

Department of Electrical and Computer Engineering  
ENEL453: Digital System Design

## Lab 2: Digital Clock

### Pre-lab Exercises

#### 1 Overview

You are required to complete the pre-lab exercises **individually** prior to the scheduled lab period. These will help you complete the actual lab work faster, and provide the necessary background knowledge to make any required changes to the design later on.

Yellow boxes in this document will provide you with details as to what is required in the pre-lab hand in. You can simply hand-write your pre-lab answers. It is recommended, however, that you use a computer (Notepad, Notepad++, or the Xilinx tools) to create and edit the VHDL files.

**This lab is worth 8% of your term grade and broken up as:**

- Pre-lab: 2%
- In-lab Demo: 3%
- Lab Report: 3%

#### 2 Goals of Lab 2

The goals of this lab is to design a circuit displaying minutes and seconds on the FPGAs four 7-segment displays. The input clock frequency coming from the FPGA is 25MHz. To display seconds, we need to downscale to 1 Hz (1/sec). To display ten seconds, we have to downscale to 0.1Hz etc. Therefore, we have to divide the input frequency 25MHz by using divide-by-25 counter as the divider. It shall be followed by divide-by-10 counter, and another one, and so on.

The sub-goals are as follows:

1. Create a clock divider circuit capable of producing clock-like signals of different frequencies.
2. Create a decoder to handle translation of regular numbers to seven-segment display codes.
3. Create a state machine to switch between four of the seven-segment displays (to display seconds, tens of seconds, minutes and tens of minutes).
4. Connect all three blocks together and display a digital clock on the Basys2 board.

In the Pre-lab, we will address the first 3 points in the pre-lab, leaving the last one for the laboratory.

### 3 Basys2 Board Areas for this Lab

For Lab 2, we will expand on the pins and peripherals that will now be connected to the design. Figure 1 shows a photo of a Basys2 that is running Lab 2.

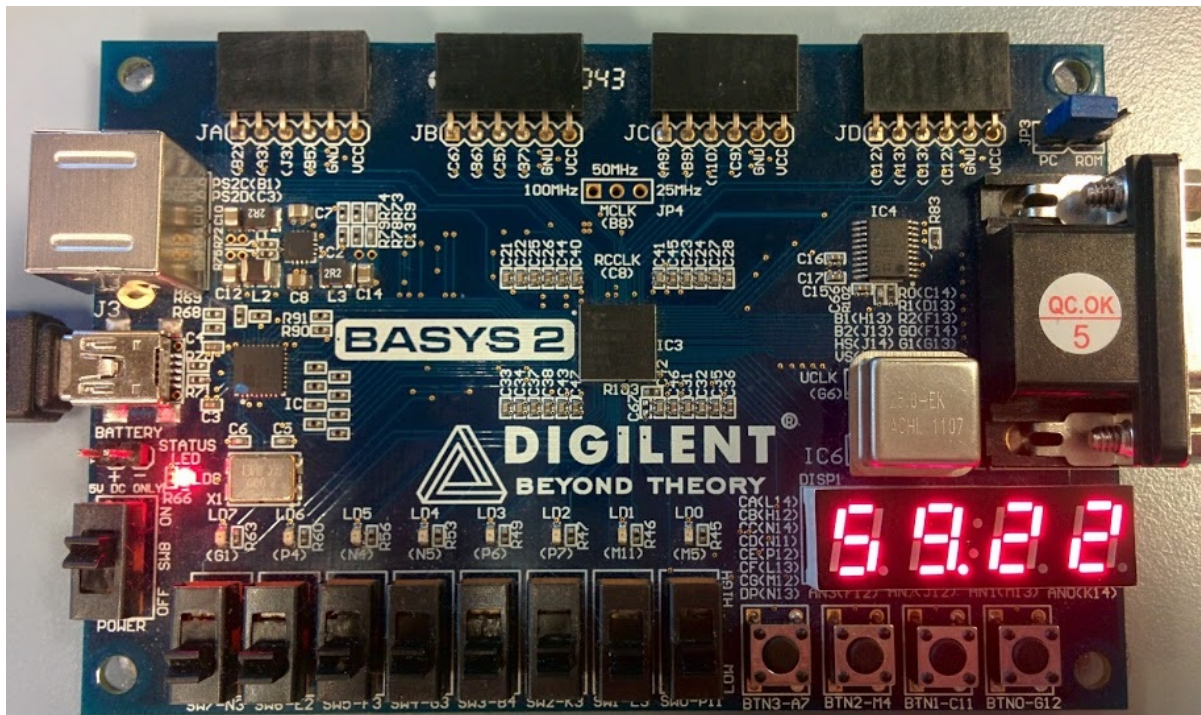


Figure 1: Basys2 Board with a digital clock shown in seven-segment displays.

Refer to the Basys2 manual online ([https://www.digilentinc.com/data/products/basys2/basys2\\_rm.pdf](https://www.digilentinc.com/data/products/basys2/basys2_rm.pdf)) and obtain pin numbers (such as "B8", or "M11"), for the following items:

- Clock (use the one that can change between 25MHz and 100MHz)
- A push button of your choice to use for reset
- All anodes belonging to the seven-segment displays (these are labeled as "AN" in the Basys2 manual). There are 4 in total.
- All cathodes belonging to the seven-segment displays (these are labeled as "CA" in the manual). There are 7 in total.
- The decimal point for the seven-segment display, this is labeled as "DP" in the manual. There is just one.
- A slide switch of your choice to use as a pause/enable input into the clock.



#### Begin your pre-lab:

Record your answers (page 11 can also be useful as it shows the format for pin names). These values will be used when we need to specify how to connect our design in the .ucf file.

## 4 A Downcounter in VHDL

A downcounter can be implemented in many ways in VHDL. It can be built out of DFFs, utilizing a **structural** description. We will use a **behavioural** description. Figure 2 shows the block diagram of the downcounter. Consult the listing below for [functional] code implementing a downcounter provided below (this code is also available amongst your pre-lab documents as `downcounter.vhd`).

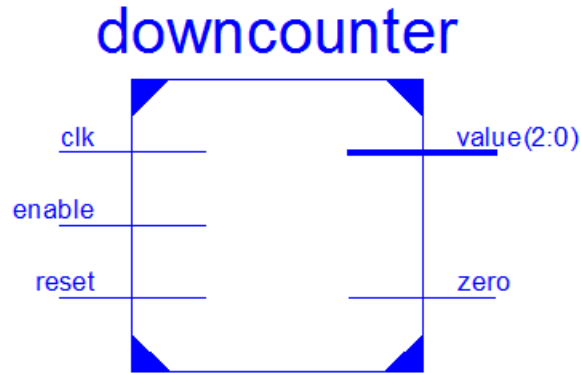


Figure 2: **Downcounter block diagram:** This shows all inputs (left-hand side) and outputs (right-hand side) of the entity.

```

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

entity downcounter is
    Generic (
        period: integer:= 4; -- input clock period
        WIDTH: integer:= 3); -- its binary length
    Port (
        clk : in STD_LOGIC;
        reset : in STD_LOGIC;
        enable : in STD_LOGIC;
        zero : out STD_LOGIC;
        value: out STD_LOGIC_VECTOR(WIDTH-1 downto 0));
end downcounter;

architecture Behavioral of downcounter is
    signal current_count: std_logic_vector(WIDTH-1 downto 0);
    signal i_zero: std_logic;

    -- Convert the max counter to a logic vector (this is done during synthesis)
    constant max_count: std_logic_vector(WIDTH-1 downto 0) :=
        std_logic_vector(to_unsigned(period, WIDTH));
    -- Create a logic vector of proper length filled with zeros (also done during synthesis)
    constant zeros: std_logic_vector(WIDTH-1 downto 0) := (others => '0');

begin

```

```

count: process(clk,reset) begin
  if (rising_edge(clk)) then
    if (reset = '1') then    -- Synchronous reset
      current_count <= max_count;
      i_zero <= '0';
    elsif (enable = '1') then -- When counter is enabled
      if (current_count = zeros) then
        current_count <= max_count;
        i_zero <= '1';
      else
        current_count <= current_count - '1';
        i_zero <= '0';
      end if;
    else
      i_zero <= '0';
    end if;
  end if;
end process;

-- Connect internal signals to output
value <= current_count;
zero <= i_zero; -- Connect internal signals to output

end Behavioral;

```

Note that this design uses a synchronous **reset**. On reset, the maximum value is latched into the counter. When the chip is enabled, the counter value is decremented and checked against 0. If there is a match (or, in other words, the counter has reached 0), the zero signal pulses high and the counter resets.

A few new features are introduced in thi code:

- **Generics:** entity defines two generics - **WIDTH** and **period** - outside of the normal port list. Generics are used to make a design just that, generic. This means it can be adapted for various purposes by modifying the generic variables when using the component without changing the already designed functionality. By default, the above downcounter will count down from 4 (max number of bits is set to 3, and 4 is binary 100, so it fits). However, we could just as easily make it count down from 1000 or 25000.
- **Constants:** you will notice that some signals are defined as **constants** (**max\_count** and **zeros** in the code above). This provides the synthesis tool with the information that these signals will not change, allowing the design to be optimized. This also keeps the synthesis tool from getting upset about the logic vector never being assigned a value within the process.

The testbench output of this circuit, with `period=5`, and `WIDTH=3`, is shown in Figure 3. Note that the output produces a pulse when count reaches 0 (signal `zero`); it is not a wide pulse, certainly nothing close to a 50% duty cycle. Theoretically, it shall only last a single clock cycle. This means that if we were to connect these blocks in a cascade, we could use the output “`zero`” signal to enable another downcounter. Note also that the downcounter resets itself to the value “`period`” after it reaches 0.

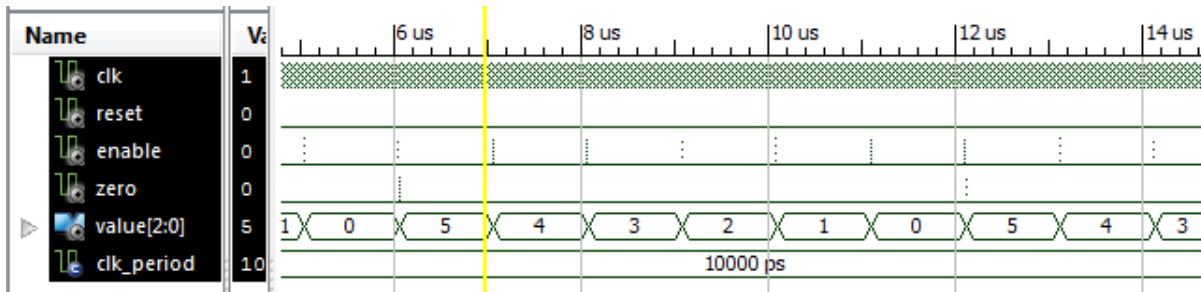


Figure 3: **Downcounter waveform:** Shown here are the outputs of a generic-based downcounter design. Then value of `period` was set to 5, and `WIDTH` to 3.

## 5 A Cascade of Downcounters

You will cascade downcounters in the lab to produce a variety of signal frequencies to drive the clock. For example, Figure 4 shows the beginning of the cascade. The input clock is 25MHz (25000000 cycles per second). The first block is the `kiloHzClock`, sporting a period of 25000 (which fits within 15 bits); for every 25000 ticks of the real oscillator, this clock ticks once, resulting in 1000 ticks/second (1KHz). Similarly, the `hundredHzClock` divides the `kiloHzClock` by a factor of 10, resulting in 100 ticks/second. Likewise, `tenHzClock` divides the `hundredHzClock` by another factor of 10.

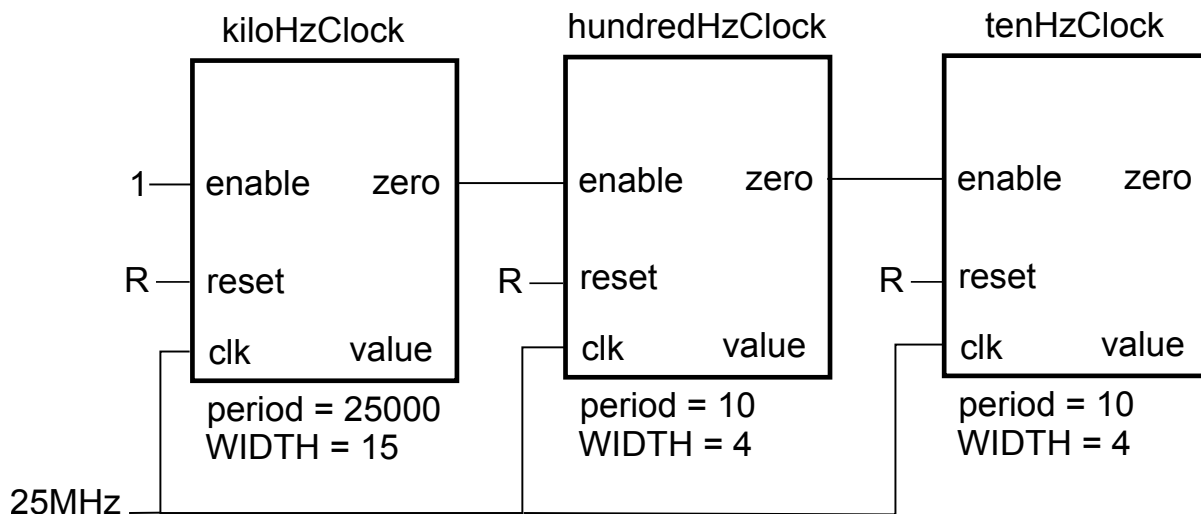


Figure 4: **A Clock Divider Cascade:** This shows all inputs (left-hand side) and outputs (right-hand side) of the entity.

### Continue the cascade:

In your prelab, continue connecting blocks to the cascade to produce signals of following frequencies. For each block determine the period and the width, and make sure you label the connections from one block to the other.

- Single seconds (1Hz)
- Tens of seconds
- Single minutes
- Tens of minutes

The diagram you have created will aid you in forming the VHDL structure of the clock divider circuit. In a hardware design with a large number of connections, things can get quite hectic quite quickly, and so it is very useful to have a block-based diagram of the design available when writing HDL code. `clock_divider.vhd` begins to describe this circuit.

### Create a clock divider:

Finish writing the code to describe a component that will produce all of the described signals as outputs, using `clock_divider.vhd` given as part of your pre-lab files. Remember to add relevant signal buses as outputs to the **entity** description.

Since this circuit only has two inputs, which are fairly normal (**clk** and **reset**), we will test the component in lab. You are welcome to create a testbench and run it either

way, but remember to change the divider of the first component to a smaller value, as 25000 takes quite a while to count.

## 6 Seven-Segment Decoder

There are 4 seven-segment display digits on your Basys2 board. Each digit is composed of 7 segments that can light up to form letters or numbers, as well as a decimal point. Figure 5 shows how the segments of one digit are named.

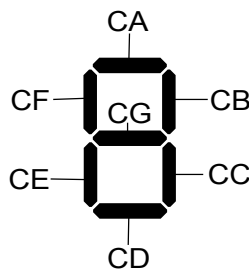


Figure 5: **Cathodes of a seven-segment display:** The segments are just LEDs that are actually active low. However, for convenience, we usually just invert the bits before displaying, rather than programming them inverted to start with.

For instance, in order to display a "1" on a digit, we have to assign the signals to be "01100000". Table 1 shows a few of these decoded values corresponding to a specific digit as an example.

In order to more-or-less automatically convert regular numbers into a code that corresponds to the appropriate segments on the display, we will use a `case` statement. The code in `sevensegment.vhd` has started to describe the decoding process.



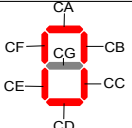
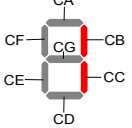
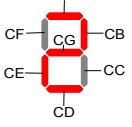
### Create a decoder circuit:

Complete the description to decode all numerical digits (0-9) into seven-segment display cathode values within `sevensegment.vhd`.

In ENEL453, and digital design in general, incremental testing is generally promoted; this means we want to test all components as they are being developed rather than leaving everything until the end. Since you have just created a small component, we now want to make sure it works as expected.



Table 1: Table showing decoding of numbers to seven-segment display cathode codes.

| Number | Segments                                                                          | CA | CB | CC | CD | CE | CF | CG |
|--------|-----------------------------------------------------------------------------------|----|----|----|----|----|----|----|
| "0"    |  | 1  | 1  | 1  | 1  | 1  | 1  | 0  |
| "1"    |  | 0  | 1  | 1  | 0  | 0  | 0  | 0  |
| "2"    |  | 1  | 1  | 0  | 1  | 1  | 0  | 1  |



### Write a testbench for the decoder circuit:

The code in `tb_sevensegment.vhd` shows most of the testbench. Complete this testbench description to test that the output is as expected. **Remember to make sure all of your inputs are connected to something.** You are not required to test your module before the lab, although you can, of course, do so if you wish.

## 7 Seven-Segment Digit Selector

Aside from the cathode lines, each seven segment display also has an anode one, typically denoted as **AN**. The segments light up **only** when the anode signal to the digit is 0 (active low). The 4 digits on the Basys2 board share the same cathode (CA, CB, CC and so on), but have different anode (AN) lines. Thus, one digit can be selected at a time, by setting an active low signal on its select line. Therefore, a digit selector is needed. Such selector shall generate AN signals that enable only one of four digits at a time (represented by AN values 1110,1101,1011,0111) or none (1111).

For this part of the pre-lab you will be completing the logic in a circuit responsible for producing these AN signals. Figure 6 shows the intended functionality of this circuit. Note that signal `output` only has a single bit on at any given time. What we are designing here is actually an **Overbeck counter**, or a **4-register one-hot counter**.

We will later invert the bits from `output` to produce the AN signals for our seven-segment display circuit. The signal names here should give something away - this design will rely on using DFFs to produce the proper output. Some details about the signals:

- **clk**: Common clock of the entire circuit, 25MHz in our case.

- **switch:** A clock-like signal that drives the outputs in this design. It essentially tells the component that the **output** bits need to shift, or that the next **AN** line should be chosen.
- **reset:** An asynchronous active-high signal that resets the output. This signal is common throughout the entire design.
- **output:** Signals that drive the anode (**AN**) signals of the seven-segment display digits.
- **d:** Inputs into the DFFs.
- **q:** Outputs of the DFFs.

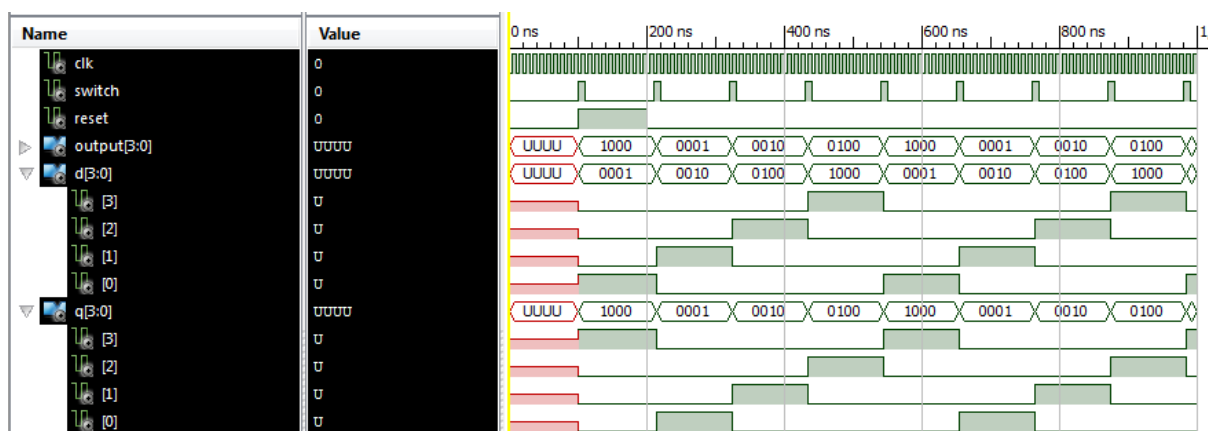


Figure 6: **Intended functionality of the selector circuit:** A single output is on at any given time. There exists a reset that puts the circuit into a pre-defined state.



### Write the logic for the selector:

File `sevenssegment_selector.vhd` contains some of the description for this component. It is your job to complete the file (in particular assignments to **d** and **q**) in order to achieve the intended functionality (shown in Figure 6). All necessary ports are already included in the **entity** description.

You will, of course, need to verify the functionality of this circuit prior to connecting it to the rest of the design.



### Finish the selector circuit's testbench:

File `tb_selector.vhd` contains testbench code with one process missing. This process needs to describe the operation of the **switch** signal. **You need to complete this process.** Refer to Figure 6 for an idea of what this signal must look like, but you do not need to replicate its frequency exactly.

## 8 Top-level diagram

The last part of this pre-lab involves something called a top-level diagram. This is an important part of digital design as it allows you to plan all internal component connections before you get to writing VHDL.

A starting point to your top-level diagram is shown in Figure 7. The signals on the outside of the large box are the top-chip inputs (left-hand side) and outputs (right-hand side); do not alter these. Within the box are the individual components on which you've worked during the pre-lab exercises. The ports (wires) for each component are determined by its entity description. For example, a downcounter entity (see the beginning of this pre-lab), would contain `clk`, `reset`, `enable` as inputs, and `zero` and `value` as outputs. Introduce and name any internal signal (connections between block internally) or logic gates you need.

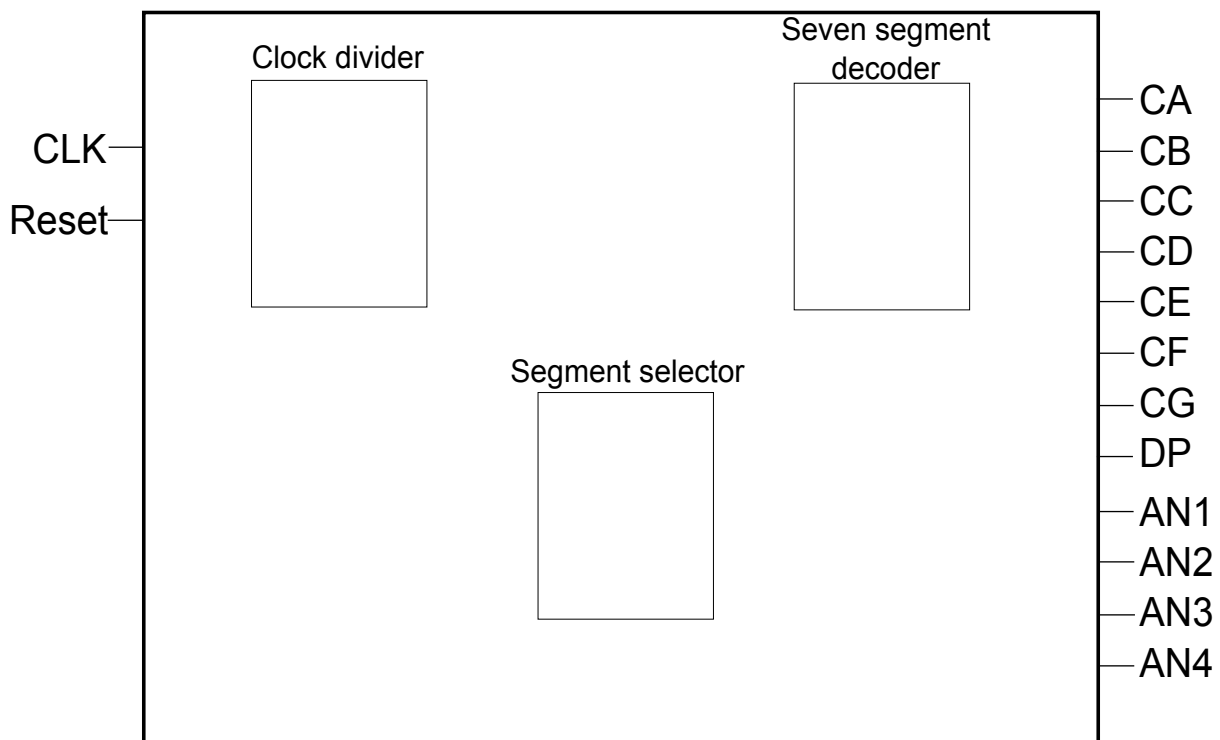


Figure 7: **Top-level diagram:** use this draft to finish the connections.



### Draw the top-level diagram:

On a sheet of paper start your top-level diagram. The outputs and inputs of the top-level should be as shown here. Within the large block, add the individual components (same as shown in Figure 7) and write in their inputs/outputs. Connect the blocks together; some hints on these connections:

- All blocks share the same `clk` and `reset`.
- The clock divider has 4 different output values (these are `STD_LOGIC_VECTORs` - one corresponding to each of single seconds, tens of seconds, single minutes and tens of minutes. You will need to use a Multiplexor (mux) to connect these to the `data` input of the seven-segment decoder block.
- `AN4` enables the right-most digit on the seven-segment display (single seconds), `AN3` enables the tens of seconds digit, and so on.
- Remember that we need the inverted versions of `CA-CG` and `AN1-AN4`.

We will double-check your diagram prior to the lab to ensure that everything makes sense.



### You are finished!

Make sure you bring the pre-lab answers/code to your section. You do not need to print the code, simply show the digital file on your computer. A TA will check this off for completion marks. **It would be beneficial for you to make yourself familiar with the lab instructions handout.**