University of Puerto Rico
Mayagüez Campus
Department of Electrical and Computer Engineering

Final Report

SWORD:
A Spherically Wound Orb Rotary Display

Prepared for:
Manuel A. Jiménez, Ph.D.

Group: Legendtronics

Willie W. González, Group Leader
Edward Betancourt
Rogelio E. Cardona
Ramón J. González

May 6, 2009

Abstract

The Spherically Wound Orb Rotary Display (hereto after “SWORD”), is a device capable of displaying images and text over a three dimensional space. Due to its eye appeal, and bright display, it can be used as marketing gimmick. Through a computer interface, users can interact with SWORD by changing what is being displayed. Users can select from several images available, or can write alphanumeric text. The display is achieved through the concept of persistence of vision, a phenomenon that allows us to perceive fluid motion in animation by refreshing an image at a rate of 10 hertz or higher. SWORD achieves a refresh rate in that range by rotating an arc covered with LEDs through the use of a DC Motor. The DC motor on average produces 655 revolutions per minute, allowing a refresh rate of approximately 11 hertz. Alternating the LEDs every 463.8 microseconds, we can create a sphere of color lights that make up the images we present. The LEDs are biased for the corresponding colors through sets of five serial in-parallel out shift registers. An image-encoding scheme was implemented for built in images, such that 48 values encode the 8 Red-Green-Blue states of colors across all the LEDs located on SWORD's arc at a single instant in time. We outline the theory, design and development of SWORD, as well as the considerations taken during design and implementation that ensured efficiency and correctness.

Table of Contents

1 Introduction

The New Oxford American Dictionary defines a gimmick as "a trick or device intended to attract attention, publicity, or business" [OAD]. Businesses (specifically "retail stores") use marketing gimmicks to attract potential customers to their shops, and come in various flavors. A relevant type of gimmick to our project is the type that virtually does nothing else other than "stand out"; it just looks pretty. Amazingly, even though these items just look pretty, they catch the eyes of people, due to the fixation that we as humans have on appearance [S]. Our project was the development a marketing gimmick code-named "SWORD": a Spherically Wound Orb Rotary Display. SWORD is a three-dimensional rotary display capable of displaying images and text through the concept of persistence-of-vision. It is a marketing gimmick due to its impressive hardware, and colorful display.

Three-dimensional rotary displays aren't a new concept. Several implementations of these rotary displays can be seen on the Internet [YT1], [YT2]. However, little to no documentation is available on the steps taken during the design and implementation. No clear starting point of what is necessary to create one was available. One particular implementation did have documentation in the form of personal blogging [JNS], and was influential during the development of product specifications. Although the designer's version of a rotary display was deemed too complex (with approximately 320 Red-Green-Blue light-emitting diodes, 960 surface-mount resistors, and 16 80-pin PIC Microcontrollers) several key aspects were considered in our design. For example, the designer points out that his design was modular: he designed one-fourth of the system hardware, and the rest replicated that design. Another contribution of the designer's work was that he posted pictures of his prototype through various steps in the development prototype. This allowed us to visually grasp the concept of what we were dealing with, and provided a reference frame of what our project should look like.

Our technical scope includes as its main component "the arc", a transversal cut of a cylinder, which is twelve inches in diameter. The cut is two inches thick. This arc has 78 light-emitting diodes (LED's) vertically aligned on the outer surface. The arc rotates through the use of a DC motor, which is external to the primary circuit. With a sufficiently high speed, the LED's create iconic memory and create the illusion of a complete sphere [PLS]. Using proper timings, and color changes, (which are controlled with an MSP430 based microcontroller) spherical images can be created. Our prototype was developed under several constraints. First, project completion was expected by April 21st, 2009. Second, our team had no working capital with which we could immediately finance the project. Third, SWORD employs the use of powerful machinery and can pose a threat to physical safety. Fourth, the Microprocessor Interfacing Laboratory (which has tools to streamline prototype development) was unavailable during the initial and middle stages of our prototype development lifecycle. The initial specification promised the display of two pre-coded and hardwired images,

along with the ability to display written text via a Graphical User Interface. This design was presented to and evaluated by Dr. Manuel A. Jiménez, who pointed out two considerations that should be taken for the project, before his approval. The first was the creation of images through the use of interlaced LED's. The second was creating images in a closed loop through hardware by adding an optocoupler circuit, which provided a constant starting point for the image display. These considerations were incorporated into the final design and were approved by Dr. Jiménez shortly thereafter. In the following sections, we outline our development process from design to implementation, as well as point out any considerations taken along the way to ensure the project's success.

2 Theoretical Background

2.1 Persistence of Vision Effect

Persistence of vision is a phenomenon in which a human eye retains an image for a very short time after seeing it [T]. Another way to think of it is that the information "extracted" from visual stimuli persists through what is known as "iconic memory" for some time after its first appearance [C]. This is due to the eye being a human organ, which acts with chemical reactions [HAR]. The chemical reactions are slower than the speed of light. Light, which moves faster than the eye can see it, will generate the illusion of a continuous stream if it is in the same place every 0.0416 seconds (which would be equivalent to refreshing the rate at which we see light at 24 hertz).

The most important aspect of the persistence-of-vision concept relevant to SWORD answers the question: "how many frames per second are necessary for fluid motion?" Put differently: "how fast must SWORD enable its lights to travel in order to 'see' a continuous stream of light?" There is no general consensus on the exact speed. Several argue 18 frames-per-second (a refresh rate of 18 hertz) is an approximate number needed to achieve persistence-of-vision [100]. Others argue that 16 frames-per-second (a refresh rate of 16 hertz) is enough [PF]. Guided by the refresh rate chosen for American television and cinema, our original design aimed to achieve a refresh rate of 24 hertz for the entire display. In other words, a LED rotating on SWORD must leave a starting position x, and travel 360 degrees in 0.0416 seconds (1/24). If this condition was satisfied, the persistence-of-vision effect could be achieved. Figure 1 demonstrates the visual effect created through persistence of vision. Notice how the LEDs located on SWORD seem to "draw" continuous streams of light as it rotates.

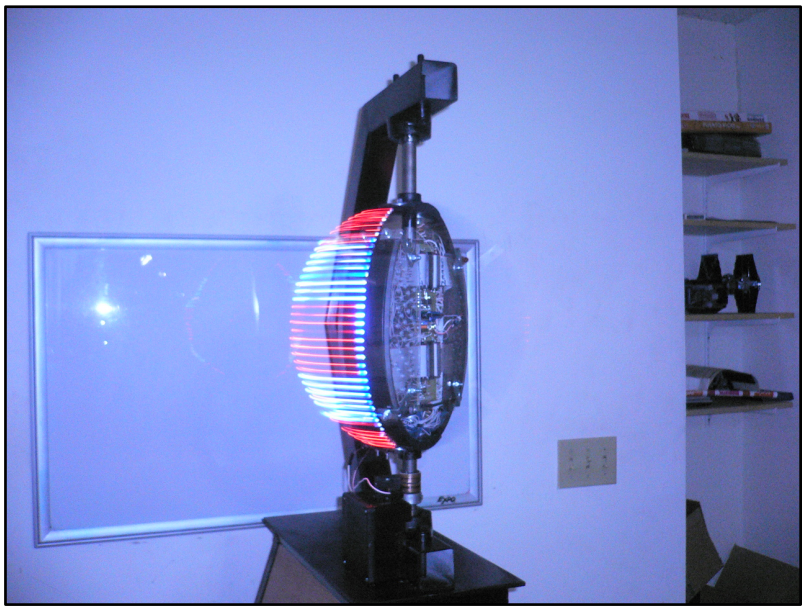


Figure 1: Persistence of vision demonstrated on SWORD.

2.2 Image and Timing Considerations

There are several considerations around SWORD that complicate system calculations for image timing and representation. They are presented in logical order, and are handled as they are presented.

2.2.1 LED Spherical Distance and Refresh Rate

Not all LEDs travel the same distance. This is because a LED does not necessarily represent one degree, and therefore a LED might not need 360 "steps" around a circle to complete one revolution. As an example, consider a case where a LED occupies 90 degrees of a circle. In four steps, this LED completes one revolution. However, if a LED occupies 45 degrees, it takes double the amount of steps to complete one revolution. In reality, the problem of "not all LEDs travel the same distance", stems from the problem that "not all LED's occupy the same number of degrees".

Degree size depends on the radius of the circle it's being measured against, as stated by the formula:

(angle) * (radius) = (arc length) $\Leftrightarrow$ $\Theta * r = s$
Solving for theta: $\Theta = s / r$

In our case, the arc length was constant: the length of the LEDs (approximately 0.93"). However, the radius in the formula varies. This is because the radius in the equation refers to the distance from the LED to the axis of rotation, and not to the distance from the LED to the center of the arc, which is constant for all LEDs (see Figure 2).

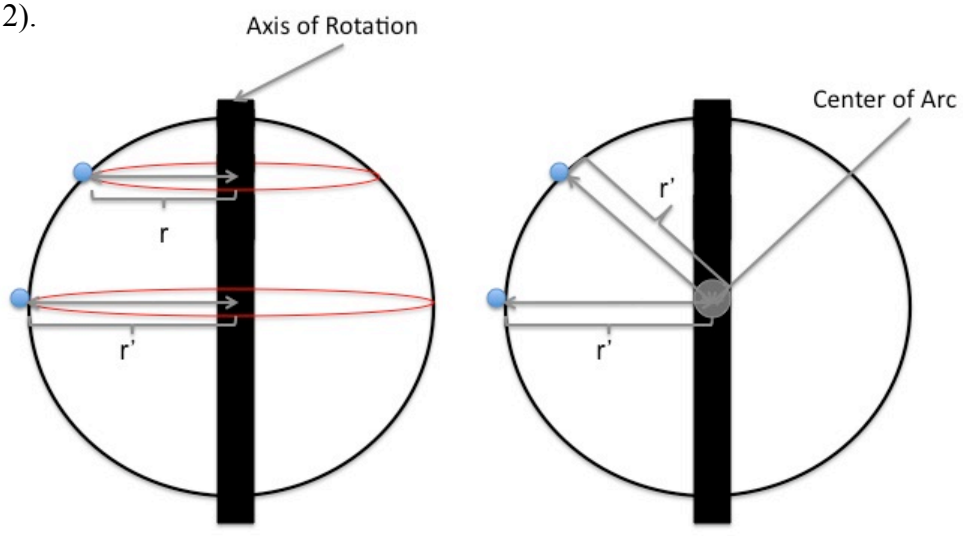


Figure 2: Image of SWORD arc that compares the various distances that LEDs travel vs. the constant distance from the center of the arc

LED degree occupancy varies vertically on SWORD's arc. LEDs located at the top of the arc are geometrically closer to the axis of rotation than LED's located at the arc's equator. Therefore, the closer the LED is to the equator (which has maximum radius), the smaller the degree occupancy of the LED. Going back to our original problem of LED's not traveling the same distance, we conclude that the farther away the LED is from the axis of rotation, the longer it travels. This means that LEDs located on SWORD's diameter (which represents the farthest distance from the axis of rotation), travel the longest distance. Therefore, if we can assure that the system can display properly at the equator of SWORD's arc, the rest of the system is safe for image display. The width occupancy of the LED at this position is:

$\Theta = s / r$, where $s = 0.93"$ and $r = 6.0"$ $\therefore \Theta = 1.84$ degrees

Since the LED occupies 1.84 degrees, to cover the entire circumference of the circle, we need:

1.84 degrees x (number of LEDs) = 360 degrees
(number of LEDs) = (360 / 1.84) = 195.33 LEDs

Since fractional amounts for LEDs are meaningless in our context, we decided to overshoot the circumference by rounding upwards to the nearest unit. This makes (number of LEDs) = 196.

Going back, this would mean that in reality, we're assuming each LED occupies 1.83 degrees:

360 degrees / 196 = 1.83 degrees
(This causes an overlap amongst "continuous" LED's by approximately 0.34%)

The number of LEDs needed to cover the circumference indicates the number of "steps" the LED at the equator must take to be able to complete one revolution. Figure 3 shows the trajectory of the LED at the diameter. At this point we feel it is necessary to point out that with this quantity, we have defined the maximum image resolution we can achieve. With 78 vertically aligned LEDs, and 196 steps, our "drawing field" amounts to 196 x 78. We discuss the image representation considerations in Section 2.2.2.

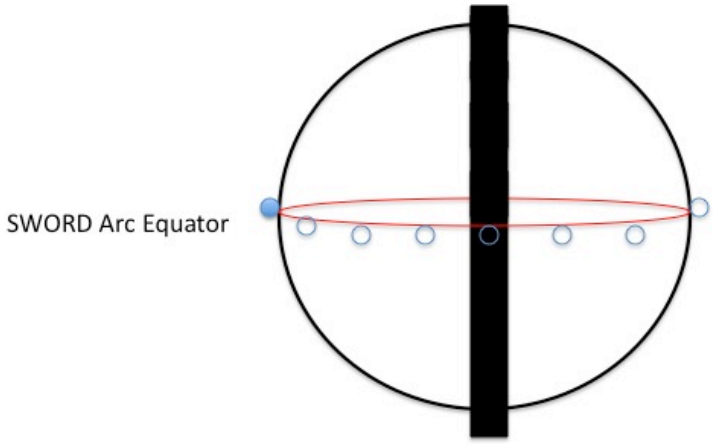


Figure 3: One LED at the equator, and various “shadow” LEDs along its trajectory

We remind the reader that we aimed to achieve a display frequency of 24 hertz at the diameter. This implied that the LED at the diameter must complete one revolution in 0.0416 seconds. However, after initial testing, we noticed that our display was still visible at less frequency. The DC motor that we utilized was rotating SWORD’s arc at approximately 655 revolutions per minute (rpm). We discuss the DC motor in Section 2.2.4. 655 rpm is equivalent to approximately 11 hertz, and therefore, the LED at the diameter must complete one revolution in 0.0909 seconds. An important quantity that stems from this number is the time taken from step to step. In other words, how long does the LED take in going from step n to step n+1? To determine this quantity we divide the time taken for one revolution by the number of steps in the revolution:

0.0909 seconds / 196 steps = 0.000463821 seconds = 463.8 microseconds

This means that the LED located at the diameter must be able to change color every 463.8 microseconds in order to uniquely display 196 steps across its entire revolution. LED's located above and below the diameter must actually change color faster in order to achieve the same effect, but we concluded that the complexity behind controlling every individual LED did not merit handling this situation. What results is that they travel "196" steps, only that their steps are much shorter. This means that in the time that the LED at the diameter completes 1 step, the LED above might complete 2 or 3, and so on.

2.2.2 Image Representation in Three Dimensions

Traditional images are seen planar, which correspond to the two-dimensional Cartesian coordinate system. For SWORD, two-dimensional images must be transformed into three-dimensional ones. Consider figures 4 and 5, which represent a two-dimensional image of Earth, and a three-dimensional image of Earth, respectively.

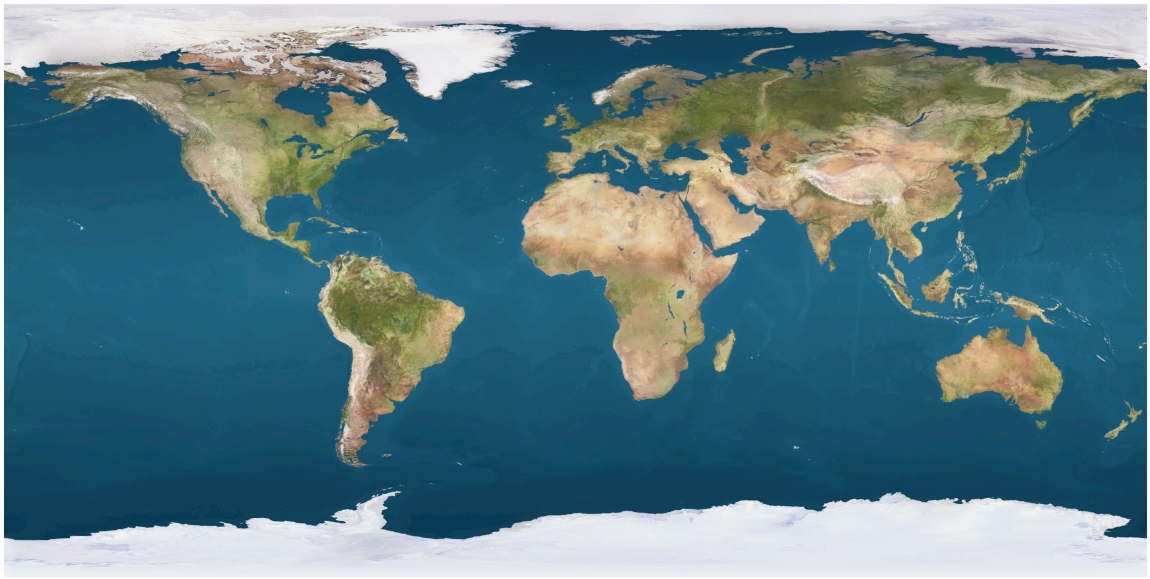


Figure 4: Cartesian coordinate system map

Figure 5: Spherical coordinate system map

To be able to view an image like that seen in figure B on SWORD, we must convert figure 4 into figure 5. This corresponds to a mathematical transformation known as an inverse map projection [MW]. In its purest form, an inverse map projection is achieved by converting planar coordinates (x,y) to spherical coordinates (r,Θ,φ). In our particular case, the r dimension of all spherical coordinates is constant: it is the radius of SWORD's arc. If we were to view the inverse map projection of the Earth on an (x,y) coordinate plane, we would view the projection as is seen in figure 6.

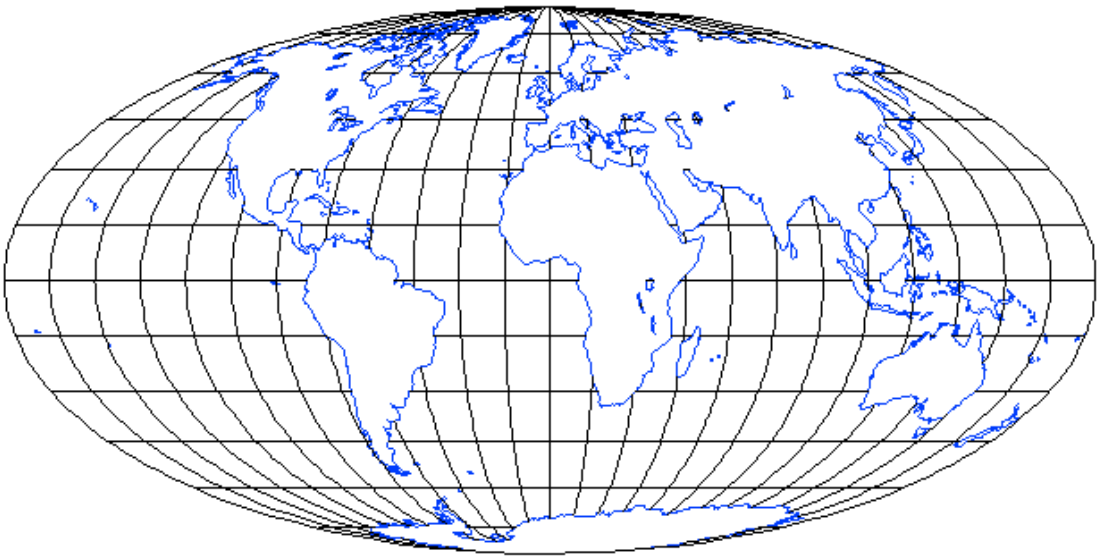


Figure 6: Planar spherical transformation

We previously mentioned that the image resolution granted by SWORD was 78 x 196. In truth, it's a bit less. That is because when a spherical coordinate system is represented on a Cartesian coordinate system, there is loss of resolution [MAP]. Therefore, we cannot represent with 100% accuracy a Cartesian coordinate system on a spherical one. Which implies that in reality our drawing plane is not 78 x 196, but rather a complex drawing plane made up of irregularly sized columns (see Figure 7).

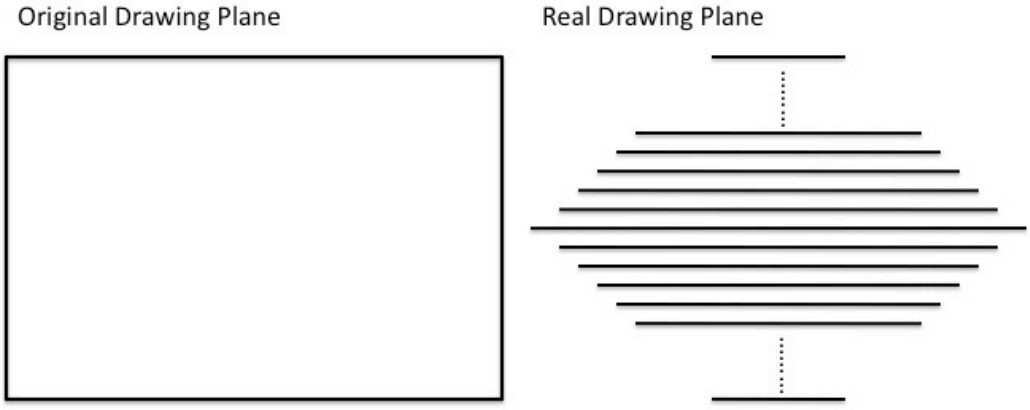


Figure 7: Originally mentioned drawing plane vs. actual one

This agrees with what was mentioned earlier regarding the distance that LEDs travel. The LED located at the arc's equator travels longer than LEDs above or below it. Therefore the "drawing path" for LEDs at the arcs poles is much shorter than the drawing path for the LED at the equator.

2.2.3 Image Timing Considerations

Our main project stakeholder pointed out to considerations that should be observed when dealing with image timings.

The first was that since the image was going to be displayed on a sphere, the system should ensure that the image would be displayed in a closed loop. Since the display was going to repeat itself every revolution, the system needs a way to guarantee that the image timings will be accurate. If the image timing fluctuates and causes the image to rotate, the system must provide a mechanism to catch the runaway image so that it does not continue to rotate. Dr. Jiménez therefore suggested the inclusion of an optocoupler circuit so that with every revolution, a signal would be sent to the system informing it that it must reset the image display because a revolution has just completed.

The second consideration comes as a consequence of a recommendation from Dr. Jiménez himself. Since the LEDs are wide components, placing them side by side on SWORD's arc was deemed uncomfortable to work with, and therefore Dr. Jiménez suggested displaying images interlaced. That is, LEDs are placed on both sides of the arc in such a way that the spaces between the LEDs on one side are "filled" by the LEDs on the other side during arc revolution. Figure 8 provides a representation of this phenomenon, and serves as an example of image interlacing.

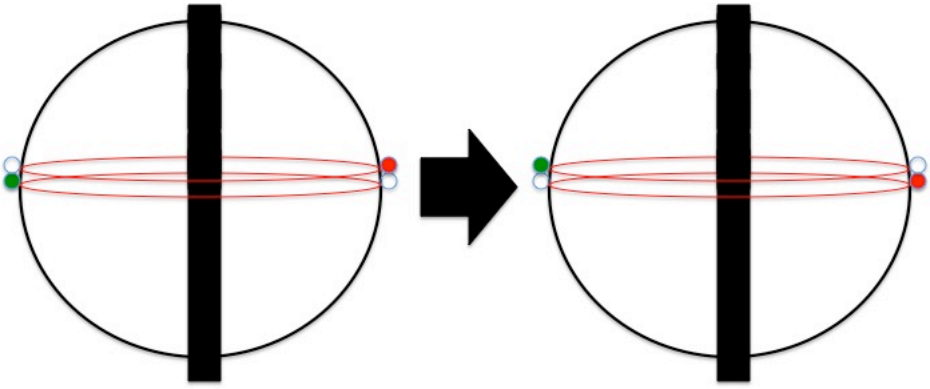


Figure 8: Image interlacing with two complementary LEDs

Using figure 8 as an example, suppose that during one half of the image to be presented on SWORD all LEDs are Green, and during the other half all LEDs are Red. On the left-hand side of figure 8, we can observe that in the Western hemisphere the LED is Green, and in the Eastern hemisphere the LED above is Red. However, when the LEDs complete 98 (which is 196 / 2) steps, to be able to present the complete Red and Green halves of the image, they must change color. The right-hand side of figure 8 demonstrates this. Extending this concept for all LEDs, interlaced LEDs must present the corresponding image colors 98 steps after their complementary LEDs present them. This concept applies to both images and text. The consideration is that whatever is used to represent images within code must make sure to represent the image interlaced, to be able to display both sides of the arc at the same time.

2.2.4 Mechanical Mounting

At 24 hertz, the DC motor, which moves SWORD's arc, would have to operate at 24 revolutions per second (rps). Motors are typically listed by revolutions per minute (rpm). Therefore, the DC motor would have to operate at 24 x 60 = 1440 revolutions per minute. As mentioned, after testing, we realized that due to SWORD's weight we could not achieve a speed greater than 655 rpm (as measured by an external tachometer). However, we were still able to experiment the persistence-of-vision effect. An 11-hertz rotary display is enough. All this implies that system calculations for the time to complete one revolution, as well as the time taken from step to step are wrong. At 11 hertz, the time taken for one revolution is 0.0909 seconds, and the time from step to step is 0.000463821 seconds (463.8 microseconds).

Some final considerations that influenced the overall design of SWORD were observations pointed out in the Introduction. Regarding our working capital, extreme care was taken in selecting the system architecture so as not to incur in unnecessary costs. Competing prices for discrete components as well as integrated circuits were evaluated against each other to determine the cheapest possible solution. Regarding the threat to safety that SWORD produces, we ensured (with the help of external resources) that the mechanical frame that would house SWORD was adequate in order to reduce vibration, and strong enough to keep the rapid moving parts in place.

3 Mechanical Design and Development

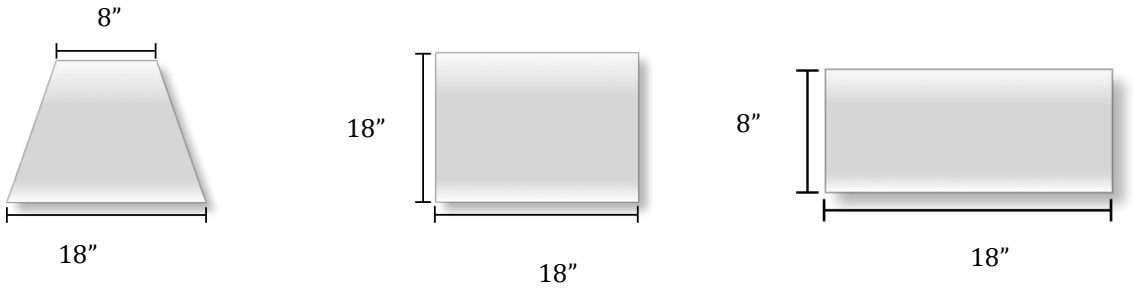
3.1 Overview

One key factor for the proper operation of SWORD is its mechanical mounting. The design was developed around one main concern: the system would rotate at a high speed of more than a 1,000 RPM's. Given this fact, is that all of the materials and parts were chosen accordingly.

3.2 Base

The base of the prototype was designed in a trapezoidal shape. It was made out of galvanized steel tube of 1" x 1" x 14GA. The idea behind this mounting was to create a metal skeleton with a trapezoidal form that will allow the setup of the different parts that are located inside this area. Amongst these parts, it is included the DC Motor, the motor controller and the power supply for the system.

The measurements of the base are detailed in the following diagram:



Note: Diagrams are not to scale.

On top of this skeleton, a metal sheet of 9" x 18" x ¼" was welded. It was specified this way in order to strengthen the whole base, since it will have to withstand the vibration that will produce the rotation of the ring.

3.3 Arm

The arm is the main support of the rotating ring and the electronic components. The material that was used for the arm is galvanized squared steel tube of 2" x 2" x 14GA. The original design specified a circular arm, instead of the C-shaped that was developed. After analyzing the possibility of giving the curve to the squared tube, it was concluded

that it was not achievable. Since the arm has a 26" diameter, the 180 degrees curve would cause either one or both of two things:

- During the process of curving the tube, it was possible that the tube will eventually break due to the excess of stress that is applied. Certainly, this depends on the gage of the tube, but according to the technical advice that was received it was not recommended.
- The deformation of the square tube will cause the arm to look bad because of the deflection and will make it weaker.

3.4 Rotational Mounting

The rotational mounting of SWORD includes the central shaft and the flange bearing. Having in mind that high speeds of rotation are expected, the specifications for these two components were chosen based on the highest speed that the motor could achieve. The DC motor had the capability of reaching a top speed of 3500 RPM.

The central shaft is made of endured steel and it measures 30" of length and ¾" of diameter. The flange bearings that were used can withstand a maximum of 5000 rpm's and are made up of endured steel. The advantage that provides the bearings that were used is that they have some tolerance that makes it easier for the shaft installation.

3.5 Rotating Ring

The rotating ring holds the LEDs of the system. For this part, several aspects were taken into consideration. They are mentioned as follows:

- It could not be made out of a material that was too heavy because it would produce a great amount of rotational inertia.
- It had to withstand the maximum rotational speed of the motor.
- It had to be thick enough to drill it and setup the LED's.

After exploring several options, a material with the desired specifications was found. The rotating ring is made out of high-pressure 12" diameter PVC tube. The material of these tubes is strong enough to withstand the high speed that the system could eventually reach. Also, it is 3/8" thick, so it provides enough space for the LED's accommodation.

3.6 Slip Ring

SWORD utilizes a total of 234 LEDs and a whole set of other electrical components that are detailed throughout this report. Consecutively, if it is wanted to have the system operating for several hours, on board battery power is not an option. Also, there is limited space on board to accommodate a set of batteries that would

accomplish this task. Therefore, a way is needed to pass the current through the rotating shaft.

The prototype uses a set of rotating electrical contacts. This is commonly named as a *slip ring*. The rotating contact is mounted in the central shaft. It receives the power from a set of two brushes that are connected to the power supply. It is important to mention that the current rating for this device was taken into consideration. The slip ring that was installed is commonly used for small wind turbines that usually manage 50 kWatt of electrical power. This is way below the power consumption of SWORD's electronic components.

3.7 Prototype Embellishments

Now at this point, all of the electrical and mechanical considerations were explained, but there is still one more to go: how to make people dazzle with just looking at it without the need of powering the system. First of all, it was needed to get rid of the metallic color of the structure. Therefore, a strong color was desired that would catch one´s attention without making the person look away. After careful considerations, it was decided that it should be a flat black color. Consequently, to protect the color it was painted with an acrylic paint.

Finally, it was desired that the DC motor, the power supply and the cables underneath SWORD were not visible. The next task was to cover the bottom and give it a nice eye-candy appearance. It is important to remember, that the material that was meant to be used to cover the skeleton had to strengthen the system as well, so it will provide more stability. There were two choices to select from: 3/4" cabinet treated wood or use the same steel sheet that was used for the top of the base. Having in mind that the steel sheet would make the prototype considerably heavier, it was resolved that treated wood was the best option.

4 Hardware Design and Development

4.1 Block Diagram and Overall Functionality

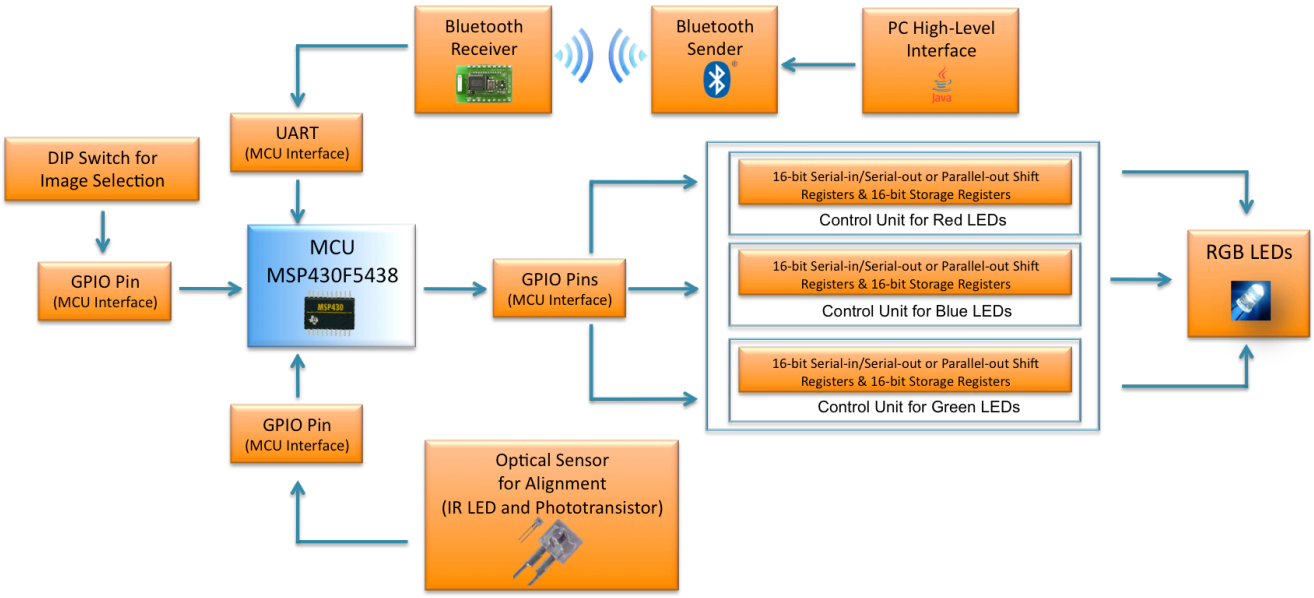


Figure 9: SWORD block diagram

Figure 9 illustrates SWORD's block diagram. Upon system Power On or Reset, SWORD verifies the status of its onboard DIP switch. The state of the switch indicates the image that will be presented upon system Power On or Reset. When the image has been verified, SWORD begins using the LED control units across the 196 steps of the display to control the colors that appear in each step. Every 196 steps (every time SWORD's arc completes one revolution), the optical sensor circuit interrupts the MCU execution to reset the image to zero. This is done to ensure the image is displayed along the same steps every revolution (thus "closing the image loop" manually). To be able to interact with SWORD, users must utilize a Bluetooth® enabled PC. This is because, when a user selects a different image for display or decides to write text to SWORD, the high level interface (which was implanted in Java™) uses the native Bluetooth® antenna to send information to the Bluetooth® receiver connected to the MCU. The MCU interprets the users command, and proceeds to display the corresponding image or text.

The following sections describe in detail the operational block diagram of our system. Its explanation is divided into five sections: 1) Optical Sensor for Alignment, 2) LED Control Units, 3) RGB LED's, 4) User Interface (Java & On-Board Dip Switch) and 5) MCU.

4.1.1 Sections Explanation

In the following sections, we explain the block diagram by individually explaining each section outlined in the overall functionality explanation.

Optical Sensor for Alignment

The optical sensor is implemented using an optocoupler configuration with an IR LED and phototransistor. It is connected to the MCU via a GPIO Pin. Its purpose is to send a signal that marks the initial point of displaying. This ensures that images are displayed in a closed loop. Our configuration fixes the IR LED on SWORD's base stand, and places the phototransistor on SWORD itself. When the phototransistor rotates 360 degrees, it hovers directly above the IR LED, activating it and generating an interrupt signal on the GPIO pin it is connected to.

LED Control Units

The LED Control Units are composed of a set of five 16-bit Serial-in Parallel-out Shift Registers & 16-bit Storage Registers. Each color has its own Control Unit, totaling fifteen shift registers. The following signals from the chip are connected to the MCU via GPIO Pins: Output Enable (**OE**), Latch Enable (LE), Clock (CLK) and Serial-Data-In (SDI). All LED Control Units share the same control signals (**OE**, LE, and CLK). Every step in SWORD's revolution, data flows from the MCU to the LED Control Units to decide which LEDs will be turned on and which will be turned off for that particular step.

RGB LEDs

There are a total of 78 RGB LED's. Each of them can display three colors, so there are a total of 234 colors that need to be managed. The signals to control the LEDs come from shift registers as previously described.

User Interface (Java & Onboard DIP Switch)

There are two ways to interact with the system. The simplest option is with an on-board switch button that allows you to decide between two pre-recorded images that are in the memory of the system. The second option to interact with the system is with a high level Java interface that controls a Bluetooth® module that sends text to the MCU to be displayed on SWORD. The Bluetooth® module is connected via GPIO pins configured as UART pins.

Microcontroller Unit

The MCU controls the whole system operation. The MCU selected for our application was an MSP430F5438. It was selected due to its amount of GPIO Pins, and its operating speed; crucial components for the success of our application. It directly controls the LED Control Units by coordinating the data sent to them as well as the control signals that shift data in, as well as shift data out. It reacts to input from the optical sensor circuit (hereto after "optocoupler circuit"), as well as input from the Bluetooth® interface. It is configured for initial operation with the DIP switch.

4.2 Power Analysis

Our Power Analysis of SWORD's hardware architecture goes individually part-by-part analyzing individual power consumptions, and concludes with the power consumption of the entire system.

RGB Light Emitting Diodes

Table 1: RGB Power Consumptions per Color

LED Color	$V_{forward}$ (V)	Luminous Intensity at 20 mA (mcd)	$I_{forward}$ (mA)	Power (mW)
Red	2	800	15	30
Blue	3.5	4000	15	52.5
Green	3.5	1500	15	52.5

Table 1 lists the different biasing conditions that were selected for each color of the RGB LEDs. High currents were selected to deal with the LEDs duty cycle. This current was adjusted to achieve an acceptable luminous intensity while SWORD's arc was rotating. Our initial design considered using 7 mA for the Red color, and 2 mA for the Green and Blue colors. However, in the worst case, a LED turns on during one step out of the 196 steps in SWORD's revolution. LEDs change colors every 463.8 microseconds. The duty cycle of the LEDs given the original design currents was not enough for us to perceive a continuous stream of light; the light was too faint. Unfortunately, a duty cycle characteristic table (or graph) was not provided by the datasheets of the LEDs. For that reason, the values of the currents were experimentally determined. They were adjusted until we could see a bright stream of lights.

The total power consumption of the LEDs depends on the operating mode. SWORD has two primary operating modes: Image Display and Text Display. In the case of Image Display, all LED colors for all LEDs could be on at the same time. That would imply a LED power consumption of:

$$([P_B + P_G + P_R] \times 78)\ mW = ([52.5 + 52.5 + 30] \times 78)\ mW = 10{,}530\ mW$$

In the case of Text Display, a maximum of 10 LED's are going to be on at a given instant in time. For Text Display Mode, we decided to use the blue color. This would imply a LED power consumption of:

$$([P_B] \times 10)\ mW = ([52.5] \times 10)\ mW = 525\ mW$$

STP16CPS05 (Serial In, Parallel Out Shift Register)

Operating voltage: 0 V-7 V (3.3 V was selected for better compatibility with MCU)
Operating current (when driving a logic LOW): 4 mA
Operating current (when driving a logic HIGH): 4.5 mA
Power consumption (when driving a logic LOW): 13.2 mW
Power consumption (when driving a logic HIGH): 14.85 mW

Bluetooth Module

Operating voltage: 3.3 V – 6 V (5 V was selected due to malfunction at 3.3 V)
Operating current: 25 mA (avg.)
Power consumption: 125 mW

MSP430F5438

Operating voltage: 2.2 V- 3.6 V (3.3 V was selected)
Operating current: 2.16 mA
Power consumption: 7.128 mW

QSE113 Phototransistor for Optical Sensor

Operating voltage: V_{SAT} = 0.4 V
Operating current: 1.5 mA (max)
Power consumption: 0.6 mW

QEE113 – Plastic IR LED

Operating voltage: 1.5 V
Operating current: 38 mA
Power consumption: 56.25 mW

Sub-Fractional DC Motor (Model #M1120059)

Operating voltage: 90V
Operating current: 1.5A
Power consumption: 135W

4.2.1 Total System Power Consumption

Total system power consumption is the sum of all individual power consumptions, including the LEDs. The different operating conditions cause the power

consumption to vary (we disregard power consumption when the device is off, since it can be assumed to be zero).

$P_{total} = P_{LEDs} + P_{Bluetooth} + P_{motor} + P_{STP16CPS05} + P_{MSP430F5438} + P_{QEE113\ \&\ QES113}$

$P_{total} = P_{LEDs} + 125\ mW + 135\ W + 14.85\ mW + 7.128\ mW + (0.6\ mW + 56.25\ mW)$

In the case of LEDs displaying text, $P_{LEDs} = 525\ mW$, $\therefore P_{total} = 135.734W$

In the case of LEDs displaying images, $P_{LEDs} = 10{,}530\ mW$, $\therefore P_{total} = 145.739\ W$

If we operate this product on a daily basis of 8 hours on regular week shifts of 40 hours during a whole month, we total 160 hours of operation.

The amount of power during that month that will be charged is (on average):

((146.659 W + 136.009 W) / 2) *160 = 22.61344 Kw/h.

The current rate for Puerto Rico is of .25$/(Kw/h). The use of SWORD in a store will cost approximately: (22.61344Kw)*(0.25$/(Kw/h))=$5.67.

Due the low cost in electricity usage of SWORD and the possible marketing attraction, many stores would probably consider it a fair trade.

4.3 Timing Analysis

There are two major aspects that need consideration regarding timings in our system. The first one concerns to the timings of the shift registers that were used and the second one concerns the transmission of data via de UART Serial Interface from the Bluetooth® Module. The next two sections explain the details that were taken into consideration for both components, as well as timing diagrams that show their interaction with the MCU.

4.3.1 Shift Register Timing Diagram

Figure 10 illustrates the timings that were followed to operate the shift registers. The shift register that was used is the STP16CPS05, which works based on an asserted HIGH logic. Based on this logic the internal shift register writes the information in the data serial in when it changes from logical zero to logical one.

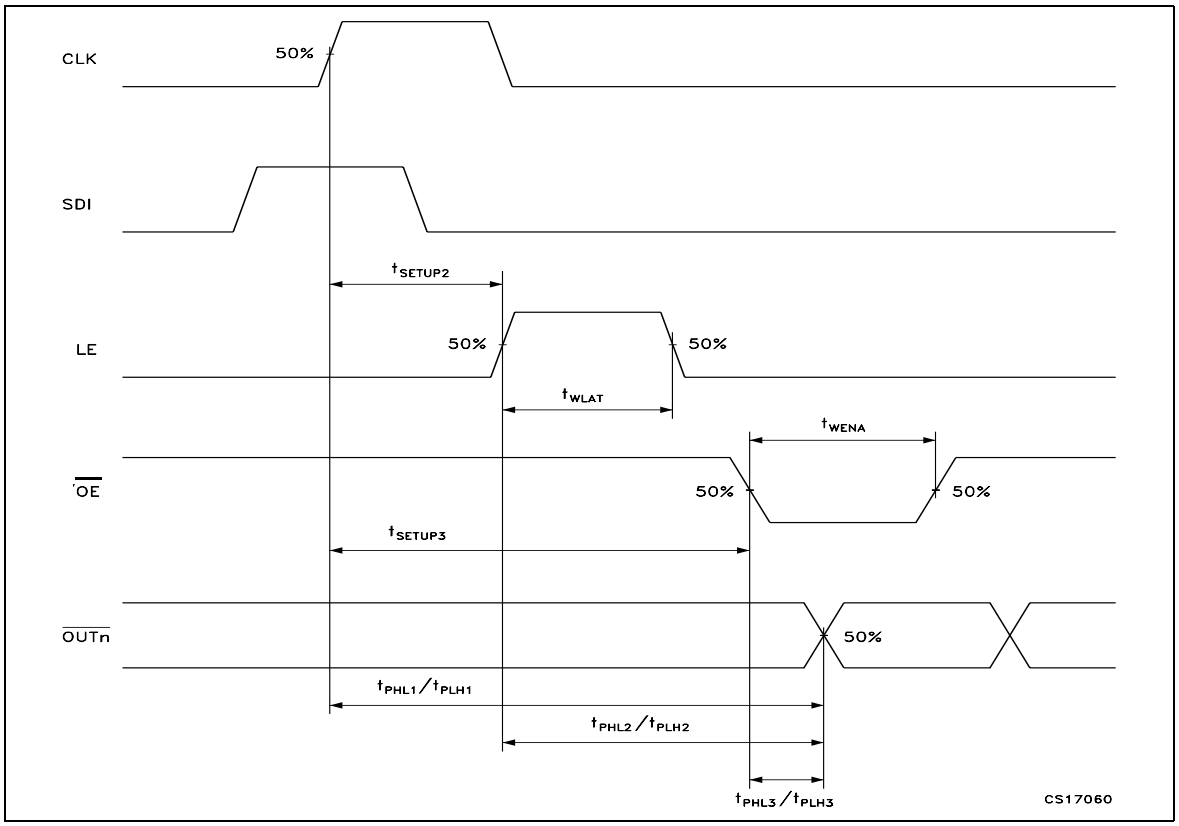


Figure 10: Timing diagram for the STP16CPS05 [SRD]

Table 2: Timings for the STP16CPS05

Symbol	Parameter	Test conditions			Min	Typ	Max	Unit
t_{PLH1}	Propagation delay time, CLK-$\overline{OUTn}$, LE = H, $\overline{OE}$ = L	V_{DD} = 3.3 V V_{IL} = GND I_O = 20 mA R_{EXT} = 1 KΩ	$V_{IH} = V_{DD}$ C_L = 10 pF V_L = 3.0 V R_L = 60 Ω	V_{DD} = 3.3 V		50	70	ns
				V_{DD} = 5 V		28	40	
t_{PLH2}	Propagation delay time, LE-$\overline{OUTn}$, $\overline{OE}$ = L			V_{DD} = 3.3 V		48	70	ns
				V_{DD} = 5 V		25	40	
t_{PLH3}	Propagation delay time, $\overline{OE}$-$\overline{OUTn}$, LE = H			V_{DD} = 3.3 V		55	75	ns
				V_{DD} = 5 V		35	45	
t_{PLH}	Propagation delay time, CLK-SDO			V_{DD} = 3.3 V		11	16	ns
				V_{DD} = 5 V		8	12	
t_{PHL1}	Propagation delay time, CLK-$\overline{OUTn}$, LE = H, $\overline{OE}$ = L			V_{DD} = 3.3 V		22	30	ns
				V_{DD} = 5 V		17	25	
t_{PHL2}	Propagation delay time, $\overline{LE}$-$\overline{OUTn}$, $\overline{OE}$ = L			V_{DD} = 3.3 V		7	10	ns
				V_{DD} = 5 V		4	6	
t_{PHL3}	Propagation delay time, $\overline{OE}$-$\overline{OUTn}$, LE = H			V_{DD} = 3.3 V		18	30	ns
				V_{DD} = 5 V		15	25	
t_{PHL}	Propagation delay time, CLK-SDO			V_{DD} = 3.3 V		12	18	ns
				V_{DD} = 5 V		8	12	
t_{ON}	Output rise time 10~90 % of voltage waveform			V_{DD} = 3.3 V		40	80	ns
				V_{DD} = 5 V		22	35	
t_{OFF}	Output fall time 90~10 % of voltage waveform			V_{DD} = 3.3 V		11	15	ns
				V_{DD} = 5 V		18	25	
t_r	CLK rise time (1)						5000	ns
t_f	CLK fall time (1)						5000	ns

For our system, we need to take into consideration the time that it takes for the shift register to stabilize the clock signal, so we may know how much time the MCU needs to leave this signal on the line to write the data in the SDI line. As we will observe later on, at the core level of our code, the signals for the shift register are put on the line using *MOV* instructions to the specified port. If we assume a best-case scenario for the MCU, it would take only one cycle to put the data on the line. Therefore, since the MCU is being operated at 12Mhz, the time one cycle consumes is:

$$\tau_{Cycle} = (1/12000000) = 83.33 \text{ ns}$$

If we look for the values in the table, we can see that the maximum it takes for the signal to propagate from the clock switching to the OUTn is $\tau_{PHL/PLH} = 70$ ns at $V_{dd} = 3.3$ V. Consequently, the MCU is slower than the SR, which is positive for the system, since it will give enough time for the data to propagate adequately through data signals.

4.4 Driver Analysis

In embedded design, one thing that is very critical for the performance of the system is proper driving throughout the system operation. To accomplish this we have to make sure that the necessary voltage is applied to each device and that those power sources are able to handle the maximum current needed for operation.

First, aside from the RGB LEDs, everything consumed little current; in the range of milliamperes. For example the MCU (MSP430F5438**),** the phototransistor (QSE113) circuit and the shift register **(**STP16CPS05) work with a voltage supply of 3.3 V. In the worst case scenario a current of 72.128 mA is needed for these nodes. For this, the voltage regulator LM317 is used. This regulator is able to maintain a constant voltage and provide up to 1A without problem. The only limitation is that the input voltage to this device has to be higher than the desire voltage to be regulated. With this configuration, there is not going to be a driving problem as long as we provide enough voltage. The Bluetooth operates at 5 V and for that reason another LM317 is needed to regulate this voltage. There are not going to be problems with the currents in this circuit, because it's the only device connected to this 5 V node and its current rating is an average 25mA.

To drive the current of the RGB LEDs, the shift registers **(**STP16CPS05) are used. These devices are sinking current and with an external resistance, are able to drive predetermined currents. These devices can drive up to 1.6 A per package but in our case we are only going to drive a maximum of 304 mA and a minimum of 240 mA. To be able to drive the maximum current we will need to be able to supply 3.822A to all of the LEDs. That current is too much for the Voltage regulator to handle so we decided to connect them directly to the power supply, which can handle that amount of current.

4.5 Hardware Schematic

What follows is an explanation of our hardware blueprint. We briefly comment on the overall hardware schematic, and then enter into detail about individual components.

Figure A.3 (Appendix A) illustrates the final result of the hardware schematic. The top part is the RGB LED configuration. The bottom right demonstrates the LED driver circuit. The bottom left demonstrates the voltage regulators of 3.3 V and 5.0 V. Also seen in the bottom left are the optocoupler circuit (sender and receiver) and the Bluetooth. In the middle we can appreciate the MCU with two slide switches. We will detail each modular component of the schematic in the sections that follow.

4.5.1 LED Circuit

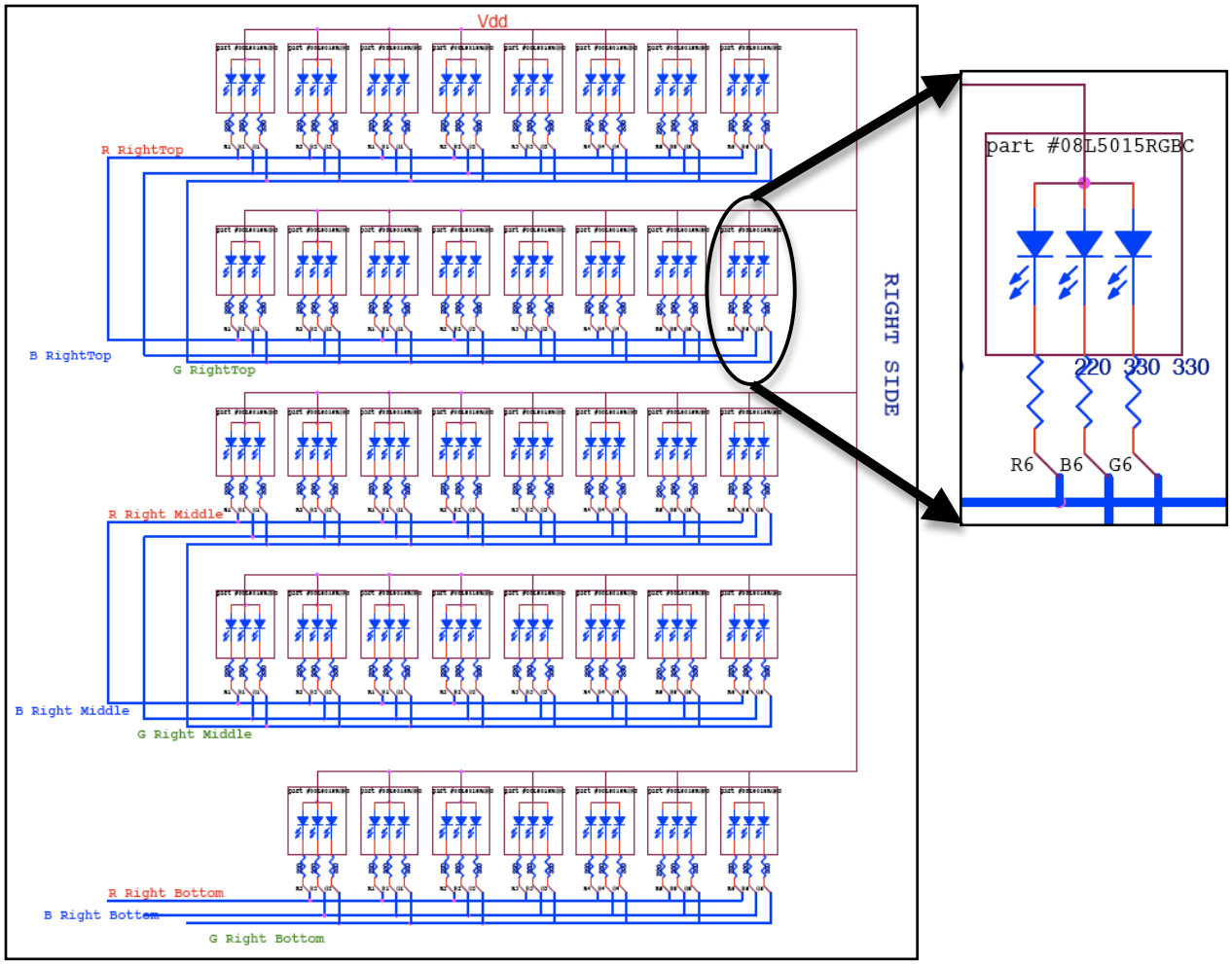


Figure 11: LED Circuit

Figure 11 shows the layout of the RGB LEDs. There are total of 78 LEDs in our system (39 for each side of the arc). The purpose of having LEDs at both sides of the arc is to have interlaced images, to achieve a better resolution. We have 3 sets of LEDs for each side, which we call Top, Middle and Bottom containing 16, 16, and 7 LEDs respectively. We can also appreciate that all LEDs are connected to V_{dd} that in our case is going to be 12 V. The LED forward voltage is 3.5V, 3.5V and 2V for the Blue, Green and Red color respectively. For that reason, resistances were put in series with each color to provide an additional voltage drop to protect the shift-register. Based on system calculations, the resistances were selected of 220 Ω (for the color Red) and 330 Ω (for the colors Green and Blue). Originally, the voltage drop in this resistance added to the voltage drop of the LEDs and left little consume almost all of the voltage provided by the sources. The original power source provided 5 volts. As previously mentioned, the LED currents had to be increased to 15mA to achieve an acceptable brightness for SWORD's display. These currents cause a higher voltage drop in the resistances and for that reason the 5 volt power source was not enough. We proceed to upgrade from a 5 V power source to a 12 V power source. With this new power source, the voltage drops are 4.95 V for the 330 Ω resistances and 3.3 V for the 220 Ω resistances. In the worst case, we have a voltage drop of 3.5 V plus 4.95 V, totaling 8.45V, which leaves large headroom from the 12V of the power supply, providing additional reliability.

The final mounting of the LEDs can be appreciated in figures 12 and 13. Figure 10 shows how the LEDs are positioned within the arc poke through its surface; during rotation the centrifugal force exerted on them keeps them in place. Figure 11 shows the connection between LEDs and their drivers (the LED Control Units). Because we had to accommodate a large volume of LEDs, we had to come up with a practical solution to connect them to the drivers; an IDE-cable was used.

Figure 12: LED's on SWORD's arc.

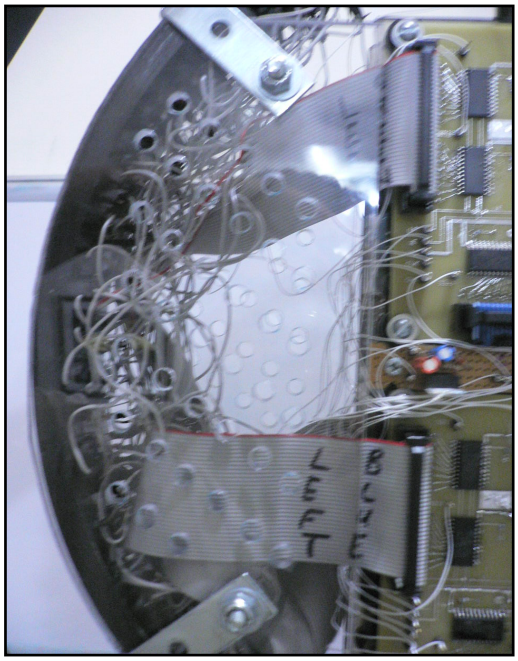


Figure 13: LED connections

4.5.2 LED Control Unit (driving circuit)

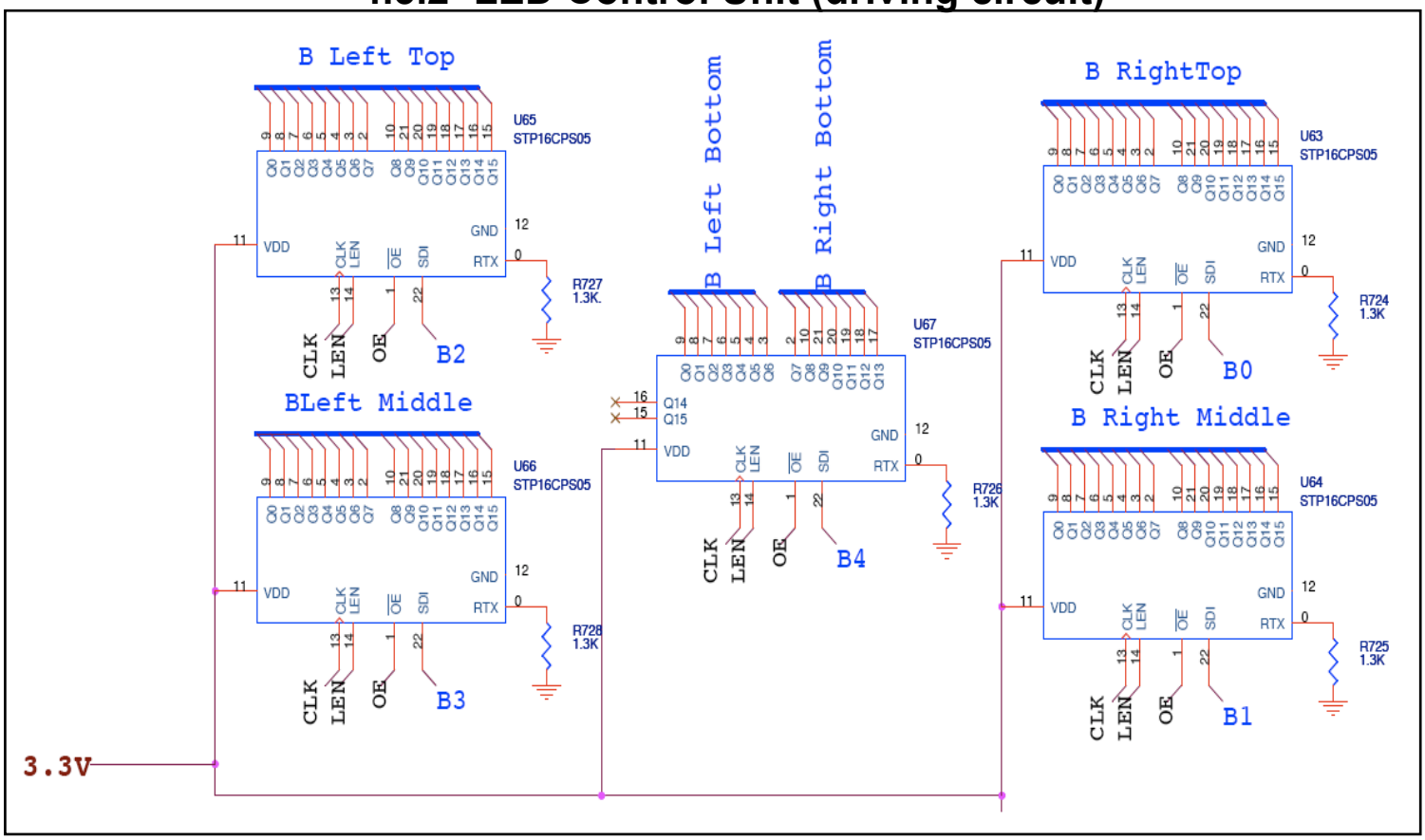


Figure 14: LED Control Unit (driving circuit)

Figure 14 shows the LED Control Unit of one color (Red). This same circuit is replicated for the other two colors. There are five shift registers (STP16CPS05). Each one is in charge of the driving of one region of the ring: Left Top, Left Middle, Bottom, Right Middle, and Right Top. Figure 15 demonstrates the layout visually.

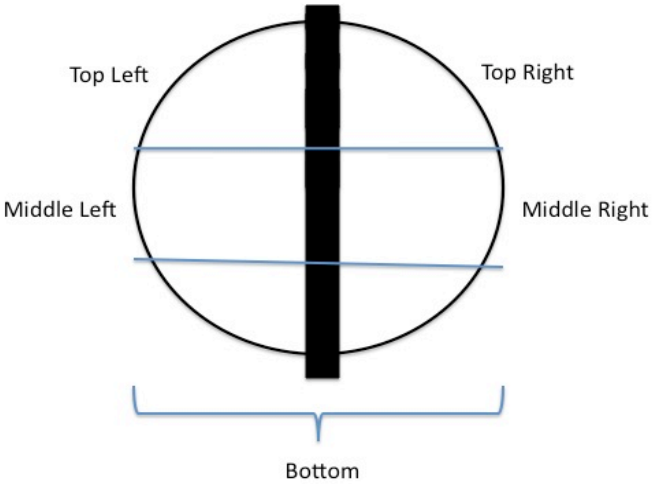


Figure 15: LED Control Unit (driving circuit)

The STP16CPS05 shift register works as a sinking current driver. The amount of current that can be drawn is adjusted with an external resistance. To calculate the R_{ext} (external resistances) the graph in figure 16 was used.

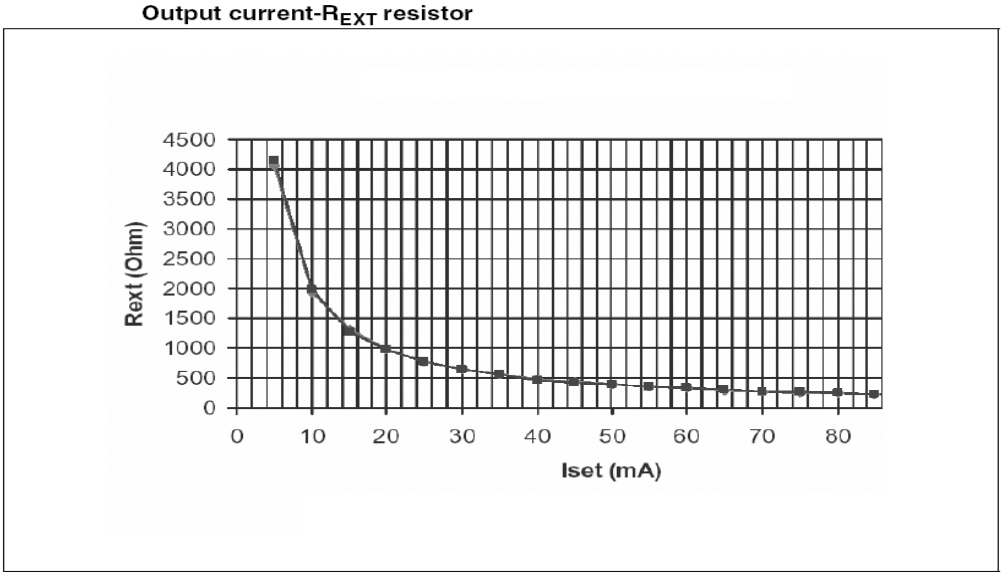


Figure 16: STP16CPS05 I_{set} vs. R_{ext} graph

For the desired current, we selected R_{ext} to be 1.3 kΩ (this produced 15mA).

It is important to mention that all control signals (CLK, LEN, OE) runs together for all 15 registers. This is possible due to the fact that these registers are voltage triggered and not current triggered. Their inputs are Schmitt trigger; hence their inputs currents should be in the range of nanoamperes. Each pin of the MCU can individually drive up to 15mA. Therefore, there is no going to be any problem due to insufficient current. The reason why each control signal is wired to the same pin is so that we are able to load data simultaneously to all shift registers at one. This minimizes the time needed to load an entire strip of data for display. In our project, execution time is crucial and the less time it takes us to load a strip the better.

As mentioned, to be able to drive all the LEDs we need three LED Control Units. However, the connections of this circuit are many, and are critical for the performance of the system. The connections have to be strong enough to withstand the high-speed rotation of SWORD. For this reason we decided to make a printed circuit board (PCB) to ensure the sturdiness of the system connections.

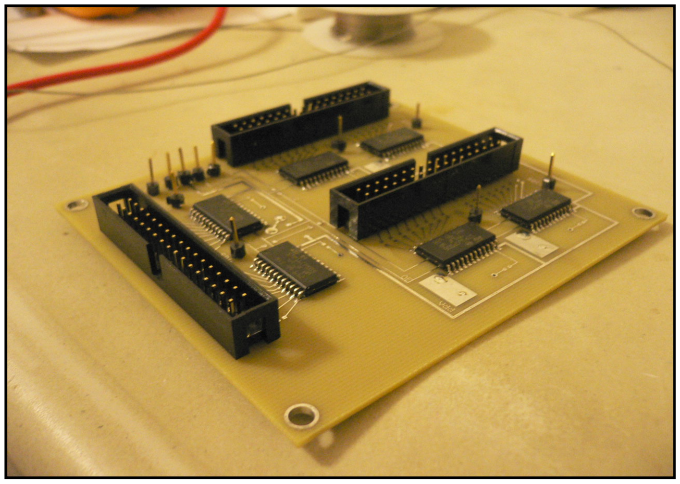


Figure 17 shows the final results of the PCB. The male's headers are there to connect the control signals to the board via wire wrapping. The IDE male sockets are for the connections to the LEDs.

Figure 17: Completed PCB used for LED Control Units

Figure 18 shows the other side of the board, where the rest of the connections to the sockets were designed to be the ground of the circuit in order to reduce the resistance in the ground connections. This is a common practice due to the formula: $R=\rho*L/A$ where R is resistance, ρ is resistivity, L is length and A is area. The idea of a whole side being ground is to have a wide area to reduce the resistance.

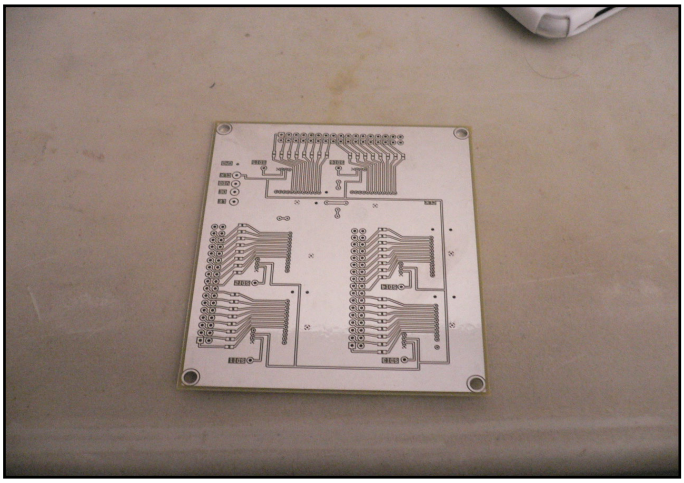
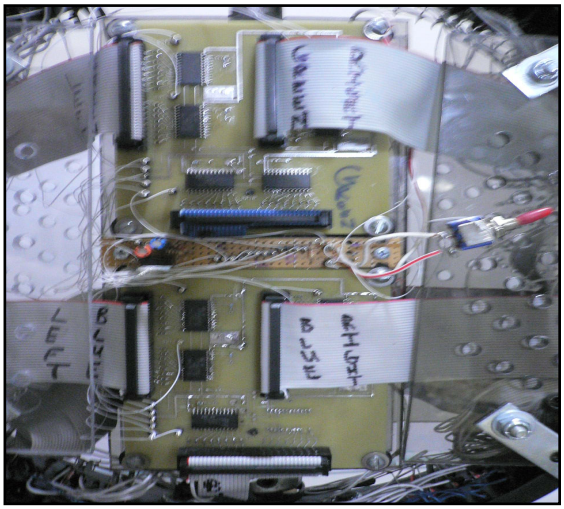


Figure 18: LED Control Unit PCB (Upside down)



The final mounting of the PCB can be appreciated in figure 19.

Figure 19: LED Control Unit PCB (fully integrated)

4.5.3 Voltage Regulator, Bluetooth®, and Bluetooth® Voltage Regulator

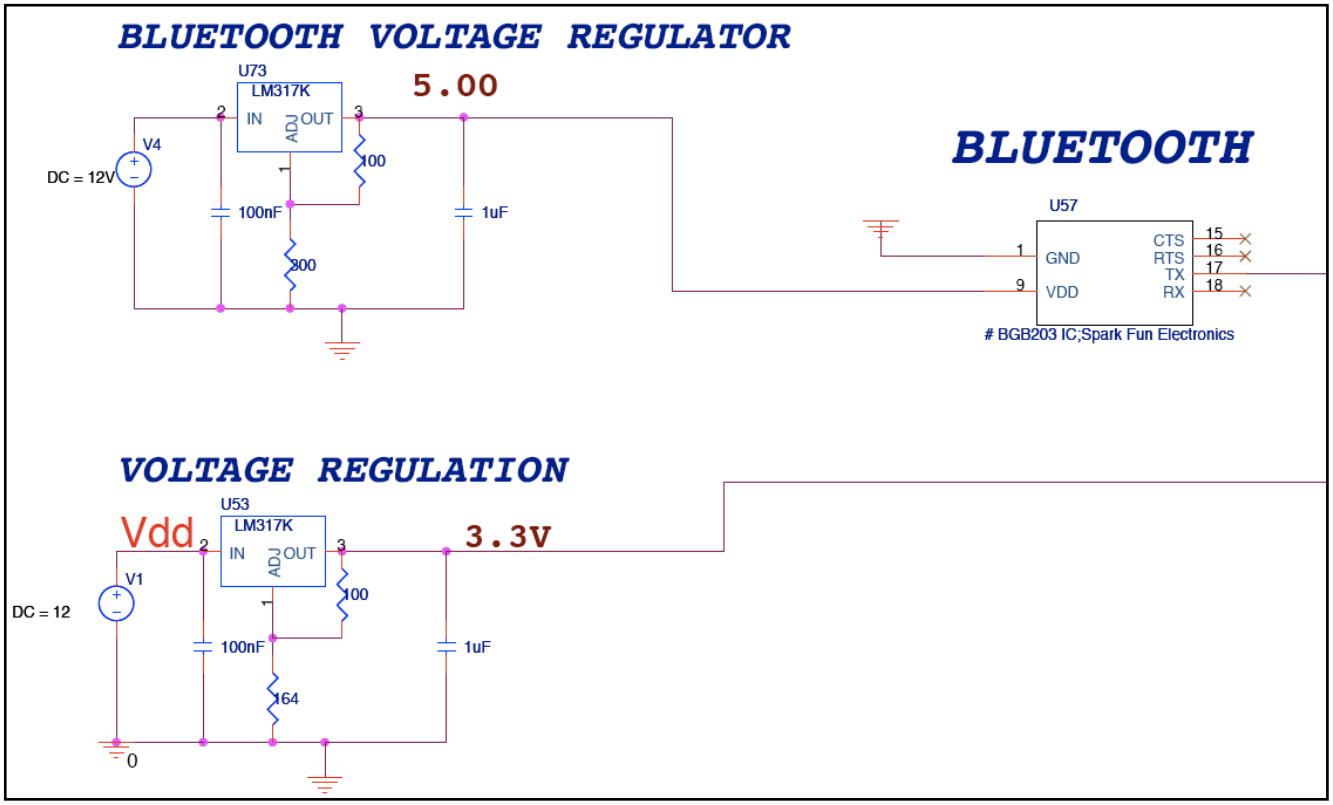


Figure 20: Voltage Regulator, Bluetooth®, and Bluetooth® Voltage Regulator circuits

The two small circuits on right-hand side of figure 20 all have in common one thing and it is the voltage regulator (LM317). This IC is ruled by the next equation: $V_{out} = 1.25 * (1 + (R2/R1))$. To regulate at 5 V, R2 was selected to be 300 Ω and R1 was selected to be 100Ω. In reality the resistances used where 330 Ω and 100 Ω yielding an output voltage of 5.38V; in both devices that use this regulator, this doesn't become a problem. The Bluetooth can work from 3V up to 6V. For the voltage regulator that has to give an output of 3.3 V, the ideal resistances would have been 164 Ω and 100 Ω but instead the resistances used where 1.7 kΩ and 1 kΩ, yielding an output voltage of (approximately) 3.375 V. The differences in these values will not bring harm to the performance of the system because the voltages stay within operating range. Since the Bluetooth® will only send data to the MCU, only one connection is needed: from

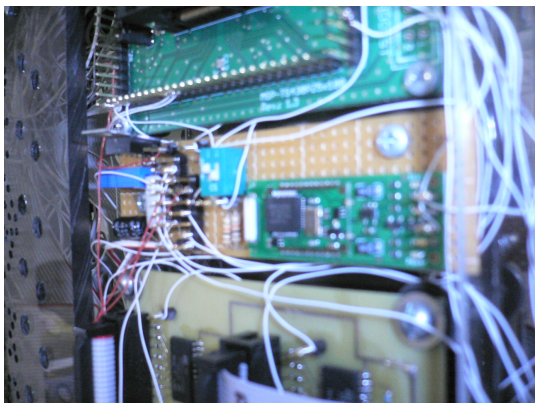


Figure 21: Bluetooth® fully integrated circuit

the transmitter pin (TX) of the Bluetooth® to the receiver pin (RX) of the MCU (pin 40). The Sparkfun™ Bluetooth® module that we bought, only requires a power to operate. When connected to the MCU, no further hardware considerations are needed aside from proper software adjustment, particularly the device baud rate. Figure 21 shows the integrated Bluetooth® circuit.

4.5.4 Optical Sensor (Optocoupler): Sender and Receiver Circuits

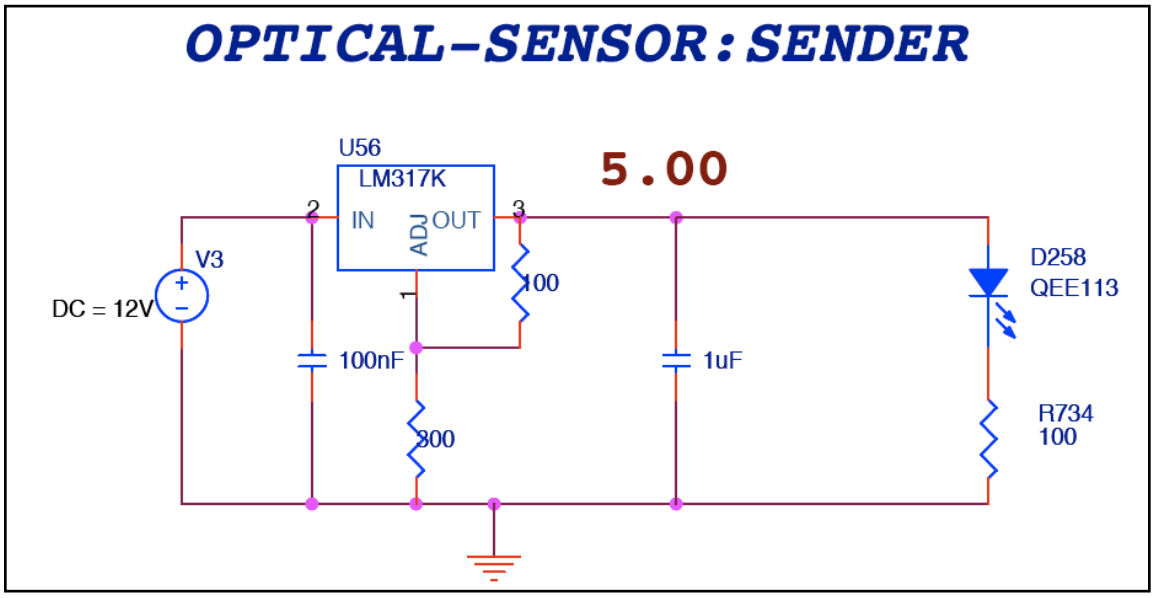


Figure 22: Optocoupler Sender Circuit

The optocoupler sender is a straightforward circuit, as seen in figure 22. For it, we used the QEE113 Plastic IR LED. We just need to apply 1.5 V for it to emit the IR wave and its current is regulated with resistances. The QEE113 can work properly from 5mA to 100mA. Ee decided to work on almost half its maximum current (around 40mA). To achieve this current we needed relatively low resistances. If we had decided to go with the voltage of the rest of the system (3.3 V) the resistances needed would have been (3.3 V-1.5 V) / 40 mA = 45 Ω. We did not have a 45 Ω resistance at the time, but we elevated the voltage value applied, which also increased the resistance needed for the same current. We changed the voltage to 5 V instead of 3 V. The value of the resistances then turn out to be (5 V - 1.5 V) / 40mA = 87.5Ω. Since the resistance that we put on the voltage regulator was of 330Ω instead of 300Ω, the output voltage turned out to be 5.38 V. This meant that the resistance we really needed was (5.38 V - 1.5 V) / 40mA = 97Ω. We therefore rounded up to 100Ω.

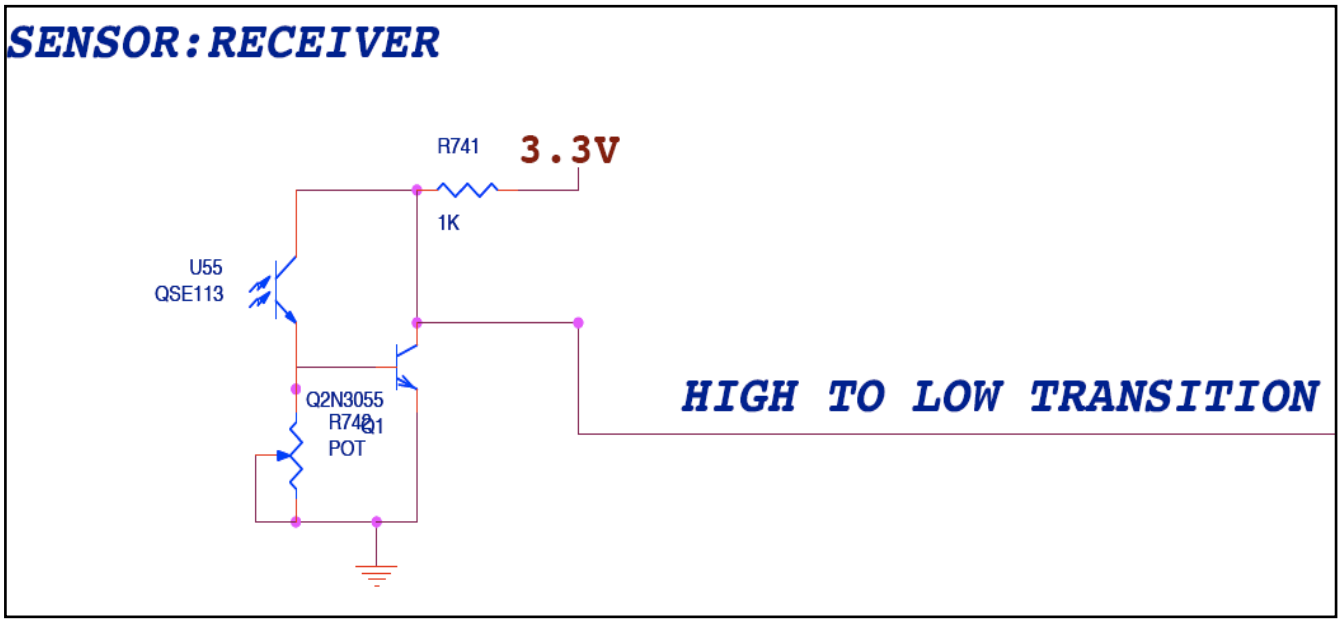


Figure 23: Optocoupler Receiver Circuit

The QSE113 is a phototransistor that is driven by an infrared light. This type of transistor was intentionally selected that way so that external or ambient lights (as well as the lights emitted by the RGB LEDs themselves) did not affect its reading. The configuration of this circuit enables us to amplify its susceptibility by adjusting the potentiometer. Since the system is going to be rotating at high speed and the phototransistor has to keep up with that speed if, we want it to work properly. How can susceptibility to light be amplified? The rise time of the QSE113 is of 8 microseconds, but that time is relative to the intensity of infrared light that reached it. The closer the receiver is from the transmitter, the higher the intensity received, and thus, the faster the time response. To ensure the response time of the circuit a Darlington pair configuration was implemented. In this configuration, the job of the phototransistor is reduced to only serve as a driver for the NPN transistor. This NPN transistor was connected to the pin of the MCU. The NPN transistor can be fully driven before the phototransistor can and with the proper adjustment of the potentiometer, the desired response time can be reasonably achieved. This configuration provides a high to low transition that will serve as an interrupt to the system. The pin used for this interrupt is P1.7.

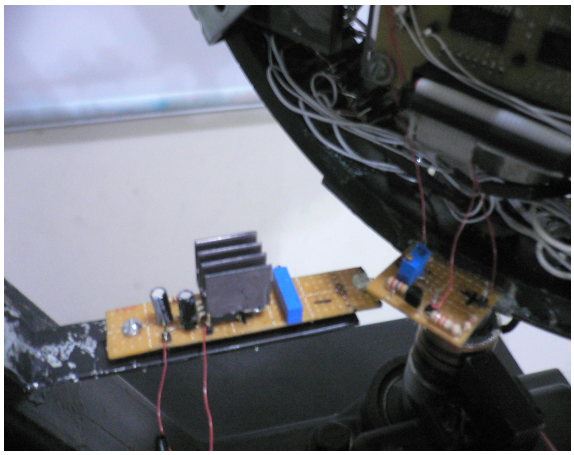


Figure 24: Optocoupler fully integrated circuit

The final mounting of both circuits was not simple and is seen in figure 24. Various considerations were observed. First, the distance between both circuits had to

be minimum. For minimum distance we glued (with epoxy) the receiver into the arc. We elevated the sender circuit as close to SWORD's arc as possible, taking in to consideration the distance from the arc itself. A second consideration was vibration. If the extended platform that holds the sender circuit vibrated too much, it was possible it could touch the arc. If this were to happen during normal operation, we could inflict harm upon SWORD's operator or bystanders. The last consideration was that during SWORD test rounds, we noticed that the sender circuit overheated extraordinarily fast, and therefore a heat sink was added.

4.5.5 MSP430F5438

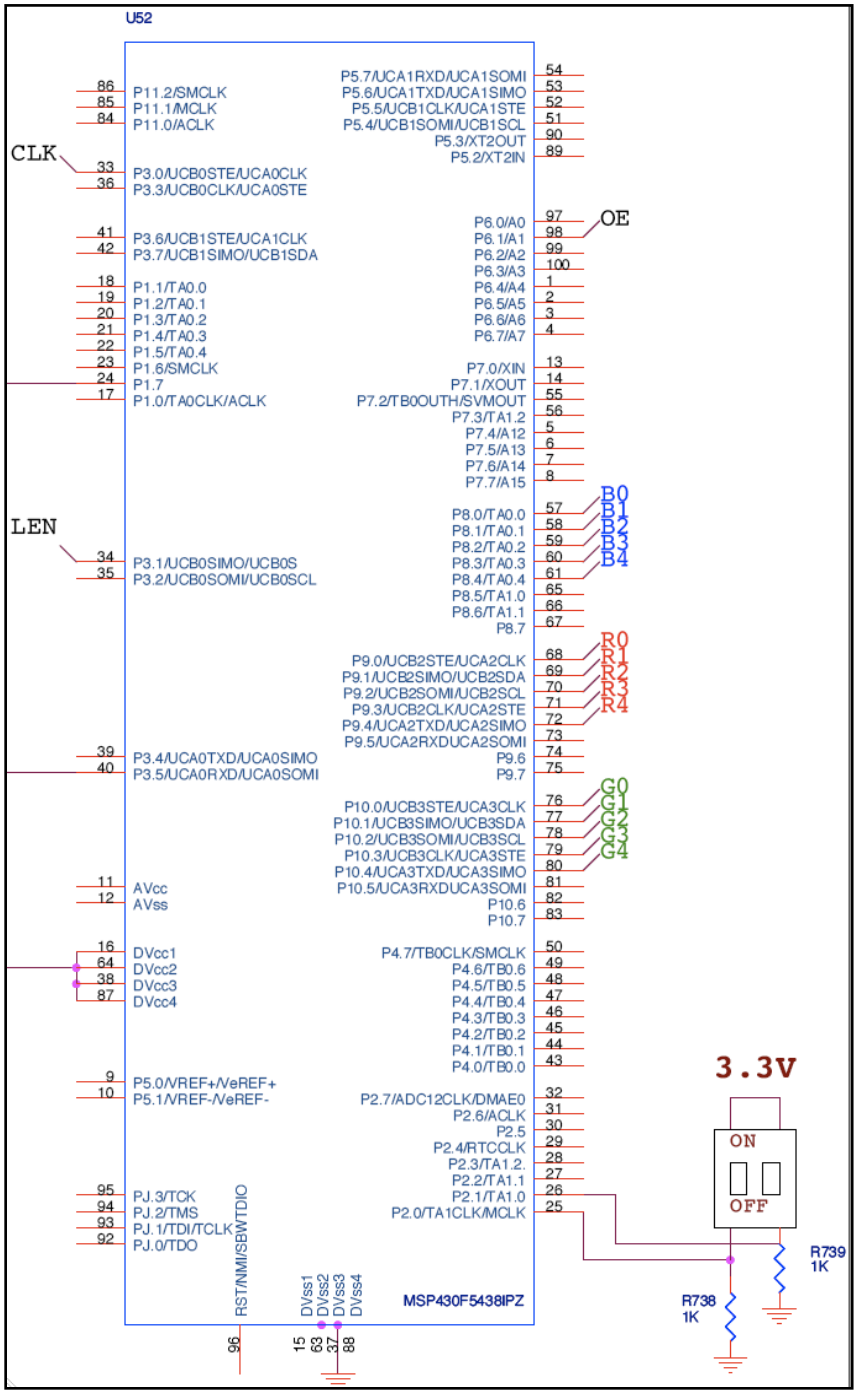


Figure 25: MSP430F5438 Schematic

Here we can appreciate all of the connections to the MCU. Ports 8 through 10 are used for the serial data in connections to all of the shift registers. Pins 2.0 and 2.1 are used to read the initial state of the switch, to load a default image on SWORD. Pin 3.5 connects

to the Bluetooth®. P3.1 connects to the Latch Enable connection, P6.1 to the Output Enable connection, and P3.0 to the Clock Signal connection. We remind the reader that all shift registers share the same control signals. Finally P1.7 goes to the optocoupler receiver circuit.

4.6 Memory Map

Our analysis of the memory map of our architecture is divided into three sections: First, an analysis of the registers in the MSP430F5438. Second, an overview the memory organization will be discussed. Lastly, we will outline the memory usage of our own particular application.

4.6.1 Microcontroller Registers' Description

The MSP430F5438 has a 16-bit RISC architecture that is highly transparent to the application. All of its operations, other than program flow instructions, are performed as register operations in conjunctions with seven addressing modes for source operand and four addressing modes for destination operand. It contains 16 16-bit registers that provide one cycle execution time for register-to-register operations. The registers from R0 to R3 are dedicated as program counter, stack pointer, status register and constant generator respectively. The other 12 registers are general-purpose registers.

4.6.2 Memory Organization

Table 3: Memory organization of the MCU

Description		Address	Size
Main: interrupt vector	Flash	00FFFFh – 00FF80h	127 B
Main: code memory	Flash	045BFFh – 005C00h	256 KB
Main: code memory	Bank 3	03FFFFh – 030000h	64 KB
	Bank 2	02FFFFh – 020000h	64 KB
	Bank 1	01FFFFh – 010000h	64 KB
	Bank 0	045BFFh – 040000h 00FFFFh – 005C00h	64 KB
	Total Size		256 KB
RAM	Sector 3	005BFF – 004C00h	4 KB
	Sector 2	004BFFh – 003C00h	4 KB
	Sector 1	003BFFh – 002C00h	4 KB
	Sector 0	002BFFh – 001C00h	4 KB
	Total Size		16 KB
Information Memory (Flash)	Info A	0019FFh – 001C00h	128 B
	Info B	00197Fh – 001900h	128 B
	Info C	0018FFh – 001880h	128 B
	Info D	00187Fh – 001800h	128 B
	Total Size		512 B
Bootstrap Loader (BSL) memory (Flash)	BSL 3	0017FFh – 001600h	512 B
	BSL 2	0015FFh – 001400h	512 B
	BSL 1	0013FFh – 001200h	512 B
	BSL 0	0011FFh – 001000h	512 B
	Total Size		2 KB
Peripherals	*Total Size*	000FFFh – 000000h	4 KB

The memory of the MSP430F5438 is divided into segments. The addresses to which every of them extend to are detailed in Table 3. The following gives a brief explanation of every of them:

- Main: code memory – This is where all of the code for the system resides in the MCU.
- RAM – Has a specific amount of sectors that are particular for every MCU. For our MCU, it has 4 sectors.
- Information memory – flash memory
- Bootstrap Loader (BSL) – It allows the users to program the flash memory or RAM using UART serial interface.

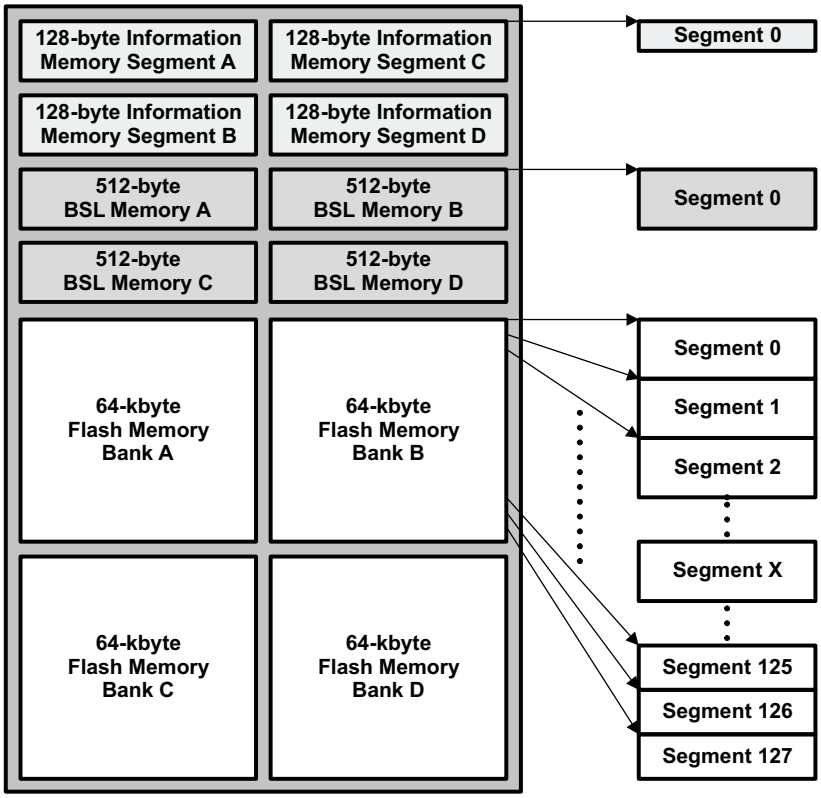


Figure 26: 256-KB Flash Memory Segments Example

Table 4: Memory organization of the interrupt vector addresses of the MSP430F5438

Interrupt Source	Interrupt Flag	System Interrupt	Word Address	Priority
System Reset Power-Up External Reset Watchdog Timeout, Key Violation, Flash Memory Key Violation	WDTIFG, KEYV (SYSRSTIV)	Reset	0FFFEh	63, highest
Watchdog Timer_A Interval Timer Mode	WDTIFG	Maskable	0FFF4h	58
USCI_A2 Receive/Transmit	UCA2RXIFG, UCA2TXIFG (UCA2IV)	Maskable	0FFE8H	57
TA0	TA0CCR CCIFG0	Maskable	0FFECh	54
TA0	TA0CCR1 CCIFG1…TA0CCR4 CCIFG4, TA0IFG (TA0IV)	Maskable	0FFEAh	53
I/O Port P1	P1IFG.0 to P1IFG.7 (P1IV)	Maskable	0FFDEh	47

4.6.3 Memory Usage

The code of our application extends from address 0x002884 till address 0x00B26E in the flash memory of the MCU. The following describes the various sections of our code and the memory space that they occupy:

- Global Programming variables – Address space from 0x006A1A to 0x001C40
- Letters repository for text display – Address space from 0x005C00 to 0x006A1B
- Text display buffer – Address space from 0x001C44 to 0x002883
- Interrupt Service Routines (ISR's)
    - System Reset – Address space from 0x00B39A to 0x00B3B4
        - This ISR includes the system-reset routine.
    - USCI_A2 Receive/Transmit – Address space from 0x00B414 to 0x00B424
        - This ISR include the instructions that are executed whenever a UART interrupt occurs.
    - TA0 – Address space from 0x00B40E to 0x00B412
        - This ISR include the instructions to be executed whenever a TA0 interrupt occurs. These instructions are used to refresh the data of the display.
    - I/O Port P1 – Address space from 0x00B402 to 0x00B40C
        - This ISR includes the instructions that are executed whenever a signal from the optocoupler is received.

4.7 General Hardware Afterthoughts, and Trustworthiness of Design

We will now outline some observations of the system hardware that consider power consumption, problems encountered during hardware development, and finally the trustworthiness of the hardware design.

The system's power consumption was great due to the DC Motor. In both operating modes, the DC Motor contributes the greatest power (as expected). Nevertheless, during Image Display mode, the rest of the system's power consumption could go up to 11.658 W due to the amount of LED's in the system. To provide enough currents for the LEDs we will need to provide over (15 mA + 15 mA + 15 mA) * 78 = 3.510 A. Probably this number needs to be revised and taken up to 4A or higher to also provide current for the rest of the system.

Due to the high power and current consumption we have chosen to provide power to the system through a computer power supply connected directly to a wall socket (120 V_{AC}). To provide the power to the motor, a DC motor controller is used. It is also connected directly to the wall. This way, we can provide the power needed for the system for long periods. These considerations allow the system to work until a component fails.

During prototype testing, two main problems arose. First, the current was not sufficient for the brightness that we wanted due to the duty cycle of the LEDs and second, there was a voltage drop in the power supply as more LEDs were turned on. To solve the first problem we changed the external resistances of the drivers, and we were able to achieve the desired brightness by increasing the current passing through the LEDs. For the second problem we simply selected another power line for the power supply that could work properly, even with a voltage drop of 3.5 volts. The voltage sources were changed from 5.25 to 12 volts. Note: More currents for the driver also demanded more voltage, due to voltage drop in the resistances. That is why the voltage drop that can be handled is of 3.5 V and not 7 V.

Failure in the system can be caused from a variety of sources. First of all, the system is very susceptible to static currents. For that reason a mechanical ground was added. However, when the when plugging into a wall socket, the user needs to be ensure that the connection provides a ground so that the safety feature works properly. That was a problem our team learned to manage the hard way; it caused us several system failures and electrostatic damage to parts.

4.8 Level of Hardware Completion

In order to describe the Hardware termination level, we present the following chart. It outlines details of completed hardware tasks, considerations made, the team member responsible for that particular hardware task, and the level of termination in percentage.

Table 5: Level of Hardware Completion

Task	Considerations Taken	Teammate(s) Responsible	Level of Completion
Mounting Board Design	1. SWORD Interior Dimensions 2. Vibration produced by SWORD 3. Speed of SWORD's arc rotation	Edward	100%
PCB Design	1. Dimension of PCB 2. Electrical Connections 3. Components to be placed in PCB	Willie & Edward	100%
PCB Component Soldering	1. Tools needed for Soldering	Willie	100%
PCB Testing	1. Electrical Connection	Willie	100%
Voltage Regulator Design	1. Operating Voltages	Willie	100%
Optocoupler Design	1. Range of device 2. Duty cycle	Willie	80%
IDC Cable Terminations	1. Length	Edward	100%
LED Soldering	1. Tools needed for soldering 2. Mapping LED cables to correct colors	Willie & Edward	100%
MCU Boards Wiring	1. Match hardware connection with software expectations	Willie & Rogelio	100%

Figures 27 and 28 visually demonstrate the level of Hardware completion achieved with SWORD.

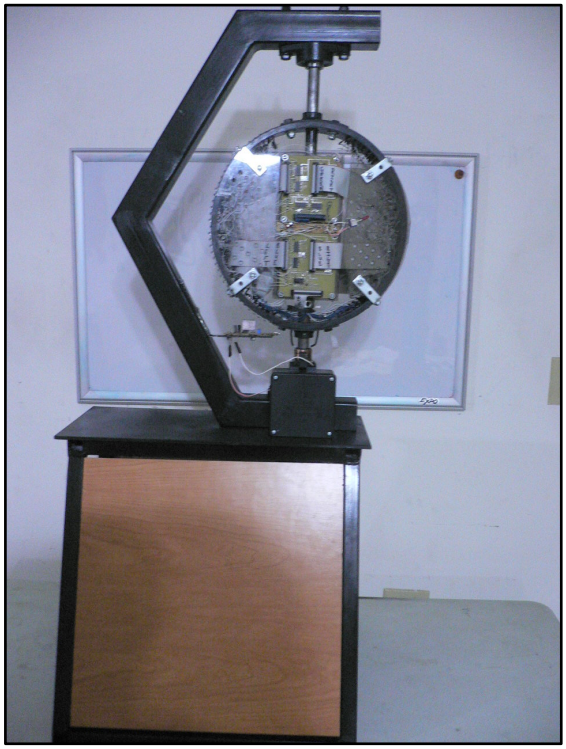


Figure 27: Total Hardware Completion

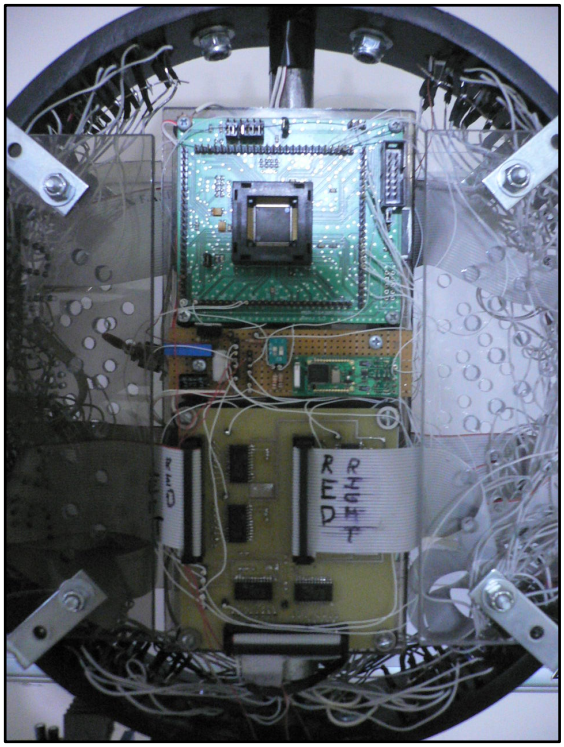


Figure 28: SWORD circuitry (forward facing)

5 Software Design and Development

5.1 Overall Software Functionality

SWORD has two main software components. The first is the Main Code, and is the software portion that takes care of image display, houses pre-defined images, and handles external user interaction. The second component is the User Interface, which provides software controls via SWORD Graphical User Interface (GUI) that allow users to externally change images in and write text for SWORD's display. The two software components act together to provide the complete SWORD experience. However, the SWORD User Interface cannot run unless the Main Code is running.

Upon system power-up, the Main Code initializes system variables and prepares the system interrupt service routines (ISR). The system then polls the onboard DIP switch to determine which image will be loaded for initial display. Images are represented in code by arrays that are 80 rows by 196 columns. Several shortcuts allow us to get away with representing the same data in a smaller array, but these will be outlined later (for now, assume that each image is encoded in an array that is 80 x 196). After determining the state of the switch, it stores a flag that indicates the array to be displayed. The system then enters an infinite loop displaying the image throughout the entire 196 steps until one of two things happen: 1) the user decides to switch the image or 2) the user decides to write text to SWORD. In both cases, the corresponding image is loaded, and the infinite loop resumes. Images are displayed by traversing the corresponding image array and sending the data that encodes the image to the LED Control Units. These indices are then reset by software when they have reached the end of the image array. What also helps the loop cycle is the optocoupler ISR. When activated (upon one complete revolution), the ISR forces the indices to reset. This is a double check on the image loop and allows us to ensure that the image begins from zero every time the SWORD arc completes one revolution.

When the Main Code is activated, it enables the Bluetooth® receiver located onboard SWORD's circuitry. When the Bluetooth® begins broadcasting its signal, the User Interface can detect it. The User Interface does not allow the user to do anything until it is connected to the Bluetooth, upon which all GUI functions are enabled. By the same token, the User Interface does not allow the GUI to close until it is disconnected from the Bluetooth. This is to avoid any problems with serial communication ports that are left open by mistake. The User Interface is what allows the user to interrupt the infinite display loop within the Main Code, and provides methods to change the image being displayed at the time, as well as methods to write text and send it for display. We will exam these two software components in detail in the following sections.

5.1.1 Main Code

The Main Code takes care of setting up the environment necessary for proper functionality: variables, interrupt service routines (ISRs), system operating speed, . Among the declared variables, the most important ones are the image arrays (which are stored in ROM), and the image indices (which are stored in RAM). The ISRs that are declared are the Timer ISR, the UART ISR, and the Port 1 ISR. The Timer ISR is used to properly time the image display. As mentioned, an LED changes color every 463.8 microseconds. A timer is set up in Capture-Compare mode as such that every 463.8 microseconds an interrupt is generated. The interrupt tells the system that it is time to display the next step in the revolution, by setting a flag that is later checked by the infinite display loop. When the system displays the next step, it begins loading the data for the following step. Ideally, this means that the MCU has 463.8 microseconds to load the next step in sequence. This algorithm is repeated throughout the entire display loop, which means that whenever image step "n" is being displayed, image step "n+1" is being loaded onto the shift registers.

Because the system enters an infinite display loop, the system **never** enters any Low Power Mode level. The reason for this is that past experience dealing with Low Power Modes has lead to buggy code: for some reason the system Program Counter and Status Register are corrupted when entering low power modes. This problem has been recognized by Texas Instruments in the device errata [TIE]. In the infinite loop, the system constantly checks two things:

1. Are we ready to display the next strip?
2. Are we ready to load the data for the next strip?

The first question is answered by checking the flag that is set by the Timer ISR. If the flag is set, SWORD displays the data that was set in the previous step, and proceeds to load the next step in sequence. If the answer to the first question is "yes", then the answer to the second question is immediately "yes".

The Main Code also has the ability to detect when SWORD's arc has completed one complete revolution (via the Optocoupler ISR), and if so, it resets the image indices used in the orb display. This allows the step just after the optocoupler step to be loaded with the data at the start of the image array. In turn, this allows the image to remained "fixed", since SWORD is activating the same sequence of LEDs at the same positions along the respective LED trajectories.

5.1.2 User Interface

The SWORD User Interface (UI) is a mechanism by which a user can interact externally to the SWORD primary system. Loading the file "SWORD_GUI.jar" accesses the UI. Upon loading, all functions are disabled, since a Bluetooth® connection to the SWORD primary system is not available. For the user to be able to

interact with the system, he or she must open up a serial link to the Bluetooth® receiver on SWORD. When the link is established all GUI options are enabled and the user will be able to select whether to display images or text. If the user decides to display images, the user will be able to select from two pre-selected images. If the user decides to display text, the user will be able to write text up to 11 alphanumeric characters.

5.2 System Flowcharts

Flowcharts were utilized to plan system code. There are two sets of flowcharts, corresponding to the two main software components of our system: Main Code flowcharts, and User Interface flowcharts. All flowcharts are listed together. Below is a list of which flowcharts belong to which software components.

Flowcharts for Main Code

- SetupEnvironment
- System Main
- Shift_NextStrip
- Analyze Msg
- PopulateTxtBuffer

Flowcharts for User Interface

- GUI Main
- Image Mode
- Text Mode
- Connect to Bluetooth
- Send message to Bluetooth

5.2.1 SetupEnvironment

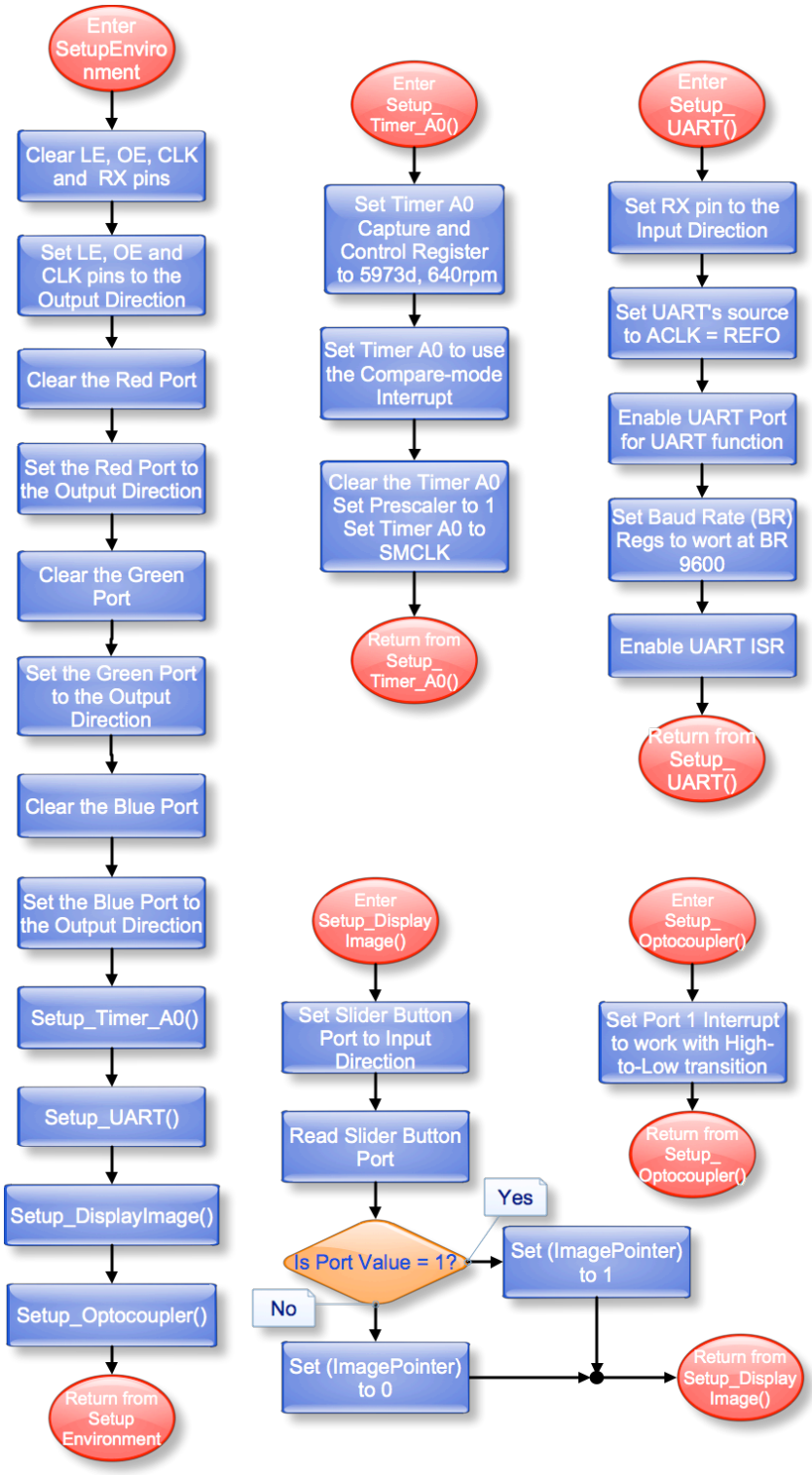


Figure 29: SetupEnvironment flowchart

The SetupEnvironment code configures all system parameters and variables. It sets all the ports to work in the needed direction, sets the UART to work with a 9600 Baud Rate, sets TimerA0 to work on our required frequency, and sets the image pointer variable based on the reading of the onboard switch. This variable determines the image that will be displayed upon startup, and can have two states: one for each displayable image.

5.2.2 System Main

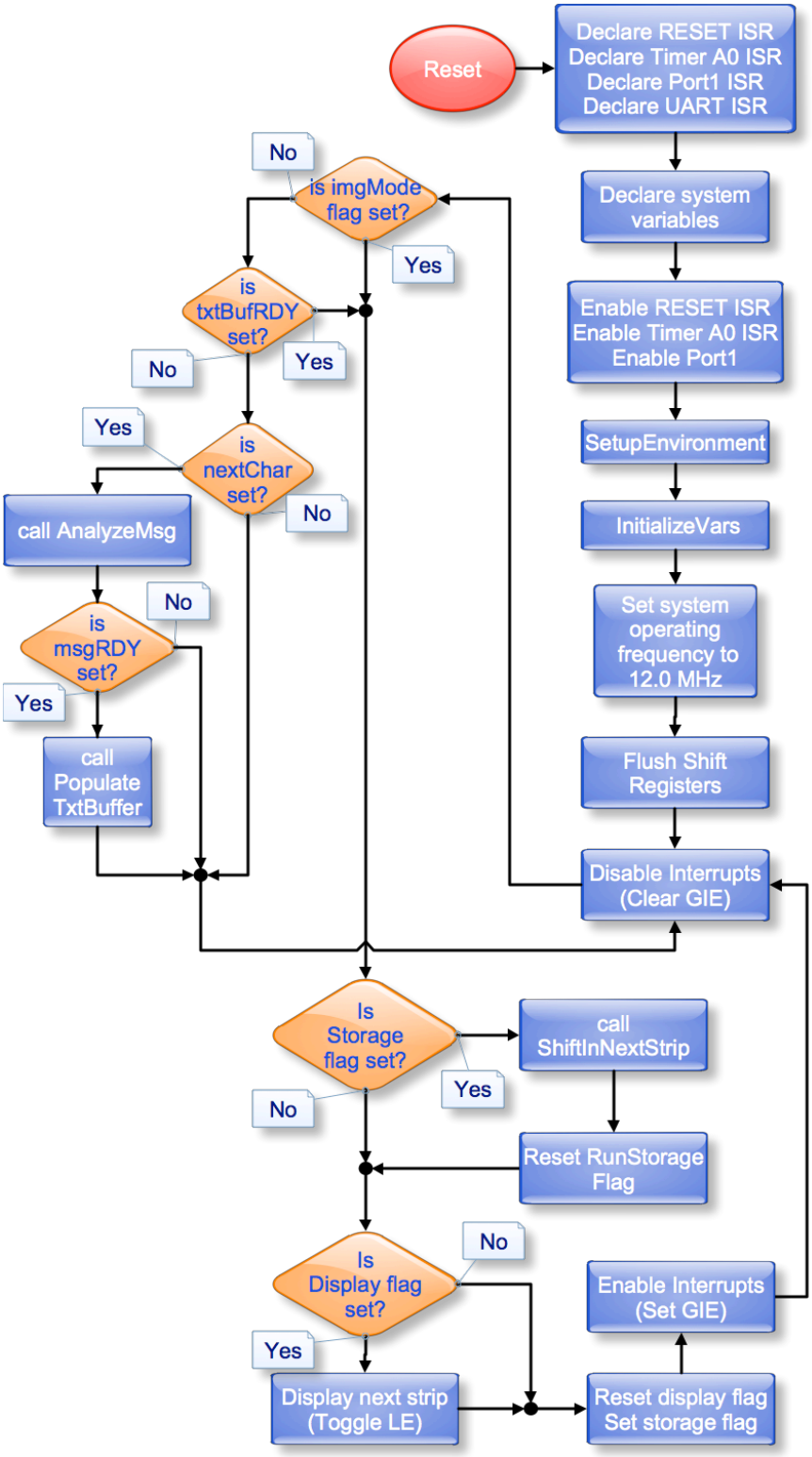


Figure 30: System Main flowchart

SWORD's main code is what takes care of the display, with help from support functions. After setting up the system environment and variables, it enters an infinite loop. This loop checks two things.

First, it checks if it should display the next step in the sequence. The next strip should be displayed when SWORD's LED strip goes from strip (n) to strip (n+1). As explained in our system calculations, this event happen every 463.8 microseconds. The main code determines whether the event has passed by checking the flag *run_display*. The *run_display* flag is set by Timer A0. Timer A0 (as will be seen later) is configured to activate an interrupt every 463.8 microseconds. Whenever the main executes the code to display the next strip in sequence, it prepares the strip that follows it. That is, if SWORD is currently displaying strip (n) and is then signaled to display (n+1), it proceeds to prepare the strip (n+2). When each strip is being displayed, SWORD loads the next strip to the shift registers controlling the LEDs. This guarantees enough time for the LEDs to be prepared for the next display in the sequence. The code that takes care of preparing the next strip is within the subroutine Shift_NextStrip.

The second thing the main code checks is what exactly is supposed to be displayed on SWORD at any given moment. We remind the reader that SWORD has two modes of display: text and image. If one mode (or the other) is selected, corresponding flags are selected as such that SWORD's main code can act accordingly. SWORD's text display is created dynamically through the user's input. The text processing and image creation process is encapsulated by the subroutines "AnalyzeMSG" and "PopulateTxtBuffer".

5.2.3 Shift_NextStrip

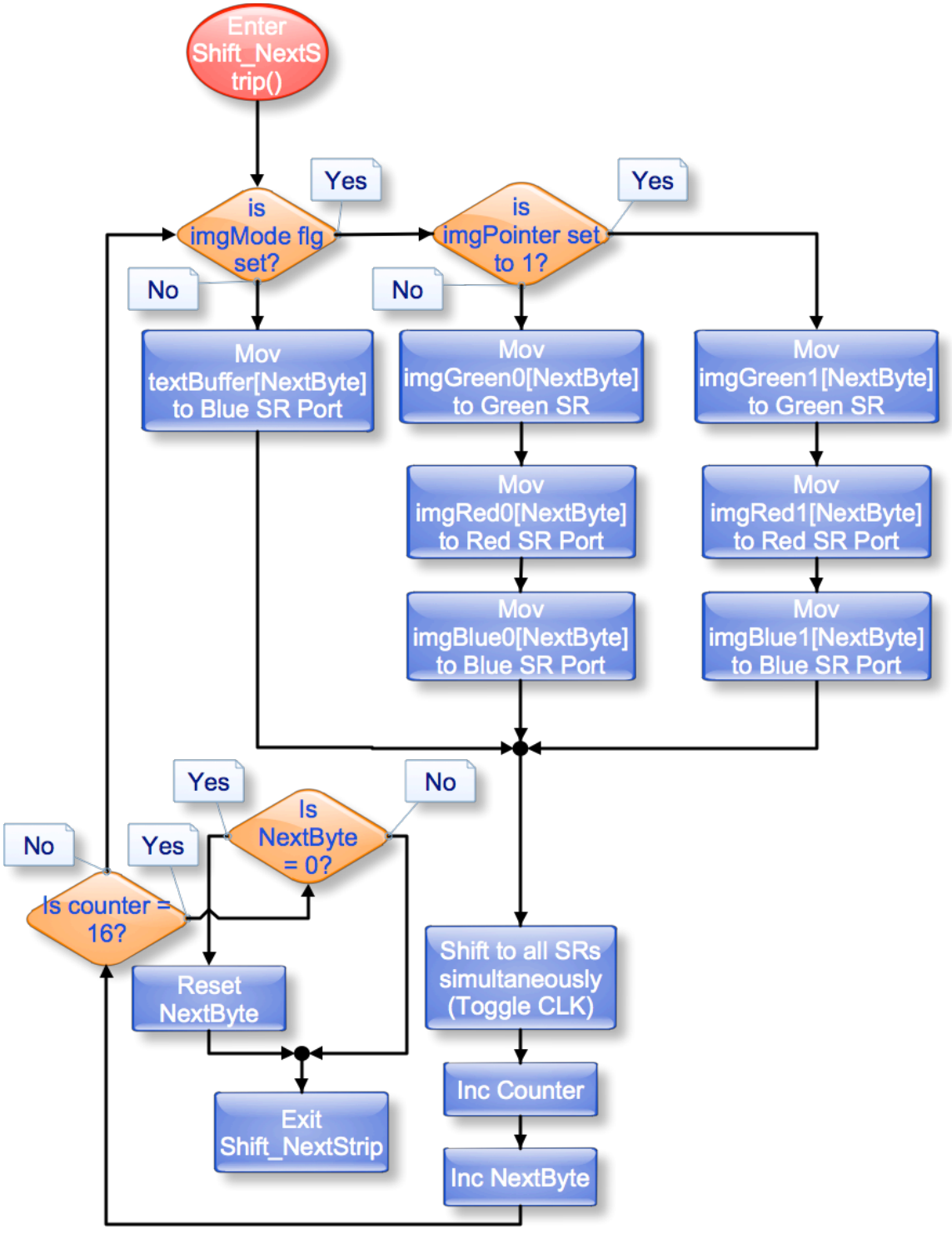


Figure 31: Shift_NextStrip() flowchart

When this subroutine is called, it checks if it is displaying text or a pre-coded image. If it is text, the subroutine loads the next byte to the blue shift registers. If it is

displaying images then it has to choose which image is loading to load the next byte for each color. After either image or text bytes have been loaded, the clock signal of the shift registers is set to load them into the registers.

Afterwards two counters are incremented if the 16 bytes of the strips have been loaded and the other one keeps track of which byte was the last loaded into the shift registers. The 16 bytes counter is used to determine whether or not the strip has been fully loaded. If the strip has not been completed, then the subroutine has to restart to load the next byte of the data it is loading. However, if the strip is finished then the other counter is taken into consideration to check if the last strip was loaded. In case that this has not occurred, the subroutine simply ends and waits to be called again. Nevertheless, if the last strip was loaded, strip 0, then the last counter is reset so that the next time the subroutine is called it starts to display the image or text from the beginning. This occurs every revolution. It is important to mention that SWORD rotates from right to left and English is written from left to right, therefore we start displaying in the last byte and end in the byte 0.

5.2.4 Analyze Msg

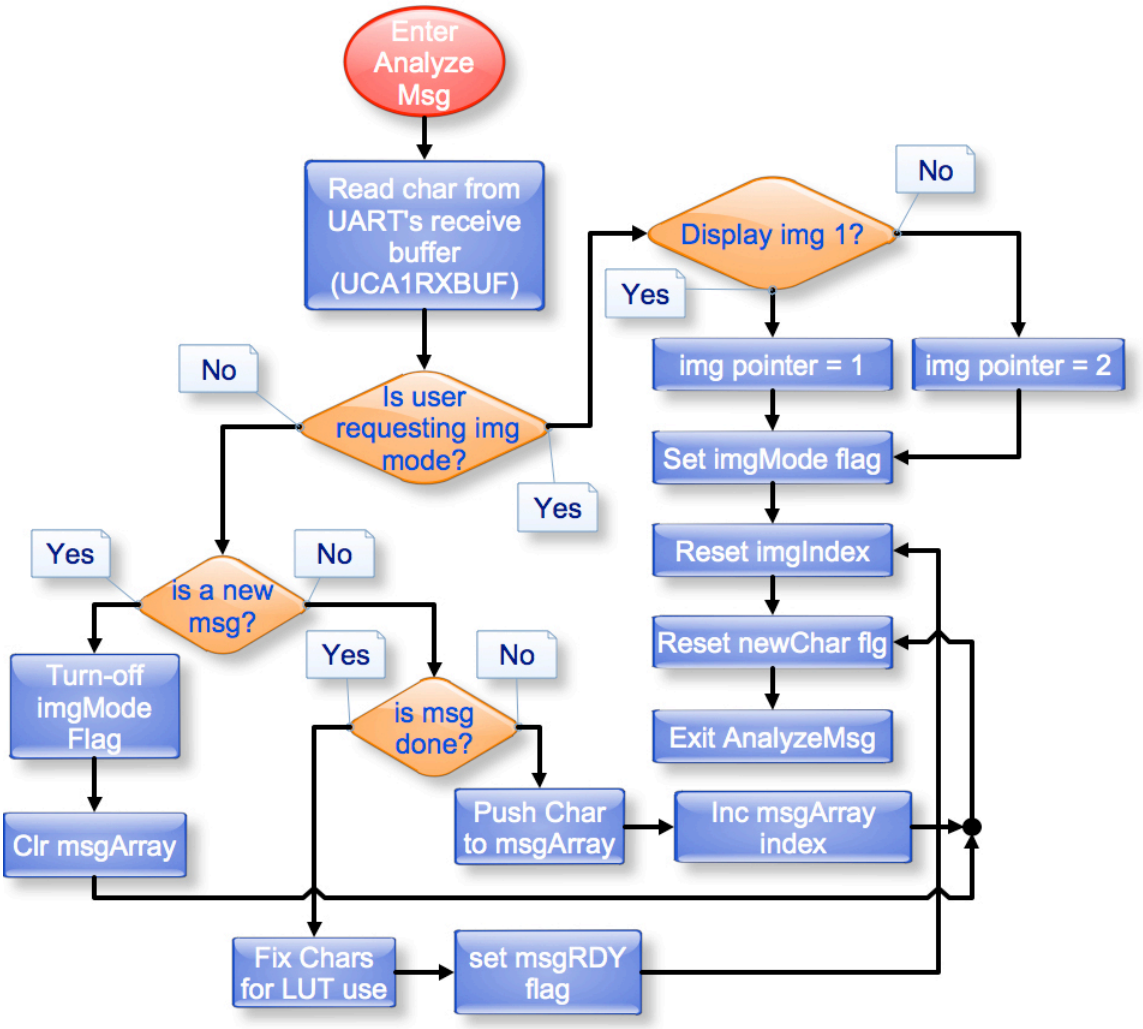


Figure 31: Analyze Msg flowchart

SWORD's user interface allows sending text and change between the available pictures over Bluetooth. This subroutine handles these options. It starts by reading the UART RX buffer (i.e. UCA1RXBUF). After this it's determined whether the user is requesting image mode or text mode by using reserved characters.

If the user is requesting image mode the next question is; is the user requesting image one or two? Each image has its own reserved character as well and with this the image pointer gets set to the corresponding image.

On the other hand, if the user is requesting to display a new message prior to be sent the message is converted to the following: "*" + message + "." The purpose of this

is to recognize easily when a there is a new incoming message and reset all needed variables such as the message array, message index, etc. This is possible with "*". Also with "." It's known when the transfer is over and the message can be processed.

As the message is transferred to SWORD is collected into a message array one byte at time. Once the transferred is over it is processed and the message ready flag gets set. Since SWORD does not represent all characters in the ASCII table the characters had to go through a fix in order to have a proper character look-up table. The fix is to subtract to necessary amount depending on the character, mainly if it's a letter or not. Once this all there is left is a message array with sequential number which can be used to access the proper character when populating the text buffer, we shall talk more about that later.

Once this subroutine is over whatever happens, the next char flag is cleared so AnalyzeMSG does not get called again until the Bluetooth sends any more information.

5.2.5 PopulateTxtBuffer

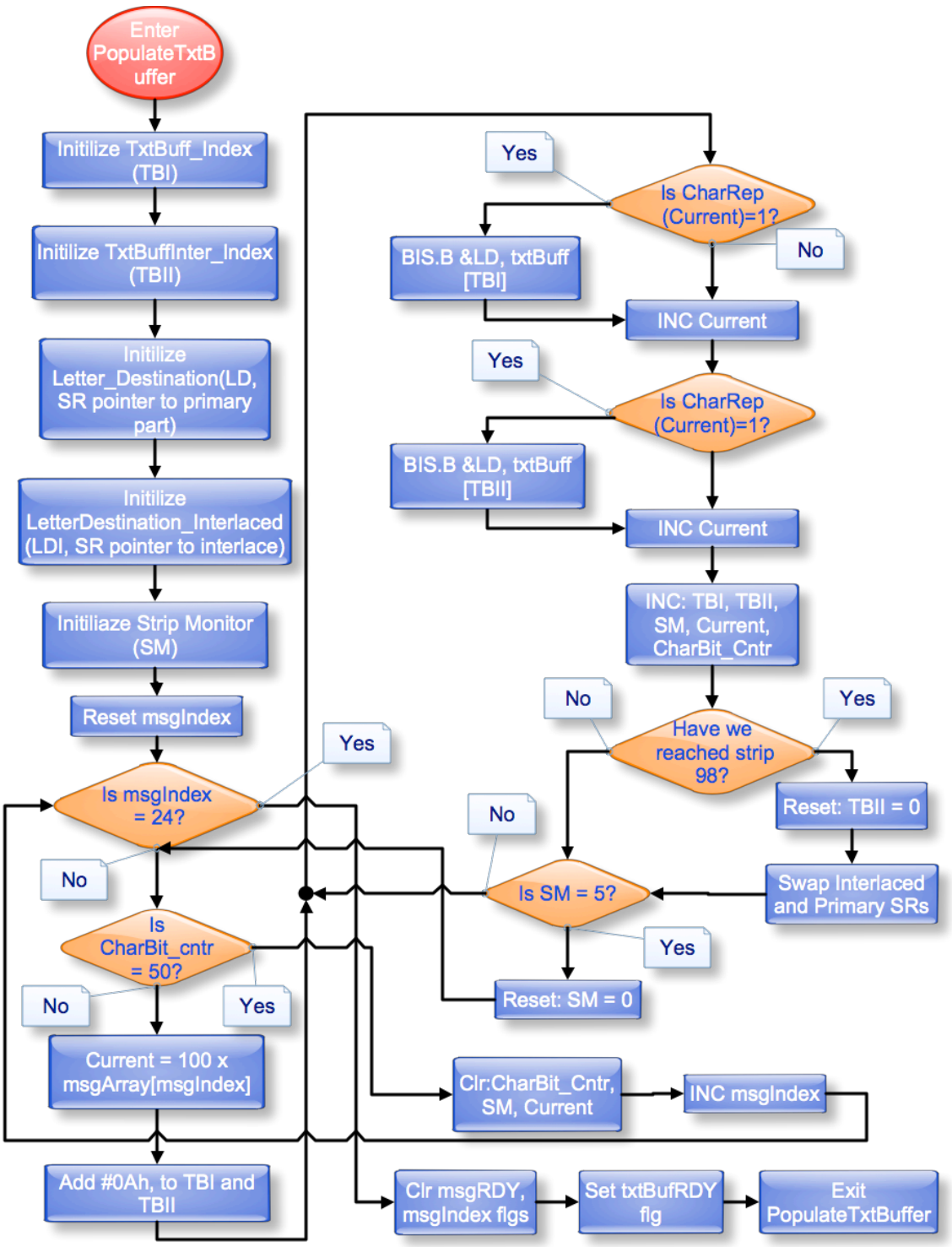


Figure 32: PopulateTxtBuffer flowchart

Once the user's input is analyzed *PopulateTxtBuffer* iterates over the message array created in the *AnalyzeMSG*. Every position in this array represents a character that can be accessed on SWORD's character look-up-table. Every character is located in msg[current] x 100. This subroutine's main job is to organize the user's input into the text buffer. In other words copy the corresponding character from the character LUT and paste them into the text buffer, however this is done taking into consideration two things: SWORD's display is interlaced and the message will be displayed on the top part of middle section of the rotating arc.

The interlaced works as follows: every letter is a 100-byte array. To make this explanation a little less abstract picture it as a 2-dimensional array (2D) of 10x10 like on figure 33.

```
 1. 0,0,0,1,1,1,0,0,0,0
 2. 0,0,1,0,0,0,1,0,0,0
 3. 0,1,0,0,0,0,0,1,0,0
 4. 1,0,0,0,0,0,0,0,1,0
 5. 1,0,0,0,0,0,0,0,1,0
 6. 1,1,1,1,1,1,1,1,1,0
 7. 1,0,0,0,0,0,0,0,1,0
 8. 1,0,0,0,0,0,0,0,1,0
 9. 1,0,0,0,0,0,0,0,1,0
10. 1,0,0,0,0,0,0,0,1,0
```

Figure 33: General concept of character 'A' lookup table representation

In order to achieve interlaced text every letter is divided in half, 5 rows are placed into the first half of the text buffer and other 5 into the second half. Since the interlacing occurs a half rotation later and there are 196 strips the second half of the letter is located 98 strips later; that is 98x16 bytes later.

Now that a general idea has been established we can go deeper into the interlacing part of this subroutine. The letters are indeed divided in two but they are not placed literary the first 5 rows on the first half and the second 5 on the other half of the text buffer. In SWORD's case interlacing means that the next available LED is located one LED down and 180º later and the one after that is another 180º more and so on. One thing that must be cleared is that every letter is display by shifting one column of the letter at a time, however to make this efficient and sequential in reality the letter is shifted like on figure 34.

```
 1. 0,0,0,1,1,1,1,1,1,1
 2. 0,0,1,0,0,1,0,0,0,0
 3. 0,1,0,0,0,1,0,0,0,0
 4. 1,0,0,0,0,1,0,0,0,0
 5. 1,0,0,0,0,1,0,0,0,0
 6. 1,0,0,0,0,1,0,0,0,0
 7. 0,1,0,0,0,1,0,0,0,0
 8. 0,0,1,0,0,1,0,0,0,0
 9. 0,0,0,1,1,1,1,1,1,1
10. 0,0,0,0,0,0,0,0,0,0
```

Figure 34: Actual character 'A' lookup table representation

The result is for every row the first byte is shifted to the *Letter Destination (LD)*, which is the next available spot in the text buffer, the second to the *Letter Destination Interlaced (LDI)* and the third one again to LD and so on. Once this done the LD ends up with the odd rows from figure 33 and LDI ends with the even rows. LD and LDI really represent the shift register where each bit is been placed. This is use once the buffer is created, during the buffer creation to place the bits on the correct locations two index must be used TextBufferIndex (TBI) and TextBufferInterlaced_Index (TBII), this is how the navigation through the text buffer is possible.

The second thing to take into consideration is that the letter will be displayed in the middle section of the rotating arc, SWORD's Ecuador. The Ecuador is distributed among the 5 top positions of the shift registers of the left side and right side of the arc. Additionally the shift registers are of 16 bytes so in order to only use the top 5 positions 11 zeroes must be shifted to each shift register prior to shift the data. As a result the text buffer is of size 16x196, 16 bytes wide and 196 strips high.

The last thing to remember is that when the middle of the text buffer is reached the strategy is completely the opposite, the character would be in the second half and the interlacing in the first half. Hence, LD and LDI must be swaped and TBII must be reset to 0. This is due to the fact that when you reach the middle of the text buffer the space for interlacing on the second half has been occupied however the space for writing is completely empty.

This is based on the encoding of images and text (which is explained in Section 4.3.3); each byte has information for five shift registers. Since we are only using one color for writing text each byte has all the information needed. Initially LD is set to 8 which sets shift register 3 out of the 5 and LDI is set to 2 which sets shift register 2. In other words when the middle of the text buffer is reached the bit equal to 8 is completely available on the second half and the bit equal to 2 is as well on the first half. This is why swapping the destination of the information works just fine.

In summary, this subroutine iterates over the msg array placing each character on the text buffer using the character look-up-table.

5.2.6 GUI Main

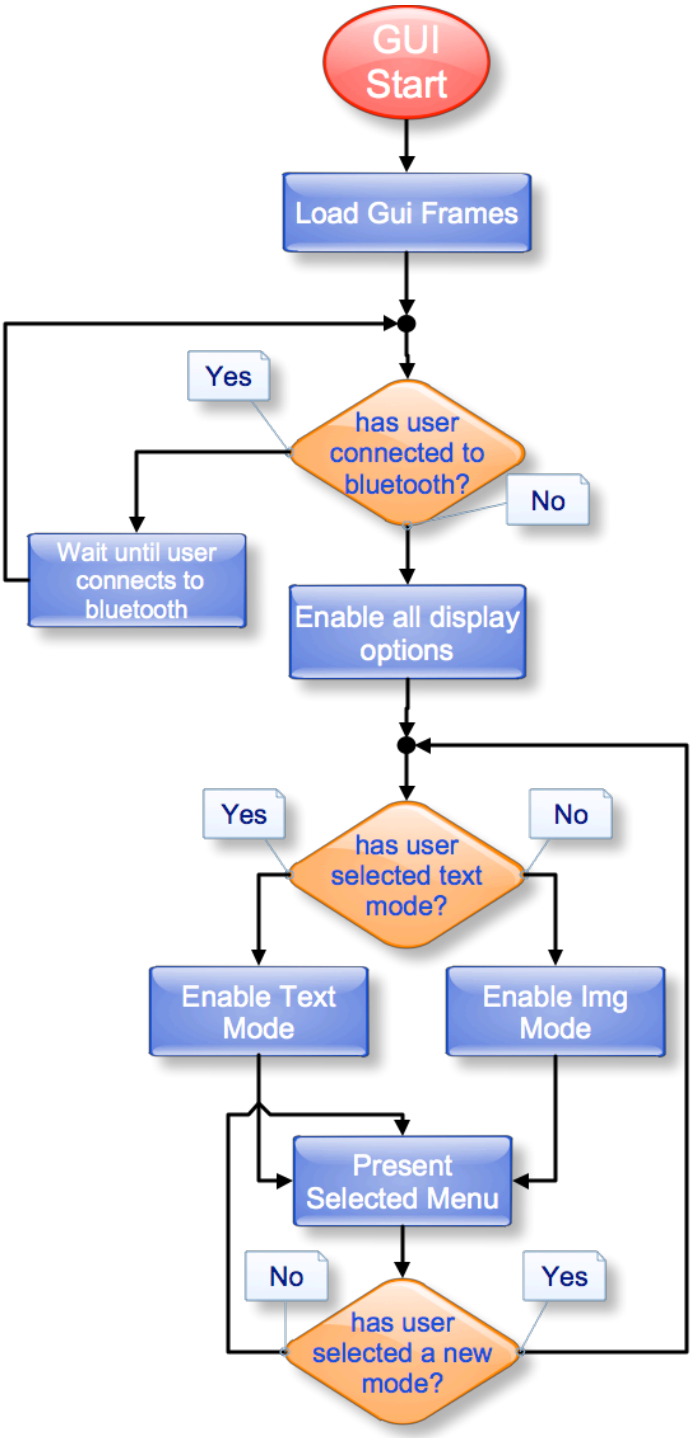


Figure 35: GUI Main flowchart

The GUI Main is responsible for displaying and acting upon all visual components presented by the User Interface. Upon loading all the GUI frames, the GUI

Main disables all interaction options and waits for the user to connect to SWORD via Bluetooth. It is important to point out that no interaction can occur with SWORD if the User Interface does not request a Serial Connection with the onboard Bluetooth Antenna. When the user connects, all interaction options (buttons, text fields) are enabled. The user then has the option of interacting with SWORD images, or SWORD text. If the user selects to interact with the User Text Mode, the GUI Main Code loads the corresponding windows, and delegates the responsibility of the user to the windows within User Text Mode. If the user suddenly switches and decided to interact with the User Image Mode, the same thing happens in this other direction; user responsibility is delegated to the windows within User Image Mode.

5.2.7 Image Mode

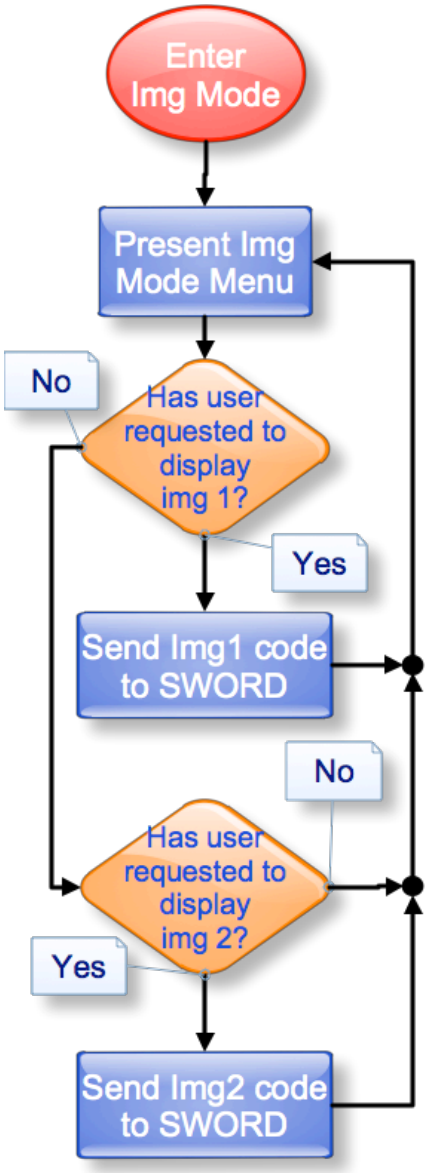


Figure 36: Image Mode flowchart

The Image Mode Menu is responsible for all interaction with SWORD Images via the User Interface. This mode is called from the GUI Main code, upon receiving input indicating that the user desires to deal with SWORD images. Since SWORD has only two onboard images, the Image Mode menu presents to the user two buttons: Display Image 1, and Display Image 2. If the user decides to display image 1, the Image Mode code sends to SWORD a special character that uniquely identifies image 1. The same process happens for image 2. While Image Mode is active, the User Interface is reactive to user inputs.

5.2.8 Text Mode

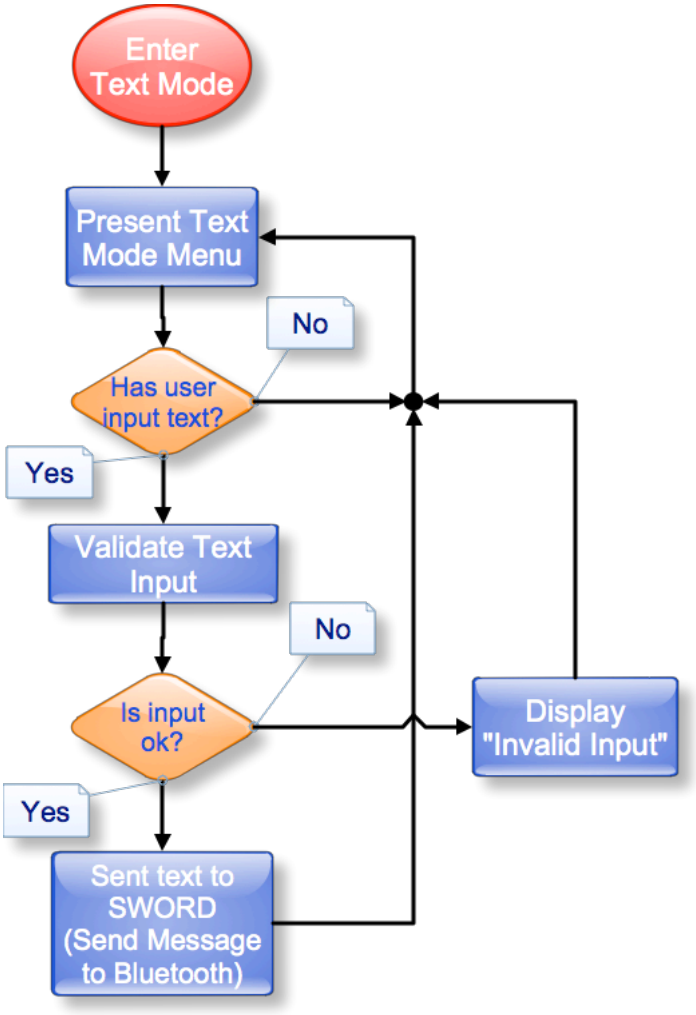


Figure 37: Text Mode flowchart

The Text Mode Menu is responsible for all interaction with SWORD Text via the User Interface. Similar to the Image Mode, while active, the Text Mode is reactive to user inputs. When the user inputs text in the provided text field (and sends it to SWORD) the User Interface determines whether the input text is valid or not. Valid text includes alphanumeric characters only. Anything outside that domain is rejected and the user is notified of the rejection. If the text is valid, the message is sent to SWORD via the function (Send message to Bluetooth).

5.2.9 Connect to Bluetooth

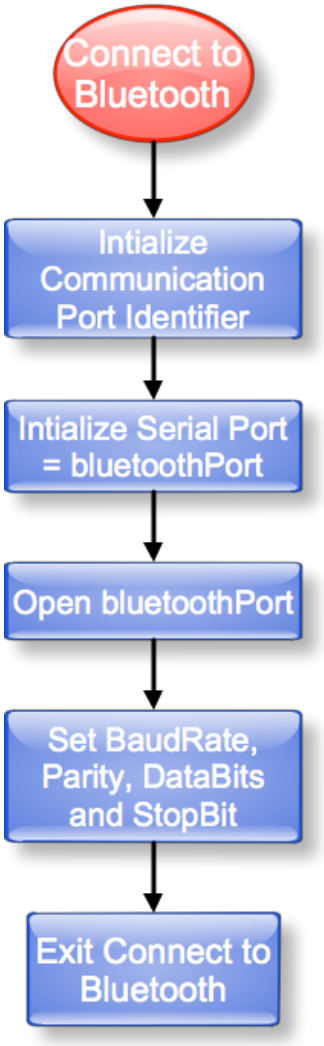


Figure 38: Connect to Bluetooth flowchart

The (Connect to Bluetooth) function is called from the User Interface, and is used to set up a Serial Communication channel with which to "talk" to external Bluetooth devices, namely the SWORD Bluetooth® Device. For the Bluetooth connection to follow through, it must first initialize the Port Identifier. As it name implies, the Port Identifier identifies possible serial ports with which it can communicate to external Bluetooth® devices. When it finds a port with which to communicate, it opens that port and sets the baud rate, the parity check (if any), the data bits, and stop bit. From that point on, the selected port will communicate under the specified configuration parameters. Upon setting the parameters, this function exits.

5.2.10 Send message to Bluetooth

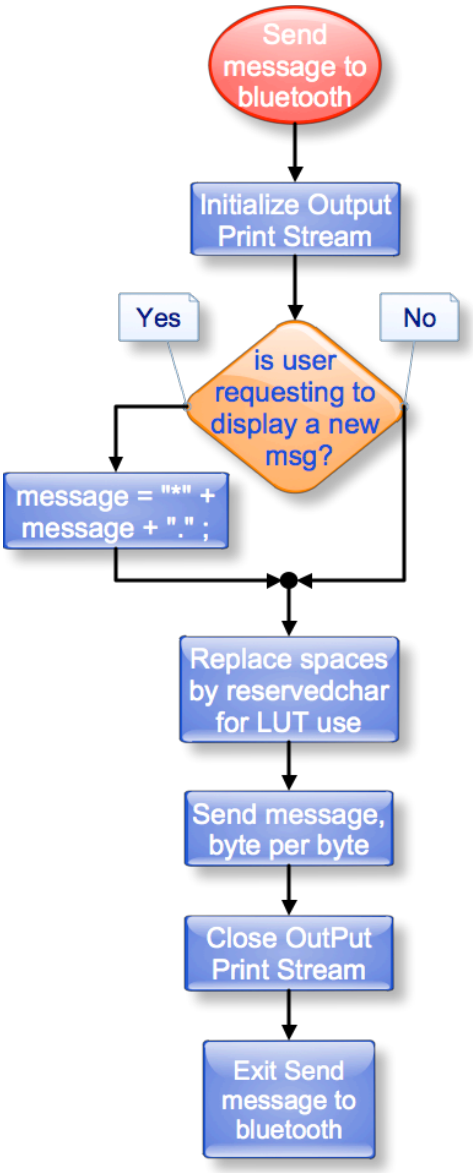


Figure 39: Send message to Bluetooth flowchart

The (Send Message to Bluetooth) function is called from the User Interface every time that the user inputs anything (be it an image selection, or text input) through the Graphical User Interface. When the GUI receives an input, it initializes the output stream that it will use to communicate the received input to SWORD itself. The User Interface then determines whether the received input is a request for a display image change or if the user is requesting to send a text message. If the former, it sends the special reserved character of the image that was selected to SWORD, and lets the Main Code interpret and handle the request. If the latter, it frames the input text with special

characters that aid the Main Code in interpreting the alphanumeric characters. Both data transfers to the Main Code occur byte per byte and upon termination, the User Interface closes the output stream it used, and the function exits.

5.3 System Efficiency

Several measures were taken to improve system software efficiency, and to make code easier to develop. They are classified into three categories: embedded software considerations, code considerations, and performance considerations. Embedded software considerations are situations in which our software takes advantage of the hardware it's running on by taking "shortcuts" that allow the desired functionality, with a smaller cost. Code considerations are situations in which our code followed good coding practices allowing for code reuse, and / or intelligent code tradeoffs. Performance considerations deal with situations in where code was developed with the intent of making the software run faster or more efficient.

5.3.1 Embedded Software Considerations

The first consideration taken into account was that the shift registers used to control the LEDs are organized by port. Therefore, if we want to write values to an entire color set of shift registers, a simple move instruction achieves this task. For example, if we want to shift in serially a 0 to the first shift register of the color Red, a 1 to the second shift register, a 0 to the third shift register, a 1 to the fourth shift register, and a 0 to the fifth shift register, the following line of code accomplishes it:

```
mov.b #10, P9OUT    ; P9 has the Red Shift Registers.
```

A second embedded software consideration deals with the fact that all control lines of the shift registers share the same pin. That is, all output enable lines share a pin, all latch enable lines share a pin, and all clock lines share a pin. This allows software to control the states of 15 individual shift registers through few instructions.

Another embedded software consideration deals with the number of LEDs that we use to display on SWORD. Our design uses 78 tri-color LEDs to present images. However, since we are driving the LEDs with the shift registers, it means that the last shift register has two pins, which are left unused. This might cause complexities in image handling since we originally stated that our design had a display resolution of 78 x 196. Therefore, to make the array that represents images compatible with the number of pins in the shift register, we made our image display array 80 x 196. This means that at some point we shift in values into the final shift register that aren't used, but by making sure that those two additional rows are full of zero's, this will not be a problem.

5.3.2 Code Considerations

A fundamental code consideration was a tradeoff we faced of memory vs. code complexity. Since the main component behind our program is the display of images, we needed a way to encode the image in software so that it could properly displayed. As

previously mentioned, our image array is 80 rows by 196 columns. However, since our LEDs are tri-color, the software array that encodes the image would also have to encode what color is being presented at an arbitrary position (x,y). In order to avoid having to write more code to interpret the color and assign the correct values to the LEDs, we decided to create three arrays per image: an array corresponding to the Red LEDs, an array corresponding to the Green LEDs, and an array corresponding to the Blue LEDs. These arrays are traversed at the same time, and this consideration makes it substantially easier to handle the color in the individual coordinates of the image.

Through the use of good coding practices, we ensured that the code that is in charge of displaying the letters on SWORD reuses the same code that displays the images.

As previously established, the time taken from step to step along SWORD's revolution is 463.8 microseconds. However, to make sure that our code can execute in that amount of time, we took an interesting approach to setting up the LEDs for display. The moment that the system displays a strip, the system begins loading data for the next strip in sequence. This allows the SWORD software to fully take advantage of the 485-microsecond time interval from strip to strip.

5.3.3 Performance Considerations

There considerations here deal with image representation and display.

The first consideration deals with images being stored internally as lookup tables. With this, iterating across the image arrays becomes very efficient, since no time is spent indexing the array to extract the correct value at an arbitrary position (x,y).

The second consideration actually complicates image representation, but the performance boost obtained is so great, that the tradeoff is worth it. Taking advantage of the fact that all shift registers share the same control lines and are organized by port, we have come up with a unique image-encoding scheme for SWORD. We will study the concept applied to one color and then extrapolate to include the other colors.

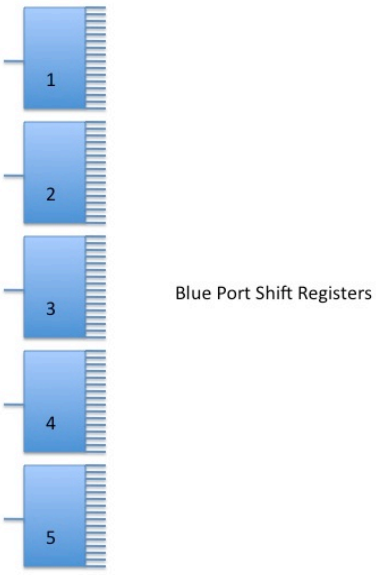


Figure 40: Image of 5 shift registers

Consider the case of one color's shift registers. This is the case seen in Figure 40. There are five, organized sequentially from Px.0 to Px.4. Each shift register has 16 parallel-out lines, which go directly to the LEDs, which total 80 parallel-out lines across the 5 shift registers. Since the control pins for the five shift registers are shared, if we were to shift in a single bit to one shift register, all other shift registers are affected. This might seem like a drawback, but it actually allows us to shift in 5 bits of information at a time. Assume that we are interested in having in the MSB of the 1st shift register the value 1, in the 2nd shift register the value 0, in the 3rd shift register the value 1, in the 4th shift register the value 0, and in the 5th shift register the value 1.

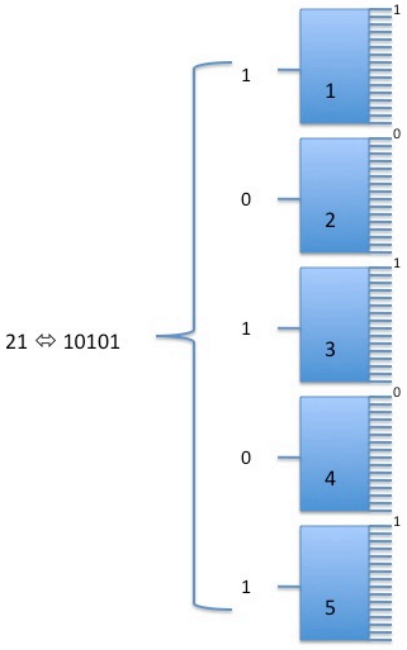


Figure 41: Represents the example configuration

Now, since the control lines are shared, we must make sure to have all the corresponding bits ready for shifting in at the serial data in pins of the 5 shift registers. Meaning that at the port Px, we must have the value 10101 = 21 (see Figure 41). Notice how one value encodes 5 states. Therefore, instead of encoding individual bits, we encode groups of 5 bits. This means that 16 values encode 80 states. By encoding the image strips in this manner, our display array can be reduced to 16 x 196. This causes a massive performance boost, since we have to iterate over less values, and are shifting 5 bits of data at a time.

5.4 Trustworthiness of Design

To ensure that SWORD's system software operated within the desired parameters, several boundary tests and crosschecks were implemented.

First, the written code executes fast enough for display. The displaying of LEDs is a crucial component to the success of SWORD. As previously mentioned, SWORD has 485 microseconds from the moment it displays one strip until the moment it displays the next. Through the use of the IAR Workbench debugger, we were able to determine how many clock cycles the code that takes care of the display lasts. After several rounds of testing, we managed to bring the display code down to approximate 540 clock cycles. At 12MHz operating frequency, this translates to 45 microseconds of duration. Since LEDs change state every 463.8 microseconds, the display code can comfortably execute the code with time to spare.

Second, the image must be displayed in a closed loop. Through the optocoupler circuit, our system can make sure that it will always complete the display array, and reset itself at the proper time. Every time that SWORD completes one revolution, the optocoupler handles image index resetting. This allows SWORD to always begin displaying from index zero every time that it passes through its initial position.

The third of the considerations relates to the accuracy of the system timers when dealing with the frequency of the interrupts used for display. Timer values were calculated through trial and error and with the use of a tachometer. This guarantees that the timers will act as close as possible to the required frequencies of operation.

The final considerations deal with the Java™ User Interface. First, the Java GUI does not allow you to quit the program **without** disconnecting the Bluetooth® first. This is because if the Bluetooth® is not disconnected, and the program quits, the serial port used to communicate with SWORD's Bluetooth® remains open and can no longer be used for communication. It is still a problem if the user decides to turn off the entire system and the Bluetooth® is still connected, but at least via software we can control this problem. The second User Interface consideration is that the GUI checks the input streams used to send messages to SWORD to make sure that the user does **not** send characters outside the accepted set. That is, that the input stream contains alphanumeric characters only.

5.5 Level of Software Completion

Similar to the chart seen for the Hardware Completion section of this document, we present the following chart. It outlines details of completed software tasks, considerations made, the team member responsible for that particular hardware task, and the level of termination in percentage.

Table 6: Level of Software Completion

Task	Considerations Taken	Teammate(s) Responsible	Level of Completion
MCU: Main code, ISR and Support Code	(None)	Ramón & Rogelio	100%
MCU: Onboard Slider Button, and Image Selection Handling	(None)	Rogelio	100%
MCU: Optocoupler Handling	1. Execution must be less than time taken from step to step in SWORD arc rotation	Ramón & Rogelio	100%
MCU: Image Display Code	1. Depends heavily on the rpm the DC motor can achieve 2. Requires fast executing code	Ramón & Rogelio	100%
MCU: Text Display Code	1. Reuses portions of the Image Display code	Ramón	100%
MCU: Bluetooth® Receiver Interface	1. LUT characters displayed aren't sequential in ASCII	Ramón	100%
MCU: Text Input Analyzer	1. Must determine when a message begins / ends 2. Must determine whether image / text was selected	Ramón	100%
UI: Bluetooth® Sender Interface	1. Uses special characters to allow for proper LUT use.	Ramón	100%
UI: Graphical User Interface	1. Must determine if text interrupts are correct 2. Makes sure the GUI does **not** exit if connected to Bluetooth®	Rogelio	100%

Figures 42, 43 and 44 visually demonstrate the level of software termination. The three photos indicate the image display capabilities that SWORD had at the end of prototype development.

Figure 42: Earth sphere image

Figure 43: Text image

Figure 44: Puertorrican flag image

5.6 User Guide

Welcome to SWORD!

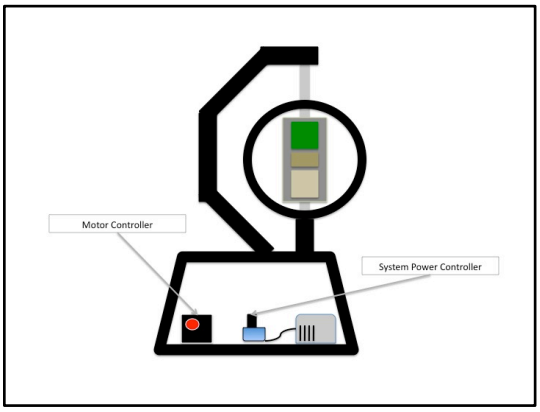
Thank you for purchasing your very own SWORD: the **S**pherically **W**ound **O**rb **R**otary **D**isplay. SWORD can display your predefined images in a three-dimensional spherical fashion. Also, across SWORD's user interface, you will be able to write custom messages, and SWORD will brightly display your messages to the world! We know that you will be very pleased with our product, and look forward to seeing you and your SWORD in the future!

Sincerely,

The Legendtronics Team

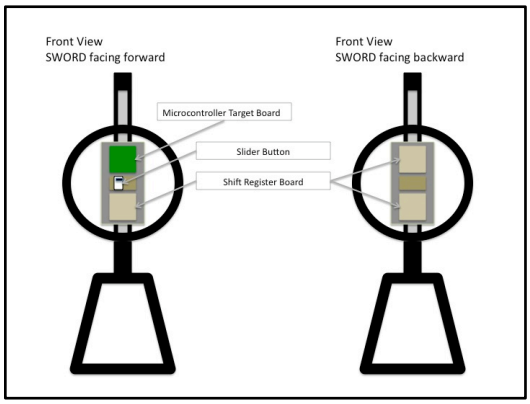
SWORD Inside Out

Please take the time to look at the following diagrams so that you may know the diverse components that SWORD has, to be able to take advantage of all it allows you to do.



Make sure to notice where the slider buttons are on SWORD, as they allow you to select between the two pre-coded images for display upon startup.

After plugging in SWORD to a wall socket (and making sure that the connection is properly grounded), you will use the two switches located underneath SWORD. The motor controller is used to enable to motor to start rotating. The system power controller powers SWORD itself; from the Microcontroller to the LEDs located on SWORD's arc.



Getting Started

To start enjoying your brand new SWORD, follow this set of instructions:

1. Make sure that you will be using Sword in a Safe environment.

2. Connect Sword to a 120V outlet.

3. Turn on the power supply for the system.

4. Select a pre-coded image with the onboard switch.

5. Turn on the motor. Right now you should be able to see the pre-coded image selected.

Congratulations! You can now see the pre-coded images!

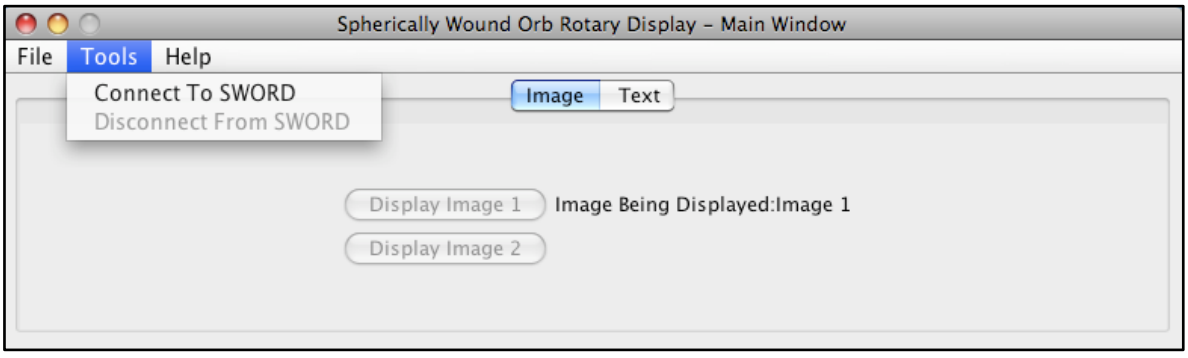
Advanced Controls

To explore other options for displaying things on SWORD, click on the SWORD_GUI.jar, and load the SWORD Graphical User Interface (GUI).

When you load the GUI, all options should be disabled.

User Guide Figure 1 – SWORD GUI Main Window

This is because in order to display text or change images in SWORD, you must first open the communication to it!



User Guide Figure 2 – Connect to SWORD!

Once connected, navigate through the available options, and proceed to either change the image being displayed, or writing text to SWORD for display. To change the text within the SWORD GUI, select the "Text" Tab on the northern part of the main window. You should arrive to the following screen:

User Guide Figure 3 – Writing to SWORD!

Click on "Send to SWORD" to send the text to display.

6 Conclusion

In this document we have outlined the design, development and capabilities of SWORD: the Spherically Wound Orb Rotary Display. Throughout the discussion, we have outlined several considerations (both Hardware and Software) that have allowed our prototype to run efficiently, withstand threshold operation, and circumvent any innate problems that it creates.

Through the development of this embedded system, our team had the opportunity to learn a wide variety of new concepts. Our team also applied a vast set of concepts previously learned in other projects and classes. Happily, our team can say that the SWORD prototype is capable of displaying patterns of images as well as any combination of text that does not exceeds its limits. With this, all project deliverables were accomplished, and all project stakeholders were satisfied.

However, we also recognize that for this prototype to serve use as a true marketing gimmick, the journey is far from over. Several considerations that were pointed out throughout the text were not handled because of the scope of this project. Nevertheless, after reviewing the level of completion of our system, we can conclude that its objectives were satisfied. The attractiveness of the images created by the final prototype is proof enough of the completion of our objectives.

7 Future Work

Many ideas have sprung since the finalization of the SWORD prototype and are enumerated below in order of the functionality that our team would most love to see:

1. Dynamic Image Representation

Dynamically uploading images to SWORD's software and displaying them on the fly would be a very impressive extension to the SWORD software. We have explored the feasibility of this extension and have found out that MATLAB® has native image processing software that would allow dynamic image (pattern) recognition. If we could integrate the MATLAB® functionality with a Java™ high-level interface, we are certain that we could dynamically display uploaded images.

2. Live streaming / Real time video

An extension of the previous item, instead of uploading static images, we would use something similar to what is utilized in YouTube™ to stream video across SWORD. At the moment, our team has not studied the feasibility of this possible future work, but needless to say that the idea is exciting!

3. Image Rotation

Image rotation was an extension of SWORD that was suggested by our project stakeholder. The idea is quite novel: controlling the image being displayed by rotating it left or right, and controlling its speed. Working with image timers, we are quite certain that this extension is feasible.

8 References

[100] Unauthored. (2002). "How many frames per second can the human eye see?" [Online] Available: http://www.100fps.com/how_many_frames_can_humans_see.htm

[C] - Coltheart M., "The persistences of vision," Philosophical trans. of the Royal Society of London. Series B, Biological Sciences, 290(1038), pp. 57–69, Jul. 1980.

[HAR] Hubel, D. (1995) "Eye, Brain, and Vision" [Online] Available: http://hubel.med.harvard.edu/bcontex.htm

[JNS] Sears, J. (2008) "The Orb" [Online] Available: http://www.jamesnsears.com/projects/orb/

[MAP] Furuti, C. (2004) "How Map Projections Worth" [Online] Available: http://www.progonos.com/furuti/MapProj/Dither/CartHow/cartHow.html

[MW] Mathworks (2009) "Inverse Map Projections" [Online] Available: http://www.mathworks.com/access/helpdesk_r13/help/toolbox/map/projec17.html

[OAD] - Oxford University Press. (2006). The New Oxford American Dictionary: "gimmick" [Online/Application]. Available: http://www.oxfordamericandictionary.com

[PF] Physics Forum (2004) "Human Eye" [Online] Available: http://www.physicsforums.com/archive/index.php/t-35727.html

[PLS] - Unauthored. (Accessed 2009, January 27) Persistence-of-vision LED Sphere [Online] Available: http://www.zigwire.com/projects/sphere/sphere.php

[S] - D. Sarwer. (2009). The Psychology of Appearance [Online]. Available: http://www.pennhealth.com/cha/psych.html

[SRD] - STMicroelectronics, STP16CPS05: Low voltage 16-bit constant current LED sink driver with auto power saving, Revision 5, February 2008.

[TIE] – Texas Instruments, MSP430F5438: Device Errata, Revision L, November 2008.

[WKP] Wikipedia (2009) "Persistence of Vision" [Online] Available: http://en.wikipedia.org/wiki/Persistence_of_vision

[YT1] Youtube (2007) "3D LED Display Globe" [Online] Available: http://www.youtube.com/watch?v=oLygWkHo9nw&feature=related

[YT2] Youtube (2007) "3D Orbs at Maker's Faire" [Online] Available: http://www.youtube.com/watch?v=4rEIFGvD4wA&feature=related

9 Appendix

9.1 Appendix A: Technical Documents

- Component Layout
- Parts List
- Schematics

9.2 Appendix B: Datasheets

- Bluetooth® Module
- LED's
- MCU: MSP4305438
- Motor Controller
- Optocoupler
- Power Supply
- Shift Registers
- Transistors
- Voltage Regulators

9.3 Appendix C: Code Listing

- MCU Source Code in Assembly
- Java Source Code for high level interface GUI

9.4 Appendix D: Project Management

- Updated Task List
- Project Journal