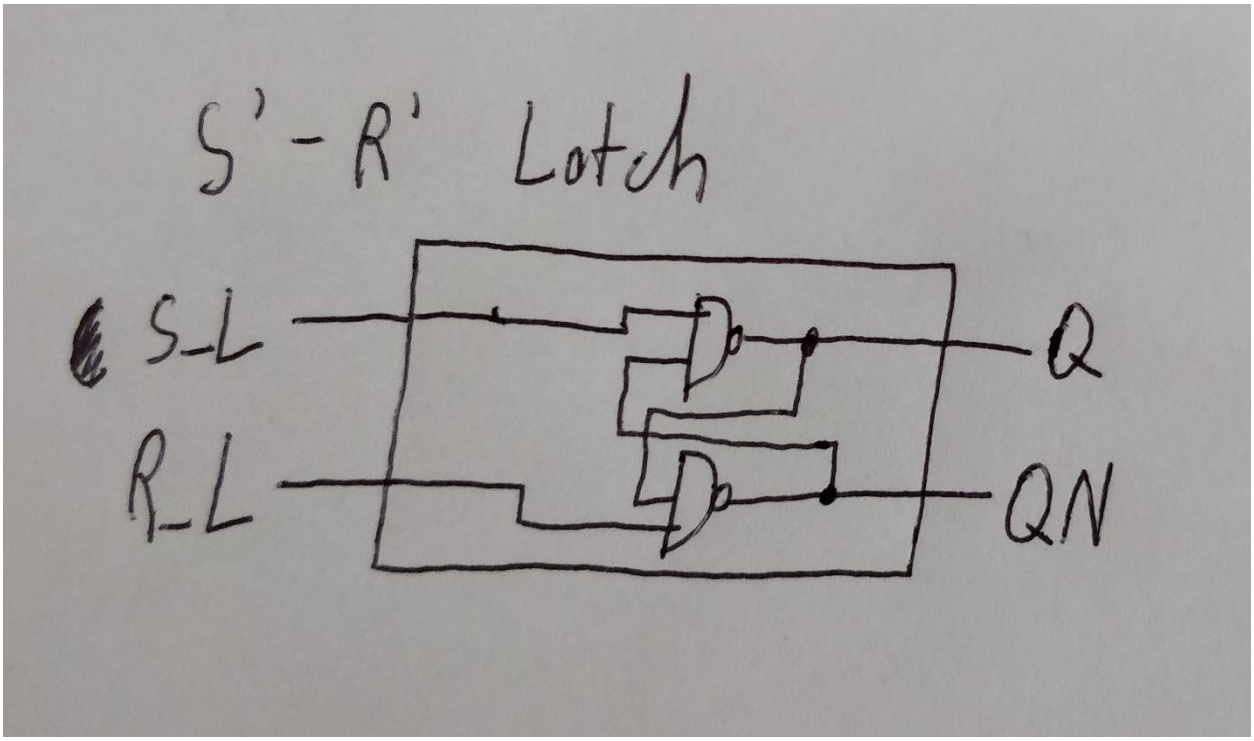
## Task 1: S'-R' Latch

A S'-R' latch is a type of sequential circuit that uses two NAND gates in a feedback loop to create a circuit that can hold on to a signal it has previously gotten. It takes two inputs S_L and R_L and outputs Q and QN based on the following truth table:

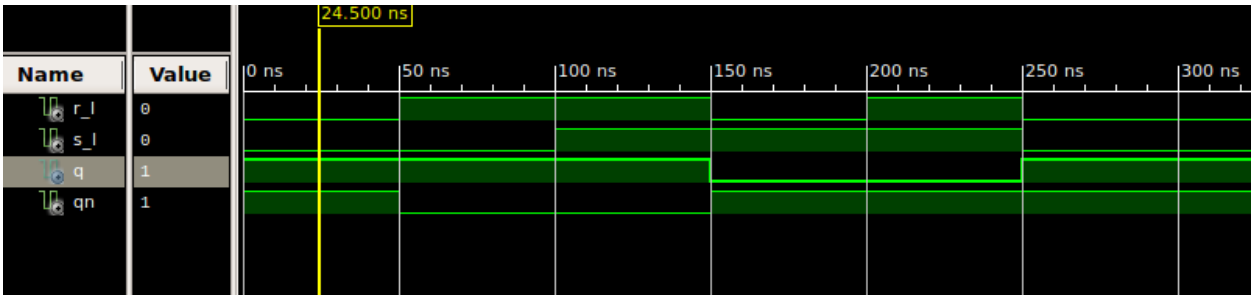
| S_L | R_L | Q | QN |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | Last Q | Last QN |



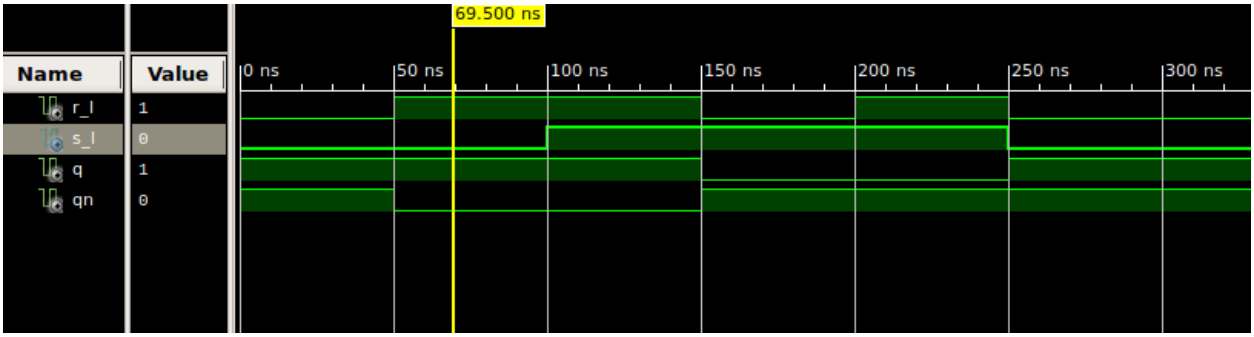
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity srlatch is
    Port (R_L: in std_logic;
          S_L: in std_logic;
          Q: out std_logic;
          QN: out std_logic);
end srlatch;

architecture Behavioral of srlatch is
   signal tmp1,tmp2: std_logic;
   component NAND2 port(I1,I0: in std_logic;
                        O: out std_logic);
   end component;
begin
   U0: NAND2 port map(S_L,tmp2,tmp1);
   U1: NAND2 port map(R_L,tmp1,tmp2);
   Q <= tmp1;
   QN <= tmp2;
end Behavioral;
```

```
1  library ieee;
2  use ieee.std_logic_1164.ALL;
3
4  entity testbench is
5  end testbench;
6
7  architecture behavior of testbench is
8     --Component Declaration
9     component srlatch
10    port(R_L: in std_logic;
11         S_L: in std_logic;
12         Q: out std_logic;
13         QN: out std_logic);
14    end component;
15
16    --Inputs
17    signal R_L:  std_logic := '0';
18    signal S_L :  std_logic := '0';
19
20    --Outputs
21    signal Q: std_logic;
22    signal QN: std_logic;
23 begin
24    uut: srlatch port map(R_L => R_L,
25                          S_L => S_L,
26                          Q => Q,
27                          QN => QN);
```

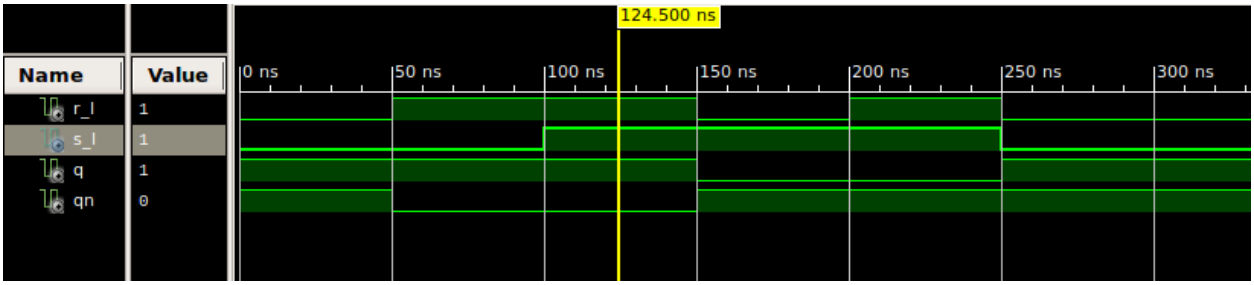
```
28    stim_proc: process
29    begin
30       s_l <= '0'; r_l <= '0'; wait for 50 ns;
31       s_l <= '0'; r_l <= '1'; wait for 50 ns;
32       s_l <= '1'; r_l <= '1'; wait for 50 ns;
33       s_l <= '1'; r_l <= '0'; wait for 50 ns;
34       s_l <= '1'; r_l <= '1'; wait for 50 ns;
35       s_l <= '0'; r_l <= '0'; wait for 50 ns;
36       wait;
37    end process;
38 end;
39
```



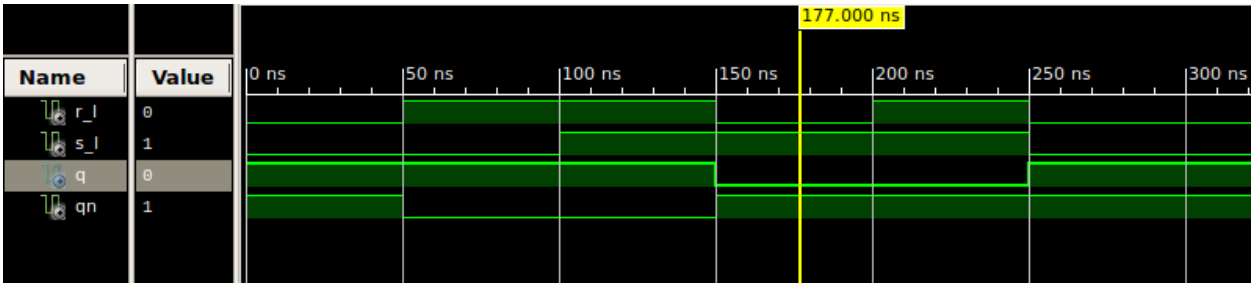

This outcome is correct because both S_L and R_L are negated, therefore Q and QN should be asserted which can be seen above.



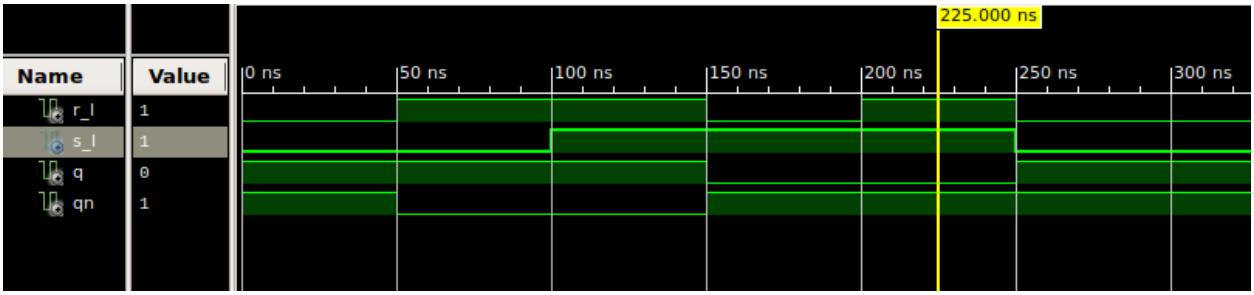

This outcome is correct because S_L is negated and R_L is asserted, therefore Q should be asserted and QN should be negated which can be seen above.



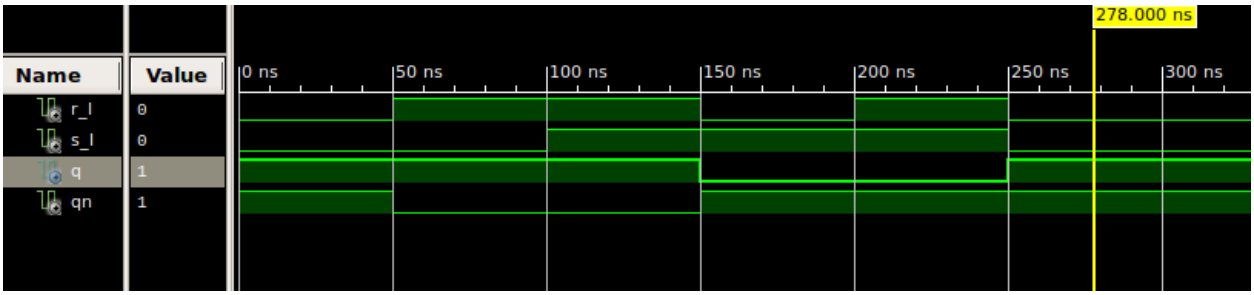

This outcome is correct because both S_L and R_L are asserted, therefore Q and QN should be the last values it had which in this case it is Q being asserted and QN being negated. This can be seen above.




This outcome is correct because S_L is asserted and R_L is negated, therefore Q should be negated and QN should be asserted which can be seen above.




This outcome is correct because both S_L and R_L are asserted, therefore Q and QN should be the last values it had which in this case it is Q being negated and QN being asserted. This can be seen above.

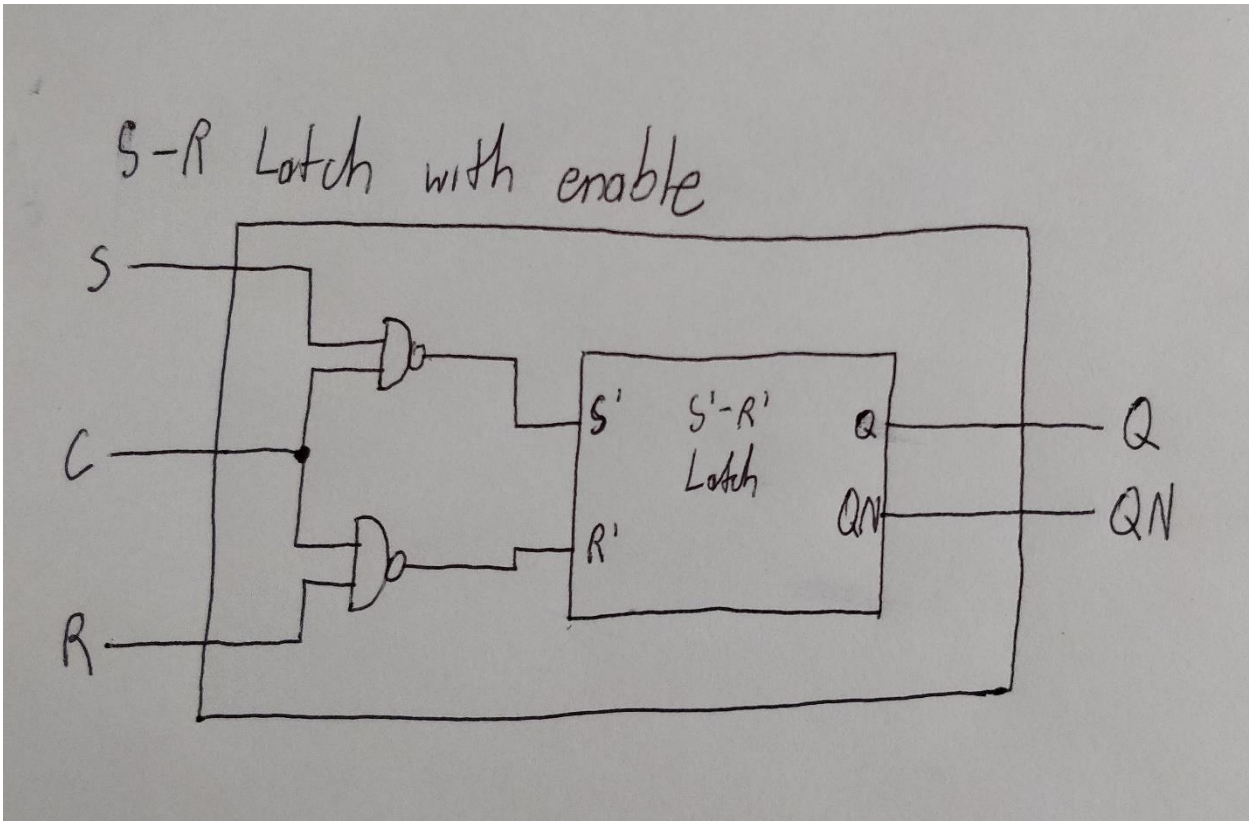



This outcome is correct because both S_L and R_L are negated, therefore Q and QN should be asserted which can be seen above.

## Task 2: S-R Latch with Enable

A S-R latch with enable is a type of sequential circuit that uses a S'-R' latch and two NAND gates to create a S'-R' latch that can be enabled and disabled using the enable input. It takes three inputs S, R, and C and outputs Q and QN based on the following truth table:

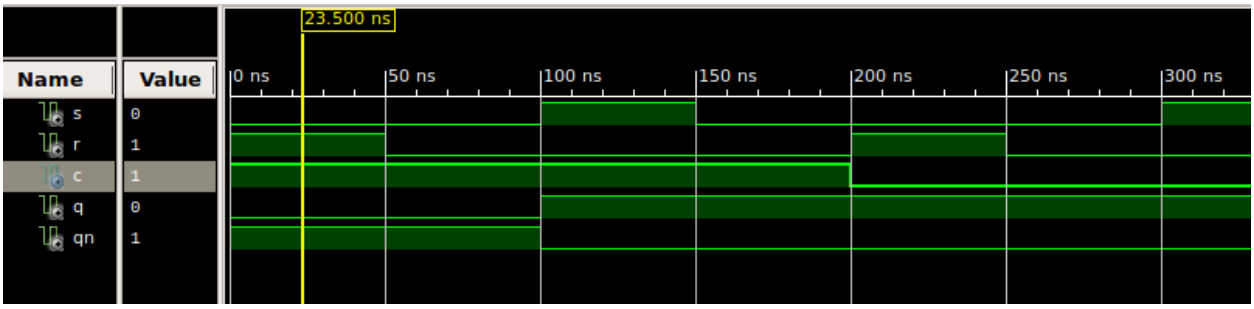
| S | R | C | Q | QN |
|---|---|---|---|---|
| 0 | 0 | 1 | Last Q | Last QN |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |
| X | X | 0 | Last Q | Last QN |



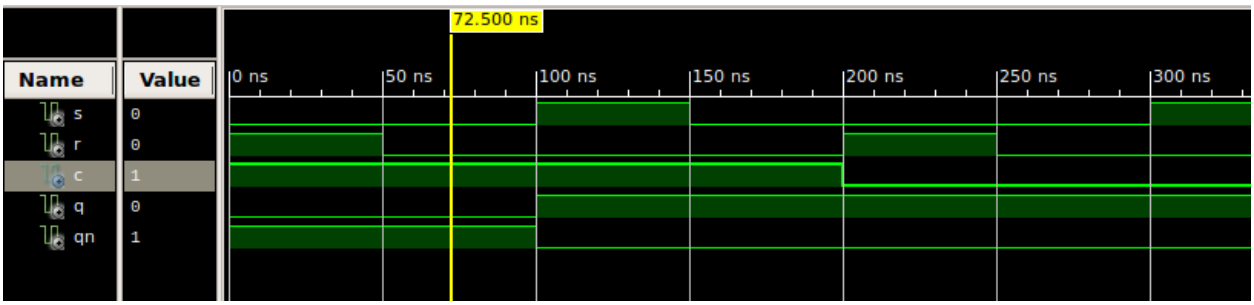
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity srlatchen is
   port(S: in std_logic;
        R: in std_logic;
        C: in std_logic;
        Q: out std_logic;
        QN: out std_logic);
end srlatchen;

architecture Behavioral of srlatchen is
   signal tmp1,tmp2: std_logic;
   component NAND2 port(I1,I0: in std_logic; O: out std_logic);
   end component;
   component srlatch port(R_L,S_L: in std_logic; Q,QN: out std_logic);
   end component;
begin
   U0: NAND2 port map(S,C,tmp1);
   U1: NAND2 port map(R,C,tmp2);
   U2: srlatch port map(tmp2,tmp1,Q,QN);
end Behavioral;

```

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

entity testbench is
end testbench;

architecture Behavior of testbench is
   --Component Declaration
   component srlatchen port(S: in std_logic;
                            R: in std_logic;
                            C: in std_logic;
                            Q: out std_logic;
                            QN: out std_logic);
   end component;

   --Inputs
   signal S: std_logic := '0';
   signal R: std_logic := '0';
   signal C: std_logic := '0';

   --Outputs
   signal Q: std_logic;
   signal QN: std_logic;
begin
   uut: srlatchen port map(S => S,
                           R => R,
                           C => C,
                           Q => Q,
                           QN => QN);
```

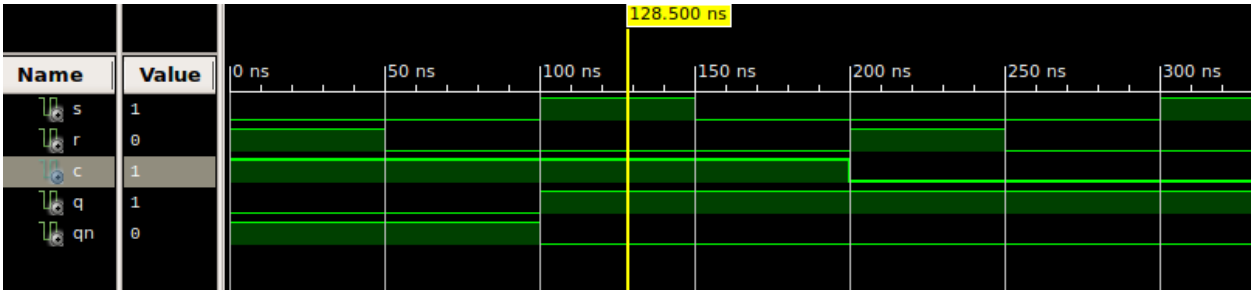
```
31      stim_proc: process
32      begin
33          --c is asserted
34          s <= '0'; r <= '1'; c <= '1'; wait for 50 ns;
35          s <= '0'; r <= '0'; c <= '1'; wait for 50 ns;
36          s <= '1'; r <= '0'; c <= '1'; wait for 50 ns;
37          s <= '0'; r <= '0'; c <= '1'; wait for 50 ns;
38
39          --c is negated
40          s <= '0'; r <= '1'; c <= '0'; wait for 50 ns;
41          s <= '0'; r <= '0'; c <= '0'; wait for 50 ns;
42          s <= '1'; r <= '0'; c <= '0'; wait for 50 ns;
43          s <= '0'; r <= '0'; c <= '0'; wait for 50 ns;
44          s <= '1'; r <= '1'; c <= '0'; wait for 50 ns;
45
46          --c is again asserted
47          s <= '1'; r <= '1'; c <= '1'; wait for 50 ns;
48
49          wait;
50      end process;
51  end;
```



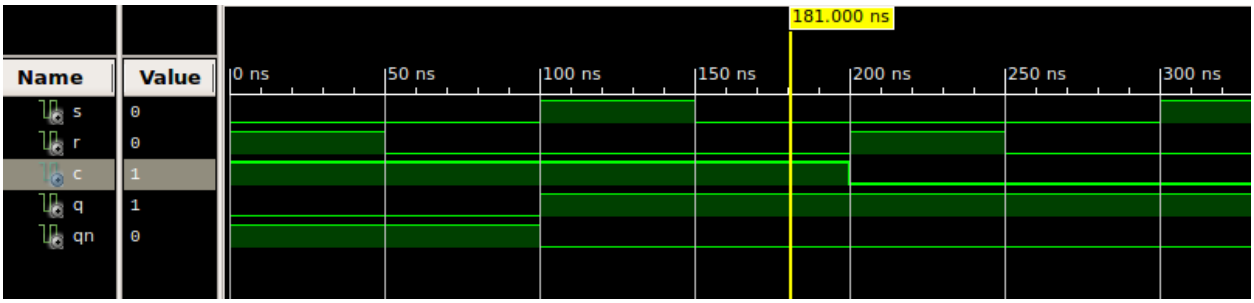

This outcome is correct because S is negated and both R and C are asserted, therefore Q should be negated and QN should be asserted which can be seen above.



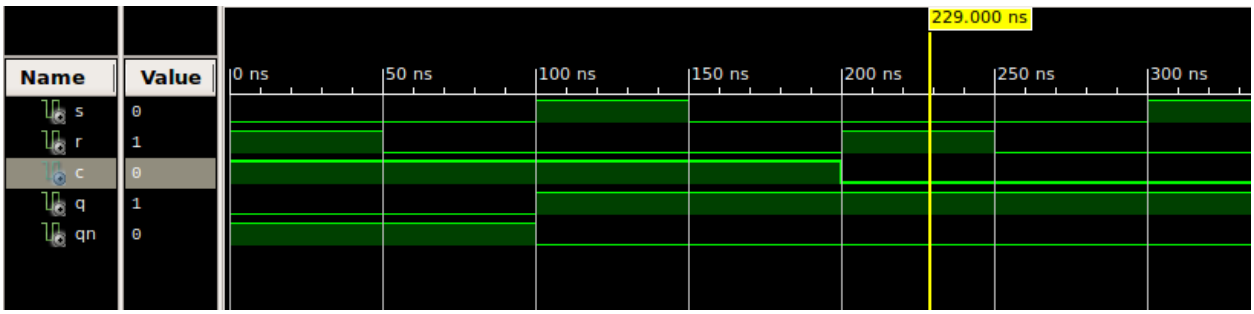
This outcome is correct because both S and R is negated and C is asserted, therefore Q and QN should be the last values it had which in this case it is Q being negated and QN being asserted. This can be seen above.



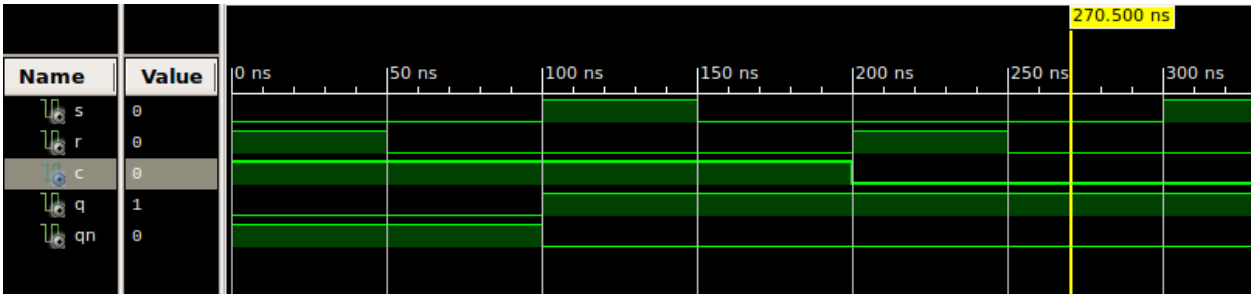

This outcome is correct because both S and C are asserted and R is negated, therefore Q should be asserted and QN should be negated which can be seen above.



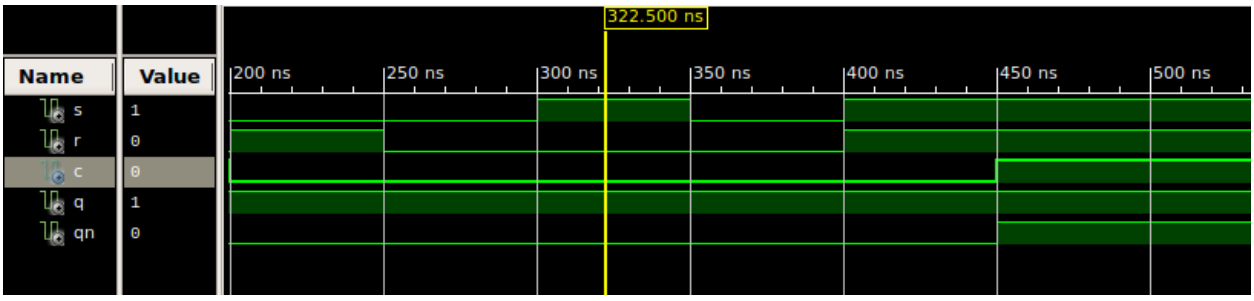

This outcome is correct because both S and R is negated and C is asserted, therefore Q and QN should be the last values it had which in this case it is Q being asserted and QN being negated. This can be seen above.



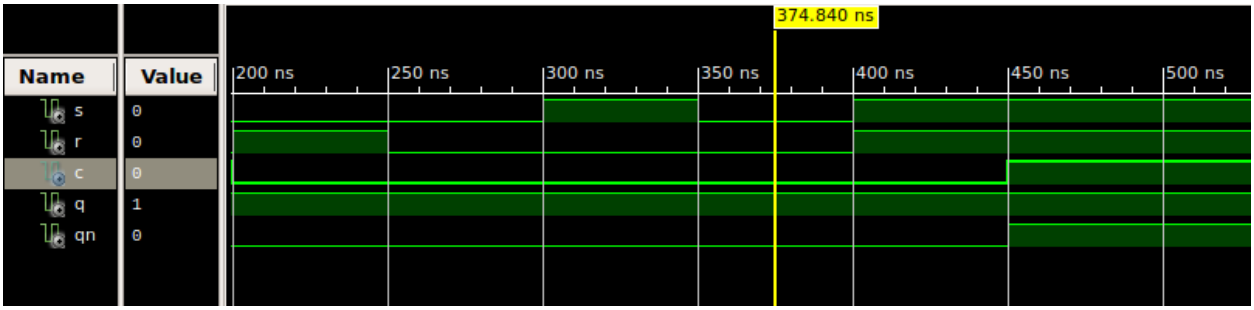

This outcome is correct because C is negated, therefore the values of S and R should be ignored and Q and QN should be the last values it had which in this case it is Q being asserted and QN being negated. This can be seen above.



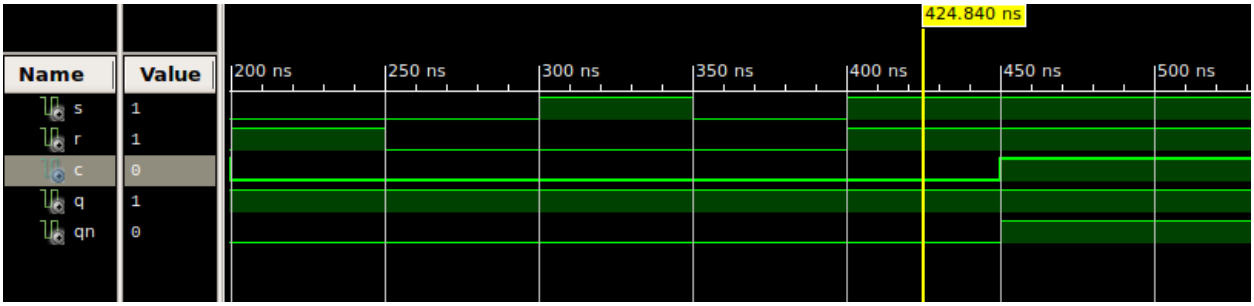

This outcome is correct because C is negated, therefore the values of S and R should be ignored and Q and QN should be the last values it had which in this case it is Q being asserted and QN being negated. This can be seen above.



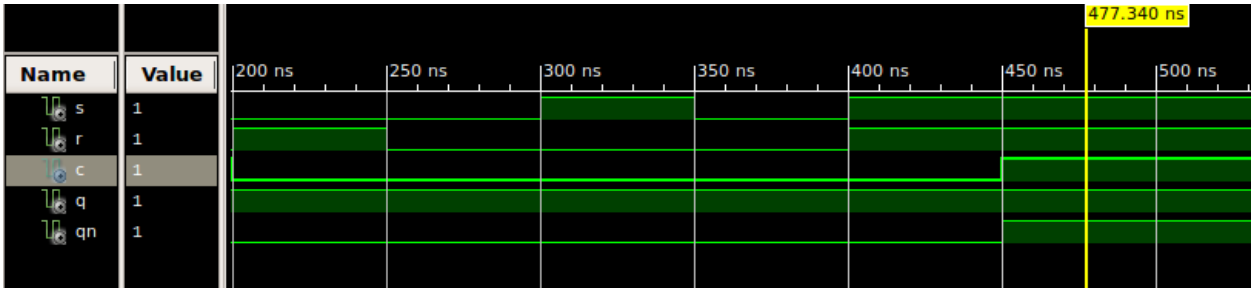

This outcome is correct because C is negated, therefore the values of S and R should be ignored and Q and QN should be the last values it had which in this case it is Q being asserted and QN being negated. This can be seen above.




This outcome is correct because C is negated, therefore the values of S and R should be ignored and Q and QN should be the last values it had which in this case it is Q being asserted and QN being negated. This can be seen above.




This outcome is correct because C is negated, therefore the values of S and R should be ignored and Q and QN should be the last values it had which in this case it is Q being asserted and QN being negated. This can be seen above.

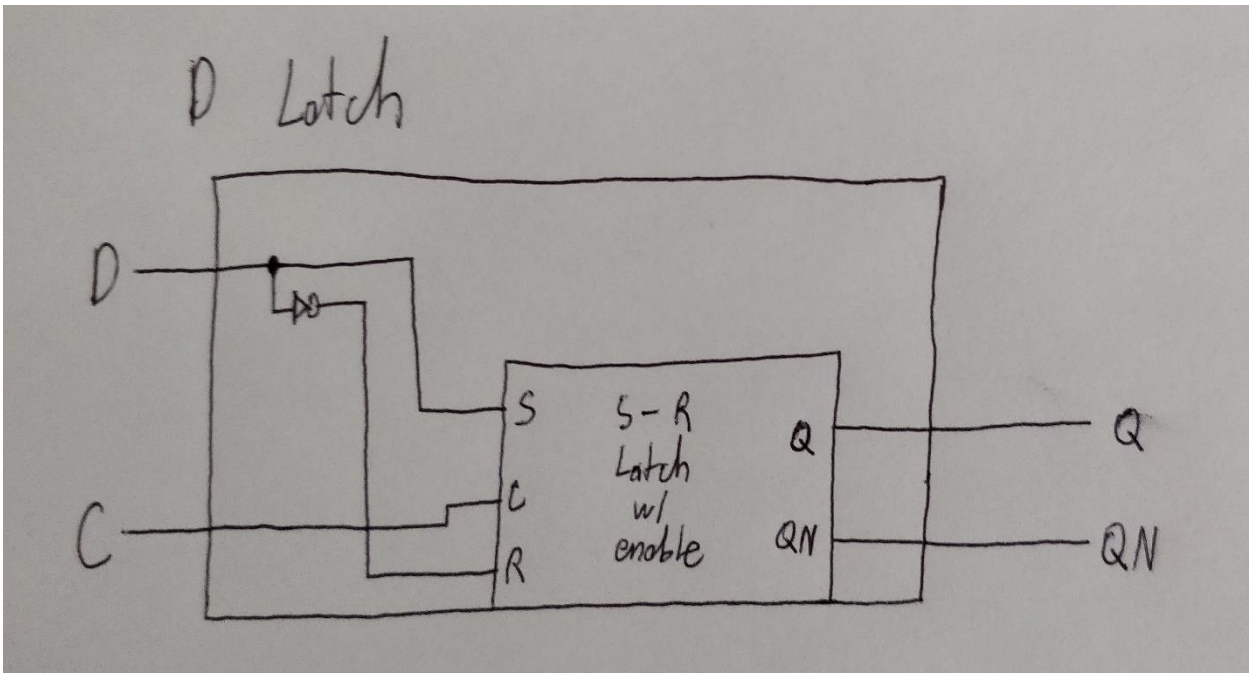



This outcome is correct because S, R, and C are all asserted, therefore Q should be asserted and QN should be asserted which can be seen above.

## Task 1: D Latch

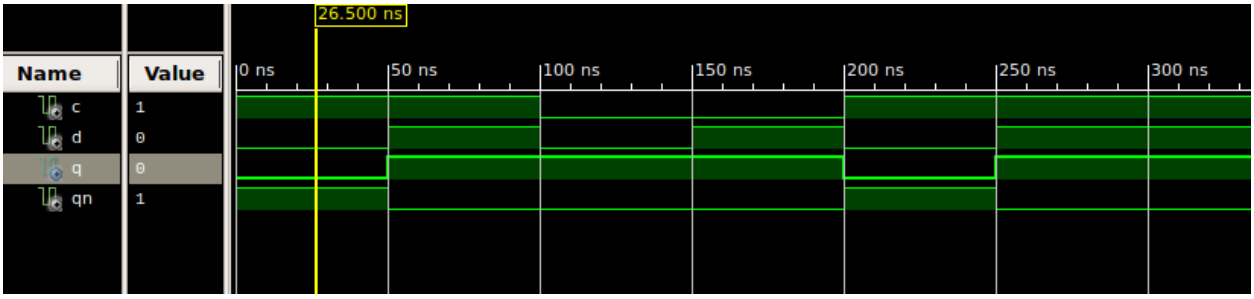
A D latch is a type of sequential circuit that uses a S-R latch with enable and an invertor to create a D latch that can be used like the S-R latch with enable but has less states allowing for fewer points of failure. It takes two inputs D and C and outputs Q and QN based on the following truth table:

| C | D | Q | QN |
|---|---|---|---|
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |
| 0 | X | Last Q | Last QN |

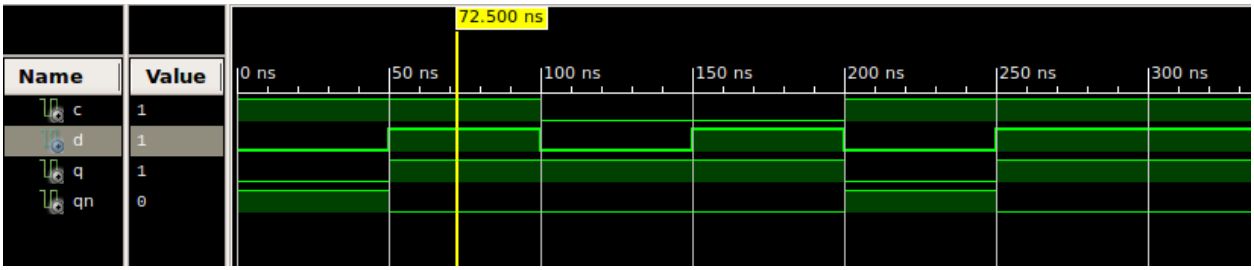



```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  library UNISIM;
4  use UNISIM.VComponents.all;
5
6  entity dlatch is
7     port(C: in std_logic;
8          D: in std_logic;
9          Q: out std_logic;
10         QN: out std_logic);
11 end dlatch;
12
13 architecture Behavioral of dlatch is
14    signal D_L: std_logic;
15    component INV port (I: in std_logic; O: out std_logic);
16    end component;
17    component srlatchen port(S,R,C: in std_logic; Q,QN: out std_logic);
18    end component;
19 begin
20    U0: INV port map(D,D_L);
21    U1: srlatchen port map(D,D_L,C,Q,QN);
22 end Behavioral;
```

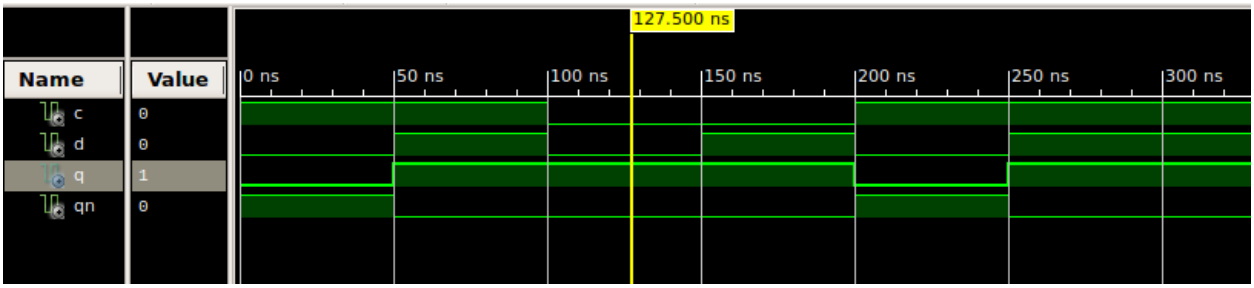
```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  entity testbench is
5  end testbench;
6
7  architecture behavior of testbench is
8     --Component Declaration
9     component dlatch port(
10       C : in  std_logic;
11       D : in  std_logic;
12       Q : out  std_logic;
13       QN : out  std_logic);
14    end component;
15
16    --Inputs
17    signal C : std_logic := '0';
18    signal D : std_logic := '0';
19
20    --Outputs
21    signal Q : std_logic;
22    signal QN : std_logic;
23 begin
24    uut: dlatch port map(C => C,
25                         D => D,
26                         Q => Q,
27                         QN => QN);
28
28
29    stim_proc: process
30    begin
31       d <= '0'; c <='1'; wait for 50 ns;
32       d <= '1'; c <='1'; wait for 50 ns;
33       d <= '0'; c <='0'; wait for 50 ns;
34       d <= '1'; c <='0'; wait for 50 ns;
35       d <= '0'; c <='1'; wait for 50 ns;
36       d <= '1'; c <='1'; wait for 50 ns;
37       wait;
38    end process;
39 end;
```



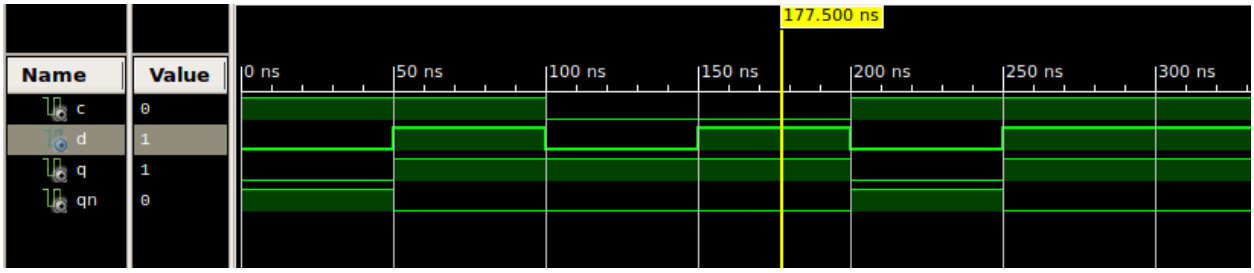

This outcome is correct because C is asserted and D is negated, therefore Q should be negated and QN should be asserted which can be seen above.



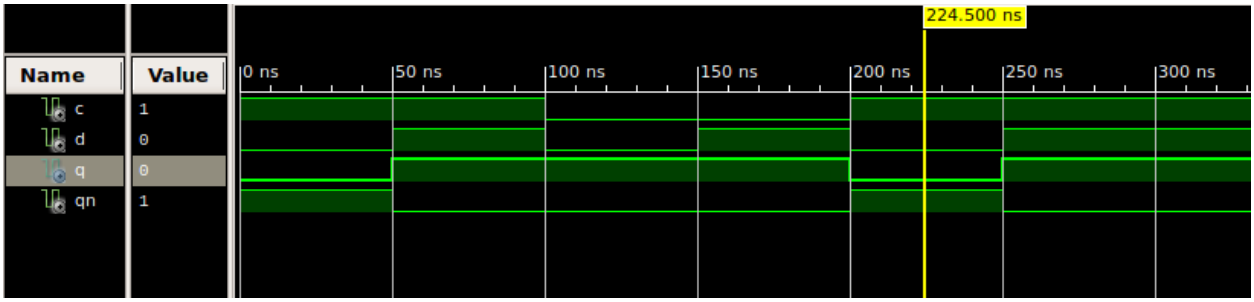

This outcome is correct because both C and D are asserted, therefore Q should be asserted and QN should be negated which can be seen above.



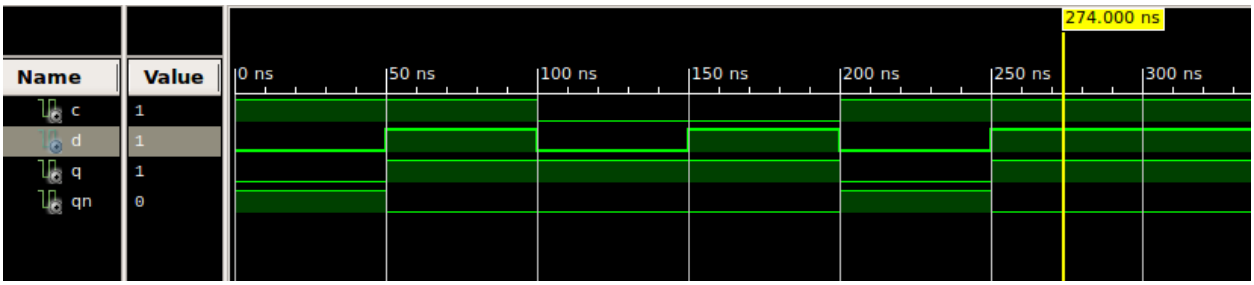

This outcome is correct because C is negated, therefore the value D should be ignored and Q and QN should be the last values it had which in this case it is Q being asserted and QN being negated. This can be seen above.



This outcome is correct because C is negated, therefore the value D should be ignored and Q and QN should be the last values it had which in this case it is Q being asserted and QN being negated. This can be seen above.




This outcome is correct because C is asserted and D is negated, therefore Q should be negated and QN should be asserted which can be seen above.




This outcome is correct because both C and D are asserted, therefore Q should be asserted and QN should be negated which can be seen above.