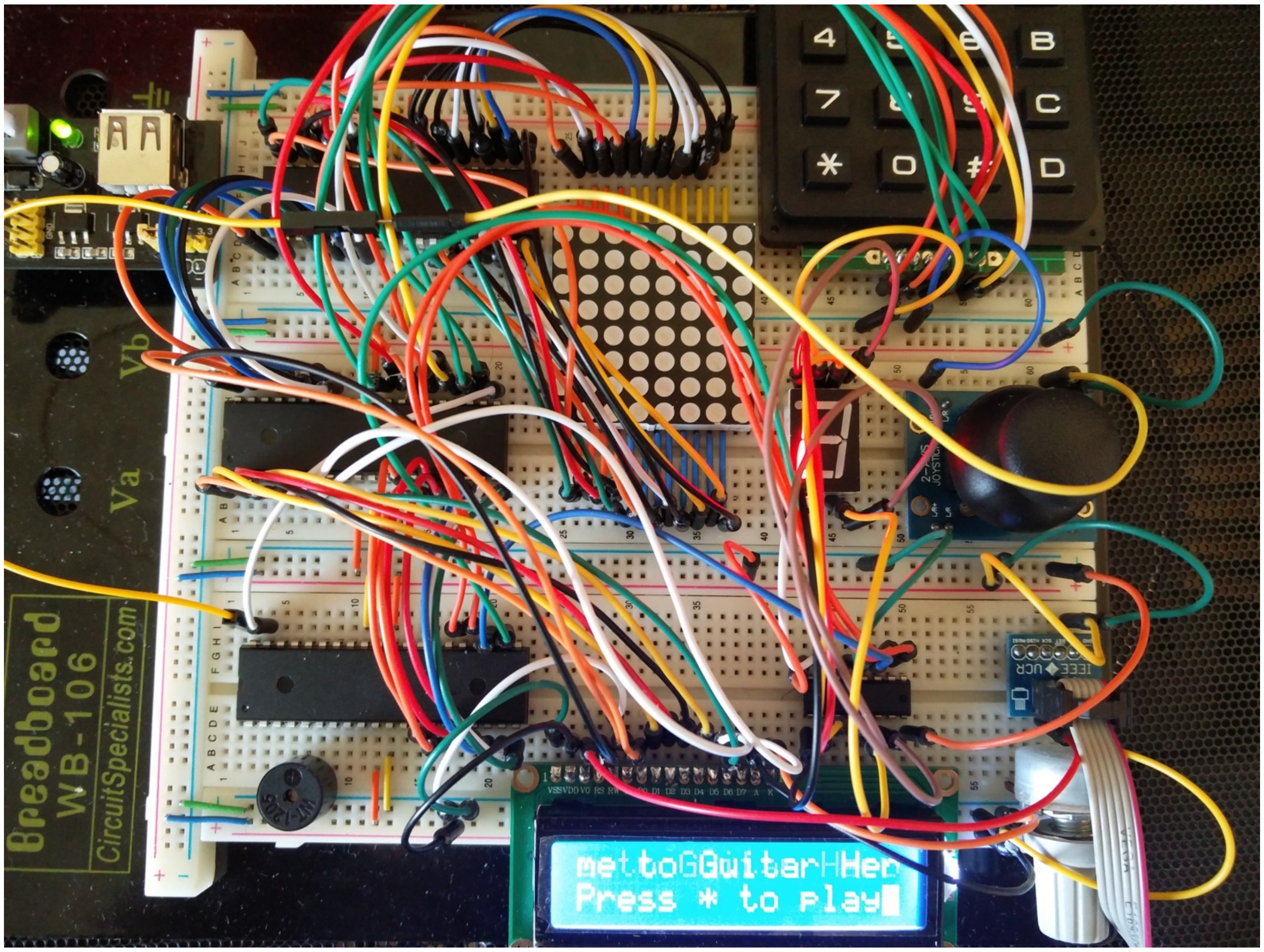
CS/EE120B Final Project Report Winter 2016

Guitar Hero

Nguyen, Kevin (knguy092@ucr.edu)
Thakkar, Jeet (jthak002@ucr.edu)



About

The ***Guitar Hero*** series (sometimes referred to as the ***Hero*** series) is a series of music rhythm games first published in 2005 by Red Octane and Harmonix, and distributed by Activision, in which players use a guitar-shaped game controller to simulate playing lead, bass guitar, and rhythm guitar across numerous rock music songs. Players match notes that scroll on-screen to colored fret buttons on the controller, strumming the controller in time to the music in order to score points.

Build Upons

- RGY LED Matrix **(Implemented)**
- Joystick for Strumming **(Implemented)**
- Multiple Microcontrollers and their Communication **(Implemented)**
- Shift Registers **(Implemented)**
- 7 Segment Display for multipliers **(Implemented)**
- Song Complexity (Final Countdown, Crazy Train, Super Mario Theme, Star Wars Theme, Sweet Child'o Mine, Iron Man-Black Sabbath) **(Implemented)**

Schematic

3 Microcontroller:

- Microcontroller_1 (Slave):
    - A0 : Joystick U/D Pin
    - A1 : Microcontroller_2 Pin D7
    - C0 – C4 : Shift Register 7595
    - D0 – D6 : Microcontroller_2 D0-D6
- Microcontroller_2 (Master):
    - A0 - A1: Control lines for the LCD Display
    - A2 – A3: B0 – B1 Microcontroller_3
    - A4 – A7: B4 – B7 Microcontroller_3
    - B0 – B7: Pins D0 to D7 to LCD Display
    - C0 – C7: Keypad
    - D0 – D6 : Microcontroller_3 D0-D6
    - D7: A1 Microcontroller_1
- Microcontroller_3 (Slave):
    - A0 – A7: Pins 1-4 and 21-24 of the LED Matrix (COLUMNS)

  - B0-B1: A2-A3 Microcontroller_2
  - B4 – B7: A4 – A7 Microcontroller_2
  - C0 – C7: Pins 13 – 20 LED Matrix (ROWS RED)
  - D0 – D7: Pins 5 – 12 LED Matrix (ROWS GREEN)
- Joystick
  - Right GND: Ground
  - Right U/D: A0 Microcontroller
  - Right U/D+: VCC
- LED Matrix: Find Pin 1 on the LED Matrix (denoted by a silver 1 next to the Pins) and the pins are counted counter clockwise.
  - Pins 1 – 4 and 21 – 24: Column Manipulations PORTA Microcontroller_3
  - Pins 5 – 12: Green Row Manipulations PORTD Microcontroller_3
  - Pins 13 – 20: Red Row Manipulations PORTC Microcontroller_3

State Machine Descriptions:

microcontroller_1.c

SM_Melody: This is state machine has fourteen states. This state machine is called every 35 milliseconds inside a loop in the main function. Each state is responsible for iterating through an array. There are three arrays for every melody, an array that stores the note frequencies to play the melody, an array that stores how long each note is carried out for, and an array that stores a slight pause between each note so that the melodies play smoother. We start in the WAIT state waiting for a signal to tell the program which song to play. Once we receive a signal from PORTD through USART0, we go into the respective song state. We play a note while checking the other array for how long we sustain the note, once we play the note to its duration, we have a slight pause until we read in the next note.

SM_state5: This state machine receives the multiplier value from the microcontroller_2 through PINS D2-D6 on PORTD of microcontroller_3. This value is then broadcasted to the shift register through the transmit function, which transfers the value to the shift register (SPI).

SM_state6: This state machine handles the input from the analog stick via ADC and passes it to microcontroller_2 to be processed, by setting the PIN A0 of PORTA of microcontroller_1 to 0 or 1 to detect analog toggle on or off and send that data to PIN D7 of microcontroller_2.

microcontroller_2.c

SM_Tick1: This is the state machine that encapsulates the menu of the game. The game starts with Startup which displays 2 strings: "*Welcome to Guitar Hero*" and "*Press * to play*". Since, the length of the first string is greater than 16 characters we have to scroll the first string across the display. This is achieved by the function shift_1bit. The second state select_song allows the user to choose songs and sets the gameplay variable startvar to 0x01; This is the variable that lets all of the state machines execute their gameplay states. The Play_screen state counts down the time left for each song and then tallies the final score based on correct button presses. . The final screen chooses the user to select the next song.

SM_Tick2: Responsible for taking input and avoiding multiple button presses by setting the button to an indeterminate setting NOT the null value. The tick function is called every 100ms for the next button input to check for other button presses.

SM_Tick3: Checks the startvar and sends out the signal for starting gameplay, by setting A0 of microcontroller_2 to 1, which signals microcontroller_3 to start generating the beats on the LED_Matrix and keep doing that until startvar is set to zero. Disabling startvar results in the last beats falling down on the LED Matrix and tallying the final scores. This state machine also adds to the score by checking if the right buttons have been pressed and then adding to the multiplier accordingly. For the purposes of this demo, we have not introduced a systems for penalizing th e players for wrong button presses or setting the multiplier to zero, incase of no button presses. The buttons 'D', 'C', 'B' and 'A' correspond to the 1st 2nd 3rd and the 4th row.

SM_Tick4: Sends the signal for the multiplier to microcontroller_3, which sets all the LED's to yellow, once the multiplier reaches 8. The multiplier is indicated on the 7 segment LED display.

SM_Tick5: This state machine transmits the value of the multiplier through PINS D3- D6 on PORTD of microcontroller_2 to microcontroller_1, which in turn displays the value of the 7 seg display through the shift register.

microcontroller_3.c

SM_display: This state machine displays the LED's which light up at a particular point of time. The State machine gets this information from the 16 element array which lets this SM know which LED blocks to light up. Also, based on the event that B1 is 0 or 1, the multiplier is enabled, which causes the yellow LED's to light up.

SM_shift: This state machines moves the LED block down by one; This is achieved by setting the a[n+4th] block to the same value as a[n]. This gives us the effect of blocks falling down. This state machine is called every 400ms and the difficulty of the game can be varied by increasing or decreasing the time when this function is called.

SM_button_identify: This state machine detects, if block is at the bottom and sets the PINS 4 - 7 of PORTB to identify the button that has to be pressed and pass that value from microcontroller_3 to microcontroller_2 for processing the SM_Tick3 i.e The gameplay state..

SM_Generate_Random: This state machine generates the random blocks that fall in the matrix. Only one of the first 4 elements of the array are set to true; that generated block falls to the next level in the next cycle of SM_shift.

Link to Video:

https://youtu.be/8IulDbzZ6uc

Bugs:

- **[FIXED]** For the purposes of the demonstration and due to space constraints, we have faced difficulties strumming and pressing the frets at the same time. Therefore, we have set the analog stick to work while being pressed down. This can be fixed by designing a state machine that disables the joystick until the ADC_val is detected to be zero, a SM similar to that of a button
- We are having trouble playing the next song on the replay screen as the game proceeds to play but the next song that is selected doesn't play.

Instructions:

Switch ON and follow the instructions to select the song. Once the song starts playing and the blocks start falling, press the key 'D', 'B', 'C' or 'A' corresponding to the column in which the block is at the LAST row. Each successful button press coupled with a well timed Strum gives you a point. Also, each point increases the multiplier by 1, stopping at 8, where the LED's turn YELLOW, making it more difficult to distinguish between columns. Best of Luck!