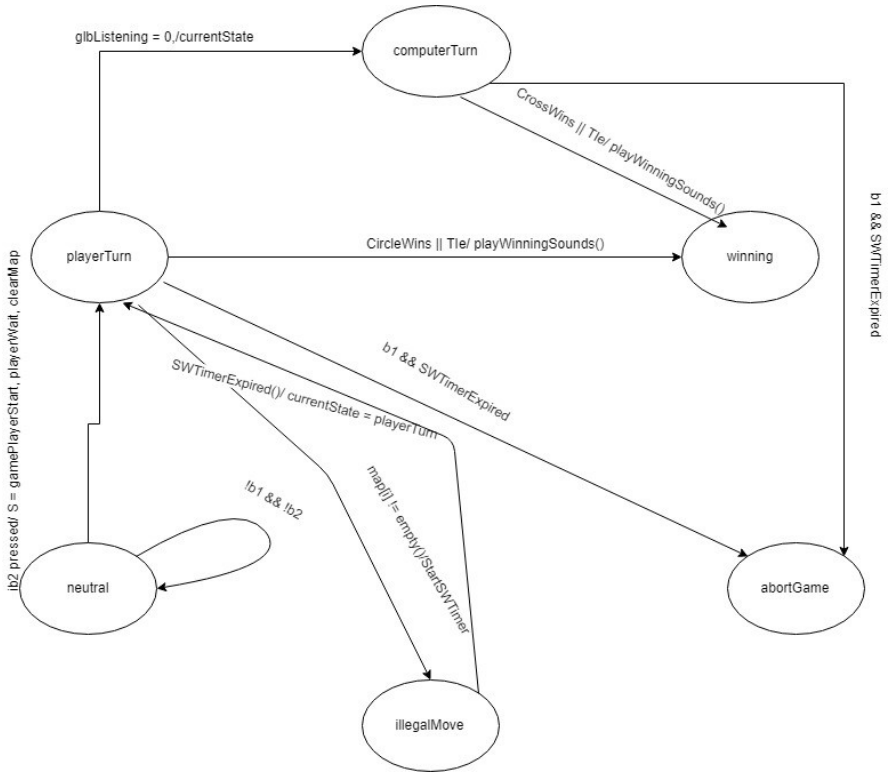
Section 1: Description

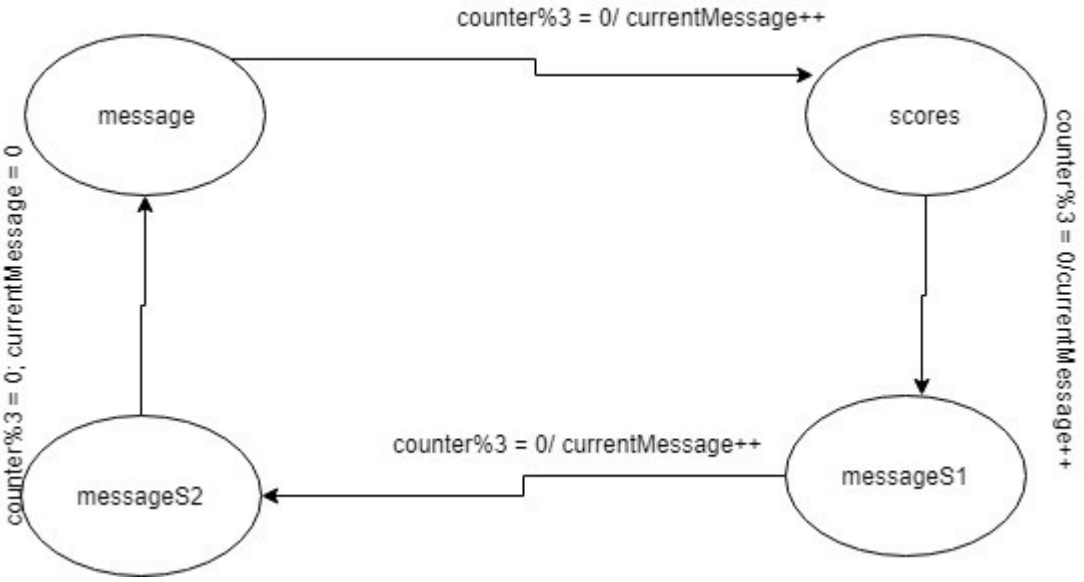
My lab 4 can accomplish everything listed in the specification. In this tic tac toe game, the loading screen is where the computer plays itself and goes through its own playing animation. The message area also alternates between the different messages such as score, name of game, and what the buttons do. In the game mode, the message area alternates between the computer and player depending on the button pressed and shows listening for capturing the DMTF signal and thinking for the computer's turn. Additionally, the microphone correctly samples and determines the proper cell to draw an X in, while playing the proper two tones according to where the computer dropped the cross in the cell. If there is a winner between the computer and player, there is a long string of notes played and then the game state goes back to idle. As extra credit, I implemented lights during and at the end of the game as well as changing the LCD colors during the game. I also implemented the power meter bonus at the bottom of the screen during the GAME mode.

Section 2: Finite State Machine Design

I used a big finite state machine to handle the game mode of the lab. My FSM is divided into many states, such as playerTurn, computerTurn, neutral, illegalMove, abortGame, and winning. In the playerTurn state, I check if the game has been aborted by the player, or I wait until a tone has been heard before placing the O in the correct cell, then the state goes to the computerTurn. If the tone heard does not correspond to a cell or if the cell is already taken, the state will shift to the illegalMove state. For the computer turn, I first start the timer and then use RandomAdd() to place a cross in a valid space. If the timer has expired, the computer will place an X in an empty spot using RandomAdd(). The abortGame state will begin once button 1 is pressed during the playing of the game. The winning state will perform everything related if someone has won, and use the extra credit light show to display the lights and playSound_winning() to play tunes. The below FSM showcases the process I created to take care of the GAME state entirely.



Another FSM I created was for the message area to toggle in the idle state. Every 3 seconds, the message area would toggle between the name of the game, score, I start and you start. I used a counter and modulus to accomplish this.



Section 3: HAL Design

InitColorLED() – Initializes the LED on the Booster Board

ColorLEDSet(color_t t) – sets the color of the booster board LED depending on color_t

playColorsRandom() – the light show (extra credit) after someone wins

DTMFfindSpot(tcellstate map[9]) – uses the DTMF encoder to play a spot depending on the tone heard through the microphone

playerWins(**int** winner) – goes through the process of winning for the player case

computerWins(**int** winner) – goes through the process of winning for the computer case

process(**int** b1) – sub-level FSM that takes care of the tic tac toe game

messageFSM(messageState *currentMessage) – alternates the message area between score, player message, etc.

playWinningSounds() – plays a song once someone has won, computer or user and plays the color show

playSound_tie() – plays a different song if there is a tie with different color animations

playSound_Cells(**int** arrayLocation) – plays the notes according to when the computer places a cross

Section 4: Program Structure

The structure of my program is divided up into many files that were taken from the homework and imported into the lab folders. In class, we call it HAL. The major FSM is within the function process, which handles the GAME mode. It consists of timers, and other functions that use HAL functions to complete the game. The interrupt global variable is also taken into account during the playerTurn state. I did not implement a better than random strategy for the computer. SO I simply use RandomAdd() for the computer's turn.