Programable Logic Circuts
Lab 03

Sølve Kjelseth

September 5, 2025

0. Introduction

This is the third report in this course, detailing the completion of the third lab exercise. With the purpose is to investigate latches, flip-flops, and registers.

Note: The VHDL and LaTeX code is open source github link.

0.0. Table of Contents

0.1. List of Figures

0.2. List of Tables

1. Part 1

The purpose of this task was to examine the structure of the code, both to see how it was synthesized and also to simulate it to see if it was working.

1.0. Code

Code 1.0: Code with equal function to the one included in the task

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

entity L03P01 is
    port(Clk, R, S : in  std_logic;
         Q         : out std_logic);
end L03P01;

architecture structural of L03P01 is
    signal R_g, S_g, Qa, Qb : std_logic;
    attribute keep : boolean;
    attribute keep of R_g, S_g, Qa, Qb : signal is true;
begin
    R_g <= R and Clk;
    S_g <= S and Clk;
    Qa  <= not(R_g or Qb);
    Qb  <= not(S_g or Qa);
    Q   <= Qa;
end structural;
```

1.1. RTL

The synthesization of the VHDL code is shown in figure 1.0. It looks logically equal to the schematic provided by the task.

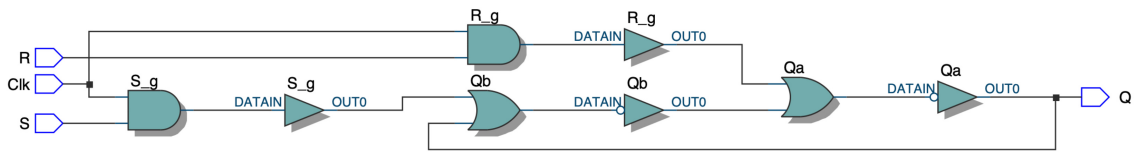


Figure 1.0: RTL for the TLE

1.2. Results

Doing a simulation using Quartus, the resulting output is as expected. Simulation both R and S to be on at the same time resulted in some out of bounds calculating and the simulation timing out. Needless to say it does not really make sense to have such an input anyway and therefore is omitted. Output is as expected.

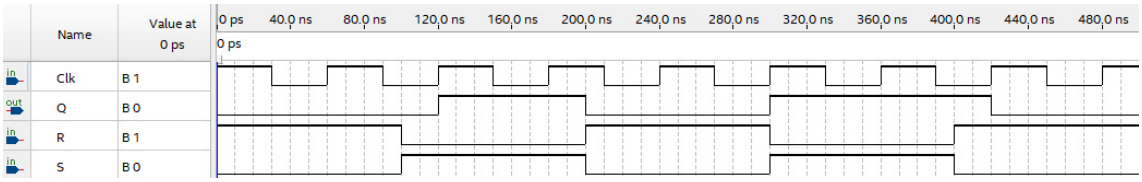


Figure 1.1: Simulation results

2. Part 2

The purpose of this task was to first make a D-latch and then make another project that implemented it as an instance of a component and testing it on the board.

Note: For the first part only, the standalone D-latch, I made a mistake when restructuring the code for readability. Looking at code 2.0 at line 18 and 19. Here `R_g` and `S_g` should be swapped. This makes the circuit equivalent to either having swapped inputs or inverted the output.

2.0. Code

Code 2.0: Simple D-latch with a small I/O mistake

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

entity L03P02_1 is
    port(Clk, D : in  std_logic;
         Q      : out std_logic);
end L03P02_1;

architecture structural of L03P02_1 is
    signal R, S, R_g, S_g, Qa, Qb : std_logic;
    attribute keep : boolean;
    attribute keep of R, S, R_g, S_g, Qa, Qb : signal is true;
begin
    R   <= not(D);
    S   <= D;
    R_g <= not(R and Clk);
    S_g <= not(S and Clk);
    Qa  <= not(R_g and Qb);
    Qb  <= not(S_g and Qa);
    Q   <= Qa;
end structural;
```

Now to implement this as a component, the TLE was first made. Import of such a component and mapping it to the inputs and outputs used by the FPGA was done in the TLE. Then the code 2.0 was copied over to code 2.2, a more general VHDL file where I/O is not tied to the FPGA board. This was done before the error noted above and therefore will not follow through more tasks.

Code 2.1: TLE importing a D-latch

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

entity L03P02_2 is
    port(SW   : in  std_logic_vector(1 downto 0);
         LEDR : out std_logic_vector(9 downto 0));
end L03P02_2;


architecture structural of L03P02_2 is
    component Dl
        port(Clk, D : in  std_logic;
             Q      : out std_logic);
    end component;
begin
    LEDR(9 downto 1) <= "000000000";
    D0: Dl port map(
        Clk => SW(1),
        D   => SW(0),
        Q   => LEDR(0));
end structural;
```

Code 2.2: General D-latch component

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

entity Dl is
    port(Clk, D : in  std_logic;
         Q      : out std_logic);
end Dl;

architecture structural of Dl is
    signal R, S, R_g, S_g, Qa, Qb : std_logic;
    attribute keep : boolean;
    attribute keep of R, S, R_g, S_g, Qa, Qb : signal is true;
begin
    R   <= not(D);
    S   <= D;
    R_g <= not(R and Clk);
    S_g <= not(S and Clk);
    Qb  <= not(R_g and Qa);
    Qa  <= not(S_g and Qb);
    Q   <= Qa;
end structural;
```

2.1. RTL

RTL of the first D-latch with the I/O mistake is omitted as there is no value in adding it to the report, when the general D-latch shown in the expanded RTL in figure 2.1 is the same without the error.

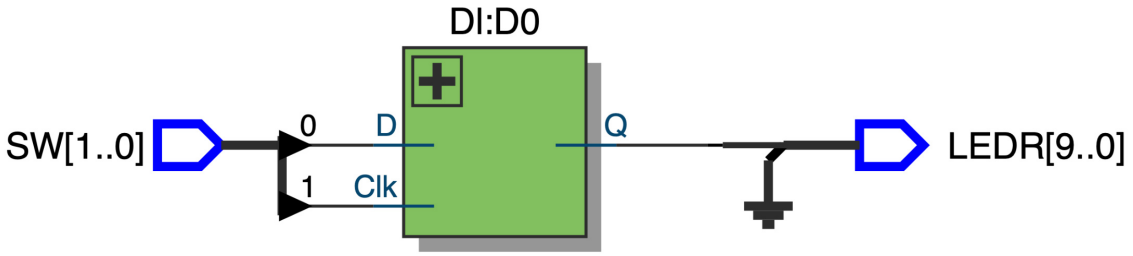


Figure 2.0: TLE with a D-latch component

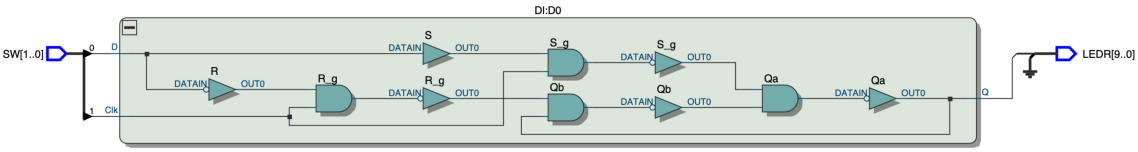


Figure 2.1: The D-latch component expanded

2.2. Results

The first part was simulated, here you can see that the input/output is reversed to what it should have been. Implementing the second part code on the FPGA the results was as

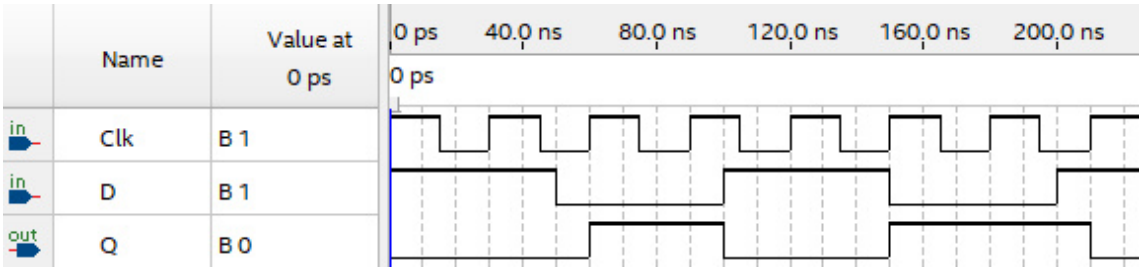
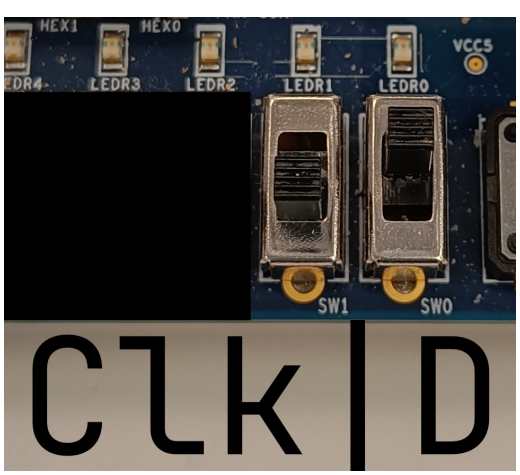
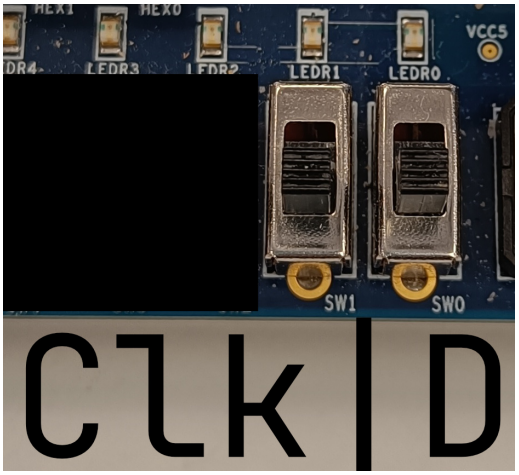


Figure 2.2: Simulation of the first D-latch with the I/O mistake

expected. Here it is clear that when `Clk` is high, `D` is continuously overwriting the output LED and when `Clk` is low, the output is equal to the last saved state of `D`.



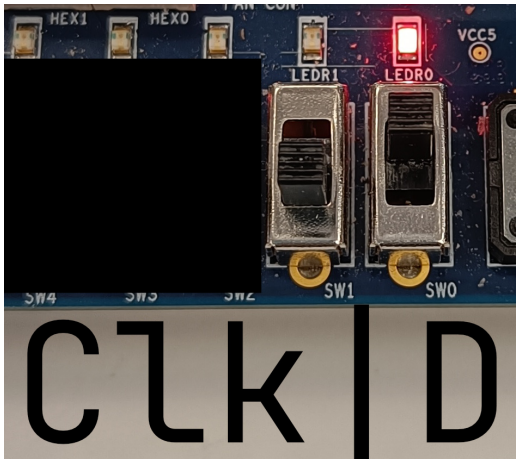
(a) Clk low, D high not registered



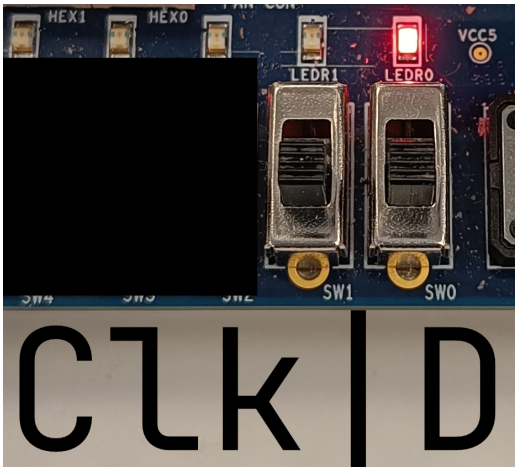
(b) Clk low, D low already stored

(c) Clk high, D low already stored

(d) Clk high, D high registered



(e) Clk low, D high already stored



(f) Clk low, D low not registered

Figure 2.3: D-latch on the FPGA using a component works as expected

3. Part 3

The purpose of this task was to continue last part and instead of a single instance of a D-latch, use two instances, to make a D-flip-flop.

3.0. Code

Code 3.0: TLE using two instances of a D-latch

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

entity L03P03 is
    port(SW   : in  std_logic_vector(1 downto 0);
         LEDR : out std_logic_vector(9 downto 0));
end L03P03;


architecture structural of L03P03 is
    component Dl
        port(Clk, D : in  std_logic;
             Q      : out std_logic);
    end component;
    signal m : std_logic;
begin
    LEDR(9 downto 1) <= "000000000";
    D0: Dl port map(
        Clk => not(SW(1)),
        D   => SW(0),
        Q   => m);
    D1: Dl port map(
        Clk => SW(1),
        D   => m,
        Q   => LEDR(0));
end structural;
```

The D-latch component used here is code 2.2.

3.1. RTL

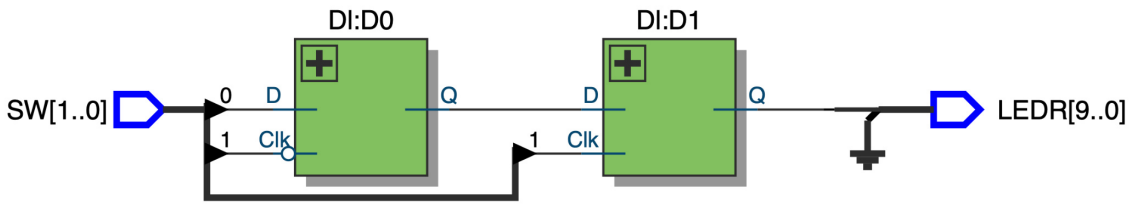
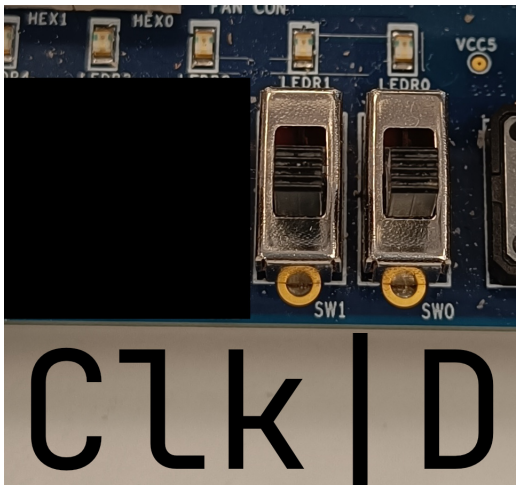


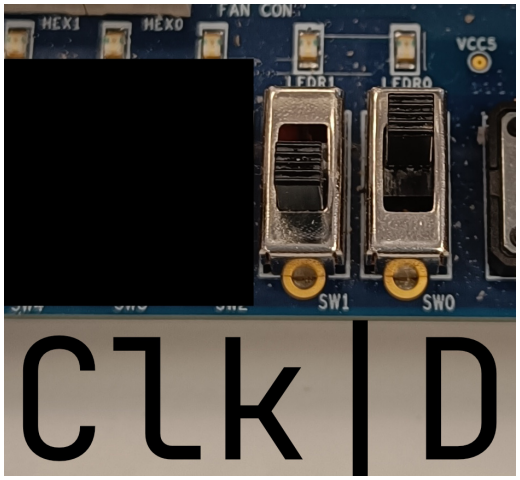
Figure 3.0: RTL of the TLE with two instances

3.2. Results

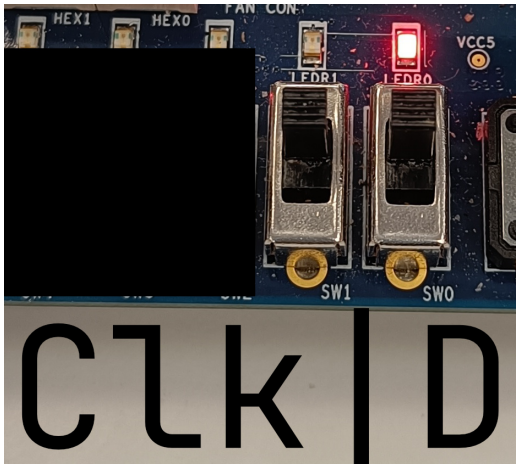
Implementing the code on the FPGA the results was as expected. Here it is clear that when `Clk` has a rising edge, `D` is saved to the output LED and when `Clk` is high or low, the output is equal to the last saved state of `D`.



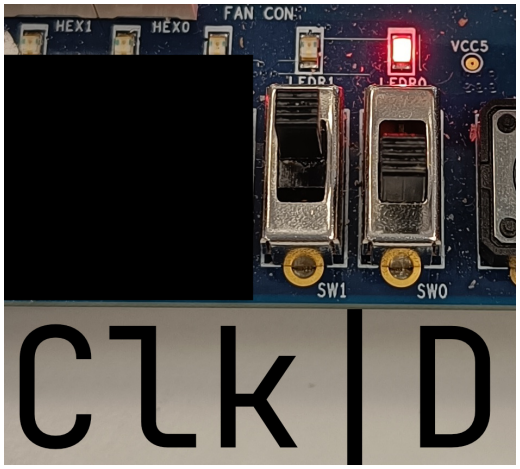
(a) Clk low, D low already stored



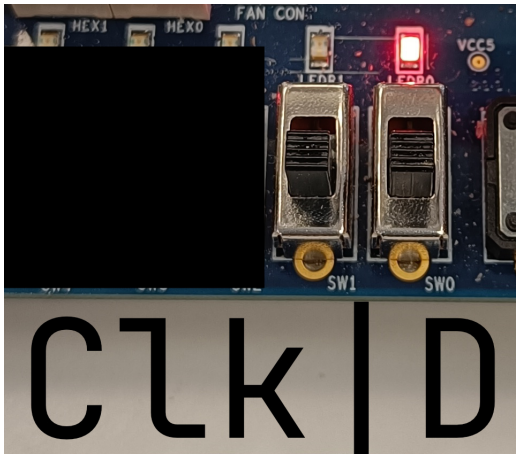
(b) Clk low, D high not registered



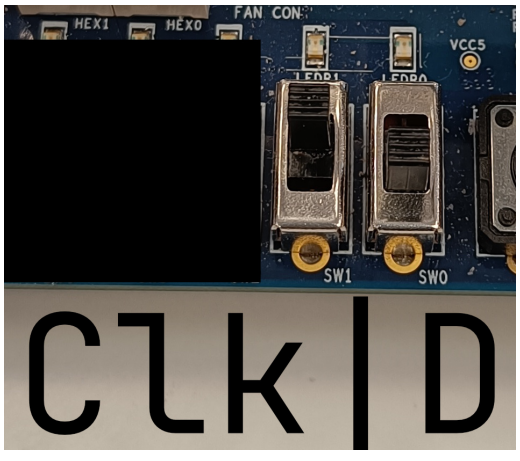
(c) Clk high, D high registered



(d) Clk high, D low not registered



(e) Clk low, D low not registered



(f) Clk high, D low registered

Figure 3.1: D-flip-flop on the FPGA using two D-latches works as expected

4. Part 4

The purpose of this task was to use behavioural code to implement a D-latch and a D-flip-flop. Then to see how the different data outputs are stored between a D-latch, a positive edge D-flip-flop and a negative edge D-flip-flop.

4.0. Code

Code 4.0: TLE of a circuit to compare different data storage

VHDL

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

entity L03P04 is
    port(SW   : in  std_logic_vector(1 downto 0);
         KEY  : in  std_logic_vector(3 downto 0);
         LEDR : out std_logic_vector(9 downto 0));
end L03P04;


architecture structural of L03P04 is

    component Dl
        port(Clk, D : in  std_logic;
             Q      : out std_logic);
    end component;

    component Dff
        port(Clk, D : in  std_logic;
             Q      : out std_logic);
    end component;

begin

    LEDR(9 downto 3) <= "0000000";

    D0: Dl port map(
        Clk => KEY(3),
        D   => SW(0),
        Q   => LEDR(0));

    D1: Dff port map(
        Clk => KEY(3),
        D   => SW(0),
        Q   => LEDR(1));

    D2: Dff port map(
        Clk => not(KEY(3)),
        D   => SW(0),
        Q   => LEDR(2));

end structural;
```

The D-latch code was already provided by the task, and instead of doing both a rising and a falling edge trigger separate for the D-flip-flops a common component was made and the `Clk` input inverted for the second instance.

Code 4.1: Code with equal function to the one included in the task

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

entity Dl is
    port(Clk, D : in  std_logic;
         Q      : out std_logic);
end Dl;

architecture behavioural of Dl is
begin
    process(Clk, D)
    begin
        if Clk = '1' then
            Q <= D;
        end if;
    end process;
end behavioural;
```

Code 4.2: Behavioural code for a D-flip-flop

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

entity Dff is
    port(Clk, D : in  std_logic;
         Q      : out std_logic);
end Dff;

architecture behavioural of Dff is
begin
    process(Clk, D)
    begin
        if rising_edge(Clk) then
            Q <= D;
        end if;
    end process;
end behavioural;
```

4.1. RTL

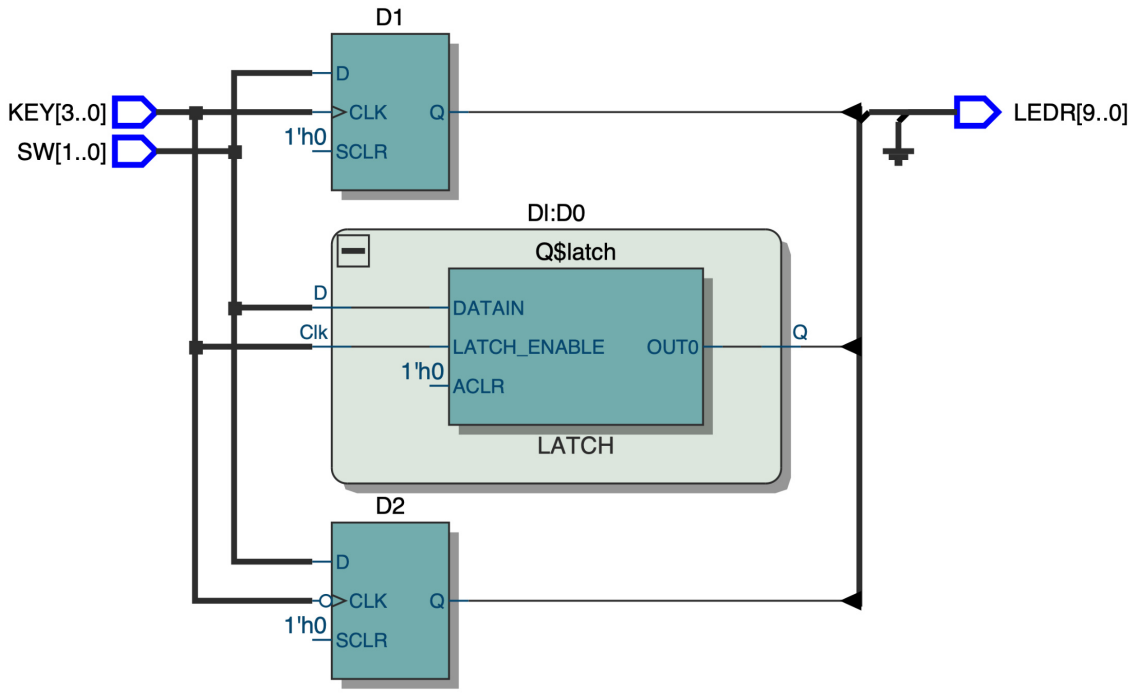
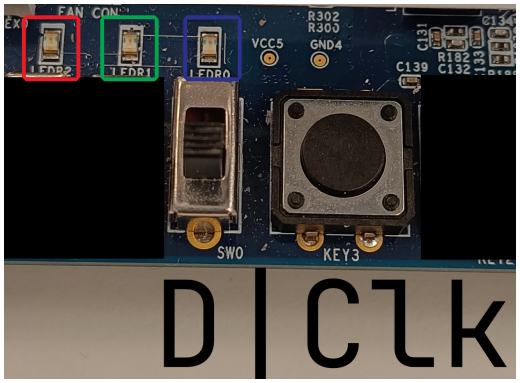


Figure 4.0: RTL of the TLE with all three instances

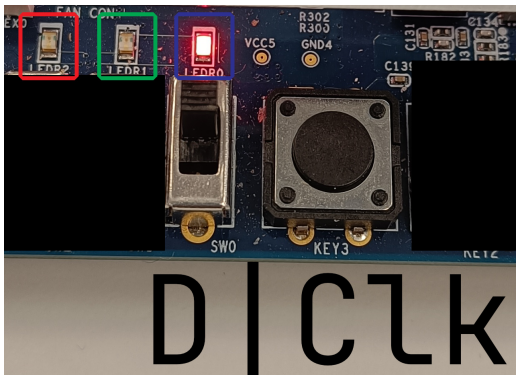
4.2. Results

Implementing the code on the FPGA the results was as expected. The output is shown as the following: In red, the negative edge D-flip-flop, in green the positive edge D-flip flop and in blue, the D-latch. It is also worth noting that the button `Clk` is connected to is pull-up, as it is normally high.

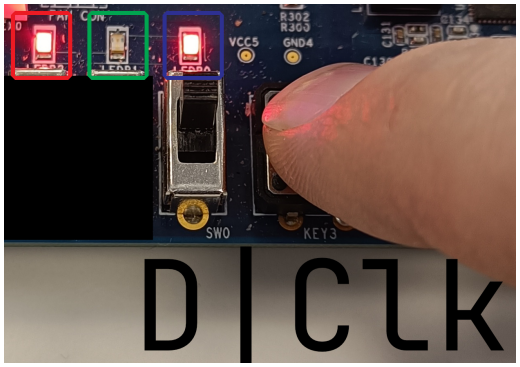
Observations shows that when `Clk` is high, unpressed, the, `D` is continuously controlling the blue output, and the others are unaffected. While depressing the button, giving a falling edge to `Clk`, the state of `D` is saved to the red output. When letting go of the button, giving a rising edge to `Clk`, the state of `D` is stored in the green output, as expected.



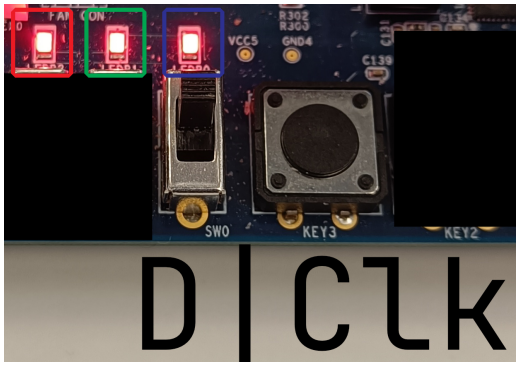
(a) D low, Clk high



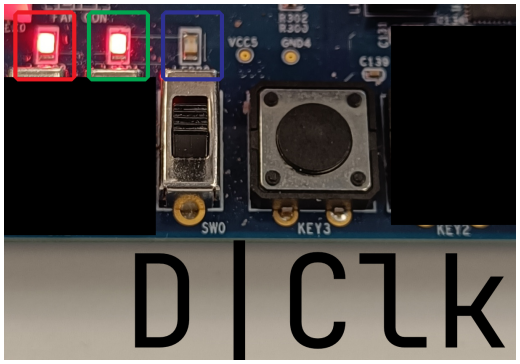
(b) D high, Clk high



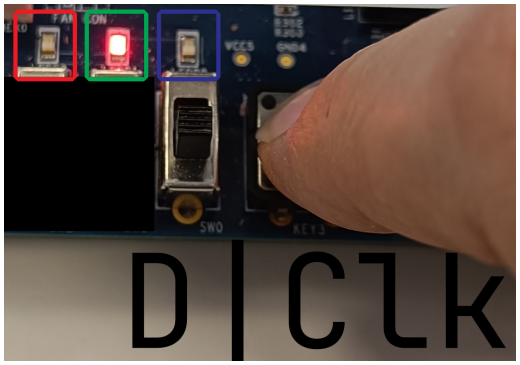
(c) D high, Clk low(falling)



(d) D high, Clk high(rising)



(e) D low, Clk high



(f) D low, Clk low(falling)

Figure 4.1: D-flip-flop and D-latches

5. Part 5

The purpose of this task was to use a register to store a value, and combine with knowledge from previous lab exercises to integrate an adder, and some 7-segment displays.

5.0. Solving

The starting point was making a decoder for the 7-segment displays, the task specified to operate in hexadecimal. That made it much simpler as instead of needing a BCD converter or similar, the 4 most significant bits could control the most significant digit and the 4 least significant bits could control the least significant digit, completely independently. Using figure 5.0 table 5.0 was made for use in the decoder.

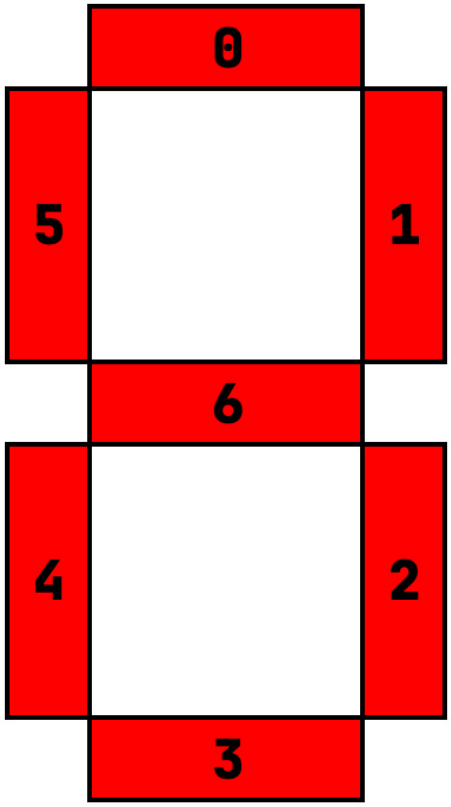


Figure 5.0: 7 segment FPGA bit order

Table 5.0: Output values for decoder based on input

NR	Num				HEX							String
	3	2	1	0	6	5	4	3	2	1	0	"6543210"
0	0	0	0	0	1	0	0	0	0	0	0	"1000000"
1	0	0	0	1	1	1	1	1	0	0	1	"1111001"
2	0	0	1	0	0	1	0	0	1	0	0	"0100100"
3	0	0	1	1	0	1	1	0	0	0	0	"0110000"
4	0	1	0	0	0	0	1	1	0	0	1	"0011001"
5	0	1	0	1	0	0	1	0	0	1	0	"0010010"
6	0	1	1	0	0	0	0	0	0	1	1	"0000011"
7	0	1	1	1	1	1	1	1	0	0	0	"1111000"
8	1	0	0	0	0	0	0	0	0	0	0	"0000000"
9	1	0	0	1	0	0	1	1	0	0	0	"0011000"
A	1	0	1	0	0	0	0	1	0	0	0	"0001000"
B	1	0	1	1	0	0	0	0	0	1	1	"0000011"
C	1	1	0	0	1	0	0	0	1	1	0	"1000110"
D	1	1	0	1	0	1	0	0	0	0	1	"0100001"
E	1	1	1	0	0	0	0	0	1	1	0	"0000110"
F	1	1	1	1	0	0	0	1	1	1	0	"0001110"

Then for readability and organization a display component was made, that took in a 8 bit number, split it in two and send it out to each respective 7 segment display using two instances of the decoder.

The first number, `A`, was not stored anywhere and was simply equal to whatever the current switch position was at the time. The second number, `B`, was a stored number in a simple 8 bit registry. Upon receiving a falling edge of the `Clk` button, `A` would overwrite whatever `B` was at the time so that this is now the new value of `B`. Also if the reset button is pressed, the value of `B` is set to 0. The final number was the sum of `A` and `B` and was simply calculated in the TLE, and if there were a overflow it would light up on the first LED.

5.1. Code

Code 5.0: TLE code for Part 5

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

entity L03P05 is
    port(SW         : in  std_logic_vector(7 downto 0);
         KEY        : in  std_logic_vector(1 downto 0);
         LEDR       : out std_logic_vector(9 downto 0);
         HEX0, HEX1 : out std_logic_vector(6 downto 0);
         HEX2, HEX3 : out std_logic_vector(6 downto 0);
         HEX4, HEX5 : out std_logic_vector(6 downto 0));
end L03P05;


architecture behavioural of L03P05 is

    component display
        port(Num  : in  std_logic_vector(7 downto 0);
             HEXa : out std_logic_vector(6 downto 0);
             HEXB : out std_logic_vector(6 downto 0));
    end component;

    component Reg
        port(Clk : in  std_logic;
             Rst : in  std_logic;
             D   : in  std_logic_vector(7 downto 0);
             Q   : out std_logic_vector(7 downto 0));
    end component;

    signal A, B : std_logic_vector(7 downto 0);
    signal S    : std_logic_vector(8 downto 0);

begin

    LEDR(9 downto 1) <= "000000000";

    process(SW, KEY, A, B, S)
    begin

        A <= SW;

        S <= ('0' & A) + ('0' & B);
        LEDR(0) <= S(8);

        end process;

    R0: Reg port map(
        Clk => KEY(1),
        Rst => KEY(0),
        D   => A,
        Q   => B);

    H0: display port map(
        Num  => B,
        HEXa => HEX0,
        HEXb => HEX1);

```

```
    H1: display port map(
        Num  => A,
        HEXa => HEX2,
        HEXb => HEX3);

    H2: display port map(
        Num  => S(7 downto 0),
        HEXa => HEX4,
        HEXb => HEX5);

end behavioural;
```

Code 5.1: The Display component used in the TLE

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

entity display is
    port(Num : in  std_logic_vector(7 downto 0);
         HEXa : out std_logic_vector(6 downto 0);
         HEXB : out std_logic_vector(6 downto 0));
end display;

architecture behavioural of display is

    component decoder7seg
        port(N   : in  std_logic_vector(3 downto 0);
             HEX : out std_logic_vector(6 downto 0));
    end component;

begin

    H0: decoder7seg port map(
        N => Num(3 downto 0),
        HEX => HEXa);

    H1: decoder7seg port map(
        N => Num(7 downto 4),
        HEX => HEXb);

end behavioural;
```

Code 5.2: The Decoder component used in the Display component

VHDL

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

entity decoder7seg is
    port(N   : in  std_logic_vector(3 downto 0);
         HEX : out std_logic_vector(6 downto 0));
end decoder7seg;

architecture behavioural of decoder7seg is

begin
    process(N)
    begin
        case N is
            when "0000" =>
                HEX <= "1000000";
            when "0001" =>
                HEX <= "1111001";
            when "0010" =>
                HEX <= "0100100";
            when "0011" =>
                HEX <= "0110000";
            when "0100" =>
                HEX <= "0011001";
            when "0101" =>
                HEX <= "0010010";
            when "0110" =>
                HEX <= "0000011";
            when "0111" =>
                HEX <= "1111000";
            when "1000" =>
                HEX <= "0000000";
            when "1001" =>
                HEX <= "0011000";
            when "1010" =>
                HEX <= "0001000";
            when "1011" =>
                HEX <= "0000011";
            when "1100" =>
                HEX <= "1000110";
            when "1101" =>
                HEX <= "0100001";
            when "1110" =>
                HEX <= "0000110";
            when "1111" =>
                HEX <= "0001110";
        end case;
    end process;
end behavioural;
```

Code 5.3: The Register component used in the TLE

VHDL

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

entity Reg is
    port(Clk : in  std_logic;
         Rst : in  std_logic;
         D   : in  std_logic_vector(7 downto 0);
         Q   : out std_logic_vector(7 downto 0));
end Reg;

architecture behavioural of Reg is
begin
    process(Clk, D)
    begin
        if falling_edge(Clk) then
            Q <= D;
        end if;
        if Rst = '0' then
            Q <= "00000000";
        end if;
    end process;
end behavioural;
```

5.2. RTL

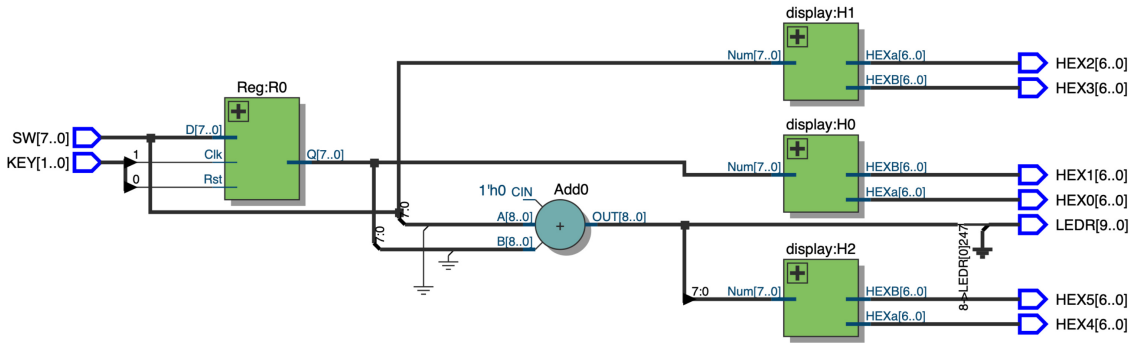


Figure 5.1: RTL of the TLE

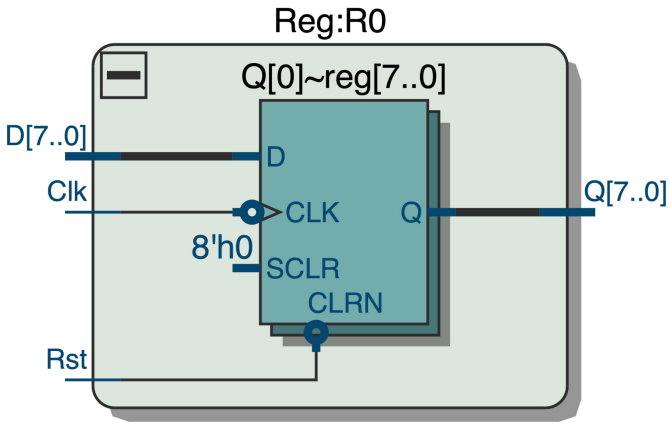


Figure 5.2: RTL of registry used in the TLE

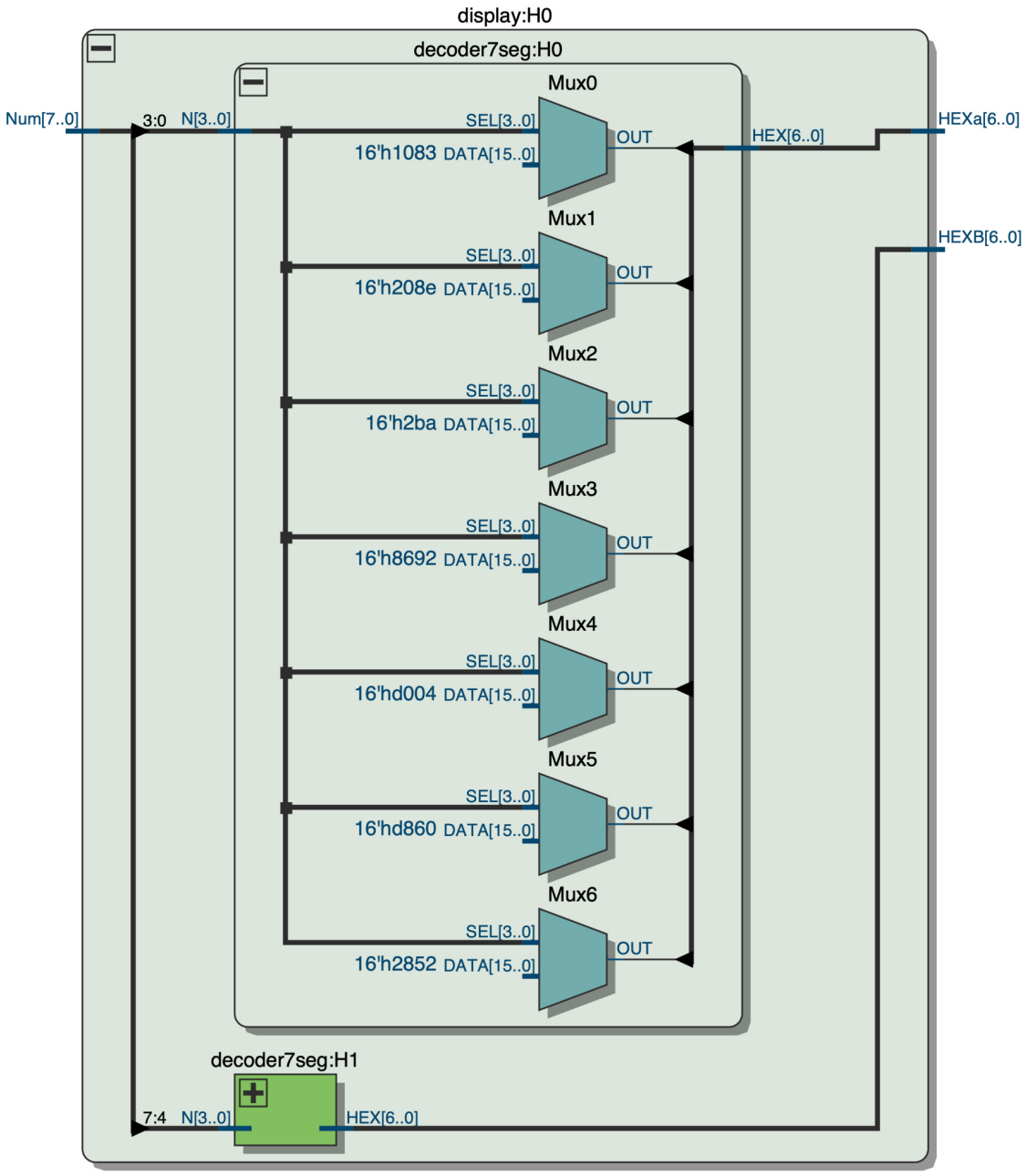
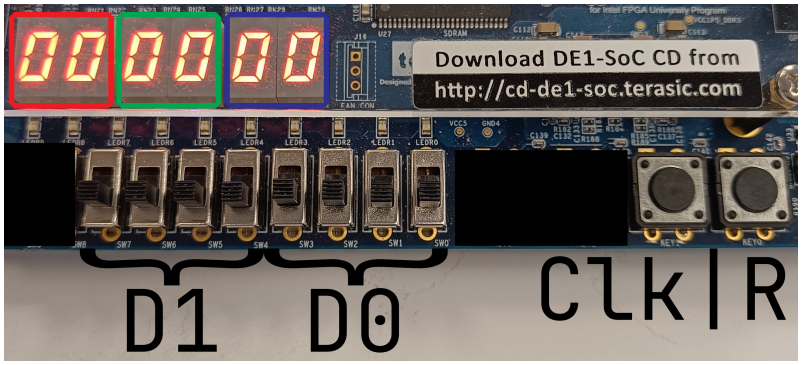


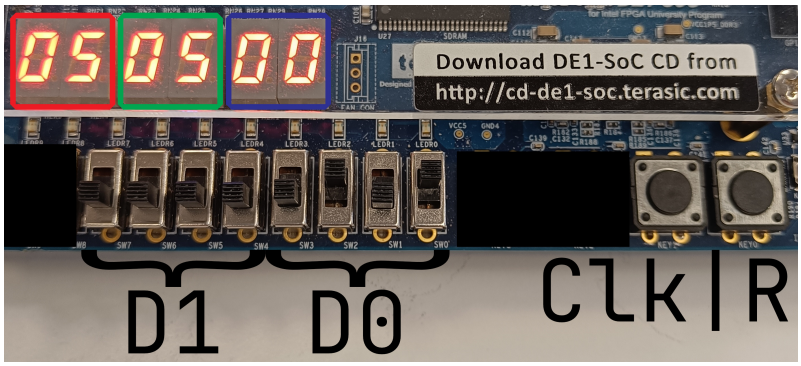
Figure 5.3: RTL of the Display component with two decoder instances

5.3. Results

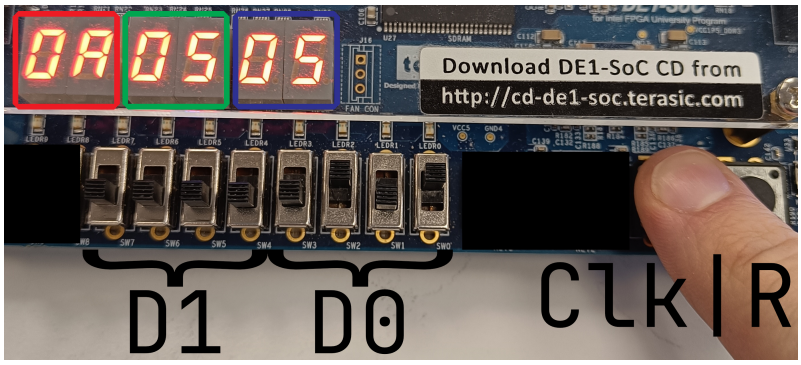
Implementing the code on the FPGA the results was as expected. The output is shown as the following: In red, the sum of A and B, in green the current switch value, A, is show and in blue, the stored number B.



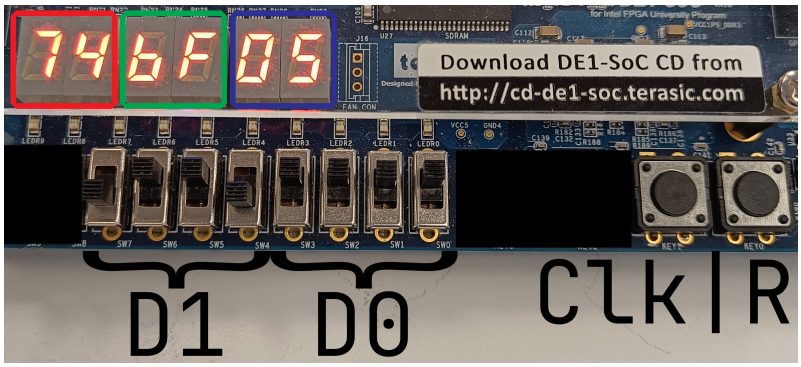
(a) 00 = 00 + 00



(b) 05 = 05 + 00

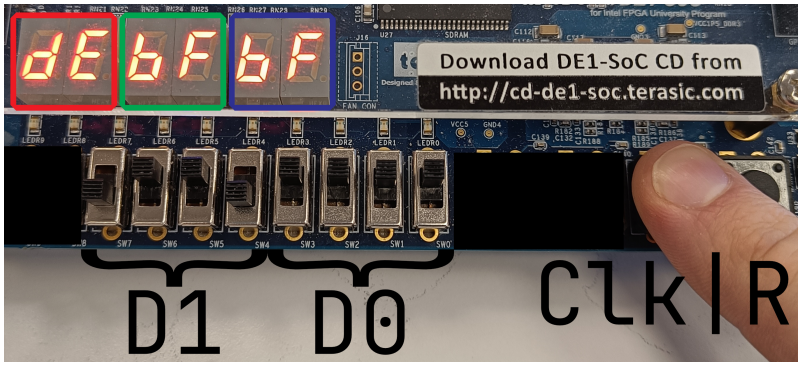


(c) 0A = 05 + 05, Clk falling

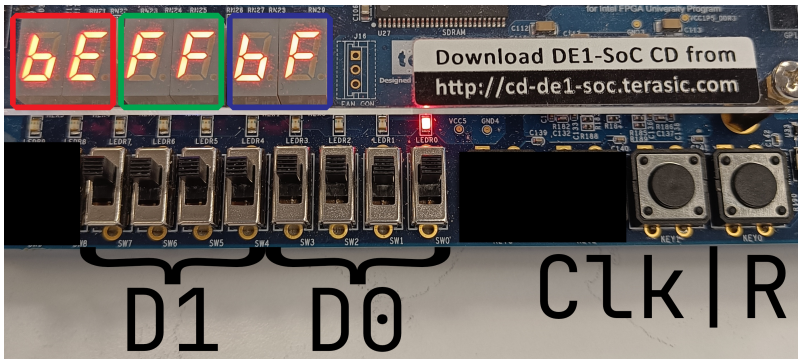


(d) 74 = BF + 05

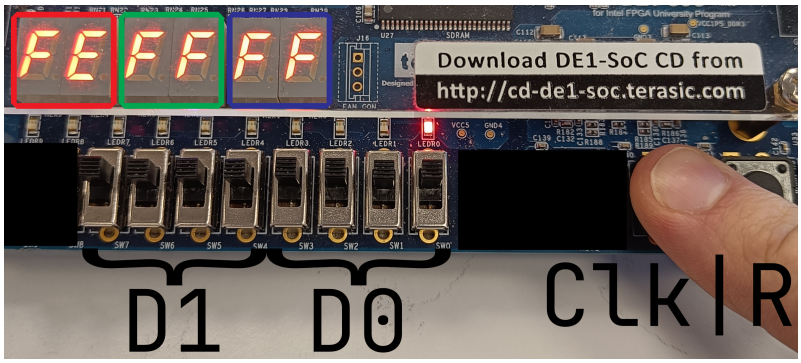
Figure 5.4: Results of part 5



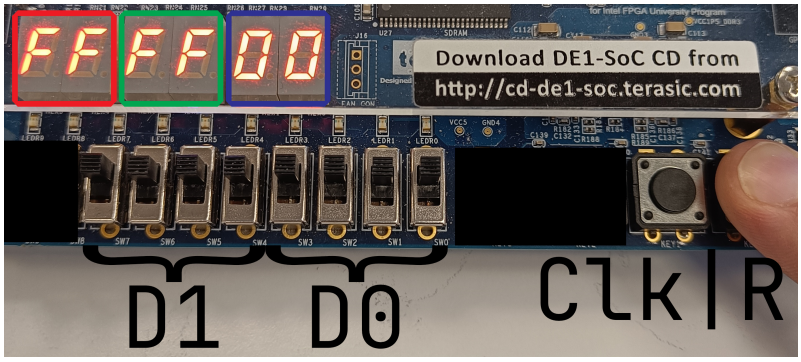
(e) DE = BF + BF, Clk falling



(f) BE = FF + BF (overflow)



(g) FE = FF + FF (overflow), Clk falling



(h) FF = FF + 00, Reset falling

Figure 5.4: Results of part 5 (continued)