Microcontrollers Lab 4

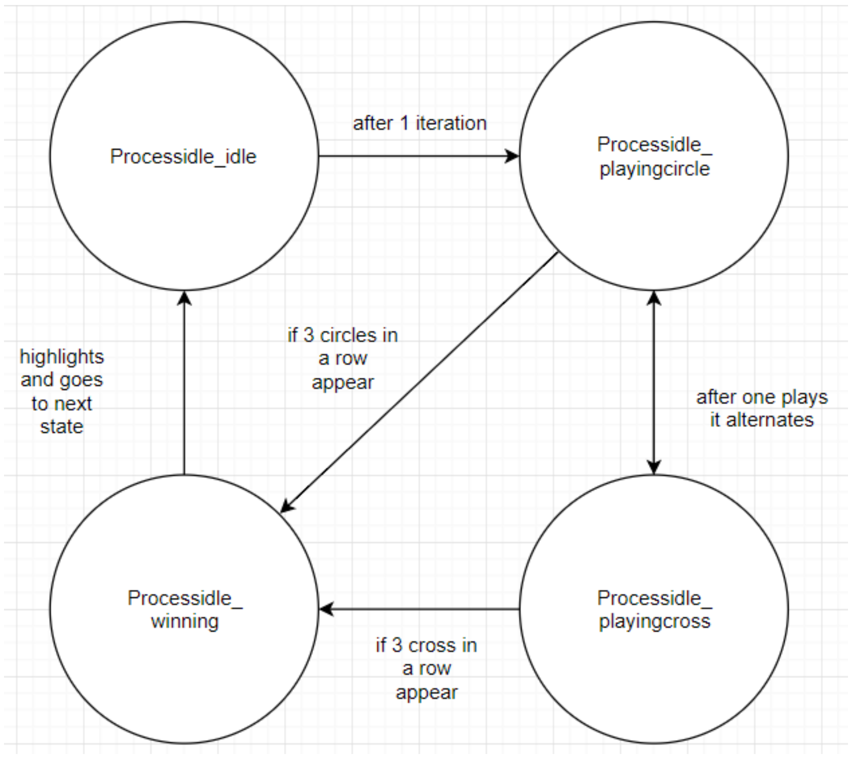
Section 1: Description

This lab is built upon a variation of Tic Tac Toe wherein which the user communicates to the microcontroller board by using DTMF tones similar to what you hear when dialing a phone number on your cell phone. The game begins when the user clicks button 1 or button 2. By clicking button 1, the computer gets to play first, whereas, if the second button is pressed, then the user gets to go first. The menu screen has a rotating display that has a completely automated game playing until the user selects a button. The top of the screen also displays the current overall score as well as which button corresponds to who plays first. Ultimately the goal of the game is to beat in the in-game AI at a game of Tic Tac Toe. This can be done by getting three circles in a row, column, or diagonal. While it is the user's turn to play, the unsigned representation of the signal power is displayed at the bottom if the screen.

There are a few sub-optimal sections for the application, the main one being once the game is over, the game is a little finicky with how long "game over" will be displayed to the screen. On top of that, occasionally, while it is the users turn, the code will very briefly show invalid move at the top of the screen. This usually happens when there is a lot of outside noise; this is due to my method of checking to see if a previous map location is currently used up.

Section 2: FSM Design

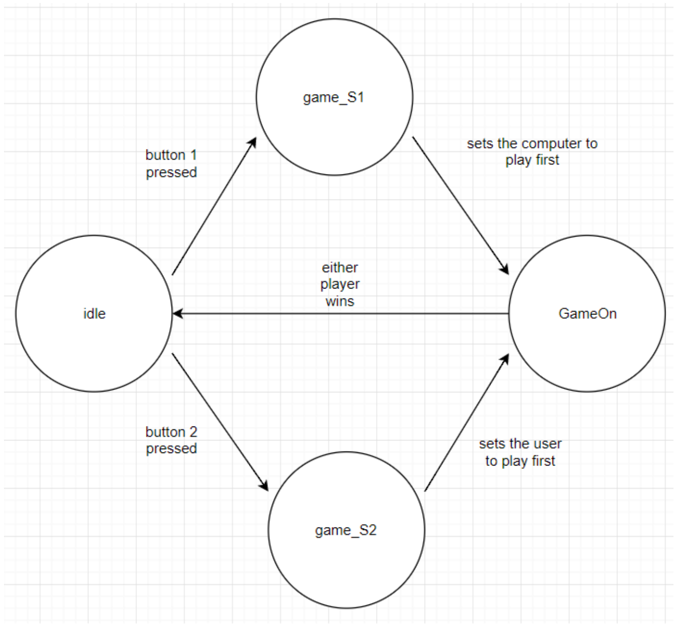
Main screen FSM:



Processidle_idle
after 1 iteration
Processidle_ playingcircle
highlights and goes to next state
if 3 circles in a row appear
after one plays it alternates
Processidle_ winning
Processidle_ playingcross
if 3 cross in a row appear

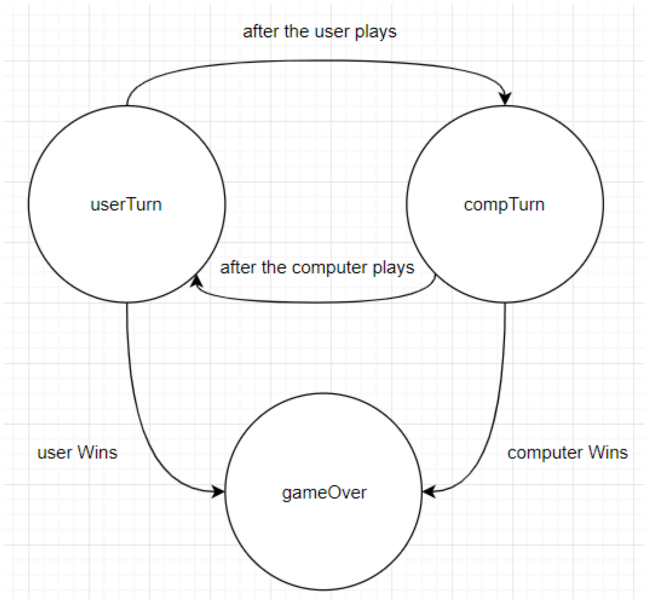
This FSM was given to us within the starter code. This takes care of the entire main menu screen saver. The FSM shifts from state to state every iteration. It'll move from the idle state, to the playing circle state. Then proceed to randomly place the corresponding piece, and switching between playingCircle, and playingCross until one symbol wins. The state then switches to the winning state, where it highlights the winner, then returns back to the idle state to repeat the process.

Button press FSM



This FSM cycles through the user initialing the game. The state stays at idle until the user presses button 1 or button 2. Once either button is pressed, it sets the current player and then immediately switches to the gameOn state. The FSM will stay in the gameOn state until either the computer or the player wins the game. After this event occurs, the game then switches back into the idle state and begins to iterate through the previous FSM.

GameOn FSM



Within the GameOn state of the above FSM, the following FSM runs. Based on the current player value determined by the button press, the FSM will start at either userTurn or compTurn. From there the FSM will alternate states between userTurn and compTurn until either a winner is declared, or a tie occurs. Once this happens, the game will shift to the gameOver state, print game over and highlight the winning sequence, then the previous FSM returns to the idle state. In the event button 1 is pressed during this FSM, the state is immediately switched to gameOver, and calls an abort function to fill the board with x’s.

Section 3: HAL Design

- void InitButtonS1();
  - initializes S1 button for our use.
- void InitButtonS2();
  - initializes S2 button for our use.
- void InitTimer();
  - initializes the timers and sets the counters for each.
- void InitSound();
  - initializes the buzzer peripheral.
- void InitDisplay();
  - initializes the display for us to print to screen later.
- void InitMicrophone();
  - initializes the microphone to read the DTMF tones later.
- extern void Interrupt_enableInterrupt(uint32_t interruptNumber);
  - enables the interrupt and gets ready to listen.
- extern bool Interrupt_enableMaster(void);
  - enables the processor interrupt.
- void InitSWTimer(tSWTimer *T, uint32_t hwtimer, uint32_t period);
  - fills in the values for the one second timer, and the 50 ms timer.
- void StartSWTimer(tSWTimer *T);
  - begins the count down for each timer.
- int SWTimerExpired(tSWTimer *T);
  - checks when the timers have expired.
- extern void Graphics_drawStringCentered();
  - draws the messages to the screen centered.

Section 4: Program Structure

The menu/screensaver portion of the code utilizes an FSM to switch between the beginning of an autonomous game, to crosses playing, to circles playing, and repeats until it hits the end of the game. This FSM was provided to us in the starter code. I modified this to calculate the total number of seconds elapsed. Whenever the modulus of the total seconds elapsed with 3 equals to 0, I change the banner of the screen. This causes the banner to change every 3 seconds as per the lab specs. At any point in time, if b1 is pressed, the board is cleared, and the computer plays a move. Similarly, if b2 is pressed, the board is cleared yet again, and the user gets to play first. From this point. The program goes into a general gameOn state.

The gameOn state will continue until either a win or a tie occurs. The gameOn state switches between playing circle and playing cross, and in doing so will change the banner appropriately to either "Listening?" or "Thinking….". At any point in time if b1 is pressed, the board will be filled with x's, and show that the computer wins, updates the scores, then goes back to the menu/screensaver portion. During the game the user can pick a location to put a circle by utilizing a DTMF tone. If the user attempts to put a circle in a position that is already occupied, then the banner will display "Invalid Move". The computer utilizes slightly better than AI technique. It will check to see if it has a possibility of winning and put down a cross there. If there is no way the computer can win, it will check to see if the user is close to winning by checking for any set of two circles in a row, and places a cross to block the user from winning. If neither party is close to winning, then the computer will randomly pick a location. Thus, the most effective method to defeating the AI is by creating two possibilities of winning. It will block one possibility of winning, and then the user can drop a circle in the last place.

Once either party wins, the game will highlight the winning combo in yellow, say “Game Over” in the banner, and play a small victory jingle. This also occurs if the user aborts the game at any point. After the jingle plays, the scores are updated, and the grid is cleared. The game then goes back to the idle menu/screensaver state until the user presses either b1 or b2 again.

The program is broken up into several files. The main game is run in the maingame.c file. Anything button related occurs in the button.c file; display related is in display.c, and so on and so forth. The below table breaks this down further.

Maingame.c	Contains the main game, and the FSMs.
Button.c	Contains anything button related. Any processes used to trigger the button along with debouncing happens here.
Display.c	Contains helper functions that print the game to the screen, as well as the banner updating.
Dtmf.c	Accesses the Goertzel file to read the sounds read in by the microphone.
Goertzel.c	Converts the microphone sound into the individual signals by applying filters to it.
Hwtimer.c	Creates the timers.
Swtimer.c	Initializes the timers and has functions to start the timers and check if they’ve expired.
Maplogic.c	Contains all the functions that effect the map in any way. Including checking for winners, and randomly adding symbols into the map.
Microphone.c	Initializes the microphone and gets the samples from the microphone.
Sound.c	Creates the sounds for the buzzers to play by setting the appropriate frequencies, and setting the PWM values.