Homework 10: Software Design Considerations

Due: Friday, September 29, at NOON

Team Code Name: _Digital Sheet Music Reader and Player___________ **Group No.** _4_____

Team Member Completing This Homework: _David Hartman_________________________

e-mail Address of Team Member: __drhartma_____ **@ purdue.edu**

NOTE: This is the last in a series of four "design component" homework assignments, each of which is to be completed by one team member. The completed homework will count for 20% of the individual component of the team member's grade. The body of the report should be 3-5 pages, **not** including this cover sheet, references, attachments or appendices.

Evaluation:

Component/Criterion	**Score**	**Multiplier**	**Points**
Introduction & Summary	0 1 2 3 4 5 6 7 8 9 10	X 1	
Software Design Considerations	0 1 2 3 4 5 6 7 8 9 10	X 3	
Software Design Narrative	0 1 2 3 4 5 6 7 8 9 10	X 3	
List of References	0 1 2 3 4 5 6 7 8 9 10	X 1	
Appendices	0 1 2 3 4 5 6 7 8 9 10	X 1	
Technical Writing Style	0 1 2 3 4 5 6 7 8 9 10	X 1	
		TOTAL	

Comments:

Comments from the grader will be inserted here.

1.0 Introduction

The Digital Sheet Music Reader and Player is a portable device that takes bitmap images of sheet music off of a USB flash drive and plays all of the selected music. This is a very heavily software dependant device using a SHARC ADSP-1262 [1] embedded processor to convert the images into code stored as intermediate music symbols (IMS) files that represents the notes' pitch, length, and location in the song. The code will then need to be converted into the MIDI format to be played through a MIDI processor. The better this software is, the more musically complex the scanned music can be. Additional peripheral units include the LCD screen and the user interface consisting of pushbuttons and a rotary pulse generator (RPG). This software will be command driven through the user interface, but since the device needs files from the user before it can do anything useful, the microcontroller will have an external interrupt looking for a USB flash drive to be connected so that the files from the USB drive will be immediately put on the external SRAM. Software to access all of the IMS files and selected images on a FAT [2] formatted USB drive must be created so that the files can be saved on our device's external memory and files created by our device can be saved back on the USB flash drive.

2.0 Software Design Considerations

2.1 Memory Mapping:

0x0000 0000 – 0x0003 FFFF	Registers
0x0004 0000 – 0x0007 FFFF	Bank 1 (long word)
0x0008 0000 – 0x000F FFFF	Bank 2 (normal word)
0x0010 0000 – 0x001F FFFF	Bank 3 (short word)
0x0020 0000 – 0x00FF FFFF	Reserved
0x0100 0000 – 0x010F FFFF	External SRAM
0x0110 0000	USB controller (chip select generation)
0x0110 0001 – 0x01FF FFFF	Unmapped memory addresses
0x0200 0000	MIDI processor (chip select generation)
0x0200 0001 – 0x02FF FFFF	Unmapped memory addresses
0x0300 0000 – 0x3FFF FFFF	Reserved

Bank 1

0x0004 0000 – 0x0004 3FFF	Block 0 SRAM (Stack)
0x0004 4000 – 0x0005 7FFF	Reserved
0x0005 8000 – 0x0005 FFFF	Block 0 ROM (Code)
0x0006 0000 – 0x0006 3FFF	Block 1 SRAM (Static Data and Variables)
0x0006 4000 – 0x0007 7FFF	Reserved
0x0007 8000 – 0x0007 FFFF	Block 1 ROM (Code)

The structure of Banks 2 and 3 are arranged similar to Bank 1. The SRAM in Banks 2 and 3 are reserved for an image being processed. The chip select generation for the USB controller and the MIDI processor will be made by sending the top four address lines to a PLD.

2.2 External Interfacing Mapping:

The LCD will use the SPI with register at 0x0000 1002. The pushbuttons and rpg will use Port A [0:4] of the Digital Applications Interface (DAI). The Signal Routing Unit (SRU) is very flexible in customizing the DAI pins by modifying the SRU register at 0x0000 2440. The external bus uses the parallel port [AD0:AD15]. It receives data at 0x000 1809, and it transmits data at 0x0000 1808. The external bus has three control lines [ALE, Read, Write].

2.3 Utilization of Integrated Peripherals:

The DMA will handle data transfers to the SPI as well as the parallel ports without intervention by the core. This is much more efficient than having the core handle the data transfers. The DMA can be enabled on the parallel port on the register at 0x000 1818. The DMA can be enabled on the SPI using the SPIDMA configuration register address at 0x000 1084.

2.4 Organization of Application Code:

This device is a hybrid of command driven, polling loop, and interrupt driven. The device does not save any images or IMS files to non-volatile space; therefore it cannot do anything useful without a USB flash drive connected to it. While there is no USB connected, the device will be in a sleep mode waiting for an external interrupt from the USB controller. In this sleep mode, the microprocessor's speed will be clocked down a lot to save power. When the USB is inserted, the code will then become a polling loop looking for pushbuttons from the user to select which function to do. Then the code will become command driven since there is a lot of software to process an image or turn IMS files into a MIDI file.

2.5 Flowchart: (see Appendix A)

2.6 Provisions Made for Debugging

There are 6 pins on the microprocessor that are used for JTAG/ICE debugging. It is interfaced to the computer so that we can observe the errors while the chip is embedded into the system and not on a development board anymore. Once the LCD screen is working, we will be able to display our own error messages placed inside of our code to debug our software problems.

3.0 Software Design Narrative (see Appendix B for hierarchical diagram)

init

- This module initializes the dynamic memory allocation (BSS) to zero. This space will be used for the image later. It copies values of initialized variables to the data section. It initializes the stack pointer. In LCD initialization, the cursor will be set to home position, the flashing cursor will be turned off, and the backlight and contrast will be set to 50%.
- Completion Status: Still in development.

usb_interrupt

- This module is an external interrupt that will be given by the USB controller when a USB flash drive is plugged in. This interrupt will take the microprocessor out of sleep mode, and it will call the module import_ims.
- Completion Status: Still in development.

import_ims

- This module will take all IMS files from the FAT formatted USB flash drive, and store them in the external SRAM. The module will also take all of the names of the bitmap images on the USB and display them in the menu when the user looks to select an image to process. This module will have to be programmed so that it is compatible to copy files from a FAT file system.

- Completion Status: Still in development.

lcd_pmessage

- This module will be used to display the menu for the user to select what he or she wants to do. It will need to display all of the images names that can be selected from the USB for processing as well as the names of all of the IMS files stored on the SRAM for playing the music. This function will send a string of characters to fill up the entire 4x20 character screen by using LCD_outchar to print it one character at a time. This will refresh the screen every time a change in the menu happens.
- Completion Status: Still in development.

lcd outchar

- This module displays a single character in the LCD screen wherever the cursor is located in the screen, and it increments the cursor by one every time a character is printed to the screen.
- Completion Status: Still in development.

binary converter

- This module converts an image from grayscale to binary. It first looks at the how many bits wide the image is to determine if the image is binary or not. If the image is binary, the module is done with the image, otherwise the image will be brought from the USB drive to the microchip's RAM in chunks of 200 KB to be converted into binary. The image must be brought over in chunks of 200 KB since the image can be as much as 5 MB and the microchip only has 250 KB of RAM. The binary converter halftones the image using an error diffusion technique [3]. This works by setting a threshold of what gray values will be black and what will be white, but then it takes the error of how far the pixel was from being what the threshold determined it to be to help determine along with the threshold if the next pixel will be black or white. The converted image is stored on the external SRAM.
- Completion Status: Successfully coded and tested. Ready for integration.

fix_image

- This module deskews the image as well as helps fill in specks of white in a horizontal or vertical line. The image is deskewed fixing the horizontal lines with one pass through the image and then the vertical lines by a second pass through the image. All horizontal and vertical lines are assumed to be linear. Two fiduciary points are found marking the beginning of one of the staff lines and then end. A third fiduciary point is set to where the end of the line should be assuming that the beginning pixel is correct. The columns are then moved appropriately for all of the rows to make all of the lines perfectly horizontal. This same process is then applied to correct the lines that are supposed to be vertical. After this process is done, the image is scanned for white pixels surrounded by two black pixels in the same column or the same row, and then those white pixels are made white. This will fix the speckled line look the binary converter makes.
- Completion Status: Still in development.

staff_lines_locator

- This module locates all of the staff lines in an image, stores each staff's location, and deletes any lines that are not part of the staff. It looks for columns that have more than 50% black pixels in it, and the module calls that a staff. The column is stored in a vector for later use. The module then looks to see if all of those lines are equally spaced in sets of 5. It deletes any lines that are not part of the staff.
- Completion Status: Successfully coded and tested. Ready for integration.

measure_locator

- This module looks for columns of 90% black pixels in between all of the lines of a staff. This will distinguish between a measure line and the stem of a music note because a stem of a note cannot go through all of the lines of a staff. All locations of the measures are stored in a vector in order that they appear in the music.
- Completion Status: Successfully coded and tested. Ready for integration.

note_finder

- This module parses the image looking for black pixels that are not part of the staff or the measure line. It parses the image from top left down to bottom right, and it only looks at the current staff being processed and not the next one. When a pixel is found that could be a note, it calls on check_note to see if it is a note.
- Completion Status: Successfully coded and tested. Ready for integration.

check_note

- This checks a pixel that has been flagged as possibly a note, to see if it is in fact a note. It does this by comparing the surrounding pixels to each other looking at shape and size. It calls check_stem to make sure that a pixel being looked at is circle of the note and not the stem. This subroutine also identifies what type of note the note is by looking at the inside of the note and the stem. The note length is stored right next to the note pitch in memory.
- Completion Status: Still in development. Software has been made that will find all of the notes on the staff, but it also finds a few extra because the clef symbols mess it up.

check_stem

- This module looks at the thickness of the line the pixel is a part of and determines if the pixel is part of the stem or the circle of the note. Check_note can only identify a note if it starts at the edge of the circle of the note, so it is important to know where on the note the pixel is.
- Completion Status: Successfully coded and tested. Ready for integration.

note_value

- This module finds the pitch of the note by finding where on the staff the note is located. It also takes into consideration the key signature and the clef the staff is in. This data is retrieved from the answer the user gave in the menu. The note value is stored right next to the note length in memory.
- Completion Status: Still in development.

create_ims

- This module takes all of the saved data in memory and creates an IMS file from it to be later made into MIDI. The IMS files will store a char that represents the note's length and another char that represents the note's pitch for each note for that instrument. It also will store the song's tempo, key signature, type of clef, and time signature. All of this data is necessary to create a MIDI file. This file will then be saved onto a FAT formatted USB flash drive. This will require that our device be programmed to be compatible with FAT.
- Completion Status: Still in development.

ims_2_midi

- This module will take all selected IMS files and convert the stored data in these files into one MIDI file. This module will look at the first byte of the IMS file for the song's tempo, the second byte for the key signature, the third for clef, the fourth for the time signature, and then it will look at two bytes at a time for each note's length and pitch. Each IMS file will be made into a track with an instrument assigned to it, and it will be assigned to the same MIDI file as all of the other tracks that are selected by a user. Because MIDI files can be separated into separate tracks, it works perfectly with our IMS files. [4]
- Completion Status: Still in development.

4.0 Summary

The Digital Sheet Music Reader and Player will not have a shortage of challenges in creating all of the different software. Some of the challenges in DSP are too great for the time frame, so the device will only be able to handle simple notes and one note per beat on an image. There will also be challenges in creating a MIDI file from scratch and controlling the USB drive. Although it will be difficult, the integration of all these parts will certainly be music to team four's ears.

List of References

[1] Analog Devices, "SHARC Embedded Processor ADSP-21262," Aug. 2005, Available at HTTP: **http://www.analog.com/UploadedFiles/Data_Sheets/462724356ADSP_21262_b.pdf#search=%22data%20sheet%20ADSP-21262%22**

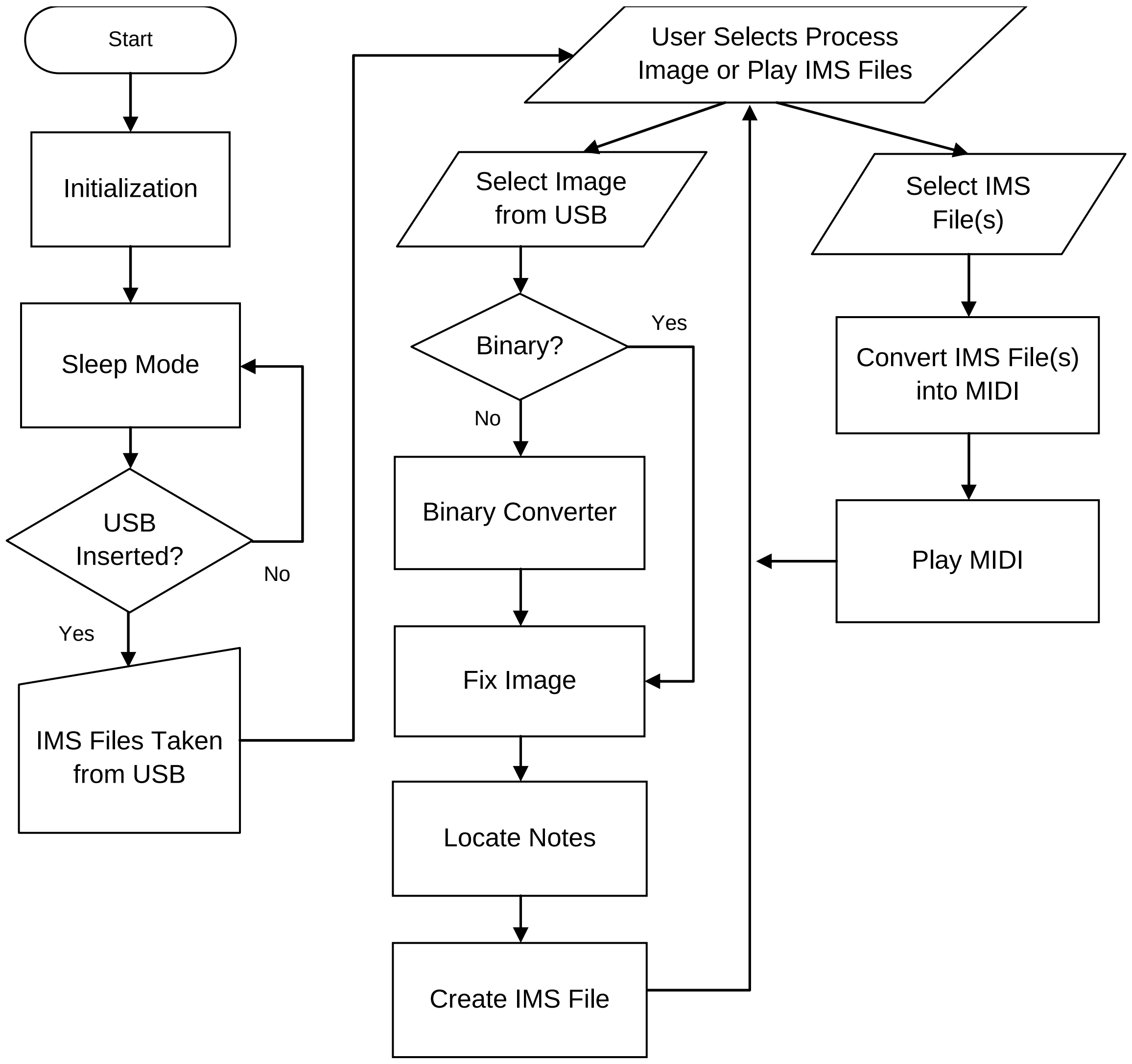
[2] Microsoft, "NTFS vs. FAT: Which Is Right for You?" Oct. 2001, Available at HTTP: **http://www.microsoft.com/windowsxp/using/setup/expert/russel_october01.mspx**

[3] Purdue University, "EE438 – Laboratory 10: Image Processing (Week 2)," Jun. 2004, Available at HTTP: **http://vise.www.ecn.purdue.edu/VISE/ee438L/lab10/pdf/lab10b.pdf**

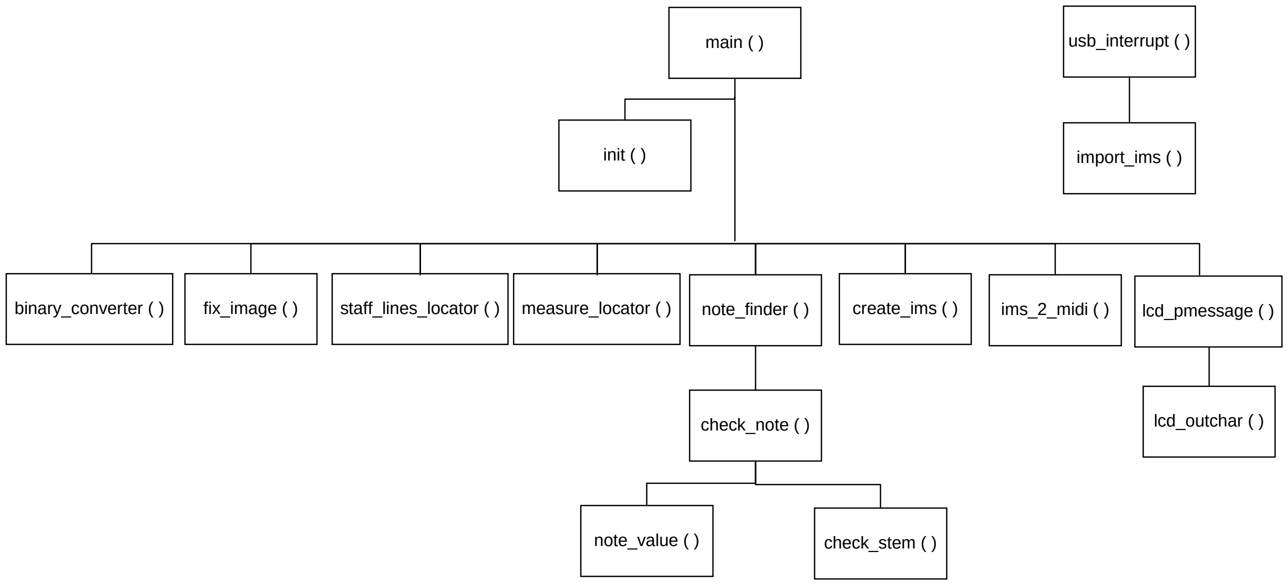
[4] "MIDI is the language of the gods." Available at HTTP: **http://www.borg.com/~jglatt/**

IMPORTANT: Use standard IEEE format for references, and CITE ALL REFERENCES listed in the body of your report. Any URLs cited should be "hot" links.

Appendix A: Flowchart/Pseudo-code for Main Program



Appendix B: Hierarchical Block Diagram of Code Organization


main ( )
usb_interrupt ( )
init ( )
import_ims ( )
binary_converter ( )
fix_image ( )
staff_lines_locator ( )
measure_locator ( )
note_finder ( )
create_ims ( )
ims_2_midi ( )
lcd_pmessage ( )
check_note ( )
lcd_outchar ( )
note_value ( )
check_stem ( )