Digital timer

DSD first year project

Name: MOCIAN ANDREI

Group: 30414

Course professor: CRET OCTAVIAN

Project supervisor: ALBU CRISTIAN

Table of Contents

Specifications

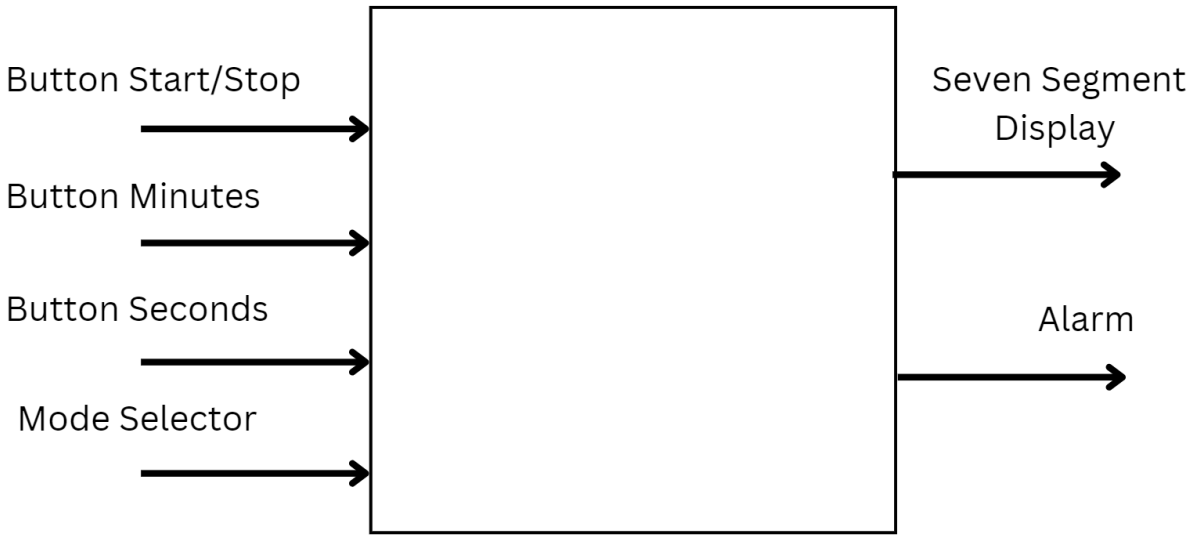
Design a timer with the following functionality: the device has 4 BCD displays - 7 segments. The first two displays are for minutes, the next two are for seconds. Thus, the maximum value that can be displayed is 99 minutes and 59 seconds.

The device has 3 buttons: M (from Minutes), S (from Seconds) and START / STOP. Assuming it is initially in the ZERO state, if the START / STOP button is pressed, the timer starts counting up. If the START / STOP button is pressed again, the timer stops at the value reached at that moment. If the START / STOP button is pressed again, the timer continues to count, etc. If it reaches 99 minutes and 59 seconds, ZERO follows again. If the M (Minute) and S (Second) buttons are pressed simultaneously, the timer is reset (becomes ZERO).

In any state, pressing the M button will increment and display the minute value. In any state, pressing the S button will increment and display the second value. Once a value for minutes and / or seconds has been set (by pressing the M or S buttons), and after using the mode switch, when the START / STOP button is pressed, the timer starts counting down from the current value "Minutes / Seconds" to ZERO, and when it reaches the ZERO state, an audible signal (alarm) is emitted.

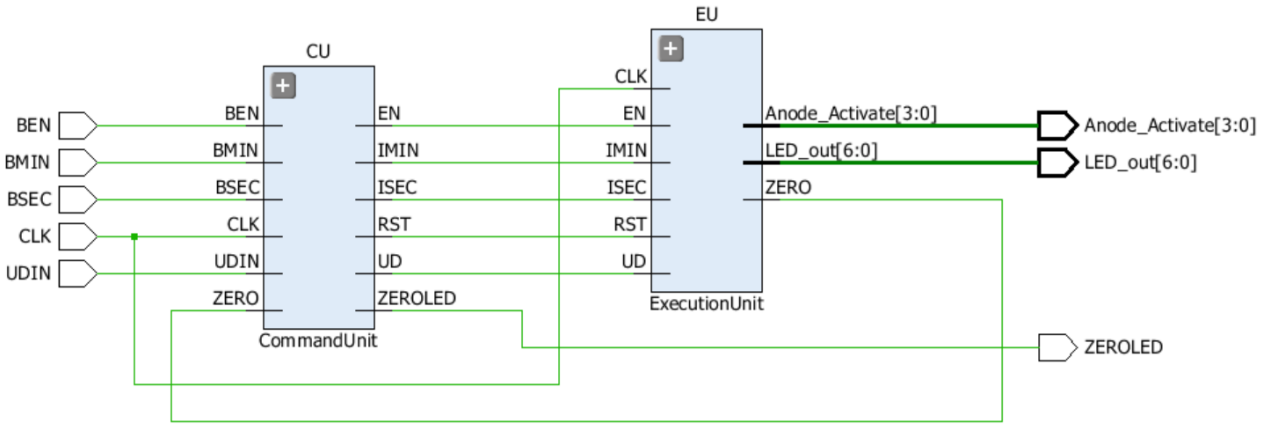
Design

Black Box



Control and Execution Unit

The control logic is represented by the Control Unit (CU) and the resources are represented by the Execution Unit (EU). The Execution Unit (EU) is responsible for driving the Seven Segment Display and the counting unit, while the Control Unit (CU) modifies the states of the machine.



```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity TOP is
    Port(CLK: in STD_LOGIC;
         BMIN: in STD_LOGIC;
         BSEC: in STD_LOGIC;
         BEN: in STD_LOGIC;
         UDIN: in STD_LOGIC;
         ZEROLED: out STD_LOGIC;
         Anode_Activate : out STD_LOGIC_VECTOR (3 downto 0);
         LED_out : out STD_LOGIC_VECTOR (6 downto 0));
end TOP;

architecture Behavioral of TOP is

signal AUXUD: STD_LOGIC;
signal AUXIMIN: STD_LOGIC;
signal AUXISEC: STD_LOGIC;
signal AUXRST: STD_LOGIC;
signal AUXEN: STD_LOGIC;
signal AUXZERO: STD_LOGIC;

component CommandUnit is
    Port(CLK: in STD_LOGIC;
         BMIN: in STD_LOGIC;
         BSEC: in STD_LOGIC;
         BEN: in STD_LOGIC;
         UDIN: in STD_LOGIC;
         ZERO: in STD_LOGIC;
         UD: out STD_LOGIC;
         IMIN: out STD_LOGIC;
         ISEC: out STD_LOGIC;
         RST: out STD_LOGIC;
         EN: out STD_LOGIC;
         ZEROLED: out STD_LOGIC);
end component;

component ExecutionUnit is
    port(RST: in STD_LOGIC;
         CLK: in STD_LOGIC;
         UD: in STD_LOGIC;
         EN: in STD_LOGIC;
         IMIN: in STD_LOGIC;
         ISEC: in STD_LOGIC;
         ZERO: out STD_LOGIC;
         Anode_Activate : out STD_LOGIC_VECTOR (3 downto 0);
         LED_out : out STD_LOGIC_VECTOR (6 downto 0));
end component;

begin
    CU: CommandUnit port map(CLK => CLK, BMIN => BMIN, BSEC => BSEC, BEN => BEN, UDIN
=> UDIN, ZERO => AUXZERO, UD => AUXUD, IMIN => AUXIMIN, ISEC => AUXISEC, RST =>
AUXRST, EN => AUXEN, ZEROLED => ZEROLED);
    EU: ExecutionUnit port map(RST => AUXRST, CLK => CLK, UD => AUXUD, EN => AUXEN,
IMIN => AUXIMIN, ISEC => AUXISEC, ZERO => AUXZERO, Anode_Activate => Anode_Activate,
LED_out => LED_out);
end Behavioral;
```

Code of the TOP file

Resources (breakdown of the EU)

The EU is composed of a **1 Hz Frequency Divider**, a **Counting unit**, a **Seven Segment Display controller**, and logic to find if the display shows “0000”.

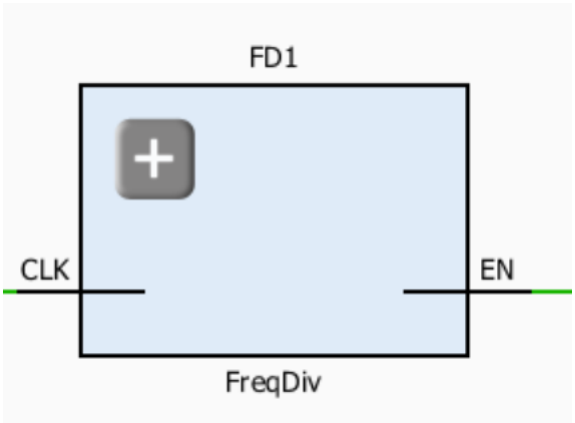
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ExecutionUnit is
    port(RST: in STD_LOGIC;
         CLK: in STD_LOGIC;
         UD: in STD_LOGIC;
         EN: in STD_LOGIC;
         IMIN: in STD_LOGIC;
         ISEC: in STD_LOGIC;
         ZERO: out STD_LOGIC;
         Anode_Activate : out STD_LOGIC_VECTOR (3 downto 0);
         LED_out : out STD_LOGIC_VECTOR (6 downto 0));
end ExecutionUnit;

architecture Behavioral of ExecutionUnit is
signal CLK_1hz: STD_LOGIC;
signal CLK_EN: STD_LOGIC;
signal AUX_EN: STD_LOGIC;
signal displayed_minutes_temp10: STD_LOGIC_VECTOR (3 downto 0);
signal displayed_minutes_temp01: STD_LOGIC_VECTOR (3 downto 0);
signal displayed_seconds_temp10: STD_LOGIC_VECTOR (3 downto 0);
signal displayed_seconds_temp01: STD_LOGIC_VECTOR (3 downto 0);

    component Debouncer is
    Port ( CLK : in STD_LOGIC;
           button : in STD_LOGIC;
           en : out STD_LOGIC);
    end component;

    component FreqDiv is
        port(CLK: in STD_LOGIC;
             EN: out STD_LOGIC);
    end component;

    component CountingUnit is
        Port(CLK: in STD_LOGIC;
             UD: in STD_LOGIC;
             RST: in STD_LOGIC;
             EN: in STD_LOGIC;
             IMIN: in STD_LOGIC;
             ISEC: in STD_LOGIC;
             displayed_minutes10: out STD_LOGIC_VECTOR (3 downto 0);
             displayed_minutes01: out STD_LOGIC_VECTOR (3 downto 0);
             displayed_seconds10: out STD_LOGIC_VECTOR (3 downto 0);
             displayed_seconds01: out STD_LOGIC_VECTOR (3 downto 0));
    end component;

    component SevenSegmentDisplay is
        Port(CLK: in STD_LOGIC;
             displayed_minutes10_ssd: in STD_LOGIC_VECTOR (3 downto 0);
             displayed_minutes01_ssd: in STD_LOGIC_VECTOR (3 downto 0);
             displayed_seconds10_ssd: in STD_LOGIC_VECTOR (3 downto 0);
             displayed_seconds01_ssd: in STD_LOGIC_VECTOR (3 downto 0);
             Anode_Activate : out STD_LOGIC_VECTOR (3 downto 0);
             LED_out : out STD_LOGIC_VECTOR (6 downto 0));
```

```
    end component;
begin
    CLK_EN <= CLK_1hz and EN;

    FD1: FreqDiv port map(CLK => CLK, EN => CLK_1hz);
    CU: CountingUnit port map(CLK => CLK_EN, UD => UD, RST => RST, EN => EN, IMIN =>
IMIN, ISEC => ISEC, displayed_minutes10 => displayed_minutes_temp10,
displayed_minutes01 => displayed_minutes_temp01, displayed_seconds10 =>
displayed_seconds_temp10, displayed_seconds01 => displayed_seconds_temp01);
    SSD: SevenSegmentDisplay port map(CLK => CLK, displayed_minutes10_ssd =>
displayed_minutes_temp10, displayed_minutes01_ssd => displayed_minutes_temp01,
displayed_seconds10_ssd => displayed_seconds_temp10, displayed_seconds01_ssd =>
displayed_seconds_temp01, Anode_Activate => Anode_Activate, LED_out => LED_out);

    --logic for finding if the display is "0000"
    process(displayed_minutes_temp10, displayed_minutes_temp01,
displayed_seconds_temp10, displayed_seconds_temp01)
    begin
        if displayed_minutes_temp10 = "0000" and displayed_minutes_temp01 = "0000" and
displayed_seconds_temp10 = "0000" and displayed_seconds_temp01 = "0000" then
            ZERO <= '1';
        else
            ZERO <= '0';
        end if;
    end process;
end Behavioral;
```

Code for the Counting Unit

1Hz Frequency divider

This VHDL module divides the input clock frequency to generate a 1Hz signal. It achieves this by counting 50 million clock pulses and toggling an output signal, effectively creating a clock enable signal that operates at 1Hz.

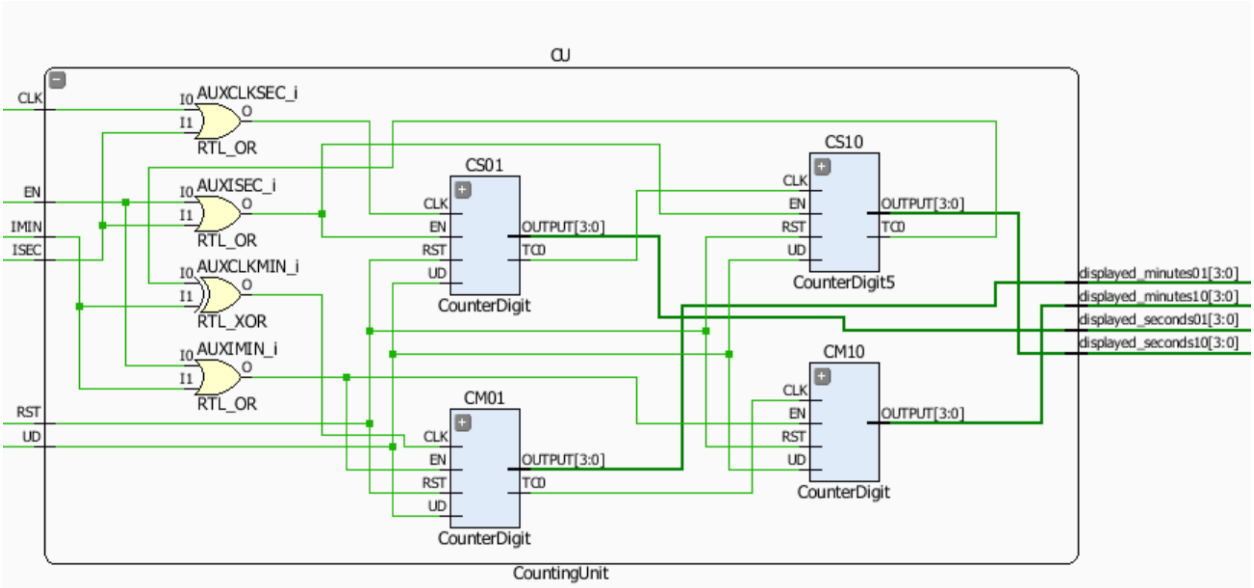


```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity FreqDiv is
    port(CLK: in STD_LOGIC;
         EN: out STD_LOGIC);
end FreqDiv;

architecture Behavioral of FreqDiv is
signal COUNT: STD_LOGIC_VECTOR(25 downto 0);
signal TEMP: STD_LOGIC;
begin
    process(CLK)
    begin
        if CLK = '1' and CLK'EVENT then
            COUNT <= COUNT + 1;
            if COUNT >= 49_999_999 then
                COUNT <= (others => '0');
                TEMP <= not TEMP;
            end if;
        end if;
        EN <= TEMP;
    end process;
end Behavioral;
```

Code for frequency divider

Counting unit

This unit uses **4 counters** for each digit (2 for displaying minutes, 2 for displaying seconds). The architecture ensures proper counting and cascading of digits, with enable and clock signals being appropriately controlled.

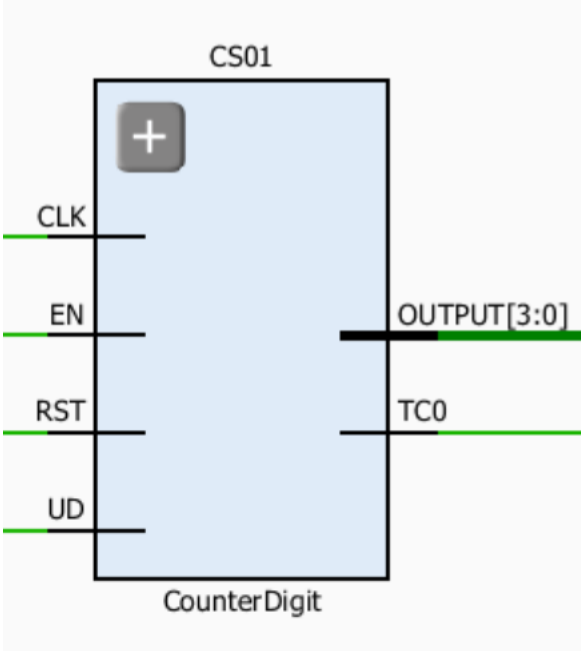


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity CountingUnit is
    Port(CLK: in STD_LOGIC;
         UD: in STD_LOGIC;
         RST: in STD_LOGIC;
         EN: in STD_LOGIC;
         IMIN: in STD_LOGIC;
         ISEC: in STD_LOGIC;
         displayed_minutes10: out STD_LOGIC_VECTOR (3 downto 0);
         displayed_minutes01: out STD_LOGIC_VECTOR (3 downto 0);
         displayed_seconds10: out STD_LOGIC_VECTOR (3 downto 0);
         displayed_seconds01: out STD_LOGIC_VECTOR (3 downto 0));
end CountingUnit;

architecture Behavioral of CountingUnit is

signal AUX0: STD_LOGIC;
signal AUX1: STD_LOGIC;
signal AUX2: STD_LOGIC;
signal AUX3: STD_LOGIC;
signal AUXISEC: STD_LOGIC;
signal AUXIMIN: STD_LOGIC;
signal AUXCLKMIN: STD_LOGIC;
signal AUXCLKSEC: STD_LOGIC;

component CounterDigit is
    port(RST: in STD_LOGIC;
         CLK: in STD_LOGIC;
         UD: in STD_LOGIC;
         EN: in STD_LOGIC;
         TC0: out STD_LOGIC;
         OUTPUT: out STD_LOGIC_VECTOR(3 downto 0));
end component;

component CounterDigit5 is
    port(RST: in STD_LOGIC;
         CLK: in STD_LOGIC;
         UD: in STD_LOGIC;
         EN: in STD_LOGIC;
         TC0: out STD_LOGIC;
         OUTPUT: out STD_LOGIC_VECTOR(3 downto 0));
end component;
begin
    AUXISEC <= EN or ISEC;
    AUXIMIN <= EN or IMIN;
    AUXCLKSEC <= CLK or ISEC;
    AUXCLKMIN <= AUX1 xor IMIN;

    CS01: CounterDigit port map(RST => RST, CLK => AUXCLKSEC, UD => UD, EN => AUXISEC,
TC0 => AUX0, OUTPUT => displayed_seconds01);
    CS10: CounterDigit5 port map(RST => RST, CLK => AUX0, UD => UD, EN => AUXISEC, TC0
=> AUX1, OUTPUT => displayed_seconds10);
    CM01: CounterDigit port map(RST => RST, CLK => AUXCLKMIN, UD => UD, EN => AUXIMIN,
TC0 => AUX2, OUTPUT => displayed_minutes01);
    CM10: CounterDigit port map(RST => RST, CLK => AUX2, UD => UD, EN => AUXIMIN,
TC0=> AUX3, OUTPUT => displayed_minutes10);
end Behavioral;
```

Code of the counting unit

Counters

The counting unit uses 2 different types of counters. They are pretty much the same, except that one is **modulo 5**, and the others are **modulo 9**. A module like this is a 4-bit counter capable of both incrementing and decrementing between 0 and 9, or 0 to 5. It includes a synchronous reset, enable functionality, and terminal count detection for overflow and underflow conditions. The output TC0 indicates when the counter has reached its terminal value in either direction. The OUTPUT provides the current count value.



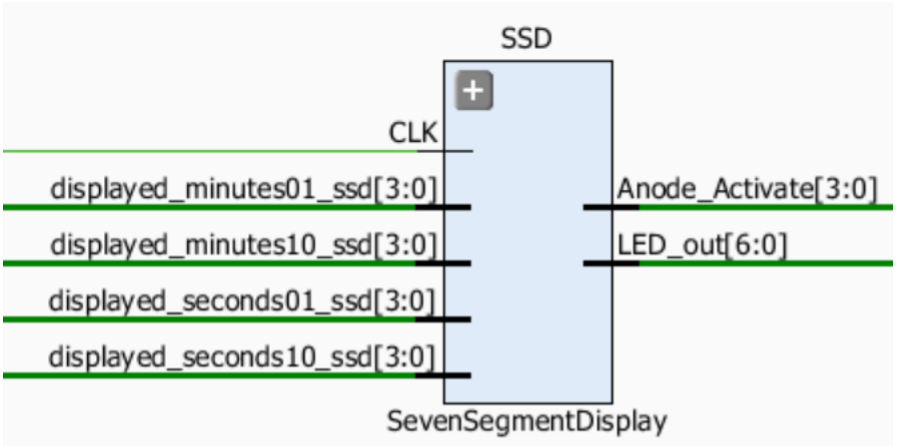

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity CounterDigit is
    port(RST: in STD_LOGIC;
         CLK: in STD_LOGIC;
         UD: in STD_LOGIC;
         EN: in STD_LOGIC;
         TC0: out STD_LOGIC;
         OUTPUT: out STD_LOGIC_VECTOR(3 downto 0));
end CounterDigit;

architecture Behavioral of CounterDigit is
signal COUNT: STD_LOGIC_VECTOR(3 downto 0);
begin
    process(CLK, RST)
        begin
            if RST = '1' then
                COUNT <= (others => '0');
            elsif rising_edge(CLK) then
                if EN = '1' then
                    if UD = '0' then
                        COUNT <= COUNT + 1;
                    else
```

```
                    COUNT <= COUNT - 1;
                end if;

                if COUNT >= x"9" and UD = '0' then
                    COUNT <= (others => '0');
                end if;
                if COUNT = x"0" and UD = '1' then
                    COUNT <= x"9";
                end if;
            end if;
        end if;
    end process;
process(COUNT, UD)
    begin
        if UD = '0' then
            if COUNT = x"0" then
                TC0 <= '1';
            else
                TC0 <= '0';
            end if;
        else
            if COUNT = x"9" then
                TC0 <= '1';
            else
                TC0 <= '0';
            end if;
        end if;
    end process;

OUTPUT <= COUNT;
end Behavioral;
```

Code of a counter

Seven Segment Display Controller

The Seven Segment Display Controller uses the display to show digits between 0 and 9. It consists of a decoder and a frequency divider to maintain the refresh rate of the display.

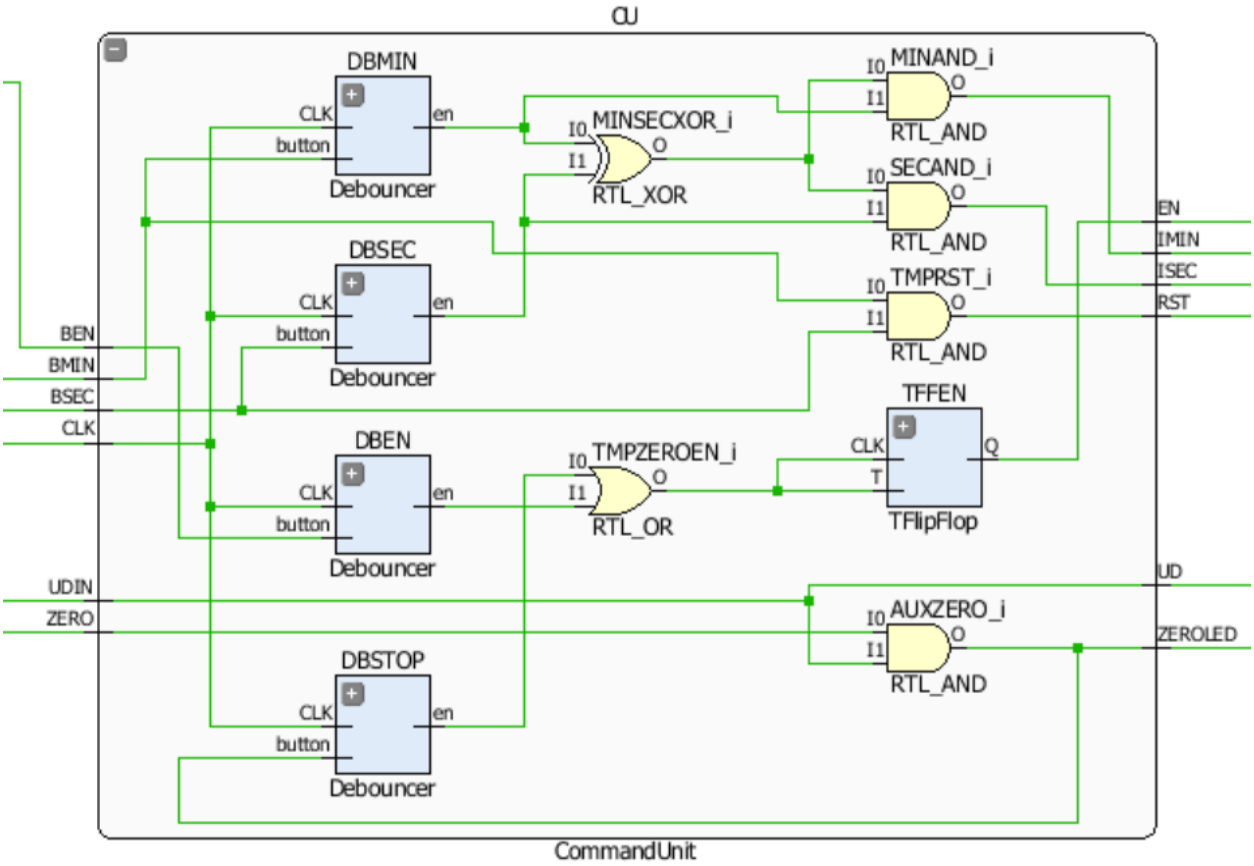


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity SevenSegmentDisplay is
    Port(CLK: in STD_LOGIC;
         displayed_minutes10_ssd: in STD_LOGIC_VECTOR (3 downto 0);
         displayed_minutes01_ssd: in STD_LOGIC_VECTOR (3 downto 0);
         displayed_seconds10_ssd: in STD_LOGIC_VECTOR (3 downto 0);
         displayed_seconds01_ssd: in STD_LOGIC_VECTOR (3 downto 0);
         Anode_Activate : out STD_LOGIC_VECTOR (3 downto 0);
         LED_out : out STD_LOGIC_VECTOR (6 downto 0));
end SevenSegmentDisplay;

architecture Behavioral of SevenSegmentDisplay is
signal LED_BCD: STD_LOGIC_VECTOR (3 downto 0);
    signal refresh_counter: STD_LOGIC_VECTOR (16 downto 0);
signal LED_activating_counter: std_logic_vector(1 downto 0);
begin
    process(LED_BCD)
    begin
        case LED_BCD is
            when "0000" => LED_out <= "0000001"; -- "0"
            when "0001" => LED_out <= "1001111"; -- "1"
            when "0010" => LED_out <= "0010010"; -- "2"
            when "0011" => LED_out <= "0000110"; -- "3"
            when "0100" => LED_out <= "1001100"; -- "4"
            when "0101" => LED_out <= "0100100"; -- "5"
            when "0110" => LED_out <= "0100000"; -- "6"
            when "0111" => LED_out <= "0001111"; -- "7"
            when "1000" => LED_out <= "0000000"; -- "8"
            when "1001" => LED_out <= "0000100"; -- "9"
            when others =>
        end case;
    end process;
    process(CLK)
    begin
        if CLK = '1' and CLK'EVENT then
            refresh_counter <= refresh_counter + 1;
        end if;
    end process;
    LED_activating_counter <= refresh_counter(16 downto 15);

    process(LED_activating_counter)
    begin
        case LED_activating_counter is
            when "00" =>
                Anode_Activate <= "0111";
                LED_BCD <= displayed_minutes10_ssd(3 downto 0);
            when "01" =>
                Anode_Activate <= "1011";
                LED_BCD <= displayed_minutes01_ssd(3 downto 0);
            when "10" =>
                Anode_Activate <= "1101";
                LED_BCD <= displayed_seconds10_ssd(3 downto 0);
            when "11" =>
                Anode_Activate <= "1110";
                LED_BCD <= displayed_seconds01_ssd(3 downto 0);
            when others =>
            end case;
    end process;
end Behavioral;
```

Code for the SSD Controller

Breakdown of the CU

The CU is used to control the states of the EU. It also deals with user input. It consists of **debounces**, a **T Flip-Flop**, and some additional logic. The XOR gate is used to make sure if we press the min and sec buttons, it will only happen a reset.



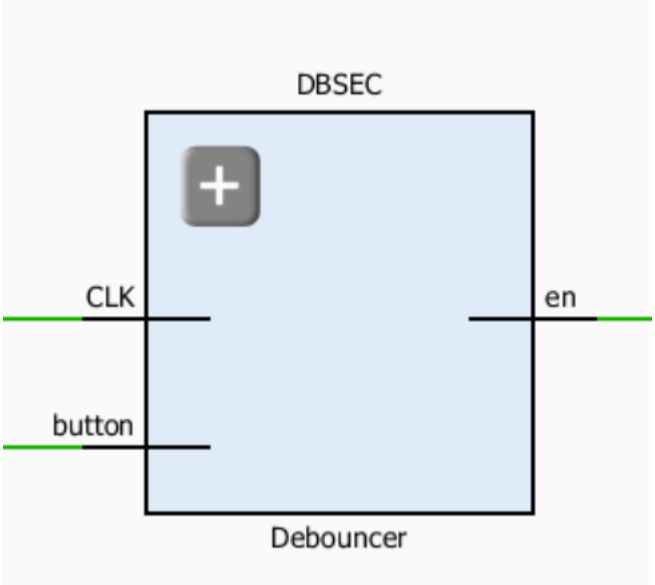

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity CommandUnit is
    Port(CLK: in STD_LOGIC;
         BMIN: in STD_LOGIC;
         BSEC: in STD_LOGIC;
         BEN: in STD_LOGIC;
         UDIN: in STD_LOGIC;
         ZERO: in STD_LOGIC;
         UD: out STD_LOGIC;
         IMIN: out STD_LOGIC;
         ISEC: out STD_LOGIC;
         RST: out STD_LOGIC;
         EN: out STD_LOGIC;
         ZEROLED: out STD_LOGIC);
end CommandUnit;

architecture Behavioral of CommandUnit is
    signal MINSECXOR: STD_LOGIC;
```

```
    signal MINAND: STD_LOGIC;
    signal SECAND: STD_LOGIC;

    signal TMPBEN: STD_LOGIC;
    signal TMPMIN: STD_LOGIC;
    signal TMPSEC: STD_LOGIC;

    signal TMPRST: STD_LOGIC;

    signal TMPEN: STD_LOGIC;
    signal TMPUD: STD_LOGIC;
    signal TMPZEROEN: STD_LOGIC;

    signal AUXZERO: STD_LOGIC;
    signal ZEROEX: STD_LOGIC;

    component Debouncer is
    Port ( CLK : in STD_LOGIC;
           button : in STD_LOGIC;
           en : out STD_LOGIC);
    end component;

    component TFlipFlop is
        Port(T: in STD_LOGIC;
             CLK: in STD_LOGIC;
             Q: out STD_LOGIC);
    end component;
begin
    DBEN: Debouncer port map(CLK => CLK, button => BEN, EN => TMPBEN);
    DBMIN: Debouncer port map(CLK => CLK, button => BMIN, EN => TMPMIN);
    DBSEC: Debouncer port map(CLK => CLK, button => BSEC, EN => TMPSEC);
    DBSTOP: Debouncer port map(CLK => CLK, button => AUXZERO, EN => ZEROEX);
    TFFEN: TFlipFlop port map(CLK => TMPZEROEN, T => TMPZEROEN, Q => TMPEN);


    MINSECXOR <= TMPMIN xor TMPSEC;
    MINAND <= MINSECXOR and TMPMIN;
    SECAND <= MINSECXOR and TMPSEC;

    TMPRST <= BMIN and BSEC;

    TMPUD <= UDIN;
    AUXZERO <= ZERO and TMPUD;
    TMPZEROEN <= ZEROEX or TMPBEN;

    EN <= TMPEN;
    IMIN <= MINAND;
    ISEC <= SECAND;
    RST <= TMPRST;
    UD <= TMPUD;
    ZEROLED <= AUXZERO;
end Behavioral;
```

A top view of the Command unit

Debouncer

The Debouncer is used to make sure a single button press is registered. It works by using a counter to periodically sample the button state, effectively filtering out the high-frequency noise associated with mechanical button presses. The sampled state is then passed through a chain of flip-flops to ensure that only stable button presses are detected. The output **en** is asserted only when a stable transition from '0' to '1' is detected in the button state, providing a clean, debounced signal.



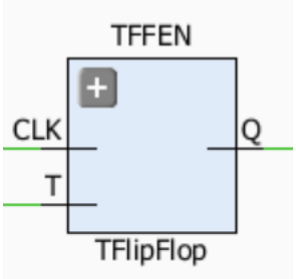

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Debouncer is
    Port ( CLK : in STD_LOGIC;
           button : in STD_LOGIC;
           en : out STD_LOGIC);
end Debouncer;

architecture Behavioral of Debouncer is
signal count : std_logic_vector (15 downto 0) := (others => '0');
signal t : std_logic;
signal q1, q2, q3 : std_logic;
begin
    process(CLK)
    begin
        if rising_edge(CLK) then
            count <= count + 1;
        end if;
    end process;

    t <= '1' when count = x"FFFF" else '0';

    process(CLK)
```

```
    begin
        if rising_edge(CLK) then
            if t = '1' then
                q1 <= button;
            end if;
        end if;
    end process;

    process(CLK)
    begin
        if rising_edge(CLK) then
            q2 <= q1;
            q3 <= q2;
        end if;
    end process;

    en <= q2 and not q3;

end Behavioral;
```

Code for the debouncer

T Flip-Flop

The T Flip-Flop is used for toggling a signal after a button press. By putting the input, the same on the T entry and the CLK, the output will be a toggle between '1' and '0'.




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TFlipFlop is
    Port(T: in STD_LOGIC;
         CLK: in STD_LOGIC;
         Q: out STD_LOGIC);
end TFlipFlop;

architecture Behavioral of TFlipFlop is
signal temp: STD_LOGIC;
begin
    process(CLK)
    begin
        if CLK = '1' and CLK'EVENT then
            temp <= not temp;
        end if;
    end process;
    Q <= temp;
end Behavioral;
```

Code for T Flip-Flop

Constraint file

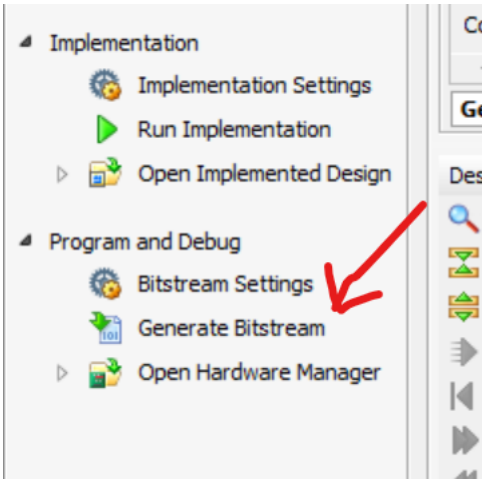
```
## Clock signal
set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports CLK]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports CLK]

## Switches
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports {UDIN}]

## LEDs
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports {ZEROLED}]

##7 Segment Display
set_property -dict { PACKAGE_PIN W7   IOSTANDARD LVCMOS33 } [get_ports {LED_out[6]}]
set_property -dict { PACKAGE_PIN W6   IOSTANDARD LVCMOS33 } [get_ports {LED_out[5]}]
set_property -dict { PACKAGE_PIN U8   IOSTANDARD LVCMOS33 } [get_ports {LED_out[4]}]
set_property -dict { PACKAGE_PIN V8   IOSTANDARD LVCMOS33 } [get_ports {LED_out[3]}]
set_property -dict { PACKAGE_PIN U5   IOSTANDARD LVCMOS33 } [get_ports {LED_out[2]}]
set_property -dict { PACKAGE_PIN V5   IOSTANDARD LVCMOS33 } [get_ports {LED_out[1]}]
set_property -dict { PACKAGE_PIN U7   IOSTANDARD LVCMOS33 } [get_ports {LED_out[0]}]

#set_property -dict { PACKAGE_PIN V7   IOSTANDARD LVCMOS33 } [get_ports dp]

set_property -dict { PACKAGE_PIN U2   IOSTANDARD LVCMOS33 } [get_ports
{Anode_Activate[0]}]
set_property -dict { PACKAGE_PIN U4   IOSTANDARD LVCMOS33 } [get_ports
{Anode_Activate[1]}]
set_property -dict { PACKAGE_PIN V4   IOSTANDARD LVCMOS33 } [get_ports
{Anode_Activate[2]}]
set_property -dict { PACKAGE_PIN W4   IOSTANDARD LVCMOS33 } [get_ports
{Anode_Activate[3]}]


##Buttons
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports BEN]
set_property -dict { PACKAGE_PIN W19   IOSTANDARD LVCMOS33 } [get_ports BMIN]
set_property -dict { PACKAGE_PIN T17   IOSTANDARD LVCMOS33 } [get_ports BSEC]
```

User manual

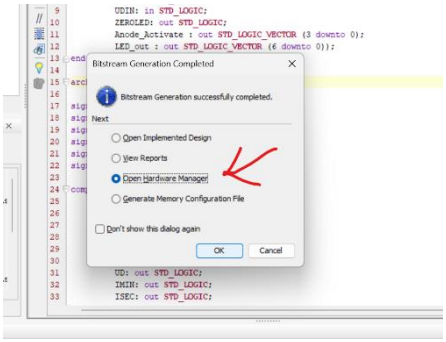
The Basys 3 is one of the best boards on the market for getting started with FPGA. It is an entry-level development board built around a Xilinx Artix-7 FPGA.

As a complete and ready-to use digital circuit development platform, it includes enough switches, LEDs, and other I/O devices to allow a large number of designs to be completed without the need for any additional hardware. There are also enough uncommitted FPGA I/O pins to allow designs to be expanded using Digilent Pmods or other custom boards and circuits, and all of this at a student-friendly price point. To load the source on the board we need to follow these steps:

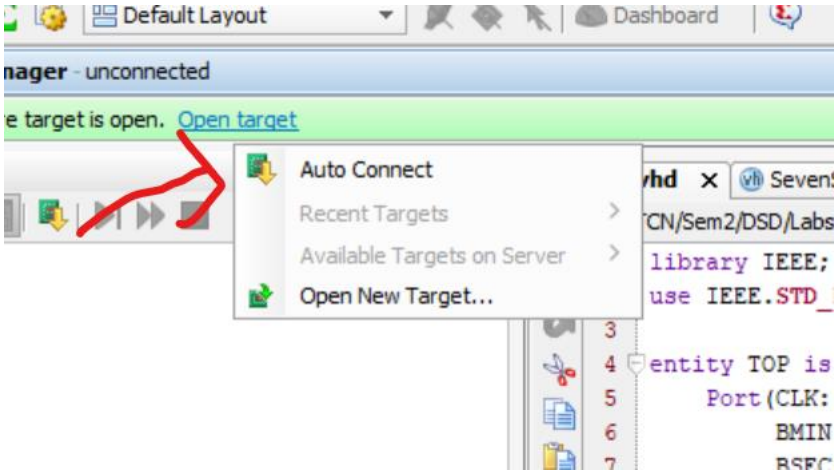
1. Generate bitstream



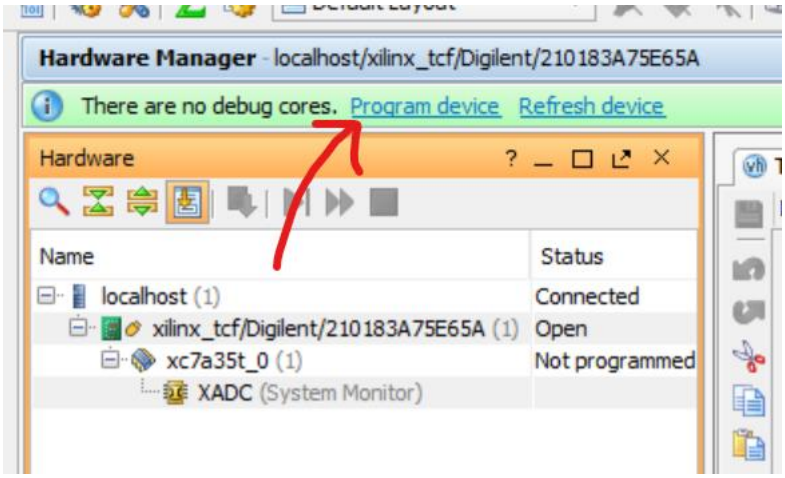
2. Open Hardware Manager



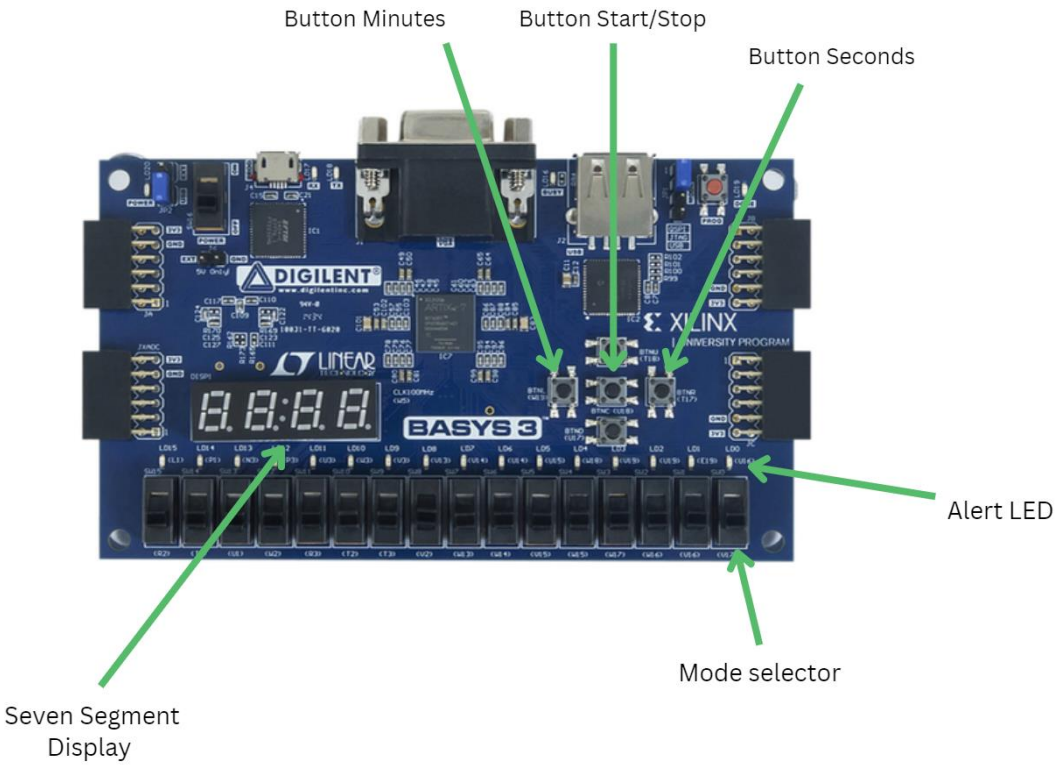
3. Connect board



4. Program device



How to use



- After power-up, the display will show 0000.
- If the user presses the Start/Stop button, it will start in stopwatch mode, up to 99 minutes and 59 seconds. After that it will go back to 0
- If the timer is in pause, we can use the Button for minutes and for seconds to increment them respectively.
- After the incrementation, we can use the mode selector to put it in timer mode.
- If the timer reaches 0, the timer will stop, and the alert LED will be illuminated.
- In any state, if the user presses the button minutes and button seconds simultaneously, the timer will reset.

Technical justifications for design

The design of a timer with four BCD displays for minutes and seconds, controlled by three buttons (Minutes, Seconds, Start/Stop), is technically justified by using four individual counters, each responsible for one digit. This modular approach simplifies the logic required for incrementing, decrementing, and resetting values, as each counter independently manages a single decimal place, aligning directly with the 7-segment display requirements. Handling the Minutes and Seconds buttons is straightforward, with each counter incrementing its respective digit and managing rollover efficiently. The Start/Stop button facilitates smooth transitions between counting and pausing by enabling or disabling the counters, and the design easily supports both count-up and count-down modes. In count-up mode, counters increment and carry over values, while in count-down mode, they decrement and borrow values until reaching the zero state, at which point an LED will illuminate. The reset functionality is efficiently implemented by resetting all counters simultaneously when both M and S buttons are pressed. This clear, scalable, and manageable design ensures reliable operation, straightforward state management, and easy future modifications.

Future developments

Future developments for this timer design could include allowing for hours to be displayed in addition to minutes and seconds. This should be possible on a more advanced board.

Also, adding a memory module to store timer presets, allowing users to quickly select and start commonly used countdown values could be viable solution for the future.

References

- Course Logic Design Year 1, sem. 1
- Course Digital System Design Year 1, sem. 2
- fpga4student

- The complete project can be also found at https://github.com/andreimocian/Digital_Timer.