ECEN 4013 Individual Prototyping Datasheet

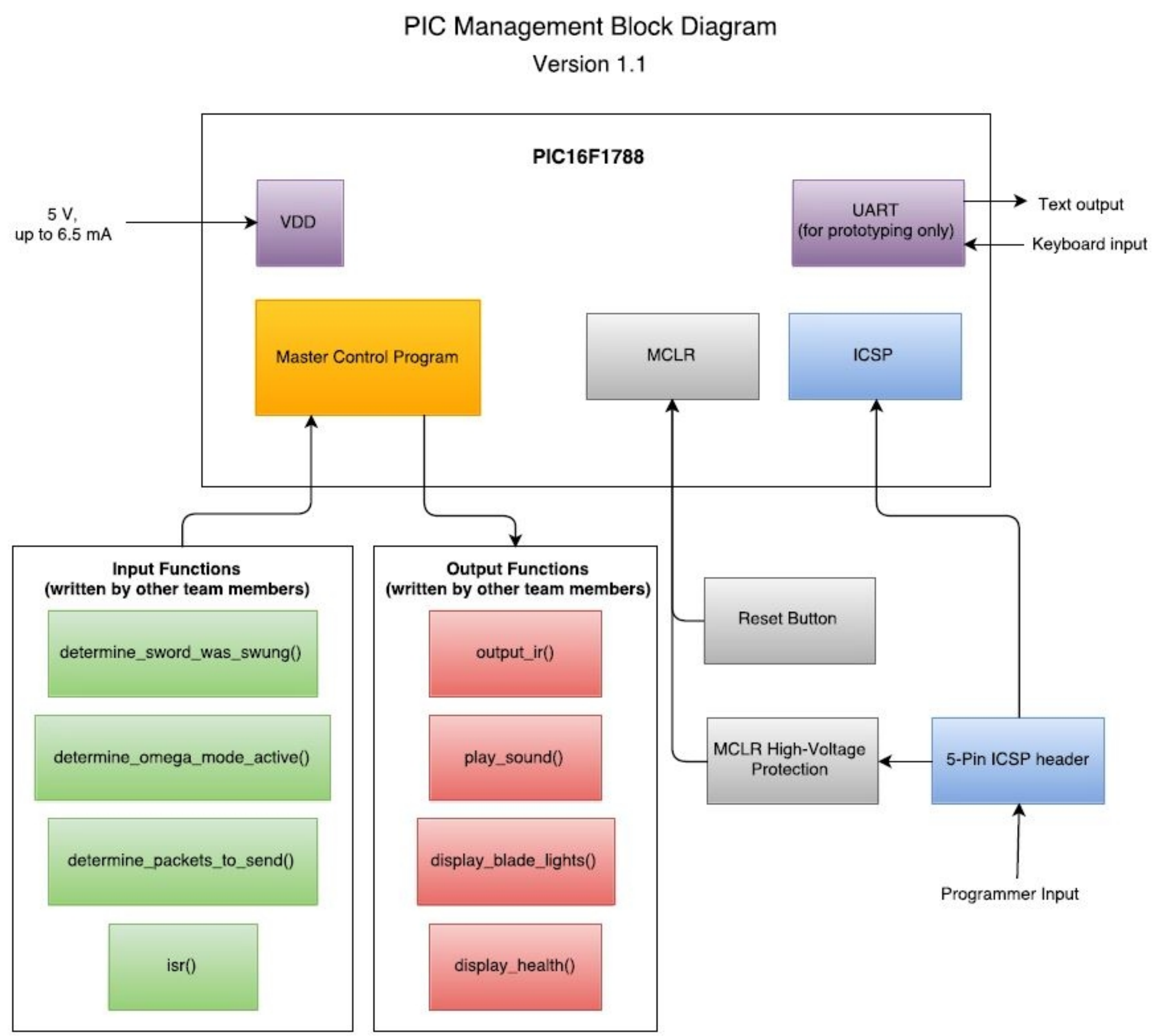
Ben Jespersen

Team 2 – Omega Blade

<u>Diagrams</u>

Block Diagram

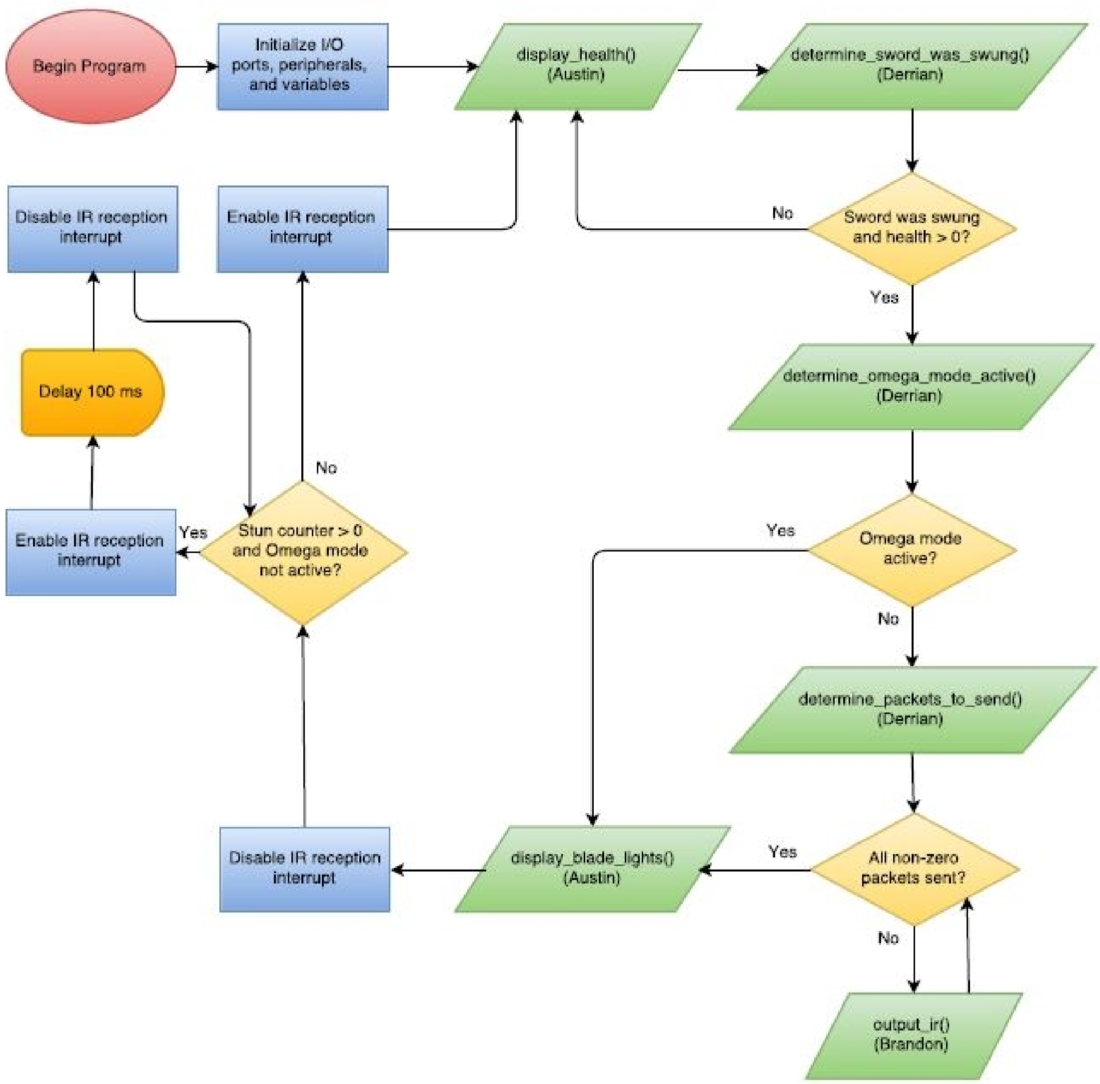
The following block diagram shows the various aspects of the PIC management and master control program blocks. Note that the UART block is not part of the project; this exists only to aid in prototyping.



Flowcharts

Each of the four blades has its own program to provide different functionality. However, the alpha and gamma blades have identical main programs; the differences between these blades are in the lower level functions which are not part of the master control program block. The following three flowcharts show the process of the master control program for each blade.

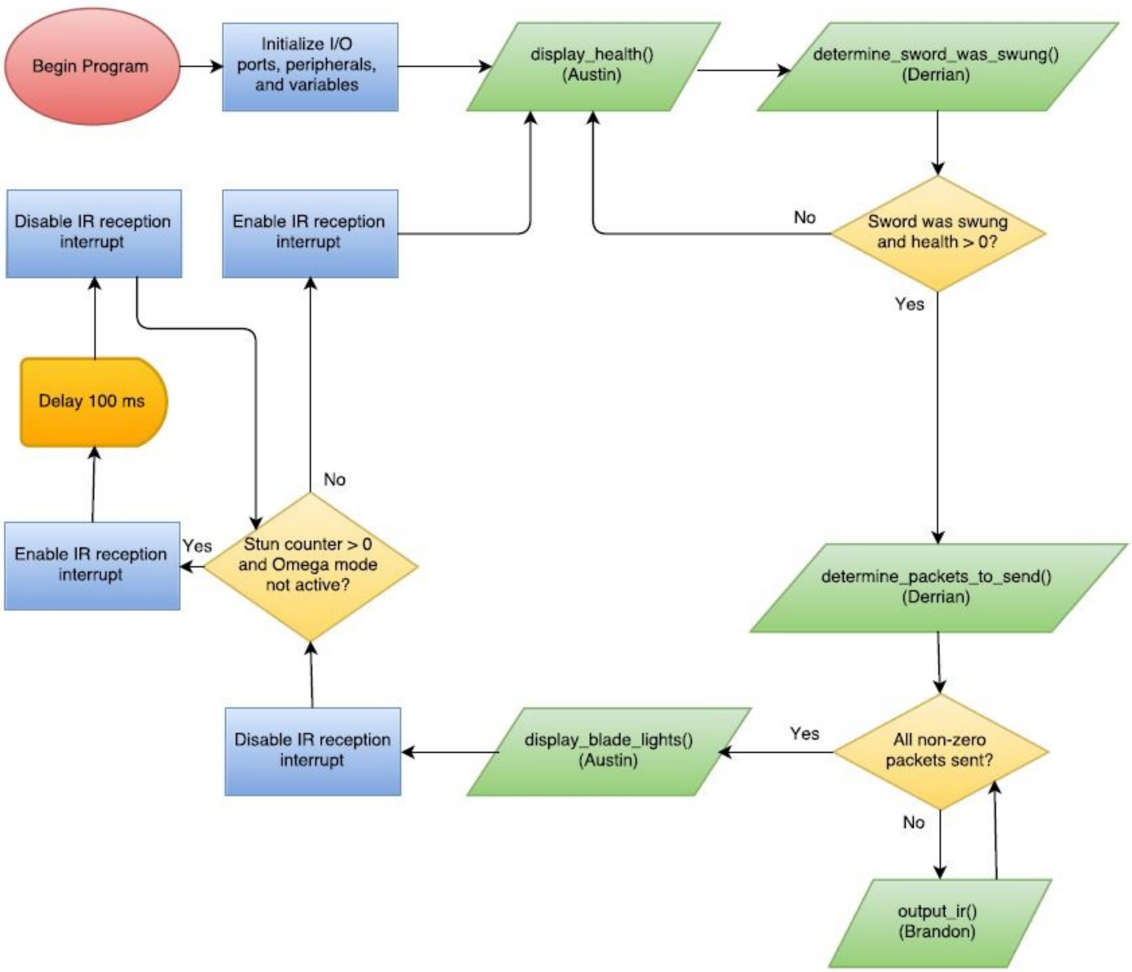
Main Program for Alpha and Gamma Blades
Version 1.3



The Swordsmiths

Main Program for Delta Blades

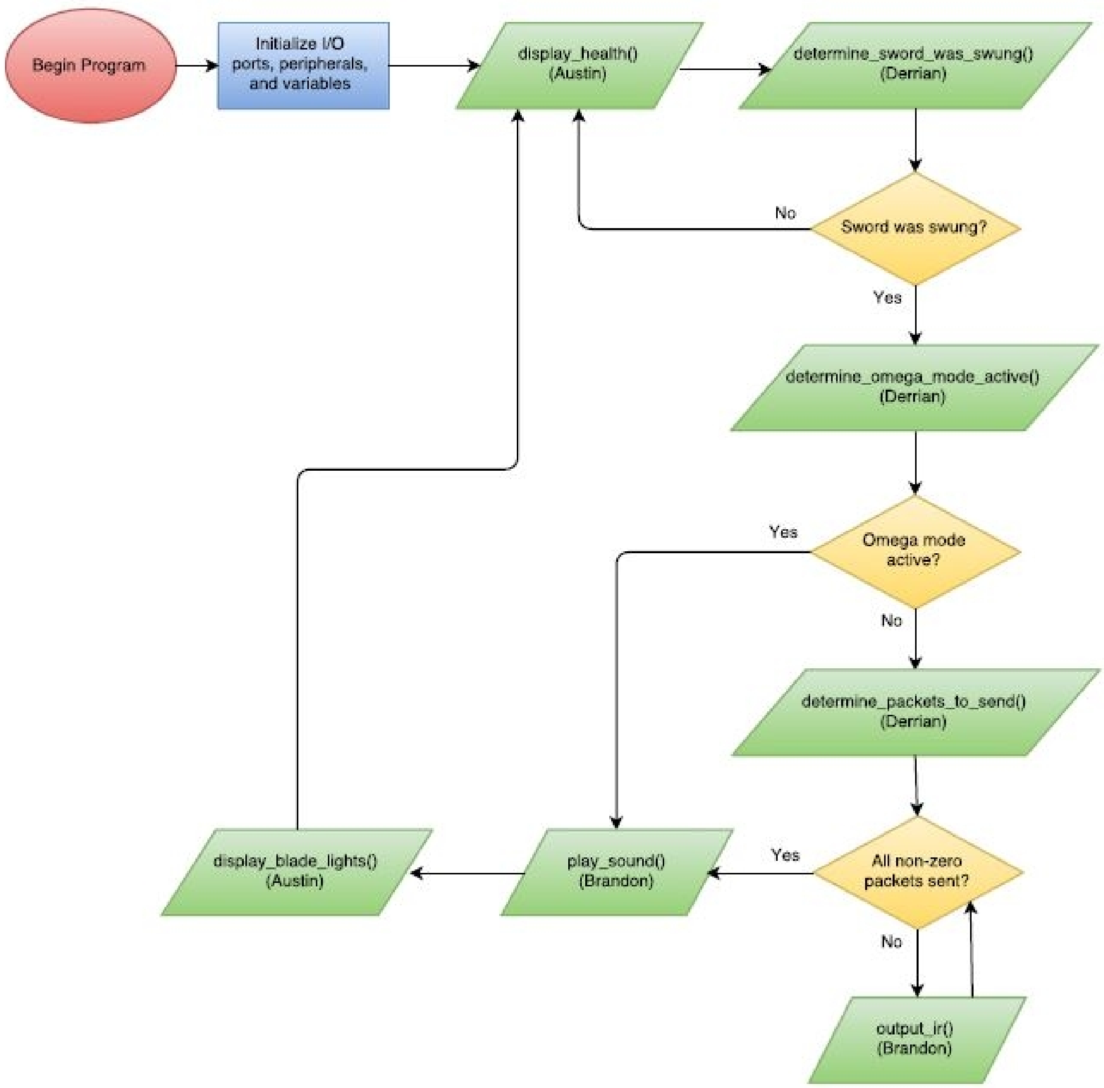
Version 1.3



The Swordsmiths

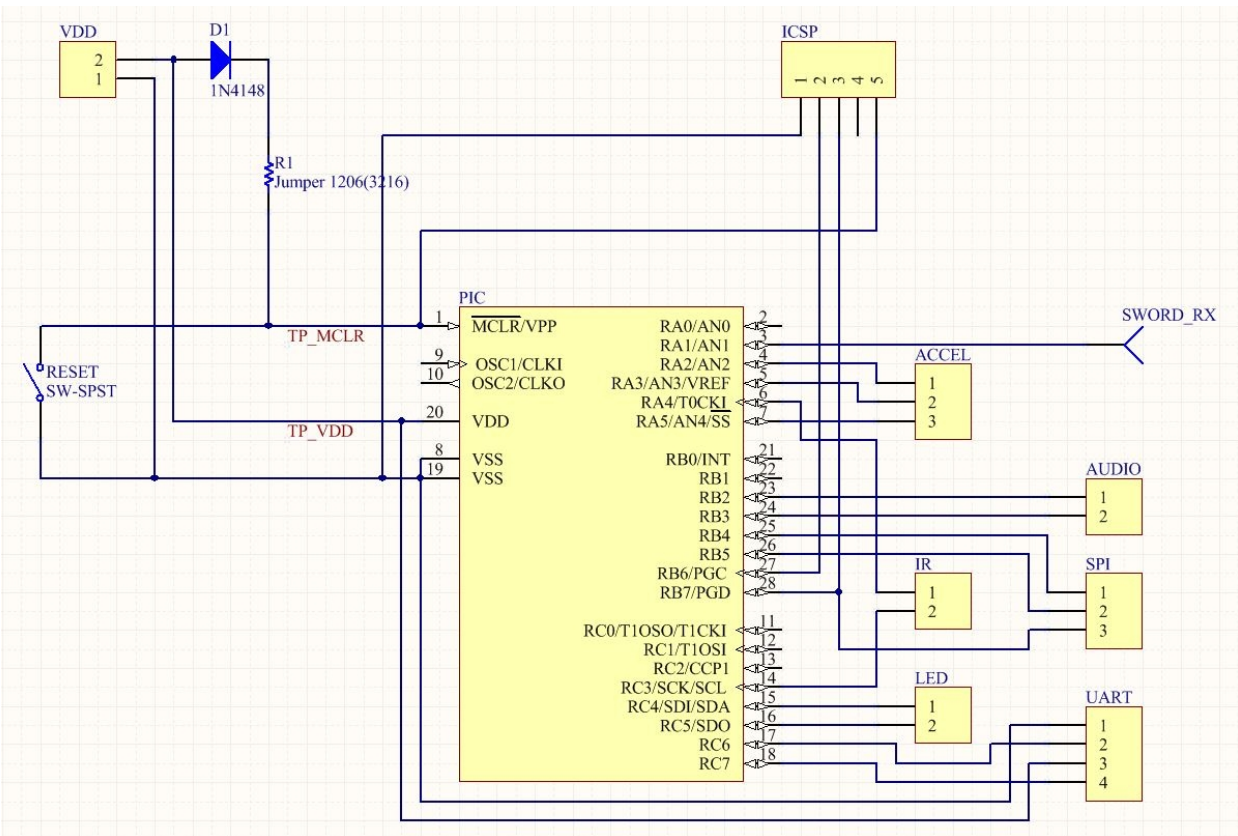
Main Program for Beta Blade

Version 1.3



Schematic

The following schematic shows the design of the prototyping PCB. Note that the design provides several headers for accessing various I/O pins, but the only headers which will be used for this test are VDD, ICSP, and UART. This is because, after the PCB had been designed, it was discovered that using only UART for debugging would be far more insightful for testing the code than connecting indicator LEDs and buttons to the I/O pins.



Inputs

Hardware

Input Name	Description	Expected Signal
In-Circuit Serial Programming (ICSP) signals	The ICSP input receives a hex file from a PIC programmer. This allows the PIC to be programmed multiple times without removing it from the circuit each time.	Since the programming signal is determined by the programmer and is unique to each model of PIC, this signal was not directly measured. Instead it was tested by using it to successfully load a program onto the PIC.
VDD	Power supply to the PIC.	5V, up to 3.5 mA when PIC is running with no external devices.
Reset button	User input to reset the program.	The MCLR pin is normally equal to VDD. When the button is pressed, it will be pulled to ground.

Software

The following table shows the software functions which will provide input to the master control program. These functions are not part of the master control program itself and will be written as part of other team members' blocks.

Input Function Name	Description	Input Parameters	Return Value
determine_sword_was_swung	Read accelerometer and determine if the sword has been swung.	Void	Char 1 or 0, representing true or false.
determine_packets_to_send	Assuming sword was swung, determine the MIRP packets which should be sent based on which sword is in use and whether the other 3 are connected to it.	Pointer to 3-element char array representing number of damage, health, and stun packets.	Void; input array is directly modified by the function.
determine_omega_mode_active	Read sword connectors and determine if all four swords are connected, forming the Omega blade.	Void	Char 1 or 0, representing true or false.
isr	Uses interrupt on change to receive MIRP packets as soon	Void	Void. Function modifies global variables for stun

	as they arrive.		time and health.

Outputs

There are no hardware outputs directly from the master control program. The following table shows the functions, written as part of other team members' blocks, which will be called by the master control program to produce output.

Output Function Name	Description	Input Parameters	Return Value
output_ir	Outputs MIRP packets on an IR LED as instructed by the master control program.	Pointer to 3-element char array representing number of damage, health, and stun packets.	Void
play_sound	Plays one of two sounds on a speaker. Only applicable to the Beta blade, which is the only blade with a speaker.	Char 1 or 0, correspondin g to a particular sound selection.	Void
display_blade_lights	Displays a lighting effect on the RGB LEDs placed along the blade.	Char 1 or 0, correspondin g to a particular lighting effect.	Void
display_health	Updates the PWM signals being sent to the RGB health LED to display a color appropriate to the amount of health remaining.	Void; function directly reads global health variable.	Void

Test Points

Test Point Name	Description	Range of Values
TP_VDD	Used to measure power supply voltage and current drawn by the PIC.	4.0 V – 5.5 V (5.0 V nominal), 2.0 – 3.5 mA
TP_MCLR	Used to verify reset button functionality. Should read 5V when button is not pressed and 0V when button is pressed.	0 V when pressed, VDD when not pressed

As shown in the block diagram on page 1, UART functionality has been added to the microcontroller solely for the purposes of prototyping. This feature will not be present in the final design, but it is useful for demonstrating the program at this stage. All software tests are carried out using keyboard input to the UART module and text output from the UART module.

Software debugging points exist throughout the program. Some of these are executed by stubbed versions of the functions which other team members are writing, while others are executed by the master program itself. For testing purposes, the program continually listens for keyboard input from the user. The user may select which blade to test and whether the blades are connected in Omega mode. In addition, the user may press ‘s’, ‘d’, ‘h’, ‘t’, or ‘o’ to simulate swinging the sword, receiving a damage packet, receiving a health packet, receiving a stun packet, or toggling Omega mode, respectively. The debugging points show the user the results of these simulated actions. The following table shows the locations of the debugging points. The expected outputs will be described in more detail in tables to follow

Debug Location	Description
output_ir()	Displays the type and number of packets to be sent. This also serves to verify that determine_packets_to_send() was called, because if it wasn’t there would be no packets to send.
play_sound()	Indicates that a sound was played, and indicates which of the two sounds it was.
display_blade_lights()	Indicates that a light effect was displayed on the blade, and indicates which effect it was.
main(), when ‘s’ is pressed	Indicates that the sword was swung (if appropriate; see expected output tables). This verifies that determine_sword_was_swung() executed, because this is the function which is called to detect a swing.
main(), when ‘d’, ‘h’, or ‘t’ is pressed	Indicates that a simulated MIRP signal was received and displays the result of the signal to the user.
Main(), when ‘o’ is pressed	Indicates the resulting status of Omega mode to the user.

The tables on the next few pages show the output which should be expected in response to the different input actions available. Since the alpha and gamma blades have identical master programs, they share a table. Note that the delta blade is the only one which sends packets in Omega mode. In addition, the beta blade never receives any packets because it cannot be damaged or stunned.

Alpha and Gamma Blades	
User Input	**Expected Text Output**
‘s’	Sword was swung Sent 5 damage. (If sword is not in Omega mode and health has not Sent 10 health. been reduced to 0) Sent 15 stun. Displayed individual sword swing light show. OR Sword was swung (If sword is in Omega mode) Displayed Omega blade swing light show.
‘d’	The blade has been damaged. Health = X. (Where X is remaining health. Only displayed when not in Omega mode.)
‘h’	The blade has been healed. Health = X. (Where X is remaining health. Only displayed when not in Omega mode.)
‘t’	The blade has been stunned. (This input also causes the program to pause for one second. Only displayed when not in Omega mode.)
‘o’	Omega mode enabled. (If Omega mode was previously disabled) OR Omega mode disabled. (If Omega mode was previously enabled)

Delta Blade	
User Input	Expected Text Output
‘s’	Sword was swung Sent 5 damage. (If sword is not in Omega mode and health has not been reduced to 0) Sent 10 health. Sent 15 stun. Displayed individual sword swing light show. OR Sword was swung Sent 5 damage. (If sword is in Omega mode) Sent 10 health. Sent 15 stun. Displayed Omega blade swing light show.
‘d’	The blade has been damaged. Health = X. (Where X is remaining health. Only displayed when not in Omega mode.)
‘h’	The blade has been healed. Health = X. (Where X is remaining health. Only displayed when not in Omega mode.)
‘t’	The blade has been stunned. (This input also causes the program to pause for one second. Only displayed when not in Omega mode.)
‘o’	Omega mode enabled. (If Omega mode was previously disabled) OR Omega mode disabled. (If Omega mode was previously enabled)

Beta Blade	
User Input	Expected Text Output
‘s’	Sword was swung Sent 5 damage. (If sword is not in Omega mode) Sent 10 health. Sent 15 stun. Played Beta blade swing sound. Displayed individual sword swing light show. OR Sword was swung (If sword is in Omega mode) Played Omega blade swing sound. Displayed Omega blade swing light show.
‘d’	(no output)
‘h’	(no output)
‘t’	(no output)
‘o’	Omega mode enabled. (If Omega mode was previously disabled) OR Omega mode disabled. (If Omega mode was previously enabled)

Plans for Integration

Between prototyping and integration, I will make some changes to the master program. First, the UART functionality will be removed to free up the required pins for other purposes. The UART feature is only needed for testing and will not be required in the final project.

In addition, the program as it stands now contains code for all four blades to allow easy testing for any blade without re-programming. After prototyping, compiler directives will instead be used to ensure only the code for one specific blade is compiled at any given time. This will produce separate programs that will go on each blade, and it will save memory space and decrease program complexity on each PIC. Each blade will only contain its own logic.

A minor change is also planned for the PIC circuitry. As it stands now, the ICSP header is only compatible with the pinout of the melabs u2 programmer, which I own and have been using. It was recently discovered that it is not compatible with the PICkit 3. It would be more convenient for the team as a whole if the header was instead compatible with the PICkit 3 programmers found in the senior design lab. This would allow more team members to program the blades if necessary when I am not present, and it would also simplify the programming process for team members who are not yet familiar with PIC programming. This change will be made before the integration board is ordered.