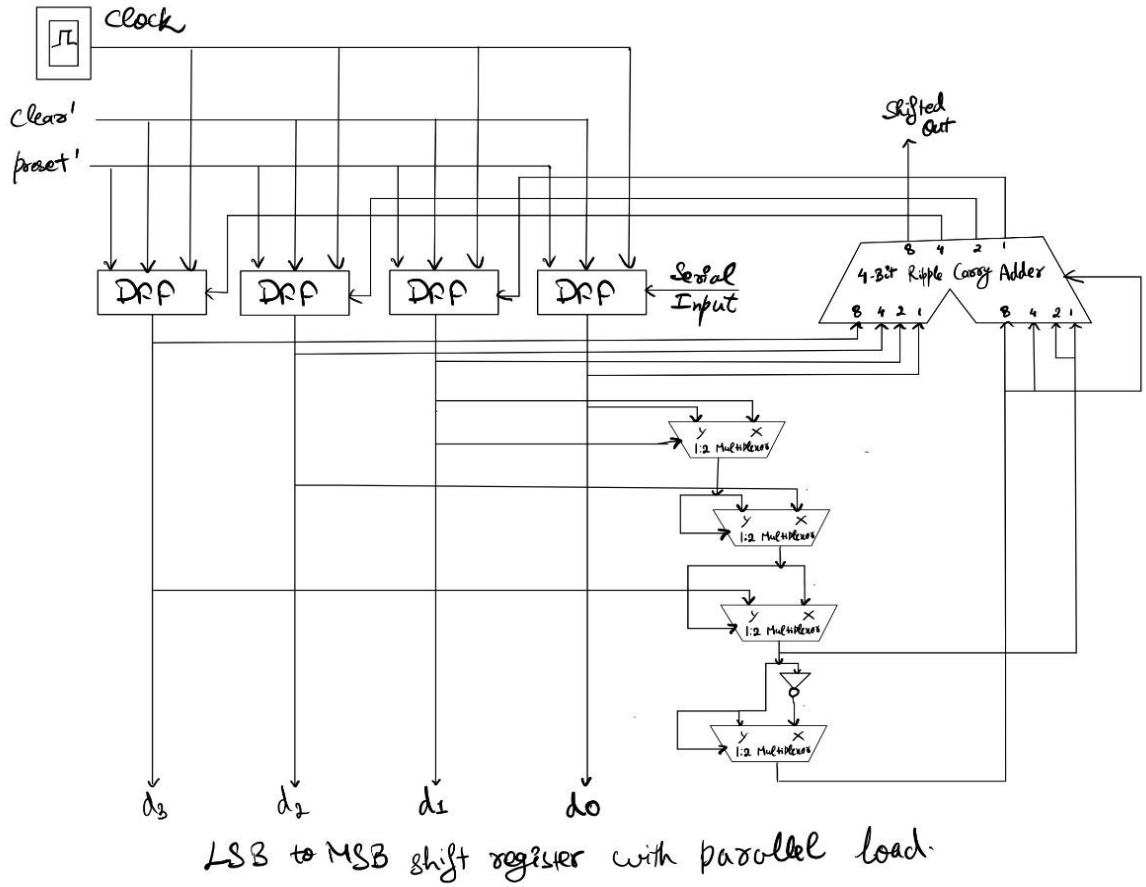
EXPERIMENT O6
SEQUENTIAL DOUBE-DABBLE

SHIFT REGISTER WITH PARALLEL LOAD



LSB to MSB shift register with parallel load.

This is a shift register that shifts from *LSB* to *MSB*. This shift register is responsible in performing the most crucial step of *double-dabble algorithm* -- *ADD 3 and SHIFT LEFT*. The component does both the steps simultaneously.

This shift register handles a single nibble (*4 bits*). To be able to construct the logic for a complete *binary to BCD converter*, several *shift registers* might need to be connected in cascade. This realization is discussed in the next section.

Observe the four bits *d3, d2, d1* and *d0* shown in the output. The values of these bits change at every *negative/falling edge* of the clock pulse. Two operations happen when the falling edge of the clock is witnessed.

- *ADD 3* which is the characteristic induced by the parallel circuitry that basically acts as a *>4 conditional 3 adder*.
- *SHIFT LEFT* which is the intrinsic characteristic of an *LSB to MSB shift register. SHIFT LEFT* operation not only shifts the present nibble $d_3 d_2 d_1 d_0$ in the output towards the left in order to push the *leading bit d3* out, but also simultaneously receives a new bit i from the *serial stream of input bits* to display the nibble $d_2 d_1 d_0 i$ in the output.

Observe that these two operations are the core of the *double-dabble algorithm*. The algorithm is made up of alternately applying two operations *ADD 3* and *LEFT SHIFT*. While the former operation is conditional, the latter must be applied after every one step.
One first checks the decimal value of each of the *nibbles* in the *current intermediate BCD output* starting from the *least significant bit.* If any such nibble has a decimal value of *greater than 4*, *binary 3 (0011)* is added to it. After this, the very next step is the *LEFT SHIFT* in which the *next bit* from the *binary input string* is taken in.

Since the *ADD 3* operation involves checking groups of 4 bits and *conditionally* adding 3 to them, the *shift register* we have designed in this section is the one that takes *4 inputs*.

Note that the *parallel load* that acts as a *>4 conditional 3 adder* is designed by *multiplexors* using *Shannon's decomposition* on the boolean function that was used in implementing the *first part* of *experiment 2*. Addition of two 4-bit binary strings is done using a *ripple carry adder*.

The simulation of the above circuit diagram can be illustrated in the following way.

Since in this experiment we are dealing only with *7 bit binary strings*, let the *input stream* consist of the 7 bits $d_7 d_6 d_5 d_4 d_3 d_2 d_1$.

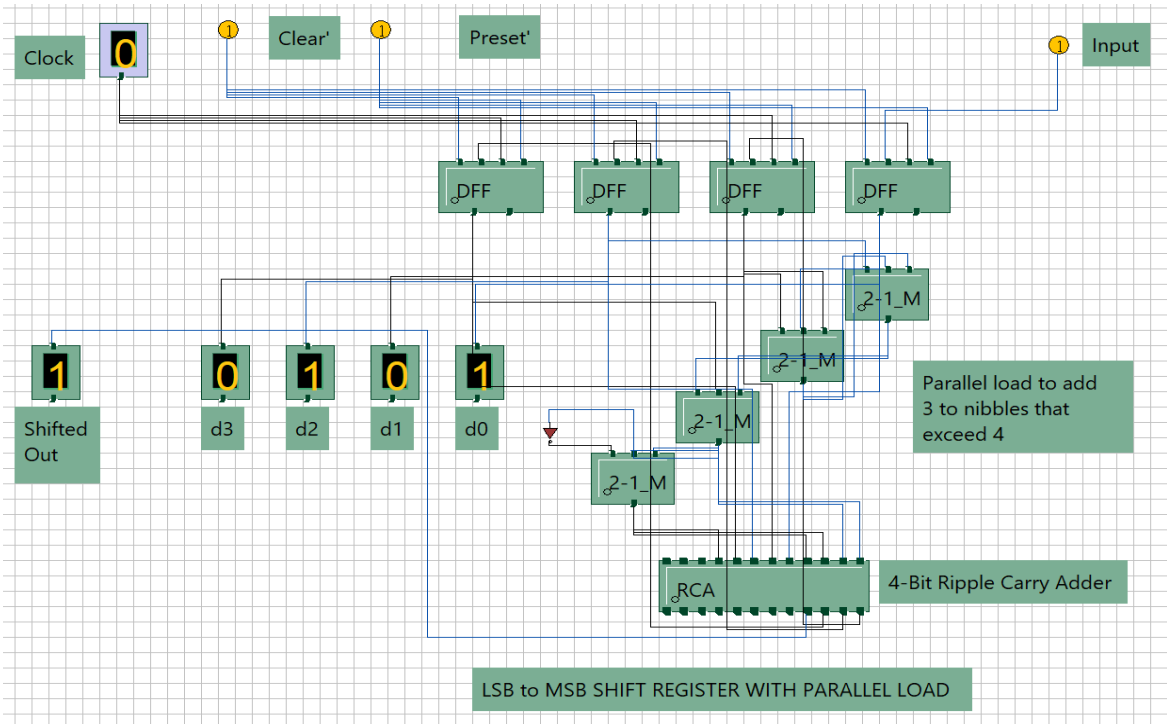
Set the *clear'* input to 0 (*preset'* will remain one) to *clear* the bits being initially displayed in the output. Start the clock, wait until one falling edge is completely witnessed. The output bits are now cleared (similar to *initialization step* in *double-dabble algorithm*). Set *clear'* input to 1 and restart the clock. The working of the above circuit for a general 7 bit input binary string can be understood as follows.

Falling Edge Index	*Operation the shift register is performing*	*Current Output Nibble*	*Next Bit in the Stream*
0	*INITIALIZATION*	0000	d_7
1	*ADD 3*	0000	d_7
1	*SHIFT LEFT*	$000d_7$	d_6
2	*ADD 3*	$000d_7$	d_6
2	*SHIFT LEFT*	$00d_7d_6$	d_5
3	*ADD 3*	$00d_7d_6$	d_5
3	*SHIFT LEFT*	$0d_7d_6d_5$	d_4
4	*ADD 3*	$e_4e_3e_2e_1$	d_4
4	*SHIFT LEFT*	$e_3e_2e_1d_4$	d_3
5	*ADD 3*	$f_4f_3f_2f_1$	d_3

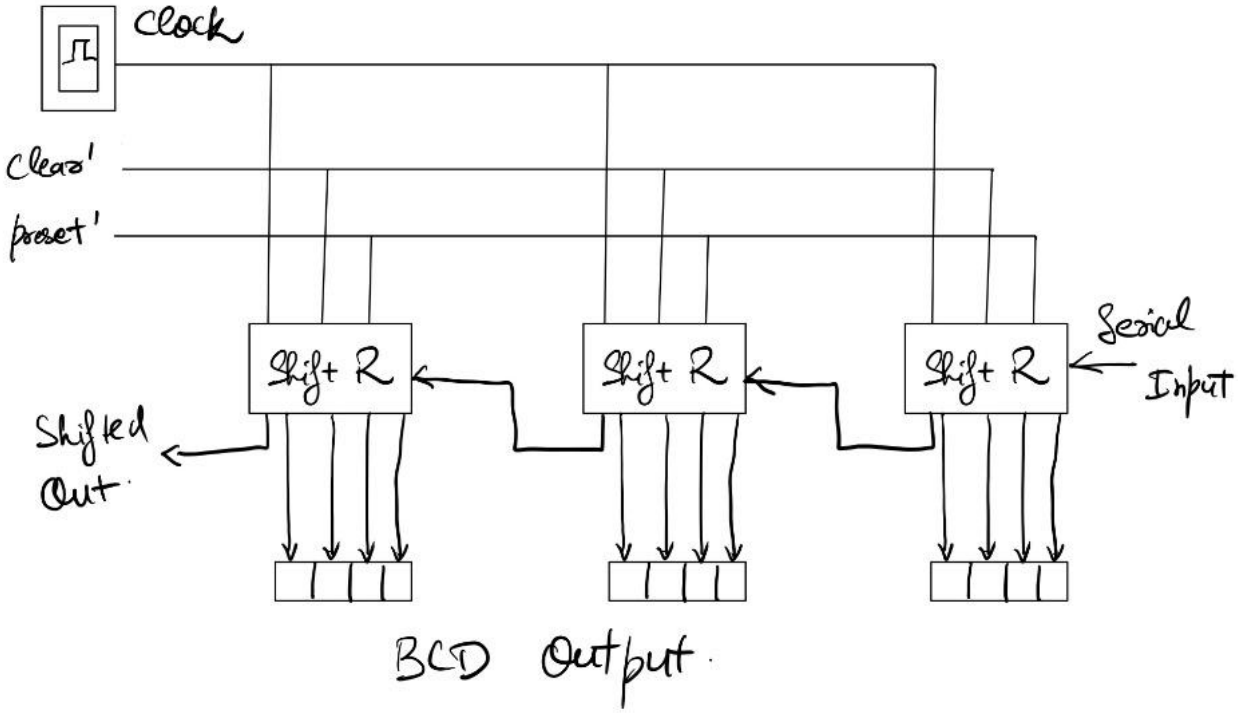
5	*SHIFT LEFT*	$f_3 f_2 f_1 d_3$	d_2
6	*ADD 3*	$g_4 g_3 g_2 g_1$	d_2
6	*SHIFT LEFT*	$g_3 g_2 g_1 d_2$	d_1
7	*ADD 3*	$h_4 h_3 h_2 h_1$	d_1
7	*SHIFT LEFT*	$h_3 h_2 h_1 d_1$	$--$

(*NOTE: In the first three ADD 3 operations, the output nibble will definitely not change because a 4 bit binary string with the two most significant bits as 0 can never be greater than 4 in decimal*.
This is a simulation for a general input. There might be other ADD 3 operations too where the output nibble stays intact.)
The simulation signifies that the clock must completely witness 7 falling edges in order to reach the end of the simulated algorithm. So *stop the clock after 7 falling edges are seen*.

The same circuit diagram was implemented as follows.



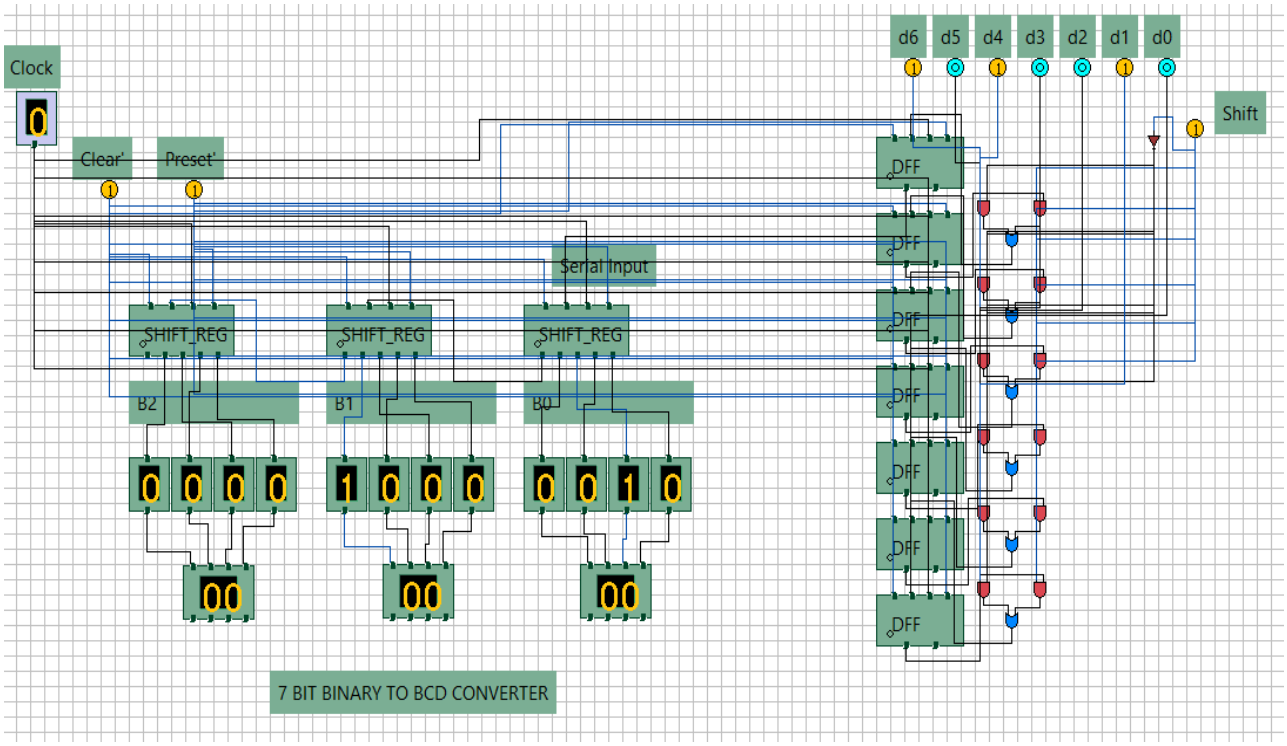
7 BIT BINARY TO BCD CONVERTER USING SHIFT REGISTERS



As was subtly mentioned in the last section, the *shift registers* can be cascaded in order to realize a *7 bit binary to BCD converter*. In this cascade design, the *shifted out* bit of one register is the *serial input* of the other register. The *operations* (*ADD 3 and LEFT SHIFT*) of all the three registers are *synchronized* because they are connected to the same clock.

First of all, in the design exactly three registers were taken because the *BCD representation* of a 7 bit binary string can have at most 12 bits (3 nibbles); therefore one register is needed to deal with each of the nibbles. The working of this circuit diagram is *exactly the same* as the shift register itself (as was discussed in the last section), excluding the fact that though the rightmost shift register receives its input bit from the *serial stream of input bits*, for the other two registers, the input bit is the output bit of the register on the right.
So each of the three registers have there own *input streams* and a *similar simulation chart* can be drawn for each one of them.
As in the last section the clock must completely witness 7 falling edges in order to reach the end of the simulated algorithm. So *stop the clock after 7 falling edges are seen*. *It is at this point that the three output nibbles represent the BCD representation of the 7 bit binary string.*
(The *initialization step* of initially setting *clear'* as *0* is the same as before)

The above circuit diagram was implemented as follows.



How is the 7 bit binary string given altogether converted into a serial stream of bits?

For this we need a similar shift register. This register should also shift from *LSB* to *MSB*. We cannot use the shift register we designed in the first section because *it has a parallel circuitry to handle conditional addition of binary 3.* We do not want this while converting a *binary string* into a *bit stream* because it can corrupt/modify the original bits. Therefore, a simple *parallel-in serial-out (PISO)* register is designed (*see the right extreme*) that is also synchronized with the other registers by being connected to the same clock. It *shifts out* each bit in the input binary string (from the *MSB*) one by one, whenever a falling edge is encountered in the clock. To enable shifting, the *shift* input parameter must be set to 1. In the initialization step, the *shift* parameter must be first set to 0 (with *clear'* obviously equal to 1) to enable *loading* of the 7 bit binary string and *latching* of data in the *D flip flops*.

Since here we have used the same *clear'* parameter for both the sub-units, you should first set *clear'* to 0 and clear all the output bits. The you should set *clear'* to 1 and *shift* to 0 to *load* data. Then *shift* must also be set to 1 to start the simulation.