Homework 11 Due: Monday-Tuesday 4/15-16 2pm in Class (turn two pieces of paper)
The purpose of this homework will cover chapter 7 of the online C Programming
resource, http://utaspr13.zyante.com/. Exercises are activities you should do using
Zyante and Keil uVision, but will not turn in for a grade. Assignments will be turned in.
You are allowed to work in groups of two students on homework, but each student must
turn in their own solutions.

Exercise 11.1: Complete Chapter 7: all sections. Please enter some feedback for the nice
folks at Zyante into the feedback box at the end of the chapter.

Assignment 11.1: Come fall we may have to charge EE319K students to use Zyante.
They want $40, we were thinking of suggesting each EE319K student pay $20.
Part a) Should we continue to use Zyante?
Part b) Do you think $20 per student is too much, too little or just right?
FYI, the new board will be a lot cheaper than the $50 we spent this semester.

Assignment 11.2: Submit one sheet (screenshot) showing your responses to the question
set 7.4 from the Zyante book.

<table>
<thead>
<tr>
<th>Num</th>
<th>Question</th>
<th>Your answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The preprocessor executes the beginning of a program before the main function is executed.</td>
<td>☐ True ☐ False</td>
</tr>
<tr>
<td>2</td>
<td>Using separate files helps to improve code readability.</td>
<td>☐ True ☐ False</td>
</tr>
<tr>
<td>3</td>
<td>Separate files are often used for structs to allow the code for structs to be developed separately from other program code.</td>
<td>☐ True ☐ False</td>
</tr>
<tr>
<td>4</td>
<td>Guarding header files is only useful for header files used for struct definitions.</td>
<td>☐ True ☐ False</td>
</tr>
<tr>
<td>5</td>
<td>The preprocessor directive &quot;#include &lt;conversions.h&gt;&quot; will search the current directory/folder for the file conversions.h.</td>
<td>☐ True ☐ False</td>
</tr>
</tbody>
</table>
Assignment 11.3: Go to http://users.ece.utexas.edu/~valvano/Volume1/ and download this grading engine. You can run the HW with the simulator or on your real board. http://users.ece.utexas.edu/~valvano/Volume1/HW_Moore.zip
If you simulate, then you will need to open a UART#1 window to see the score.

1) Unzip HW_Moore.zip
2) Launch HW_Moore.uvproj that starts Keil uVision
3) Build the object code Target->Rebuild all target files
4) Start the debugger and hit run (look in the UART#1 window)

My main program will call your subroutines multiple times, and will give your solution a performance score of 0 to 100. You should not modify my main program or my example data. When you have written your functions, you should run my main program, which will output the results to the UART.

You will write three functions and one FSM controller. My main program will call your functions multiple times, giving you a grade on these functions. After testing these functions, my program will jump to your FSM controller. During each loop of your FSM controller, you will call my grader function and points will be awarded. When my grader function is done testing your FSM controller it will output a performance score of 0 to 100. You should not modify my main program. When you have written the first function, you should run the system, which will output the results to the UART#1 window. The inputs are on Port V bits 3 and 1, and the outputs are on Port V bits 6, 5, and 4. For example, if you are in the Stop state the output is 0102. If the input is 102, then change to the Go state. The input/output values are shown in binary. The input/output testing will be performed by my grading software.

Part a) Your first function, called YourFSMInit, will initialize the FSM. This subroutine should initialize the I/O port. Bits 6, 5 and 4 are the output; bits 3 and 1 are the input. All accesses to I/O port registers must be friendly. Bits 7, 2 and 0 are not part of the FSM and should not be changed.

Part b) Your second function, called YourFSMOutput, will output to Port V bits 6, 5 and 4. All accesses to I/O port registers must be friendly. The value to output (0 to 7) is passed in as a parameter to your function. When writing to Port V you must write back the original values for the input pins, bits 3 and 1. E.g., if bit 3 is a 1, do not write a 0 to this bit. Bits 7, 2 and 0 are not part of the FSM and should not be changed. This is not a real I/O, so there is no bit-specific addressing allowed.
Part c) Your third function, called **YourFSMInput**, will input from Port V bits 3 and 1. All accesses to I/O port registers must be friendly. The input from Port V (bits 3 and 1) is returned as a 2-bit number.

- if PV3=0 and PV1=0, then return 0
- if PV3=0 and PV1=1, then return 1
- if PV3=1 and PV1=0, then return 2
- if PV3=1 and PV1=1, then return 3

Part d) Finally, you will write a controller for a Moore FSM. *Your controller should call your three functions.* All I/O accesses should be performed in a friendly manner. Your FSM controller does not return. However, my grader program will stop after the tests are complete. There are three parts to this FSM controller. Convert the Moore FSM graph to a linked data structure (a graph) and store it as a C structure in EEPROM. You will need to save a pointer to the current state in global memory. The initial state is **Stop**. The proper sequence is output, input, and go to next state. Place your FSM controller software in places labeled “put your FSM controller here”. Notice that the grader function must be called before you execute the output, input, next sequence. Your FSM controller will follow this sequence.

```c
void YourFSMController(void){
    YourFSMInit(); // your initialization
    while(1){
        MyGrader(); //do not move or remove this line
        // Part d) put your output-input-next engine here
        // the Grader will check outputs and make inputs happen
        // 1. output to PTV bits 6,4,3 (be friendly)
        // 2. input from PTV bits 3,1
        // 3. next
        // put your FSM controller here
    }
}
```

*Note that calling your functions in parts (a) (b) (c) will greatly reduce the amount of code you will need to write.*

Print one piece of paper that contains some of your HW_Moore C source code, your name, and the output results showing your score.