# Mechtron 3TB4: Embedded Systems Design II
# Tutorial Lab 2

Building a Hardware Interface using an FPGA

**Note:** Connect to the VPN prior to starting Quartus Prime, otherwise compilation will be unavailable.

## Goals

In this tutorial lab session, you will create a project to implement a stopwatch using the FPGA on the DE1-SOC board. Some modules developed here can be used in Lab 2 (some may require modifications). The objective of Lab 2 is to create a reaction time contest device.

To implement a stopwatch, you will design a clock divider, a counter, a hexidecimal (or binary) to binary-coded decimal (BCD) translator, a seven-segment display handler, and a top-level module interfacing the submodules. You will describe your circuits using the Verilog Hardware Description Language (HDL) and implement them on the FPGA on the DE1-SoC board. This lab requires knowledge of basic Verilog.

## System Description

The system we are building is shown in Figure 1. The diagram is intended only for guidance.

You will use the clock divider to slow down the signal from the 50 MHz oscillator crystal on the DE1-SoC to 1 KHz. The slower signal can then be used as the input for a counter so the counter can increment every 1 millisecond. The count from the counter module is fed into the hex(or binary)-to-BCD converter. Once the count is translated to BCD, it is fed into the seven-segment display handler to output the counter value on the seven-segment HEX LEDs on the DE1-SoC board.

Your system has three user buttons: reset, start/resume, stop. The system should work in the following way:

- After your design is downloaded to the FPGA, or after the reset button is pressed, the HEX LEDs on the DE1-SoC display 0s.
- When the start/resume button is pressed, the stopwatch starts to run, and the HEX LEDs on the DE1-SoC board display time in milliseconds.
- Pressing the stop button will halt the stopwatch.
- Pressing the start/resume button after the stopwatch is halted will resume the stopwatch.

### Clock Divider

The clock divider is a special circuit that can limit the rate of a given clock signal. The DE1-SoC board carries 50 MHz clocks that you will be using. However, this is too fast for our application. You will create a clock divider to slow down the signal from 50 MHz to 1 KHz. The slower signal is then fed into a counter so the counter can increment every millisecond.

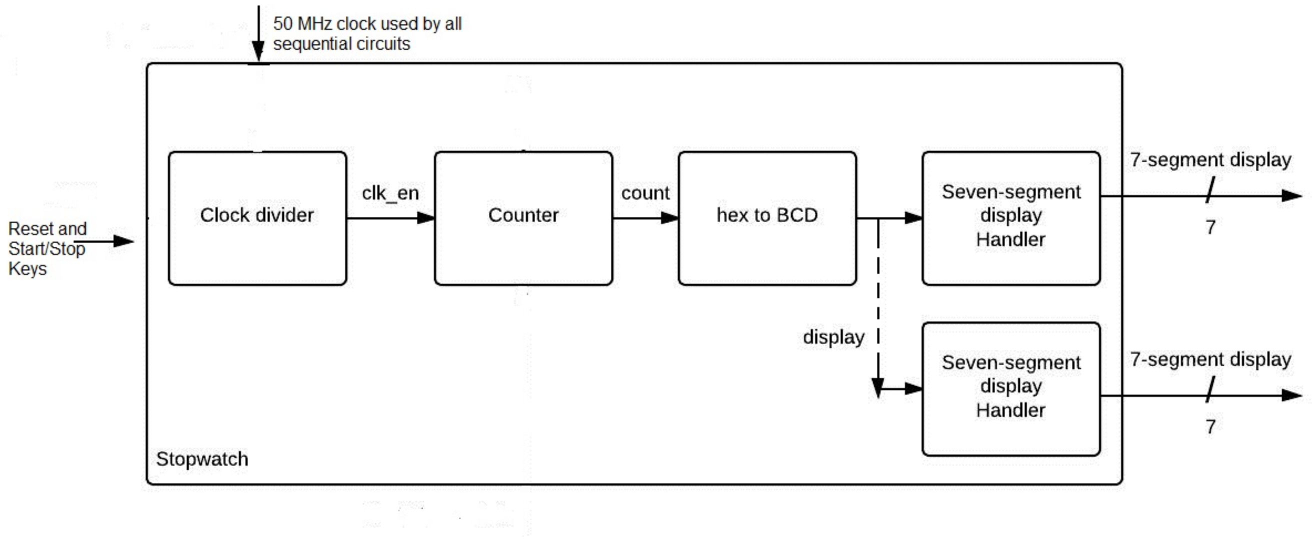
50 MHz clock used by all sequential circuits
Reset and Start/Stop Keys
Clock divider
clk_en
Counter
count
hex to BCD
display
Seven-segment display Handler
Seven-segment display Handler
7-segment display
7
7-segment display
7
Stopwatch

Figure 1: System Block Diagram

## Binary to BCD converter

BCD (Binary-Coded Decimal) is a digital encoding method for decimal numbers, in which each decimal digit is represented by a corresponding binary sequence. For example, a binary number 1100, which is 12 in decimal, is 0001 0010 when represented in BCD.

Converting a binary or a decimal number to its BCD format makes it easy to display the decimal number on the seven-segment displays.

One algorithm to convert a binary representation of a decimal number to its corresponding BCD representation is to use a "Shift-Add-3" algorithm. The algorithm works in the following way:

1. According to the maximum value of the number to be represented, declare a register of appropriate size for the BCD representation. The register for the BCD representation should have sufficient tetrads (each tetrad corresponds to one decimal digit). Initiate the BCD register to 0.

   For example: The largest value for an 8 bit unsigned binary number is 255, so the register for its BCD representation should have 3 tetrads (12 bits).

2. Check the value of each BCD tetrad. If a tetrad's value is greater than or equal to 5, then add 3 to that tetrad.

3. Shift the BCD register one bit left. Shift the binary representation one bit left into the BCD register, i.e., shift the most significant bit of the binary number into the BCD register as the least significant bit.

4. Repeat step 2 and step 3 until all of the binary bits are shifted into the BCD register. (NOTE that after the last bit shift, step 2 is performed a final time).

You may use the following code skeleton to complete your module:

```
module hex_to_bcd_converter(input wire clk, reset, input wire [19:0] hex_number,
        output  [3:0] bcd_digit_0,bcd_digit_1,bcd_digit_2,bcd_digit_3, bcd_digit_4,
        bcd_digit_5);
                // DE1-SoC has 6 7_seg_LEDs, 20 bits can represent decimal 999999.
                //This module is designed to convert a 20 bit binary representation
                //to BCD

                integer i, k;
                wire [19:0] hex_number1; // the last 20 bits of input hex_number
                reg [3:0] bcd_digit [5:0];  //simplifies program
                assign hex_number1=hex_number[19:0];

                assign  bcd_digit_0=bcd_digit[0];
                assign  bcd_digit_1=bcd_digit[1];
                assign  bcd_digit_2=bcd_digit[2];
                assign  bcd_digit_3=bcd_digit[3];
                assign  bcd_digit_4=bcd_digit[4];
                assign  bcd_digit_5=bcd_digit[5];

                always @ (*)
                        begin
                                //set all 6 digits to 0
                                /* fill your code here */

                                //shift 20 times
                                for (i=19; i>=0; i=i-1)
                                begin
                                        //check all 6 BCD tetrads, if >=5 then add 3
                                        for (k=5; k>=0; k=k-1)
                                        begin
                                                /* fill your code here */
                                        end

                                        //shift one bit of BIN/HEX left
                                        //for the first 5 tetrads
                                        for (k=5; k>=1; k=k-1)
                                        begin
                                                bcd_digit[k]=bcd_digit[k] << 1;
                                                bcd_digit[k][0]=bcd_digit[k-1][3];
                                        end

                                        //shift one bit of BIN/HEX left, for the last tetrad
                                        /* fill your code here */
                                end //end for loop

                        end  //end of always.
endmodule
```

# Activities

## Pre-tutorial [10]

The following activities must be completed and shown to one of the TAs at the start of your Lab 2 Tutorial session. Prepare these as a group.

1. Write Verilog code that implements the clock divider circuit and try to simulate it to verify its functionality. You may use the following skeleton code for your module (you do not HAVE to use this code). Remember that in a full clock cycle, a clock is 1 for half of the cycle and 0 for the other half, leading to the two cases involving `factor` below.

   ```
   module clock_divider (input Clock, Reset_n, output reg clk_ms);
           parameter factor=10; //50000;   // 32'h000061a7;

           reg [31:0] countQ;

           always @ (posedge Clock, negedge Reset_n)
           begin
                   if (!Reset_n) begin
                                   /* fill in your code here */
                           end
                   else
                           begin
                                   if (countQ<factor/2)
                                           begin
                                           /* fill in your code here */
                                           end
                                   else if (countQ<factor)
                                           begin
                                           /* fill in your code here */
                                           end
                                   else    //countQ==factor
                                           begin
                                           /* fill in your code here */
                                           end

                           end
           end
   endmodule
   ```

   To simulate a clock divider that slows the clock signal from 50 MHz to 1KHz will take a long time. You can simulate your clock divider using a much faster output clock. You may write this module as a parameterized module to allow the slow-down rate to be changed easily and for easy simulation.

2. Write Verilog counter logic to reset the counter, to stop (enable) the counter without resetting the current value, and to resume counting after being paused. Try to simulate the module to make sure it functions as expected.

   ```
   module counter(input clk, input reset_n,  start_n, stop_n, , output reg [19:0] ms_count);
   /* fill in your code here */
   endmodule
   ```

3. Create a top level module that takes the 50MHz clock and three of the four KEYs of the DE1-SoC board as inputs, and outputs signals to the six HEX LED displays of the DE1-SoC board. Use KEY[0] for the "Reset_n" signal, KEY[1] for the "Start_n" signal, and KEY[2] for the "Stop_n" signal.

   Please note that pressing any of these KEYs results in a logic low signal. You may use the following module declaration and code skeleton for the top level module:

```
module lab2tut (input CLOCK_50, input [2:0] KEY, output [6:0] HEX0, HEX1, HEX2,
                HEX3, HEX4, HEX5);
        wire clk_en;
        wire [19:0] ms;
        wire [3:0] digit0,digit1,digit2,digit3, digit4,digit5;

        clock_divider #(.factor(50000)) (.Clock(CLOCK_50), .Reset_n(KEY[0]), .Pulse_ms(clk_en));
        counter(.clk(clk_en), .reset_n(KEY[0]), .start_n(KEY[1]),  .stop_n(KEY[2]),  .ms_count(ms));
        hex_to_bcd_converter(CLOCK_50, KEY[0], ms, digit0,digit1,digit2,digit3, digit4,digit5);

        seven_seg_decoder  decoder0(digit0, HEX0);

        /* fill in your code here */

endmodule
```

## In the lab [20]

1. Simulate your clock divider and your counter. Show the simulation results to one of your TAs.

2. Create a new Quartus project and name this project "`lab2tut`". Use your code from the pre-lab and Lab1. Create a top level module named `lab2tut` that binds all of the components together. Write code for all other necessary modules. Import pin assignments from the `DE1-SoC.qsf` file provided on the course web page. Before compiling, make sure that all unused pins are reserved as **Input tri-stated**. This option is available under `Assignments | Device > Device and Pin Options > Unused Pins`. Compile your design and debug your code if necessary.

3. Download your design to the FPGA and test the behavior of your circuit. Demonstrate your working project to one of the TAs.