ECEN 4013 Individual Research Datasheet

Ben Jespersen
Team 2 – Swordsmiths

Introduction

This datasheet describes the portion of the project involving selecting a microcontroller, managing its connection to the rest of the system, and writing the main program which will run on it. It details the research conducted in this area and the resulting design decisions. Four main research topics will be discussed: microcontroller selection, microcontroller circuit design, selection of a programming language, and breakdown of main program structure.

Microcontroller Selection

Selection Criteria

The first priority in selecting a microcontroller for this project was ensuring that its functions fit the needs of the project. The microcontroller must contain all the peripherals, I/O pins, and features that the product design requires. The figure below shows the overall block diagram for this project, illustrating the connections which will be used between the microcontroller and other components.

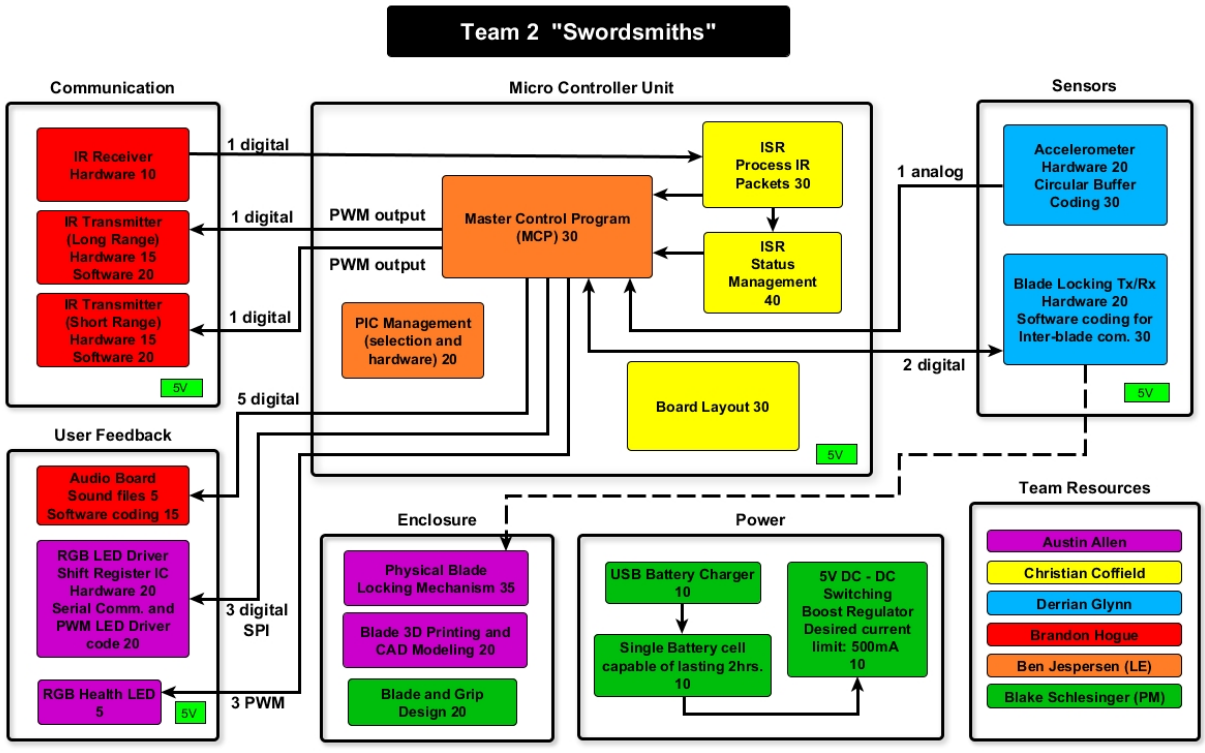


Figure 1: Functional Block Diagram

Based on the block diagram, the microcontroller will need to have a minimum of 17 I/O pins. In addition, it will need to have the following peripherals: analog to digital converter (ADC), pulse width modulation (PWM), and serial peripheral interface (SPI). The required I/O pins are listed below:

- 1 PWM output for IR transmitter
- 1 digital input for IR receiver
- 1 analog input for accelerometer
- 2 digital inputs for communication between blades
- 1 digital output for communication between blades
- 3 pins for SPI communication to blade LED driver
- 3 PWM outputs (or analog outputs) to drive RGB LED health indicator
- 5 digital outputs for audio control

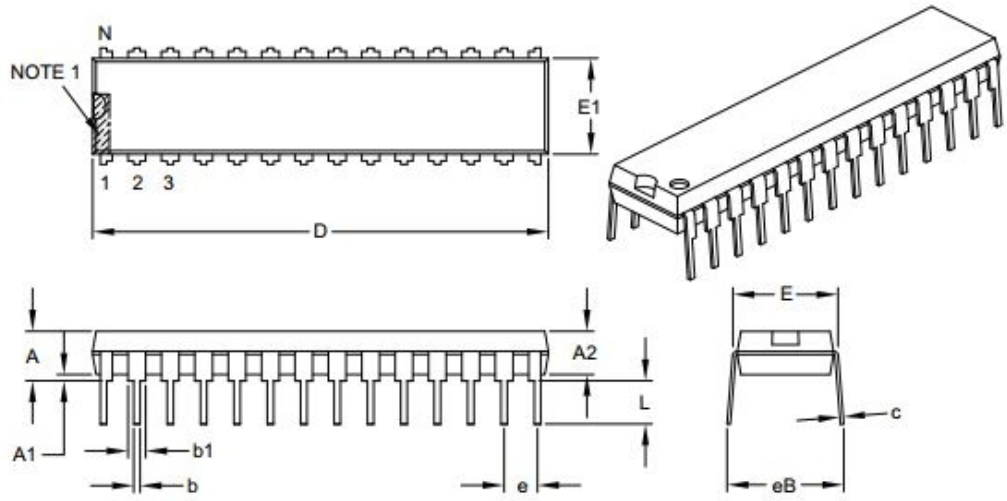
Note that the block diagram shows two IR transmitters with a digital output to each one, but the list above shows only one IR output pin. This is because each sword only needs one transmitter; three of them will have the short range transmitter, and one of them will have the long range transmitter. Also note that the ICSP pins which will be used for programming are not included in this count; this is because the ICSP pins on the PIC can be used as GPIO as well, so they do not need to be counted separately.

Another important consideration for this particular project is the physical dimensions of the microcontroller. The electronics for the omega blade will need to be housed inside a 1 in. wide space within each sword's hilt, so it is important that the microcontroller be small enough to allow a PCB with traces on either side of the microcontroller to fit inside this space. This also made it important to find a microcontroller which provides only the features needed, with as few extra pins as possible, because each extra pin wastes space. At the same time, however, the short time period designated for this project requires rapid prototyping as well, so a very small surface mount microcontroller would not be an ideal choice because it would be more difficult to quickly program and test than a standard dual inline package (DIP) chip.

After these primary considerations, there were other factors to consider which are more open to design preferences. One of these is the question of which microcontroller manufacturer to use. This is a much less important consideration for this project because each manufacturer generally produces a huge number of processors, so it is likely that the main features needed can be found from multiple manufacturers. Therefore, rather than comparing the merits of the devices produced by each manufacturer, the primary criteria for manufacturer selection is tool availability. Because the ECEN 4013 design lab already contains PICkit 3 programmers and has MPLAB X installed on its computers, it would be advantageous to use a PIC manufactured by Microchip. Finally, as the least critical consideration, I also have personal experience with PIC microcontrollers and own a PIC programmer, so using a PIC will allow the team to immediately begin prototyping without spending time learning an unfamiliar processor architecture.

Primary Choice

The primary choice to satisfy these design goals is the PIC16F1788. This is a 28-pin microcontroller which exceeds the requirements listed in the previous section. Once all I/O, power, and programming connections have been made, only 6 of the chip's pins will go unused, so it is an efficient use of the available space. The chip is a skinny plastic dual inline package (SPDIP), which is the smaller of two standard through-hole IC sizes. The through-hole package will allow prototyping and testing to begin immediately on a breadboard if necessary, and it will also allow easy connection to the target board either through direct soldering or an IC socket. The SPDIP package size is a maximum of 1.400 in. by 0.335 in., which is small enough to fit within our planned enclosure and provide a fair amount of space to either side for PCB traces. The following image, obtained directly from Microchip's datasheet, shows the physical dimensions of the PIC16F1788.



Units		INCHES		
Dimension Limits		MIN	NOM	MAX
Number of Pins	N		28	
Pitch	e		.100 BSC	
Top to Seating Plane	A	–	–	.200
Molded Package Thickness	A2	.120	.135	.150
Base to Seating Plane	A1	.015	–	–
Shoulder to Shoulder Width	E	.290	.310	.335
Molded Package Width	E1	.240	.285	.295
Overall Length	D	1.345	1.365	1.400
Tip to Seating Plane	L	.110	.130	.150
Lead Thickness	c	.008	.010	.015
Upper Lead Width	b1	.040	.050	.070
Lower Lead Width	b	.014	.018	.022
Overall Row Spacing §	eB	–	–	.430

Figure 2: PIC16F1788 Dimensions (Source: Microchip)

The following table lists each I/O function the omega blade requires for the microcontroller, which pin on the PIC16F1788 provides each function, and which peripheral modules of the PIC16F1788 are used. This table demonstrates that the PIC16F1788 is sufficient for our needs by showing that all of the required I/O functions can be mapped to pins with the necessary peripherals, without leaving any function lacking a valid pin.

I/O Function	Pin	Peripheral Used
Infrared signal input	RA4	-
Infrared signal output	RC3	PSMC4A
Health LED PWM output 1	RC4	PSMC1E
Health LED PWM output 2	RC5	PSMC3A
Health LED PWM output 3	RC6	PSMC2A
Accelerometer input	RA5	ADC
Sword-to-sword communication TX	RA0	-
Sword-to-sword communication RX1	RA1	-
Sword-to-sword communication RX2	RA2	-
Audio output 1	RB0	-
Audio output 2	RB1	-
Audio output 3	RB2	-
Audio output 4	RB3	-
Audio output 5	RC7	-
SPI bus SS	RB4	SPI
SPI bus SDO	RB5	SPI
SPI bus SCK	**RB7**	**SPI**
ICSP data	RB7	-
ICSP clock	RB6	-

Notice that the SPI bus SCK pin and the ICSP data pin are both connected to RB7. This is acceptable because, while the PIC is being programmed, the LED driver receiving a clock signal from the SPI SCK pin will not be receiving a data signal from the PIC. The LED driver will therefore not receive any invalid data; it will simply receive a constant 0V on the data line. When the program is running, the programmer will not be in use, so it will not be a problem for the ICSP data line to have the SPI signal on it.

Other benefits of this microcontroller are its internal oscillator and low cost. The internal oscillator is programmable, so it can be set to a custom speed. This speed will primarily be determined by the PWM frequency of 56 kHz required for MIRP transmission. Regarding cost, one microcontroller will be purchased for each sword as well as one extra for prototyping and testing. The cost per unit is $2.22. Including the shipping cost of $6.78, this results in a total cost of $11.10, which is easily manageable for this project.

Secondary Choice

The second choice for the microcontroller is the PIC16F1789. This device is very similar to the PIC16F1789; in fact, the two devices share the same datasheet. The main difference is pin count. As mentioned in the previous section, there will be very few unused pins on the primary choice microcontroller. In addition, some of the different peripherals overlap each other and use the same I/O pins, so there is very little room to change the design if an error is found during prototyping. Using a PIC16F1789 instead would reduce the risk of running out of pins with access to the correct peripherals, and it would provide many extra pins in case more are needed. For example, it would provide the option to control individual blade LEDs directly from I/O pins instead of through an SPI interface. It would also simplify the setup of the peripherals; the primary choice will require some configuration in order to use the SPI connections on the desired pins. The following table shows that the PIC16F1789 also has sufficient pins and peripherals to satisfy the project's needs.

I/O Function	Pin	Peripheral Used
Infrared signal input	RA1	-
Infrared signal output	RC0	PSMC1A
Health LED PWM output 1	RC6	PSMC2A
Health LED PWM output 2	RE2	PSMC3A
Health LED PWM output 3	RD3	PSMC4A
Accelerometer input	RA0	ADC
Sword-to-sword communication TX	RB0	-
Sword-to-sword communication RX1	RB1	-
Sword-to-sword communication RX2	RB2	-
Audio output 1	RB3	-
Audio output 2	RB4	-
Audio output 3	RB5	-
Audio output 4	RB6	-
Audio output 5	RB7	-
SPI bus SS	RA5	SPI
SPI bus SDO	RC5	SPI
SPI bus SCK	RC3	SPI

The main reason for choosing the PIC16F1788 instead of the PIC16f1789 is size. Packaging the product will be easiest if the components are as small as they can be while still satisfying the design constraints. The PIC16F1788 through-hole package is a plastic dual inline package (PDIP), which has a width of up to 0.625 in. This is significantly wider than the SPDIP package of the primary choice, and it would not fit as well in the available 1 in. space. Another reason is design efficiency; it would be wasteful to use a 40-pin PIC16F1789 if only 17 of its I/O pins would be used. Finally, the PIC16F1789 costs $2.51, which is approximately 13% more than the PIC16F1788. This is not a major concern for this project because only five chips will be purchased, but nevertheless it is still a minor reason for using the primary choice.

Microcontroller Circuit Design

Design Considerations

The microcontroller circuit is primarily tasked with three things: connecting power to the microcontroller, providing a reset function, and providing programming connections. In addition, the schematic serves to clearly indicate which pins will be connected to which functions provided by other team members. The most important factors which determined the circuit design are the need for protection of the MCLR pin and the locations of the I/O pins associated with the required peripherals.

The melabs U2 programmer and the PICkit 3 will be used to program the PIC. According to a document provided by melabs describing ICSP connections to the U2, the MCLR pin on the PIC will be driven to 13V during the programming process. This is significantly higher than the swords' VDD of 5V. The MCLR pin is normally connected to VDD, so the high voltage could potentially damage other components in the circuit or the power supply itself. For this reason, it is important to include circuitry that prevents the 13V from appearing on the VDD rail.

The schematic design also depends on the peripherals needed. Certain peripherals can only be used on certain pins, so it is important to ensure other team members' circuits are connected to the correct pins. Similarly, certain pins must be used for ICSP programming as well.

Primary Choice

The figure on the following page presents a block diagram of the primary design choice for the microcontroller connections. It shows the peripheral modules which will be used as well as the inputs and outputs connected to them, and it is specific to the primary microcontroller choice (PIC16F1788).

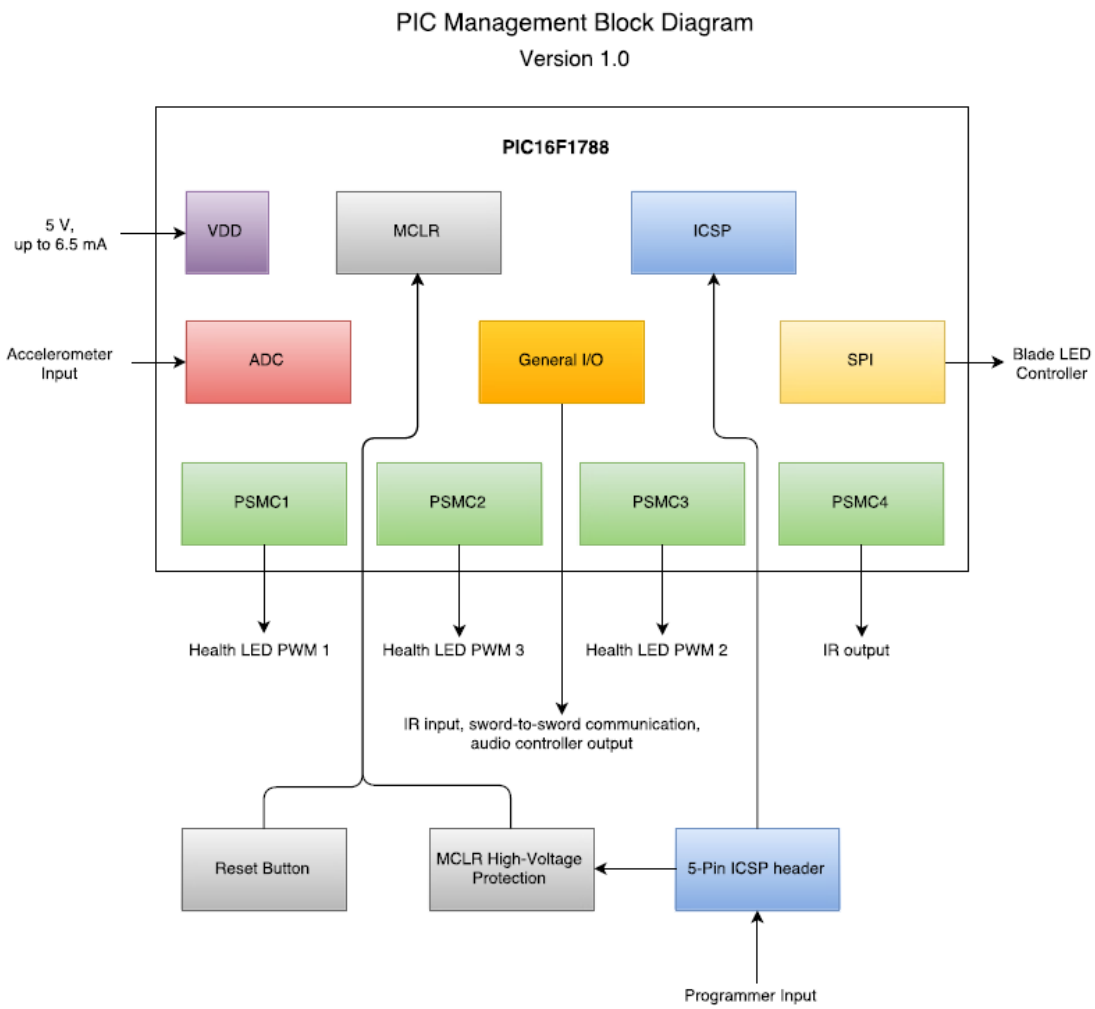


Figure 3: Primary Choice Block Diagram

The schematic on the following page shows the design for the microcontroller circuit and the connections to other functions of the sword. The primary purpose of the schematic is to show how the microcontroller's pins will be connected; the only actual circuitry is the reset button and the MCLR protection diode. An example circuit provided in melabs' description of ICSP uses a 1N4148 diode to protect the VDD rail. Inspection of the datasheet for this diode revealed that it has a maximum continuous reverse voltage of 100V, which is more than enough for this design. This is also a very common diode, so it is easy to obtain. A 1kΩ pull-up resistor was also added to keep the MCLR pin connected to VDD until the reset button is pressed. The reset button is not necessary for operation of the microcontroller, but it is very useful for testing purposes, so it is included in the primary choice.

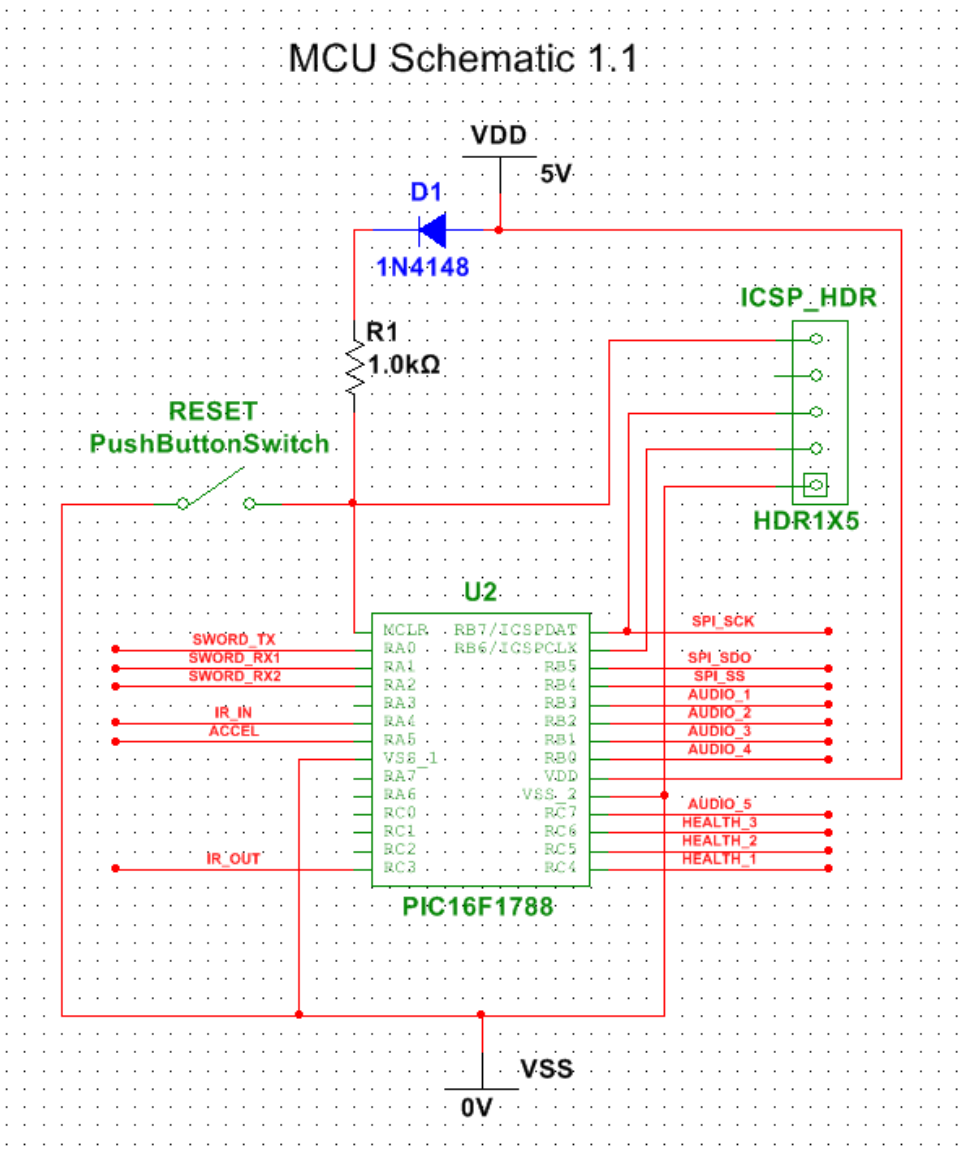


Figure 4: Primary Choice Schematic

Note the input current on the block diagram is labeled as “up to 6.5 mA.” This includes only the operating current of the microcontroller. More current may be needed to drive other team members’ circuits from the I/O pins, but this is dependent on the design of those circuits. Each team member will determine the required current for their own design. The operating current of the microcontroller alone depends on the selected oscillator frequency, and at the maximum frequency of 32 MHz, the current is 2.2 mA according to the microcontroller datasheet. From this, the internal resistance of the microcontroller when it is not driving any external circuits can be estimated to be approximately 2.3 kΩ. Adding a 2.3 kΩ resistor between VDD and ground in the above circuit and closing the switch allowed the worst case current to be roughly estimated by simulation in MultiSim. This simulation resulted in a maximum supply current of 6.5 mA.

The inputs and outputs cannot be simulated in MultiSim because all the functionality of the circuit is determined by software. However, certain types of signals, which can be graphed, are expected to be sent and received on these pins. This applies to the three SPI pins, the IR_IN and IR_OUT pins, and the health LED pins. The remaining pins are simple high/low indicators rather than periodic signals. Note that all digital pins will use a logic level of 5V. If another member of the team designs a circuit requiring a different logic level from the microcontroller, that person must also design a way to boost or regulate the signal.

Learn.sparkfun.com provides a very clear graph illustrating the functions of the different connections in an SPI bus. This graph is reproduced on the following page. Note that the graph labels the data pins as Master-Out-Slave-In (MOSI) and Master-In-Slave-Out (MISO). These pins correspond to SDO and SDI, respectively, on the PIC16F1788. For this project, the microcontroller will not need to receive SPI signals back from the LED driver, so SDI will not be used.

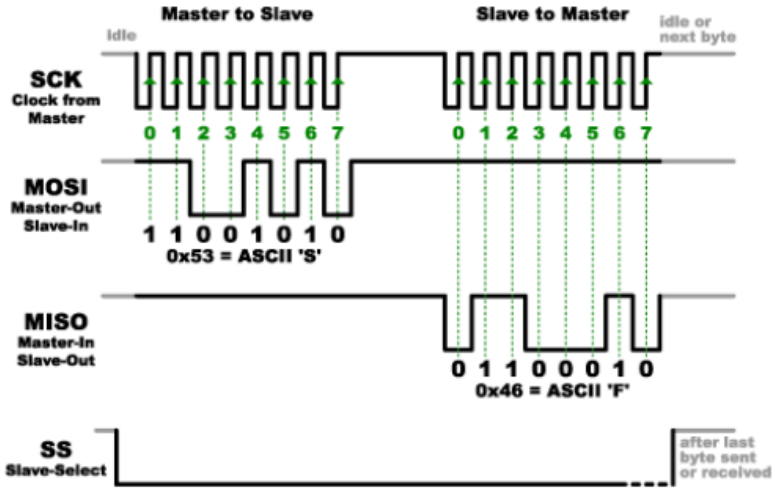


Figure 5: Graph of SPI signal (Source: sparkfun.com)

The IR_IN and IR_OUT pins will also have a very specific signal transmitted on them. This is the Mage Infrared Protocol (MIRP). The details of IR input and output will be handled by Christian Coffield and Brandon Hogue, but to demonstrate what signals the pins will need to handle, the following image has been reproduced from the ECEN 4013 MAGE website.

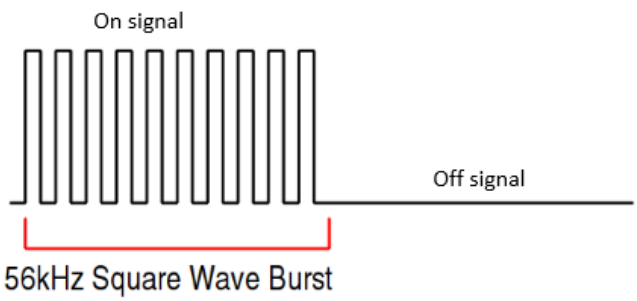


Figure 6: PWM Signal for IR Transmission

The IR receiver will demodulate this signal before transmitting it to the microcontroller, so the microcontroller will see only a digital high constant for the duration of the on signal and a digital low for the off signal. The IR transmitter, however, will be driven by a PWM signal provided by one of the PSMC modules on the microcontroller. The PWM signal will then simply be turned on and off for the required durations to produce the MIRP signal.

Secondary Choice

The following figure presents a block diagram for the secondary design choice for the microcontroller circuit. It shows the peripheral modules which would be used as well as the inputs and outputs connected to them. Unlike the primary block diagram, it is specific to the secondary microcontroller choice (PIC16F1789).

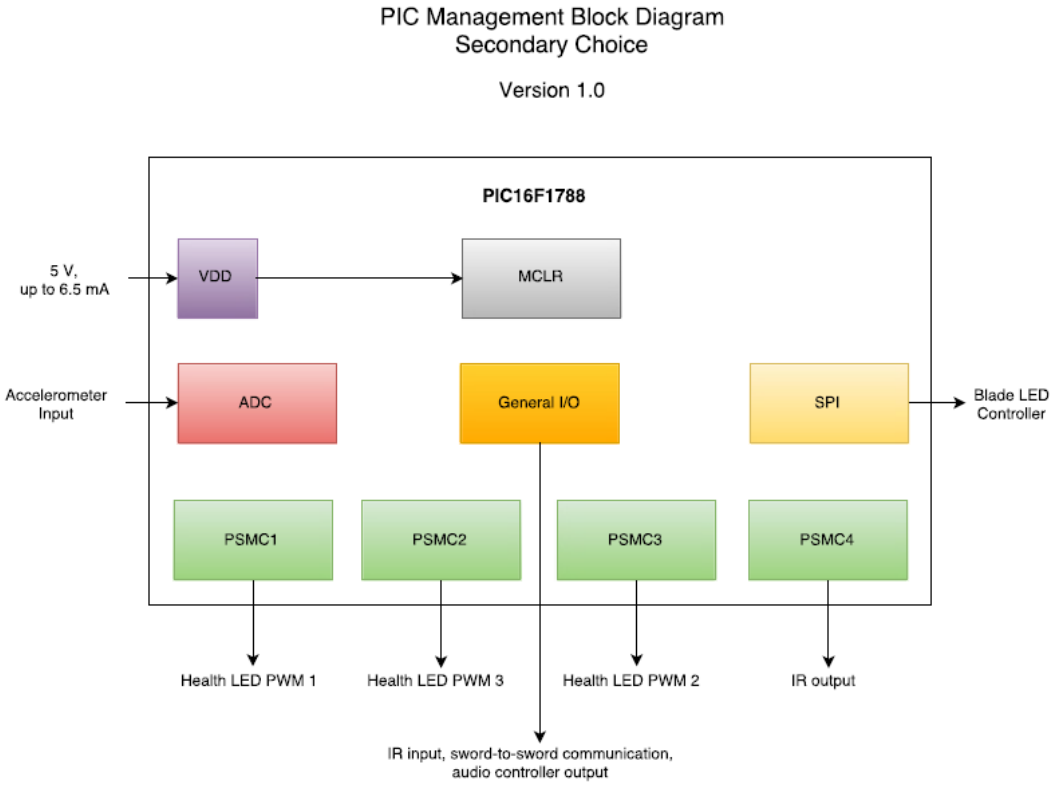


Figure 7: Secondary Choice Block Diagram

It can be seen that, in addition to using a different microcontroller, the secondary design also eliminates some unnecessary features. These include the reset button, the ICSP header, and the MCLR protection circuitry. A reset button, while both convenient and helpful, is not essential to the project; the microcontroller could also be reset simply by cycling the main power. The ICSP header is also not critically necessary; instead of being programmed on-board, the microcontroller could instead be removed from the board and programmed elsewhere. This would be much less convenient than an ICSP header, but it could be accomplished. Finally, the MCLR protection circuit is only necessary when using ICSP, so this can be eliminated as well. The main reason for these omissions in the secondary choice is, again, size. Because the secondary microcontroller is much larger than the primary one, the additional space required will be provided by removing unnecessary components.

The figure below is a schematic showing the pin connections for the secondary choice. Note that, with this option, the only circuitry needed for the PIC is connections to VDD and VSS and a pull-up resistor for MCLR. This makes the schematic extremely simple, but it also eliminates helpful features.

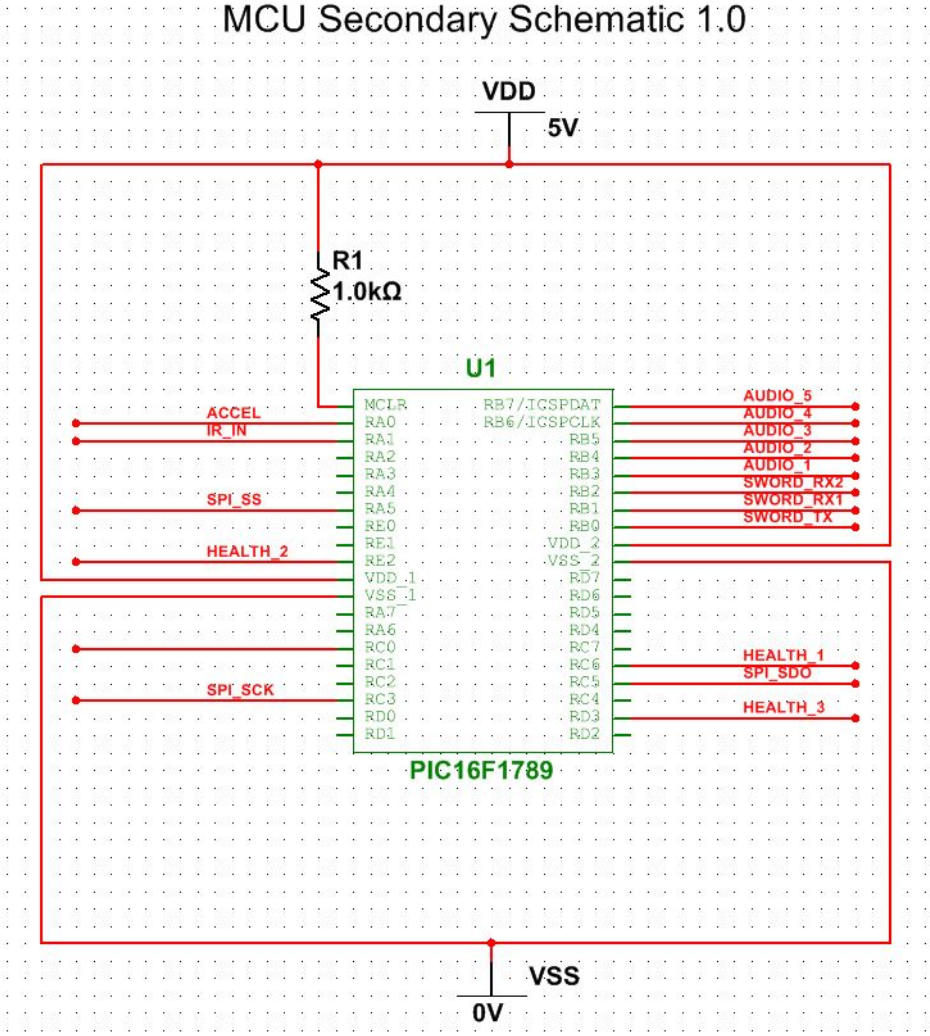


Figure 8: Secondary Choice Schematic

The reasons for selecting the primary choice instead of the secondary choice are that the primary choice has more functionality and it makes more efficient use of the microcontroller's pins. The secondary choice is meant to be used as a backup in case one or more of the features intended for the primary choice fail, so it is intentionally kept very simple to reduce the risk of errors. This design results in 18 unused pins, which is just under half of all the pins on the microcontroller and is three times the number of unused pins in the primary choice. Such a waste of space is not cost-effective and does not satisfy our goal of keeping the microcontroller as small as possible.

Selection of a Programming Language

Selection Criteria

An article by Mouaaz Nahas and Adi Maaita titled “Choosing Appropriate Programming Language to Implement Software for Real-Time Resource-Constrained Embedded Systems” describes what should be taken into consideration when selecting a programming language. The considerations mentioned by Nahas and Maaita are efficiency of resource usage, low-level hardware access, language popularity, and code portability and reusability.

Embedded systems have very limited resources, so it is important to use a language that makes efficient use of them. Object-oriented languages tend to be less efficient than non-object-oriented ones due to the overhead involved. In addition, some languages are simply designed to perform optimally on larger systems with more resources and will not work well in a limited environment.

Low-level hardware access is also important for language selection. Too much abstraction would make it difficult to perform the low-level functions the project requires, such as blinking IR LEDs and reading sensor data. These are tasks that require direct access to the microcontroller’s individual pins and registers.

In addition, it is also very helpful to choose a language with at least some degree of popularity. This is due to a number of reasons, but one of particular importance is that it will be far easier to find instructions and examples for a popular language than an obscure one. Selecting a language with little access to assistance would cause the team to spend a great deal of time learning the language and researching it when the team could be using its time to actually develop the software. Similarly, it would be valuable to select a language in which at least one member of our team already has past experience. This will facilitate beginning software development early with less time spent on learning a language.

The final consideration mentioned by Nahas and Maaita is code portability and reusability. This is not much of a concern for this project because the code will only be used on a single processor for a single project. There will be no need for the code to be reused in the future (except perhaps for instructional purposes) or transferred to a very different processor.

Primary Choice

Given the considerations described above, the first choice for our programming language is C. One compelling reason for this decision is the fact that I have a great deal of past experience in programming microcontrollers in C. For most aspects of design, this is not a very good basis for selecting an alternative; however, in the area of software design, experience with a language significantly reduces the development time. As the team member responsible for the main program for this project, it will be very valuable for me to have past experience with the

language, and I will be able to assist those among my teammates who do not yet have this experience.

Nahas and Maaita point out that C satisfies all the criteria listed in the previous section. It has both low-level features for hardware access as well as higher-level features to aid in abstraction and organizing program structure. It is extremely popular for use on microcontrollers, more so than any other language, so there is a wealth of resources for assistance available should the team need them. As a non-object-oriented language, it is more efficient than a language such as C++, which produces more overhead.

Finally, C is supported by Microchip's MPLAB X IDE, which is designed for programming PIC microcontrollers. MPLAB is available for use on the computers in the ECEN 4013 design lab, so it would be wise to use C rather than spend time and energy to set up an alternative programming technique for a different language that MPLAB does not support.

Secondary Choice

The secondary choice for a programming language is C++. This language is also supported by Microchip's compilers, so the tools available in the ECEN 4013 design lab could still be used to program the PIC. As an object-oriented language, C++ would allow for greater abstraction in the code and may even aid in breaking down the software into smaller sub-problems for each team member.

There are three main reasons for choosing C over C++: past experience, popularity, and efficiency. Given the small expected size of the program, it is not anticipated that there will be a need for the abstraction benefits of C++. It may be more convenient for some purposes, but it is not necessary. Efficiency should not be much of a problem (again because of the small expected size of our program), but as there is no pressing need to use C++, there is no need to sacrifice the efficiency of C.

The team will be able to develop the software much more quickly and effectively in C due to my past experience with it and its popularity. Even though not all the members of the team have experience with C, I will be able to assist those who don't, and C's popularity means all the team members will have plenty of resources available online for more assistance. Given that there is no language which all the members of the team have experience using on a microcontroller, C is the best option for those without experience to learn.

Breakdown of Main Program Structure

Design Considerations

The software for this project will be divided according to the blocks on the overall team block diagram. Each team member who has a software block will be responsible for writing one or more functions which the main program will call. The goal is for the main program to only handle the overall logic of the sword's operation; any code that is specific to a particular function will be written by the team member responsible for that function.

The Beta Blade requires a slightly different program flow than the Alpha, Delta, and Gamma Blades due to the fact that it must play sounds and is not vulnerable to damage or stuns. The sections below will therefore each have two separate flowcharts: one for the Beta Blade and one for the other three blades. The code itself, however, will all be written in a single C project that covers all four blades. Compiler directives will be used to ensure certain code is compiled only for certain blades.

Primary Choice

The following two flowcharts show the primary choice for the main program structure and flow. To handle the different program for the Beta Blade, there is a separate flowchart dedicated to it. However, the code for all four blades will be written in a single C project that covers all four blades. Compiler directives will be used to compile only certain sections of the code for each blade. The benefits of this are that code which is shared between the blades only needs to be written once, all the code for different blades can be viewed at the same time in the same files, and it will be very easy to see the differences between the programs in each file.

Each team member who has a software block on the overall team block diagram is assigned one or more functions to write which the main program will call. See Appendix A for a C header file containing declarations of the functions which the main program will call. Each team member may also write as many additional functions as required to accomplish the required functionality, but only the functions listed in Appendix A will directly interact with the main program.

The Swordsmiths

Main Program for Alpha, Delta, and Gamma Blades

Version 1.1

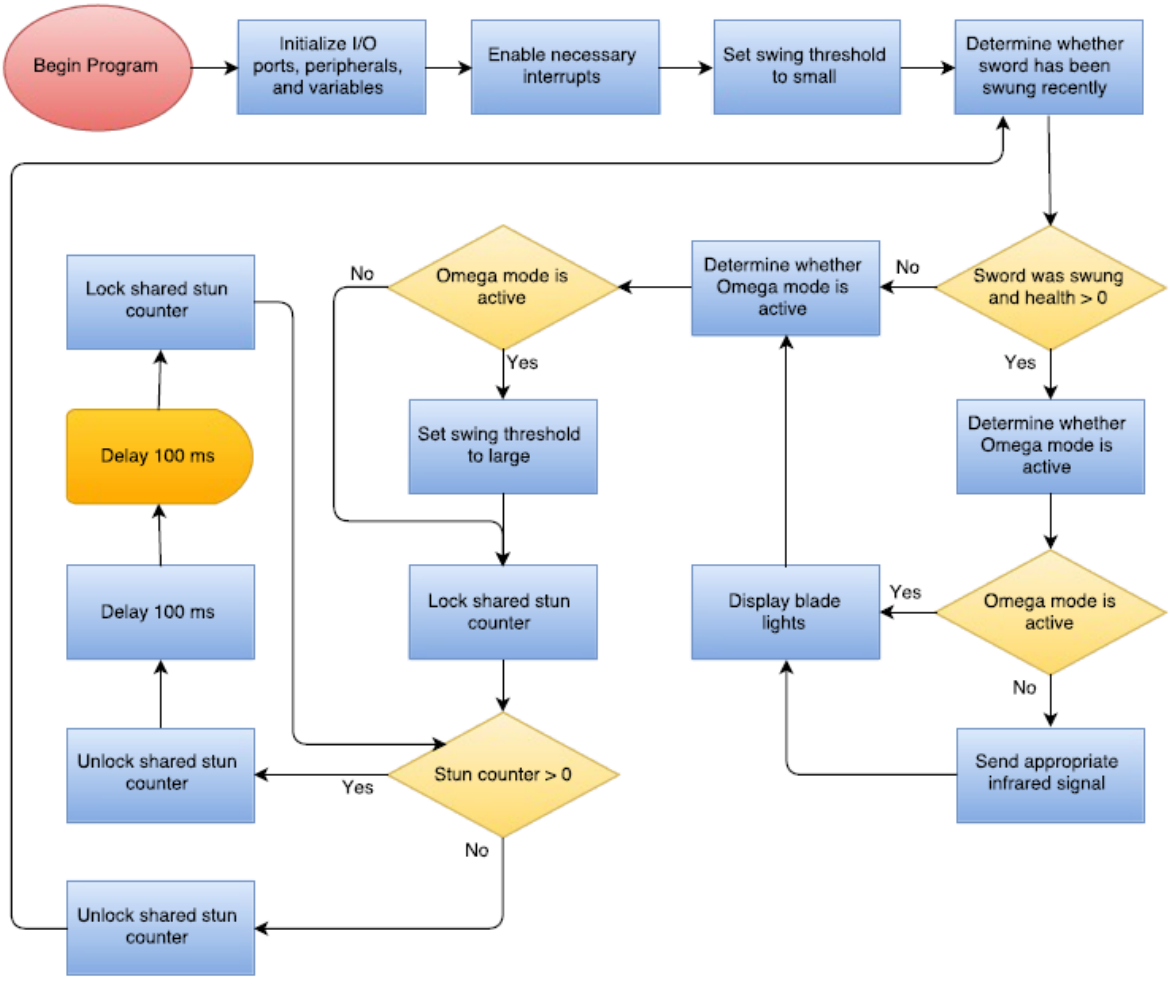


Figure 9: Primary Choice Program Flowchart for Alpha, Delta, and Gamma Blades

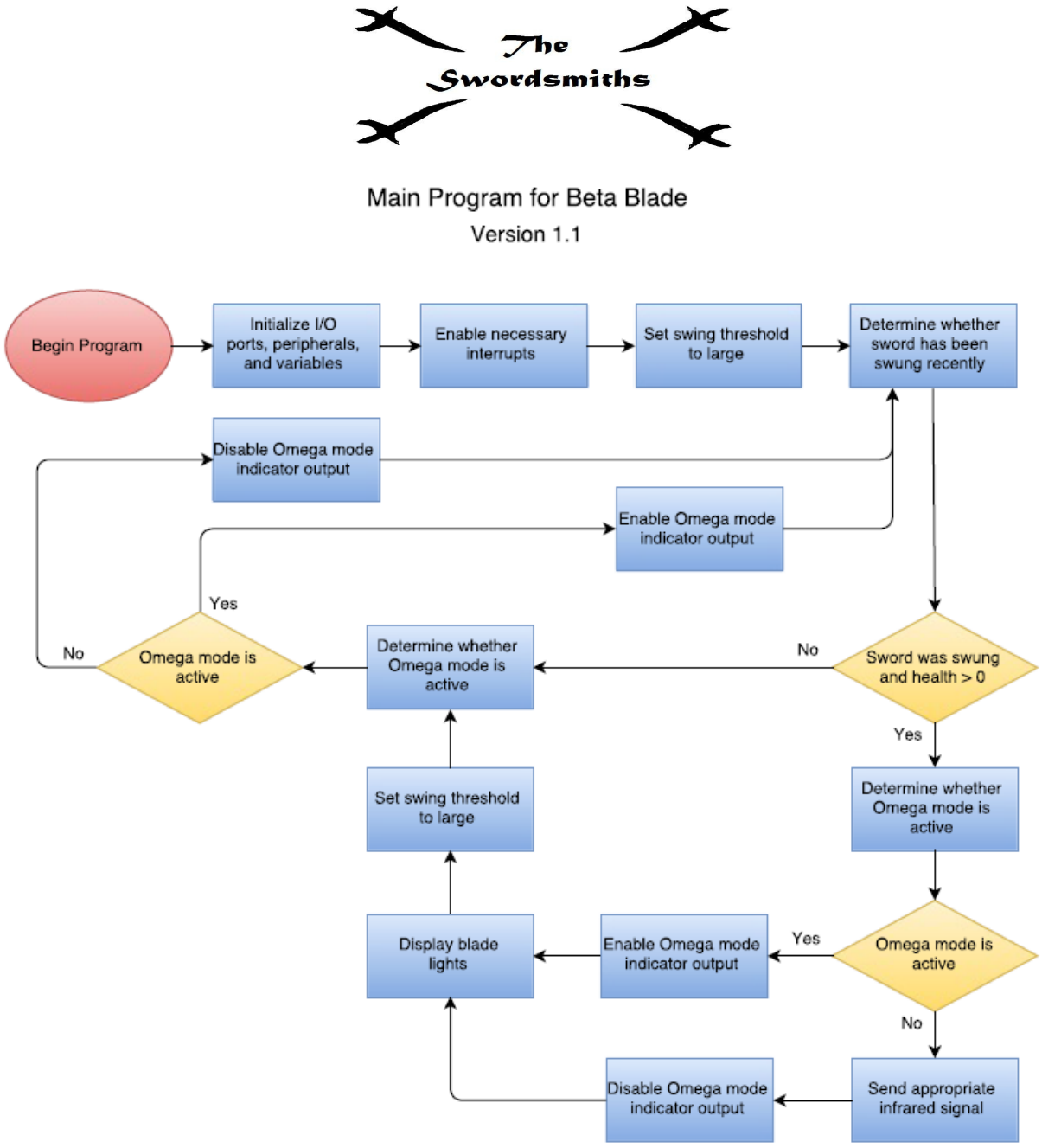


Figure 10: Primary Choice Program Flowchart for Beta Blade

Secondary Choice

The following flowchart shows a secondary option for the Alpha, Delta, and Gamma Blades. The difference from the primary choice is that there is no longer a delay loop to handle stun input. In this option, the delay would instead be handled within the ISR being developed by Christian Coffield. There are two main benefits to this approach. The first is that the flowchart becomes much more similar to that of the Beta blade, so it will be simpler to share code between the different blades. The second is that there will no longer be a need to share a stun counter variable between the main program and the ISR, eliminating the extra code required to lock and unlock this variable every time it is used.

The Swordsmiths

Main Program for Alpha, Delta, and Gamma Blades
Secondary Option

Version 1.0

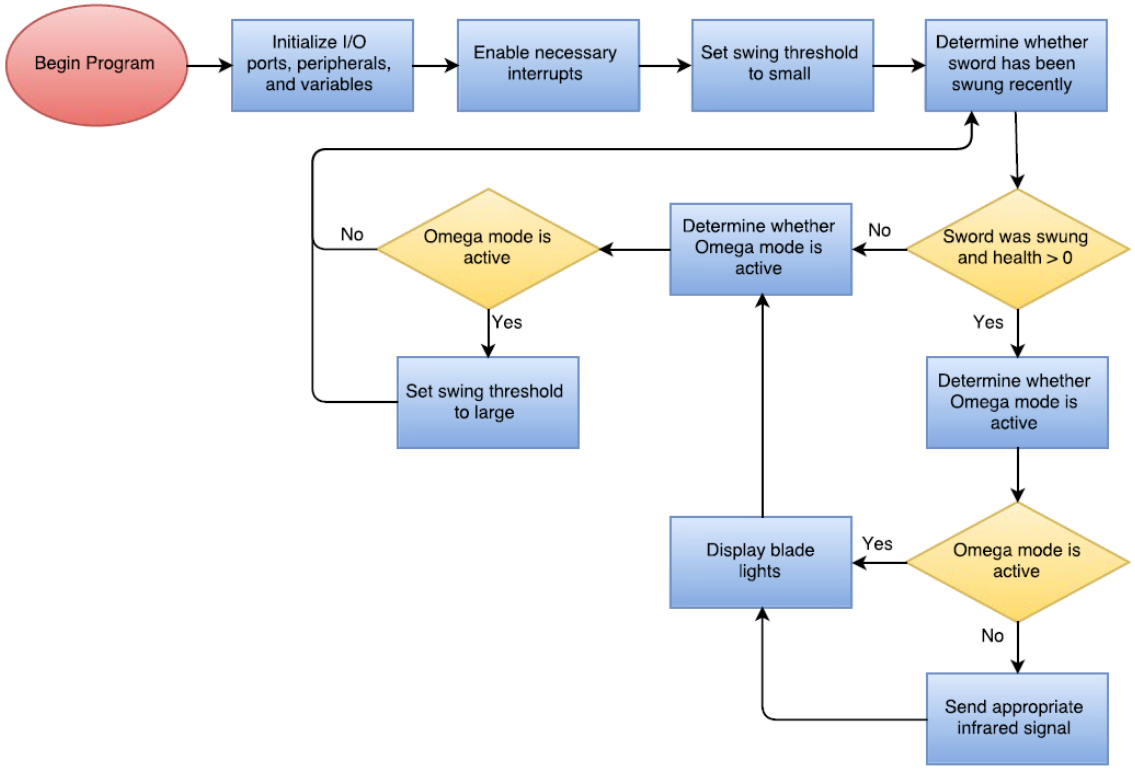


Figure 11: Secondary Choice Program Flowchart for Alpha, Delta, and Gamma Blades

The reason for selecting the primary choice instead of this choice is because, if the delay occurs during the ISR, the program will not be able to receive IR input during the delay. This nullifies the benefit of a stun; no damage can be received while the blade is stunned, so a stun attack by another weapon effectively makes the blades invincible for a short period. This is not optimal behavior. In addition, the stun will always be very short because the program cannot receive additional stun packets while it is in the delay, so it will not recognize if a powerful weapon sent a long stun by sending several packets. Some design could be done to improve the performance (such as making the delay itself very short within the ISR so it can interrupt again soon), but this approach is still error-prone. Also, in the interests of breaking the program into manageable pieces, it would be better to keep the ISR only processing input instead of handling delays as well.

Appendix A: Program Header File

```
/*-------------------------------------------------------------------------
Name: output_ir
Inputs: pkt_type - determines whether to send damage or stun packets
        amount - determines amount of damage or stun to send
Return: void
Purpose: This function is responsible for sending IR signals to other MAGE
         devices. The main function passes the type of signal and amount to
         send, and the function returns after completing its transmission.
         This will frequently require multiple packets to be sent within this
         function. Note that the IR input interrupt may trigger while this
         function is running; therefore the function should periodically
         check the sword's health or stun status and stop transmission if
         necessary.
         If possible, it would be preferable to use this function
         to set up one or more timer or CCP interrupts rather than waiting
         while IR is sent. This is because, when the sword is swung, we would
         like to be able to play sounds and flash lights at the same time as
         the IR packets are being sent.
Developer: Brandon Hogue
-------------------------------------------------------------------------*/
void output_ir(char pkt_type, char amount);

/*-------------------------------------------------------------------------
Name: play_sound
Inputs: sound_selection - indicates which sound the function should play
Return: void
Purpose: This function should play a selected sound based on the input
         variable.
Developer: Brandon Hogue
-------------------------------------------------------------------------*/
void play_sound(char sound_selection)

/*-------------------------------------------------------------------------
Name: determine_sword_was_swung
Inputs: void
Return: This function should return an appropriate value indicating a weak
        swing, medium swing, heavy swing, or no swing.
Purpose: This function is responsible for reading the accelerometer once,
         updating the buffer with the new reading, and determining whether
         the sword has just been swung. If the sword has been swung, the
         function must determine what type of swing it was.
Developer: Derrian Glynn
-------------------------------------------------------------------------*/
char determine_sword_was_swung();

/*-------------------------------------------------------------------------
Name: omega_mode_active
Inputs: void
Return: This function should return true if omega mode is active or false if
        it is not.
Purpose: This function is responsible for checking the inter-blade connection
```


```
        Inputs to determine if the sword is in omega mode. The function may
        not behave exactly the same on every sword, but every sword must
        implement it in some way. The function should simply output true if
        omega mode is active and false if it is not.
Developer: Derrian Glynn
-------------------------------------------------------------------------*/
bool determine_omega_mode_active();

/*-------------------------------------------------------------------------
Name: display_health
Inputs: health - the current health value of the sword
Return: void
Purpose: This function updates the PWM outputs to the RGB health LED. This
        Includes calculating the correct values for the PWM registers based
        on the health input.
Developer: Austin Allen
-------------------------------------------------------------------------*/
void display_health(char health);

/*-------------------------------------------------------------------------
Name: display_blade_lights
Inputs: mode - determines which lighting mode to use
Return: void
Purpose: This function displays a particular lighting mode on the sword's
        Blade LEDs. Different modes may have different colors, flashing
        patterns, etc. This will involve SPI communication to the RGB LED
        strip on the blade.
Developer: Austin Allen
-------------------------------------------------------------------------*/
void display_blade_lights(char mode);

/*-------------------------------------------------------------------------
Name: isr (to be changed once the correct name for the compiler is known)
Inputs: void
Return: void
Purpose: This function will be responsible for interrupting the main program
        when an IR packet is received. It will then process the IR packet
        and alter the health or stun time counter as necessary. It cannot
        receive input from or send output to the main program, so it will
        need to interact with one or more shared variables in the main
        program.
Developer: Christian Coffield
-------------------------------------------------------------------------*/
void isr();
```

Sources

[1] *Datasheet: 1N4148; 1N4448 High-Speed Diodes*. NXP Semiconductors, 2004.

[2] EETimes, 'How to go about selecting a microcontroller', 2012. [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1279383. [Accessed: 21- Sep- 2015].

[3] mage.okstate.edu, 'MAGE Infrared Protocol Specifications', 2013. [Online]. Available: http://mage.okstate.edu/doc/ir/MIRP.pdf. [Accessed: 01- Oct- 2015].

[4] Melabs.com, 'In-Circuit Serial Programming (ICSP) with the EPIC™ Programmer, melabs Serial Programmer, or melabs U2/USB Programmer', 2011. [Online]. Available: http://melabs.com/support/icsp.htm. [Accessed: 21- Sep- 2015].

[5] Microchip.com, 'Microchip Advanced Part Selector'. [Online]. Available: http://www.microchip.com/maps/microcontroller.aspx. [Accessed: 21- Sep- 2015].

[6] Microchip.com, 'PIC16(L)F1773/6', 2015. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/40001810A.pdf. [Accessed: 22- Sep- 2015].

[7] Microchip.com, ‘PIC16(L)F1788/9’, 2013. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/41675A.pdf. [Accessed: 28-Sep-2015].

[8]M. Grusen, 'Serial Peripheral Interface (SPI)', *Learn.sparkfun.com*. [Online]. Available: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all. [Accessed: 01- Oct- 2015].

[9] M. Nahas and A. Maaita, 'Choosing Appropriate Programming Language to Implement Software for Real-Time Resource-Constrained Embedded Systems', in *Embedded Systems - Theory and Design Methodology*, 1st ed., K. Tanaka, Ed. InTech, 2012, pp. 323-334.

[10] *PICkit 3 Programmer/Debugger User’s Guide.* Microchip, 2009.