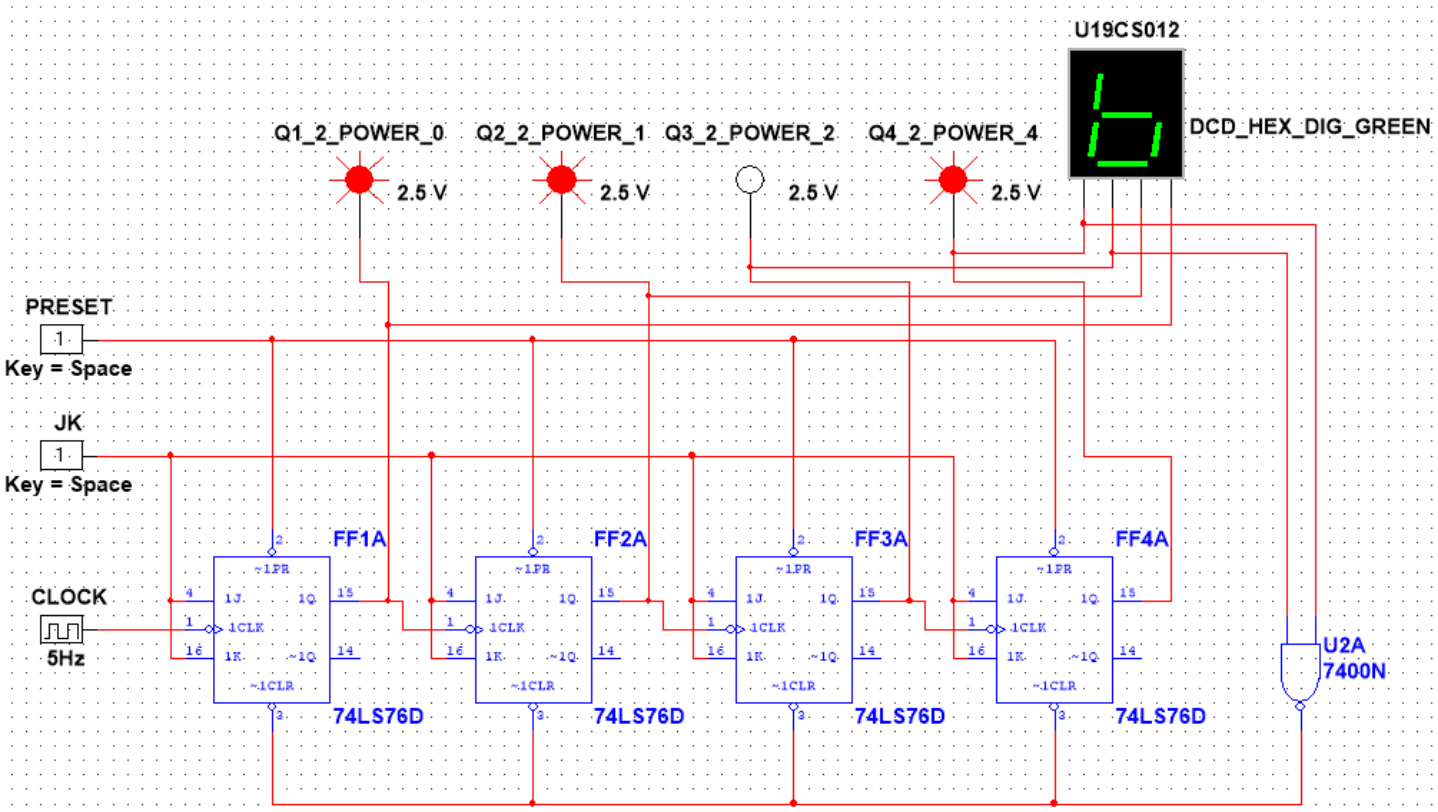
10.) Valid State: 9 [1001]



11.) Valid State: 10 [1010]

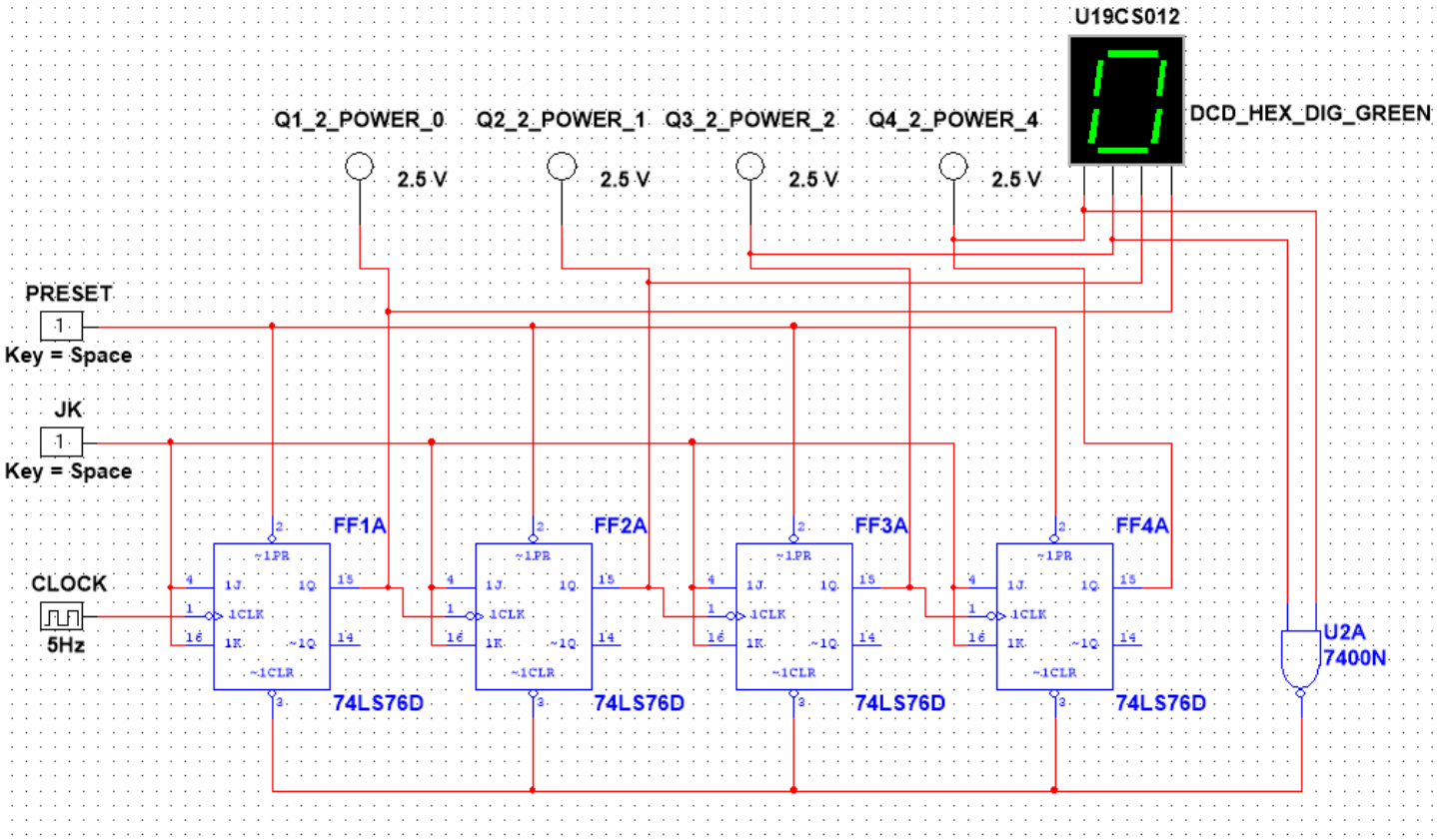


12.) Valid State: 11 [1011]

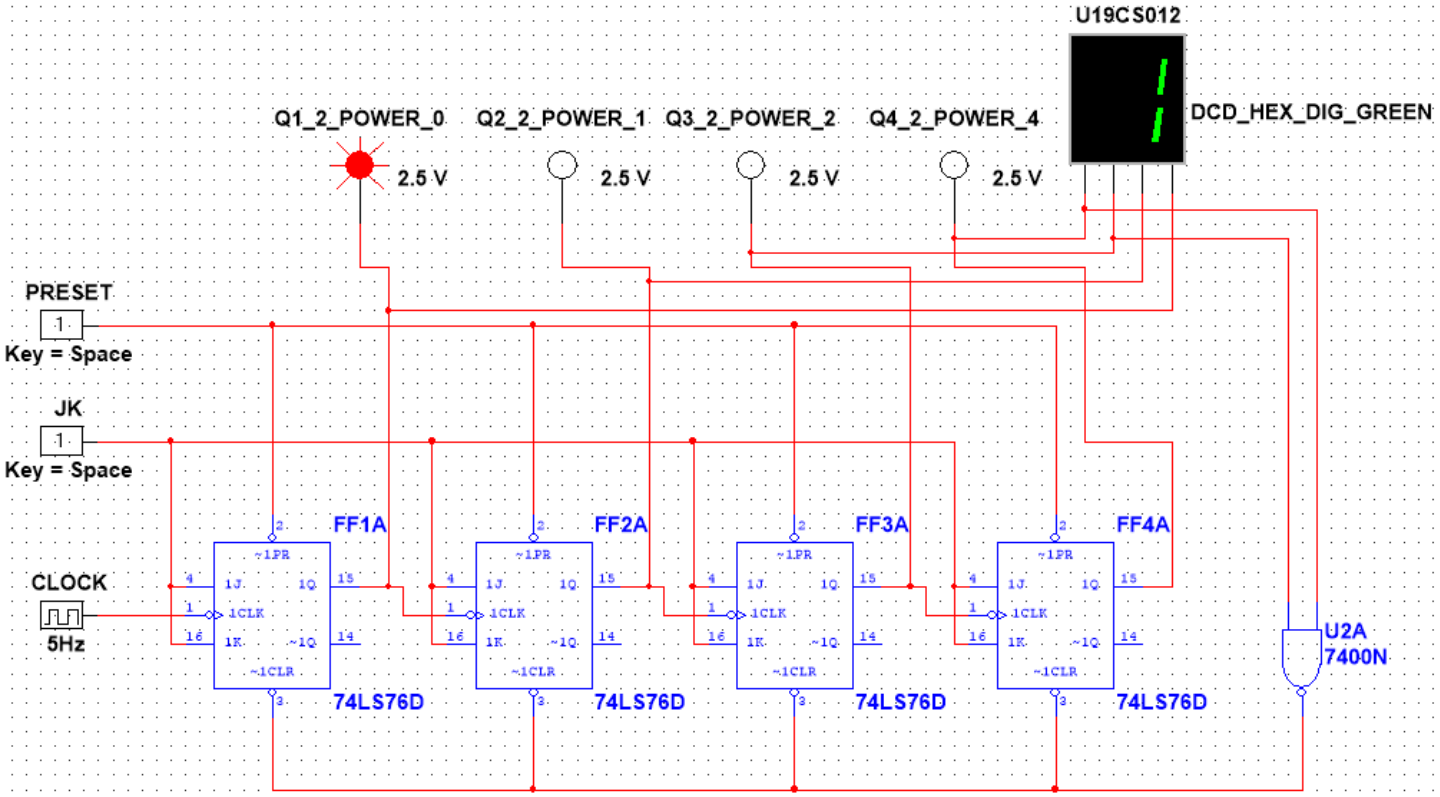


Cycle Starts Repeating Again

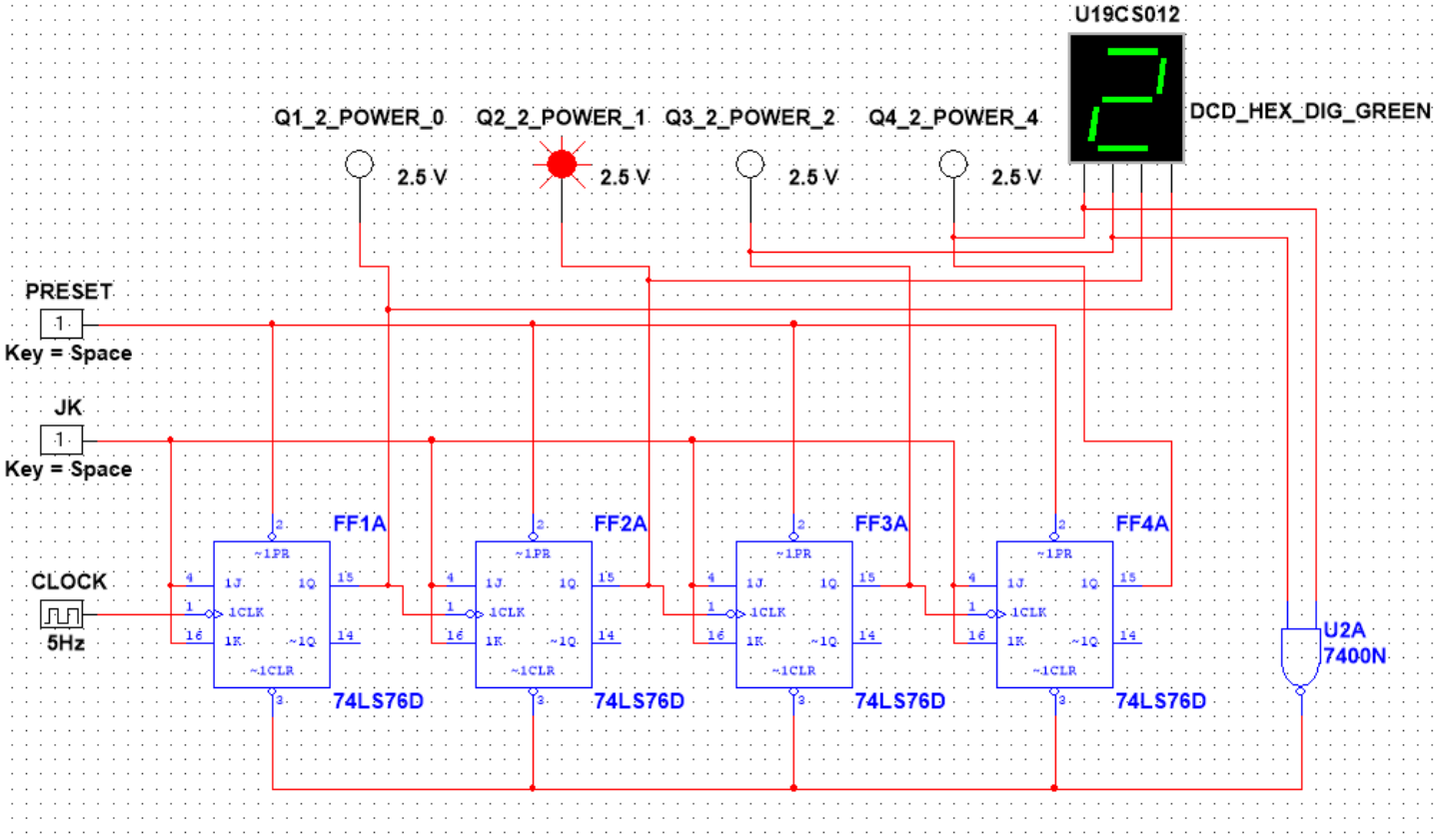
13.) Valid State: 12 [1100] -> 0 [0000]



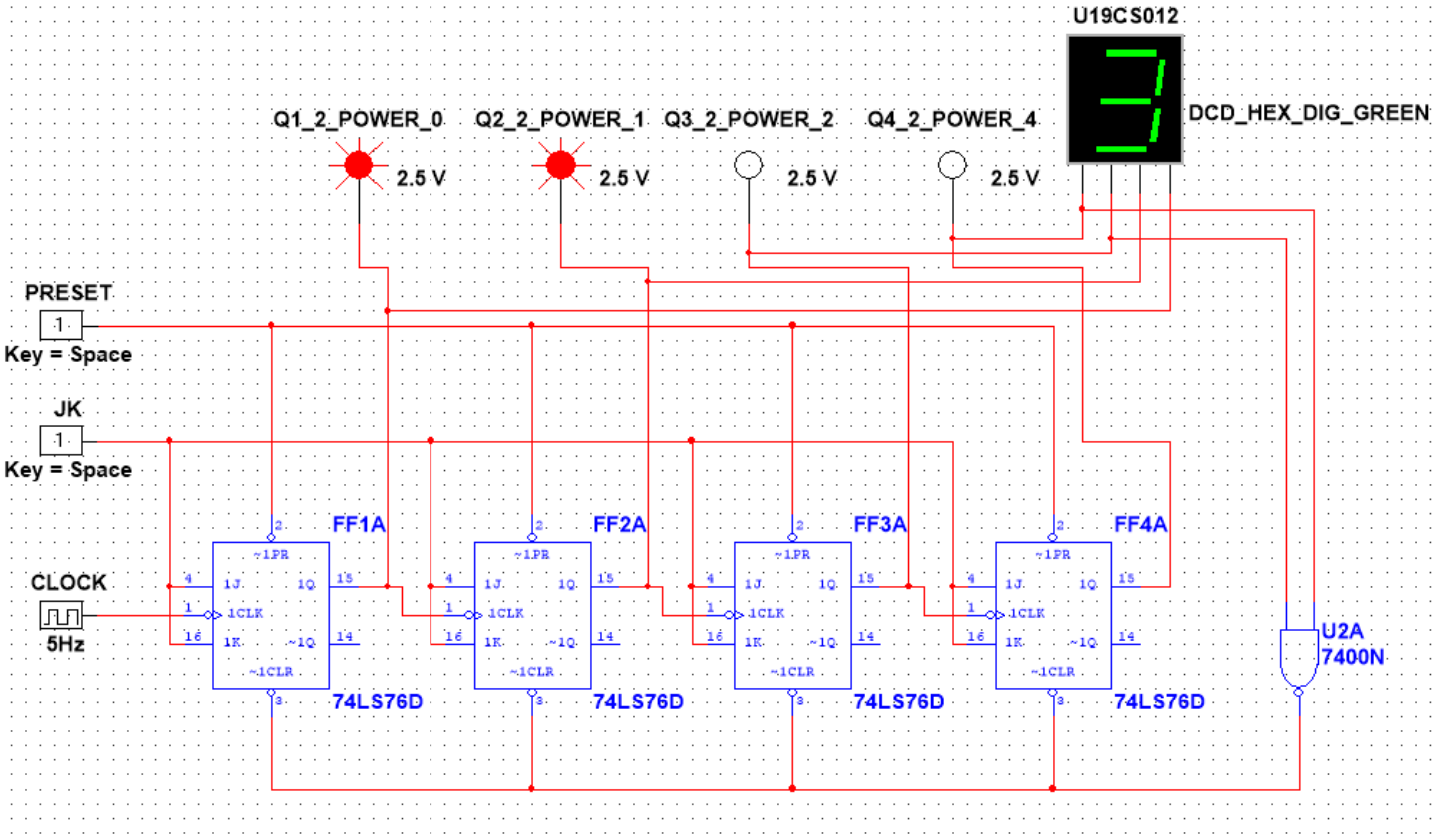
14.) Valid State: 13 [1101] -> 1 [0001]



15.) Valid State: 14 [1110] -> 2 [0010]

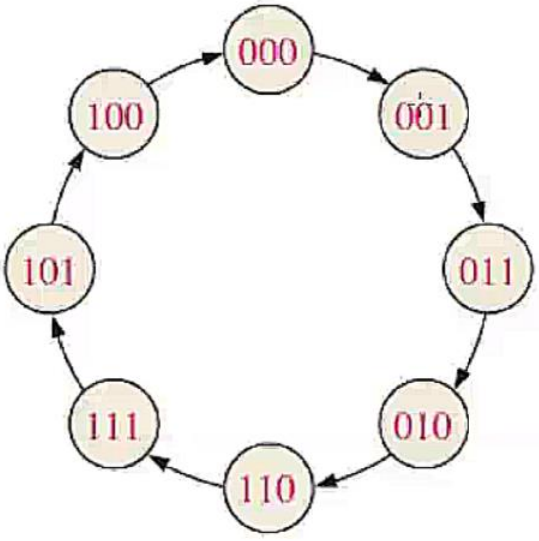


16.) Valid State: 15 [1111] -> 3 [0011]

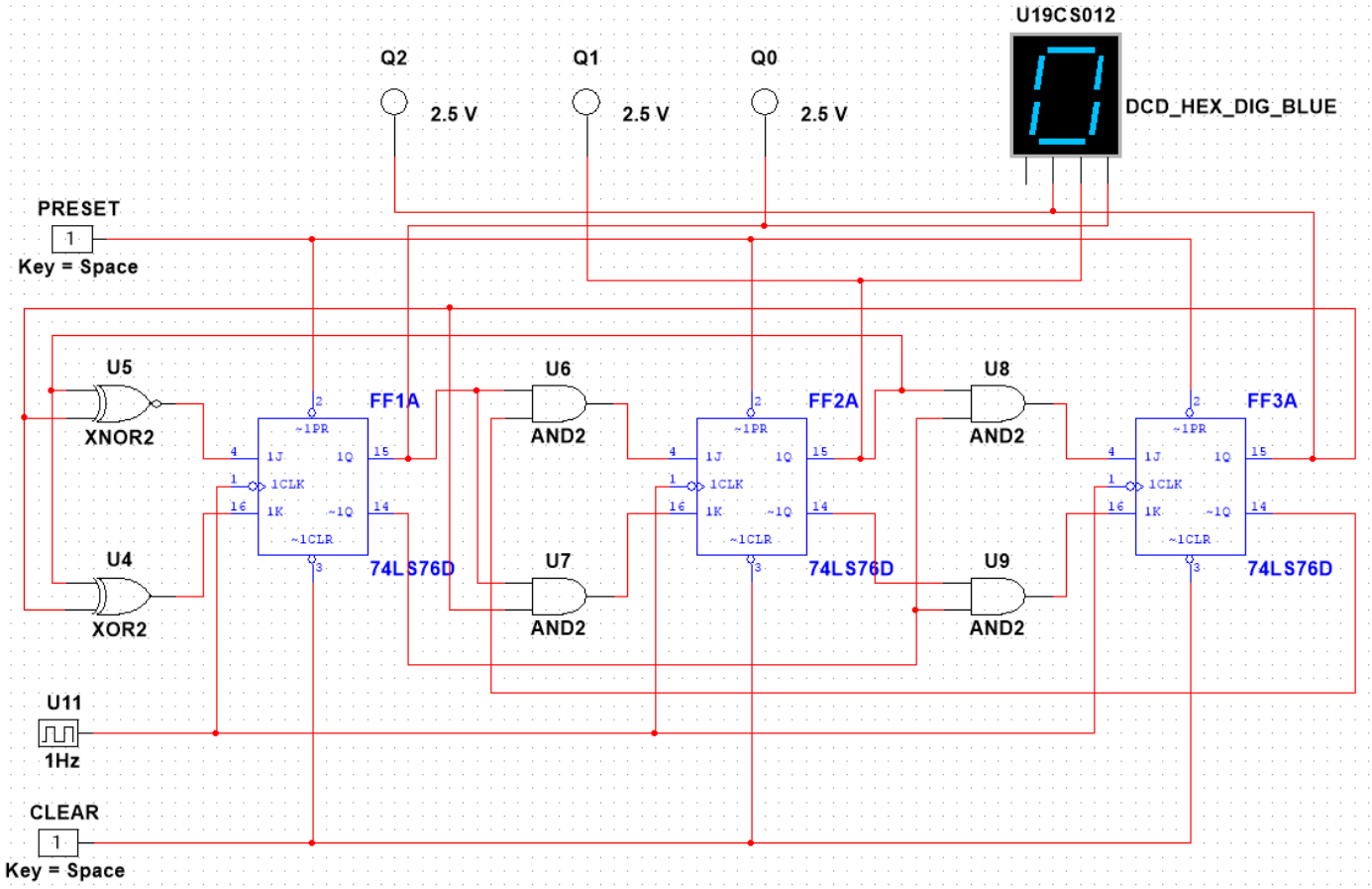


And the Loop Continues…

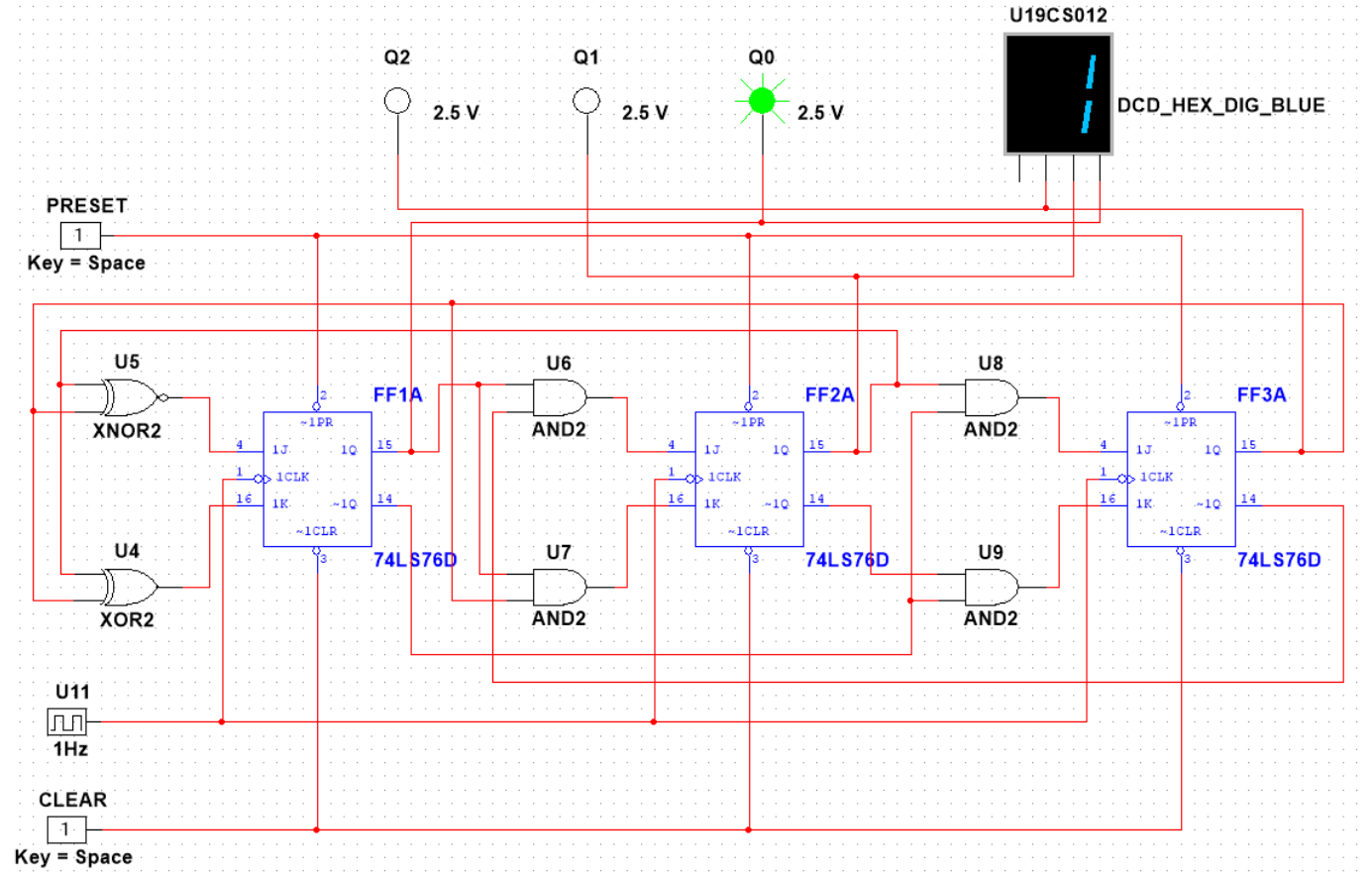
2.) Design and Implement *3-BIT Gray Code UP Synchronous Counter* in Multisim using JK Flip-Flops and Logic Gates.



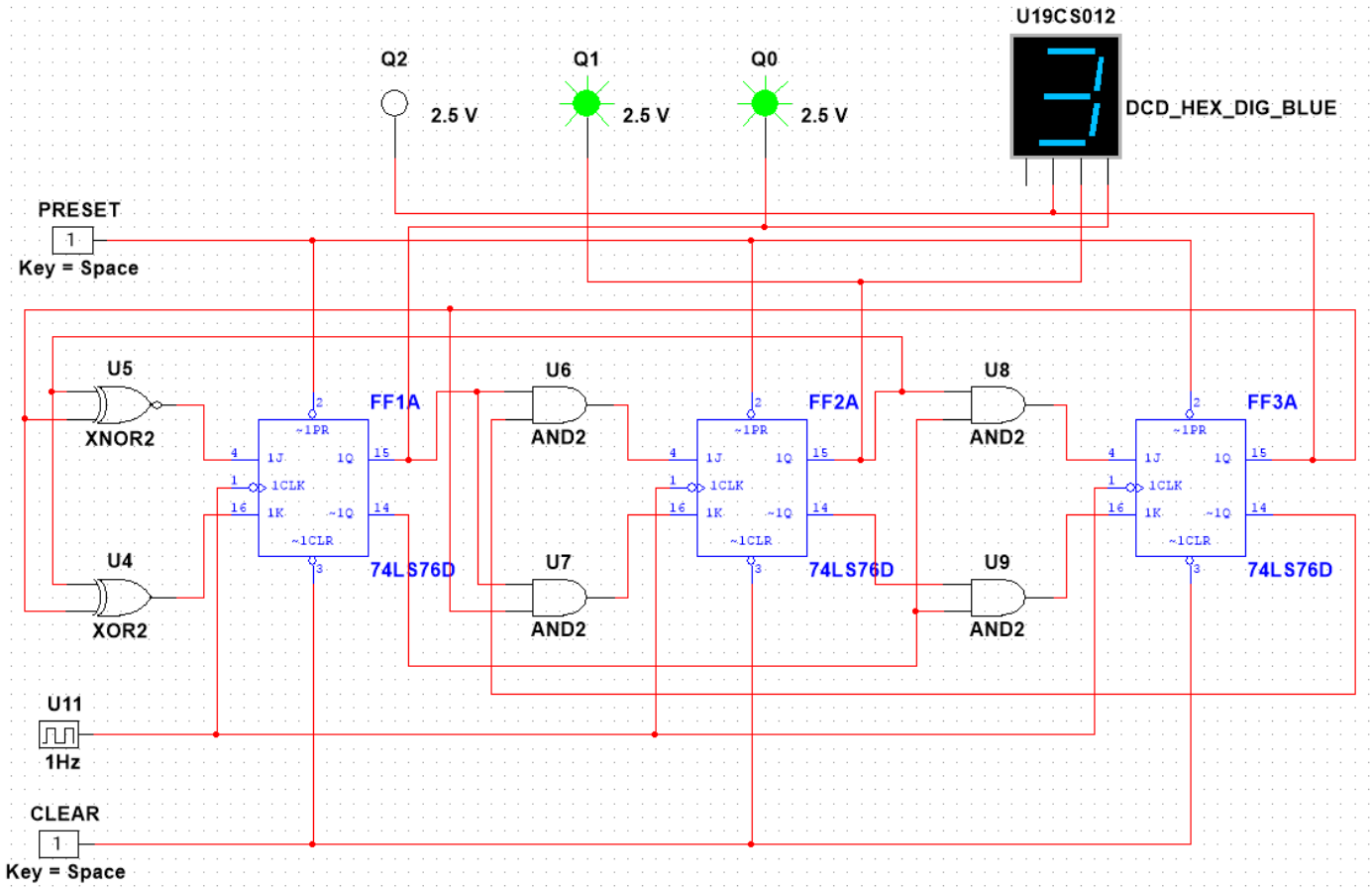
1.) Gray Code State: 000 [0]



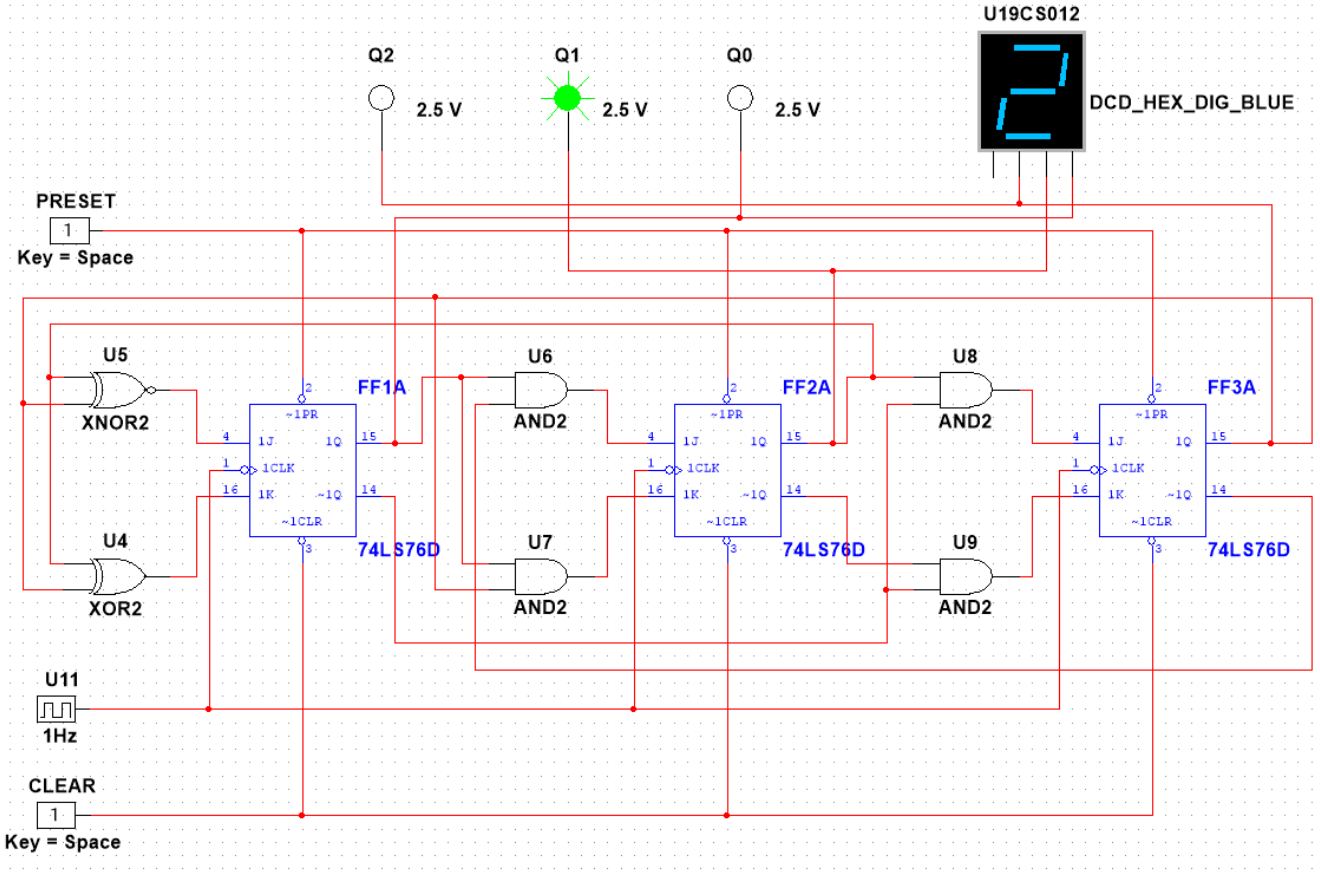
2.) Gray Code State: 001 [1]



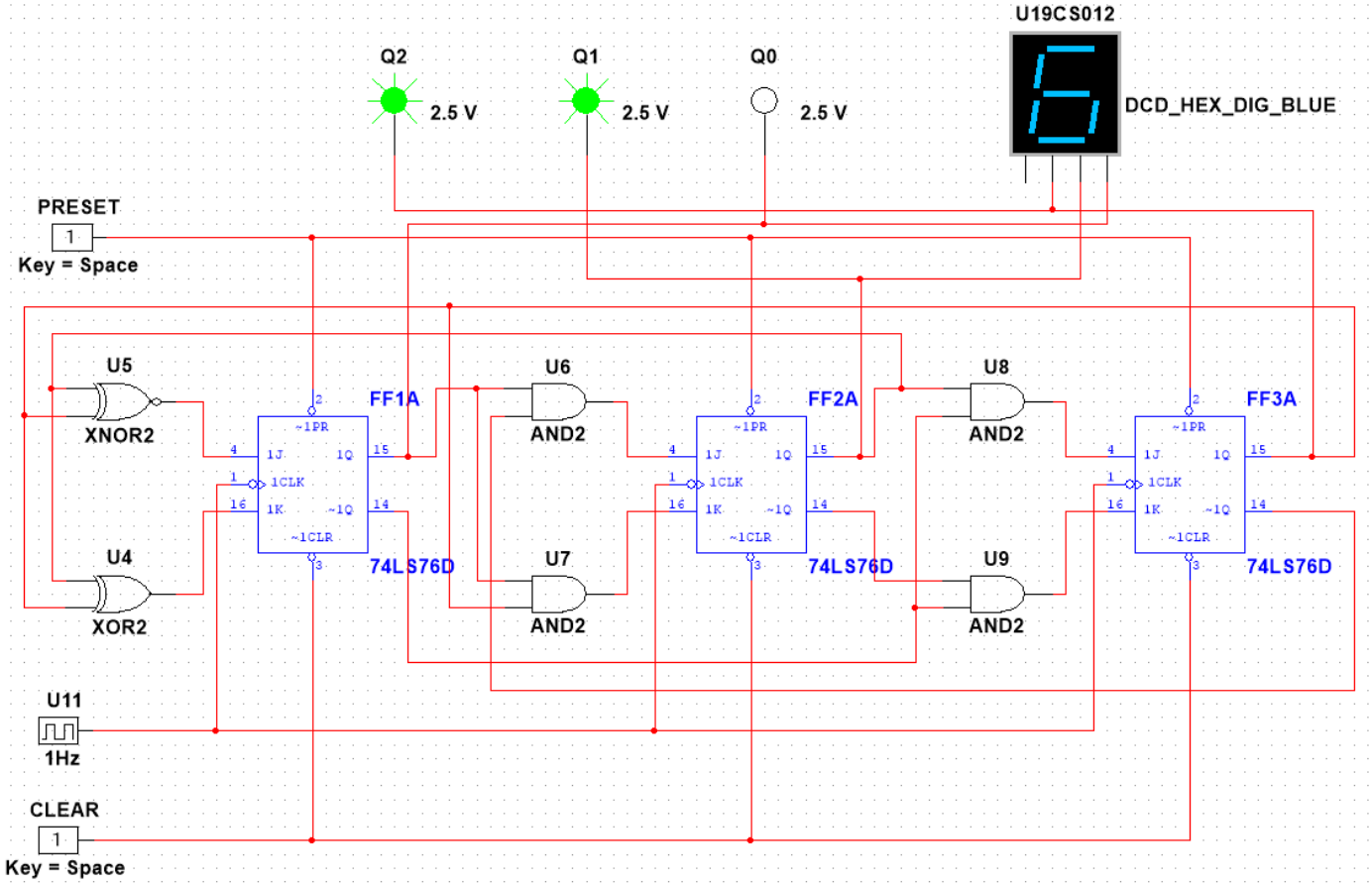
3.) Gray Code State: 011 [3]



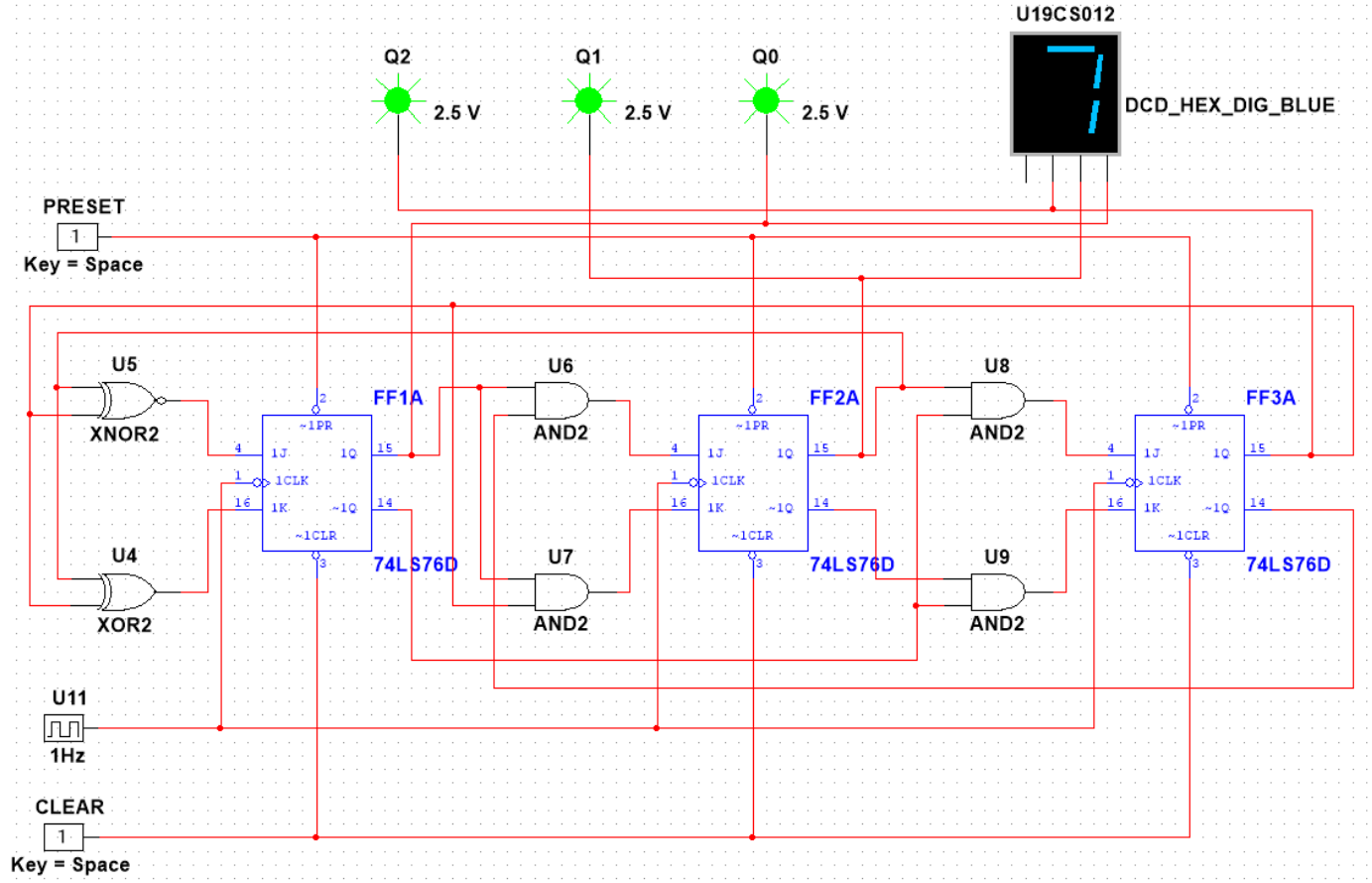
4.) Gray Code State: 010 [2]



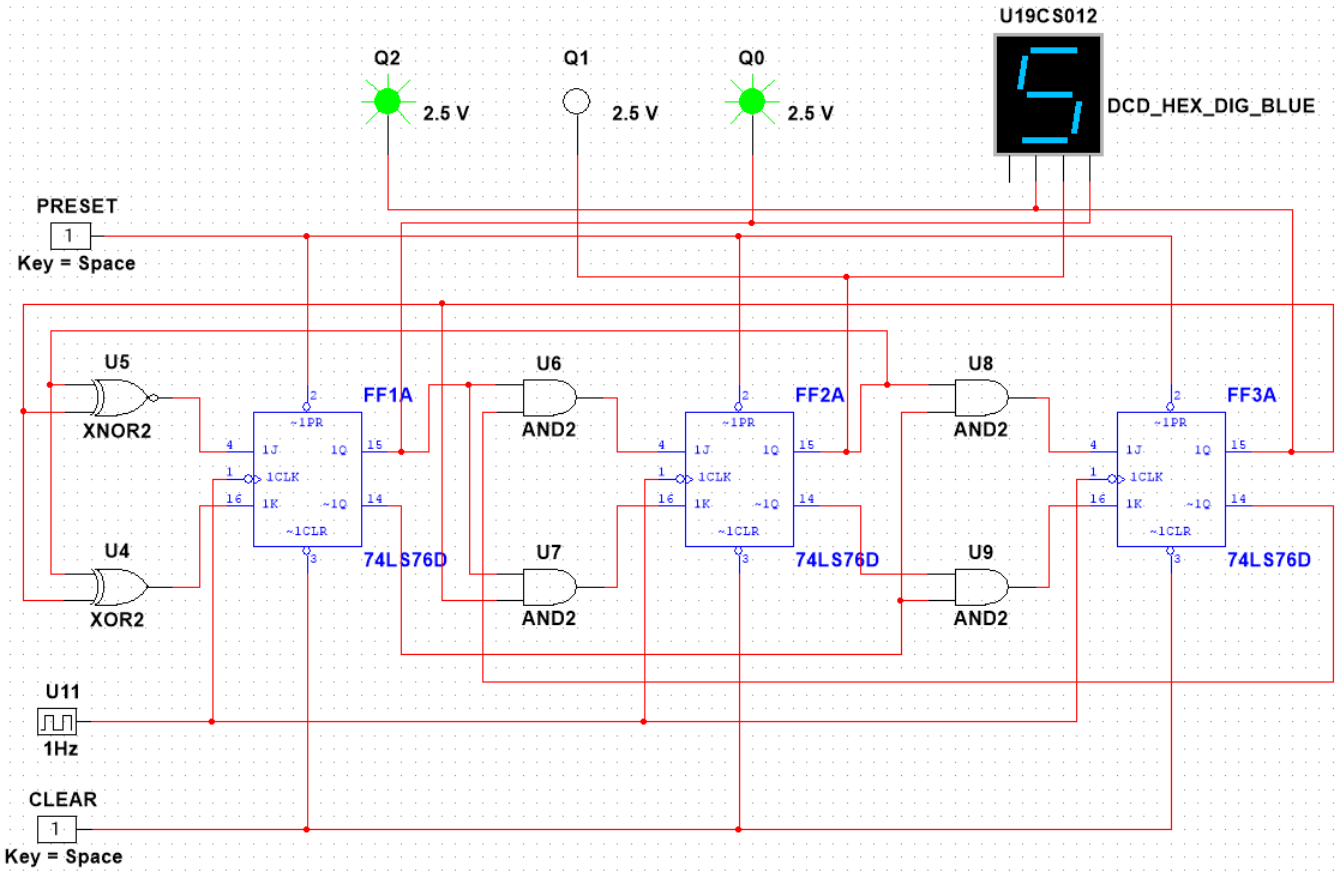
5.) Gray Code State: 110 [6]



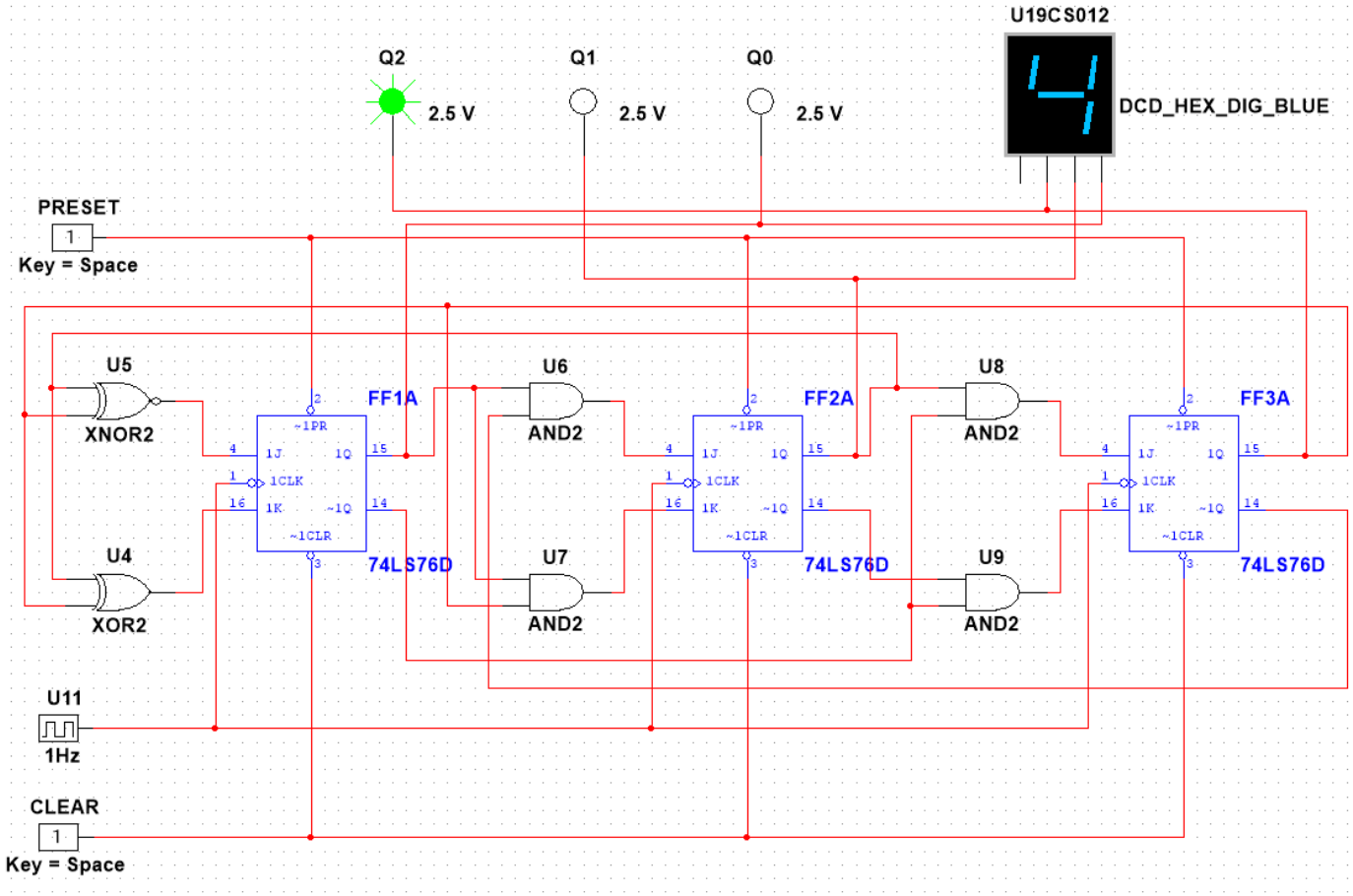
6.) Gray Code State: 111 [7]



7.) Gray Code State: 101 [5]



8.) Gray Code State: 100 [4]

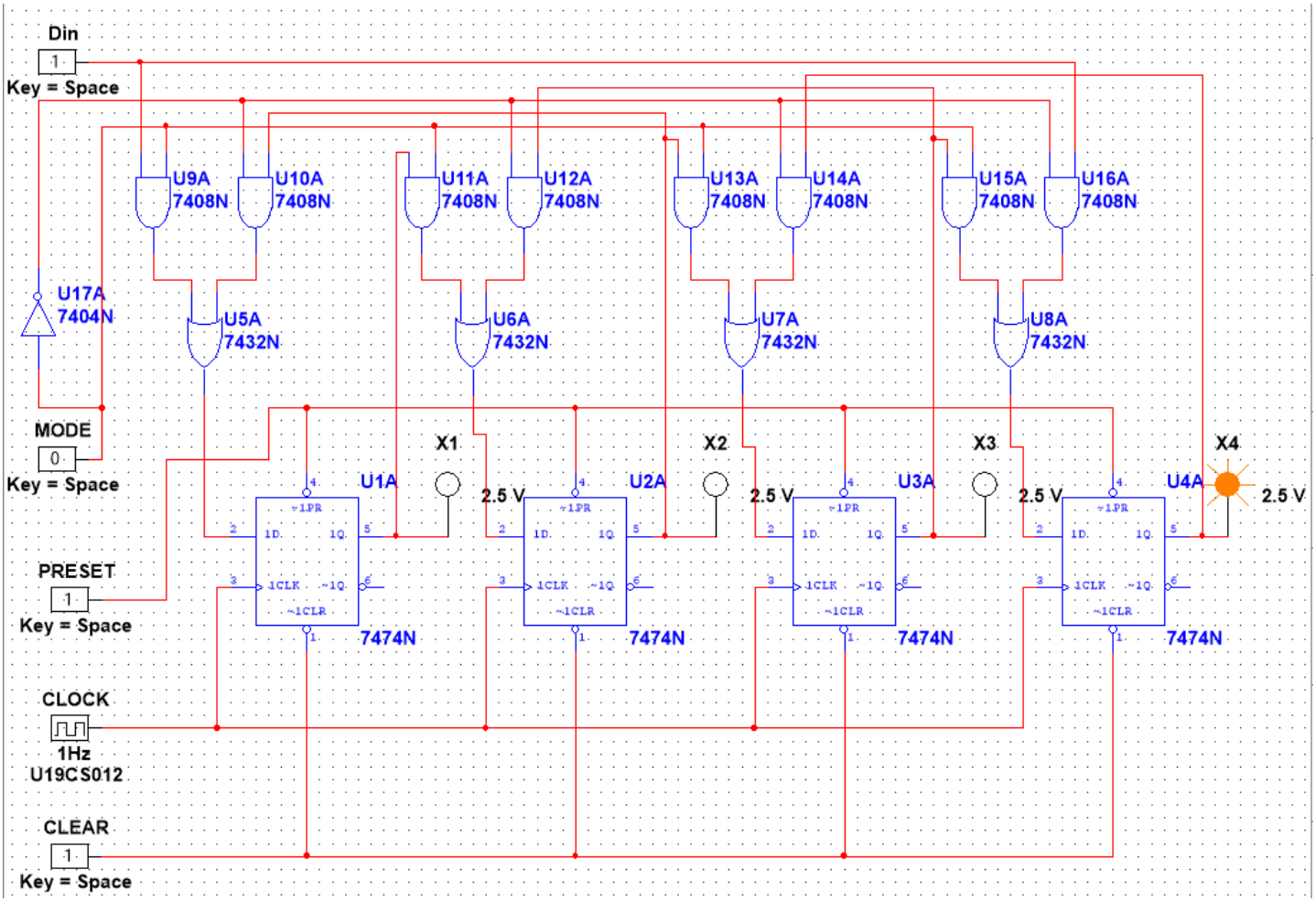


3.) Design and Implement *Bidirectional Shift Registers Using Mode Control* in Multisim using JK Flip-Flops and Logic Gates.

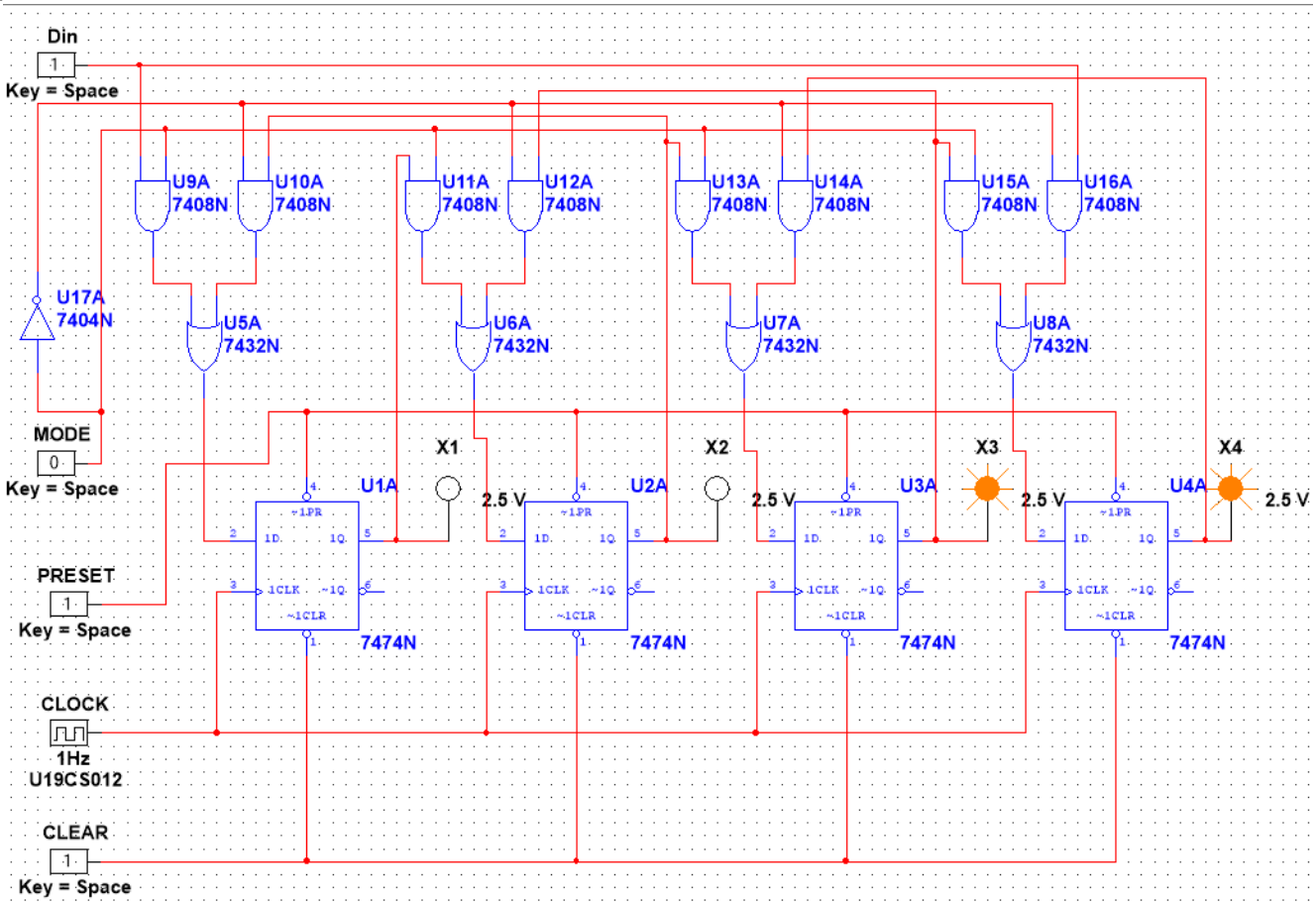
A1.) Implementation:

Mode Control: ***0 -> Shift LEFT Register***

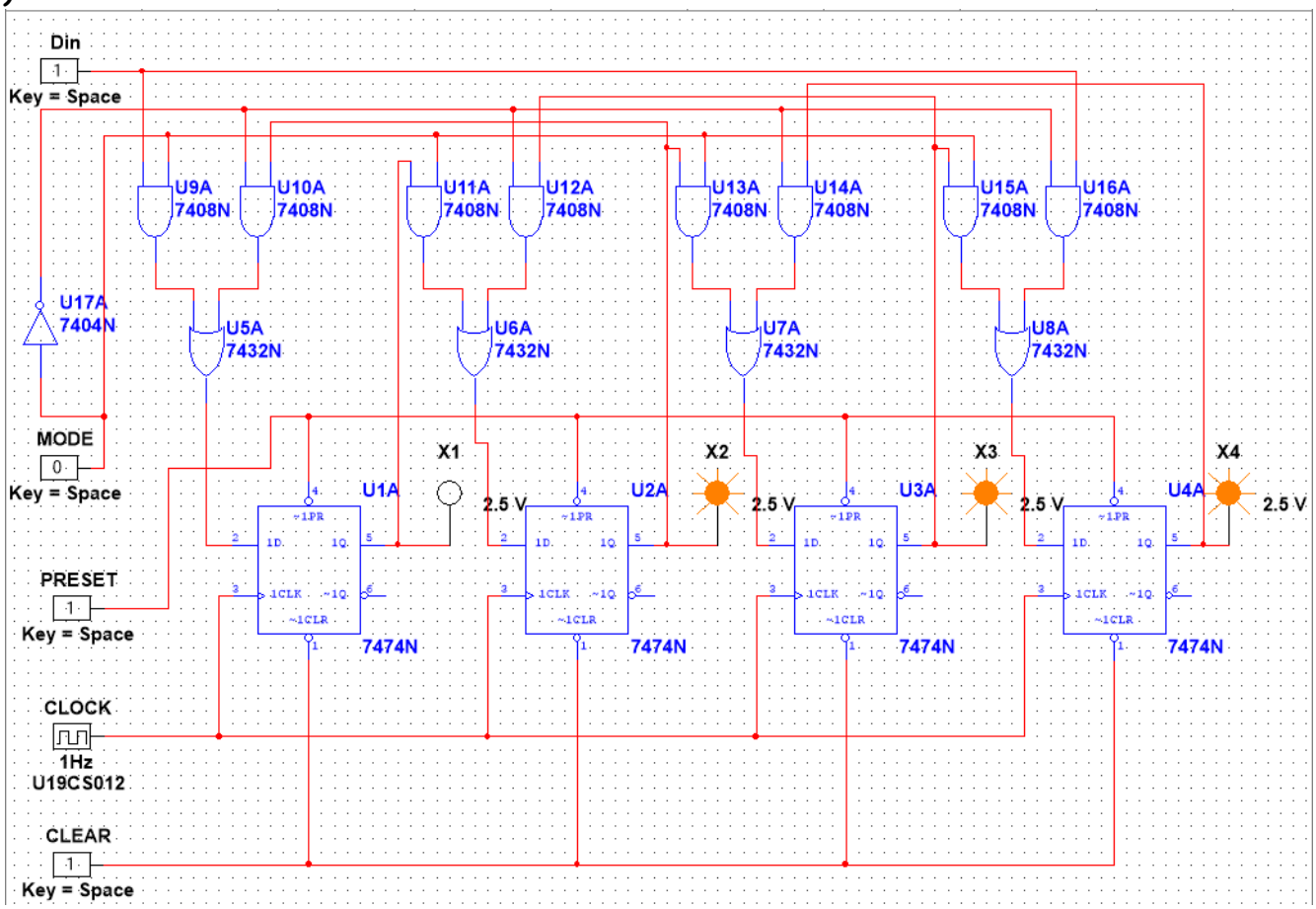
1.) State 1: 0001



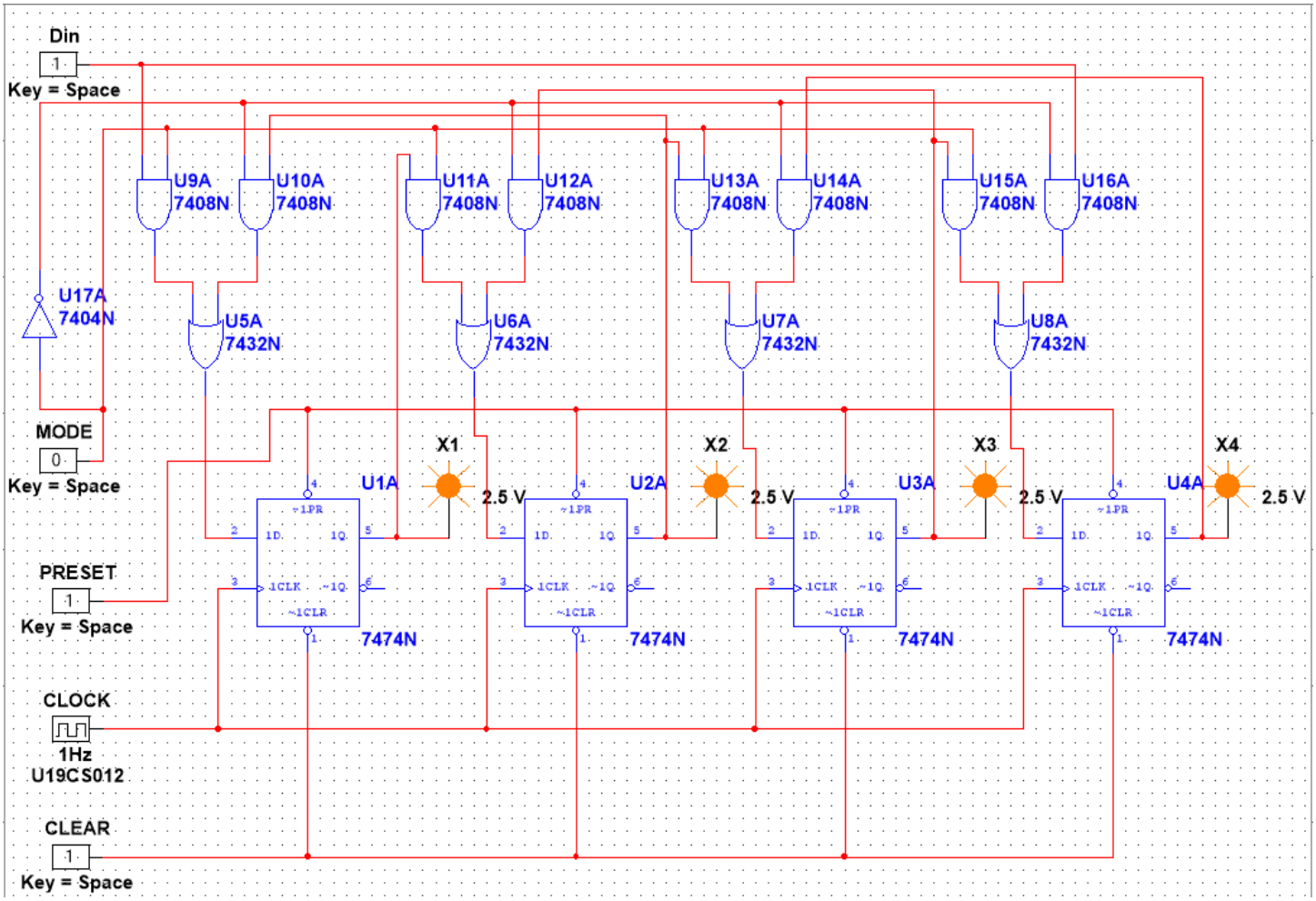
2.) State 2: 0011



3.) State 3: 0111



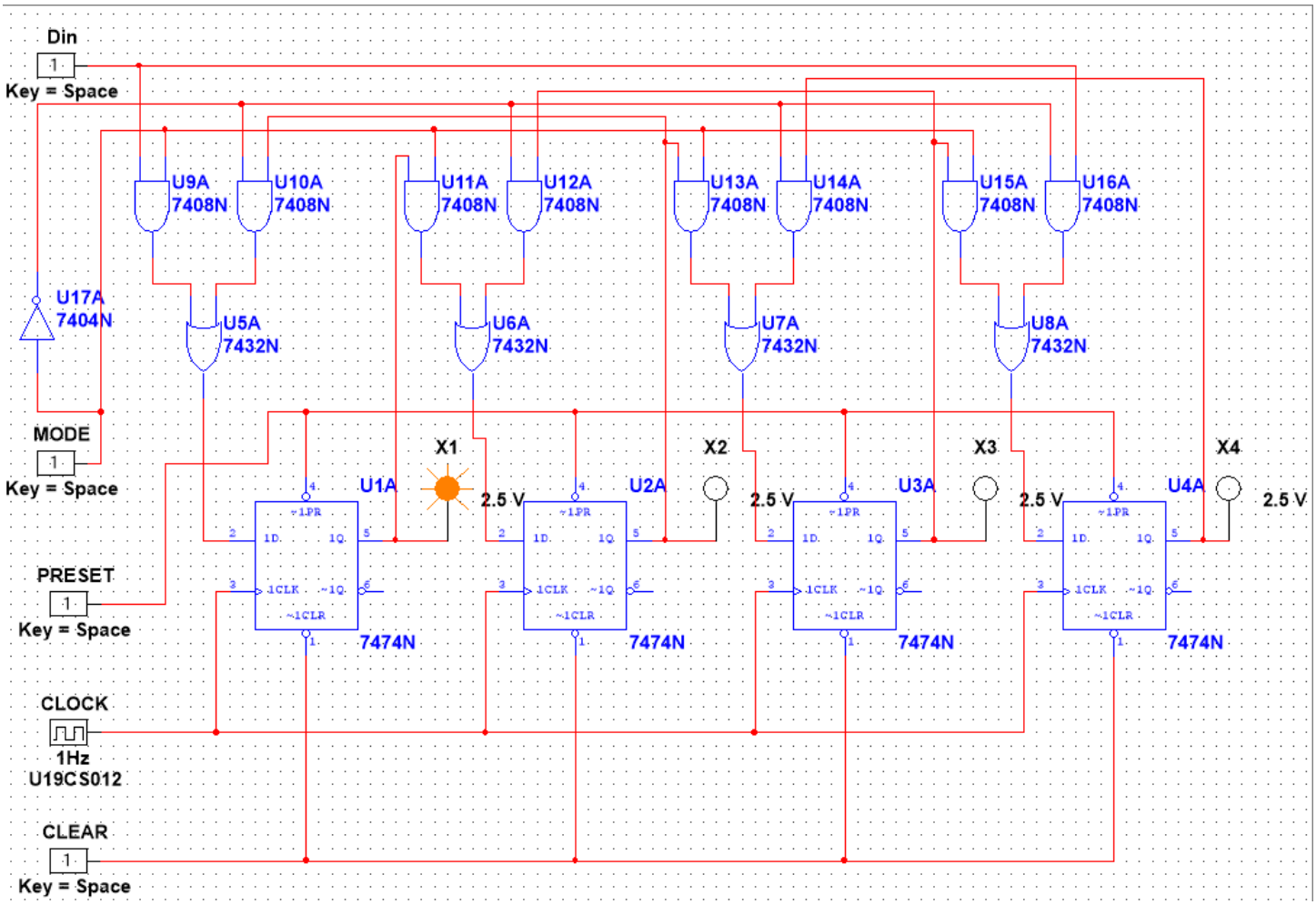
4.) State 4: 1111



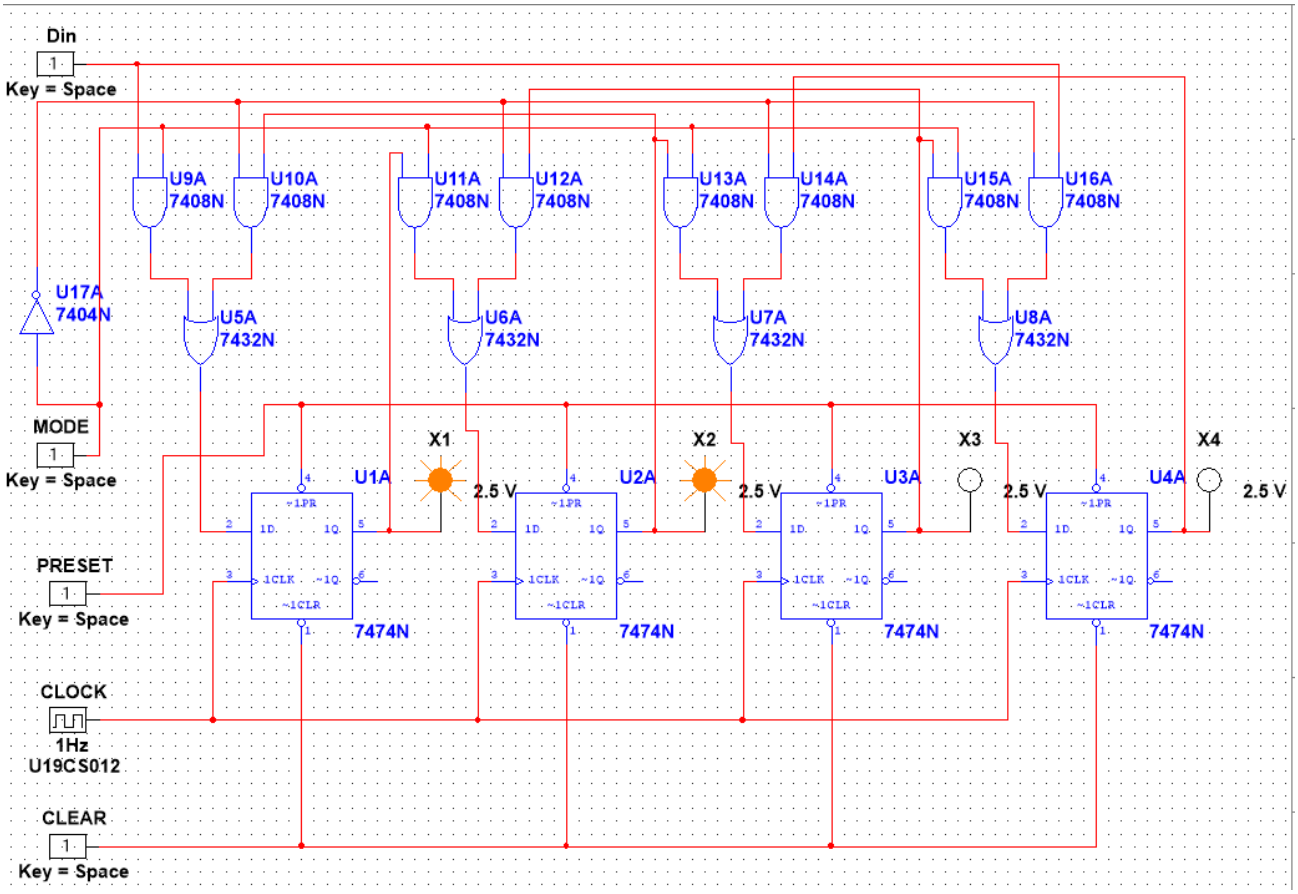
A2.) Implementation:

Mode Control: 1 -> Shift Right Register

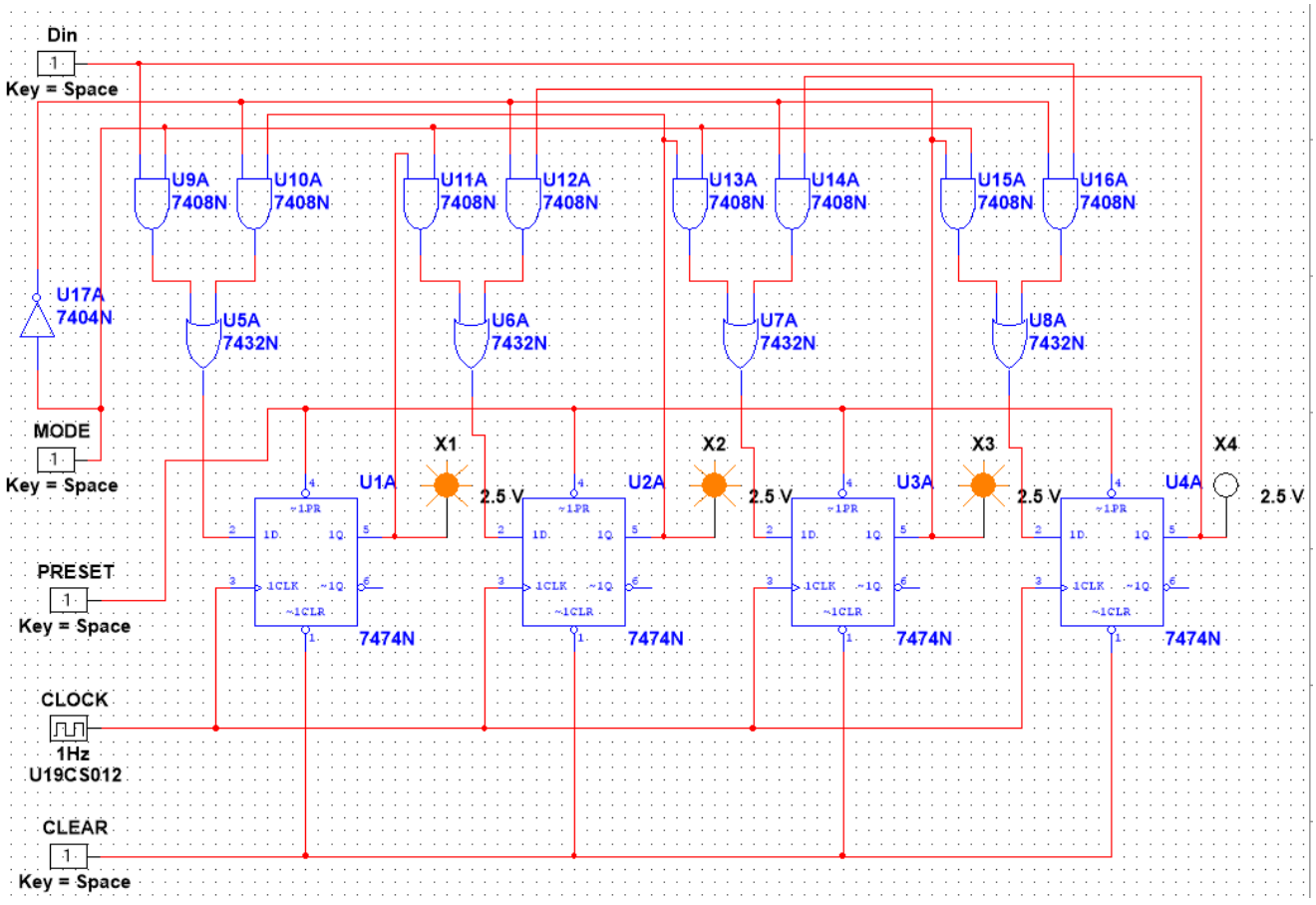
1.) State 1: 1000



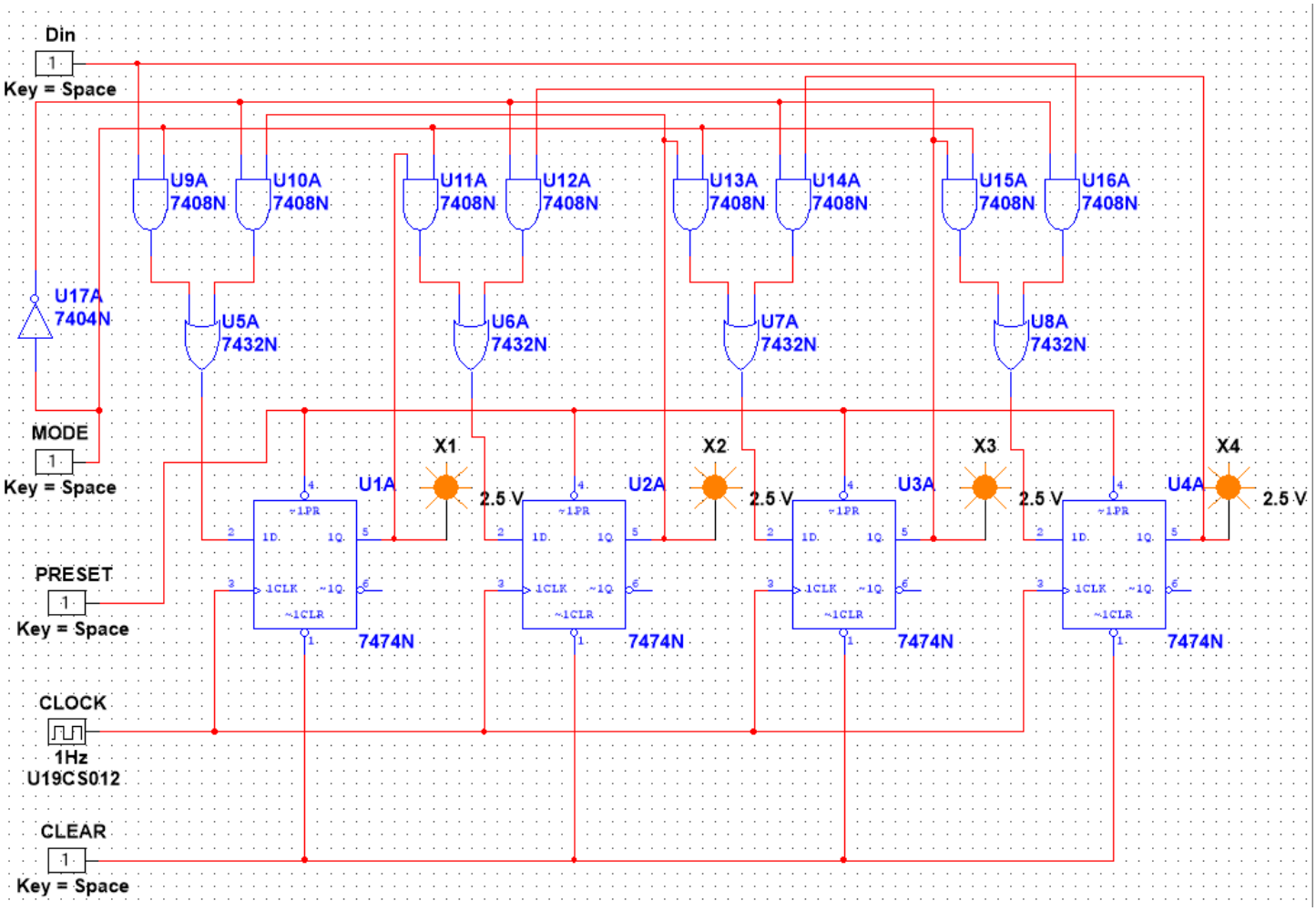
2.) State 2: 1100



3.) State 3: 1110



4.) State 4: 1111



D.) *CONCLUSION*:

We have Successfully Implemented *MOD-12 Counter, Synchronous Gray Counter and Bi-directional Shift Registers [Using Mode Control]* with the Help of JK Flip-flop and Logic Gates and **verified** our **MULTISIM Outputs** and **Results** from *Theoretical Knowledge of these Circuits taught in DELD Classes.*