Programable Logic Circuts
Lab 04

Sølve Kjelseth

September 5, 2025

0. Introduction

This is the fourth report in this course, detailing the completion of the fourth lab exercise. With the purpose is to investigate counters.

Note: As always, the VHDL and LaTeX code is open source, see my GitHub 

0.0. Table of Contents

0.1. List of Figures

1. Part 1

This Part is about making a structural coded counter using T-flip-flop instances and displaying the counted value on some 7-segment displays.

1.0. Solving

Solving this task was done by first making a T-flip-flop, as the task did not specify how this should be made, the choice of behavioural code was made. Then the counter was made as described in the task with structural code using eight instances of the T-flip-flop. The display component, code 1.3 and its decoder component, code 1.4 was copied from previous lab tasks. Then finally a TLE was made that simply connected the switches and button to the counter and connected the counter to the display.

1.1. Code

Code 1.0: TLE for the structural counter

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

entity L04P01 is
    port(SW   : in  std_logic_vector(1 downto 0);
         KEY  : in  std_logic_vector(3 downto 0);
         HEX0 : out std_logic_vector(6 downto 0);
         HEX1 : out std_logic_vector(6 downto 0));
end L04P01;


architecture structural of L04P01 is

    component counter
        port(E   : in  std_logic;
             Clk : in  std_logic;
             Rst : in  std_logic;
             Q_a : out std_logic_vector(7 downto 0);
             Q_b : out std_logic_vector(7 downto 0));
    end component;

    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;

    signal N_a : std_logic_vector(7 downto 0);
    signal N_b : std_logic_vector(7 downto 0);

begin

    C0: counter port map(
        E   => SW(1),
        Clk => KEY(3),
        Rst => SW(0),
        Q_a => N_a,
        Q_b => N_b);

    D0: display port map(
        Num  => N_a,
        HEXa => HEX0,
        HEXb => HEX1);

end structural;
```

Code 1.1: Counter component used in the TLE

VHDL

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

entity counter is
    port(E   : in  std_logic;
         Clk : in  std_logic;
         Rst : in  std_logic;
         Q_a : out std_logic_vector(7 downto 0);
         Q_b : out std_logic_vector(7 downto 0));
end counter;


architecture structural of counter is

    component Tflipflop
        port(T   : in  std_logic;
             Clk : in  std_logic;
             Rst : in  std_logic;
             Q_a : out std_logic;
             Q_b : out std_logic);
    end component;

    signal T : std_logic_vector(6 downto 0);
    Signal Q : std_logic_vector(7 downto 0);

begin

    T(0) <= Q(0) and E;
    T(1) <= Q(1) and T(0);
    T(2) <= Q(2) and T(1);
    T(3) <= Q(3) and T(2);
    T(4) <= Q(4) and T(3);
    T(5) <= Q(5) and T(4);
    T(6) <= Q(6) and T(5);

    Q_a <= Q;

    T0: Tflipflop port map(
        T   => E,
        Clk => Clk,
        Rst => Rst,
        Q_a => Q(0),
        Q_b => Q_b(0));

    T1: Tflipflop port map(
        T   => T(0),
        Clk => Clk,
        Rst => Rst,
        Q_a => Q(1),
        Q_b => Q_b(1));

    T2: Tflipflop port map(
        T   => T(1),
        Clk => Clk,
        Rst => Rst,
        Q_a => Q(2),
        Q_b => Q_b(2));

    T3: Tflipflop port map(
```

```
60          T   => T(2),
61          Clk => Clk,
62          Rst => Rst,
63          Q_a => Q(3),
64          Q_b => Q_b(3));
65
66      T4: Tflipflop port map(
67          T   => T(3),
68          Clk => Clk,
69          Rst => Rst,
70          Q_a => Q(4),
71          Q_b => Q_b(4));
72
73      T5: Tflipflop port map(
74          T   => T(4),
75          Clk => Clk,
76          Rst => Rst,
77          Q_a => Q(5),
78          Q_b => Q_b(5));
79
80      T6: Tflipflop port map(
81          T   => T(5),
82          Clk => Clk,
83          Rst => Rst,
84          Q_a => Q(6),
85          Q_b => Q_b(6));
86
87      T7: Tflipflop port map(
88          T   => T(6),
89          Clk => Clk,
90          Rst => Rst,
91          Q_a => Q(7),
92          Q_b => Q_b(7));
93
94  end structural;
```

Code 1.2: T-flip-flop component used in counter

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

entity Tflipflop is
    port(T   : in  std_logic;
         Clk : in  std_logic;
         Rst : in  std_logic;
         Q_a : out std_logic;
         Q_b : out std_logic);
end Tflipflop;


architecture behavioural of Tflipflop is
    signal Q : std_logic;
begin
    process(Clk, T, Rst)
    begin
        if (rising_edge(Clk) and T = '1') then
            Q <= not Q;
        end if;
        if (rising_edge(Clk) and Rst = '0') then
            Q <= '0';
        end if;
        Q_a <= Q;
        Q_b <= not Q;
    end process;
end behavioural;
```

Code 1.3: 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 1.4: Decoder component used in the display

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;
```

1.2. RTL

The expanded RTL of the decoder components is not included as it is not relevant for the purpose of this Lab. Check out the GitHub at `PLK_lab/Lab03/Temp/Part5_RTL_Display.pdf` for the RTL, and for more general information regarding this component, check out `PLK_lab/Lab03/main.tex` and read part 5 in the rendered document.

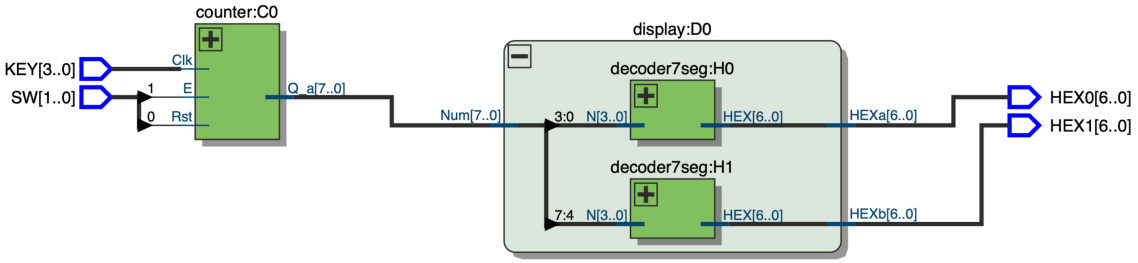


Figure 1.0: RTL of the TLE

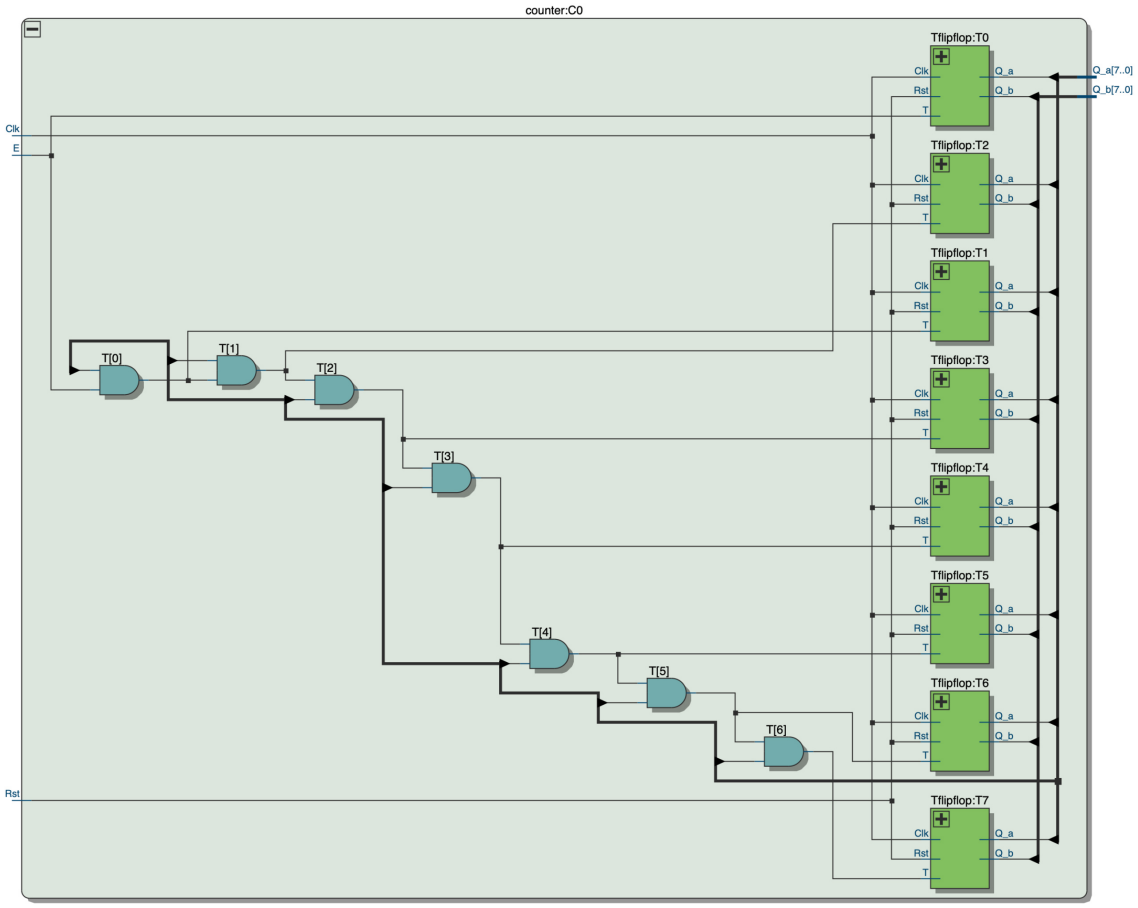


Figure 1.1: RTL for the counter used in the TLE

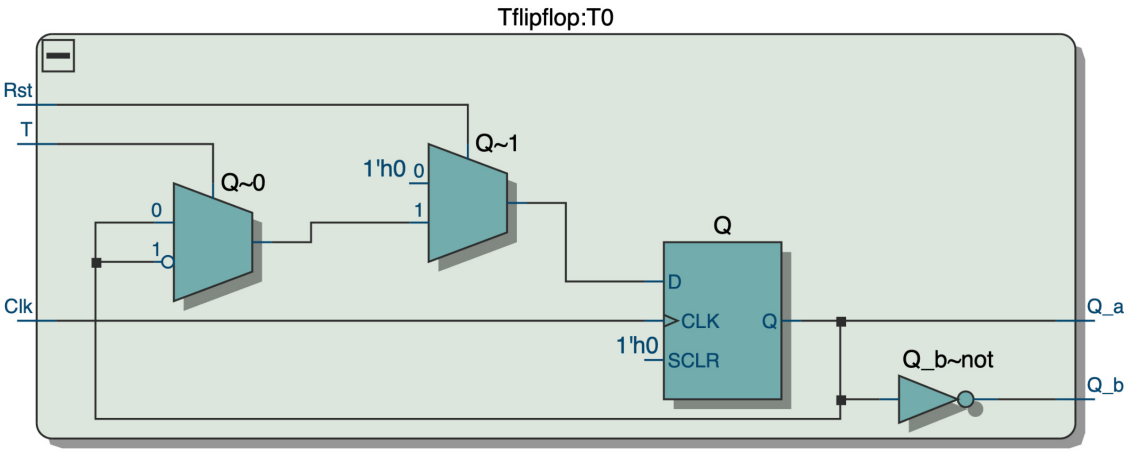


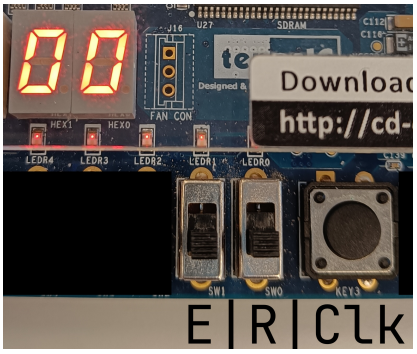
Figure 1.2: RTL for the T-flip-flop used in the counter

1.3. Results

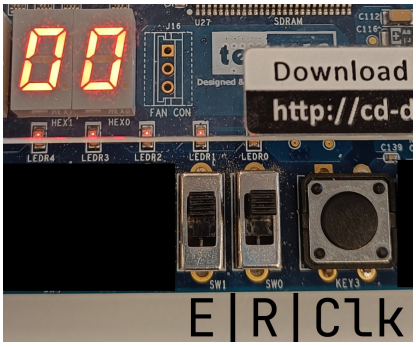
As read from figure 1.3, under ”Logic utilization”, this circuit uses a total of 13 logic elements to realize this configuration.

Flow Status	Successful - Tue Mar 18 12:34:08 2025
Quartus Prime Version	23.1std.1 Build 993 05/14/2024 SC Lite Edition
Revision Name	L04P01
Top-level Entity Name	L04P01
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	13 / 32,070 (< 1 %)
Total registers	8
Total pins	20 / 457 (4 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

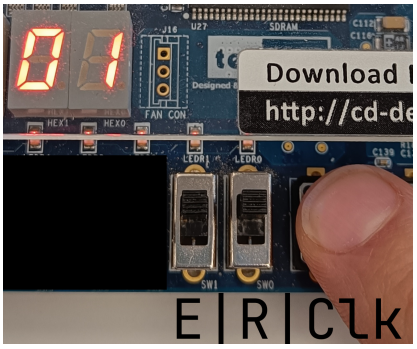
Figure 1.3: Compilation results from structural counter



(a) Disabled, reset low, no change



(b) Enabled, reset high, no change

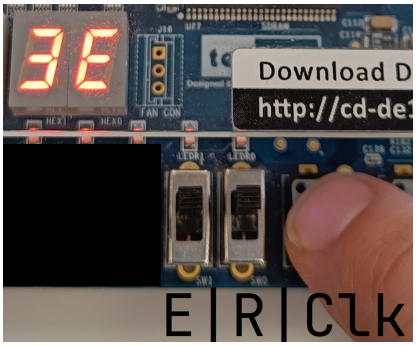


(c) Clock cycle 1, counts up

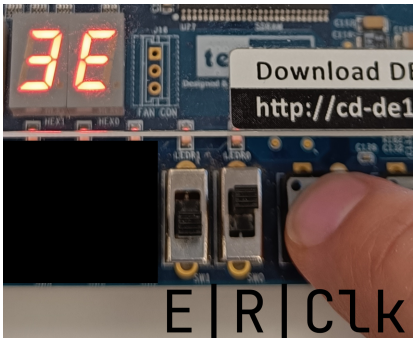
(d) Clock cycle 2, counts up

(e) Reset low, no change

(f) Clock cycle 3, resets to 0



(g) Enabled, clock cycle n, counts up



(h) Disabled, clock cycle n+1, no change

Figure 1.4: Some functional tests, works as expected

2. Part 2

This is part is about making a similar circuit as part 1 but instead making it with a behavioural code and in 16-bit.

2.0. Solving

The behavioural code to make a simple resettable counter was so small that it was made inside the TLE. The statement supplied in the task combined with a simple if statement was enough to make the counter, then a similar pair for clearing the value was also made. instead of splitting the 16-bit number in two, the display component was slightly modified to accept a 16 bit number and use four instances of the decoder instead.

2.1. Code

Code 2.0: TLE for the behavioural counter

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

entity L04P02 is
    port(SW   : in  std_logic_vector(1 downto 0);
         KEY  : in  std_logic_vector(3 downto 0);
         HEX0 : out std_logic_vector(6 downto 0);
         HEX1 : out std_logic_vector(6 downto 0);
         HEX2 : out std_logic_vector(6 downto 0);
         HEX3 : out std_logic_vector(6 downto 0));
end L04P02;


architecture behavioural of L04P02 is

    component display
        port(Num  : in  std_logic_vector(15 downto 0);
             HEXa : out std_logic_vector(6  downto 0);
             HEXb : out std_logic_vector(6  downto 0);
             HEXc : out std_logic_vector(6  downto 0);
             HEXd : out std_logic_vector(6  downto 0));
    end component;

    signal Q : std_logic_vector(15 downto 0) := (others => '0');

begin
    process(SW, KEY, Q)
    begin
        if (rising_edge(KEY(3)) and SW(1) = '1') then
            Q <= Q + 1;
        end if;
        if (rising_edge(KEY(3)) and SW(0) = '0') then
            Q <= (others => '0');
        end if;
    end process;

    D0: display port map(
        Num  => Q,
        HEXa => HEX0,
        HEXb => HEX1,
        HEXc => HEX2,
        HEXd => HEX3);
end behavioural;
```

Code 2.1: Modified display component used in the TLE

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

entity display is
    port(Num  : in  std_logic_vector(15 downto 0);
         HEXa : out std_logic_vector(6  downto 0);
         HEXb : out std_logic_vector(6  downto 0);
         HEXc : out std_logic_vector(6  downto 0);
         HEXd : 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);

    H2: decoder7seg port map(
        N => Num(11 downto 8),
        HEX => HEXc);

    H3: decoder7seg port map(
        N => Num(15 downto 12),
        HEX => HEXd);

end behavioural;
```

2.2. RTL

Figure 2.0 shows that this is implemented using two sets of comparators in series, the first for checking the enable and then count up using an adder each clock cycle. The second for checking the reset and replace the whole number with 0 if reset is low.

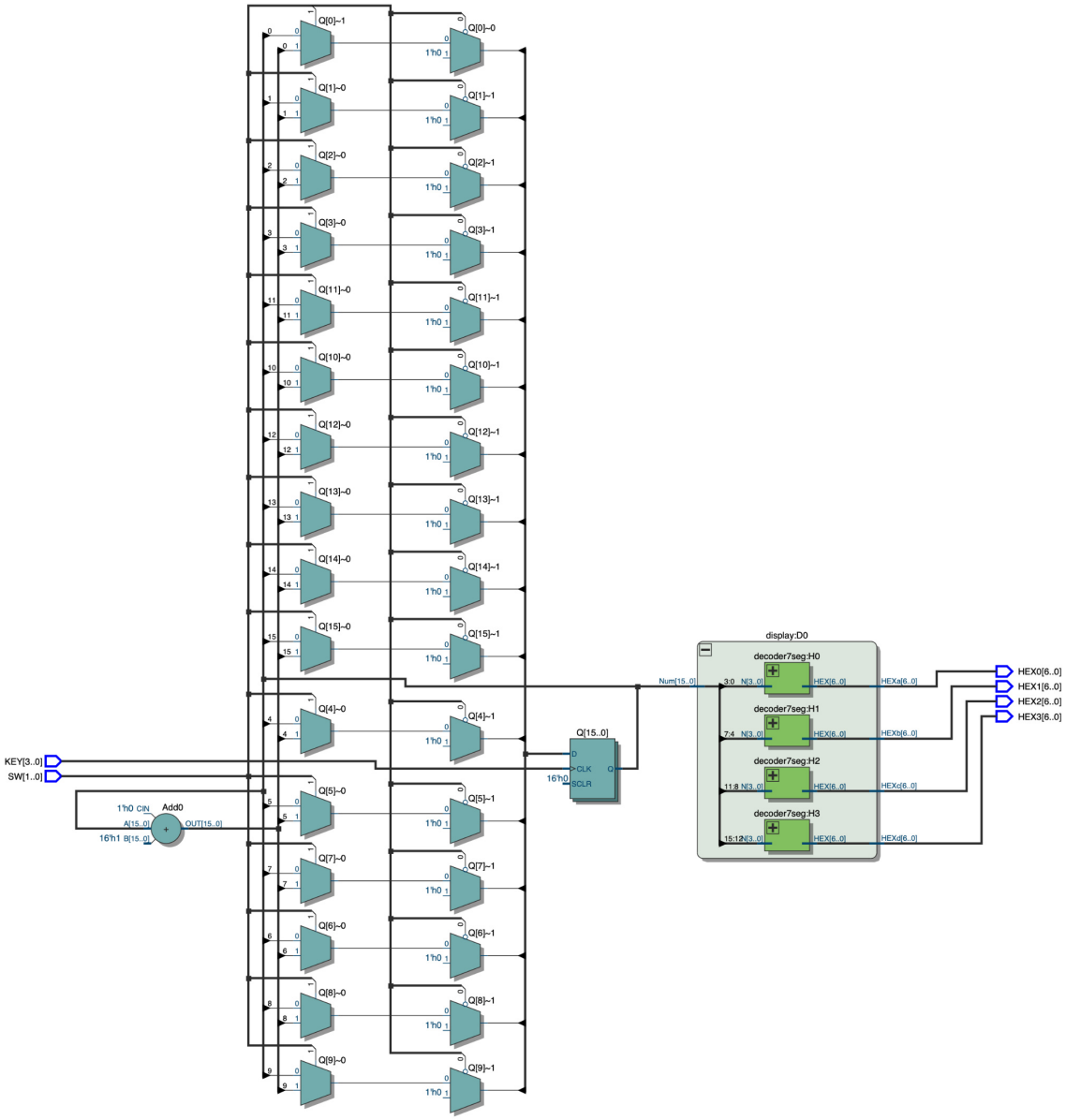


Figure 2.0: RTL of the TLE

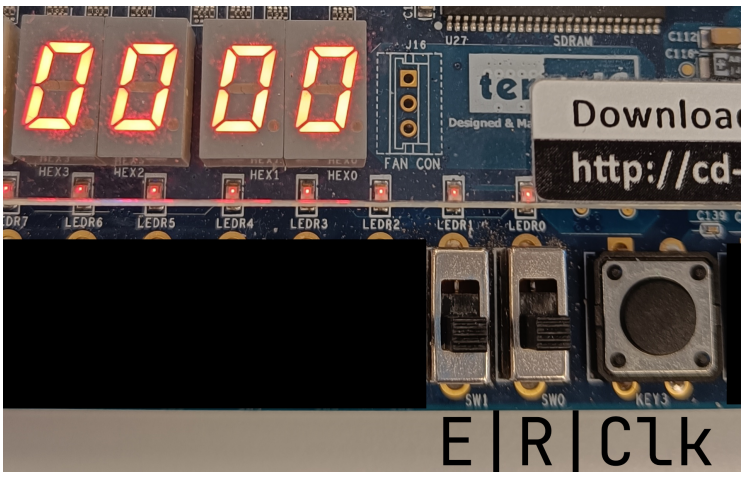
2.3. Results

As read from figure 2.1, under ”Logic utilization”, this circuit uses a total of 23 logic elements to realize this configuration. Not much different from the structural code in part 1, especially since this is 16-bit and not 8-bit and twice the amount of 7-segment displays. An argument of this being more efficient/compact could be made, but there would be a better comparison if equal circuits were compared.

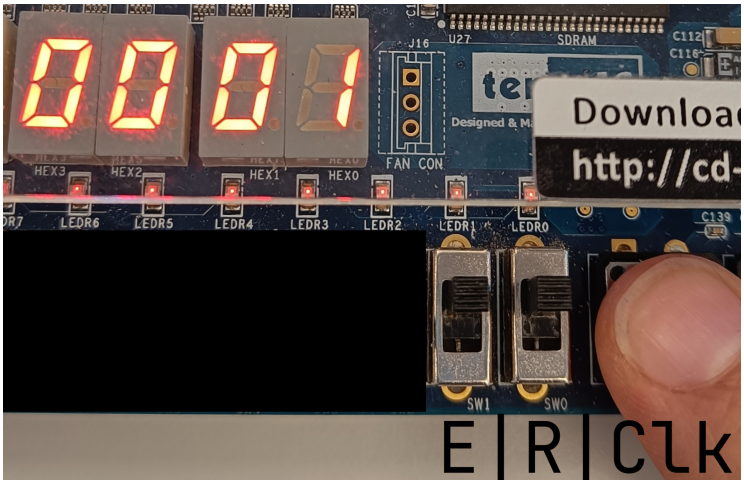
Flow Status	Successful - Tue Mar 18 12:59:18 2025
Quartus Prime Version	23.1std.1 Build 993 05/14/2024 SC Lite Edition
Revision Name	L04P02
Top-level Entity Name	L04P02
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	23 / 32,070 (< 1 %)
Total registers	16
Total pins	34 / 457 (7 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 2.1: Compilation results from behavioural counter

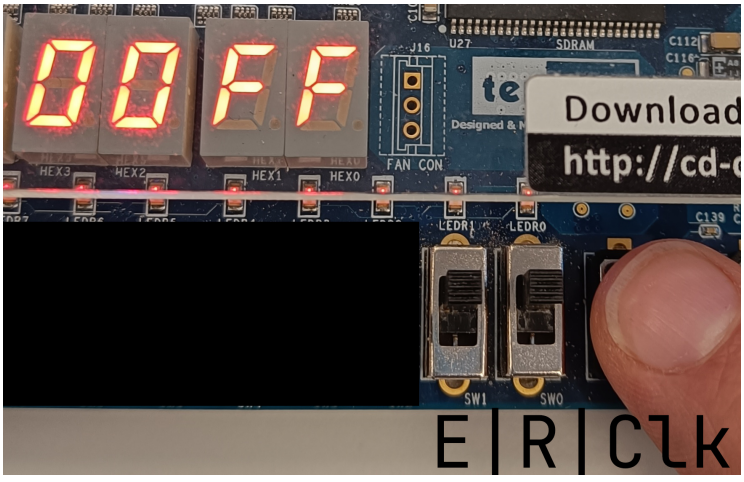
The pictures from the testing in figure 2.2 of this circuit are unfortunately very limited, but it was tested thoroughly off camera to ensure that it worked the way as the part 1 circuit, with the only change being compatibility for larger numbers.



(a) Disabled, reset low, no change



(b) Enabled, reset high, clock cycle 1, counts up



(c) clock cycle n, counts up

Figure 2.2: Some limited testing results

3. Part 3

The purpose of this task was to learn how to use a counter to make a ”slower clock” from fast clock.

3.0. Solving

The first thing to do for solving this kind of problem is some math. we know the following:

$$f_{clock} = 50\,\text{MHz}$$

$$f_{desired} = 1\,\text{Hz}$$

$$Ratio = \frac{50\,\text{MHz}}{1\,\text{Hz}}$$

$$Ratio = 5,0 \cdot 10^7 \tag{1}$$

$$2^{25} < 5,0 \cdot 10^7 < 2^{26} \tag{2}$$

Instructions for this task was that it was good enough to make an approximate 1 second clock, as the purpose was to be able to see each clock cycle happen, in for example a counter. Using equation 2 shows that making a 25 or 26 bit counter to trigger the slower counter each time it resets would work for this approximation as these are the closest to the ratio found in equation 1. I don't think that was acuate enough so I choose to make a more precise 1 second clock. This means using a 26 bit counter and when it reaches the exact ratio value, the counter resets and counts up the slow clock.

$$5,0 \cdot 10^7_{10} = 10\ 1111\ 1010\ 1111\ 0000\ 1000\ 0000_2 \tag{3}$$

The binary value of the ratio was found in equation 3 and used to reset the counter. In addition the slow clock had a similar setup, instead of being a 4-bit counter that resets at decimal 15 it should only go from 0 to 9, therefore it resets once it hits 10. And there was also included both enable and reset switches to control wether the counter was active or not and if it should clear all counters. As this task only used a single 7-segment display, the decoder component was used directly.

3.1. Code

Code 3.0: TLE of the clock divider

VHDL

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

entity L04P03 is
    port(CLOCK_50 : in  std_logic;
         SW       : in  std_logic_vector(1 downto 0);
         HEX0     : out std_logic_vector(6 downto 0));
end L04P03;


architecture behavioural of L04P03 is

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

    signal Q_fast : std_logic_vector(25 downto 0) := (others => '0');
    signal Q_slow : std_logic_vector(3 downto 0) := (others => '0');

begin
    process(SW, Q_fast, Q_slow, CLOCK_50)
    begin
        if (rising_edge(CLOCK_50) and SW(1) = '1') then
            if Q_fast = "10111110101111000010000000" then
                Q_fast <= (others => '0');
                Q_slow <= Q_slow + 1;
            else
            Q_fast <= Q_fast + 1;
            end if;
        end if;
        if (rising_edge(CLOCK_50) and Q_slow = "1010") then
            Q_slow <= (others => '0');
        end if;
        if (rising_edge(CLOCK_50) and SW(0) = '0') then
            Q_fast <= (others => '0');
            Q_slow <= (others => '0');
        end if;

    end process;

    D0: decoder7seg port map(
        N  => Q_slow,
        HEX => HEX0);
end behavioural;
```

3.2. RTL

figure 3.0 show that similar to part 2, it is a bunch of comparators,with some adders. It is kind of possible to see the 26 bit counter with comparators and the 4 bit counter with comparators and how they are connected. A pdf with a scalable graphic is in the GitHub at `[PLK_lab/Lab04/Extra/Part3_RTL_TLE.pdf]` if studying the RTL closer is of interest.

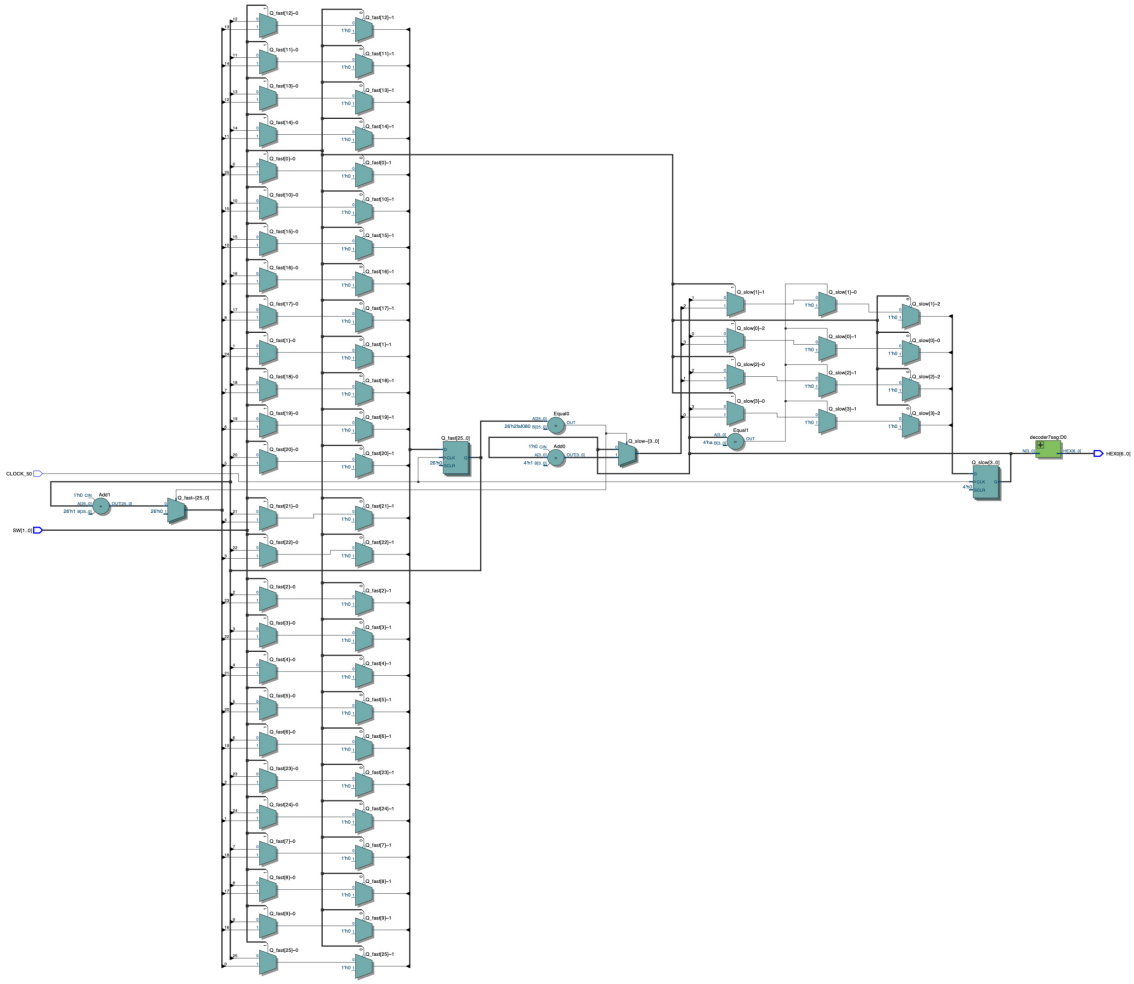


Figure 3.0: RTL of the TLE

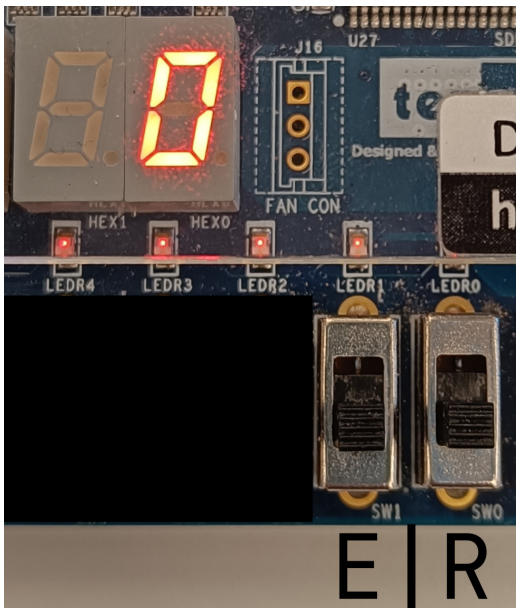
3.3. Results

As read from figure 3.1, under ”Logic utilization”, this circuit uses a total of 28 logic elements to realize this configuration. That is not a lot as it is dealing with a 26 bit number and a 4 bit number, both with multiple comparators.

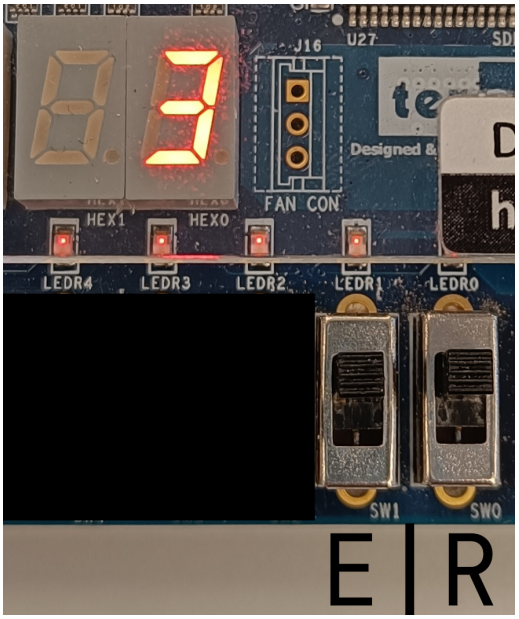
Flow Status	Successful - Tue Mar 18 14:01:13 2025
Quartus Prime Version	23.1std.1 Build 993 05/14/2024 SC Lite Edition
Revision Name	L04P03
Top-level Entity Name	L04P03
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	28 / 32,070 (< 1 %)
Total registers	30
Total pins	10 / 457 (2 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 3.1: Compilation results from clock divider

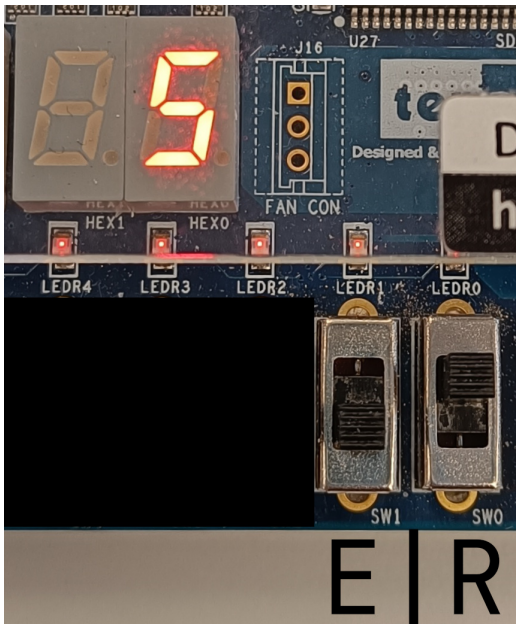
A video of the counter counting would be the only true way to show how it functions. That can be found, also in the mentioned GitHub at `[PLK_lab/Lab04/Extra/Part3.mp4]` if that is of interest. The pictures in figure 3.2 shows function but is limited.



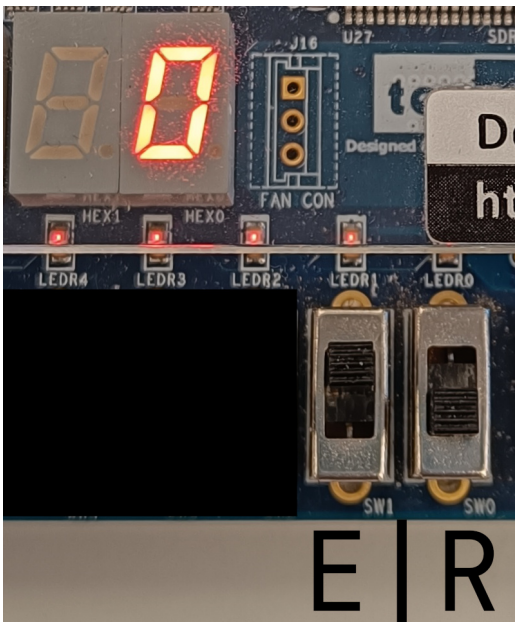
(a) Disabled, reset low, value still



(b) Enabled, reset high, counting



(c) continuing to count



(d) Reset low, value set still at 0

Figure 3.2: Testing results for clock divider