Design of a Prototype Digital Sheet Music Folder

Project Number: MQP-SMJ-B01

A Major Qualifying Project
submitted to the faculty of
WORCESTER POLYTECHNIC INSTITUTE
this 26th of April, 2012
in partial fulfillment of the requirements for the
Degree of Bachelor of Science.
Submitted by:

Kari Rehkugler, Electrical Engineering _____

Geoffrey Hook, Mechanical Engineering _____

Date: April 26th, 2012

Approved by: _____

_____

Keywords:

1. e ink display
2. music folder
3. serial peripheral interface

Abstract

The tradition of reading printed paper sheet music in choral and instrumental settings has not changed for thousands of years. This project endeavored to build a prototype device using e-Ink electrophoretic displays to display sheet music electronically. The device had to look as similar to a choral music folder as possible. A successful prototype was built to load a page of music on two sides of a folder, and to turn pages when a button is pressed. In the future, this device will be streamlined and enhanced for ease of use and excellence.

Acknowledgements

Professor Susan Jarvis

Professor John Delorey

Daniel Miller

Matthew Goon

Brian Grabowski

Austin Noto-Moniz

Nomenclature

SPI	Serial Peripheral Interface
WPI	Worcester Polytechnic Institute
GPIO	General Purpose Input/Output
EPD	Electrophoretic Display

Table of Contents

List of Tables and Figures

1 - Introduction

Human's method of reading music has not changed in thousands of years. At the origin of the species, they learned by rote, or by ear. One person would sing the song, and everyone else would follow along. As civilization evolved, the music that was sung or played got transferred to symbols to represent the different notes, first very basic but as time went on it got more specific, until it reached the music known by the world around today. These notes are printed onto sheets of paper, which anyone can pick up and read, if they know how. Paper sheet music has been the method for coordinating orchestras and choirs for thousands of years, and it has not been updated to the current times. This needs to change.

This project is the new generation method of distributing, organizing and reading sheet music. Instead of putting sheets of paper with the music into a folder or a book, digital copies of the music can be uploaded onto this device and read from paper-like displays. It is still the same sheet music, but in a new and improved form. The digital copies do not need to be printed at all, so it saves paper and the environment. Composers can also distribute the music they write themselves over the internet, so they don't have to give a cut to a music distributor. This product could revolutionize the sheet music industry.

The team working on this project is very involved with music, and both members sing in a choir. Using the knowledge gained while singing with the group, specifications were made for how the device had to work, look and feel. It opens just like a conventional choral folder, and has a page of music on both sides. It uses a special display made by e-Ink called an electrophoretic display, which is what is used in a Kindle. This means it does not have a backlit screen like a computer or tablet, but instead uses ambient light to show the page. It looks just

like paper, so changing from paper music to this device will be easy and simple for choir or orchestra members. The interface is intuitive and simple, so anyone can use it.

This paper will discuss the method of designing the device, from inception to completion. The thought process used will be made clear, so the reader understands what the team was thinking when they chose to do something a certain way. It will start with some background research about music reading and the electrophoretic display (EPD) technology, and go into the hardware and software design, as well as how the case was designed. Finally, it will discuss the marketability of this product were it ever to be made en masse, and explore what future renditions might include in the Digital Choral Folder.

2 - Background Research

2.1 - Related MQPs and IQPs.

Some of the teams most valuable research resources were previous WPI MQP papers. The concept behind the Digital Choral Folder project was first explored by Jamie P. Barriga in 2008 in an MQP entitled “Digital Choral Folder”. The major points of Barriga’s project were the effects of page turn delay and method on choral singers comfort. Through a study of 95 singers on a basic two-screen testing rig, Barriga found that a majority of singers (about 70%) prefer the traditional “book-style” paging method (wherein pages 1 and 2 would be replaced by 3 and 4) vs. a “sliding-style” method (wherein pages 1 and 2 would be replaced with 2 and 3 on a page turn). Barriga also found that singers could tolerate up to a 500 millisecond delay between button press and page turn before it because noticable and uncomfortable. Although Barriga’s MQP provided good background information, the project failed to produce a proposed system design or

prototype.

The Digital Choral Folder project was revisited in 2009 in Joeseph McCarthy's MQP entitled "Design of a Digital Choral Folder". Unlike Barriga's 2008 MQP, McCarthy's MQP focuses much more on the technical aspects of the device. McCarthy's proposed design centers around the "Metronome" controller provided in an eInk development kit purchased for the project. The Metronome controller is a Gumstix-based board that hosts a customized Linux kernel and drives the Vizplex line of eInk displays. Along with the Metronome, eInk also included a "tinyX" Linux distribution and "Thinspace" eInk display API. McCarthy proposed using the included Linux kernel along with some open source software to allow the system to read PDF files and print them to the eInk display. Though McCarthy came far closer to a workable solution in his MQP than Barriga, he was ultimately unable to produce a working system.

Since McCarthy's MQP in 2009, this project is the first to revisit the concept. After researching both MQP reports and acquiring the original eInk development kit from McCarthy's MQP, the team found some good conceptual suggestions but very little usable technical information. Aside from the fact that McCarthy's MQP was heavily redacted, eInk display technology evolved rapidly between 2009 and 2012. Consequently, the Metronome board and display are obsolete and are not worth pursuing currently.

2.2 - Electrophoretic Display Technology

There was also other research that had to be done before starting this project besides prior art. The previous MQP had a lot of research but it was outdated information, and this project was probably going to use a different processor board and newer e-ink technology. Therefore, the team had to do some research on other options for the board and how it could be connected to

the papers, as well as other specifications on how the folder had to work and look.

E-ink uses an electrophoretic display to make their paper. Electrophoretic displays use submicron particles suspended in a dielectric fluid-filled capsule to differentiate between white or black. The capsules are held between two sheets of electrodes to separate pixels. A cross section of the capsules is shown below. [4]

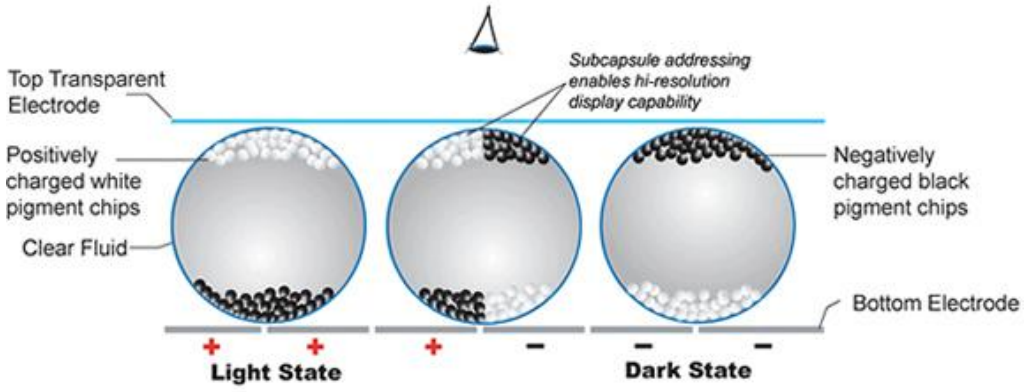


Figure 1 - Cross section of EPD display [4]

Inside the capsules, the white particles are positively charged, and the black are negatively charged. When a negative electric field is applied across it, the white particles move to the top of the capsule to be visible by the viewer, and the black move to the back. The particles can be "printed" onto a variety of surfaces, with laminates on either side and some tiny transistors in the center to create enough charge to change colors, and the liquid in the center with the capsules suspended in it. Eventually the ink will be able to be printed onto just about any surface. [4]

2.3 - Prior Art

There are a few products on the market that are similar to the digital choral folder, but none that provide all the same functionality. The most relevant product is the Prodigy Digital

Music Stand, a music stand with an LCD screen in the place of music. The Prodigy Digital Music stand is shown below in Figure 2.

Figure 2 - The Prodigy Digital Music Stand [10]

The screen displays two pages of sheet music at once. The pages can be turned by a wireless foot pedal. However, it can only be used as a music stand, and cannot be held in a choir singers hands. It is only usable for band members and instrumentalists, and this project was specifically aimed toward choral singers. The other problem with the Prodigy stand is that the screen is a backlit LCD screen, and the glow from the screen reflects off the faces of the people reading from it and it can be very distracting to the audience members.

There are also some computer programs and tablet apps that can be used to read music. Tablet apps have touchscreen and note-taking capabilities using a stylus or finger, which is very attractive to most choral singers who want to take notes on their music. However, these programs can usually only display one page of music at a time. Most paper pieces show two at a time, and having only one visible is a detriment to the effectiveness of choir singers. Bringing a computer on stage is also not an option for most concerts, so only the tablets can be

used. However, these have backlit LCD screens just like the Prodigy stand, and the blue glow on the members of a choir is very visible and distracting to the audience, and should be avoided. Taking out all devices with an LCD screen leaves only devices with e-Ink screens, mainly the Kindle. The Kindle does have PDF capabilities and can display sheet music, but the screen on a Kindle is very small. The notes can barely be seen clearly, or the music takes up more than the full screen and the user has to scroll around on it. It can also only display one page at a time. While all of these devices have some of the capabilities of the Digital choral Folder, and some have more, none of them have the correct combination of functions. This project required a handheld device with an e-Ink screen and two page display capabilities, and nothing else on the market has that.

3 - Project Goals

3.1 - Project Objective

The goal of this project is to design a prototype digital music folder for use in choirs and bands. The project arises from a number of apparent needs, including the cost of professionally printed music, the time required to organize and distribute physical copies, and the desire to reduce paper use.

3.2 - Design Specifications

The Digital Choral Folder had a few specifications that were non-negotiable. It had to open like a folder and have one page of e-Ink paper on each side. It had to have page-turning capabilities, like turning pages in a piece of music. It also had to be black, because most music folders are black to blend in with the normal concert attire. Some other things the team aimed

for were lightweight, so it is easy to hold up, and easy to use. The page transitions also had to be as quick as possible, and the folder has to hold at least as many pieces as a full length concert.

These qualitative specifications are translated to some easily identifiable quantitative specifications. The folder had to have a hinge in the center, with a piece of e-Ink paper in each side and small borders around the edges, ideally a half inch each. The folder had to be hollow to hold everything inside and hide the electronics, but also thin enough to be easy to hold. The goal was half an inch thick on each side or less, so the whole folder is an inch thick. Ideally, the folder would be much thinner than that, but as a prototype half an inch per side is fine. There had to be easily accessible buttons for the user to press and turn the pages. The memory card had to be at least 1Gb in order to hold any software as well as plenty of music files, but much bigger memory cards are easily available.

The most difficult specifications to meet were lightweight and quick transitions between pages. The quick transitions could be made with a fast processor, but an appropriate one had to be chosen. Lightweight was another big issue, because the folder ideally had to fit the processor board inside so it had to be thick, which made the half-inch sides very difficult. The material it was made of was also a consideration, because it had to be light but also durable and hopefully water-resistant, just in case. Many options were considered for both the processor and the folder material.

Larger display size than Amazon Kindle DX (8” diagonally)
Two displays
Total weight less than 2.0 lbs.
Overall size less than
Capable of withstanding stresses from daily use
Functional battery life greater than 24 hours (in use)
Central folding hinge to mimic traditional folder feel and function
Slim and sleek form

Noiseless operation
Data storage capacity of at least 1 gigabyte
Visually and structurally simple user interface
Four-button operation (home, select, page right, page left)

Table 1 - Design specifications

3.3 - Initial Design

The first consideration for the processor was the Gumstix processor, which was used by the previous MQP. However, the team decided this did not have the processing power that was necessary, nor did it have the memory. In further research, some others were looked at a little bit, but almost immediately the Freescale i.MX50 processor was found, and it specifically had an e-Ink connector and capabilities. Other processors were researched but no others had e-Ink capability already built in, so the team quickly decided on the i.MX50. One of the options for purchase was the i.MX50EVK, or evaluation kit. It consisted of the i.MX50 processor and a power management IC (PMIC) that was already set up for it, as well as all the connectors that were commonly used with the processor like an HDMI port, analog video, Ethernet port, USB, and many others, including an LVDS port for e-Ink uses. The onboard memory is a 4Gb SD card. It also had buttons, indicators and switches, and it came with a CD to load Ubuntu linux onto the SD card. Although with its additional peripherals the i.MX50EVK was taller than ½", this was the clear choice for processor board.

Other things that were considered was the fact that choirs do not only need one folder, they need many. This meant that for the Digital Choral Folder to be a viable option for choirs, there had to be the possibility of having many of them. Part of the perk of the digital folder is ease of uploading and organizing music, and being able to sync many folders. This brought up the idea of a full-scale folder case, with slots for many folders to sit in and be uploaded to. This

could also be where it is charged. Knowing that a larger scale system was the eventual goal, the team continued to consider options.

The case design was another thing the group had to consider carefully. It had to look as close to a conventional folder as possible, so it was not distracting or too difficult to get used to. This meant a folder with two sides and a hinge in the center. It had to be light, as stated earlier, but there were many options for materials. For maximum durability aluminum was considered, but knowing that the project had limited time and budget, this was ruled out. An aluminum case would also have been the heaviest option. Plastic was the next option. Almost any material property can be made with plastics, so the most important properties had to be decided on and a good plastic chosen. Delrin seemed to be the best choice after some consideration, but it would have been very difficult to work with and form into a case. Eventually, very thin acrylic backing and front plate, and a thicker middle gasket, was the best and easiest choice.

4 - Design

4.1 - Hardware Design

4.1.1 - Hardware Overview

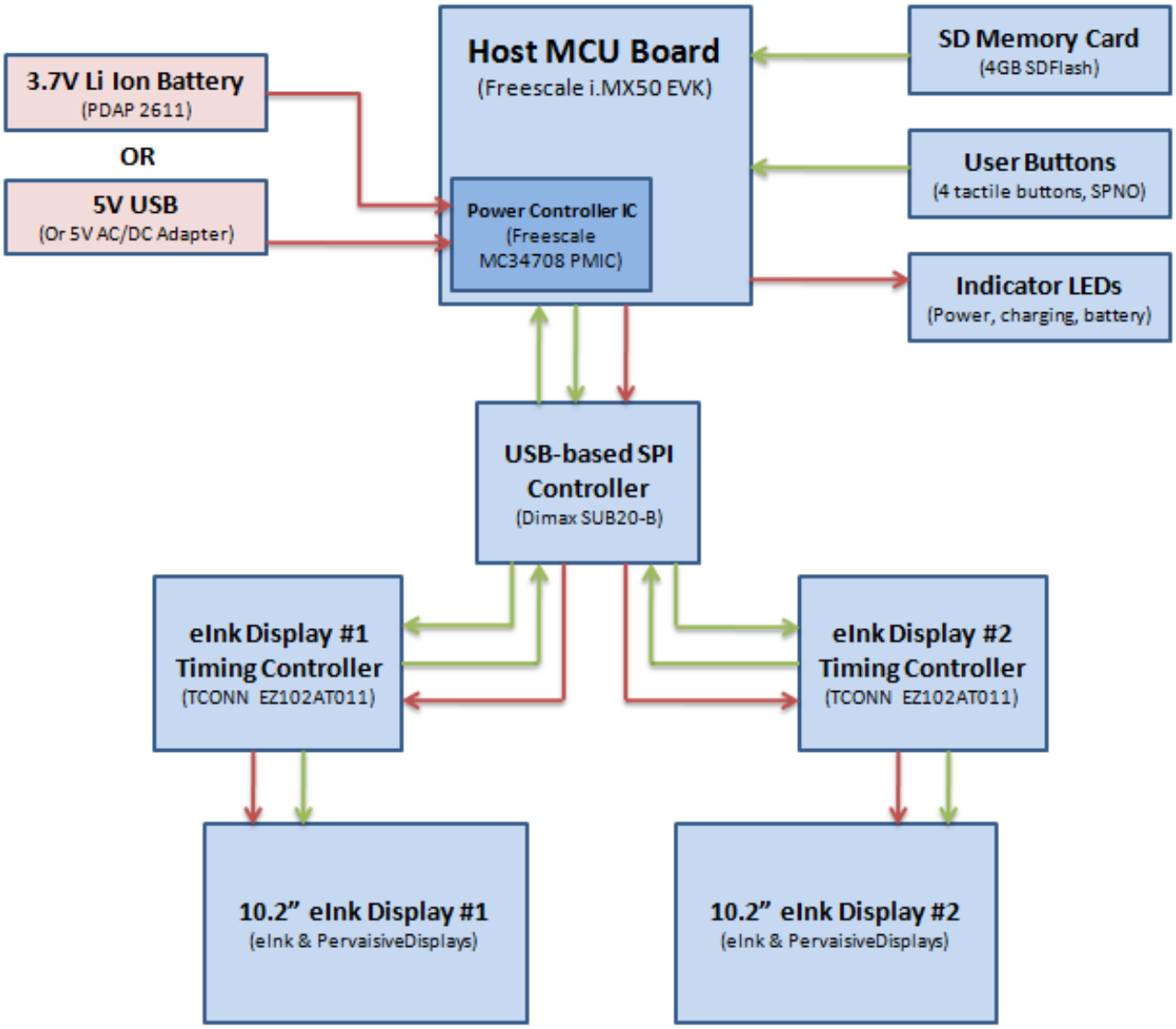


Figure 3 - Hardware block diagram

4.1.2 – E-Ink papers

The electrophoretic displays from e-Ink were actually purchased from a distribution company called Pervasive Displays. The screen surfaces are 10.2" diagonal, and have a small flexible circuit on the top to help control the signal from the timing controllers. They take in a simple stream of bits to show the images, either 1 for white or 0 for black. All of the necessary configuration to go from a simple waveform to an image is done in the timing controller, so the panels are very simple and easy to use.

4.1.3 – Timing Controllers

The timing controllers are a small printed circuit board from e-Ink that connects to the EPD through a 40-pin connector. It has a special SPI connector it uses to connect to other devices and take in the bit stream. Within the timing controller, it takes care of all the configuring and timing between the input waveform and the e-Ink panels, so the user does not have to worry about the timing. Besides the very small SPI input connector, the timing controller was very simple to use.

4.1.4 - Freescale i.MX processor

The i.MX50EVK was an excellent choice for its processing power, but the one issue with it was that it is very large, and putting it inside the folder would either necessitate de-soldering some of the connectors or making the folder very thick and therefore heavy. The team considered both of these options, but finally decided to instead make a tether for the folder attached to the board. Inside the folder goes the e-Ink papers and the timing controllers, and outside attached to a cable is the board. This means the folder is not portable, but for this prototype this was considered acceptable.

4.1.5 - xDimax USB interface crosspoint

The xDimax Sub20 USB device is described by the manufacturer as being a multi-function USB-based interface crosspoint. The device allows a user to send and receive data from any supported board interface (USB, SPI, I2C, UART, among others) to any other supported board interface. For this project in particular, the Sub20 was used to provide an interface between USB (from the host computer) and SPI (to the display timing controllers). In addition, the team also utilized GPIO functionality on the board to read input from the buttons and logic signals from the timing controllers. The Sub20 board layout is shown below in Figure 4.

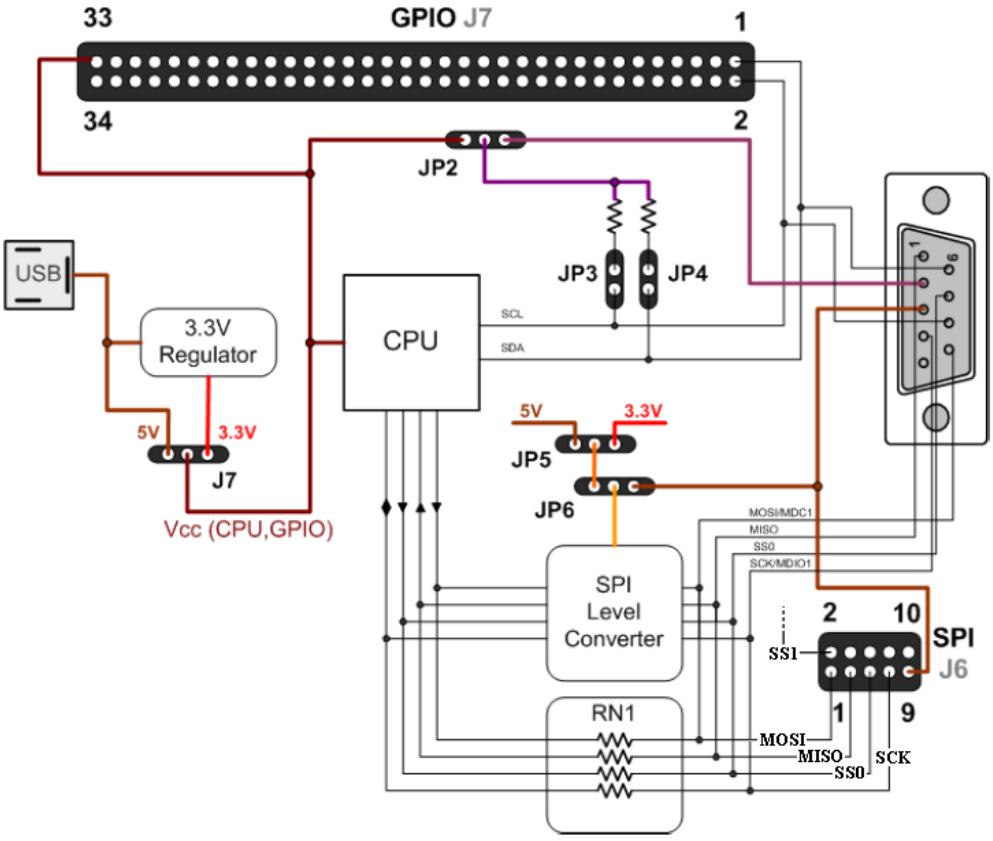


Figure 4 - xDimax Sub20 I/O Layout [11]

4.1.6 - Battery charging circuit

Despite having a tether in this rendition, future folders will be run from a battery. This necessitated a charging circuit for the battery. As the e-Ink panels take very little power to run, but the folder needs to be able to be charged all night without any adverse effects, it needed to have a slow and fast charge capability. Knowing that the Lenmar 3.7V lithium-ion battery was being used, an integrated circuit specifically for charging lithium ion batteries between 4.5 and 12 volts was found. The ADP2291 is an IC made by Analog Devices, and it has both fast charging and trickle charging built in. Depending on the values of the resistors and capacitors outside the chip, the times and values could be changed for charging. Around 5V was desired, and a long trickle charge time. It also has many other perks, like overshoot protection and a very small leakage current, as well as the ability to light an LED when the circuit is charging. As the typical values shown below were what was needed, this typical application circuit from the datasheet is what was used.

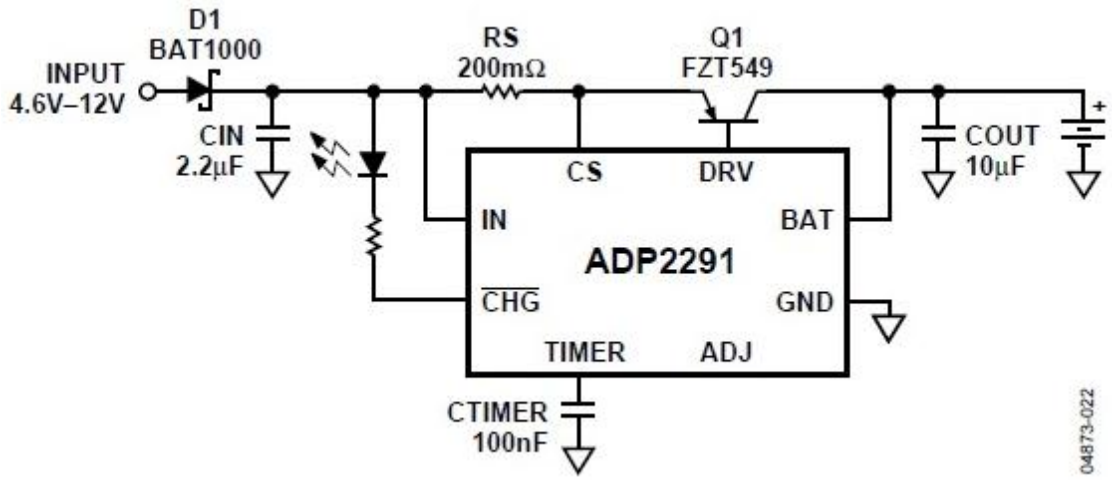


Figure 5 - Typical application circuit for ADP2291 [1]

This circuit charges the battery at a voltage of less than 5V and a current of 750mA during fast charge. Once the battery is full, the voltage and current go low but continue to charge

with a very slight trickle until a time is reached, determined by the equation:

$$t_{CHG(mins)} = C_{timer} * {}^{1800\ mins}/_{1\mu F}$$

Equation 1 - Trickle charge time

Using a 100 nF capacitor as CTIMER, the trickle charge time would be approximately 180 minutes.

4.1.7 - xDimax I/O configuration

The Digital Choral Folder needs at least 4 buttons to navigate through the menus. The xDimax provides 32 general purpose I/O pins which can be used to implement the buttons. The GPIO pins on the xDimax device were relatively easy to program as they were needed. Using functions from the API, they could be configured to be an input or an output, and then either high or low. The GPIO configuration for this project is shown below in Table 2.

GPIO #	Function	I/O Direction
0	Home button	Input
1	Select button	Input
2	Page Right / Up	Input
3	Page Left / Down	Input
4	Button source	Output (default Hi)
5	Display #0 busy pin	Input
6	Display #1 busy pin	Input
7	“ON” pins for displays	Output (default Hi)

Table 2 - GPIO configuration

GPIO 4 and 7 were configured as outputs and written to always be logic high. The rest were kept as inputs to be checked within the code. It should be noted that on the xDimax device, there are 32 GPIO pins, GPIO0..31, but pin number 1 does not correspond to GPIO1. In fact, the

pin numbers have almost no correlation to the GPIO number, and they all had to be looked up in the manual.

4.2 - Software Design

4.2.1 - Software Design Overview

The control software for the Digital Choral Folder was coded and compiled using Microsoft Visual Studio 2010. The program was coded in the C programming language partially because C is platform independent, but mainly because the Application Programming Interface (or API) for the xDimax device is language dependent on C. Code had to be written using the functions from the Sub-20 API to properly send data over SPI and get an image on the display. While the program was originally meant to run in Ubuntu Linux on the Freescale processor, the team found it was easier to test functionality in Windows XP on a laptop PC. The Sub-20 API and drivers work well with Windows, and xDimax provided a precompiled debugging tool (called subtool.exe) that can be used to send and receive data from the Sub-20 device. The Sub20 tool was very useful in debugging the DCF control program as it allowed the team to directly read data from the device hardware while running the program. Although the DCF software was written and compiled in Windows XP for the project prototype, the code would require very little modification to adapt to running on the i.MX50 platform in a commercial device. When the compiled control program (DCF.exe) is executed on the laptop, the Sub20 device and eInk displays are initialized and the user interface routines are started.

4.2.2 - xDimax API

An Application Programming Interface, or API, is a source-code based specification that

facilitates easy communication between software components. In order to communicate with the Sub20 device, the control program utilized functions from the Sub20 API provided in the xDimax software package. To access the Sub20 device and utilize the API functions in a program, the Sub20 USB driver must be installed on the host system and a C header file called "libsub.h" must be included in the program source. Though the API includes over 40 unique device functions, the control program utilized only seven. The relevant API functions are detailed below.

sub_open()

sub_reset()

sub_spi_config()

sub_spi_transfer()

sub_gpio_config()

sub_gpio_write()

sub_gpio_read()

4.2.3 - DCF.exe Program Functionality

The program that controls the DCF is contained in a single executable file called "*DCF.exe*" on the host device. The program and its resources are contained in a folder called "*/DCF/*". The program is compiled from a C source file called DCF.c, When the program is executed, the main function in DCF.c is called. Local variables, global variables, and pointers are first defined and declared. After the definitions, the program calls the *initializeDCF(...)* function. The *initializeDCF(...)* function performs the following 7 actions.

1. Find and open Sub20 USB I/O controller

First, the Sub20 device is found and initialized using the "sub20*handle = sub_open(0);*" function. The open function internally calls "*sub_find_devices(0);*" to identify the first Sub20 on the system, opens and initializes the device, and returns a nonzero handler to "*sub20handle*" to identify the device in subsequent functions. Once the handler has been returned the Sub20 device is ready to use. On error, the function returns a specific error number and prints error text.

2. Enable Sub20 for SPI master, configure SPI protocol for eInk displays

The Sub20 device is enabled and configured for SPI master mode using the "*sub_spi_config(...)*" function. Overall SPI settings (such as sampling mode,clock speed, etc.) are set using specific flags (such as *SPI_SETUP_SMPL* and *SPI_CLK_500KHZ*) passed to the function. On a successful configuration, the function returns "0". On error, the function returns a specific error number and prints error text.

3. Enable GPIO pins on Sub20, declare as inputs or outputs

Third, the IO pins on the SUB20 device are declared and configured using the "*rc = sub_gpio_config(...)*". To set inputs and outputs, 2 four-byte hex numbers (between 0x00000000 and 0xFFFFFFFF) are passed to the function. Each binary bit in the two hex numbers (bits 0..31) corresponds to a general purpose IO pin on the SUB20 device (pins GPIO0..GPIO31). For example, the GPIO19 pin corresponds to the 19th bit of the two input numbers. To effect GPIO7, GPIO11, and GPIO23, the input number would be hexadecimal "0x00800880" or binary "00000000100000000000100010000000".

Each bit in the first input number (the direction number) specifies the direction of an IO pin. If the bit is “0” the pin will be an input; if the bit is a “1” the pin will be an output. Each bit in the second number (the mask number) specifies whether or not the corresponding IO pin will be enabled with the direction specified by the first input number. More specifically, if a pin is not yet enabled when “*sub_gpio_config(...)*“ is called and its corresponding mask bit is “0”, the pin will be left disabled. If the pin is already enabled and its corresponding mask bit is “0”, the direction of the pin will not be changed. If a mask bit is “1”, the corresponding pin will be enabled and configured to be an input or output based on the corresponding bit in the direction number. On a successful configuration, the function returns “0”. On error, the function returns a specific error number and prints error text.

4. Open left/right default menu bitmaps, read into buffer arrays, close files

Fourth, the startup menu bitmap images are loaded into memory. The bitmaps are opened as “file” using f*open(...)*, then loaded into the left and right image buffer arrays using *fread(...)*. Once the images have been stored in the arrays, the bitmap files are closed using *fclose(...)* to avoid memory leaks. If the images load successfully, header and raw pixel data can now be read from the buffer arrays.

5. Call *sendImage(...)* to send default menu images to displays, clean up variables

Fifth, the startup menu images are read from the image buffer arrays and sent to the displays via SPI transfers. The *sendImage(...)* function is called and passed a pointer to first image data byte in the left image buffer array. The function then writes the image to the left display using the Sub20 function *sub_spi_transfer(...)*. The *sendImage(...)* function is then called again for the right display. Once both displays have been

successfully updated, the variables, pointers, and image buffer arrays used during the image update are cleared. This is done to avoid memory leaks and to prepare the program to update new images.

6. Start button listener routine to read state of user input buttons

   Finally, the button listener routine *buttonListener(...)* is called. At this point, the device is fully initialized and ready to receive user input. The button listener routine runs continually and monitors the state of the button GPIO pins. When the user presses a button, the listener routine reads the GPIO pin and calls image updates based on the menu state and currently selected music piece.

Using these steps, data was sent through USB from the laptop and converted to SPI through the timing controllers into the page. The full code can be found in Appendix B.

4.3 - Case Design

4.3.1 - Case Overview

One of the major goals of this project was to create a prototype that approximates a real commercial device. A functional assembly of electronics may prove the concept of operation, but it does not fully explain the device design. Thus, a case enclosure was needed to house the electronics and provide a general form for a future commercial product. Based on the project's initial design specifications, the team designed and fabricated a lightweight and thin acrylic plastic case. Fabrication of the case was performed largely on the laser cutting mill in Washburn shops at WPI, while assembly was performed in the MQP lab in Atwater-Kent. The prototype case was designed with a "clamshell" style to allow for quick disassembly and access to the internal electronics.

4.3.2 - Material Choice

Initially, the team planned to use a vertical CNC mill to fabricate parts from plates of thick Delrin plastic. The team selected Delrin as it is light, strong, non-reactive, and has good machinability. However, due to project time constraints the group decided to save time by using acrylic plastic and the Washburn Laser cutter to fabricate parts. While acrylic plastic is not as resilient as Delrin, it can be cut easily on the laser cutter with very little set up time. In addition, acrylic pieces can be bonded to other plastics by using common super glue (cyanoacrylate). Though the prototype was constructed from plastic, the team also considered aluminum as a possible case material for a future commercial device. Important physical properties for all 3 case material options are shown below in Table 3.

	Delrin (Polyoxymethylene)	**Acrylic** (Poly-methyl methacrylate)	**Aluminum** (6061 alloy)
Density	0.0556 lb/in³	0.043 lb/in^3	0.0975 lb/in³
Hardness	R120	M93	B60
Tens. Strength	8800 Psi	7000 Psi	45000 Psi
Opacity/Color	Opaque, off-white	92% translucent	Opaque, silver
Mach. Process	Vertical CNC Mill	Laser CNC cutter	Die-cast molding
Melting Point	320 F	212 F	1100 F

Table 3 - Physical properties of case material options

4.3.3 - CAD/CAM Models

These are the CAD models.

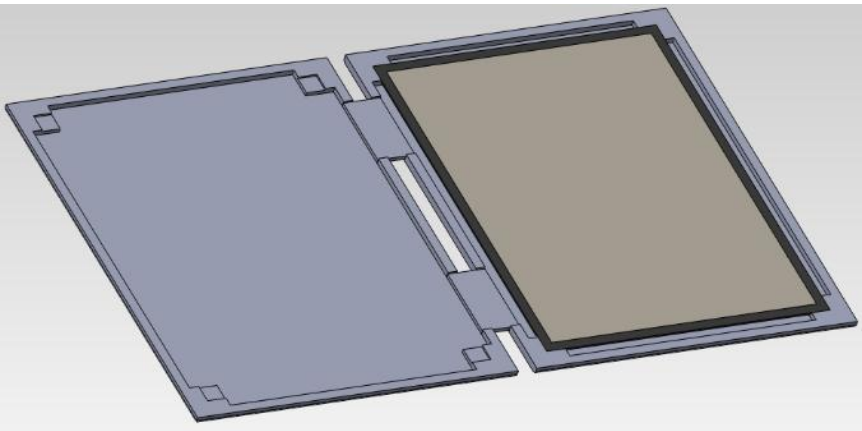


Figure 6 - Solidworks assembly model of case (disassembled)

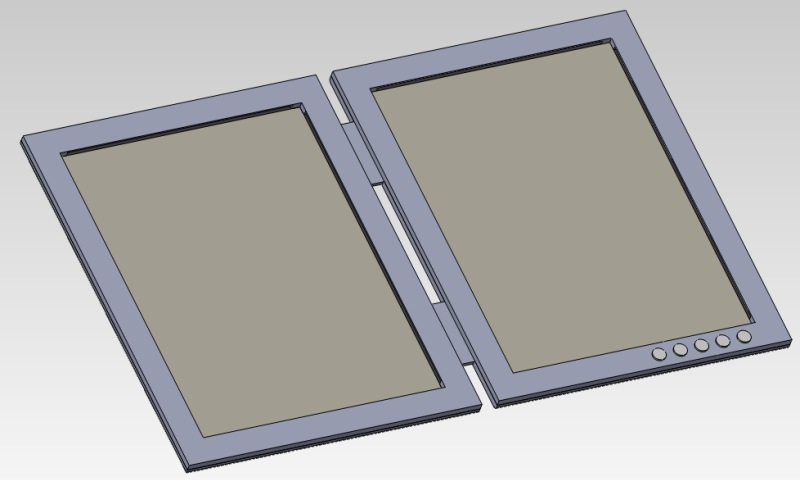


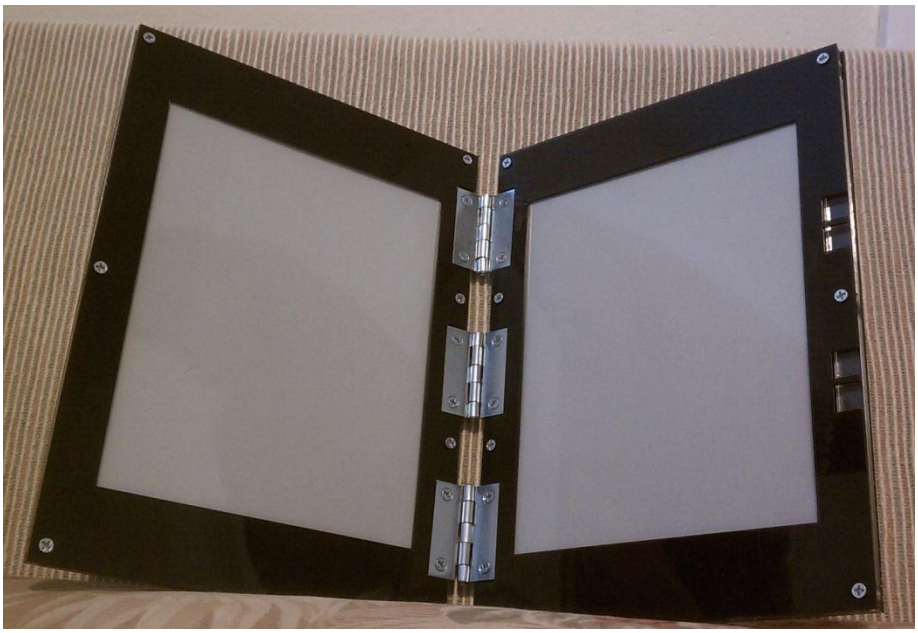
Figure 7 - Solidworks assembly model of case (assembled)

4.3.4 - Fabrication Process

After completing the solid model of the device, the parts were simply exported into drawings and loaded into the laser system's control software. Below is a photograph of the laser cutter in the process of cutting our gasket plates.

Figure 8 - Washburn laser cutter processing case parts

The laser cutter allowed us to fabricate precise parts with very little setup time. Additionally, it allows the group to easily produce additional parts and cases if need be. Below is a photo of the assembled case parts after being cut. Note that the blue coloration is from the protective plastic backing.



5 - Testing and Market Research

5.1 – Testing

The battery charging circuit was simple to test. The circuit was built on a breadboard, and 5V put into the input. It successfully put out around 4.2V and 700mA, and the LED came on when the voltage at the output was low and the circuit was charging. The circuit was then transferred to a solder board and soldered together, and when tested again it worked perfectly again.

The program that was written for SPI configuration, GPIO configuration, data transfer and page turning was somewhat difficult to debug and test. First it had to correctly compile, which took some time to debug. Some simple mistakes were made that were easy to fix, but some issues arose from incorrect includes and linking. There was also many issues of incorrect pointers, improper memory allocation, and trying to send an incorrect data type into a function.

Once all of those were fixed, the code could be compiled but when ran it came up with an Unhandled Exception. This meant that a null pointer was being de-referenced, but it was not immediately clear where this was occurring. To fix this, the team added printf() statement after each section of code to see where it was getting stopped. The code output the statements until it broke, and then it became clear that the GPIO configuration line was the issue, and this was then fixed.

The hardest thing to test was the SPI data transfer. From the visual UI, sub-20 tool, and the typical application examples in the API, the general idea was understood but some specifics had to be changed and tested in order to find the correct value. For example, it was unsure whether the device sampled on the rising or falling clock edge. The timing diagrams from Pervasive Displays looked like it sampled on the falling edge, but it was not clear. The program that was written also may have been incorrect, despite compiling, so there were many things to debug before a good picture was sent to the display.

At first, no picture was sent, but it soon became clear after studying the testing setup that GPIO5 is not pin 5 on the GPIO, its pin 22. Once that was fixed, the connector between the e-Ink panel and the timing controller had to be studied, because the connector between them is symmetrical but the pins are not reversible. Finally, the SPI power was not working correctly, so this had to be fixed as well. Once all the issues with the testing setup were fixed, then the data transfer could be tested.

The first picture sent from the program was nothing like the picture it was supposed to send. This is the picture being sent to the display.

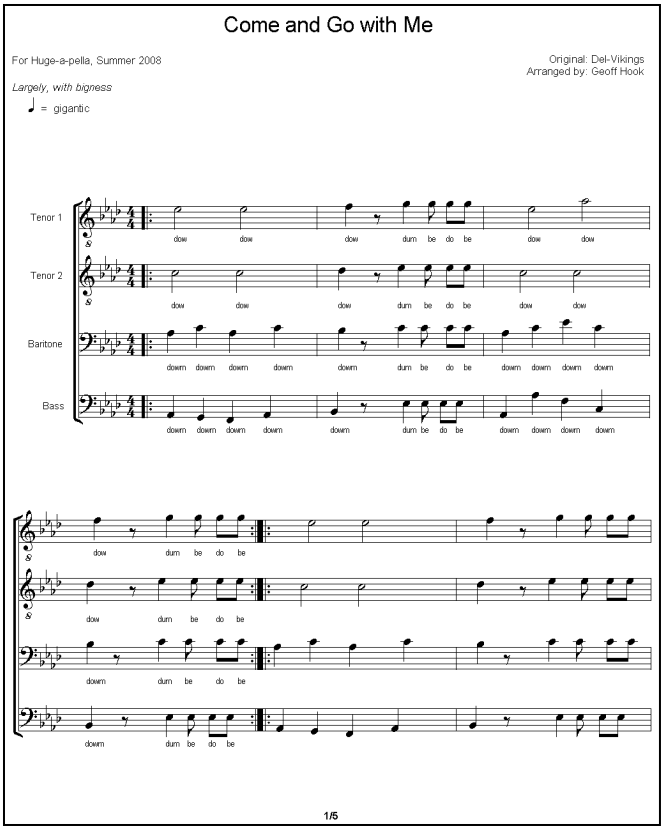


Figure 9 - Testing Sheet Music Sample

This is what was output.

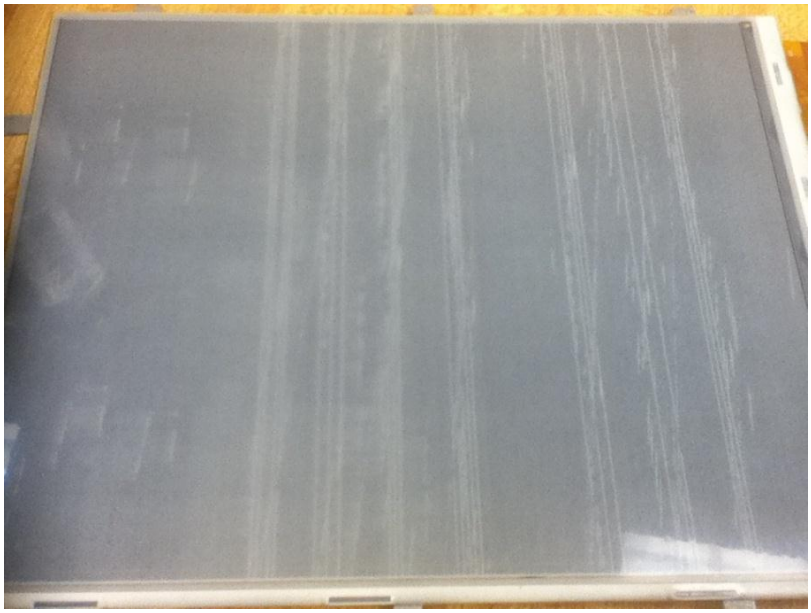


Figure 10 - First Test Output

The ledger lines are somewhat visible, but there are clearly no notes, and there are artifacts all over the page. Debugging this took many hours of confusion and trying things out, but eventually it was discovered that the timing controller samples on the rising edge, not the falling edge, and the sampling was also incorrect. The bits also had to be inverted, because the background was black and the music was white. The bits had to be inverted, which was a simple fix. After some work, this is the final output picture on the panel.

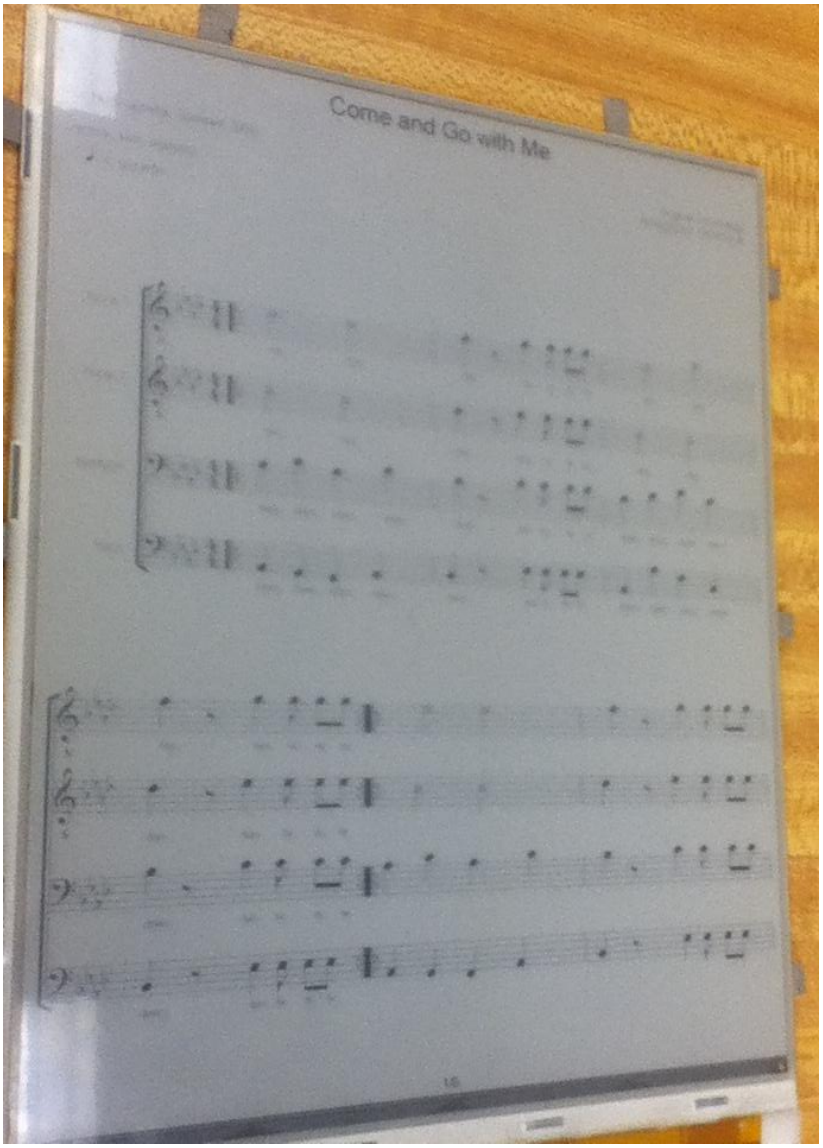


Figure 11 - Final Working Image

5.2 - Market Research

5.2.1 - Providence

The prototype was taken to the American Choral Directors Association (ACDA) Conference in Providence, RI. While there, the team showed it to the conference goers and surveyed their reactions to it. Many were just passing by so did not want to take a full survey, but did discuss it with us. All reactions were positive. Every person we talked to thought it was a very interesting idea, connecting music with technology. They all agreed that having the e-Ink panels was better than having an LCD screen because it was not backlit, but some expressed concern that it would be too dark to see it in concerts. These people though a flip-up LED was a good idea. One band director made some good points about how the folder was geared towards choruses, and how bands would find it less useful.

6 - Future Work

The first thing that should be done for this project to become a viable and marketable product would be to contain all of the hardware within the case. The i.MX50EVK is a useful and powerful board that facilitated prototyping, but it has unneeded components that conflict with the desired board height requirement. The team chose it for the EPD capabilities, but then ended up not using them. All that is necessary is a board with fast processing and a lot of memory, and a USB port. The i.MX50EVK was overkill for this purpose. It might still be the best option, but more research should be done.

Once a chip is chosen, only the chip should be used and a printed circuit board designed specially for it to keep it as small as possible. The EVK is too big and heavy to put inside a case, but keeping the folder portable is a very important factor, so everything has to fit inside. The

battery can be a cell phone battery so it is powerful enough to power the board but small enough to fit flat inside the case. The sub20 SPI converter is also large, but if the analog video connector is removed it will fit inside the case as well. Once the folder is fully portable and as light as possible, other interesting features can be included.

The first option that would be important and easy to implement would be a wireless foot switch for orchestra players to turn the pages. While a choir member can simply press a button and continue to sing, a violinist cannot easily reach over and press a button to turn the page. Having a footswitch to turn the page would obviate this issue, and help members of an orchestra play better. This was the most interesting option to many of the attendees at the ACDA conference.

Another small and easy feature would be a pop-up LED to light up the music in dark venues. A pianist during a musical performance often has a light over their music so they can see it even during blackouts, and this would be analogous to that. It would be a small light that can be popped out of the case, and folds back into it neatly when not in use. It would have to be very small and preferably a warm white LED so as to not distract the audience very much, but still light up the page enough to see it in the dark.

One significant issue with the folder is the lack of note-taking capabilities. Many musicians like to take notes on their music so they remember things about performing it, and with the Digital Choral Folder this is not an option. However, e-Ink technology is growing quickly, and it is very likely that a touchscreen e-Ink panel will come onto the market in the next few years. This means that future renditions of the folder can be made with the ability to take notes on the page of music and save it to each folder individually.

The team's vision for the future of the Digital Choral Folder consists of a bank of folders, just like a normal choir folder bay, with a slot for each folder to be inserted. When plugged in, the folders would charge, and they would have the ability to be updated from a computer built into the bay. On the computer would be a program to put pieces of music on all or some of the folders in the dock, so that every one would always be the same and fully updated. A choir would buy 50 or more folders and a docking station, and obviate the need for someone to organize the folders almost every rehearsal.

7 - Conclusion

Overall, the team was able to achieve the established project goals. The most important goal of the project was to dynamically display sheet music on two eInk displays. Though there were many delays and issues in its development, the team was able to create a fully functional final prototype. The second most important goal of the project was to create a prototype that approaches the form of a real commercial device. The team was also able to meet this expectation and produced a good approximation of a traditional sheet music folder. Although the final project prototype is tethered to a laptop, the system the team developed is platform independent and can be easily applied to a commercial prototype utilizing the Freescale i.MX processor.

After working with it for many months, the team believes that the Digital Choral Folder is a novel, realistic, and marketable device concept. Based on project results, research of current eInk technologies, possible target markets, and musician testimonies, the Digital Choral Folder has real potential to modernize the way musicians purchase, organize, and handle sheet music.

The team thoroughly enjoyed working on this project and hopes that the Digital Choral Folder project will someday be available to singers and instrumentalists worldwide.

Works Cited

[1] "ADP2201." *ADP2291 Datasheet*. Analog Devices. Web. 13 Dec. 2011. <http://www.analog.com/static/imported-files/Data_Sheets/ADP2291.pdf>.

[2] Allain, Alex. "C File I/O and Binary File I/O." *C File I/O Tutorial*. Web. 8 Apr. 2012. <http://www.cprogramming.com/tutorial/cfileio.html>.

[3] "E Ink Pearl Imaging Film." *E Ink: Technology: Display Products:*. E-Ink Technologies. Web. 24 Sept. 2011. <http://eink.com/display_products_pearl.html>.

[4] "E-paper Technologies Reference Guide." *Epaper Central*. Mar. 2009. Web. 23 Sept. 2011. <http://www.epapercentral.com/epaper-technologies-guide>.

[5] "Fopen Function." *C Function Reference*. Web. 8 Apr. 2012. <http://www.friedspace.com/fopen.html>.

[6] "FZT549 Datasheet." Web. 14 Dec. 2011. <http://www.diodes.com/datasheets/FZT549.pdf>.

[7] "I.MX50 Evaluation Kit." *I.MX50 Evaluation Kit Product Summary Page*. Freescale Semiconductor. Web. 8 Oct. 2011. <http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=IMX50EVK>.

[8] "Lenmar - Lithium-Ion Battery." *Best Buy*. Web. 13 Dec. 2011. <http://www.bestbuy.com/site/Lenmar - Lithium-Ion Battery for Most Palm Mobile Phones/9344042.p?id=1218088196543>.

[9] "Microsoft Windows Bitmap File Format Summary." *FileFormat.info*. Web. 6 Apr. 2012. <http://www.fileformat.info/format/bmp/egff.htm>.

[10] "The Prodigy." *The Prodigy*. Web. 23 Sept. 2011. <http://www.gettheprodigynow.com/>.

[11] "SUB-20 Multi Interface USB Adapter USB to I2C SPI GPIO RS232 RS485 Ir." *XDimax*. Web. 29 Feb. 2012. <http://www.xdimax.com/sub20/sub20.html>.

[12] "Welcome to IMXCommunity.org." *IMXCommunity.org*. Web. 14 Jan. 2012. <http://imxcommunity.org/>.

Appendix A – Battery Charging State Diagram

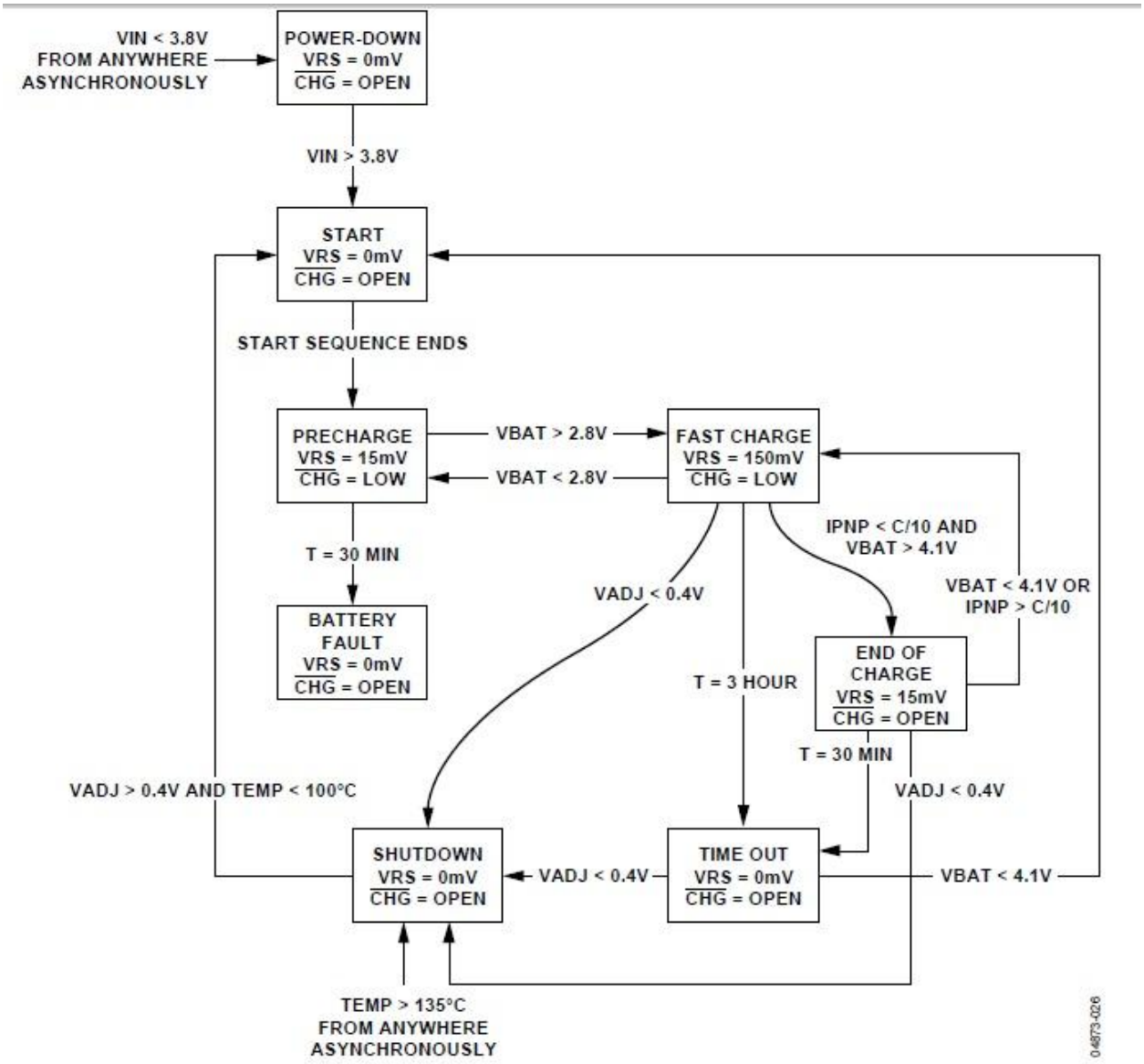


Figure 26. State Diagram for the ADP2291
CTIMER = 0.1 µF

Appendix B – C Code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "libsub.h"
#include "DCF.h"
#include <time.h>
#include <windows.h>


//GLOBAL DEFINITONS
sub_handle sub20handle;
int selection;
int rightpage;
int leftpage;
int menustate;
int maxpages;
int numpieces = 5;


void main() {
   int returnError;
   FILE *blankfile;
   blankfile = fopen("blankpage.bmp", "rb");
   printf("About to initialize. /n");
   returnError = initializeDCF();
   if( returnError !=0  ) {
   printf("sub_open: %s\n", sub_strerror(sub_errno));
   return;
   } else {
   buttonListener();
   }
}

void home() {
   if (menustate != 0) {
       FILE *menu0;
       menustate = 0;
       selection = 0;
       rightpage = 0;
       leftpage = 0;
       menu0 = fopen("menu00.bmp", "rb");
       sendImage(menu0, 1);
   }
}



void enter() {
   char* rchar;
   char* lchar;
   char* schar;
   char* rpiece;
   char* lpiece;
   char* lfileloc;
   char* rfileloc;
```

```
    FILE *rfile;
    FILE *lfile;
    if (menustate != 1) {
        rightpage = 1;
        rchar = (char*)rightpage;
        leftpage = 0;
        lchar = (char*)leftpage;
        schar = (char*)selection;
        rpiece = strcat(schar, rchar);
        lpiece = strcat(schar, lchar);
        rfileloc = strcat(rpiece, ".bmp");
        rfile = fopen(rfileloc, "rb");
        lfileloc = strcat(lpiece, ".bmp");
        lfile = fopen(lfileloc, "rb");
        sendImage(lfile, 0);
        sendImage(rfile, 1);
    }
}

void pageRight_up() {
    char* rchar;
    char* lchar;
    char* schar;
    char* selected_piece_img;
    char* rpiece;
    char* lpiece;
    char* selected_piece_file;
    char* lfileloc;
    char* rfileloc;
    FILE *rfile;
    FILE *lfile;
    FILE *sfile;
    FILE *blankfile;

    if (menustate = 0) {
        if (selection != numpieces) {
            selection++;
            selected_piece_img = strcat("menu", (char*)selection);
            selected_piece_file = strcat(selected_piece_img, ".bmp");
            sfile = fopen(selected_piece_file, "rb");
            sendImage(sfile, 1);
        }
    } if (menustate = 1) {
        rightpage = rightpage+2;
        rchar = (char*)rightpage;
        leftpage = leftpage + 2;
        lchar = (char*)leftpage;
        schar = (char*)selection;
        rpiece = strcat(schar, rchar);
        lpiece = strcat(schar, lchar);
        rfileloc = strcat(rpiece, ".bmp");    //location of rfile
        lfileloc = strcat(lpiece, ".bmp");    //location of lfile

        rfile = fopen(rfileloc, "rb");
        if(rfile == NULL) {
            sendImage(blankfile, 1);
        } else {
            sendImage(rfile, 1);
```

```
            fclose(rfile);
        }

        lfile = fopen(lfileloc, "rb");
        if(lfile == NULL) {
            sendImage(blankfile, 0);
        } else {
            sendImage(lfile, 0);
            fclose(lfile);
        }
    }
}

void pageLeft_down() {
    char* rchar;
    char* lchar;
    char* schar;
    char* selected_piece_img;
    char* rpiece;
    char* lpiece;
    char* selected_piece_file;
    char* lfileloc;
    char* rfileloc;
    FILE *rfile;
    FILE *lfile;
    FILE *sfile;
    FILE *blankfile;
    if (menustate = 0) {
        if (selection != 0) {
            selection--;
            selected_piece_img = strcat("menu", (char*)selection);
            selected_piece_file = strcat(selected_piece_img, ".bmp");
            sfile = fopen(selected_piece_file, "rb");
            sendImage(sfile, 1);
        }
    } if (menustate = 1) {
        if (leftpage != 0) {
            rightpage = rightpage - 2;
            rchar = (char*)rightpage;
            leftpage  = leftpage - 2;
            lchar = (char*)leftpage;
            schar = (char*)selection;
            rpiece = strcat(schar, rchar);
            lpiece = strcat(schar, lchar);
            rfileloc = strcat(rpiece, ".bmp");
            lfileloc = strcat(lpiece, ".bmp");

            rfile = fopen((strcat(rfileloc, ".bmp")), "rb");
            if(rfile == NULL) {
                sendImage(blankfile, 1);
            } else {
                sendImage(rfile, 1);
                fclose(rfile);
            }
            lfile = fopen(lfileloc, "rb");
            if(lfile == NULL) {
                sendImage(blankfile, 0);
            } else {
```

```
                sendImage(lfile, 0);
                fclose(lfile);
            }
        }
    }
}


int sendImage( FILE *imageFile , int displaySelect )
{

    int row;
    char invert;
    char imageBufferArray[200000];
    char* p = &imageBufferArray[0];
    int i;
    int n = 0;
    char* outBuffer;
    int busyPin = 1;
    int headerBytes=0x06A0;

    fseek(imageFile, 0, SEEK_SET);
    fseek(imageFile, 62, SEEK_CUR);
    fread(p, 1, 200000, imageFile);    // Transfer imageFile to
imageBufferArray[] 1 byte per
    for (i=0; i<170000; i++) {
        invert = imageBufferArray[i];
        invert^=0xff;
        imageBufferArray[i]=invert;
    }
    sub_spi_transfer(
sub20handle,(char*)&headerBytes,0,2,SS_CONF(displaySelect,SS_L));
    printf("Sent header bytes /n");


    while ( busyPin == 1 )
    {
        busyPin = readGPIO(5);
    }
    printf("Busy pin went low /n");

    for (row = 0; row<=1279; row++)
    {
        outBuffer = &imageBufferArray[n];
        if (row==1279) {
            sub_spi_transfer(sub20handle,outBuffer,0,128,SS_CONF(displa
ySelect ,SS_LO));
        } else {
            sub_spi_transfer(sub20handle,outBuffer,0,128,SS_CONF(displa
ySelect ,SS_L));
            n = n + 128;
            busyPin = 1;
            while ( busyPin == 1 )
            {
                busyPin = readGPIO(5);
            }
        }
    }
```

```
   fclose(imageFile);
   return 0;        // Return 0 if there were no errors and image was
successfully written
}

int readGPIO(int pinNumber)
{
   /*
   GPIO#        Function        I/O (Input is 0 and Output is 1 in
set)
   ------------------------------------------------------------------
------
   0            Home                          Input
   1            Select                          Input
   2            Pg Right/Up                      Input
   3            Pg Left/Down                  Input
   4            Button source                  Output (always 1)
   5            Display #0 busy pin              Input
   6            Display #1 busy pin              Input
   7            "On" pins for displays          Output (always 1)
   ------------------------------------------------------------------
------
   */

   int pinState;
   int bitshift;
   int homeGPIO;
   int selectGPIO;
   int pgRightUpGPIO;
   int pgLeftDownGPIO;

   int status;
   status = sub_gpio_read(sub20handle, &pinState);            // Read
GPIO states and put into pinState

       if (pinNumber == 0) {                                  // If
pinNumber is 0, read all 4 buttons
           bitshift =  1<<1;
           homeGPIO = pinState & bitshift;
           bitshift =  1<<2;
           selectGPIO = pinState & bitshift;
           bitshift =  1<<3;
           pgRightUpGPIO = pinState & bitshift;
           bitshift =  1<<4;
           pgLeftDownGPIO = pinState & bitshift;

           if (homeGPIO == 1) {                                   //
Return value of button pressed
               return 1;
           } else if (selectGPIO == 1) {
               return 2;
           } else if (pgRightUpGPIO == 1) {
               return 3;
           } else if (pgLeftDownGPIO == 1) {
               return 4;
           }
       } else {
           bitshift =  1<<pinNumber;
```

```
            pinState = pinState & bitshift;
            return pinState;

        }
}

/*
GPIO Configuration values:
Set Inputs/Outputs = 0x00000090
Pin mask = 0x000000FF
*/

int initializeDCF(  )
{
   FILE *menu0;
   int returnError = 1;
   int config;

   FILE *blankfile;
   sub20handle = sub_open(0);
   menu0 = fopen("SheetMusicTestBMP.bmp", "rb");
   blankfile = fopen("blankfile.bmp", "rb");
   if( !sub20handle  )         // return error if handle DNE
   {
       printf("sub_open: %s\n", sub_strerror(sub_errno));
       return -1;
   }
   printf("Checked sub20handle\n");
   returnError = sub_spi_config( sub20handle,
SPI_ENABLE|SPI_CPOL_RISE|SPI_SMPL_SETUP|SPI_MSB_FIRST|SPI_CLK_250KHZ,
0 );
   if( returnError !=0  )      // return error if spi initialize fails
   {
       printf("spi_config: %s\n", sub_strerror(sub_errno));
       return -1;
   }
   printf("Set SPI config\n");
   returnError = sub_gpio_config( sub20handle, 0x00000090, &config,
0x000000FF );
   if( returnError !=0  )      // return error if gpio config fails
   {
       printf("gpio_config: %s\n", sub_strerror(sub_errno));
       return -1;
   }
   printf("Set GPIO config\n");
   returnError = sub_gpio_write( sub20handle, 0x00000090, &config,
0x00000090 );
   if( returnError !=0  )      // return error if gpio write fails
   {
       printf("gpio_write: %s\n", sub_strerror(sub_errno));
       return -1;
   }
   printf("Wrote values to GPIO\n");
   returnError = sendImage( menu0, 0 );    //send beginning menu to
left

   if( returnError!=0  )       // return error if send image fails
   {
```

```
        printf("send_image: %s\n", sub_strerror(sub_errno));
        return -1;
    }

    printf("Sent Image\n");

    Sleep(5000);
    sub_close(sub20handle);
    return 0;                   // return 0 if successful
}


int buttonListener() {
    int caseselect;

    for(;;) {
        caseselect = readGPIO(0);
        if (caseselect = 1) {
            home();
            Sleep(10);
        } else if (caseselect = 2) {
            enter();
            Sleep(10);
        } else if (caseselect = 3) {
            pageRight_up();
            Sleep(10);
        } else if (caseselect = 4) {
            pageLeft_down();
            Sleep(10);
        }
    }
}
```