# Design Assignment #3 v20170517

(Deadline: 11:59PM PDT, Friday June 2, 2017)

Name (Last, First):

Student Id #:

## INSTRUCTIONS

**NOTE: This design assignment is to be done individually using EDA Playground. https://www.edaplayground.com. You should have already gone through the startup guide.**

**SUBMISSION PROCEDURE: You need to submit your solution online via Gradescope. https://www.gradescope.com/courses/4278 (code: MDZ4EM). The website will automatically grade your submission.**

DESIGN PROCEDURE: Note that for each Verilog design task a dassignX_X.v and optionally dassignX_X.tb are provided. The .v file contains the module you are to design. You are to fill in these modules for each design task.

*The XYZ_tb.v files are test benches that tests your design. The test benches, led_fsm_tb.v, code_reg_tb.v, and dassign3_tb.v will be very similar to the code used for the autograder. Your code needs to work with them without any modification to the test benches. You are of course welcome to create your own test benches to do your own testing.*

Note that these files have module names and interface signal names already defined. These MUST NOT be changed (deleted from or added to) in any way - doing so will in all likelihood result in a design that will not comply with the grading program and thus result in a zero score.

*Only submit the files provided on gradescope. Do not change the filenames since the autograder is looking for specific filenames.*

This assignment is intended to be more challenging than the previous two. Read the instruction and description carefully. The grading will be roughly 1/3 for led_fsm.v, 1/3 for code_reg.v, and 1/3 for dassign3.v. The led_fsm.v is intended for you to demonstrate the design of sequence generating FSM. The code_reg.v is intended for you to demonstrate the design of a datapath FSM. The final dassign3.v is showing that you can tie it all together and hence is the more challenging final portion.

File provided: *dassign3.v, led_fsm.v, code_reg.v, ascii_morse.txt, test benches for each.*

Problem #5 in Homework #3 introduced a design of an FSM that can light an LED with the Morse code for SOS. In this extended design assignment, the goal of is to implement a Morse Code encoder that lights an LED accordingly. The LED is represented by a signal LED_DRV.

In Morse Code, Dash ("-") is 3 continuous time units of beep and Dot (".") is 1 time unit of beep. Dots and Dashes are separated by 1 time unit. Space between characters can be 2 or 3 time units. A "space" between words is indicated by at least 7 consecutive time units of blank.

The figure below shows a block diagram of the design. The test bench will specify a 3-kbit memory that maps 8-bit ASCII symbols (2 hexadecimal digits) into 12-bit outputs. A file that contains the mapping of the desired set of ASCII symbols to Morse Code is provided (ascii_morse.txt). The test bench will output to your code (dassign3.v) two inputs, *charcode_data*[7:0] and *charlen_data*[3:0]. The bus, *charcode_data*[7:0], is a sequence of 1's and 0's where a 1 represents a Dash and a 0 represents a Dot for a given character. The *charlen_data*[3:0] is a binary representation of the length of the sequence for that character. Note that *charlen_data*=4'b0001 represents a length of 1. For example,

| Character | Morse Code | charcode_data[7:0] | charlen_data[3:0] |
|---|---|---|---|
| A | .- | 8'b01XXXXXX | 4b'0010 |
| E | . | 8'b0XXXXXXX | 4b'0001 |
| I | .. | 8'b00XXXXXX | 4b'0010 |
| <space> | / | 8'bXXXXXXXX | 4'b0000 |

Note that the X (don't care) in the memory are loaded as 0's, hence the *charcode_data* can be the same for letters such as I and E if not for the differences in *charlen_data*. "Space" between words is a special character (8'h20 in ASCII) and is demarked by *charlen_data*=4'b0000.

First, write the Verilog code for the state machines described in Problem #5 (a) to implement the Dash and Dot symbols of Morse Code. This corresponds to the module ***led_fsm*** (LED_FSM). While not specified in Problem 5(a), to make your timing easier for the final step of the project you should assert the LED_DRV signal on the arc leaving the START state. That way you can have your output respond more quickly to an input symbol. Note that inputs *clock* and *reset* are not shown in the figure. Proper behavior of this FSM should result in no more than 1 blank cycle between DASH or DOT symbols for a given character. The test bench tests for this.

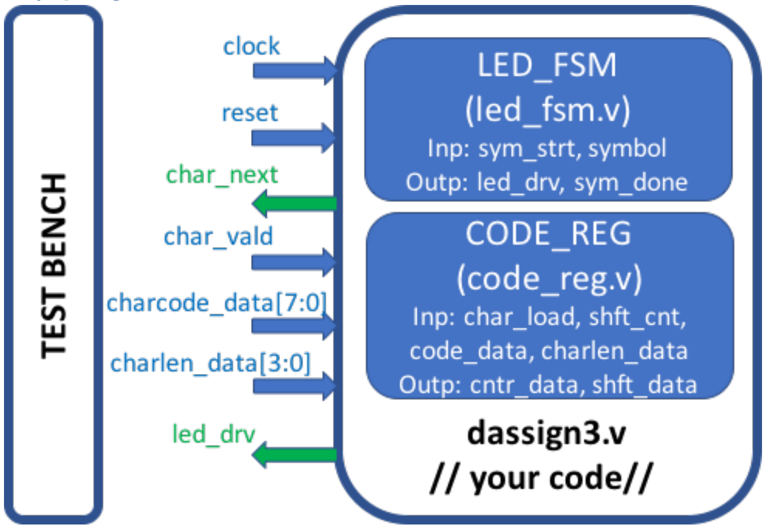
Next, implement a module ***code_reg*** (CODE_REG shown in the figure below) contains two blocks: (a) an 8-bit shift register that is holds *charcode_data*[7:0] and (b) a 4-bit counter that is loaded with the *charlen_data*[3:0]. Note that inputs *clock* and *reset* are not shown in the figure. Both blocks take a 1-bit input signal, *char_load*. The signal when asserted, loads the shift register with *charcode_data*, and loads the counter with *charlen_data*. The MSB of the shift register is also the output *shft_data* which indicates the symbol that needs to be decoded by the

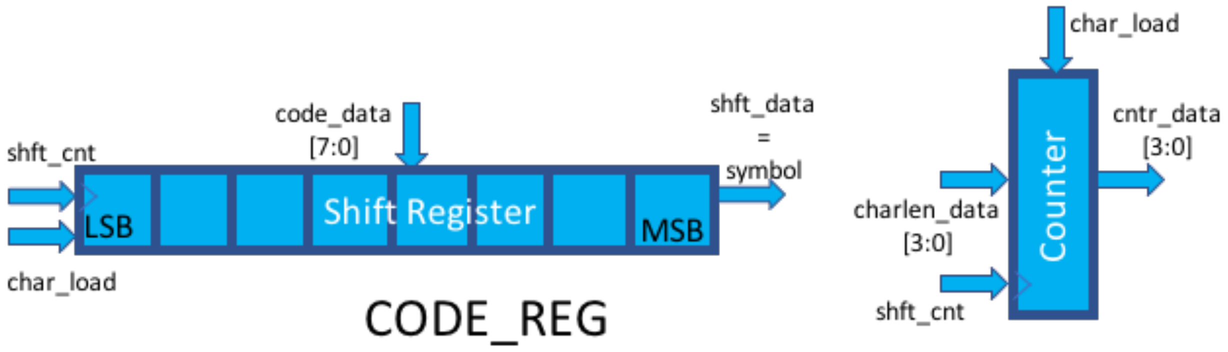
LED_FSM corresponding to Morse Dot or Dash. The 4-bit output, *cntr_data[3:0]*, is the value held in the counter. You should assume that *char_load* is asserted for only 1 cycle. When *char_load* is not asserted, the input signal, *shft_cnt*, is used to trigger both the shift register and counter. The shift register shifts left (toward the MSB) for every clock cycle that *shft_cnt* is asserted which changes the *shft_data* correspondingly. The counter counts down for each cycle that *shft_cnt* is asserted. Note that to handle a "space" symbol, you can choose to load the counter with 4'b0111 so that the counter can be used to count down.

Finally, the main module, ***dassign3*** (MORSE_ENCODER), takes inputs corresponding to the character it is to encode (*charcode_data* and *charlen_data*), an output that drives an LED (*led_drv*) to light up based on the code, and some handshaking signals. A clock signal, *clock*, and a reset signal, *reset*, are provided where *reset* is a synchronous reset that brings all FSMs into its starting/initial position. The input clock is the primary synchronizing signal. To handshake between the testbench and your module, a *char_vald*=1'b1 for 1 clock cycle accompanies the *charcode_data* and *charlen_data* to indicate the code to be valid for encoding so that you can load the registers and start your state machine. The MORSE_ENCODER should determine when to count down or shift the blocks in CODE_REG. When *cntr_data* reaches 4'b0000, the encoding of the character is considered complete. When the module completes a character, a *char_next* is asserted to 1'b1 to request the next character. The test bench receives this request and makes available the next character to be encoded (loaded) and the *char_vald* is asserted. The *char_next* remains asserted until a new *char_vald* arrives. Because "space" is a special character, *charlen_data*=4'b0000 needs to be identified and handled properly.

Proper behavior of the MORSE_ENCODER should result in consecutive characters to be transmitted. Each character has symbols (DASH or DOT) that are separated by not more than 1 blank cycle. Between characters, 2 or 3 blank cycles are permitted. Between words, an explicit "space" character is provided and should be at least 7 blank cycles and should not exceed 12 blank cycles.

The MORSE_ENCODER module is yours to design. You may choose to use another FSM to perform the sequencing for handshaking for the next character, handling each symbol, and handling a "space". You may choose to create another module to do this.





Since the design has three parts, you will write led_fsm.v, code_reg.v and dassign3.v. Three different test benches are provided to test each of these three parts. These test benches provided is also the test bench for the autograder so make sure that your design works without modifying the test benches. You can of course test other combinations.

## EE89 Design Task

Adapt this design so that it can be embedded in a Xilinx FPGA. The memory will need to be loaded. The dash and dot symbols will need to be generated using more cycles of clocks in order for the symbol flashing on the LED to be visible. You can use toggle switches [11:4] to set the *charcode_data* and toggle switches [3:0] to set the *charlen_data*. Pushing the center push-button will trigger the *char_vald*. Up push-button is the *reset*. The *char_next* should indicated on an LED (other than the one flashing the Morse code). You will need to synthesize the design using Vivado (Xilinx FPGA Design Environment) and compile the design for the Basys3 board.