Assignment 1 – FPGA
Digital clock

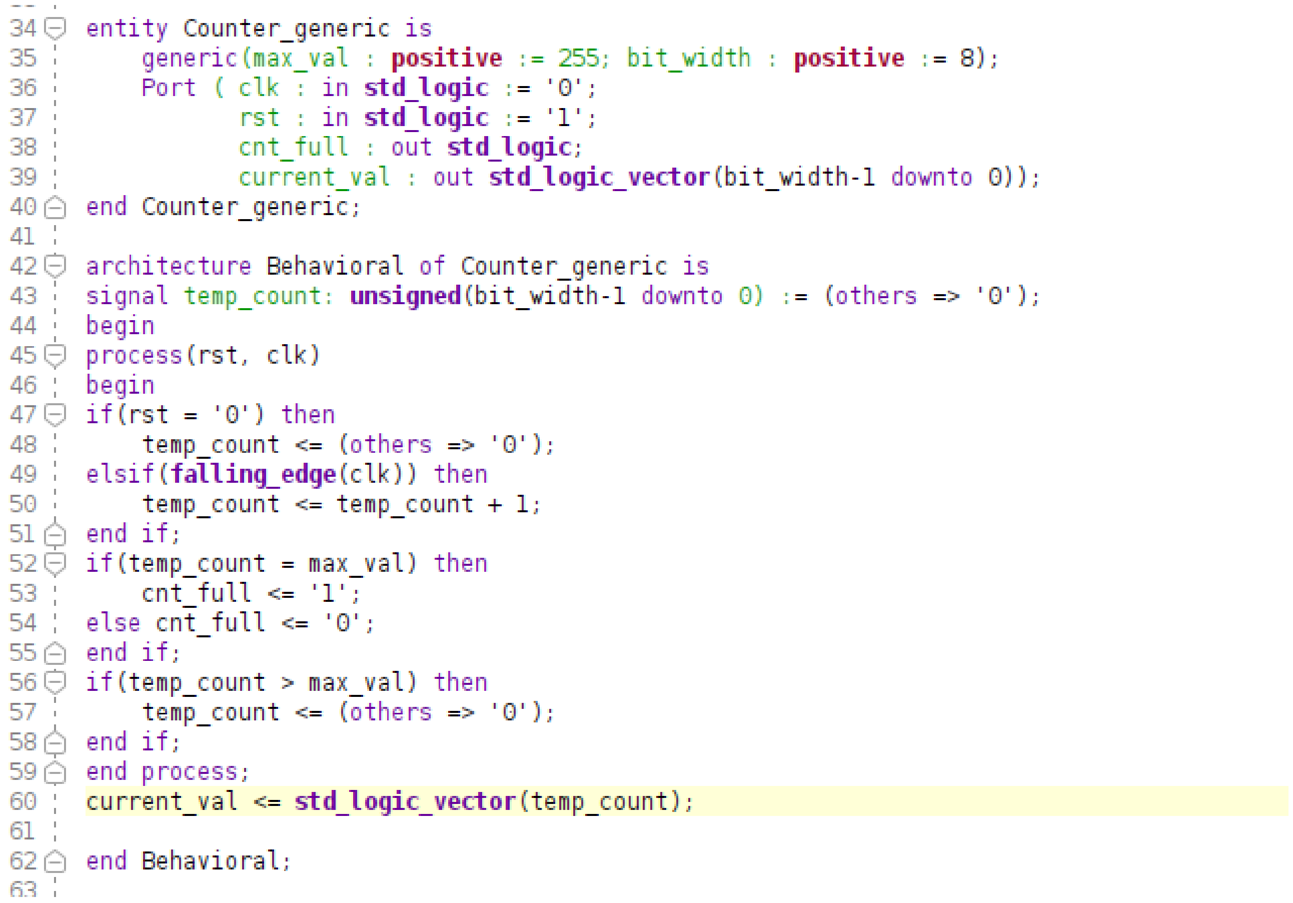
Used Components:

- Counter_generic
- Digit-splitter
- BCD_to_7seg

Explanation of components:

- Counter_generic:

The counter is implemented as a generic counter, where the bit-width and the max value of the counter is defined by the user.
The counter counts up on the falling edge of the clk input pin, and is reset to 0, if the input on the rst pin is low.
The current value of the counter is put out on the current_val output bus. When the counter has counted up to the defined max value, the cnt_full output is set to high, and when it recieves one more clock pulse, it will reset it's value to 0, and the cnt_full is set low again. In this application, this behavior can be used to count up the next counter, for example when the seconds counter has counted to 59, and receives the next clock signal it will reset itself and the falling edge of the cnt_full pin can count up the minutes counter.
The code of the counter can be seen here:



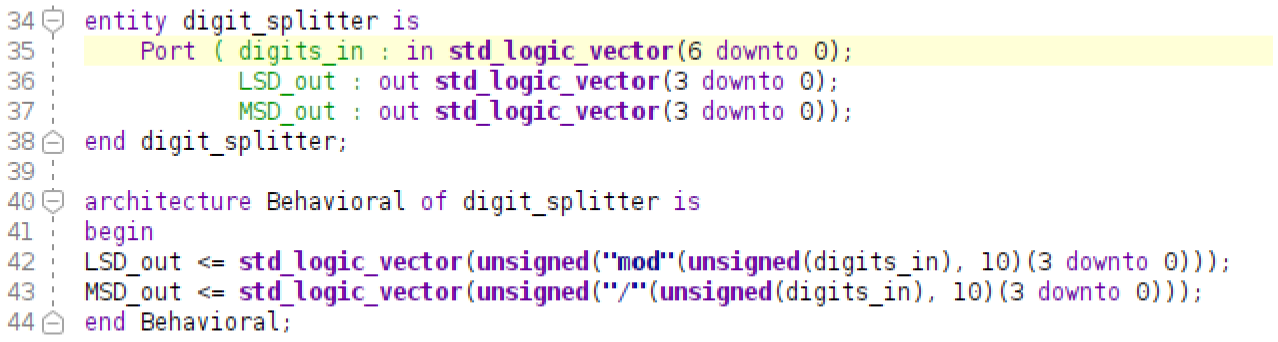
```
entity Counter_generic is
    generic(max_val : positive := 255; bit_width : positive := 8);
    Port ( clk : in std_logic := '0';
           rst : in std_logic := '1';
           cnt_full : out std_logic;
           current_val : out std_logic_vector(bit_width-1 downto 0));
end Counter_generic;

architecture Behavioral of Counter_generic is
signal temp_count: unsigned(bit_width-1 downto 0) := (others => '0');
begin
process(rst, clk)
begin
if(rst = '0') then
    temp_count <= (others => '0');
elsif(falling_edge(clk)) then
    temp_count <= temp_count + 1;
end if;
if(temp_count = max_val) then
    cnt_full <= '1';
else cnt_full <= '0';
end if;
if(temp_count > max_val) then
    temp_count <= (others => '0');
end if;
end process;
current_val <= std_logic_vector(temp_count);

end Behavioral;
```

It can be noted that the current_val output bus is updated outside of the process, so this output will always be the same as the internal temp_count variable.

- Digit_splitter:

The digit splitter can take a two digit number (0-99) and split it into its individual digits. This is done using the mod and division operations on the input number.

The code can be seen here:

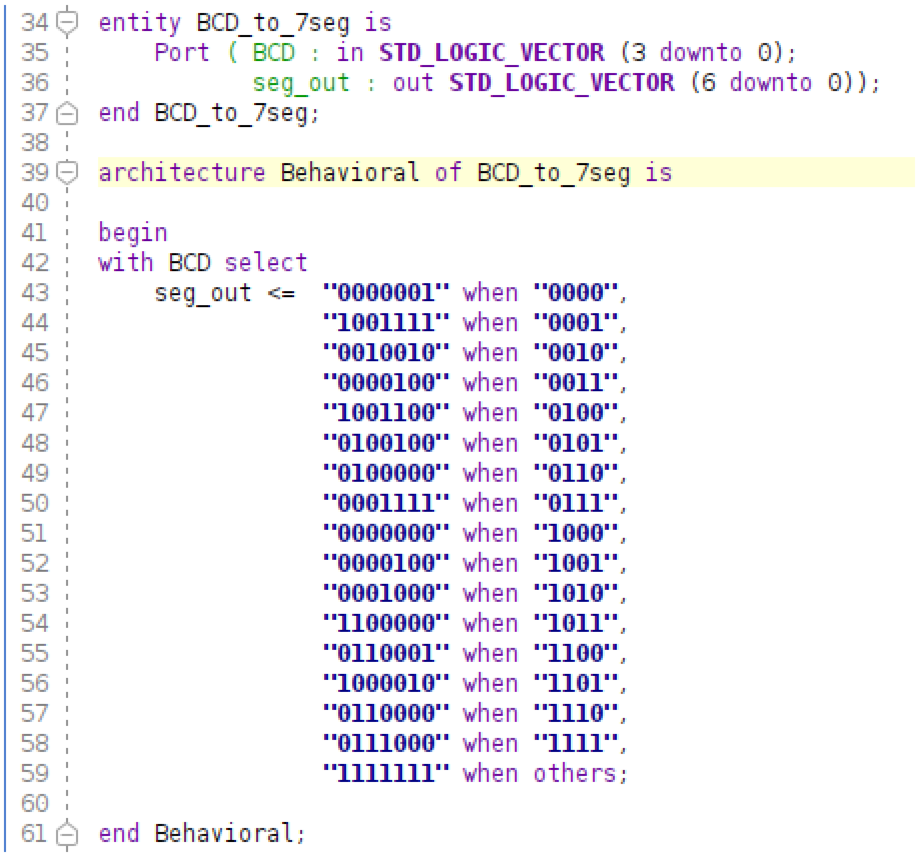


```vhdl
entity digit_splitter is
    Port ( digits_in : in std_logic_vector(6 downto 0);
           LSD_out : out std_logic_vector(3 downto 0);
           MSD_out : out std_logic_vector(3 downto 0));
end digit_splitter;

architecture Behavioral of digit_splitter is
begin
LSD_out <= std_logic_vector(unsigned("mod"(unsigned(digits_in), 10)(3 downto 0)));
MSD_out <= std_logic_vector(unsigned("/"(unsigned(digits_in), 10)(3 downto 0)));
end Behavioral;
```

- BCD_to_7seg:

This block takes a single digit number (like the one from the digit splitter) and converts it to the format required by seven-segment displays. This is done by hardcoding the output values for each input value from 0 to 9.

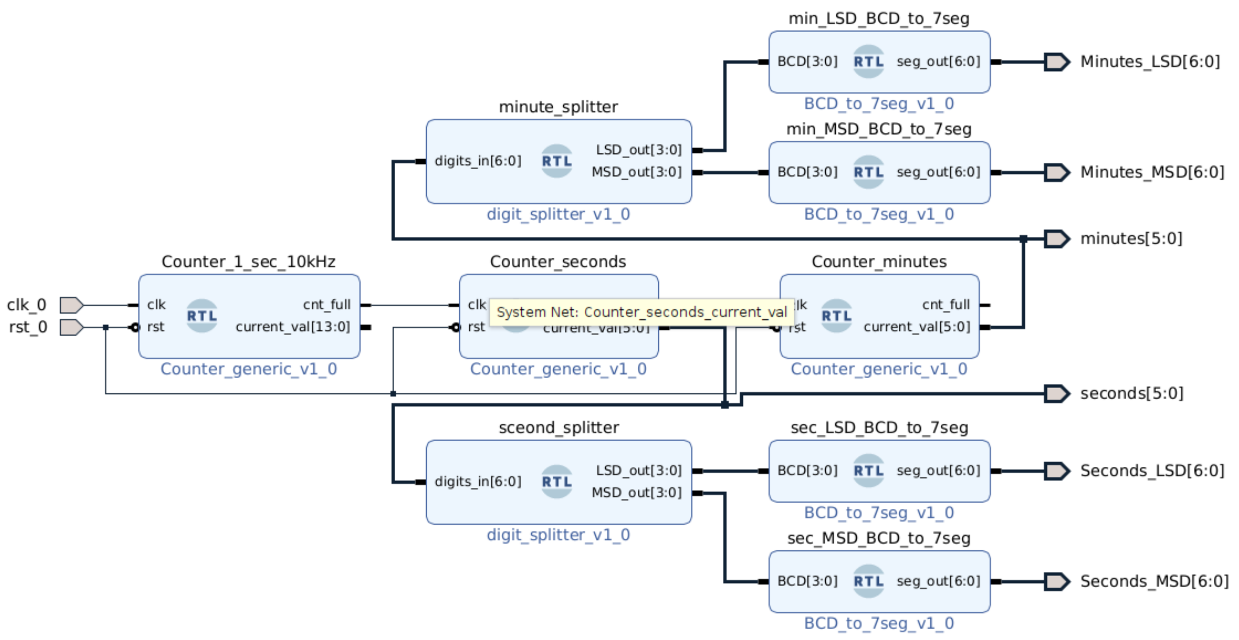
The code can be seen here:



```vhdl
entity BCD_to_7seg is
    Port ( BCD : in STD_LOGIC_VECTOR (3 downto 0);
           seg_out : out STD_LOGIC_VECTOR (6 downto 0));
end BCD_to_7seg;

architecture Behavioral of BCD_to_7seg is

begin
with BCD select
    seg_out <=  "0000001" when "0000",
                "1001111" when "0001",
                "0010010" when "0010",
                "0000100" when "0011",
                "1001100" when "0100",
                "0100100" when "0101",
                "0100000" when "0110",
                "0001111" when "0111",
                "0000000" when "1000",
                "0000100" when "1001",
                "0001000" when "1010",
                "1100000" when "1011",
                "0110001" when "1100",
                "1000010" when "1101",
                "0110000" when "1110",
                "0111000" when "1111",
                "1111111" when others;

end Behavioral;
```

Block design of the clock

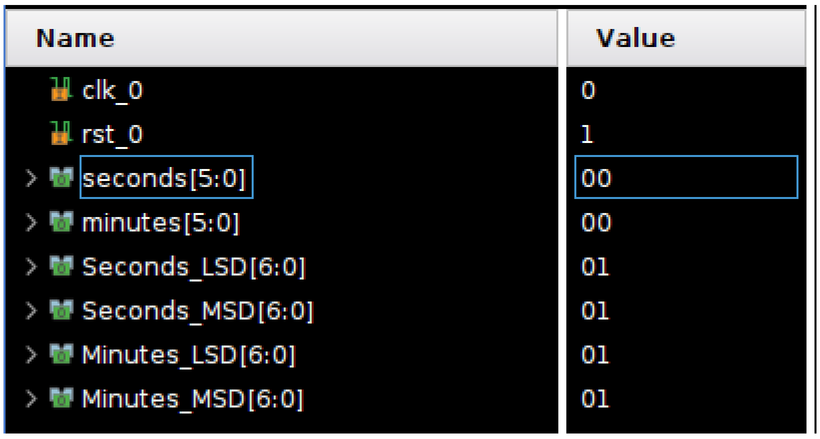
The clock itself is implemented as a block design that incorporates all of the previously mentioned components. In this case the clock works with a clk input of 10 kHz.



One counter counts the clock cycles that occur during one second, in this case it is assumed that the frequency is 10 kHz. When it has counted 10k signals it sends a signal to Counter_seconds and resets itself on the next clock signal. The Counter_seconds then counts this signal 60 times before sending a signal to Counter_minutes and resetting itself. Both the Counter_seconds and the Counter_minutes output their current value to digit_splitters which then feed in to two BCD_to_7seg blocks each.

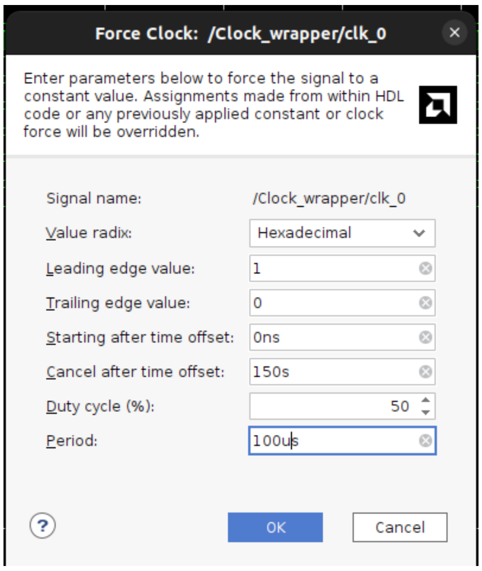
Verification and simulation

Before start of simulation the inputs and outputs look like this:



Name	Value
clk_0	0
rst_0	1
seconds[5:0]	00
minutes[5:0]	00
Seconds_LSD[6:0]	01
Seconds_MSD[6:0]	01
Minutes_LSD[6:0]	01
Minutes_MSD[6:0]	01

The clk input is then forced with this clock signal:

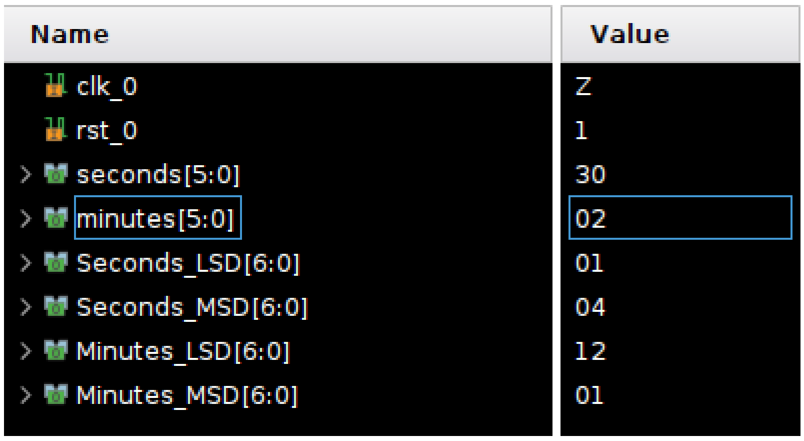


Force Clock: /Clock_wrapper/clk_0

Enter parameters below to force the signal to a constant value. Assignments made from within HDL code or any previously applied constant or clock force will be overridden.

Signal name:	/Clock_wrapper/clk_0
Value radix:	Hexadecimal
Leading edge value:	1
Trailing edge value:	0
Starting after time offset:	0ns
Cancel after time offset:	150s
Duty cycle (%):	50
Period:	100us

After simulation the clock should have counted to 150 seconds, which is 2 minutes and 30 seconds, and it has:



Name	Value
clk_0	Z
rst_0	1
seconds[5:0]	30
minutes[5:0]	02
Seconds_LSD[6:0]	01
Seconds_MSD[6:0]	04
Minutes_LSD[6:0]	12
Minutes_MSD[6:0]	01

The outputs for the seven segment display also matches with the output of the clock, with the code for “0” being displayed at the most significant digit of the minutes, then the code for “2” at the least significant digit, then the code for “3” as the MSD of the seconds, and “0” again for the LSD of the seconds.