ECE4514: A Rather Considerate Timer

This assignment is to be completed individually

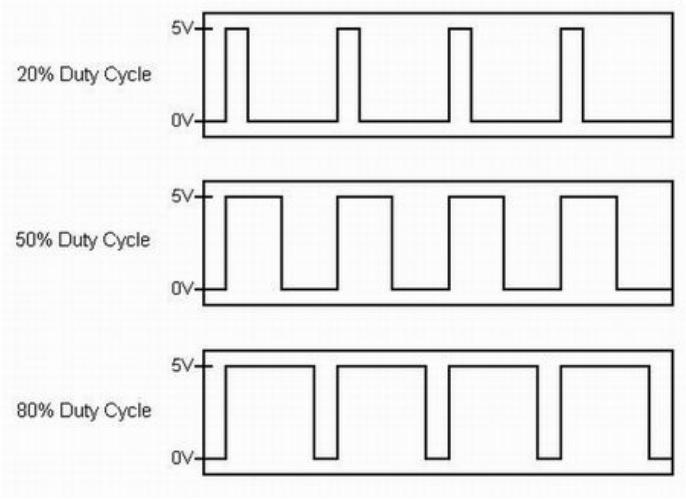
The purpose of this exercise is to become acquainted with:

- the ECE4514 development environment,
- pulse-width modulation operations,
- Alint style analyzer, and
- Verilog synthesis.

In this assignment, you are to construct a *considerate* timer, that measures elapsed time in 1/10ths of a second, that has a dimmable display for not disturbing those who wish to sleep. The right two HEX displays are to present the light intensity value (from 0=off, to 15=full on). These two digits are not dimmable. You are to use a *pulse-width modulator* signal to control the intensity of the four digits displaying the elapsed time through two push-buttons -- one brightens, one dims. The displays should brighten/dim smoothly through 16 different light intensity levels. When the brighten/dim buttons are held down, the display should brighten/dim at a rate of one level per ¼ to ½ a second. You cannot brighten above Level 15, or dim below Level 0. Both the time display and the intensity display should use leading-zero blanking. The *toggle* button alternates between two state: a *freeze* state (holds the current time) and *run* state (the timer is running). Below is a depiction of the display at Level 3 intensity. Time is in base-10.

A *Pulse Width Modulator* (PWM) is a common circuit used in everything from engine controls to light dimmers. A PWM produces a continuous stream of pulses at a fixed frequency like those shown in the figure below. The duty cycle, or *on* time, is defined as the ratio of the pulse width to the period, and is controlled with a digital command. The brightness of an LED bank can be controlled by the PWM duty cycle. You are to construct a synthesizable Verilog model of a pulse-width modulator.

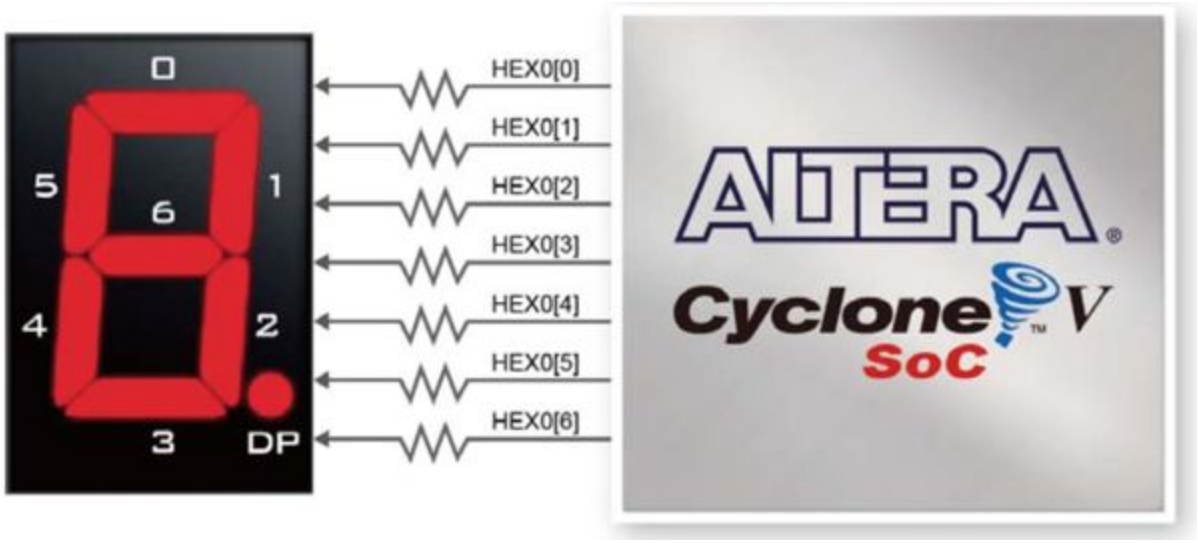
%PWM duty cycle = (command / N) * 100)



The frequency of your PWM waveform should be sufficiently slow so that the LEDs can respond, yet sufficiently fast so that the human observer cannot see flickering. A frequency selected in the range between 100 Hz and 1000 Hz should be adequate. Experiment. Your module is to be called: *nicetimer*. This module must have *exactly* the following I/O signals:

NAME	**TYPE**	**DESCRIPTION**
clock_50	input	50 MHz continuous clock
reset_n	input	System reset, active low, left-most push button. Resets time to zero, and intensity to 15.
toggle_n	input	Toggles between pausing the timer, and letting it run. Active low.
up_n	input	Increases display intensity when pressed, right-most push button, active low
down_n	input	Decreases display intensity when pressed, to the left of up_n, active low, right-most push button.
hex5[6:0]	output	Hex display output - time MSB segments are on when low
hex4[6:0]	output	Hex display output
hex3[6:0]	output	Hex display output
hex2[6:0]	output	Hex display output - time LSB
hex1[6:0]	output	Hex display output - intensity MSB
hex0[6:0]	output	Hex display output - intensity LSB

The 7-segment outputs (hex0 - hex5) are *active low* ('0' turns a segment on).



Simulate the design as needed using ModelSim. Use ALINT to style-check your design. You can submit your code to the web service by following these instructions:

1. Compose an email to vtevaluator@autograde.ece.vt.edu making sure your FROM address is your PID@vt.edu address. The subject should be:
   style_check_only
2. The body of your email should consist of your honor pledge:
   I have neither given nor received unauthorized assistance on this assignment.
3. Attach your Verilog files. Send. Await the response.

All identified ALINT warnings and errors must be resolved, else a penalty of 20% will be levied.

Your design must abide by the following constraints:

- All state is synchronously reset
- The design should be fully synchronous using one clock domain (clock_50)
- Usage of the ARM processor is not allowed
- You cannot have any timing violations

There are no area or power constraints on this assignment.

NOTE: Create your project within the Linux file space in your virtual machine. DO NOT create your project under your VM shared folder (/media/sf_*). DO NOT copy a project created in your shared folder to your Linux file space.

Make this model operational on your DE1-SOC board. Use the Quartus project file posted on the Canvas Assignments page.

GRADING RUBRIC

Functionality: **70%**

- Does your design address all of the design criteria?
- Does the design follow the listed constraints, such as being fully synchronous, synchronous reset?
- Are your modules and signals all named correctly?

Alint Style Warnings: **20%**

- Have ALL ALINT warnings been resolved

Follow Submission Instructions / Report: **10%**

- Did you create your project within the Linux file space, and not under a shared host folder?
- Did you submit a .qar, .sof, and .pdf to canvas and are they named correctly
- Have you included a report with the requested information

SUBMISSION INSTRUCTIONS

1. Create a Quartus archive called exactly `nicetimer_<lastname>.qar`, where <lastname> is replaced with your last name in lowercase letters, and submit it to the Canvas Assignments page. Note: your project MUST have been created within the virtual machine's file system -- not a shared folder. Do not copy a Quartus project from your shared (host) folder to your virtual machine. This
2. Rename your SOF file as `nicetimer_hw_<lastname>.sof`, and submit it on Canvas.
3. Create a **brief and informal** report, 1/2 - 1 page in length, called exactly `nicetimer_<lastname>.pdf` which outlines the following:
    a. Does your design work? What works and what doesn't?
    b. Extra resources, libraries, etc. that are needed to compile and run your design? (Skip if not applicable)
    c. Short description of design decisions.