*DSD Project 2022*

Medication Scheduler

# The idea behind the project:

Construct a medication dispenser where the user can input the time and an alarm time according to his schedule and when this time comes the medication dispenser rings to remind the user to take his medicine,once the user approaches it a lid will be opened to dispense the medicine then it will wait a few seconds before closing the lid.

## Inputs and Outputs:

Clk:a 50 MHz clock signal
Rst_n:reset signal
H_in1, H_in0, M_in1, M_in0 ,S_in1 ,S_in0:inputs to set the time.
H_out1, H_out0, M_out1, M_out0 ,S_out1 ,S_out0:seven-segment displays for hours, minutes, and seconds
Sensor:input from IR sensor
Buzzer:buzzer output
Servo:output from motor
time_input :set the time
Alarm_input:activate the alarm
digit_confirm: input to confirm the time and alarm inputs.
Bin: an input port for a 4-bit binary number
Hout: an output port for a 7-bit hexadecimal number
Clk50MHz: an input port for the 50 MHz clock
Clk_stb: an output port for the 1 Hz clock after dividing with “clk_div” component.

1. **Entity Declaration:**

The entity declaration defines the top-level design of the system, and it specifies the input and output ports that are used to interface with the rest of the system. The entity is called "project1", and it is declared as follows:

```
ENTITY project1 IS
    PORT (
        clk : IN STD_LOGIC; -- clock 50 MHz
        rst_n : IN STD_LOGIC;

         -- Values for reset
        H_in1 : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0);
        H_in0 : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0);
        M_in1 : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0);
        M_in0 : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0);
        S_in1 : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0);
        S_in0 : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0);

         -- 7-segment outputs
        H_out1 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
        H_out0 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
        M_out1 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
        M_out0 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
        S_out1 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
        S_out0 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);

        sensor : IN STD_LOGIC;
        buzzer : BUFFER STD_LOGIC;
        servo : BUFFER STD_LOGIC;

        -- Inputs for time editing and setting
         time_input : IN STD_LOGIC;
         alarm_input : IN STD_LOGIC;
         digit_confirm : IN STD_LOGIC;
    );
END project1;
```

The entity has a number of input and output ports, which are used to communicate with the rest of the system. The input ports include:

"clk": a 50 MHz clock signal
"rst_n": a reset signal
"H_in1", "H_in0", "M_in1", "M_in0", "S_in1", and "S_in0": inputs for setting the time
"sensor": an input from an IR sensor
"time_input" and "alarm_input": inputs for setting the time and activating the alarm
"digit_confirm": an input for confirming time-setting and alarm-setting inputs
The output ports include:

"H_out1", "H_out0", "M_out1", "M_out0", "S_out1", and "S_out0": seven-segment displays for hours, minutes, and seconds
"buzzer": a buzzer output
"servo": an output for a servo/motor
These input and output ports are defined using a number of types and attributes, such as "IN", "OUT", "BUFFER".

## 2. Component Declaration:

The component declarations are used to define the interfaces and behavior of two custom components: "bin2hex" and "clk_div". These components can then be instantiated and used within the design to perform specific tasks.

### a) "bin2hex" component:

The "bin2hex" component is used to convert a 4-bit binary number to a 7-bit hexadecimal number, which can be used to drive the seven-segment displays. The component is declared as follows:

```
COMPONENT bin2hex
    PORT (
        Bin : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        Hout : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
    );
END COMPONENT;
```

The component has two ports:

- "Bin": an input port for a 4-bit binary number
- "Hout": an output port for a 7-bit hexadecimal number

### b) "clk_div" component:

The "clk_div" component is used to divide the 50 MHz clock signal down to a 1-second clock signal. The component is declared as follows:

```
COMPONENT clk_div
    PORT (
        CLK50MHZ : IN STD_LOGIC;
        clk_stb : BUFFER STD_LOGIC
    );
END COMPONENT;
```

The component has two ports:

- "CLK50MHZ": an input port for the 50 MHz clock
- “Clk_stb”: an output port for the 1 Hz clock

## 3. Component Declaration:

The signal declarations are used to define intermediate values that are used to store data or perform calculations during the design process. These signals are typically defined within the architecture section of the design, and they are used within processes and component instantiations to perform various tasks.

The code includes the following signal declarations:

```
SIGNAL clk_1s : STD_LOGIC; -- 1-s clock
SIGNAL counter_hour, counter_minute, counter_second : INTEGER;
-- counter using for create time
SIGNAL H_out1_bin : STD_LOGIC_VECTOR(3 DOWNTO 0); --The most significant
digit of the hour
SIGNAL H_out0_bin : STD_LOGIC_VECTOR(3 DOWNTO 0);--The least significant
digit of the hour
SIGNAL M_out1_bin : STD_LOGIC_VECTOR(3 DOWNTO 0);--The most significant
digit of the minute
SIGNAL M_out0_bin : STD_LOGIC_VECTOR(3 DOWNTO 0);--The least significant
digit of the minute
SIGNAL S_out1_bin : STD_LOGIC_VECTOR(3 DOWNTO 0);--The most significant
digit of the second
SIGNAL S_out0_bin : STD_LOGIC_VECTOR(3 DOWNTO 0);--The least significant
digit of the second
SIGNAL buzzCounter : INTEGER RANGE 0 TO 10;
```

The first signal, "clk_1s", is a 1-second clock signal that is generated by the "clk_div" component. The second set of signals, "counter_hour", "counter_minute", and "counter_second", are counters that are used to keep track of the time.

The next set of signals, "H_out1_bin", "H_out0_bin", "M_out1_bin", "M_out0_bin", "S_out1_bin", and "S_out0_bin", are intermediate signals that are used to store the binary values for the digits of the time, which are then converted to hexadecimal values and displayed on the seven-segment displays.
Finally, the "buzzCounter" signal is an integer that is used to keep track of the number of seconds since the buzzer has been activated. This signal is used to limit the duration of the buzzer sound, so that it doesn't continue indefinitely.

## 4. Processes:

This part defines several processes that are responsible for different aspects of the design's behavior.

### a) "create_1s_clock" process:

This process creates a 1-second clock signal using the "clk_div" component. The "clk_div" component takes the 50 MHz clock signal as input and divides it down to a 1-second clock signal, which is then assigned to the "clk_1s" signal. The "create_1s_clock" process is triggered by the rising edge of the "clk_1s" signal, and it activates the "clk_div" component whenever the signal goes high. The code for this process is as follows:

```
create_1s_clock : clk_div PORT MAP(clk, clk_1s);
```

### b) "reset" process:

This process is responsible for resetting the time to the default value stored in "H_in1", "H_in0", "M_in1", "M_in0", "S_in1", and "S_in0". The code for this process is as follows:

```
PROCESS (clk_1s, rst_n) BEGIN
        IF (rst_n = '0') THEN
            counter_hour <= to_integer(unsigned(H_in1)) * 10 + to_integer(unsigned(H_in0));
            counter_minute <= to_integer(unsigned(M_in1)) * 10 + to_integer(unsigned(M_in0));
            counter_second <= to_integer(unsigned(S_in1)) * 10 + to_integer(unsigned(S_in0));
        END IF;
```

**c) <u>"sensor_check" process:</u>**

This process is responsible for checking the value of the "sensor" input and activating or deactivating the buzzer and servo/motor based on the sensor's value.

If the time is 12:50:20, it activates the buzzer. It deactivates the buzzer and activates the servo/motor when a hand is detected. If the servo/motor is active, it increments the "buzzCounter" signal and deactivates the servo/motor if the "buzzCounter" value reaches 1 (making the motor work for 2 seconds after placing your hand to give it time to dispense the medication).

The code for this process is as follows:

```
---- SENSOR CHECK --------
IF sensor = '0' AND buzzer = '1' THEN
     servo <= '1';
     buzzer <= '0';
     buzzCounter <= 0;
END IF;
IF (servo = '1') THEN
      buzzCounter <= buzzCounter + 1;
END IF;
IF (buzzCounter >= 1) THEN
        servo <= '0';
        buzzCounter <= 0;
END IF;

IF (counter_hour = 12 AND counter_minute = 50 AND counter_second = 20) THEN
        buzzer <= '1';
END IF;
```

**d) "time_counter process"**

The "time_counter" process is responsible for counting the elapsed time and updating the hour, minute, and second counters accordingly. It does this by using the "clk_1s" signal, which is a 1-second clock signal that is generated by the "clk_div" component. The "time_counter" process increments the second counter by 1 every time the "clk_1s" signal goes high, and it increments the minute and hour counters whenever the second and minute counters respectively exceed 59. The code for the "time_counter" process is as follows:

```
-- clock operation ---|
PROCESS (clk_1s, rst_n) BEGIN

    IF (rst_n = '0') THEN
        counter_hour <= to_integer(unsigned(H_in1)) * 10 + to_integer(unsigned(H_in0));
        counter_minute <= to_integer(unsigned(M_in1)) * 10 + to_integer(unsigned(M_in0));
        counter_second <= to_integer(unsigned(S_in1)) * 10 + to_integer(unsigned(S_in0));
    ELSIF (rising_edge(clk_1s)) THEN
```

```
            counter_second <= counter_second + 1;
            IF (counter_second >= 59) THEN -- second > 59 then minute increases
                counter_minute <= counter_minute + 1;
                counter_second <= 0;
                IF (counter_minute >= 59) THEN -- minute > 59 then hour increases
                    counter_minute <= 0;
                    counter_hour <= counter_hour + 1;
                    IF (counter_hour >= 24) THEN -- hour > 24 then set hour to 0
                        counter_hour <= 0;
                    END IF;
                END IF;
            END IF;
        END IF;
    END PROCESS;
```

**e) "time_conversion" process**

The "time_conversion" process is responsible for converting the binary values of the hour, minute, and second counters to hexadecimal values and displaying them on the seven-segment displays. It does this by using the "bin2hex" component, which takes a 4-bit binary value as input and outputs a 7-bit hexadecimal value that can be displayed on a seven-segment display.
The code for the "time_conversion" process is as follows:

```
-- Conversion time ---
    -- H_out1 binary value
    H_out1_bin <= x"2" WHEN counter_hour >= 20 ELSE
        x"1" WHEN counter_hour >= 10 ELSE
        x"0";
    -- 7-Segment LED display of H_out1
    convert_hex_H_out1 : bin2hex PORT MAP(Bin => H_out1_bin, Hout => H_out1);
    -- H_out0 binary value
    H_out0_bin <= STD_LOGIC_VECTOR(to_unsigned((counter_hour-to_integer(unsigned(H_out1_bin)) * 10), 4));
    -- 7-Segment LED display of H_out0
    convert_hex_H_out0 : bin2hex PORT MAP(Bin => H_out0_bin, Hout => H_out0);
    -- M_out1 binary value
    M_out1_bin <= x"5" WHEN counter_minute >= 50 ELSE
        x"4" WHEN counter_minute >= 40 ELSE
        x"3" WHEN counter_minute >= 30 ELSE
        x"2" WHEN counter_minute >= 20 ELSE
        x"1" WHEN counter_minute >= 10 ELSE
        x"0";
    -- 7-Segment LED display of M_out1
    convert_hex_M_out1 : bin2hex PORT MAP(Bin => M_out1_bin, Hout => M_out1);
    -- M_out0 binary value
    M_out0_bin <= STD_LOGIC_VECTOR(to_unsigned((counter_minute - to_integer(unsigned(M_out1_bin)) * 10), 4));
    -- 7-Segment LED display of M_out0
    convert_hex_M_out0 : bin2hex PORT MAP(Bin => M_out0_bin, Hout => M_out0);
    -- S_out1 binary value
    S_out1_bin <= x"5" WHEN counter_second >= 50 ELSE
```

```
    x"4" WHEN counter_second >= 40 ELSE
    x"3" WHEN counter_second >= 30 ELSE
    x"2" WHEN counter_second >= 20 ELSE
    x"1" WHEN counter_second >= 10 ELSE
    x"0";
-- 7-Segment LED display of S_out1
convert_hex_S_out1 : bin2hex PORT MAP(Bin => S_out1_bin, Hout => S_out1);
-- S_out0 binary value
S_out0_bin <= STD_LOGIC_VECTOR(to_unsigned((counter_second - to_integer(unsigned(S_out1_bin)) * 10), 4));
-- 7-Segment LED display of S_out0
convert_hex_S_out0 : bin2hex PORT MAP(Bin => S_out0_bin, Hout => S_out0);
```

## 5. Helper Entities:

### a) "clk_div"

The "clk_div" entity uses the 50 MHz clock to generate a 1 Hz clock. It does so by flipping the signal of the 1 Hz clock every half of a period of the 50 MHz clock which is every 25,000,000 counts

The code for this entity is as follows:

```
---1HZ CLOCK
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity clk_div is
    port(
        CLK50MHZ: in std_logic;
--        CPU_RESETN: in std_logic;
--        reset: in std_logic;
        clk_stb: buffer std_logic
    );
end clk_div;

architecture b1 of clk_div is
    constant half_period: integer := 25000000; -- 50M/2
    signal counter: integer range 0 to half_period - 1 := 0;
    signal ncount: std_logic := '0';

begin
    process(CLK50MHZ)
    begin
        if rising_edge(CLK50MHZ) then
                        if counter = half_period - 1 then
                counter <= 0;
                clk_stb <= not clk_stb;
            else
                counter <= counter + 1;
            end if;
```

```
        end if;
    end process;
end b1;
```

**b) <u>"bin2hex" entity</u>**

This component converts a 4-bit binary number into a 7-segment hexadecimal display. The "bin2hex" entity has two ports: "Bin", which is the 4-bit binary input, and "Hout", which is the 7-segment hexadecimal output.

Its architecture has a process that performs the conversion. The process has a "CASE" statement that maps each possible 4-bit binary input to a 7-segment hexadecimal output. For example, when the "Bin" input is "0000", the output "Hout" is set to "1000000". When the "Bin" input is "0001", the output "Hout" is set to "1111001", and so on.

Its code is as follows:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY bin2hex IS
    PORT (
        Bin : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        Hout : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
    );
END bin2hex;
ARCHITECTURE Behavioral OF bin2hex IS
BEGIN
    PROCESS (Bin)
    BEGIN
        CASE(Bin) IS
            WHEN "0000" => Hout <= "1000000"; --0--
            WHEN "0001" => Hout <= "1111001"; --1--
            WHEN "0010" => Hout <= "0100100"; --2--
            WHEN "0011" => Hout <= "0110000"; --3--
            WHEN "0100" => Hout <= "0011001"; --4--
            WHEN "0101" => Hout <= "0010010"; --5--
            WHEN "0110" => Hout <= "0000010"; --6--
            WHEN "0111" => Hout <= "1111000"; --7--
            WHEN "1000" => Hout <= "0000000"; --8--
            WHEN "1001" => Hout <= "0010000"; --9--
            WHEN "1010" => Hout <= "0001000"; --a--
            WHEN "1011" => Hout <= "0000011"; --b--
            WHEN "1100" => Hout <= "1000110"; --c--
            WHEN "1101" => Hout <= "0100001"; --d--
            WHEN "1110" => Hout <= "0000110"; --e--
            WHEN OTHERS => Hout <= "0001110";
        END CASE;
    END PROCESS;
END Behavioral;
```