Cody Onishi

2/13/20

Subsystem 5: User Peripheral Interfacing and Audio Digital to Analog Conversion

Subsystem Description

My subsystem focuses on managing the user’s button inputs, initialization of USB flash-drive and SD card interfacing, processing the user’s flash drive files into arrays that can be processed by the microcontroller, sending appropriate arrays of data to other subsystems based on the user’s current text position in files, and converting digital audio files into analog voltage signals.

Subsystem components:

* Play Button
* Mode Switch Button
* Where am I Button
* File/Chapter Navigation
* USB Interface
* Program Storage
* Flash Memory
* Position Control
* Audio Signal Conversion

Required Parts:

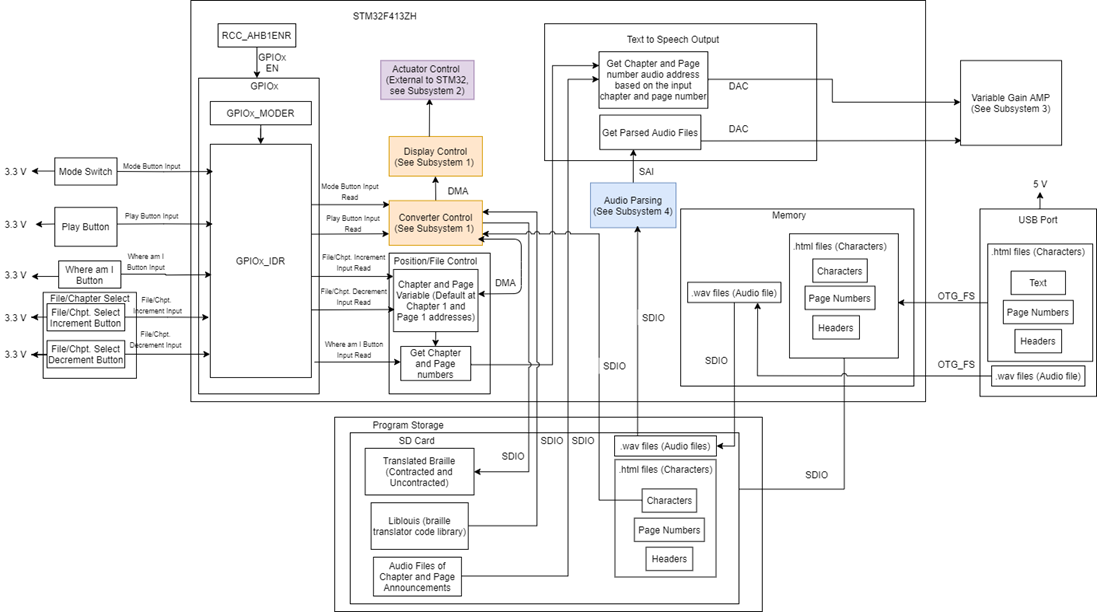
* STM32F413ZH microcontroller
* Micro USB (Male) to USB (Female) Adapter
* USB Flash Drive (for testing)
* Four SPST-NO Push buttons
* Four different CAD 3D-Printed button encasings of different textures/shapes
* Four 1 Kilo-Ohm resistors
* Micro SD 32 GB Card
* Micro SD Storage Board with built-in Micro SD Memory Shield

Required Software:

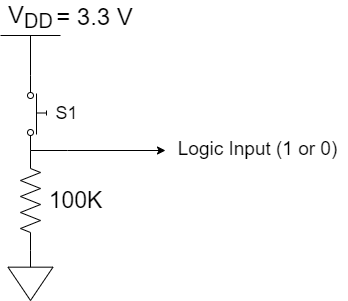
* Eclipse - STM Workbench (For use with programming  STM32F413ZH microcontroller)
* C-programming
* “Liblouis” braille translation library in C-programming language
* GPIOx initialization code
* USB\_OTG\_FS initialization code
* SDIO initialization code
* DAC code
* DMA code
* I2S audio file transfer code

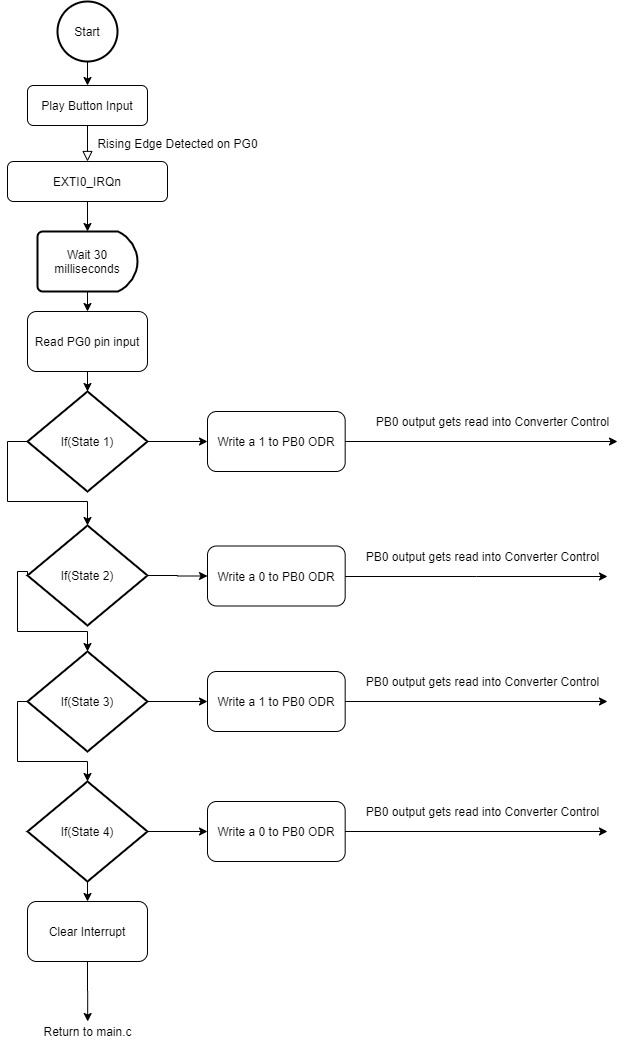
Subsystem Specifications:

1. All buttons will be Momentary SPST-NO push buttons
2. All buttons will be configured in active high configuration with a voltage supply of VDD = 3.3 V and a resistor connected to ground of at least 100 KΩ
3. All buttons will generate interrupts when pressed to avoid inefficient use of microcontroller functionality
4. All buttons will be debounced by adding a debouncing delay of at least 30 milliseconds before the state of the corresponding GPIO input pin is read again to prevent user frustration due to unexpected button behavior
5. All buttons will have different textures or shapes to help users distinguish between buttons through touch only
6. All buttons must complete their intended functions with less than 1 second of delay after being pressed
7. Channel 2 of the DAC on-board the STM32F413ZH Microcontroller will be used for generating audio output.
8. The DAC must be able to send 12 bits of data for every data transfer it conducts to be able to output audio file data effectively
9. The DAC must be able to output a voltage signal of frequency ranges between 20 Hz to 20 kHz to cover the entire spectrum of possible frequencies that could be used by the audio files supplied by the user
10. The DAC must output audio voltage signals of the displayed braille text within five seconds after the first letter of the new array of text is displayed on the rightmost braille cell
11. The microcontroller must be able to navigate to file and chapter locations accurately without error when the File/Chapter Navigation Buttons are pressed
12. The microcontroller must save the user’s mode settings as well as the data of the current chapter and page number left off from in flash memory to enhance user experience

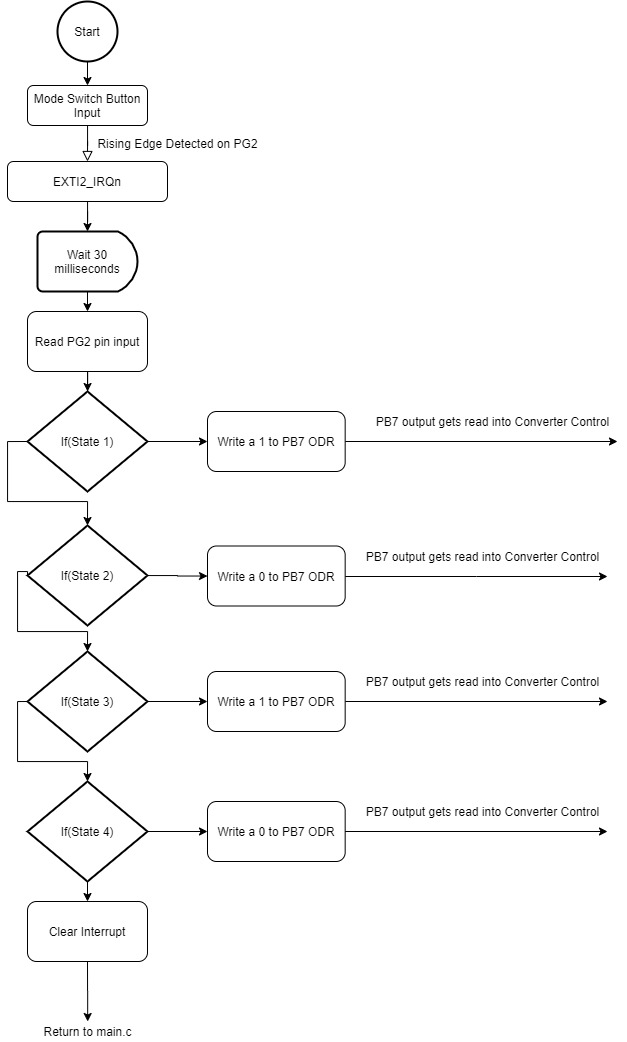


Schematic of General User Push Button:

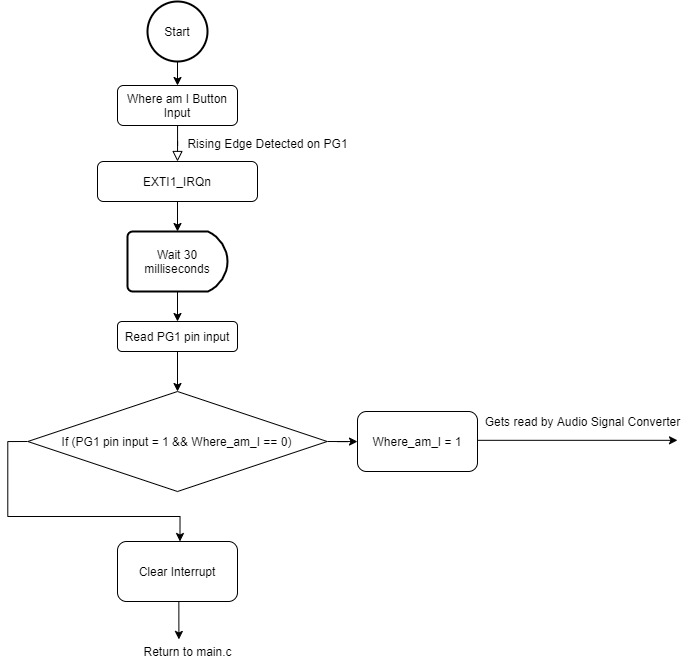


Play Button Flow Chart

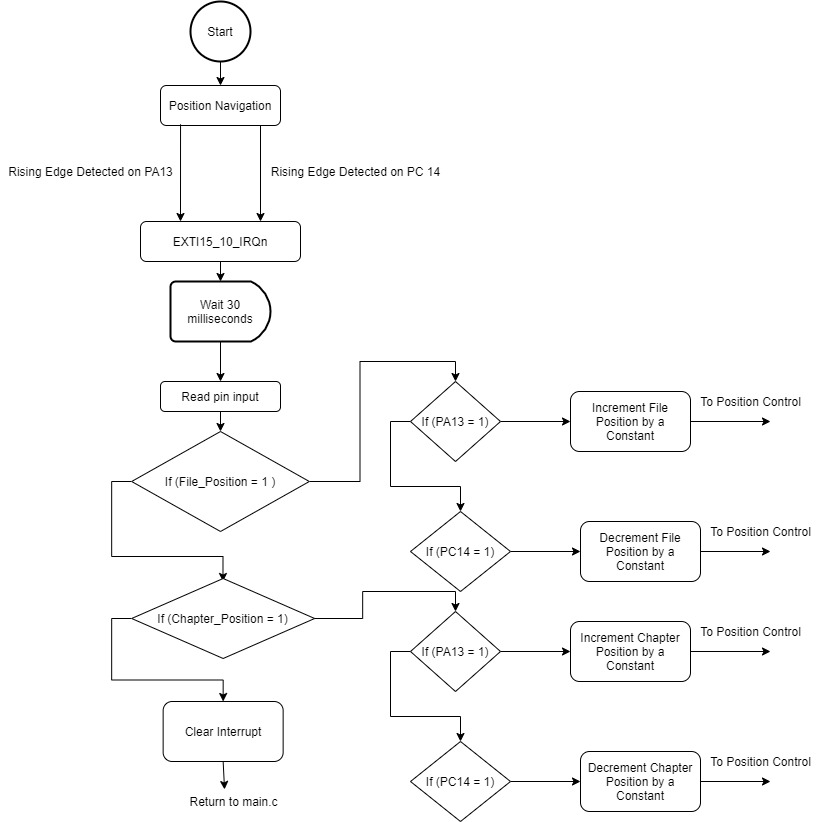
Mode Switch Button Flow Chart

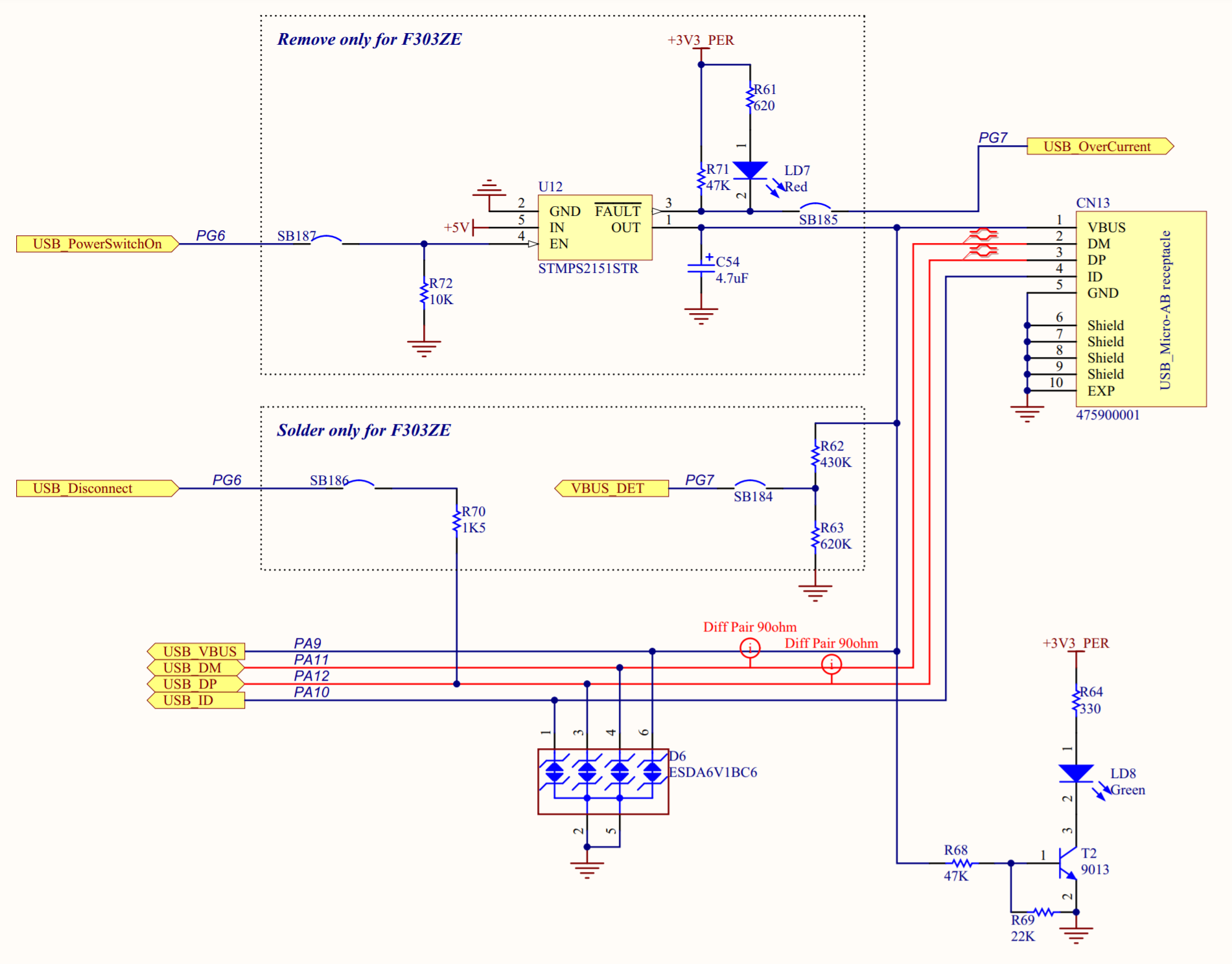


Where am I Button Flow Chart



File/Chapter Navigation:





Revised On/off Push button test code:

**void** **EXTI15\_10\_IRQHandler**(**void**)

{

lastDebounceTime = HAL\_GetTick(); //take snapshot of present operation time

Delay = debounceDelay + 1; //set the amount of time to wait for

HAL\_Delay(Delay); //wait the required amount of time

**if** ((HAL\_GetTick() - lastDebounceTime) > debounceDelay)

//after waiting long enough, button should be truly pressed

{

**int** reading = HAL\_GPIO\_ReadPin(GPIOC, GPIO\_PIN\_13); //read the state of the button into local variable

**if** (reading == 1)

{

**if** (before==0 && i==0)

{

present = 1;

}

**else** **if** (before==1 && j==1)

{

present = 2;

}

**else** **if** (before==2 && i==1)

{

present = 3;

}

**else** **if** (before==3 && j==0)

{

present = 4;

}

**else** **if** (before==4 && i==0)

{

present = 1;

}

}

}

// set the LED based on past input:

**if** (present != before)

{

**if** (present==1 && i==0)

{

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_7, 1);

j = 1;

}

**else** **if** (present==2 && j==1)

{

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_7, 0);

i=1;

}

**else** **if** (present==3 && i==1)

{

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_7, 1);

j=0;

}

**else** **if** (present==4 && j==0)

{

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_7, 0);

i=0;

}

}

before=present;

HAL\_GPIO\_EXTI\_IRQHandler(GPIO\_PIN\_13);

}

Subsystem Schedule and Deadlines:

2/10 – 2/16:

* Creating test code for transferring WAV audio files into DAC
* Purchase male micro USB to female USB 2.0 adapter
* Purchase 32 GB Micro SD Card online
* Purchase Micro SD Card Shield port online

2/17 – 2/23:

* Create test Code for DAC
* Create test Code I2S
* Create test Code for storing data into Flash Memory

2/24 – 3/1:

* Create Functional Micro SD Interfacing System
* Create Functional USB Flash Drive Interfacing System
* Create Test Demonstration Code for Play Button
* Create Test Demonstration Code for Mode Button
* Create Test Demonstration Code for File/Chapter Navigtion Buttons
* Create Test Demonstration Code for Where am I Button

3/2 – 3/4:

* Create prototype main.c function
* Integrate code from each of the components of my subsystem to be called by main.c function

3/5: Demo

Summary of work progress

Previous Work:

* Completed writing, revision, and testing of On/Off debounced Push Button Test Code with button input configured as an interrupt. This type of push button will be used for the Play, Mode, and where am I buttons.
* Gained familiarity with using STM32CubeMX program to initialize code and use it with Eclipse IDE
* Gained familiarity with programming interrupt service routines and making them able to edit variables declared in main.c function by defining and declaring the variables as volatile and making them extern when referenced in another .c file.
* Gained familiarity with making interrupt service routines have higher and lower preemptive priority settings to avoid interrupts resulting in infinite loops
* Found test code for checking the status of a USB flash drive plugged into the microcontroller as well as reading from and writing txt files into the flash drive. However, the code has not been tested yet due to not having the required male micro USB to female USB 2.0 adapter
* Found test code for reading from and writing txt files to SD cards. However, the code has not been tested yet due to not having the SD card as well as the SD card shield port required to connect the SD card to the microcontroller

Present Work:

* Creating test code for transferring WAV audio files into the Digital to Analog Converter Channel 1 via the I2S peripheral
* Purchasing the appropriate USB adapter and SD card shield port equipment needed for testing the USB and SD interfacing test code found in previous work

Future Work:

* Creating test code that can allow the microcontroller to read text, chapter headers, and page numbers from html files stored in a USB flash drive into an empty array, chapter header variable, and page number variable respectively
* Creating test code that can allow the user to begin transfer of html data from one shift register to another when the play button is pressed once and stop transfer of data when pressed again
* Creating test code that stores the following user-adjusted values into the microcontroller’s on-board Flash Memory for saving user settings as well as optimizing accessing time

1. Translation\_Mode: An int variable of a single bit that is 0 for contracted braille and 1 for uncontracted braille, where 0 is the default value unless changed by the user via the mode button.
2. Chapter\_Heading: A string variable that holds the name of the Chapter Heading that the user is currently reading from. This variable defaults to the very first chapter heading, which would be the title of the book, if not incremented by the user via the chapter select buttons.
3. Page\_Number: A string variable that holds the page number that the reader is currently reading from. This variable defaults to the very first page of the chapter that the user is currently reading from and automatically changes to the next page number when the user moves on to the next page.

* Creating test code for an incrementing push button as well as a decrementing push button that will be used to increment or decrement an int variable by a certain number. This type of push button will be used for the file/chapter select buttons
* Creating test code for the Play button that checks if the user is able to use the play button to
* Creating test code for the Position Control sub-block that will do the following:

1. Test whether the microcontroller is able to accurately update the value of the Chapter\_Heading and Page\_Number variables due to the user adjusting them via the file/chapter select buttons
2. Test whether the microcontroller is able to update the Chapter\_Heading and Page\_Number variables via an interrupt service routine that updates the two variables when the text advances to the next chapter via the play button
3. Test if the where am I button is able to prompt the microcontroller to transfer the correct audio files corresponding to the current chapter and page number values directly to the DAC via the I2S peripheral’s reception and transmission buffers

* Creating final code that incorporates all my subsystem’s code into one integrated main function that functions as intended with appropriate interrupt preemptive priority settings