Homework 9: Software Design Considerations

Team Code Name: Beat Square **Group No. 1**

Team Member Completing This Homework: Jonah Ea

E-mail Address of Team Member: jea @ purdue.edu

Evaluation:

SEC	DESCRIPTION	MAX	SCORE
1.0	Introduction	5	
2.0	Software Design Considerations	30	
3.0	Software Design Narrative	30	
4.0	Summary	5	
5.0	List of References	10	
App A	Flowchart/Pseudo-code for Main Program	10	
App B	Hierarchical Block Diagram of Code Organization	10	
	TOTAL	100	

Comments:

1.0 Introduction

Beat Square is an 8x8 push button matrix that will support multi-tone audio playback and corresponding LED lights. The matrix will light up each column, left to right, and produce the tone associated with any button that has been enabled on for that column. The speed in which the columns are selected, in other words the tempo, may vary based on the adjustment of the rotary pulse generator. Beat Square also supports loading and saving musical configurations via flash on the Tiva microcontroller being used for the design. Support for writing audio samples to an SD card is available in the form of a MIDI file. Peripherals for the design such as the RGB LEDs of the push buttons, four-line user LCD, and DAC will be interfaced through the use of SPI and I^2C.

2.0 Software Design Considerations

Software design considerations for Beat Square were greatly influenced by the microcontroller that was selected to be as part of the project design. The microcontroller that was chosen for Beat Square, a Texas Instrument Tiva TM4C123GH6PM microcontroller, can operate at speeds up to 80 MHz and contains 256 KB of flash memory. Communication interfaces supported for the Tiva include forty-three GPIO pins, eight UARTs, four SPI ports, and four I^2C ports ([1], pg. 45). A cortex debugger will be used to program the Tiva. Also, a heartbeat LED sensor will be used to detect that the microcontroller is powered on.

The 74HC595 shift register connects to PB0-PB1 and PD0-PD3 (pins 45-46 and pins 61-64). There will be four shift registers daisy chained that will be interfaced to the push buttons on the 8x8 matrix. SPI will be used to clock the data. SSI3, the third SPI port will be enabled with master mode set.

One I^2C port is used to interface to a DAC for audio output. The DAC supports a standard, fast, and high-speed mode using two wires ([2], pg. 1). For this design, the 3.4 MHz high-speed mode will be used to transmit to the DAC. This mode requires the Tiva to send out 00001xxx after an I^2C start condition to signify to the DAC that it wishes to communicate at 3.4 MHz. API for using I^2C requires that SysCtlPeripheralEnable() call the right I^2C port number. In this case, the DAC is connected to I2C0, the first I2C port. The SCL line is PB2 (pin 47) and the SDA line is PB3 (pin 48).

The SD card slot used to write audio in a MIDI file contains six pins which are connected to ports PA0-PA5 (pins 17-22). The four-line LCD display is connected to ports PB4-PB7 and PF1-PF4. The four push buttons associated with the four-line LCD are connected to port PE0-PE3 (pins 6-9). The LCD will display for the user file options. This will include loading and saving to flash, writing to SD card, and a pause feature. The two rotary pulse generators for volume and BPM connect to port PE4-PE5 (pins 59-60).

The overall functionality of the button matrix will be driven by both timer interrupts and asynchronous interrupts since the nature of the project contains synchronous and asynchronous elements. Asynchronous interrupts will be used to service a routine that toggles a flag bit when the user interacts with any of the button inputs on the board and checks debouncing. The push buttons will have a 1-bit flag while the rotary pulse generators for volume and BPM will have 4-bits. For the synchronous column checking, timer interrupts will be used to check the flags to see if they are on and call the required functions to turn the LED on and play audio. The timer interrupt will be set as fast as it can be set, and a separate flag will be asserted high at the same speed of the current tempo. This will be used to essentially "slow down" the pace of the timer interrupt as the tempo gets adjusted to a lower value. When the tempo flag is asserted high, the timer interrupt will call its function to check the input flags to see if those are enabled.

Beat Square's main program lies within flash memory. The main program includes initializations of registers and function routines. On boot up, the program data is copied to SRAM and runtime data is stored there. Flash will contain the configuration data for Beat Square when a user loads and saves (which is also specified as a project specific success criteria). The look up table for the audio is also stored in flash, and eight buffers in SRAM are used to hold the data for reading when playing various tones.

3.0 Software Design Narrative

The main program of Beat Square holds the function calls to the peripheral initializations and the normal loop routine. At power boot up, the main program will call the initializations and then proceed to stay in the normal routine indefinitely. The initializations will set the registers and enable ports for communication so that the microcontroller can talk to the different peripherals in the design. Also, the speed of the timer interrupts will be set here as well. Code has been written and prototyped for the I^2C DAC and SPI 74HC595 shift registers while the

other initializations are still in pseudo-code and testing. Source code listings are provided here:
https://engineering.purdue.edu/477grp1/docs/SPI%20Initializations.txt
https://engineering.purdue.edu/477grp1/docs/I2C%20Initializations.txt

The normal looping routine will wait for and service any timer interrupts and interrupts triggered by user input. The asynchronous interrupts uses software debouncing to assure that there is valid input. After it does, it will call a function to set the peripheral flag to its opposite state and return to the normal routine. The exception to this is the software reset triggered by the asynchronous interrupt. After debouncing, instead of calling the set flag function, it will directly proceed to clear the LCD and button matrix of their current data and reinitialize the board. The synchronous timer interrupts will check for the tempo flag. If it is set, it will call a function to check the peripheral flags to see if they are enabled, and if those are then the function associated will be called, e.g. matrix push buttons have the timer interrupt call the functions to play audio and light the LED. The code for the interrupts are written in pseudo-code for a basic understanding but have not been tested yet.

Beat Square's input methods include mainly push buttons and two rotary pulse generators. The outputs that need to be outputted by function calls are text to the LCD, audio, and LED lights under the push buttons. There will be predefined strings of text that the LCD will update. Another 74HC595 shift register will be used to select the column to check, and the 74HC165 shift register will simultaneously read the data from the selected column and send it back serially to the Tiva. Chosen colors for the RBG LEDs will be defined as constants and called when a function wants to turn them on. Four 74HC595 shift registers will send out data to light them on. The audio will be sent from the microcontroller to be read by one of the eight buffers (depending on which of the eight tones it is) when a push button is enabled. The DAC will convert the audio data to an analog signal via I^2C.

4.0 Summary

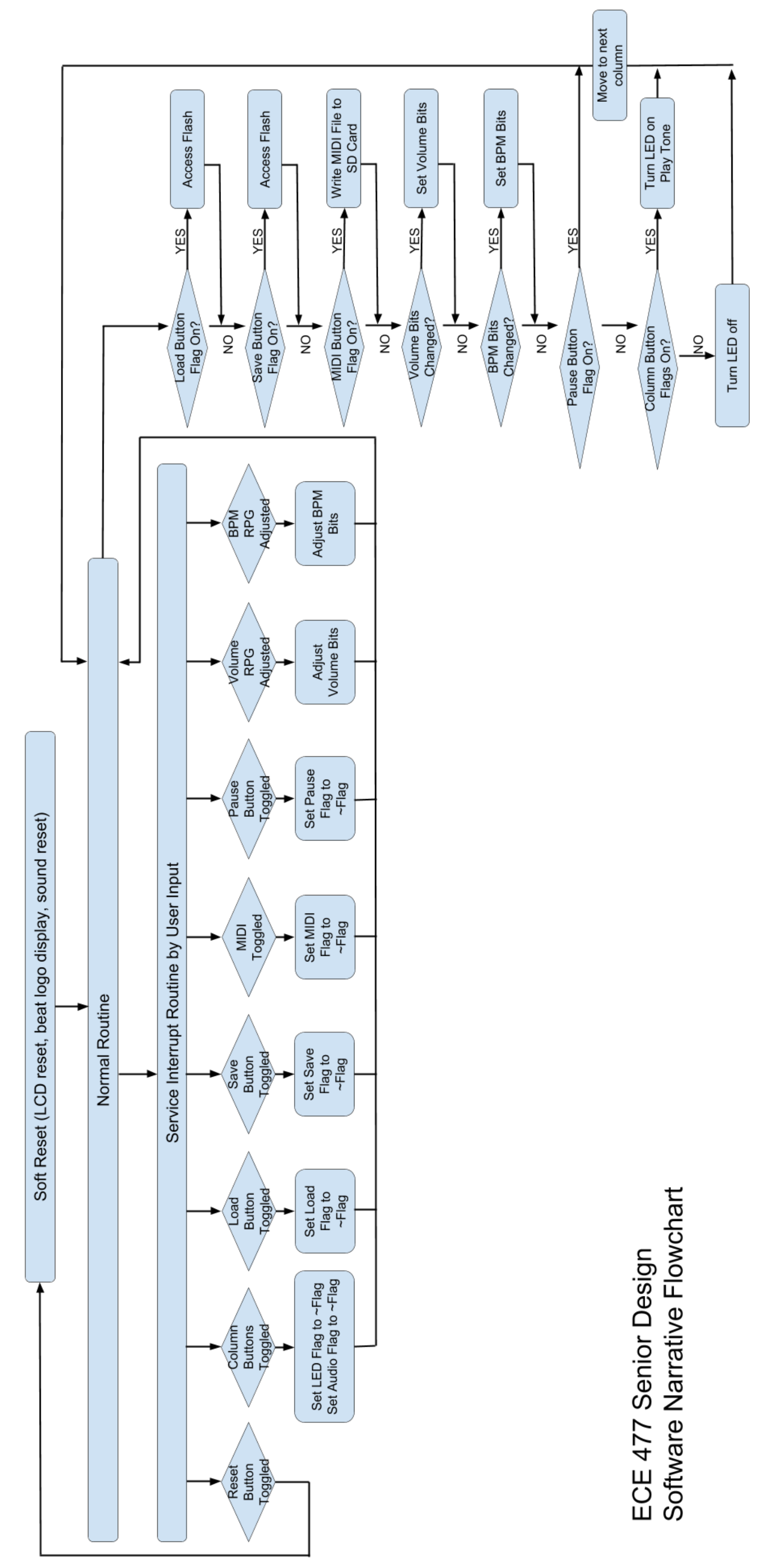
This report provides an overall description of the Beat Square project in a software design perspective. It highlights the underlying implementation details used to accomplish the main functionality of the design. The specific port mappings and associated initializations for various peripherals are listed, and the memory layout is stated for the program code and runtime data. A narrative of the program walks through the functions in a logical manner branching through the different peripherals and their roles in the design. A flowchart of the program as well as a hierarchical block diagram is provided for a visualization of the code behavior in the appendix section.

5.0 List of References

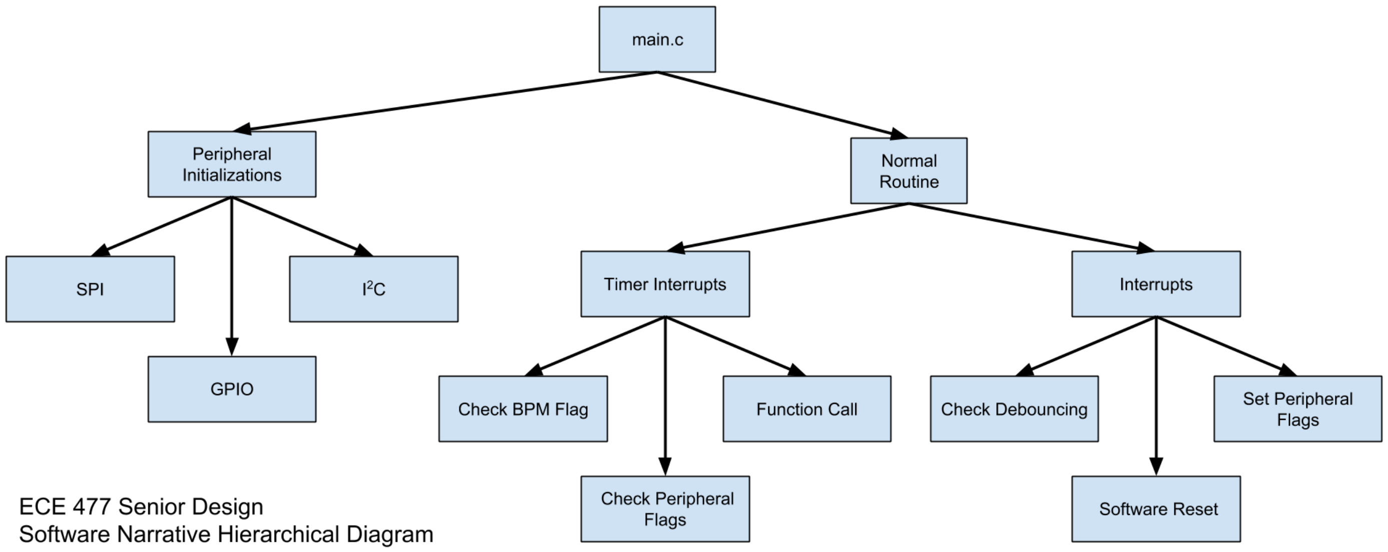
[1] Tiva^{TM} TM4C123GH6PM Microcontroller. Texas Instruments. Austin, TX. [Online]. Available:https://engineering.purdue.edu/477grp1/docs/ref/TM4C123GH6PM%20Microcontroller.pdf

[2] 16-Bit, Low Power, Voltage Output, I^{2}C Interface Digital-To-Analog Converter. Texas Instruments. Austin, TX. [Online]. Available: https://engineering.purdue.edu/477grp1/docs/ref/DAC8571%20Digital%20Analog%20Converter.pdf

Appendix A: Flowchart/Pseudo-code for Main Program



Appendix B: Hierarchical Block Diagram of Code Organization


main.c
Peripheral Initializations
Normal Routine
SPI
I²C
GPIO
Timer Interrupts
Interrupts
Check BPM Flag
Function Call
Check Peripheral Flags
Check Debouncing
Set Peripheral Flags
Software Reset
ECE 477 Senior Design
Software Narrative Hierarchical Diagram