Adama Gamby – 110955362

Judah Ben-Eliezer – 112352727

ESE 382 Laboratory 7

Lab Section 1

1.

The Latch is 0 and the FF is U - undriven

In an actual PLD the Latch would still be zero, or whatever value was on D, if the clock idled at a low level.
This is becasue the latch would be transperant. If the clock idled on a high level Qlatch would be in a random state based on noise that was
initially present before the signals were driven.

Same for the FF. U is not a valid signal in the real world. QFF would be based on noise that was present on the PLD signals before the signals
were connected in the PLD.

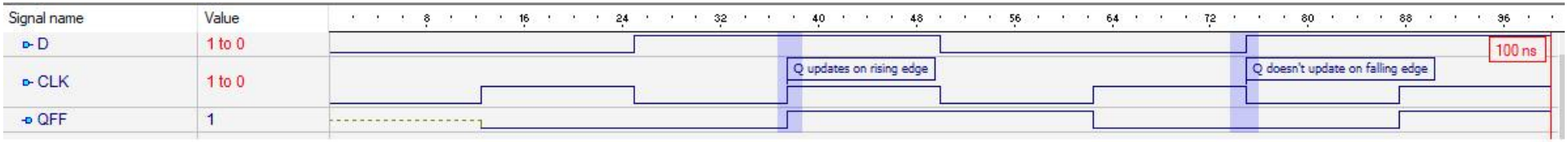
2. Requires auto test bench generation which is not posisble in the student version. I emailed the proffessor and he said there would be no penalty

--From lecture the clock signal must be type bit or type ulogic with subtyes '0' or '1' only being used for the clock edges. This is only way
that a clock signal is synthesizable and I assume this is what the auto generated tb does

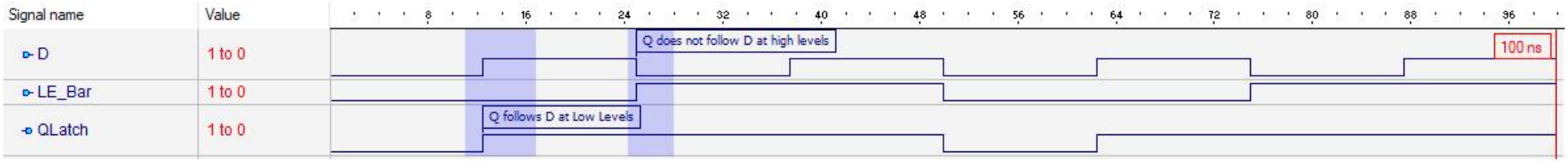
3. Requires auto test bench generation which is not posisble in the student version. I emailed the proffessor and he said there would be no penalty

4.It is important to see the code to graphics in a structural diagram because you can visually confirm that all your componenets are properly connected. The code to graphics in particular has
the added advantage of highlingting any unconnected nets in the corner of the screen. This makes for an
easy debugging experience.

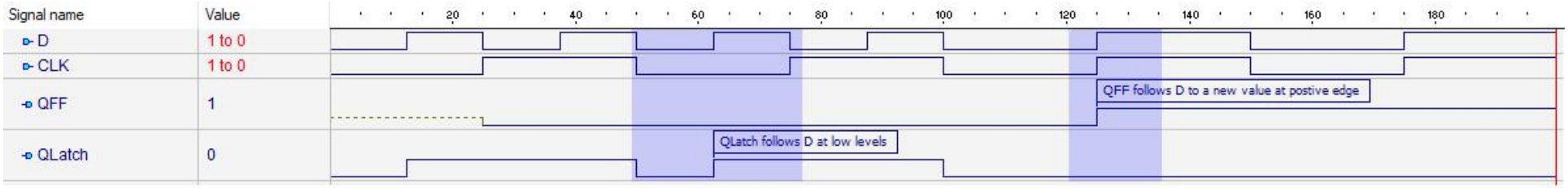
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity DFF is
    port(
    D : in std_logic;
    CLK : in std_logic;
    QFF : out std_logic
    );
end entity;

architecture behavioral of DFF is
-- declartive part of the architecture, signals can be declared here, only
begin
    ff: process(CLK)
    -- variables can be declared here (only)
    begin
    if rising_edge(CLK) then
    QFF <= D; -- on edge triggers Q gets D
    end if;
    end process;
end behavioral;
```


Signal name
Value
8
16
24
32
40
48
56
64
72
80
88
96
D
1 to 0
CLK
1 to 0
QFF
1
100 ns
Q updates on rising edge
Q doesn't update on falling edge

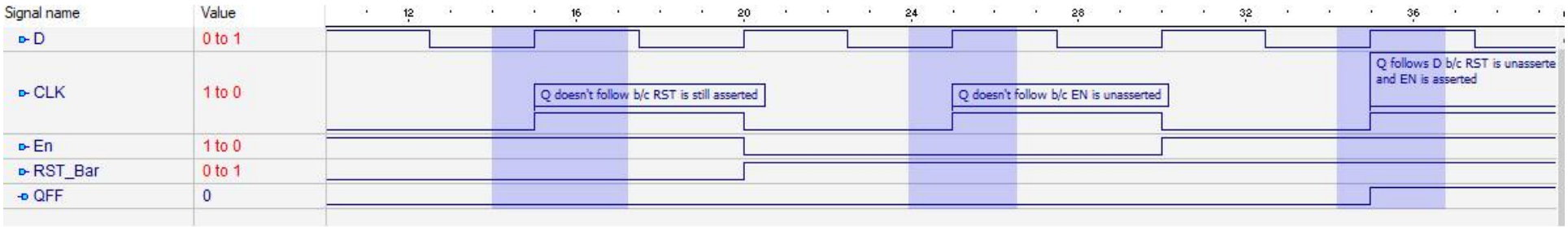
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity DLatch is
    port(
    D : in std_logic;
    LE_Bar: in std_logic;
    QLatch: out std_logic
    );
end entity;

architecture behavioral of DLatch is
--architecture declartive part, signals can be declared here
begin

    Level: process(LE_Bar,D) --complete sensitivity list
    --variabels can be delcared here only
    begin

    if LE_Bar = '0' then
        QLatch <= D;
        --infered latch when others
    end if;


    end process;

end architecture;
```


Signal name
Value
8
16
24
32
40
48
56
64
72
80
88
96
D
1 to 0
Q does not follow D at high levels
100 ns
LE_Bar
1 to 0
QLatch
1 to 0
Q follows D at Low Levels

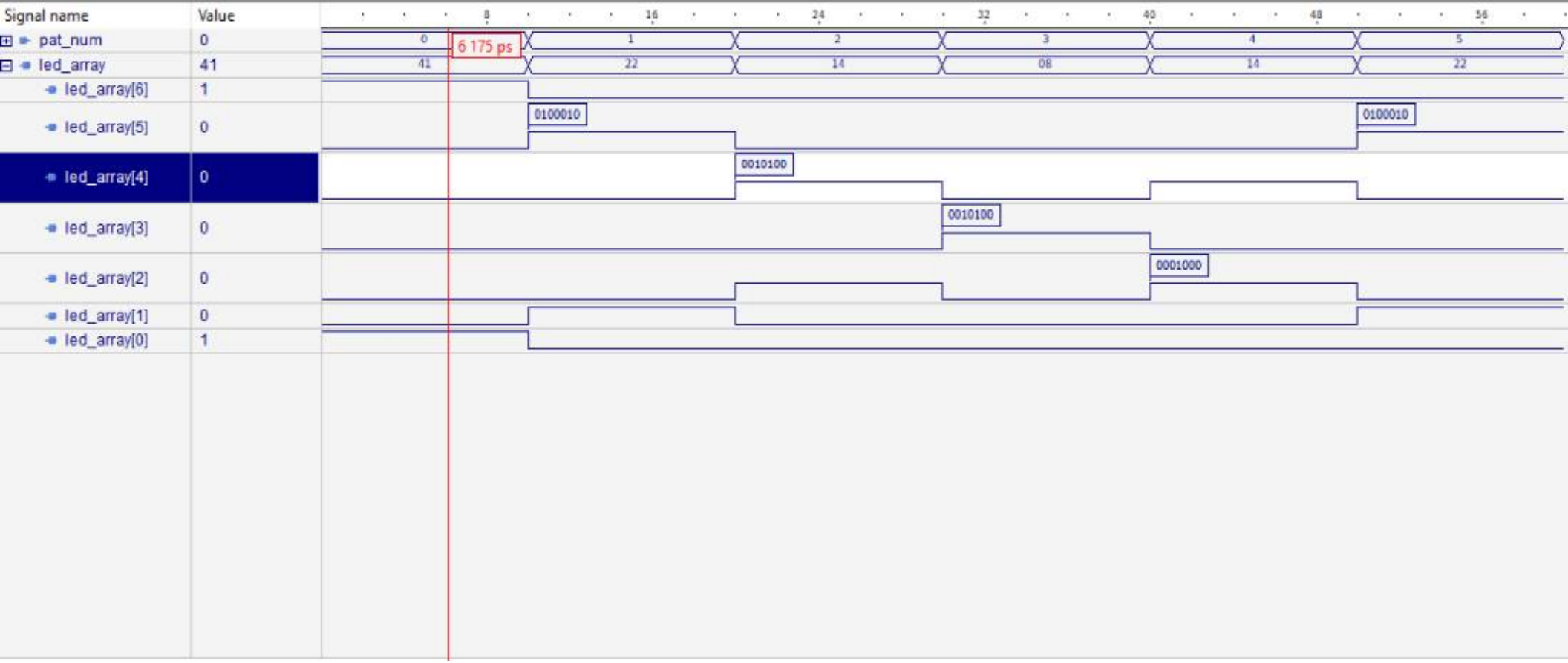
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity FF_v_Latch is
    port(
    D: in std_logic;
    CLK : in std_logic;
    QFF : out std_logic;
    QLatch: out std_logic
    );
end entity;

architecture strcutural of FF_v_Latch is
--declartive part, can declare signals here

signal s1: std_logic;
signal s2: std_logic;

begin
u1: entity DFF port map( D => s1, CLK => s2, QFF => QFF);
u2: entity DLatch port map(D => s1, LE_Bar => s2, QLatch => QLatch);
    s1 <= D;
    s2 <= CLK;
end architecture;
```


Signal name
Value
20
40
60
80
100
120
140
160
180
D
1 to 0
CLK
1 to 0
QFF
1
QFF follows D to a new value at postive edge
QLatch
0
QLatch follows D at low levels

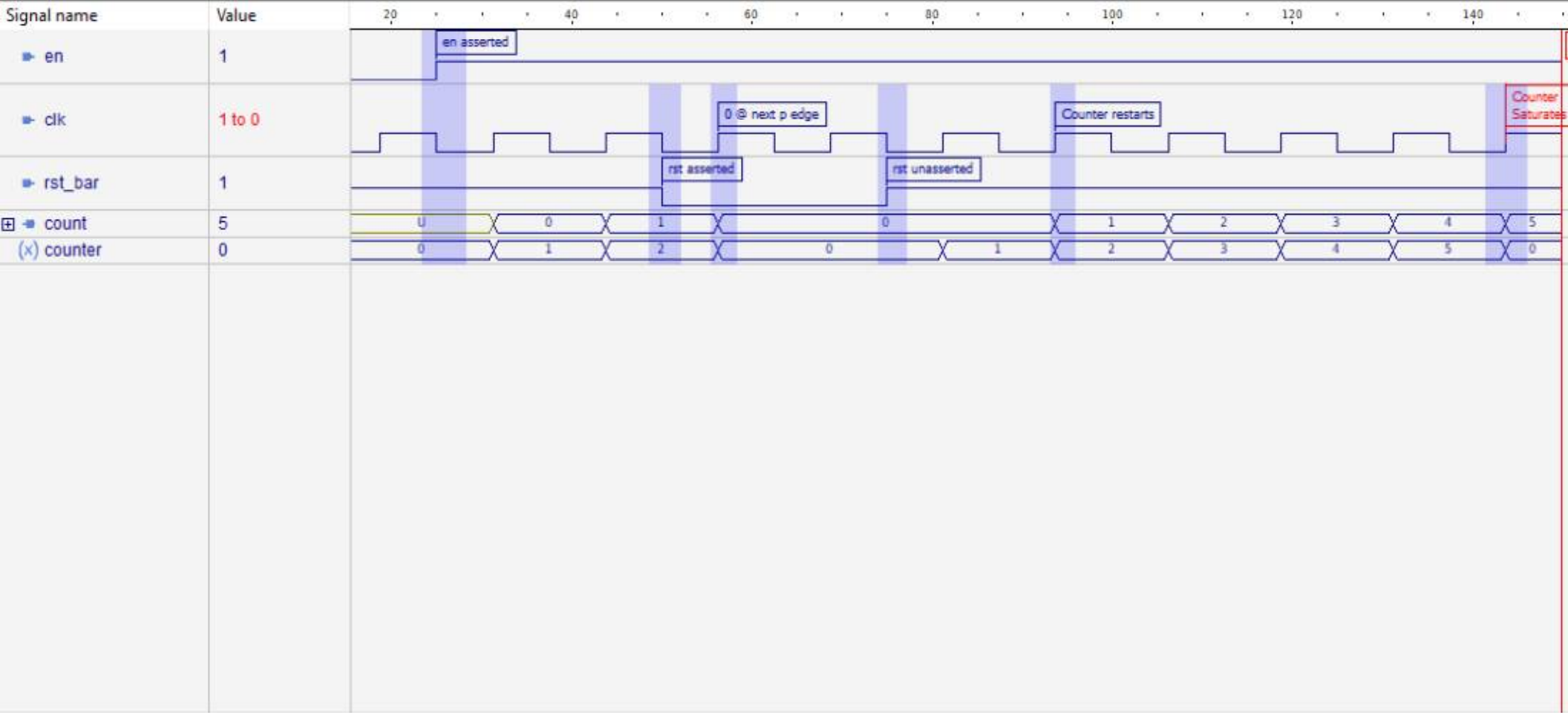
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity DFF_En is

    port(
    D : in std_logic;
    CLK : in std_logic;
    En : in std_logic;
    RST_Bar: in std_logic;
    QFF : out std_logic
    );

end entity;

architecture behavioral of DFF_En is
-- declartive part of the architecture, signals can be declared here, only
begin

    ff_en: process(CLK, RST_Bar)  -- Reset is asynchronus
    -- variables can be declared here (only)
    begin

    if RST_Bar = '0' then

    QFF <= '0';

    elsif En = '1' then
        if rising_edge(CLK) then
            QFF <= D;
        end if;

    end if;

    end process;

end behavioral;
```


Signal name
Value
12
16
20
24
28
32
36
D
0 to 1
CLK
1 to 0
En
1 to 0
RST_Bar
0 to 1
QFF
0
Q doesn't follow b/c RST is still asserted
Q doesn't follow b/c EN is unasserted
Q follows D b/c RST is unasserte
and EN is asserted

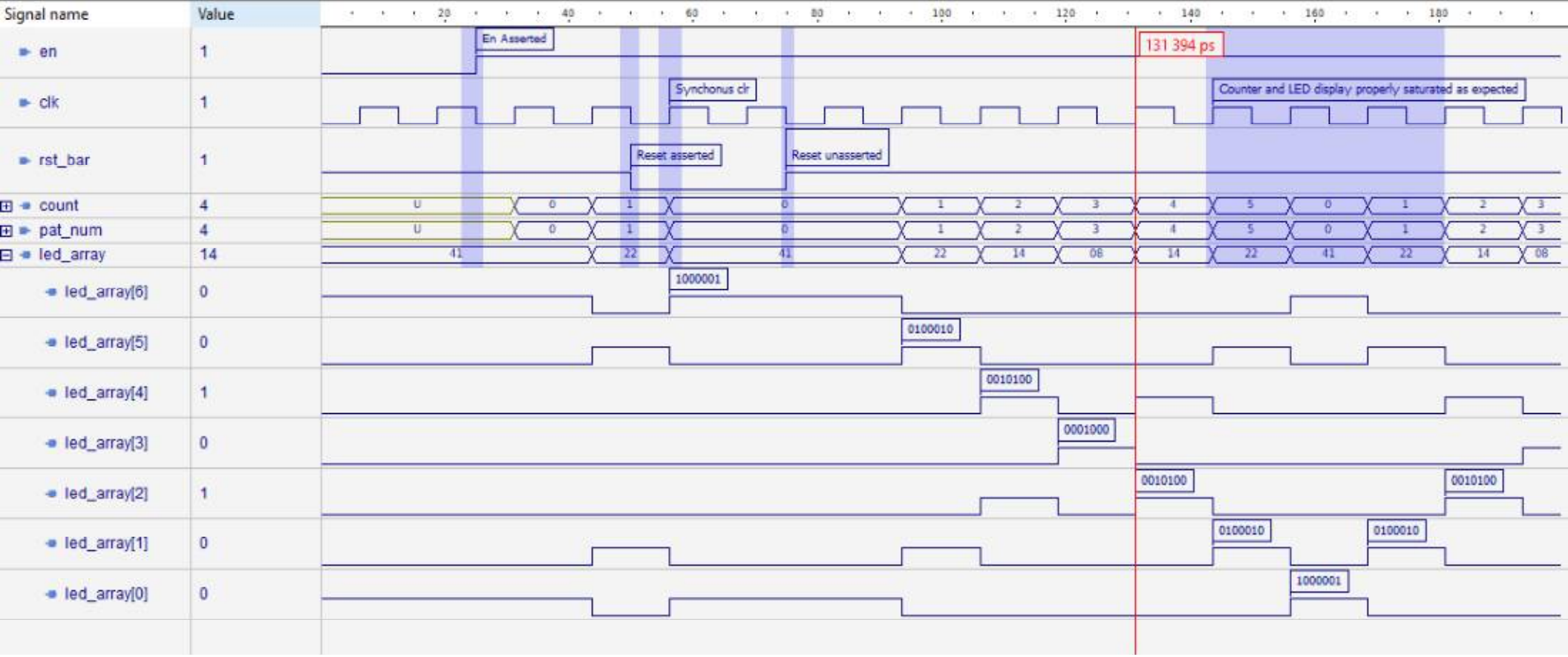
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity pattern_gen is
    port(
    pat_num: in std_logic_vector (2 downto 0);
    led_array: out std_logic_vector (6 downto 0)
    );
end entity;

architecture tablelookup of pattern_gen is
--can declare signals here


    type pattern_array is array (0 to 5) of std_logic_vector (6 downto 0);

    constant patterns : pattern_array := (
    "1000001",
    "0100010",
    "0010100",
    "0001000",
    "0010100",
    "0100010");

begin

    pat_gen: process(pat_num)

    begin

        led_array <= patterns(to_integer(unsigned(pat_num)));

    end process;

end architecture;

```


Signal name
Value
8
16
24
32
40
48
56
pat_num
0
0
6 175 ps
1
2
3
4
5
led_array
41
41
22
14
08
14
22
led_array[6]
1
led_array[5]
0
0100010
0100010
led_array[4]
0
0010100
led_array[3]
0
0010100
led_array[2]
0
0001000
led_array[1]
0
led_array[0]
1

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity counter is
    port(
    en: in std_logic;
    clk: in std_logic;
    rst_bar: in std_logic;
    count: out std_logic_vector (2 downto 0)
    );
end entity;

architecture int_count of counter is
--declaritve part of the arch signals can be delcared here
begin
    my_counter: process(CLK)

    --delcrative part of the proc, variables can be declared here
    variable counter : integer := 0;

    begin
    if(rising_edge(CLK)) then

    if(rst_bar = '0') then
        count <= "000";
        counter := 0;
    elsif (en = '1') then

        case counter is

            when 5 =>
            count <= std_logic_vector(to_unsigned(counter,3));
            counter := 0;

            when others =>
            count <= std_logic_vector(to_unsigned(counter,3));
            counter := counter + 1;

        end case;

    end if;

    end if;

    end process;


end architecture;
```

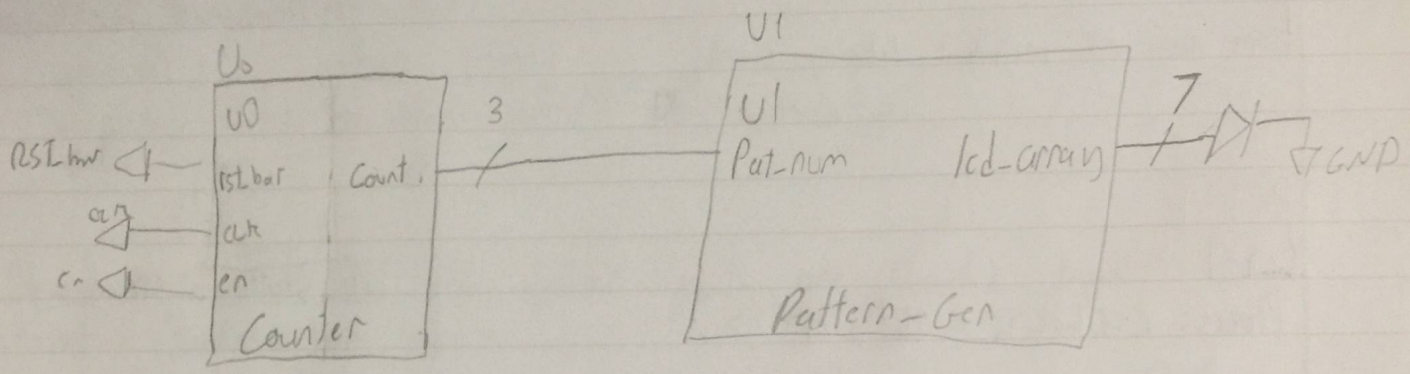

Signal name
Value
20
40
60
80
100
120
140
en
1
en asserted
clk
1 to 0
0 @ next p edge
Counter restarts
Counter Saturates
rst_bar
1
rst asserted
rst unasserted
count
5
counter
0

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library work;
use work.all;

entity mesmerizer is
    port(
    clk: in std_logic;
    rst_bar: in std_logic;
    en: in std_logic; -- this was not in the requirements but necessary for my design. I assumed it was an oversight
    led_array: out std_logic_vector (6 downto 0)
    );
end entity;

architecture structural of mesmerizer is
-- signals can be decalred here
signal b1: std_logic_vector (2 downto 0);

begin

 u0: entity counter port map(rst_bar => rst_bar, clk => clk, en => en, count => b1);
 u1: entity pattern_gen port map(pat_num => b1, led_array => led_array);

end architecture;
```


Signal name
Value
20
40
60
80
100
120
140
160
180
en
1
clk
1
rst_bar
1
count
4
pat_num
4
led_array
14
led_array[6]
0
led_array[5]
0
led_array[4]
1
led_array[3]
0
led_array[2]
1
led_array[1]
0
led_array[0]
0
En Asserted
131 394 ps
Synchonus clr
Counter and LED display properly saturated as expected
Reset asserted
Reset unasserted
1000001
0100010
0010100
0001000
0010100
0010100
0100010
0100010
1000001

Lab07

Task 3


U0
U1
RSLbar
clk
en
rstLbar
Count
3
Pat_num
lcd_arrays
7
GND
Counter
Pattern_Gen

```
DFF:
    Must verify:
    Postive edge trigger, Q follows D
    Negative edge, Q does not follow D

    Test Waveform:
    CLK is half period of D
    --> covers all conditions and shows operation every other cycle

DLatch:
    Must verify:
    negative level trigger, Q follows D
    postive level, Q does not follow D

    Test waveform:
    D is half the period of CLK
    --> shows which level the CLK triggers on

DFF v DL:
    Must verify:
    Difference between QFF and QLatch
    Latch and FF work as expected

    Testwaveform:
    D half the period of clock the first half
    CLK half hte priod of D the second hafl
    --> show both working as intedned and the difference in their outpus
```

```
DFF_En
    Must verify:
    Postive edge trigger when enable asserted and reset unasserted
    Q does not follow D when en is unasserted and reset is unasserted
    Q is 0 regardles of D when rst is asserted

    Test Waveform:
    D is half period of CLK
    CLK is half the period of EN
    EN is half the period of RST_Bar
    --> covers all conditions and shows operation in the last quarter
```

```
Counter:
    Must Verify:
    Counter is synchronusly set to zero at clock edge when rst is asserted
    Ouput count ups as expected when enable is assert and rst is unasserted
    variable is incremented as expected when enabled is asserted and rst is unasserted

    Test waveform:
    Reset starts unasserted, asserts in the middle of a count then unasserts;
    EN starts unasserted then asserts
    CLK is a clock;
    --> Test all condintions and verifies operaiton

Pattern_gen:
    Must Verify:
    Outputs the correct led patterns

    Test waveform:
    Counter on patt_num input signal
    --> confirms output as expected

Mesmerize:
    Must verify:
    Counter is working as intended (probe bus)
    Pattern gen is working as intended
    Output is the correct pattern

    Test waveform:
    Reset starts unasserted, asserts in the middle of a count then unasserts;
    EN starts unasserted then asserts
    CLK is a clock;
```