# Blue Light–Tracking Glasses and Electronic Surgical Tray

By

Erik Lundin [erikjl2]

Jane Zhao [janejz2]

David Yan [davidzy2]

Final Report for ECE 445, Senior Design, Spring 2020

TA: Charles Ross

Professor: Jing Jiang

May 2020

Team 3

## Abstract

The National Academy of Medicine has estimated that the American healthcare system wastes approximately $765 billion per year, $130 billion of which is due to inefficiently delivered services that include surgery. The main area of waste in surgical operations is the unnecessary disposal, re-sterilization, or loss of surgical tools. Clearly, a method must be devised to keep track of which surgical tools have been used during surgeries. This could help to prevent unused surgical tools from being re-sterilized or being thrown away, both of which are costly outcomes. Our Electronic Surgical Tray would be able to solve this problem because it has weight sensors that would be able to detect when surgical tools are removed from or placed back onto the tray, and a simple LED indicator would signify the whether the tool has been used or not. In contrast, the currently available solutions for tracking surgical tools such as barcodes, RFID tags, and optically detectable polymers all require that the tools be scanned before and after surgery to make sure that no tools have been lost, so they practically exclude the possibility of monitoring surgical tool inventory during surgery due to the inconvenience of manual scanning [1], [2], [3], [4]. Our device, however, would be able to allow nurses and surgeons to easily determine if tools are missing during surgery by looking at the colors of the LED indicators on the surgical tray. This document will go into further detail regarding our solution to this problem.

## Contents

# 1. Second Project Motivation

## 1.1 Problem Statement

Surgical tools are expensive and can usually be sterilized for reuse; however, unused tools are often unnecessarily sterilized and are sometimes even accidentally thrown away or lost, leading to significantly increased costs for hospitals. Currently, the existing solutions aim to prevent unnecessary sterilization of surgical tools and waste of surgical supplies by marking each instrument and tracking them electronically: these solutions include RFID tags, optically detectable polymer coatings, and barcode labels. However, there are still no devices available that help surgeons and nurses conveniently keep track of the use-status (used vs. unused) of surgical tools in the operating theatre during surgery.

## 1.2 Solution

To solve this problem, we propose to develop an Electronic Surgical Tray that can hold surgical tools and sense when they are removed from the tray. The tray will have designated spaces (with weight sensors and LED indicators) for each surgical tool to be placed onto. The device's overall function is to allow surgeons and nurses to keep track of which items have been used (red spaces) or unused (green spaces). After the tray has been prepared with surgical tools, a calibration button will be pressed, causing the Electronic Surgical Tray to register all the weights of the tools on the tray. This will generate an electronically stored array of tool weights. After calibration, every space on the tray will light up green, indicating that the tools are unused. When a tool is removed, the tray space will light up yellow. If multiple tools are removed, multiple spots light up yellow. Used items, when placed back onto the tray, will cause the tray to turn red in that space. Importantly, used items do not need to be returned to the same space, as the microprocessor should be able to register their weight and match them to one of the calibrated weights to determine that the tool is one that was previously removed from the tray. Additionally, if a mistake is made lifting up a tool, in order to reset the space in the tray, the nurse can press (and hold down) the reset button while the tool is on the tray. The tray will be easily sterilized so that it can be used for multiple surgeries. Current electronic inventory systems such as barcodes or polymer coating should be used in combination with this tray to help create an overall inventory of the surgical tools on the tray for each surgery, thus allowing the nurses to know exactly which tool was used/unused or missing during the surgery.

## 1.3 High Level Requirements

1. Thirty tray spaces can be calibrated to hold 30 surgical tools of any identity that individually weigh between 10 g to 1000 g with an accuracy of ±4 g. Red indicator LEDs in the individual spaces on the tray will light up when a tool has been placed back down onto the tray after being used.
2. Device can be sterilized in the autoclave at 121°C for at least 30 minutes at 15 psi.
3. Electronic tray has enough power to operate for 14 hours without using external power supplies.

## 1.4 Visual Aid

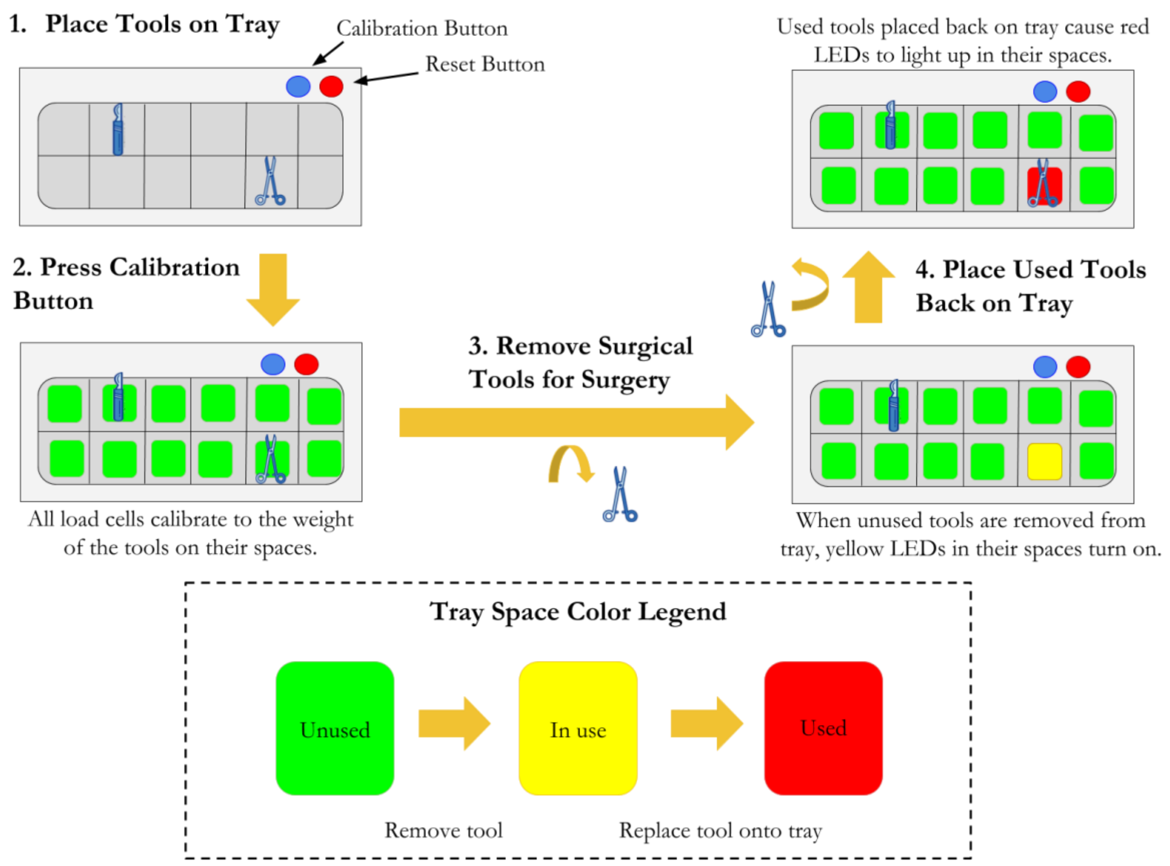
1. Place Tools on Tray
Calibration Button
Reset Button
Used tools placed back on tray cause red LEDs to light up in their spaces.
2. Press Calibration Button
4. Place Used Tools Back on Tray
3. Remove Surgical Tools for Surgery
All load cells calibrate to the weight of the tools on their spaces.
When unused tools are removed from tray, yellow LEDs in their spaces turn on.
Tray Space Color Legend
Unused
In use
Used
Remove tool
Replace tool onto tray

**Figure 1. Visual Aid for Electronic Surgical Tray.**

## 1.5 Block Diagram

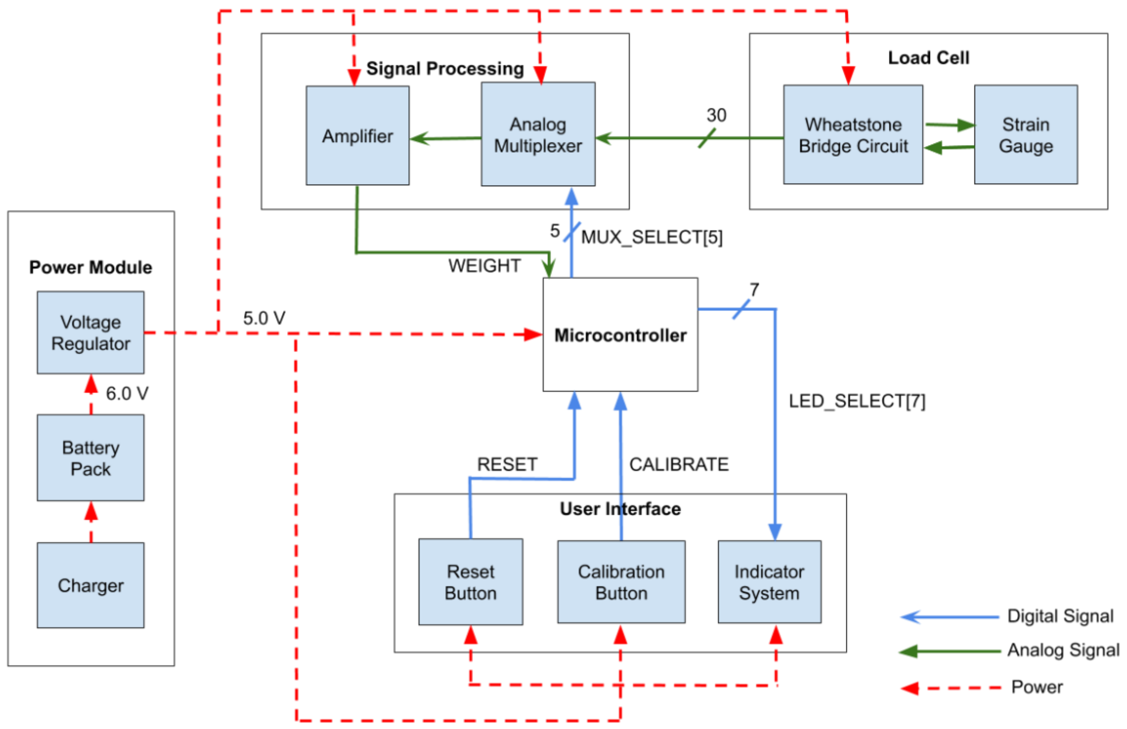

**Figure 2. Block Diagram of Electronic Surgical Tray.**

# 2. Implementation

## 2.1 Physical Design

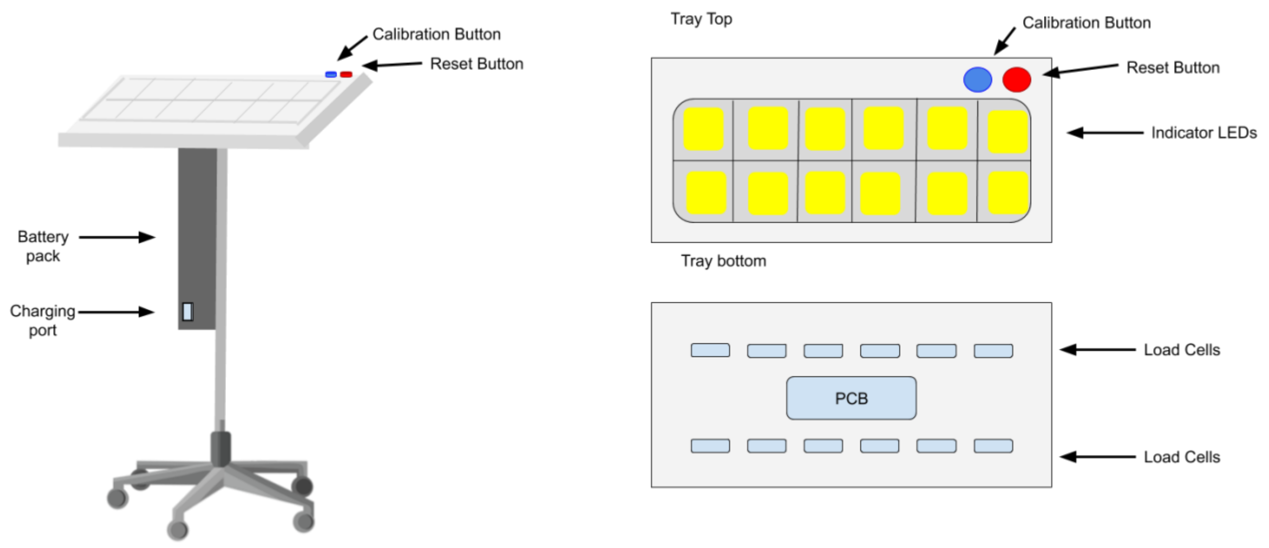

**Figure 3. Physical Design. Not all slots and LED colors are pictured.**

The physical design of the Electronic Surgical Tray is pictured in Figure 3. The device is composed of two main parts, the tray and the stand.

The tray can separate from the stand and the battery pack for easier sterilization. The tray is composed of most of the components of the device including the user interface, load cells, and microcontroller. In each of the tray spaces there are LED indicators that indicate the tool use-status. There are 3 different indicator colors. Yellow indicates that the tool is in-use, green indicates unused, red indicates used. These statuses can be controlled using the reset and calibration buttons. On the bottom of the tray the PCB and load cells are fixed into place.

The stand is used to wheel the device into the operating room. Attached to the stand is the battery pack which houses the rechargeable lithium-ion batteries used to power the device. The batteries can be charged through a charging port on the side of the battery pack as pictured.

In order to ensure that our 2nd high-level requirement is met, that the tray portion of the design can withstand the heat and pressure of the sterilization process, the tray will be encased in a layer of epoxy or resin.

## 2.2 Software

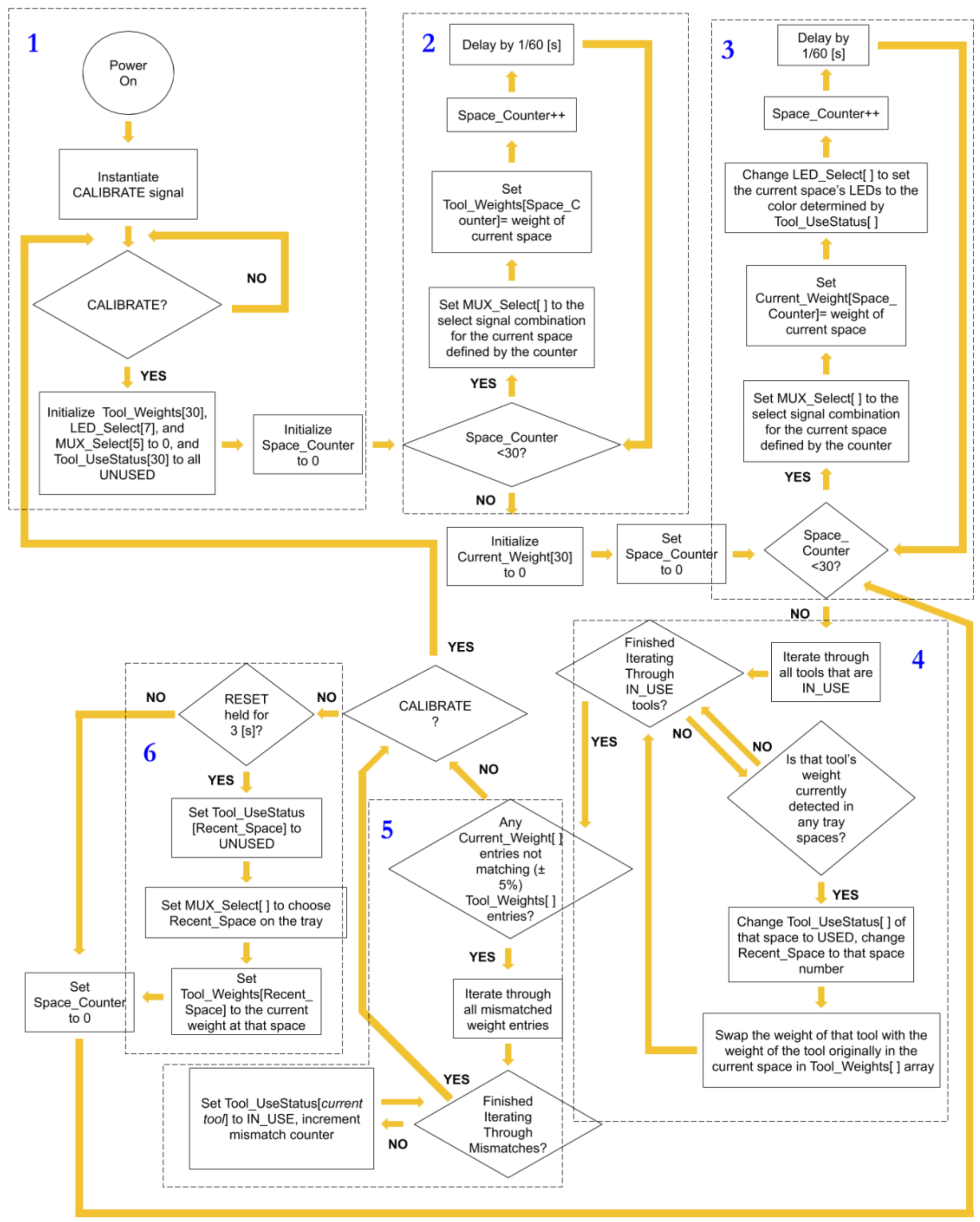
1
Power On
Instantiate CALIBRATE signal
CALIBRATE?
NO
YES
Initialize Tool_Weights[30], LED_Select[7], and MUX_Select[5] to 0, and Tool_UseStatus[30] to all UNUSED
Initialize Space_Counter to 0
2
Delay by 1/60 [s]
Space_Counter++
Set Tool_Weights[Space_Counter]= weight of current space
Set MUX_Select[ ] to the select signal combination for the current space defined by the counter
YES
Space_Counter <30?
NO
Initialize Current_Weight[30] to 0
Set Space_Counter to 0
3
Delay by 1/60 [s]
Space_Counter++
Change LED_Select[ ] to set the current space's LEDs to the color determined by Tool_UseStatus[ ]
Set Current_Weight[Space_Counter]= weight of current space
Set MUX_Select[ ] to the select signal combination for the current space defined by the counter
YES
Space_ Counter <30?
NO
4
Iterate through all tools that are IN_USE
Finished Iterating Through IN_USE tools?
YES
NO
NO
Is that tool's weight currently detected in any tray spaces?
YES
Change Tool_UseStatus[ ] of that space to USED, change Recent_Space to that space number
Swap the weight of that tool with the weight of the tool originally in the current space in Tool_Weights[ ] array
YES
CALIBRATE ?
NO
NO
NO
RESET held for 3 [s]?
6
YES
Set Tool_UseStatus [Recent_Space] to UNUSED
Set MUX_Select[ ] to choose Recent_Space on the tray
Set Tool_Weights[Recent_Space] to the current weight at that space
Set Space_Counter to 0
5
Any Current_Weight[ ] entries not matching (± 5%) Tool_Weights[ ] entries?
YES
Iterate through all mismatched weight entries
YES
Finished Iterating Through Mismatches?
NO
Set Tool_UseStatus[*current tool*] to IN_USE, increment mismatch counter

**Figure 4. Microcontroller Algorithm.**

**Table 1. Description of Variables for Algorithm in Figure 4.**

| Variable Name | Description |
|---|---|
| Tool_UseStatus[30] | An array of strings that represent the use-statuses of each tool on the tray. Possible values are "UNUSED", "IN_USE", and "USED". These values are decoded to the LED colors, which are green, yellow, and red, respectively. |
| Tool_Weights[30] | An array of integers representing the weights of all of the tools that were initially placed onto the tray. |
| LED_Select[7] | An array of select signals (binary) that chooses the LED color for a specific space on the tray. LED_Select[6:2] are used to choose a space, while LED_Select[1:0] are used to select the color of the LED that turns on in that space. |
| MUX_Select[5] | An array of select signals (binary) that chooses the weight (the output of the Load cell) from a specific space on the tray. |
| Current_Weight[30] | An array of integers representing the weight of each tool currently incident on the tray. If no tool is present, the weight will be zero. |
| Recent_Space | An integer variable that holds the number of the tray space that had a tool lifted from it most recently. This variable only holds values of spaces that have tools "IN_USE". |
| Space_Counter | Counts the spaces in the tray starting from 0 to 29, or 30 total trays. |

## Description of C Program Implementation in Appendix B:

The C program found in Appendix B is based off the algorithm in Figure 4. The code was written in the MPLAB IDE for 8-bit PIC microcontrollers made by Microchip[1]. Our C program consists of one main.c file that has multiple functions within it, including one main() function and 6 helper functions (the main function is the last one). The purpose of the main function is to execute the proposed algorithm indefinitely, for the entire time that the device is on; while the purpose of the helper functions is to assist in completing tasks that are commonly used in the algorithm: all functions are described in Table 2.

**Table 2. Descriptions of Functions in C Program in Appendix B.**

| Function Name | Purpose |
|---|---|
| **int main(void)** | Computes the algorithm in Figure 4. |

[1] https://www.microchip.com/mplab/mplab-code-configurator

| | |
|---|---|
| **void ConvertBinaryToArray(unsigned int val, unsigned int *array, int start_index, int length){** | Takes a decimal value and converts its binary form into an array with single bit elements. Modifies an input array passed to it by a pointer. |
| **void setMuxSelect(unsigned int MUX_Select_Array[])** | Sets digital outputs RC5-RC4 and RC2-RC0 to the bits held in MUX_Select[5], which chooses the space in the Analog Multiplexer subsystem. |
| **void setLEDSelect(unsigned int *LED_Select_Array)** | Sets digital outputs RB6-RB0 to bits [6:0] of LED_Select[7], which chooses the space and the LED color. |
| **void setLEDs(char UseStatus[], unsigned int *LED_Select_Array){** | Converts the use-status color code for all spaces into select bits [1:0] of LED_Select[] and then uses setLEDSelect() to set the output pins to the bits of LED_Select[]. |
| **int detectNumber_noindex(int value, unsigned int *array)** | Determines if a tool weight value is present within ±5% in the array referenced by a pointer. |
| **int detectNumber(int value, unsigned int *array, int *swapped_space){** | Same as detectNumber_noindex except that the swapped_space variable is set to the index where a number is detected if there the number is present in the array. |

In order to correctly assign the I/O pins in the code, we used the datasheet[2] and flash programming guide[3] for the PIC microcontroller, as well as online resources[4] explaining how to use I/O buffers. The code to program the A/D conversion ports and the I/O statuses of analog and digital ports can be found in the main() function under the sections ADC Setup and Tri-State Bits. The registers controlling the A/D conversion are ADCON0, ADCON1, and ADCON2 (pp. 21-23 of datasheet[2]). Bit-wise operations were used to change these registers so that only the AN0 pin was chosen for the ADC, the ADC was enabled, and only AN0 was analog while the other pins were digital. The TRIS buffers (Tri-State buffers) were set for each port (A, B, C) to set RA0 to be an analog input AN0, to set RB0-RB7 to digital outputs, to set RC0-RC5 to digital outputs, and to set RC6-RC7 to digital inputs. The analog input pin was assigned to the output of the amplifier (holding the weight signal), while the digital output pins were then assigned to bits LED_Select[6:0] and MUX_Select[4:0], and the digital inputs were assigned to the CALIBRATE and RESET signals. Specific bit assignments can be found in the code; furthermore, the pseudocode is sufficiently descriptive to be able to follow the entire algorithm pictured in Figure 4. Different areas in Figure 4 are labeled in the C program in Appendix B in blue comments. After finishing the code, we compiled it inside of the MPLAB IDE and found that it compiled properly and all input/output pins and bit access phrases were recognized.

[2] http://ww1.microchip.com/downloads/en/DeviceDoc/39887c.pdf
[3] http://ww1.microchip.com/downloads/en/DeviceDoc/30009622M.pdf
[4] http://picguides.com/beginner/digital.php
https://microcontrollerslab.com/use-input-output-ports-pic18f452/
http://www.add.ece.ufl.edu/4924/docs/adc/PIC_ADC_Sample_Code.pdf
http://www.dmi.unict.it/~santoro/teaching/lap1/slides_pic/DigitalMCU.pdf

Briefly, the “AREA 1” code waits for the CALIBRATION signal before entering the algorithm. The “AREA 2” code calibrates the electronic surgical tray to the tools that are currently on the tray when the calibration button is pressed; specifically, it creates an array of tool weights. Then, in “AREA 3,” the program checks the current weights in each space and stores those values in another array, then updates the use-statuses of the tools. In “AREA 4,” tools that are returned to the table are changed to USED status. In “AREA 5,” tools that are removed from the tray are changed to IN_USE. In “AREA 6,” the space for the most recently removed tool is reset if the reset button is pressed. After this, the program loops back to “AREA 3” unless the calibration button is pressed, in which case it will enter “AREA 2.” One final thing to note about the C program implementation is that a CALIBRATE_flag signal was used to prevent the code from going into an infinite loop inside the initial conditional calibration in the “AREA 1” code.

## 2.3 Subsystem Details

### 2.3.1 Microcontroller

The microcontroller chosen for our project is the PIC18F2553-I/SP from Microchip Technology. This module controls the circuit and tracks the tool use-status as required by our 1st high-level requirement. This module performs calibration by taking measurements from the Signal Processing Unit. It receives only one weight signal from the Signal Processing module, but controls which signal by using a 5-bit MUX_Select signal. This signal is used as the select signal for the multiplexers. Once the Calibration button is pressed, the microcontroller uses an algorithm as described above, to check if any of the tools have been removed or placed back. Using this information, the Microcontroller module sends a 7-bit LED_Select signal to the Indicator System to display the use-status in each tray space. This module takes as inputs the outputs of the Reset Button, the Calibration Button and the analog WEIGHT signal of the Signal Processing module. Its outputs are the 5-bit MUX_Select signal and the 7-bit LED_Select signal.

### 2.3.2 Power Module

This module consists of the components that deliver power to the system. The three subsystems that make up the power module are the battery pack, voltage regulator, and charger. The battery pack is composed of 6.0 V Lithium-Ion batteries that power the system. This battery pack is required to satisfy our first high-level requirement, that the system has enough power to operate for 14 hours without external power supplies. The voltage regulator chosen for our system is the TPS72501DCQR from Texas Instruments. This voltage regulator was chosen so that for an input of 6.0 V, an output of 5.0 V is produced per our design. A voltage regulator is required to ensure that the power delivered to the system is consistent and that the system is able to function reliably in the operating room. The charger is an off-the-shelf charger that was chosen in order to ensure that the battery-charging process is safe.

### 2.3.3 Load Cell

The actual sensing of the surgical tool weights will be achieved using a load cell which translates the downward force of gravity from the tools into an output voltage which can be amplified and read by the microcontroller. This module consists of several strain gauges configured in a Wheatstone Bridge circuit. When these gauges are deformed their resistance changes which can be used to infer the force applied to them if they are biased properly. Multiple strain gauge sensors can be used at different orientations to improve the overall accuracy and compensate for errors that can arise from temperature variations [5]. The strain gauges in the Strain Gauge subsystem are part of the Wheatstone Bridge Circuit, hence the bidirectional analog signals shown in Figure 2 between the Strain Gauge subsystem and the Wheatstone Bridge Circuit. The sensitivity of the strain gauge can be discerned from its gauge factor which relates the deformation of the strain gauge to its change in resistance. For this sensor a gauge factor between 2.0-2.2 is needed to achieve weight measurements within the accuracy defined in the high-level requirements. The BF350-3AA from ICStation will be used in this design since it has nominal resistance of 350 Ω and a gauge factor within our desired range.

These strain gauges as stated before will be configured in a Wheatstone Bridge circuit which consists of two parallel voltage dividers as shown in Figure 5. Each element in the voltage divider will be a strain gauge. This allows the resistance change of the strain gauges to be translated into a voltage difference between the two dividers which can then be further processed and finally sampled in order to get an accurate weight reading. For this application a Full-Bridge Type 3 Wheatstone bridge configuration is used as shown in Figure 6. This type of configuration provides the highest sensitivity of the configurations for measuring axial strain and compensates for variations caused by temperatures [5] which will allow the device to read the weight ranges to the accuracy specified in the high-level requirements. Since this circuit utilizes voltage dividers an external voltage is required. This voltage will be around 5V since the other systems in this design also require that supply voltage. The output of the Wheatstone Bridge Circuit is provided to the Analog Multiplexer subsystem, which will provide one of the signals from the 30 load cells to the Amplifier subsystem to be sampled by the Microcontroller module.

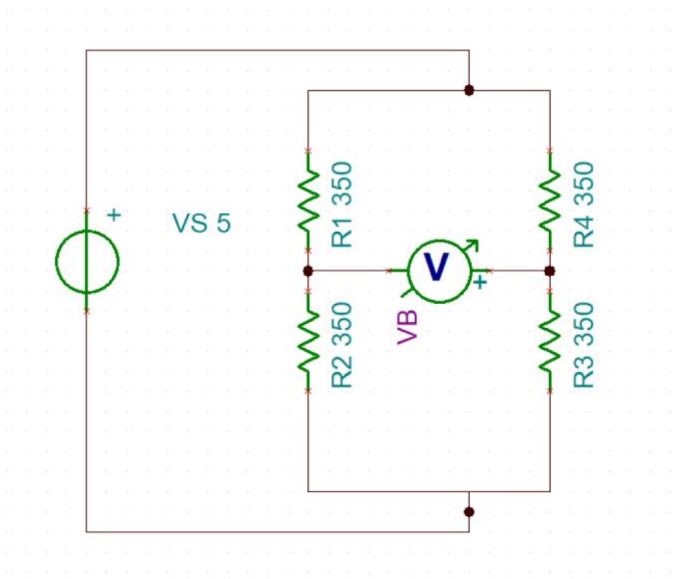

**Figure 5. Wheatstone Bridge Circuit Configuration**

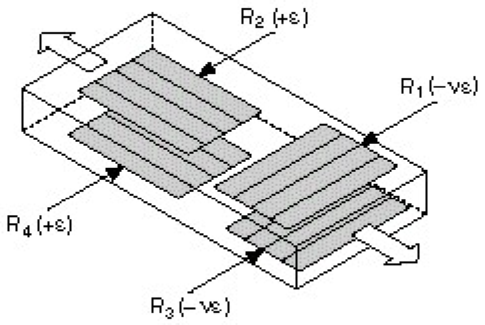

**Figure 6. Full-Bridge Type 3 Configuration [5]**

### 2.3.4 Signal Processing

Since the strain gauge's resistance will only change on the order of a few tenths of an ohm the output of the Wheatstone Bridge will be on the order of a few millivolts so amplification is necessary in order to generate a voltage range that will allow the microcontroller to make accurate measurements. However, to avoid using a large number of amplifiers a series of analog multiplexers will be used to select from each of the 30 load cell outputs. In order to achieve this two 16:1 multiplexers will be cascaded with a 2:1 multiplexer in order to read from the 30 load cells. For this the Texas Instruments CD74HC4067M96 and SN74LVC1G3157 will be used respectively. This will require that five digital outputs of the microcontroller be used to select each of the load cells. It will also require a higher sampling rate which should not be a problem since this application does not require fast data acquisition.

Once the correct load cell is selected the signal can then be amplified. Since the output of the Wheatstone bridge has an input impedance that can change, this must be considered when designing an amplifier or an amplifier configuration that does not depend on the source impedance needs to be used. This problem will be solved by using a three op-amp instrumentation amplifier [6]. The configuration of this amplifier can be seen in Figure 7 and has a high input impedance. In this configuration R1=R2 and R3=R4. With R1=10 kΩ, R4=50 kΩ and RF=8 kΩ a Rg value of 410 Ω will yield a voltage gain near 200. The INA827 instrumentation amplifier from Texas Instruments will be used. This package this already includes R1-R4 and Rf so only Rg needs to be connected to set the gain. This configuration is shown in Figure 8. The DC voltage transfer characteristics of this amplifier are shown in Figure 9.

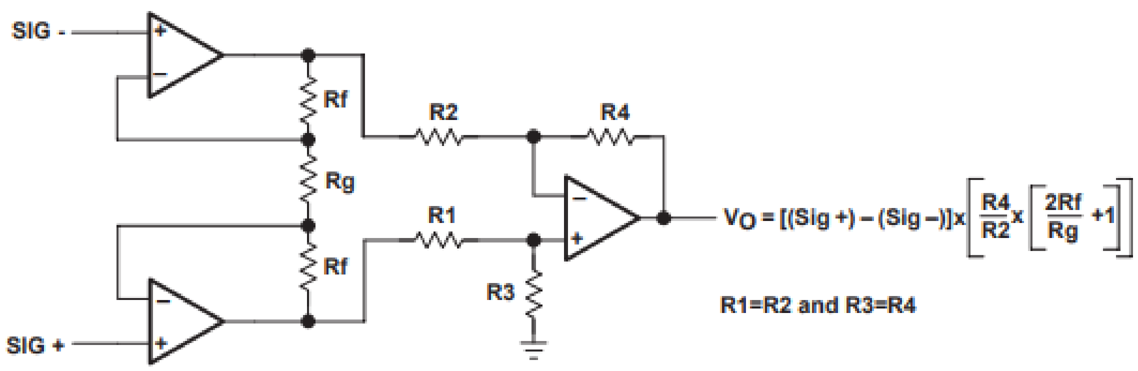
SIG -
SIG +
Rf
Rg
Rf
R2
R4
R1
R3
VO = [(Sig +) – (Sig –)] x $\frac{R4}{R2}$ x [$\frac{2Rf}{Rg}$ +1]
R1=R2 and R3=R4

Figure 7. Three Op-Amp Instrumentation Amplifier [6].

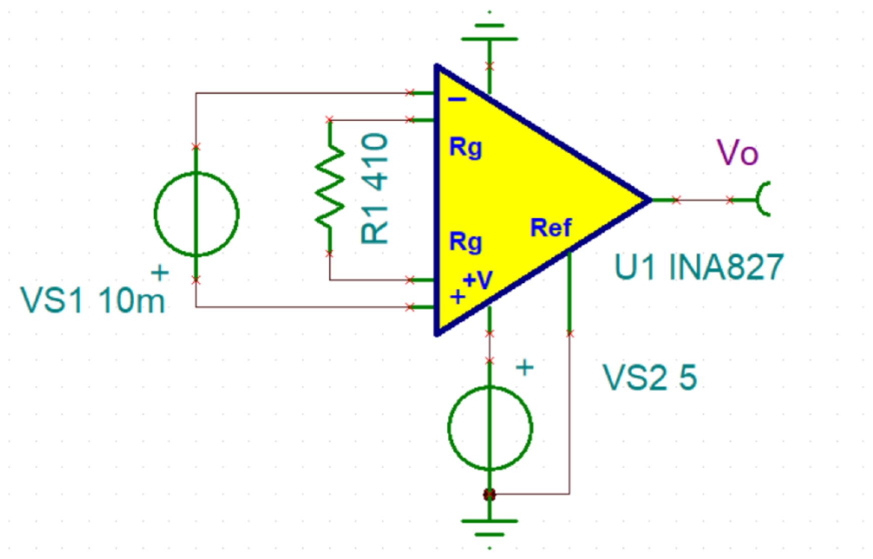

Figure 8. INA827 Configuration.

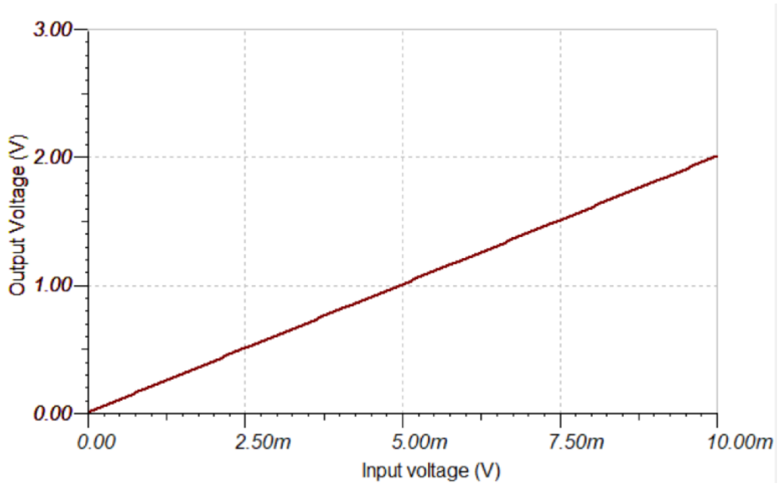

Figure 9. Instrumentation Amplifier DC Transfer Characteristics at 5V Bias.

The main concern for the final performance of the device is how the noise introduced at the various stages of sensor reading will affect the final accuracy of the results. A basic noise analysis of the Wheatstone Bridge and the amplifier follows. However, it is not clear how much noise will be introduced by the analog multiplexer which requires further analysis in the lab.

First the thermal noise from the load cell resistors is considered. This noise can be modeled by equation 1. Where k is the Boltzmann constant, T is the operating temperature, Δf is the bandwidth and R is the resistance value.

$$v_n = \sqrt{4kT\Delta fR}$$

(Equation 1)

For the purpose of this analysis a temperature of 300°K is used over a bandwidth of 10 kHz. The resistance will be the series combination of two strain gauges which totals to 700 Ω. This results in a resistor thermal noise of 340 nV. Next the amplifier noise is considered. The INA827 has an input voltage noise of 17 nV/sqrt(Hz) and an output voltage noise of 250 nV/sqrt(Hz). Equation 2 can be used to determine the total noise. Where eNI is the input voltage noise, eNO is the output voltage noise and G is the gain.

$$v_n = \sqrt{e_{NI}^2 + (\tfrac{e_{NO}}{G})^2}$$

(Equation 2)[5]

From this the amplifier noise is found to be about 1.6 µV for a gain of 200 and a bandwidth of 10 kHz. For the total noise the amplified resistor noise of 68 µV must also be included. This yields a noise voltage of about 70 µV. This result is below 1.2 mV of the ADC of the microcontroller so will not affect the accuracy of the weight measurement. However, other more complicated noise sources and the noise from the analog multiplexer was not considered in this analysis. Further analysis in the lab is needed for a more thorough result.

### 2.3.5 User Interface

The user interface allows the user to read and alter use-statuses. It consists of the calibration and reset buttons, as well as the indicator system. The calibration and reset buttons are buttons that when pressed allows the user to reset the statuses of all tools or one single respectively. The indicator system is an array of red, green, and yellow LEDs per space on the tray. These LEDs are controlled using the LED_SELECT signals from the microcontroller. When a tool is used the LED should turn red, when the tool is unused the LED should be green, when the tool is in use the LED should turn yellow. Only one LED should be on per space at any given moment. The indicator system satisfies part of the first high-level requirement by allowing users to easily identify whether a tool has been used or not.

[5] http://www.ti.com/lit/ds/symlink/ina827.pdf?&ts=1588963951570

## 2.4 Costs

### 2.4.1 Bill of Materials (BOM)

**Table 3. Parts Costs**

| Part | Part # | Manufacturer | Quantity | Unit Price ($) | Total Price ($) |
|---|---|---|---|---|---|
| Voltage Regulator | TPS72501DCQR | Texas Instruments | 1 | $3.45 | $3.45 |
| Battery | PRT-11856 | SparkFun Electronics | 1 | $15.95 | $15.95 |
| Charger | PRT-10473 | SparkFun Electronics | 1 | $33.75 | $33.75 |
| LED (Red) | GR PSLR31.13-GTHP-R1R2-1 | OSRAM Opto Semiconductors Inc | 30 | $0.74 | $22.20 |
| LED (Green) | MLEGRN-A1-0000-000X01 | Cree Inc. | 30 | $0.81 | $24.30 |
| LED (Yellow) | LCY G6SP-CBDB-5E-1-140-R18-Z | OSRAM Opto Semiconductors Inc | 30 | $1.57 | $47.10 |
| 2:1 Analog Multiplexer | SN74LVC1G3157 | Texas Instruments | 1 | $0.43 | $0.43 |
| 16:1 Analog Multiplexer | CD74HC4067M96 | Texas Instruments | 2 | $0.75 | $1.50 |
| Decoder (4:2) | CD74AC139M96 | Texas Instruments | 2 | $0.64 | $1.28 |
| Decoder (8:3) | SN74LS138N | Texas Instruments | 4 | $0.84 | $3.36 |
| Microcontroller | PIC18F2553-I/SP | Microchip Technology | 1 | $6.24 | $6.24 |
| Strain Gauge | BF350-3AA | ICStation | 120 | $0.90 | $108.00 |
| Amplifier | INA827AIDGKR | Texas Instruments | 1 | $2.73 | $2.73 |
| Casing (Estimate) | | Illinois Maker Lab | 1 | $30.00 | $30.00 |
| OR Gate | SN74LVC1G32DCKR | Texas Instruments | 90 | $0.29 | $26.10 |
| **Total** | | | | | **$326.39** |

### 2.4.2 Labor and Total Cost

Labor was calculated according to a salary of $50 per hour at 10 hours per week for a 3-week period for a 3-person team multiplied by 2.5. The cost was calculated as follows:

$50/[hr*person] * 10 [hr/wk] * 3 [wk] * 3 [people] * 2.5 = $11,250

Total cost was calculated by adding cost of parts and cost of labor. The total cost is calculated as follows:
$11,250 + $326.39 = $11,576.39

## 2.5 Schematic and PCB

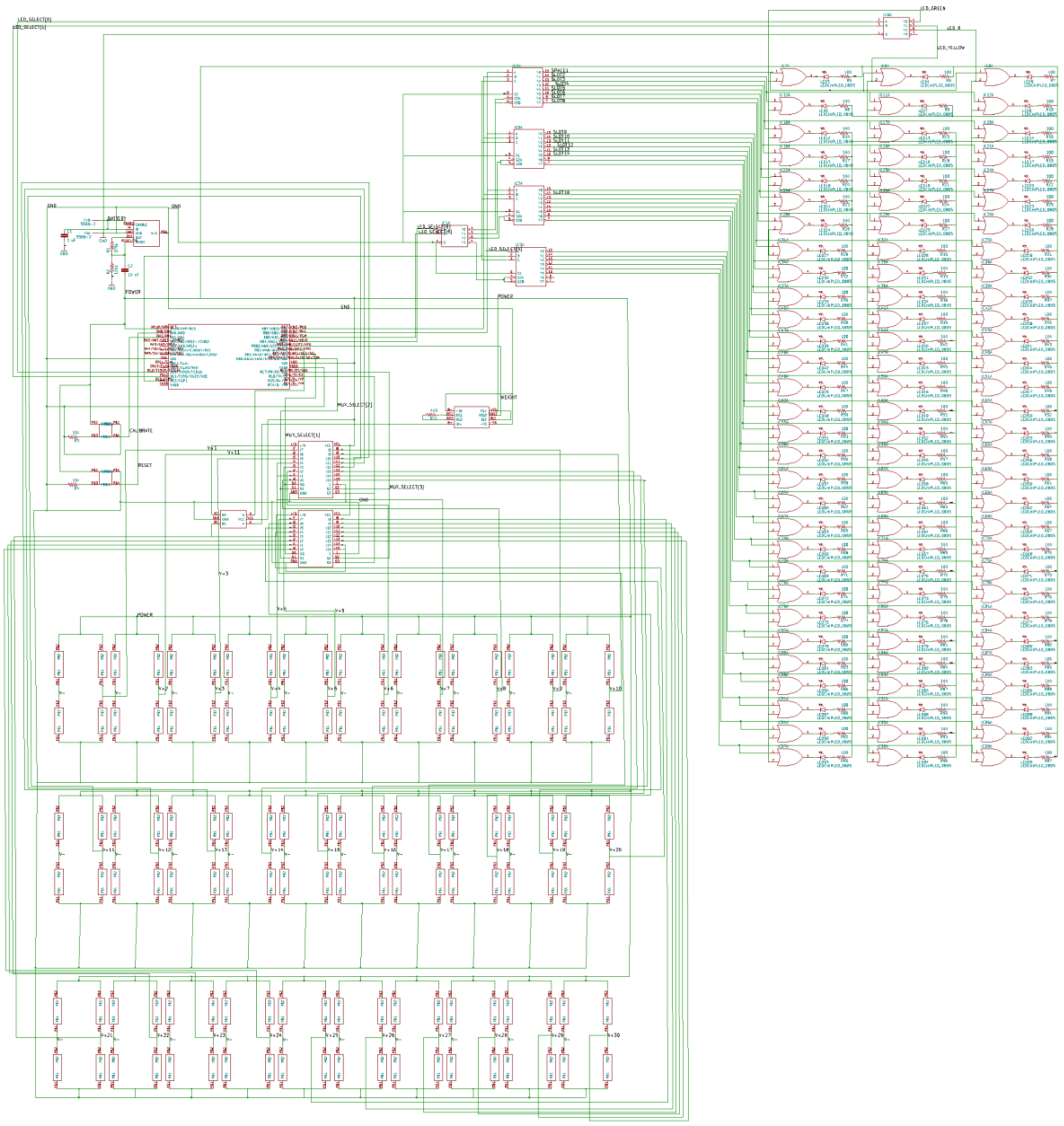

**Figure 10. Electronic Surgical Tray Circuit Schematic.**

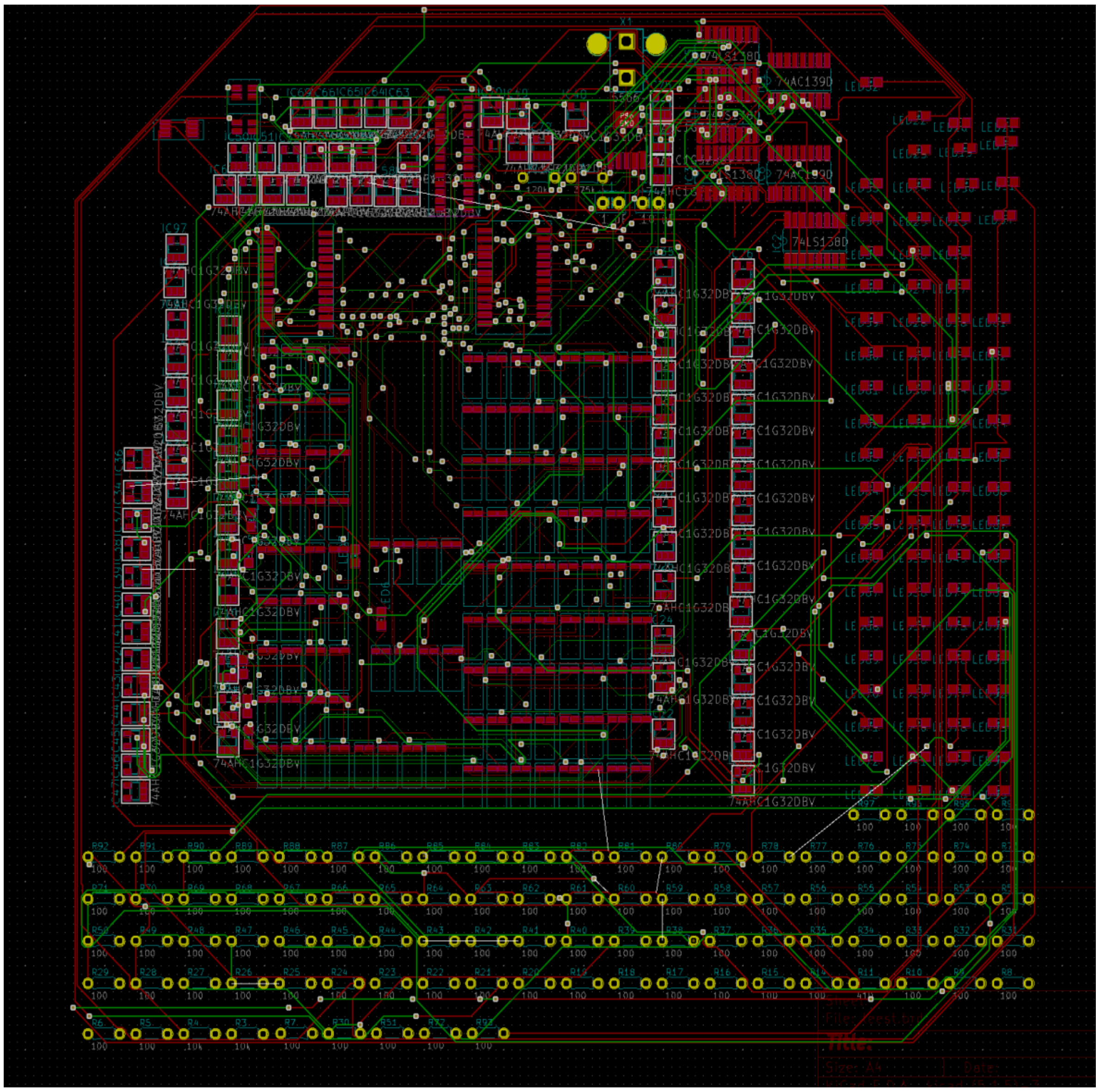

**Figure 11. Electronic Surgical Tray System PCB.**

# 3. Conclusions

## 3.1 Implementation Summary

As noted in section 2, we have been successful in determining how to create a system which would meet the high-level requirements we have set as well as solve the problem presented. In this report the sensor design and operation is presented in detail and the microcontroller algorithm and code is also explained. The overall circuit and schematic for the device is also shown but the details of the other less essential modules needed for the successful operation of this device are not presented in this report.

Erik Lundin focused on the sensor implementation which consists of the load cell and signal processing modules. This was done using four strain gauges in a Wheatstone Bridge configuration. This allows the downward force of the tools to be read as a voltage level. This voltage is then multiplexed so the 30 load cells can be read and amplified to provide a range of voltages that can be read by the microcontroller. SPICE simulations were performed to verify the correct operation of the amplifier.

David Yan focused on the microcontroller algorithm implementation in the MPLAB IDE for the microcontroller using C programming. This was done by first looking at the microcontroller datasheet and flash microprogramming specification to find out how to address the input and output ports in the C program, and then consulting online resources for I/O addressing conventions and consulting the microcontroller algorithm shown in Figure 4 in order to actually write the code. The code passed the IDE's C compiler so it is syntactically correct, and when comparing the code to the algorithm in Figure 4, it is also algorithmically correct. The C program implementation can be found in Appendix B.

Jane Zhao focused on the overall circuit implementation and power module. The schematic and PCB were created according to the specifications as listed earlier as well as the various other circuits as noted in the specific subsystem sections. Both were created using a mixture of Eagle CAD and KiCad. The power module delivers power to all system components and the specific setup including resistance values can be noted in the schematic in Figure 10.

## 3.2 Unknowns, Uncertainties, Required Testing

The biggest uncertainty is regarding noise. Introduction of noise decreases the accuracy of our weight measurements. While our design accounts for noise introduced by the load cell and amplifier, we still do not have a clear understanding of how the analog multiplexer will distort the load cell voltage signal. If this was known, we could account for the decrease in accuracy and determine the nominal number of quantization levels and if any filtering is necessary among other design choices. This analysis would have been done in the lab by feeding various voltages through the multiplexer and noting how the outputs differ from the inputs. It would also be beneficial to see if quickly switching between inputs would introduce any more noise or distortion.

In addition, due to lack of equipment we have been unable to test the microcontroller to determine if there are any optimizations that could be made and to verify that the ADC can sample fast enough at a high enough accuracy to meet the high-level requirements of the design. We would also like to verify that we are able to upload the code to the embedded system and test that each I/O was setup correctly.

## 3.3 Ethics and Safety

The Electronic Surgical Tray is designed to be used in the operating room during surgeries and therefore the first point in the IEEE Code of Ethics, which discusses the safety and health of the public, is distinctly relevant [7]. Surgeries are medical procedures wherein a human patient is operated on, and their tissues are removed, opened, and or modified. Doctors follow the principle of nonmaleficence, or "Do No Harm," and our device should also do no harm to the patients or the device operators, in keeping with nonmaleficence and the IEEE Code of Ethics point #1. Because patient tissues are being opened/operated on, there is a great risk for infection of the body or blood. Consequently, every surgical tool is sterilized, doctors wear personal protective equipment, and the patient lies within a sterile field, primarily to prevent infection of the patient and biological materials from directly contacting the doctors/nurses. In order to prevent infections of patient tissues due to bacteria or viruses, our surgical tray must be able to be sterilized properly. The standard way to sterilize materials is to use an autoclave, which is essentially an oven with high pressure steam. Autoclaves are effective if used at 121°C for at least 30 minutes at 15 psi [8]. To address this requirement, our second high-level requirement for the device is that it must be able to be autoclaved under the aforementioned conditions, meaning the electrical components must be insulated and sealed within the device so that it can withstand high temperatures, high pressures, and steam.

Additionally, given that this device must be able to track surgical tool use for the entire duration of a surgery, the device's battery must be reliable and safe to use. Our design features rechargeable lithium-ion batteries as a secondary power source. In the case of power failure these batteries must be able to power the device for at minimum 14 hours, which should allow for multiple surgeries to take place. To ensure this, the lithium-ion batteries must be able to hold charge reliably. For our design this means that the batteries must be able to charge up to, at least, 75% of the maximum capacity we have listed. To ensure this we will perform charging tests in which we discharge a battery, charge it to max capacity, and then use a voltmeter to determine whether the batteries meet our guidelines.

There is another ethical concern regarding lithium-ion batteries: due to the chemical makeup of most lithium-ion batteries, there is a chance that the batteries will explode and cause physical and/or electrical damage if overcharged [9]. To prevent damage to the device users and the people around them it is necessary to make the charging process safe. This means that the charging process should not allow batteries to be under or overcharged. To ensure this, our design uses an off-the-shelf battery charger in addition to testing. This testing will occur during the same process as the testing involved in the previous concern. However, the battery's temperature and current will also be measured to ensure that the device is operating safely.

Another ethical issue concerns electromagnetic radiation. First, all devices must be tested for their electromagnetic radiation spectrum to ensure that there is compatibility with other devices and no interference with frequencies used for wireless communication. This will not be a problem since no significant RF frequencies are radiated. Additionally, since this device will be within operating rooms, it is necessary that they do not produce significant radiation which could harm the patients as well as staff. Generally, radio frequency is a type of radiation that is emitted by radios, Wi-Fi, and Bluetooth devices, as well as radars among other devices [10]. Radio frequency can cause damage to human bodies if emitted in large enough quantities. According to the FCC's policy on radio frequency safety, "the NCRP's recommended Maximum Permissible Exposure limits for field strength and power density for the transmitters operating at frequencies of 300 kHz to 100 GHz" [11]. However, our device does not utilize RF, Wi-Fi or Bluetooth and is otherwise unlikely to radiate significant amounts of such frequencies, so there is no danger from Joule heating of EM waves emitted from this device.

Finally, with regards to the IEEE code of ethics, #3: " to be honest and realistic in stating claims or estimates based on available data", we will make certain that the requirements and abilities as noted previously in the document or in other supporting documentation are possible [7]. In the case that the device's design changes in any way we will alter the corresponding sections with the correct info, as necessary. We will seek to abide by the IEEE code of ethics, #6: "to maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations". This means that in the case we are unable to carry out any process of the development we will seek advice if needed. If we determine that training or other knowledge is required, we will endeavor to seek training or undergo further research.

## 3.4 Project Improvements

If we had more time to work on the project, the following improvements could be made in order to improve user usage as well as the overall system design.

The first improvement that could be made would be creating a function to allow users to input a list of tools. This allows for easier tray preparation. When the list is inputted into the system, a digital table would be created that holds the item weights. Then instead of all tools being set to unused, the tool status would be set to 'waiting.' As the tools are set on the tray, the system weighs the tools and determines whether the tool weight matches a weight on the list. During this preparation process, the tool-status remains 'waiting' until a separate PREP button is pressed. When the button is pressed, if the tray has been prepared properly and all the items on the list are on the tray, the use-status of all the tools will become unused and the LED indicators will all turn green. If tools are missing, a corresponding number of LED indicators will turn on yellow, to show the user that tools are missing as well as how many are missing. In the case that a tool is placed, and the weight does not match any of the weights on the list, a red LED indicator will light up in the space where the wrong tool has been placed.

The second improvement that could be made would be to add a screen to the user interface so that the user can read more of the system information. This information would include battery level as well as current inventory. To do this we would add a separate screen to the system as well as add a function to the system so that it can read the current battery value and compare it to the full percentage. Ideally this function would also calculate what time would be needed to charge the battery based on the nominal charging rate of the charger and the current battery level for the system and also display this.

The third improvement that could be made would be to introduce power saving settings to the system. Since this system is meant to be used for 14 hours, the time that most surgeries encompass, the internal battery packet should suffice most situations. However, in the case of dire circumstances, such as longterm power outages, there should be an option for the system to conserve power. To do this we would add a separate button to the user interface. When pressed, this should allow the user to lower the LED brightness as well as turn off the indicators and load cells completely in certain slots. If possible, multiple sliders could be used to adjust LED brightness as well as determine how many slots would be turned off. Beyond that, battery level LEDs could be added to the side of the tray or on the user interface panel to allow for the user to determine the current battery level.

## 4. Progress on Previous Project

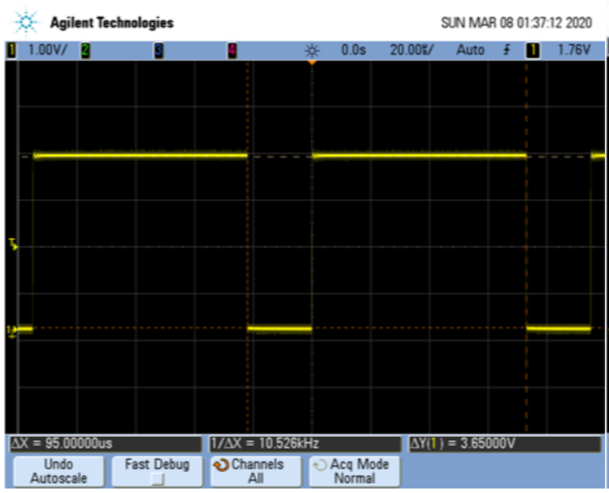

Figure 12. Oscilloscope Trace 1.

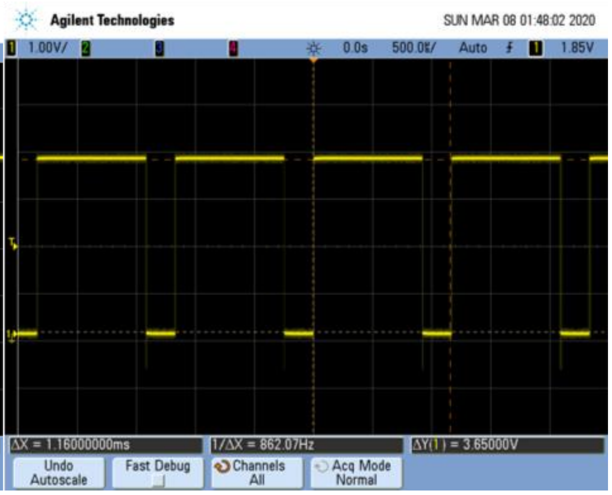

Figure 13. Oscilloscope Trace 2.

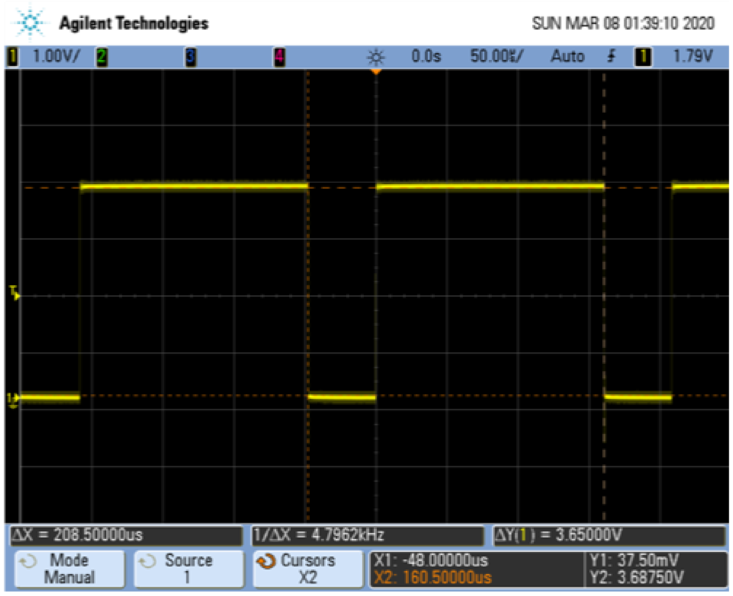

Figure 14. Oscilloscope Trace 3.

Figures 12-14 are oscilloscope traces that were taken when we were conducting requirements & verifications testing. These measurements were made to verify that the frequency of the oscillator circuit could be tuned over the desired frequency range and that the output waveform was a square wave. For all these measurements the oscillator circuit used a 44 pF capacitor with equal resistance values. Frequencies of about 10 kHz, 5 kHz and 1 kHz were realized using corresponding resistance values of 1 MΩ, 2 MΩ and 10 MΩ respectively.

## References

[1]“Digital Enhancements to Handheld and Robotic Surgical Tools Enable Tracking and Other Capabilities in the Operating Room.” Wyss Institute, Harvard University.https://wyss.harvard.edu/technology/smart-tools-rfid-tracking-for-surgical-instruments/.27Mar. 2020.

[2] “RFID for Medical Device and Surgical Instrument Tracking.” Medical Design Briefs, 1 Sept. 2018.https://www.medicaldesignbriefs.com/component/content/article/mdb/features/articles/32814.27Mar. 2020.

[3] “Surgical Tool Tracking: TraxFast.”Paragon Data Systems, Inc.,2020.https://paragondsi.com/solutions/surgical-tool-tracking/.27Mar. 2020.

[4] “An Unobtrusive and Automated System for Tracking Surgical Instruments.” UCSD, NanoMed Tracking. https://medschool.ucsd.edu/som/surgery/news-events/events/scs/Documents/Makale_NanoMedInstrumentTrackingOverview.pdf.27 Mar. 2020

[5] “Wheatstone Bridge.” Transducer Techniques, 2020. https://www.transducertechniques.com/wheatstone-bridge.aspx. 1 April 2020.

[6] Karki, James. "Signal Conditioning Wheatstone Resistive Bridge Sensors." Texas Instruments, September 1999. https://www.ti.com/lit/an/sloa034/sloa034.pdf. 1 April 2020.

[7] “IEEE Code of Ethics.” IEEE, 2020. https://www.ieee.org/about/corporate/governance/p7-8.html. 1 April 2020.

[8] “Autoclave Use.” Princeton University https://ehs.princeton.edu/book/export/html/380. 3 Apr. 2020.

[9] “Lithium Battery Safety.” University of Washington, Apr. 2018. https://www.ehs.washington.edu/system/files/resources/lithium-battery-safety.pdf. 1 April 2020.

[10] “Microwaves, Radio Waves, and Other Types of Radiofrequency Radiation.” American Cancer Society, 5 Jan 2016. https://www.cancer.org/cancer/cancer-causes/radiation-exposure/radiofrequency-radiation.html. 1 April 2020.

[11] “FCC Policy on Human Exposure to Radiofrequency Electromagnetic Fields.” Federal Communications Commission, 24 November 2015. https://www.fcc.gov/general/fcc-policy-human-exposure. 1 April 2020.

# Appendix

## A. Requirements and Verifications Tables

**Table 4. Requirements and Verifications for the Battery Pack.**

| Requirement | Verification |
|---|---|
| 1. Measures 5-7 V. | 1. <br> a. Use voltmeter to probe output voltage of the battery pack. <br> b. Verify that voltage falls within tolerance range listed. |
| 2. Provides max current draw of 750 mA. | 2. <br> a. Connect the battery to a set of 15 parallel 100 Ω, 1 Watt-rated resistors grounded at the other terminal. <br> b. Verify that the total current through the resistors is around 750 mA by wiring an ammeter or digital multimeter in series with the set of parallel resistors |

**Table 5. Requirements and Verifications for the Voltage Regulator.**

| Requirement | Verification |
|---|---|
| 1. Provides 5.0 - 5.5 V at 750 mA current draw. | 1. <br> a. Set up circuit with 1 μF capacitors to ground at both the input and output pins. <br> b. Connect the charged battery pack to the input of the circuit. <br> c. Attach a set of 15 parallel 100 Ω resistors to circuit to serve as load at the output. <br> d. Use a multimeter to measure the voltage regulator's output voltage. <br> e. Use an ammeter to measure the current through the load resistance. <br> f. Verify that the values fall within the tolerance ranges listed. |
| 2. Has a voltage difference between output and input of below 0.5 V. | 2. <br> a. Set up circuit with 1 μF capacitors to ground at both the input and output pins. <br> b. Connect the charged battery pack to the input of the circuit. |

| | |
|---|---|
| | c. Use a voltmeter to measure the voltages at the input and output of the voltage regulator.<br>d. Calculate the voltage drop and verify that it is below 0.5 V. |
| 3. Has an efficiency of above 75%. | 3.<br>a. Set up circuit with 1 µF capacitors to ground at both the input and output pins.<br>b. Connect the charged battery pack to the input of the circuit.<br>c. Use an ammeter to measure the current across the input.<br>d. Calculate the input power using the measured current and the input voltage.<br>e. Use an ammeter to measure the current across the output.<br>f. Use a voltmeter to measure the output voltage.<br>g. Calculate the output power using the measured current and the output voltage.<br>h. Calculate efficiency to verify that it is above 75%. |

**Table 6. Requirements and Verifications for the Microcontroller.**

| Requirement | Verification |
|---|---|
| 1. Has a sampling rate of at least 1500 samples per second | 1.<br>a. Attach oscilloscope cable to microcontroller output pin.<br>b. Create and run test script to output HIGH as frequently as possible for one second.<br>c. Check oscilloscope output to verify that requirement was met. |
| 2. Operates on a 5.0 - 5.5 V power supply. | 2.<br>a. Attach power supply to microcontroller.<br>b. Turn on power supply and vary the voltage through 5.0 to 5.5 V in intervals of 0.1 V.<br>c. Verify that the microcontroller is able to operate through the entire voltage range. |

**Table 7. Requirements and Verifications for the Reset Button.**

| Requirement | Verification |
|---|---|
| 1. When pressed outputs RESET signal and is easily pressed with surgical gloves | 1.<br>a. Create test circuit.<br>b. Attach power supply to input of circuit and oscilloscope to output.<br>c. Put on surgical gloves.<br>d. Set power supply to 5.0 V.<br>e. Press down on button.<br>f. Verify that button press creates an output of 5.0 V by checking the oscilloscope. |

**Table 8. Requirements and Verifications for the Calibration Button.**

| Requirement | Verification |
|---|---|
| 1. When pressed outputs CALIBRATE signal and is easily pressed with surgical gloves | 1.<br>a. Create test circuit.<br>b. Attach power supply to input of circuit and oscilloscope to output.<br>c. Put on surgical gloves.<br>d. Set power supply to 5.0 V.<br>e. Press down on button.<br>f. Verify that button press creates an output of 5.0 V by checking the oscilloscope. |

**Table 9. Requirements and Verifications for the Indicator System.**

| Requirement | Verification |
|---|---|
| 1. Red, Yellow, and Green LEDs are easily distinguishable from each other. | 1.<br>a. Set up test circuit with sections of red, green, and yellow LEDs.<br>b. Turn on all LEDs at the same time.<br>c. Verify that colors are distinguishable. |
| 2. Able to take 7 select signals from the Microprocessor unit and choose the LED colors for any of 30 spaces. | 2.<br>a. Attach power supply inputs in place of the LED_SELECT signals.<br>b. Test all select signal combinations.<br>c. Verify that LED_SELECT[6:2] can choose the space that is the decimal equivalent of the binary value of the select signals. (e.g. LED_SELECT[6:2] = 10000 selects space 16) |

|  |  |
|---|---|
|  | d. Verify that LED_SELECT[0] and LED_SELECT [1] control LED colors. (00 = green, 01 = Yellow, 10 = Red) |

**Table 10. Requirements and Verifications for the Load Cell.**

| Requirement | Verification |
|---|---|
| 1. Gauge factor is between 2.0-2.2. | 1.<br>a. Measure original length L and unstrained resistance R of gauge.<br>b. Compress strain gauge by 2mm using a vice and record new resistance value.<br>c. Calculate change in resistance.<br>d. Calculate gauge factor by using change in resistance over change in length. |

**Table 11. Requirements and Verifications for the Wheatstone Bridge Circuit.**

| Requirement | Verification |
|---|---|
| 1. Voltage output of at least 25 mV at max load at a bias of 5 V. | 1.<br>a. Create circuit shown in Figure 5 with biasing voltage of 5V with a load of 1000g on the load cell.<br>b. Use a voltmeter to read the output voltage at VB.<br>c. Verify that the output is at least 25 mV |

**Table 12. Requirements and Verifications for the Amplifier Circuit.**

| Requirement | Verification |
|---|---|
| 1. Has a voltage gain of 180 – 200. | 1.<br>a. Create circuit shown in Figure 8.<br>b. Attach power supply to inputs of the amplifier circuit.<br>c. Set power input corresponding to VS2 to 5.0 V<br>d. Set power input corresponding to VS1 to 10 mV.<br>e. Use voltmeter to measure output voltage.<br>f. Calculate gain and verify that it falls within the tolerance range. |

**Table 13. Requirements and Verifications for the Analog Multiplexer.**

| Requirement | Verification |
|---|---|
| 1. Adds less than 0.1 mV of noise. | 1.<br>a. Cascade the 16:1 multiplexers to the 2:1 multiplexer.<br>b. Provide 5V to the IC supplies.<br>c. Turn on power supply and set one input to 1.0 V while grounding all others.<br>d. Select the 1.0 V input using the select bits.<br>e. Use a voltmeter to read the output voltage.<br>f. Verify that the added noise is below 0.1 mV. |
| 2. Voltage drop between input and output is less than 1 mV. | 2.<br>a. Cascade the 16:1 multiplexers to the 2:1 multiplexer<br>b. Provide 5V to the IC supplies.<br>c. Turn on power supply and set one input to 1.0 V while grounding all others.<br>d. Select the 1.0 V input using the select bits.<br>e. Use a voltmeter to read the output voltage.<br>f. Verify that the output voltage is within 1 mV of the input voltage. |

## B. Implementation of Algorithm in Figure 4 Using C Programming

```
/*
* File:  main.c
*/

#include <stdio.h>
#include <stdlib.h>
#include <xc.h>


int detectNumber(int value, unsigned int *array, int *swapped_space){

   //assume array is size 30

   int detected=0;

   for(int i=0; i<30; i++){
       if((*(array+i))*1.05>value && *(array+i)*0.95<value){ //+-5%
           detected=1;
           *swapped_space=i;
       }
   }
   return detected;
}


int detectNumber_noindex(int value, unsigned int *array){
   //assume array is size 30
   int detected=0;
   for(int i=0; i<30; i++){
       if((*(array+i))*1.05>value && *(array+i)*0.95<value){ //+-5%
           detected=1;
       }
   }
   return detected;
}


void setLEDs(char UseStatus[], unsigned int *LED_Select_Array){
   int color_code=0;
   for(int space_counter=0; space_counter<30; space_counter++){
       switch (UseStatus[space_counter]) {
           case 'N': color_code=0; //'N' is uNused, green=00
           case 'I': color_code=1; //'I' is In use, yellow=01

           case 'U': color_code=2; //'U' is Used, red=10

           default: color_code=0;//default to unused

       }

       ConvertBinaryToArray(color_code, LED_Select_Array,0,2); //bits [1:0] are color code

       ConvertBinaryToArray(space_counter, LED_Select_Array,2,5);//set [6:2] to current space

       setLEDSelect(LED_Select_Array); //set outputs for LEDs at the current space

   }

}


void setLEDSelect(unsigned int *LED_Select_Array){ //we need pointers here

       PORTBbits.RB0=*LED_Select_Array;

       PORTBbits.RB1=*(LED_Select_Array+1);

       PORTBbits.RB2=*(LED_Select_Array+2);

       PORTBbits.RB3=*(LED_Select_Array+3);

       PORTBbits.RB4=*(LED_Select_Array+4);

       PORTBbits.RB5=*(LED_Select_Array+5);

       PORTBbits.RB6=*(LED_Select_Array+6);

   }


void setMuxSelect(unsigned int MUX_Select_Array[]){ //don't need pointers here

   PORTCbits.RC0=MUX_Select_Array[0];

   PORTCbits.RC1=MUX_Select_Array[1];

   PORTCbits.RC2=MUX_Select_Array[2];
   PORTCbits.RC4=MUX_Select_Array[3]; //Intentional mismatch of index b/c RC3 is not available

   PORTCbits.RC5=MUX_Select_Array[4];

}


void ConvertBinaryToArray(unsigned int val, unsigned int *array, int start_index, int length){

   //integers are 32 bit, we only need 8 bits so assume number less than 8 bits

   //int array_size=sizeof(array);

   for(int i=start_index; i<(length + start_index); i++){

       *(array+i)=val & 0x00000001; //only grab the last bit

       val>>1; //shift right by one bit to move next bit into LSB position

   }

   //all desired positions in the array have been set completing the function

}


int main(void) {

   /*Set the I/O pin statuses

   *12 Digital Outputs

   *2 Digital Inputs

   *1 Analog Input*/

   //TRISA are data direction registers

   /*Analog Input Pins are RA0, RA1, RA2, RA3, RA5, RE1,

   * RE2, RB2, RB3, RB1, RB4, RB0 */


  //ADC Setup
```

```
  ADCON0 &= 0b00000000; //Choose Channel AN0 for A/D conversion [5:2] =0b0000

  ADCON0 |= 0b00000001; //and Enable [0]=1

  ADCON1 &= 0b00001110; //Only AN0 Analog (see page 22 of datasheet)

  ADCON1 |= 0b00001110; //AN0 Only

  ADCON2 &= 0x00;

  ADCON2 |= 0x88;


//PBADEN=0; //Need to configure PortB [4:0] A/D pins as Digital I/O on Reset


  //Tri-State Bits: bit=0 is output, bit=1 is input

  TRISA = 0b00000001; //RA0 is set to Analog Input AN0

  TRISB = 0b00000000; //Set RB0-RB7 to Digital Outputs

  TRISC = 0b11000000; //Set RC0-RC5 to Digital Outputs, RC7-6 are Digital Inputs


  //Let input AN0 be the output of the Amplifier subsystem

  //Let RB6 to RB0 be LED_Select[7]

  //Let RC5-RC4 and RC2-RC0 be MUX_Select[5]

  //Let RC7 and RC6 be CALIBRATE and RESET respectively


  //Variable Instantiations

  volatile unsigned int Current_Tool_Weight= PORTAbits.RA0;

  volatile unsigned int CALIBRATE= PORTCbits.RC7;

  volatile unsigned int RESET= PORTCbits.RC6;
```

```
char Tool_UseStatus[30];

unsigned int MUX_Select[5];

unsigned int LED_Select[7];

unsigned int Tool_Weights[30];

unsigned int Current_Weight[30];


char *Tool_UseStatus_ptr= Tool_UseStatus;

unsigned int *MUX_Select_ptr=MUX_Select;

unsigned int *LED_Select_ptr= LED_Select;

unsigned int *Tool_Weights_ptr=Tool_Weights;

unsigned int *Current_Weight_ptr=Current_Weight;


unsigned int CALIBRATE_flag=0;

int recent_space=0;

int swapped_space=0;


while(1){       //this infinite loop is purposeful, used to repeat the algorithm during operation


  //AREA 1 Code, Wait for CALIBRATE conditionally

  while(CALIBRATE && !CALIBRATE_flag){ //Assume CALIBRATE is active LOW

    if(CALIBRATE==0) //Prevent possibility of getting stuck here afterwards

      break;

  } //CALIBRATE=0 to leave the loop
```

```
//AREA 2 Code CALIBRATE LOOP, CONDITIONAL

if(CALIBRATE_flag==0 || CALIBRATE==0){

  for(int space_counter=0; space_counter<30; space_counter++){

    Tool_UseStatus[space_counter]='N'; //"N" refers to uNused

  }

  setLEDs(Tool_UseStatus, LED_Select_ptr);//Resets all LEDs to green

  recent_space=0;

  for(int space_counter=0; space_counter<30; space_counter++){

  ConvertBinaryToArray(space_counter, MUX_Select_ptr,0, 5); //sets multiplex. to current space

    setMuxSelect(MUX_Select);

    Tool_Weights[space_counter]=Current_Tool_Weight; //CTW is volatile, so updated

    //delay(); We ignore time delay

  }

 }

//END CALIBRATE LOOP


CALIBRATE_flag=1; //has already been calibrated, do not enter loop again


//AREA 3 Code

for(int space_counter=0; space_counter<30; space_counter++){

    ConvertBinaryToArray(space_counter, MUX_Select_ptr,0, 5); //sets multiplex. to curr space

    setMuxSelect(MUX_Select);

    Current_Weight[space_counter]=Current_Tool_Weight; //CTW is volatile, so updated

    //delay(); We ignore time delay
  }

setLEDs(Tool_UseStatus, LED_Select_ptr); //this function sets all LED's, not just one


//AREA 4 Code

for(int space_counter=0; space_counter<30; space_counter++){

  if(Tool_UseStatus[space_counter]=='I')

  {

    if(detectNumber(Tool_Weights[space_counter],Current_Weight_ptr, &swapped_space)){

      Tool_UseStatus[space_counter]=='U'; //tool is now Used

      recent_space=space_counter;


      if(space_counter!=swapped_space){

      //swap tool weights in the weight array if necessary

      unsigned int temp_weight=Tool_Weights[space_counter];

      Tool_Weights[space_counter]=Tool_Weights[swapped_space];

      Tool_Weights[swapped_space]=temp_weight;

      //swap statuses

      unsigned char temp_status=Tool_UseStatus[space_counter];

      Tool_UseStatus[space_counter]=Tool_UseStatus[swapped_space];

      Tool_UseStatus[swapped_space]=temp_status;

      }


    }

  }
```

```
       }


       //AREA 5 Code

        for(int space_counter=0; space_counter<30; space_counter++){

          if(!detectNumber_noindex(Tool_Weights[space_counter], Current_Weight_ptr)){

            Tool_UseStatus[space_counter]='I';

          }

        }

       //AREA 6 Code

       if(!RESET){ //assume active LOW

         for (int i=0; i<1000000; i++); //software delay to make sure RESET was pressed

         if(!RESET){

           Tool_UseStatus[recent_space]='N';

           ConvertBinaryToArray(recent_space, MUX_Select_ptr,0, 5); //sets multiplexers to curr. space

           setMuxSelect(MUX_Select);

           Tool_Weights[recent_space]=Current_Tool_Weight;

         }

       }

             if(CALIBRATE==0){

           CALIBRATE_flag=0;

         } //the following while loop at the start will act as a software delay

     }

     return (EXIT_SUCCESS);

 }
```