LASER TURRET

Group 31

December 8th, 2012

PRODUCED BY:

Sarah Bass

Fernando Mansilla

Josh Misek

James Tillotson

TABLE OF CONTENTS

DESIGN SUMMARY

Thank you for grading Team 31 Industries wireless laser tag turret. This turret allows the user to remotely operate the turret within a laser tag game through a wireless joystick. The turret and joystick can be seen in Figure 1 and Figure 2. A magnetic dongle activates the turret and then plugs into the joystick in order to enable joystick function. A camera and FPV goggle set provides the user with a 25 degree field of view of the action. Real time information about the turrets health and shields, as well as compass heading, battery life, warnings and thermal overload are fed to the user's heads up display. Pressing button 11 on the joystick cycles through the three different reticles. The motion of the turret is controlled via a wireless joystick. Pulling back on the stick adjusts the position of the laser barrel and camera from -5 degrees to 80 degrees in the vertical direction and a left/right motion of the stick allows the turret to swing through a continuous 360 degrees of rotation horizontally. The movement speed is adjusted using the throttle control on the joystick. The gun pod has 8 white LEDs which provide a Gatling gun style muzzle flash during firing. The LEDs can also be used as headlights. The headlights are turned on by pressing button 3 on the joystick and the head light patterns are cycled through by pressing button 12 while the lights are off. Pulling the trigger fires a short range high rate of fire IR LED to tag opponents. The turret itself can only be tagged from the back. It is equipped with 8 points of shields which regenerate at a rate of one point every five seconds, and a non-regenerating health of 8 points. If all of the health points are used up, the turret automatically shuts off. The turret initiates a shutdown in the event of battery power dropping below the critical level as well as thermal shutdown due to overheat. The turret can be deactivated at any time by pressing button 6 on the joystick.

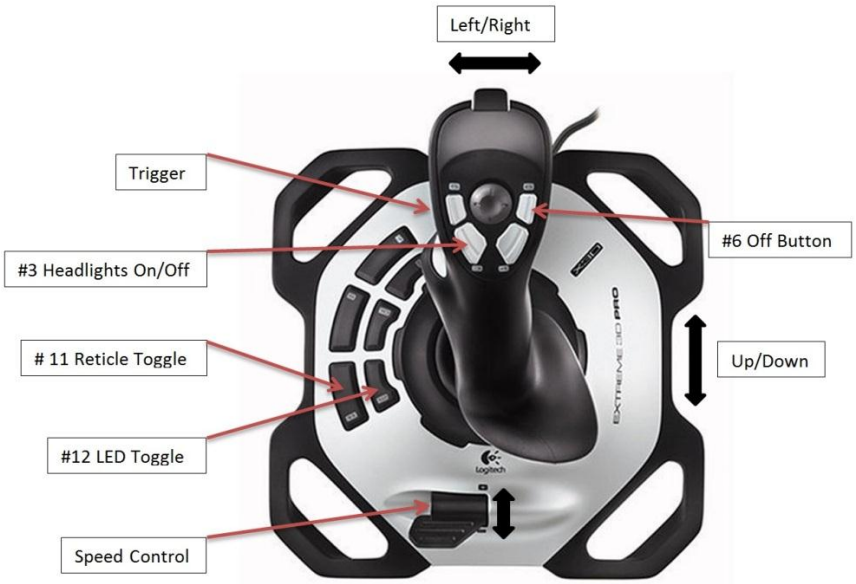


Figure 1: Joystick Configuration and Button Layout.

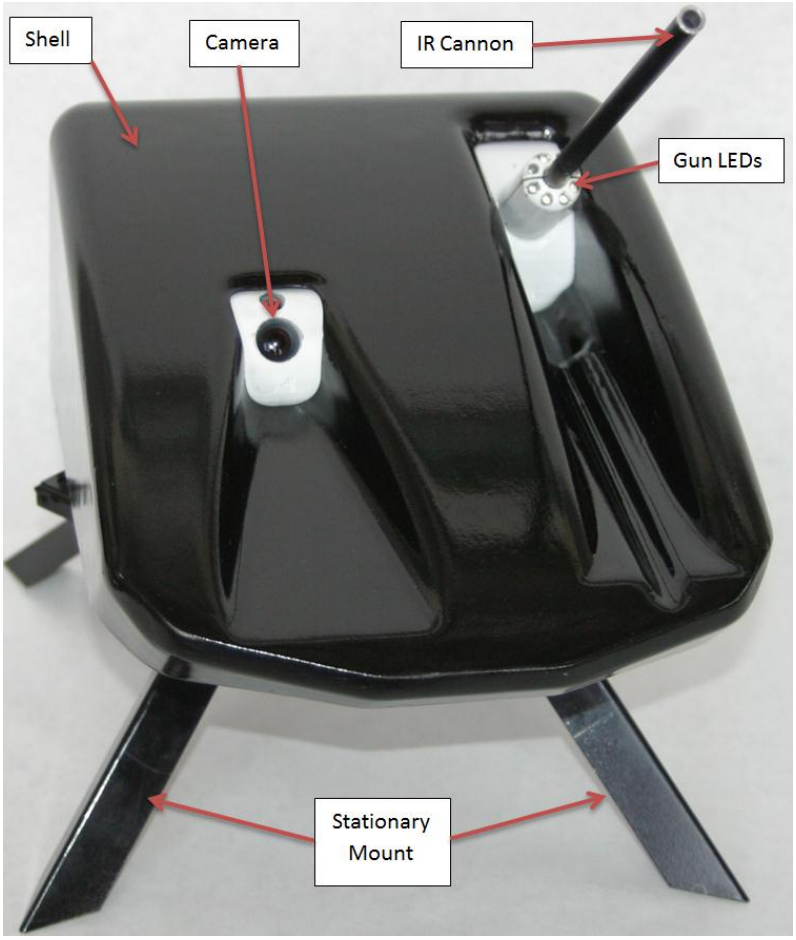


Figure 2: Turret Layout with Shell On.

SYSTEM DETAILS

Team 31 Industries designed, manufactured, and produced a functional and reliable laser turret as seen in Figure 2 allowing a user to remotely operate the turret within a laser tag game by using a joystick much like the one in Figure 1. All signals are sent wirelessly between the turret, joystick, and headset.

A fiberglass molded shell is used for keeping dirt and debris away from essential mechanical and electrical components. The shell is designed with slots to allow the camera and gun pod to interact with objects located in front of the turret. The shell fits snugly around the base of the turret which is in turn mounted to a turn table. The base of the turret is a ¼ in piece of aluminum that was water jet cut to size. It acts as part of the frame and all major components are secured to it. A turn table consisting of two plates and ball bearings is fastened to the bottom of the base. Welded to the other plate on the turn table are four sturdy legs that form a stationary mount on which the base and all other components sit. The shell, stationary mount, camera and gun pods can be seen in Figure 1.

The podules (gun pod and camera pod) were chosen to be 3D printed in order to have the exact dimensions, obtained from a CAD model, needed to precisely encase the electronic components. It was important to have the camera, laser, and IR led oriented perpendicular with the same reference plane in order to have consistent results between the two pods when rotated. Each pod is held in place by two pillow block supports, see Figure 3, which have bearings press fit into them. The supports are threaded and held to the base of the turret with fasteners. A continuous rotation servo controls the pitch of the gun pod from -5 to 70 degrees and a standard servo controls the pitch of the camera pod from -5 to 70 degrees. The camera pod houses a small NTSC CMOS camera which has a 2.5mm lens. Video feed from the camera pod is sent to the First Person Video (FPV) goggles to provide a display for the user. The goggles have an optional video output for spectators The gun pod has 8 holes in a circle around the barrel which fit white LED's in order to visually confirm the trigger is activated and display that the turret is successfully firing. The barrel of the gun is a fiberglass arrow machined down to size. Inside the barrel fits an IR LED which when activated sends an IR signal in the direction it is pointed.

Three servos, labeled in Figure 3, are used in order to move the gun pod, camera pod, and the azimuth of the turret. Two are HSR 5995TG ultra torque servos which run off a regulated 5V source. The servo running the azimuth is PWM controlled for accurate placement within 360 degrees. The other is a HS-55 sub-micro servo also run at 5V but is lighter duty compared to the other two.

In order to turn the turret on a magnetic switch is used and acts like a relay. The axon is programmed to turn the system off when button 6 is pressed A functional diagram for the Axon microcontroller can be seen in Figure 6. A 3300 mAH, 7.4 V lithium polymer battery is attached to a voltage regulator in order to supply a 5V line to the axon as well as to all three servos. The main microcontroller used on the turret is the Axon which was designed and built for robot hobbyists. It has 55+ IO pins, 16 A/D pins for sensors, and 3 UART pins. As far as memory goes it has 64KB of Flash, 4KB EEPROM, and 8KB of SRAM. It also has 6 timers and 9 PWM channels making it a beast of a microcontroller (as James would say). The XBEE is connected to one of the UART channels. The Axon, magnetic reed switch, and battery can be seen in Figure 3.

Mounted to the turret and targets are IR sensors to detect if the unit has been hit by IR light. A laser tag system consisting of a MSP430 and XBEE is then used. The MSP430 microcontroller will signal every time the IR is fired as well as whenever the turret receives fire and is successfully hit. The

MSP430 also generates sounds when firing or when hit. The wiring diagrams for the laser tag system, power board, Axon, PIC, compass, and MAX7456 can be found in the appendix Figures 8-12.

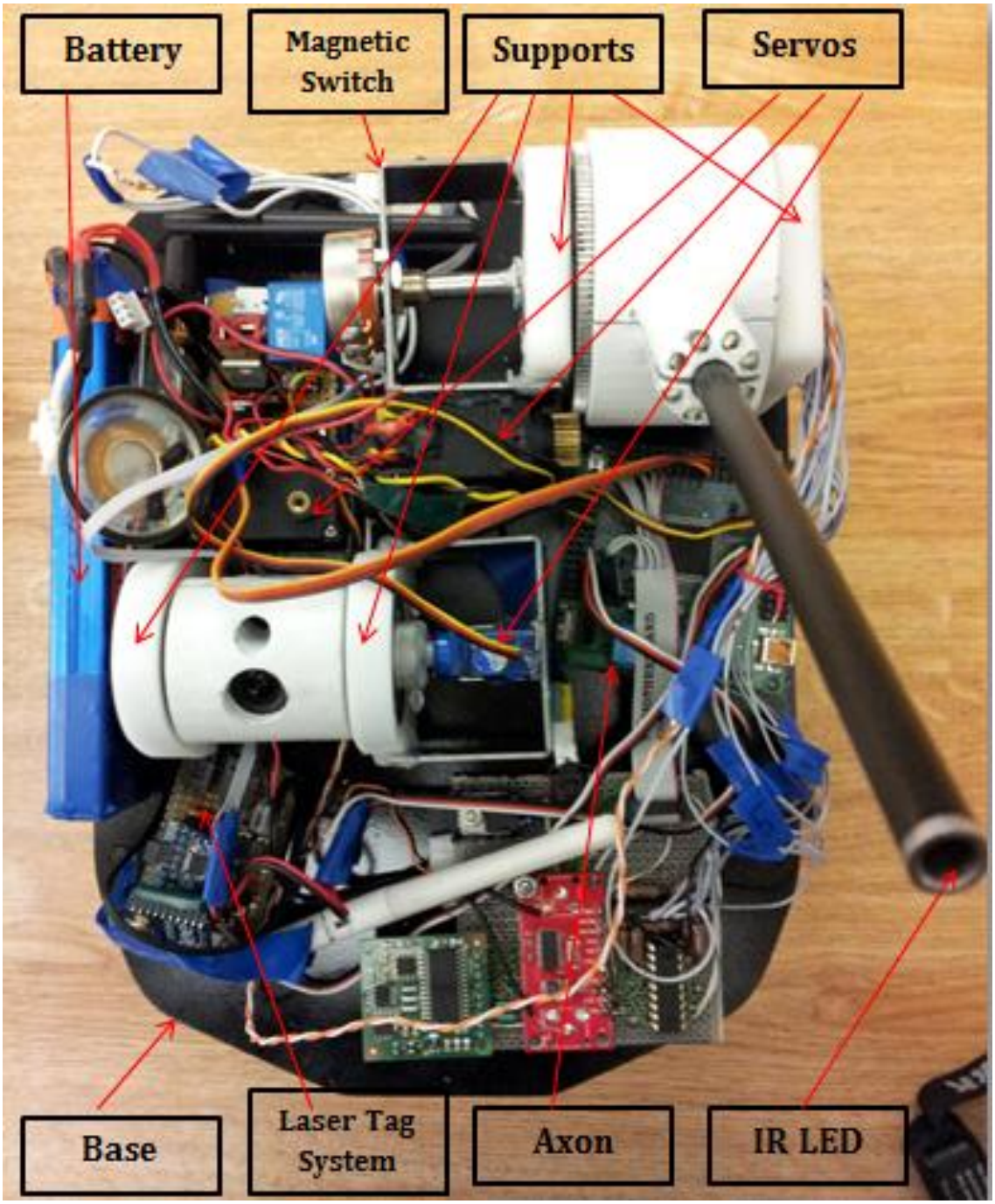


Figure 3: The Layout of the Inside of the Turret.

Mounted to a protoboard, seen in Figure 4, towards the front of the turret is a PIC, on screen display (OSD), and compass. The compass is located on the turret in order to determine the turret direction. A direction update is placed on the video feed whenever the turret is moved. A breakout board for the MAX7456 onscreen display chip is used in order to display important information on the video feed such as compass heading. The protoboard has its own 5 V regulator running off of the main battery. A PIC16F88 is used to display the muzzle flash and two different types of headlights on the 8 white LEDs. Depending on which joystick button is pressed a binary command is sent to determine the sequence of the LEDs.

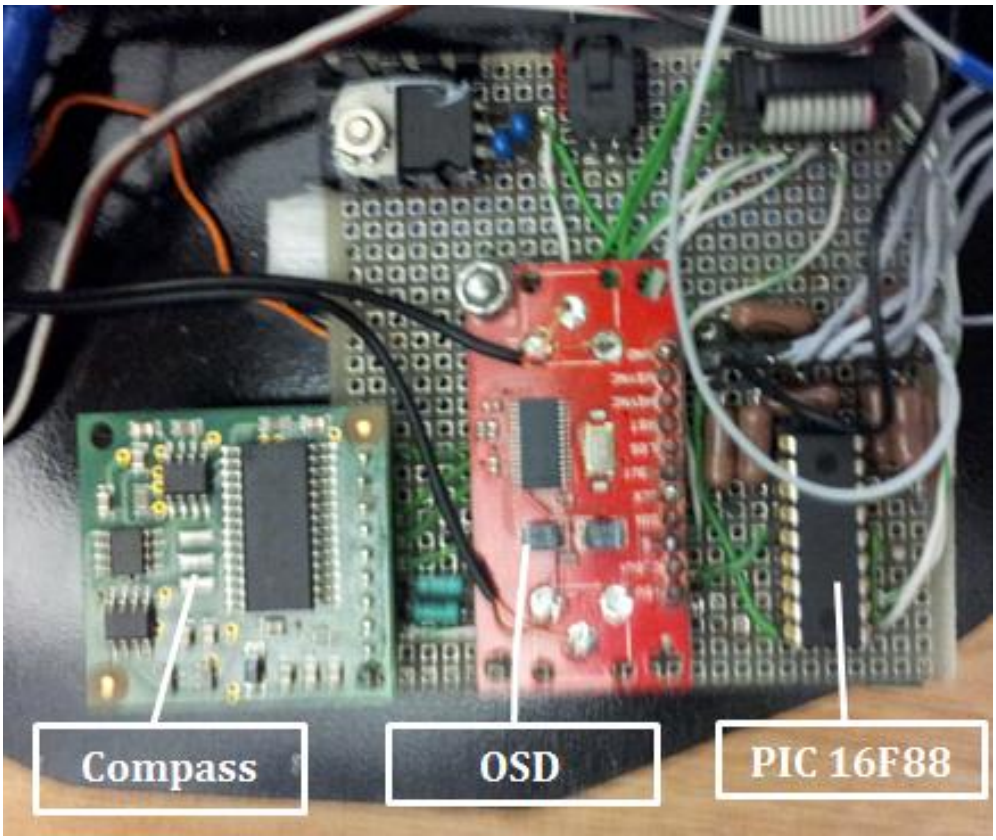


Figure 4: Breadboard Layout

The joystick has been modified with an Arduino FIO microcontroller and XBEE module. The functional diagram for the joystick can be viewed on Figure 5.

Programming for the Axon and Laser Tag system was done in C++. All the annotated programs and flowcharts of their operation can be found in the appendix.

The MAX7456 OSD chip comes with 256 character pre-programmed into memory (the default character map is in the appendix). In order to display our custom reticles, it was necessary to program our own characters in the OSD's memory. This was accomplished by writing a program for the AXON to re-program some of the Chinese characters (we sure weren't going to use them) on the OSD to our custom characters (see the appendix). To program an individual character, a section of code that writes the individual pixels (see appendix for example section of code and the pixel map of an individual character) was inserted into the code, the code was compiled and run once. The result is the character gets stored in the non-volatile ROM on the OSD chip. The characters could then be used just like any of the standard characters.

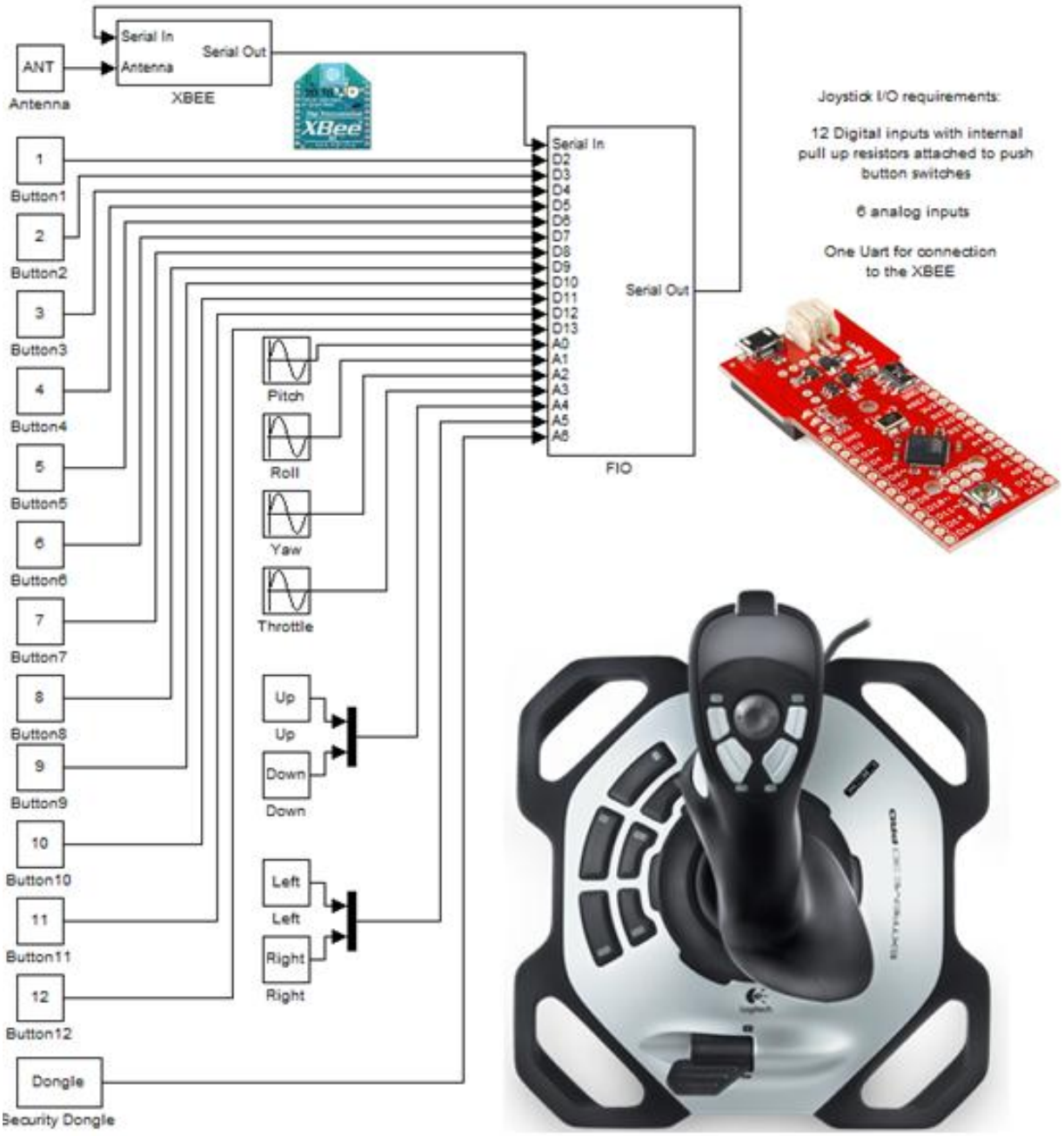


Figure 5: Functional Diagram of the Joystick

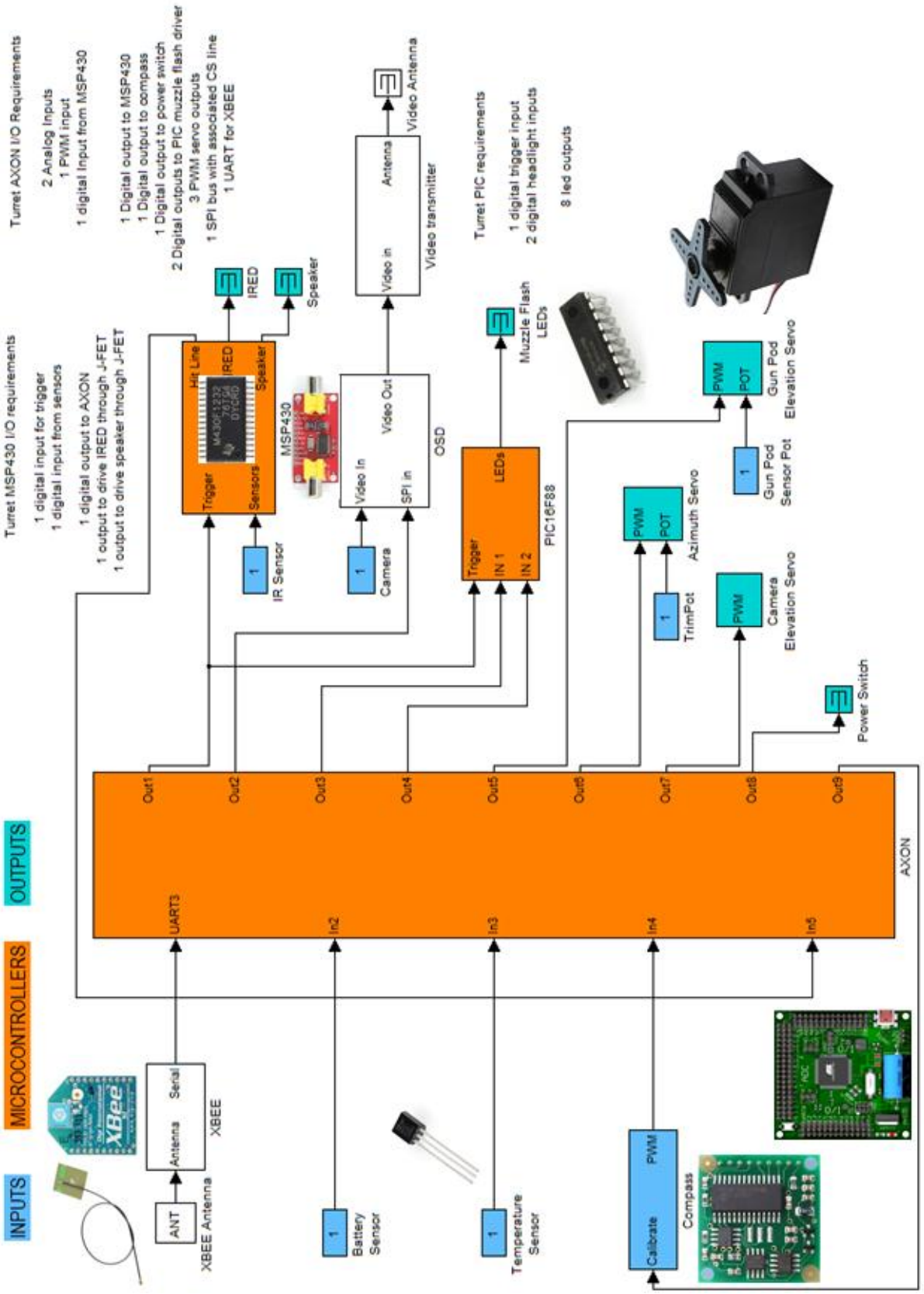


Figure 6: The Axon Microcontroller Functional Diagram

DESIGN EVALUATION

Design Evaluation: Briefly describe the success of the device in meeting the functional element categories, and provide justifications for any anticipated grading adjustments.

A. **Output Display**

- LEDs driven by a PIC16F88 microcontroller.
- Breakout Board for MAX7456 on screen display. For text based overlay onto live video feed from turret. This includes custom programed reticles as well as overlay of compass heading, and battery life. An overlay is also used as warning messages about battery life and temperature.

B. **Audio Output Device**

- Speaker with software-generated sound effects.
- Higher volume through MOSFET amplification.

C. **Manual Data Input**

- Magnetic read switch for turning on the turret.
- Buttons on joystick for selecting reticles and lighting options.
- Potentiometer on joystick for speed control (the pot provides a scaling factor for the joystick inputs to allow better user control).
- Joystick control for directional change (left/right and up/down control).
- Switch for turning off power board.
- Turret will power off through programing using a button on the joystick.

D. **Automatic Sensor Input**

- Potentiometer on gun pod provides gun turret position information which is fed to the driving servo.
- A temperature sensor will initiate automatic shutdown when the device overheats.
- Using a magnetic compass, the heading is automatically displayed in real time on the FPV video overlay.
- Voltage monitoring can be done using a voltage divider circuit; automatic shutdown is initiated when the battery voltage drops below the critical value.

E. **Actuators, Mechanisms, and Hardware**

- Continuous rotation servo with potentiometer feedback for the gun pod.
- Continuous rotation servo for the base rotation.
- 1 RC servo motor controlling the camera pod.
- CNC Water-jet cutting machine for base components courtesy of the Colorado State University Engines and Energy Conversions Laboratory (CAD work and G-code as well as operation done by us).
- 3D printing for gun and camera pods courtesy of a local printer (CAD work done by us).
- Gear drive for gun pod and azimuth drive.

- CNC milled mold for custom, hand-laid fiberglass shell (All CAD machining and laying work done by us).
- Fat Shark FPV goggles courtesy of our very own, class confirmed, critically acclaimed, Intelligent *Equus africanus asinus*, (Smart A** for short) Josh Misek (for the record only the smart part of that statement is accurate).
- Parts salvaged from broken projects and appliances for enhanced functionality on a budget. For example using old carbon fiber arrow for laser gun barrel, all servos used are salvaged, the Axon microcontroller is salvaged, all batteries were also salvaged, proto-boards were salvaged from scraps, the potentiometer was salvaged, and many other small circuit components and connectors that were used was salvaged from various locations.

F. **Logic, Processing, and Control; AND Miscellaneous (functional elements not covered in the categories above)**

- Programmed logic: PIC, Axon, MSP 430, and Arduino FIO
    - PIC logic controls the function of 8 LED lights in the gun pod. This allows the LEDs to flash in different patterns based on other functions of the turret.
    - MSP 430 logic fires the laser tag IR LED as well as generates the sound of the shot. It also can record the IR sensor pulse train and determines its validity.
    - The Arduino FIO logic records and transmits joystick inputs via a wireless XBEE module to the turret.
    - Axon logic controls the entire function of the turret. This includes processing inputs from the joystick and triggers the function in the, Pic and MSP 430. The Axon also tracks all of the automatic input sensors.

- User selectable functions. Reticules and lighting options are accessed via programing based on the button inputs received from the joystick.
- Advanced and/or multiple interfaced microcontrollers: PIC, Axon, MSP 430, and Arduino FIO all communicate within the system.
- Wireless turret control transmission from joystick using the XBEE link.
- Wireless video transmission with programed graphical overlay.
- Serial addressing for data sent from joystick to turret. The low nibble is data, the high nibble is address.

- **XBEE**

1mw UFL Connector

(Sparkfun-8666-$22.95x2)

This module allows very reliable and simple communication between microcontrollers, computers, systems, really anything with a serial port! Point to point and multi-point networks are supported.

Explorer Dongle

(Sparkfun-9818-$24.95)

With the XBee Explorer dongle we plug the unit directly into your USB port. No cables needed! This unit works with all XBee modules including the Series 1 and Series 2.5, standard and Pro version. The on-board voltage regulator is good up to 500mA.

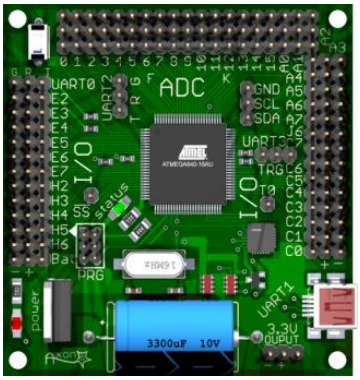
- **Arduino Fio**

Arduino Fio

(Sparkfun-11016-$24.95)

By using Fio, We can interface to sensors and actuators, in our case, we used because of the XBEE socket interface.

- **Axon microcontroller**



Axon

(Society of Robots-$89)

The best microcontroller on the market, with no external power buses required to plug servos or sensor.

- **Fat Shark Full color 640x480 FPV Video Goggles**

The Predator

(Fat Shark-$249.95)

The predator combines a headset, Camera, and a Transmitter all in one.

- Breakout Board for Max 7456 On Screen Display

OSD

(Spark fun-9168-$39.95)

The board is set up with all supporting circuitry and RCA connectors to allow easily interrupt and overlay text and/or graphics onto a video signal (PAL or NTSC).

- Temperature Sensor

TMP36

(Sparkfun-10988-$1.5)

The TMP36 is a low voltage, precision centigrade temperature sensor. It provides a voltage output that is linearly proportional to the Celsius temperature. It also doesn't require any external calibration to provide typical accuracies of ±1°C at +25°C and ±2°C, over the −40°C to +125°C temperature range.

- Antenna

2.4 GHz Antenna-Adhesive (U.FL Connector)

(Sparkfun-11320-$4.95)

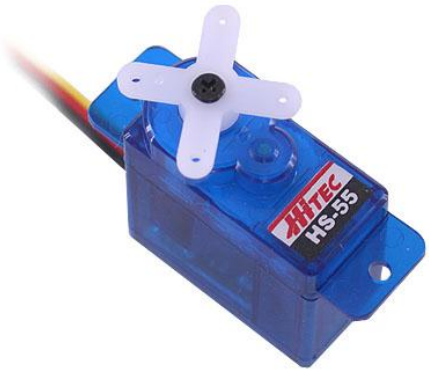
The adhesive, omni-directional antenna is perfect for situations when you're trying to get a signal out from a transmitter that you've jammed down into an enclosure. Operating on the 2.4GHz band, it works great for WiFi, Bluetooth or ZigBee. Just connect the antenna using a U.FL connector and then peel and stick!

- Servo Motors

HSR-5995TG-Ultra Torque Robot Servo

(Servo Database.com-35995s-$114.99)

This servo is a coreless Titanium gear with dual bearing support, 2.17 oz. of weight, up to 416.6 oz./in of torque and 0.12 sec/60° of speed.



HS-55 Sub-Micro Servo

(Servo City-31055s-$9.99)

This servo is a coreless, direct drive nylon gear, with 0.28 oz. of weight, up to 18 oz./in of torque and 0.14 sec/60° of speed.

Lessons Learned: Laser Turret; Team 31		
Problem	**Solution**	**Lesson Learned**
Could not get wireless XBEE conection to work between the joystick and the turret. The correct serial information was leaving the joystick but the axon was unable to receive all of the data.	In our code we had the joystick continuously transmitting each subsequent data set. Adding a 5 ms delay between each transmission solved the problem.	Always allow enough time for components to make and receive a transmission.
Burned the laser diode by over powering it.	Bought a new laser to replace the broken one.	Always check the voltage ratings of the components before
Blew up MOSFET Transistor by plugging the wire connector in backwards.	The MOSFET Transister was replaced by a new one.	In order to prevent other things from getting plugged in backward key all connectors to fit in only one way.
When using spark fun's Breakout Board for MAX7456 On Screen Display, the user can set their own character maps. If these maps are not set up properly the first time, they proved very difficult to proof read.	We proof read all code and made sure that it is correct. This solved our problem.	Proof read all code and make sure that it is correct before flashing it to the microcontroller.
Grainy video feed that flickered on and off without explanation.	We found a faulty connection between the camera and the breakout board. Once this connection was repaired the feed	Check all connections for crisp continuity.
We could not drive all of the LED's on the turret at once.	We used a PIC16F88 to source current to the LED bank, We found that the PIC microcontrollers are much better at sinking current the sourcing it.	When running higher demand components from a PIC use reverse logic to sink power through the switch rather the sourcing it from the
We accidentally used an oil based spray paint which had a set time of 5 hours. This greatly slowed down the work schedule.	We should have checked the time to dry.	Make sure you check product labels for specifications before using.
We accidentally grounded a power supply to a metal component on the turret. This caused a short in the axon microcontroller.	We taped the metal with electrical tape in order to fixed the problem.	Always check to ensure that no part of the circuit is in contact with a conductive component or other parts of the circuit.
The camera feed was glitching and produced a grainy/flickering video output.	By replacing the wire with shielded wire to reduce EMF interferance the video feed was fixed.	Always use shielded wire on complex analog signal transmission lines.

As with any major undertaking, we as a group learned a myriad of important life lessons through undergoing the completion of the mechatronics project. Our group dynamic was fantastic throughout the project; this is one of the major factors contributing to our success in this project. We began tentatively working on the project over the summer, meeting weekly and brainstorming ideas as well as researching what was possible and what was affordable. We found that our ability to produce good and exciting ideas for the project far outpaced our ability to execute them. Once we settled on a main concept things went smoothly within the group even when individual components of the project were not coming along as we expected. We were able to cooperate in a way which allowed the skills of each member of the group to complement the others adding to a very well rounded and motivated problem solving team. This leads me to the first lesson learned through the project, good teamwork trumps all else when it comes to large projects.

During the idea generation phase of the project we settled on the idea of the turret relatively early. However, the exact function of the turret was more difficult to decide on. Originally, we planned to make the turret an airsoft turret. This would have required considerable mechanical manufacturing. Because we were not graded on our ability to manufacture very small gears and other parts, we decided early on to make the switch to a laser tag turret. This switch worked very well because Josh Misek had already made a laser tag set with his brothers a few years ago, so we already had some infrastructure to use the laser tag turret with. This leads us to lesson number two, make changes as soon as possible. By switching over we saved ourselves a lot of money and possibly hundreds of hours of manufacturing.

Our project progressed relatively smoothly until we tried to switch from a cabled connection to a wireless connection via XBEEs for the joystick. We found that the main microcontroller was only receiving a sporadic set of serial data sent to it, even though we could confirm that the joystick was outputting the correct data to its XBEE. After several days of working on the problem, we decided to try and add a delay between the data packets: this solved the problem. What was happening was that the Joystick would send out a packet of data containing the information about each button and the position of the stick. Once a transmission was complete it would immediately begin the next transmission. We found that the UART could not handle the data as fast as the joystick was sending it out. When we added the delay it gave the UART a chance to keep up with all of the packets sent. There were several other times where we made the error of not leaving enough space or time to complete the task. For example, we should have left more room in the poduals for wires, and we should have left more room in the turret for the circuitry. This rounds out the third lesson from this project, always give yourself more room and time then you think you will need.

On day of the early bird turn in, we had gathered to make our presentation, in the process of setting up the cable providing a connection from the Axion to the breadboard was accidently plugged in backwards. When the power switch was thrown a transistor was blown up, this subsequently let out some of the blue smoke that Dr. Dave was always warning us about and sent the team into a frenzy of trouble shooting to diagnose the problems that may have resulted. Thankfully we found that the transistor was the only component which was damaged in the mistake and we were able to finish the presentation by powering the project via a power supply. This leads to the fourth lesson of the project, double check all of the connections before flipping the on switch.

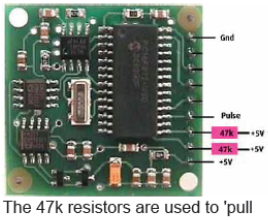
In the process of making the turret we made a few manufacturing mistakes such as drilling the mounting holes for the supports in the wrong place. These mistakes cost us precious time and energy as well as resources. At first the team decided to use drywall screws to attach the supports

to the base instead of stronger machine screws that are removable. When trying to remove the drywall screws the head of the screw would shear off leaving behind the rest of the screw. By not choosing a removable fastener in the first place, each mistake cost three times the amount of time that it took to go back and repair. We found that it would have been much better to slow down and do it right the first time as opposed to rushing a head and hoping the problems would fix themselves.

APPENDIX

Table 1: Group 31's Bill of Materials for MECH307 Project.

Bill Of Materials					
Item	**Quantity**	**Technical Name**	**Price**	**Total**	**Actual spending**
1	1	Logitec Extreme 3D Pro Gaming Joystick	$25.99	$30.98	$30.98
3	2	XBEE 1mw UFL connector	$22.95	$45.90	$45.90
5	1	MAX7456	$39.95	$39.95	$39.95
7	1	Arduino Fio	$24.95	$24.95	$24.95
8	1	HS-55 micro servo	$18.00	$18.00	$0.00
9	1	TMP 36 temp sensor	$1.50	$1.50	$1.50
10	1	2.4 Ghz antenna	$4.95	$4.95	$4.95
12	1	Axon microcontroller	$89.00	$89.00	$0.00
13	1	FatShark Full Color 640x480 FPV Video Goggles	$249.95	$249.95	$0.00
14	1	Lazy Susan Pivot Platform 6 inch	$5.99	$5.99	$5.99
15	1	Printed ABS Gun pod	$0.00	$0.00	$0.00
16	2	2 cell 3300 mAh lipo	$17.00	$34.00	$0.00
19	1	Aluminum Base Plate (.25 x 12 x 12) & sheet	$20.74	$20.74	$20.74
20	1	Steel for Legs/Base unit/Servo support Bracket	$4.00	$4.00	$4.00
22	1	Printed Laser / Camera Mount	$0.00	$0.00	$0.00
25	2	Steel Ball Bearing, 5/8" Shaft , 1-3/8" OD	$8.81	$17.62	$0.00
25	2	Steel Ball Bearing, 1/2" Shaft , 1-3/8" OD	$8.81	$17.62	$17.62
28	1	3/8 acetal	$26.84	$26.84	$26.84
29	2	HSR-5995TG - high torque servo	$115.00	$230.00	$0.00
30	1	Magnetic Switch	$6.00	$6.00	$6.00
Total:				**$867.99**	**$229.42**

Device		Variable	Image	Settings
Devantech CMPS03 (1 Pin) Compass		compass	 The 47k resistors are used to 'pull up' the unused I2C bus lines.	97 Pulse Input: F0 Supplies: 2 2
Digital Output		compassCalibrate		96 I/O Pin: F1 Start High?: true Supplies: 3
Digital Input		LTHit		16 I/O Pin: H4 Use internal pullup resistor?: true Supplies: 4
SPI Bus		spiBus		21 MOSI pin: MOSI 22 MISO pin: MISO 20 SCK pin: SCK Master Mode: true
	Generic SPI Device	OSD		SPI Mode: SPI_MODE_0 Bit Order: SPI_DATA_ORDER_MSB 93 Select Pin: F4 Filler Byte: 0xff Supplies: 6
Analogue Input		temp1		87 ADC Channel: K2 - ADC10 Supplies: 7
Servo Driver (Hardware PWM)		servoBank		
	Servo	azimuth		Reverse direction?: false 5 PWM pin: E3 - OC3A Center: 1500 Range: 500 Supplies: 9 9
	Servo	gunPod		Reverse direction?: false 6 PWM pin: E4 - OC3B Center: 1500 Range: 500 Supplies: 10 10
	Servo	cameraPod		Reverse direction?: false 7 PWM pin: E5 - OC3C Center: 1500 Range: 500 Supplies: 11 11
UART		XBEE		63 UART Number Rx: RXD3 64 UART Number Tx: TXD3 Supplies: 12
Analogue Input		battSense		82 ADC Channel: K7 - ADC15 Supplies: 13
Digital Output		power		75 I/O Pin: A3 Start High?: true Supplies: 14
Digital Input		compassCalibrating		95 I/O Pin: F2 Use internal pullup resistor?: false Supplies: 15

Device		Variable	Image	Settings
Digital Output		PIC1		92 I/O Pin: F5 Start High?: false Supplies: 16
Digital Output		PIC2		94 I/O Pin: F3 Start High?: false Supplies: 17
Analogue Output - PWM		LTfire		15 I/O Pin: H3 - OC4A Frequency (Hz): 15 Initial Duty Cycle (0 - 100): 0 Actual Hz: null

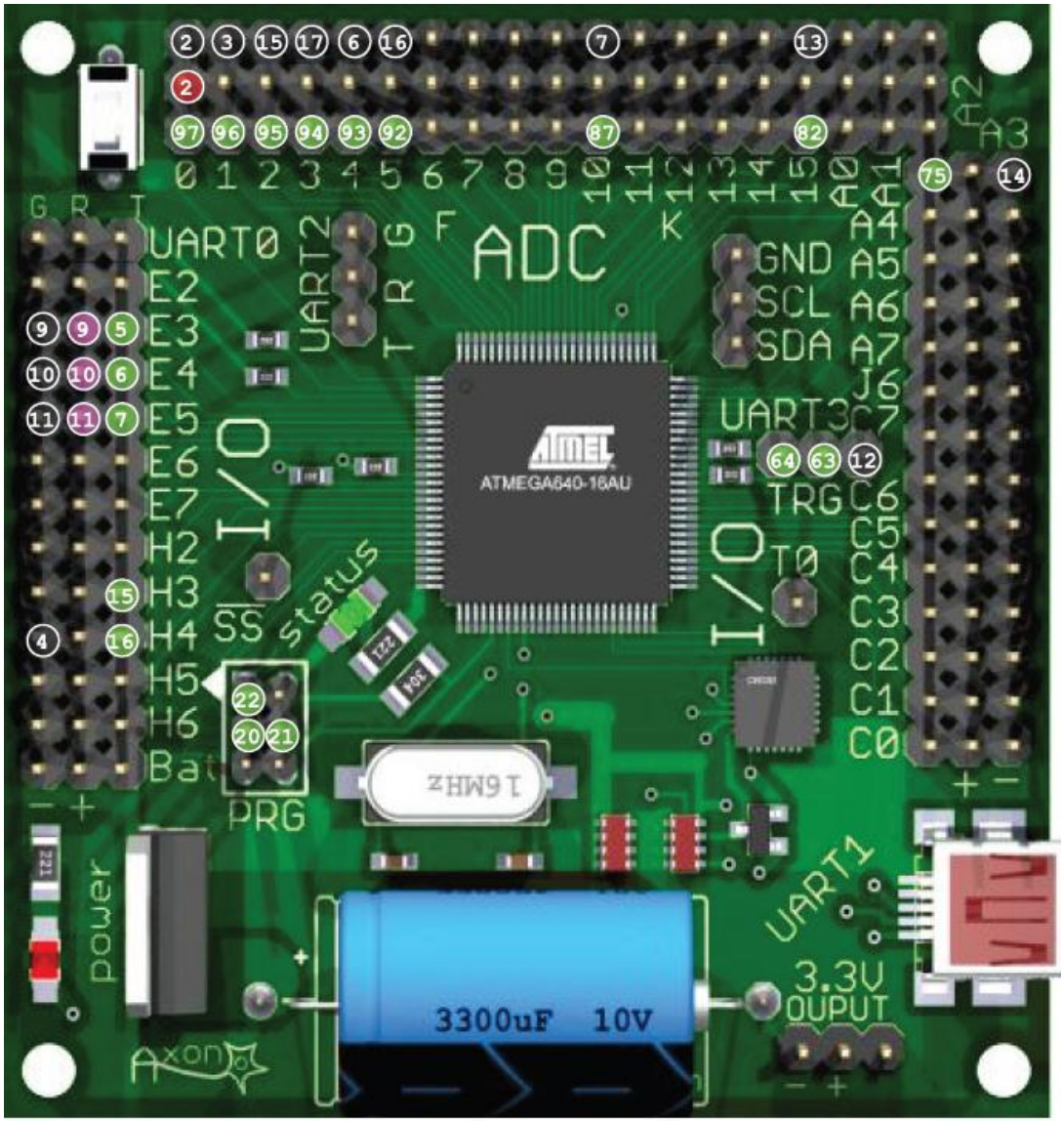


Figure 7: Axon Microcontroller Pin Diagram for Functional Table.

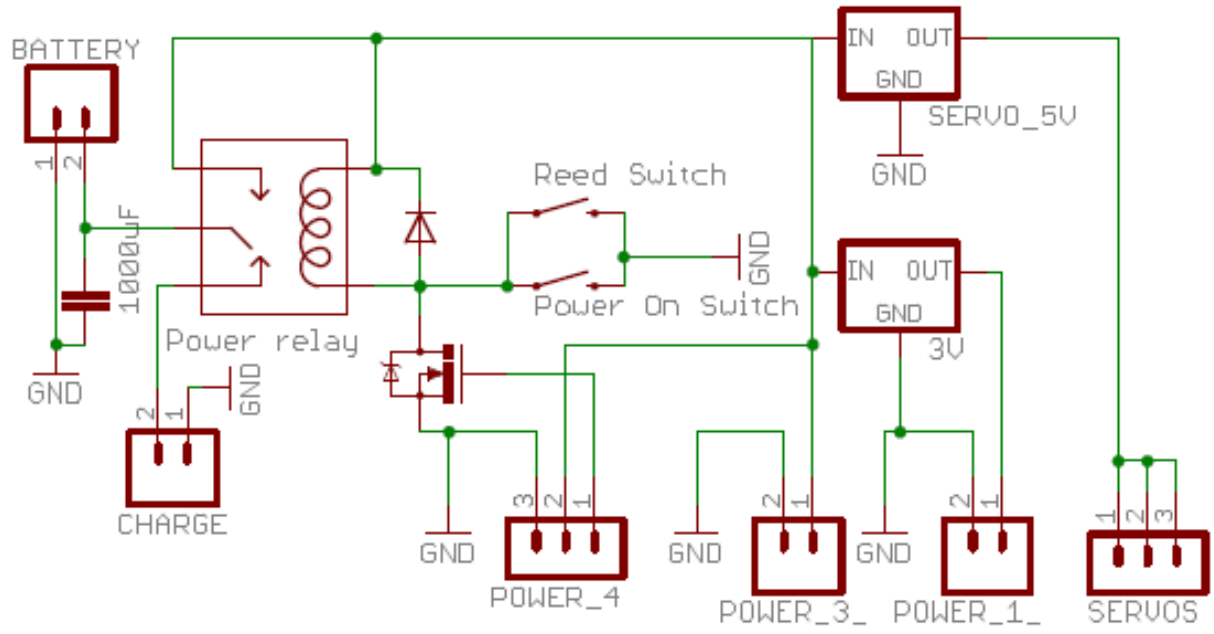


Figure 8: Power Board Wiring Diagram

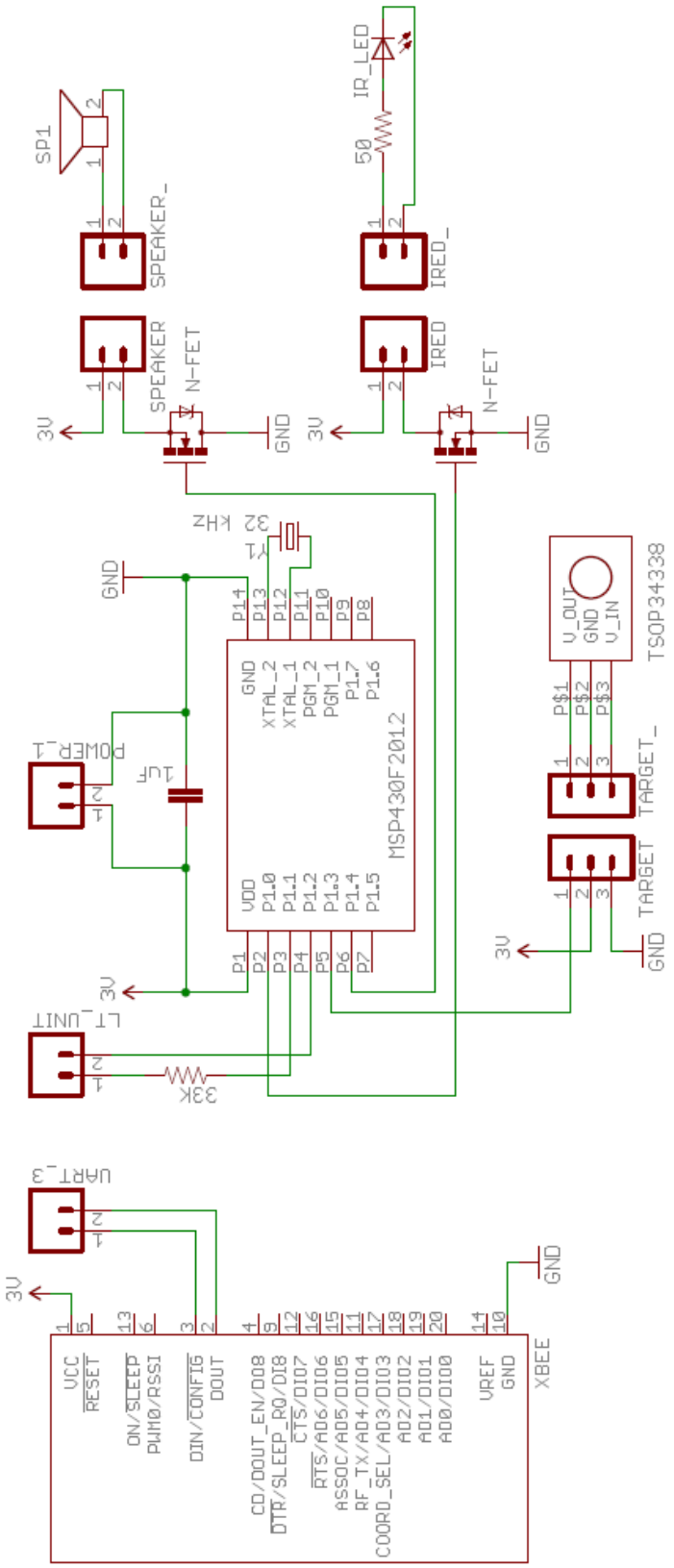


Figure 9: Laser Tag System Wiring Diagram.

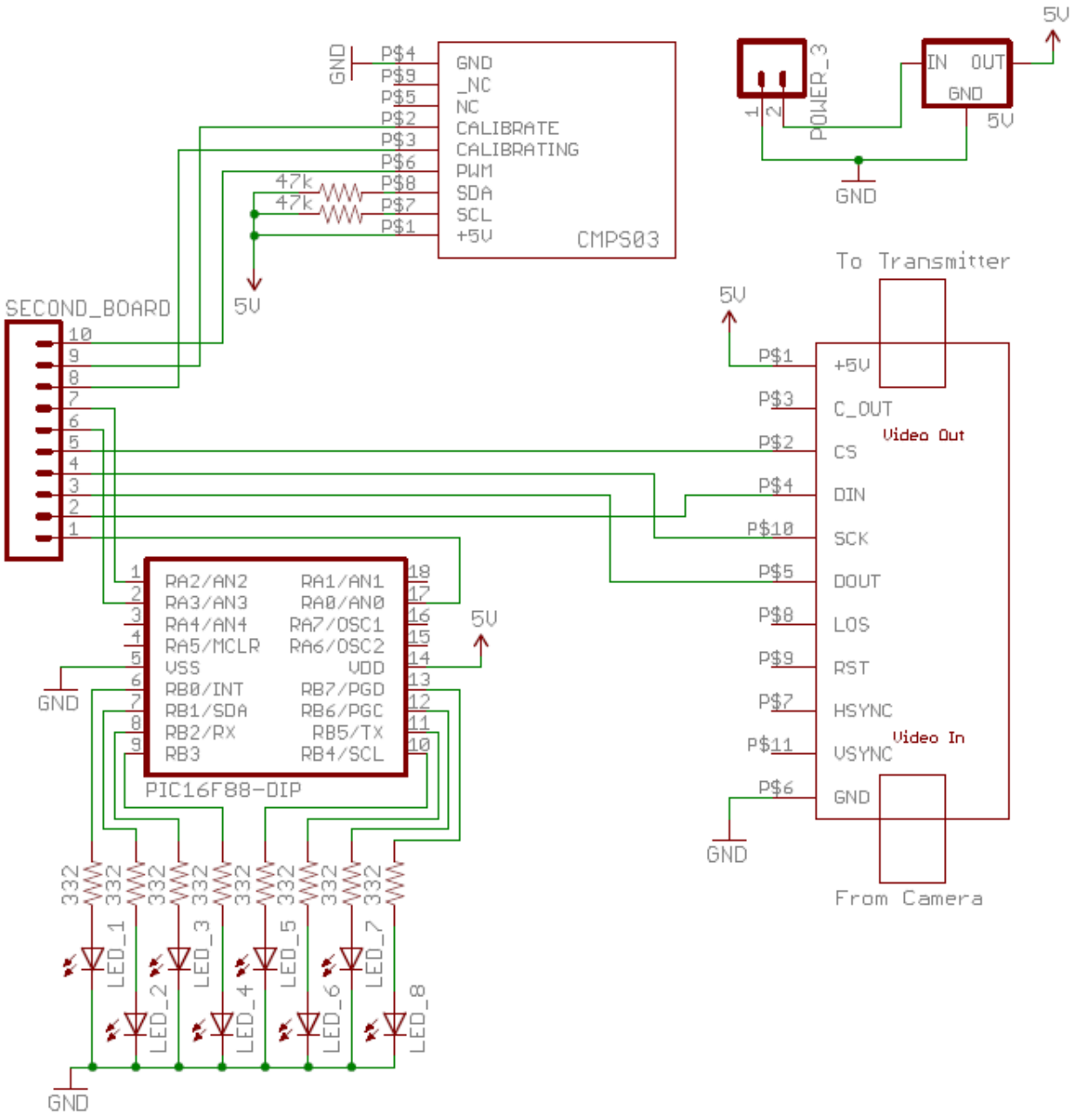


Figure 10: Wiring Diagrams for the PIC, Compass, and MAX7456.

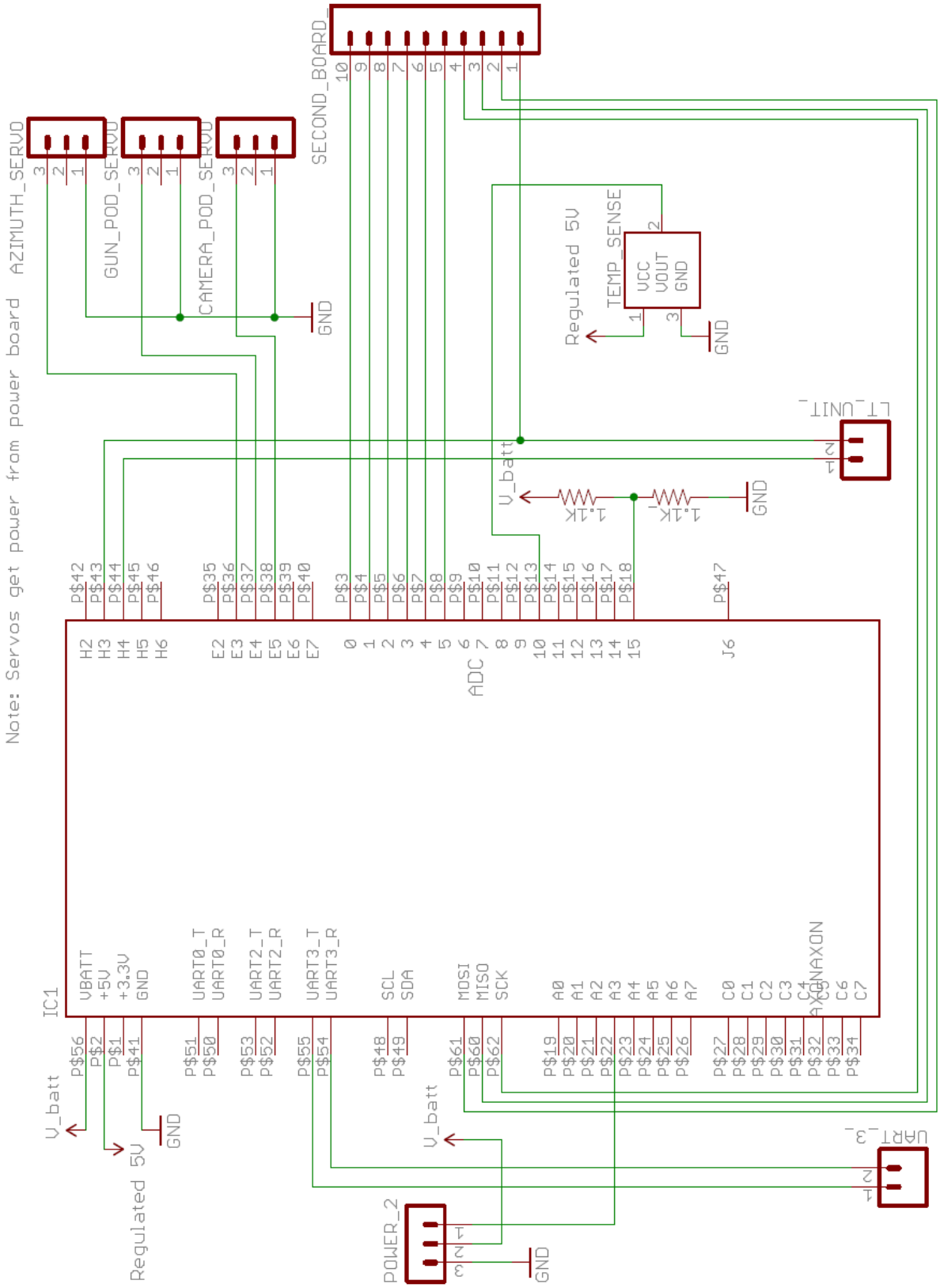


Figure 11: Axon Microcontroller Wiring Diagram

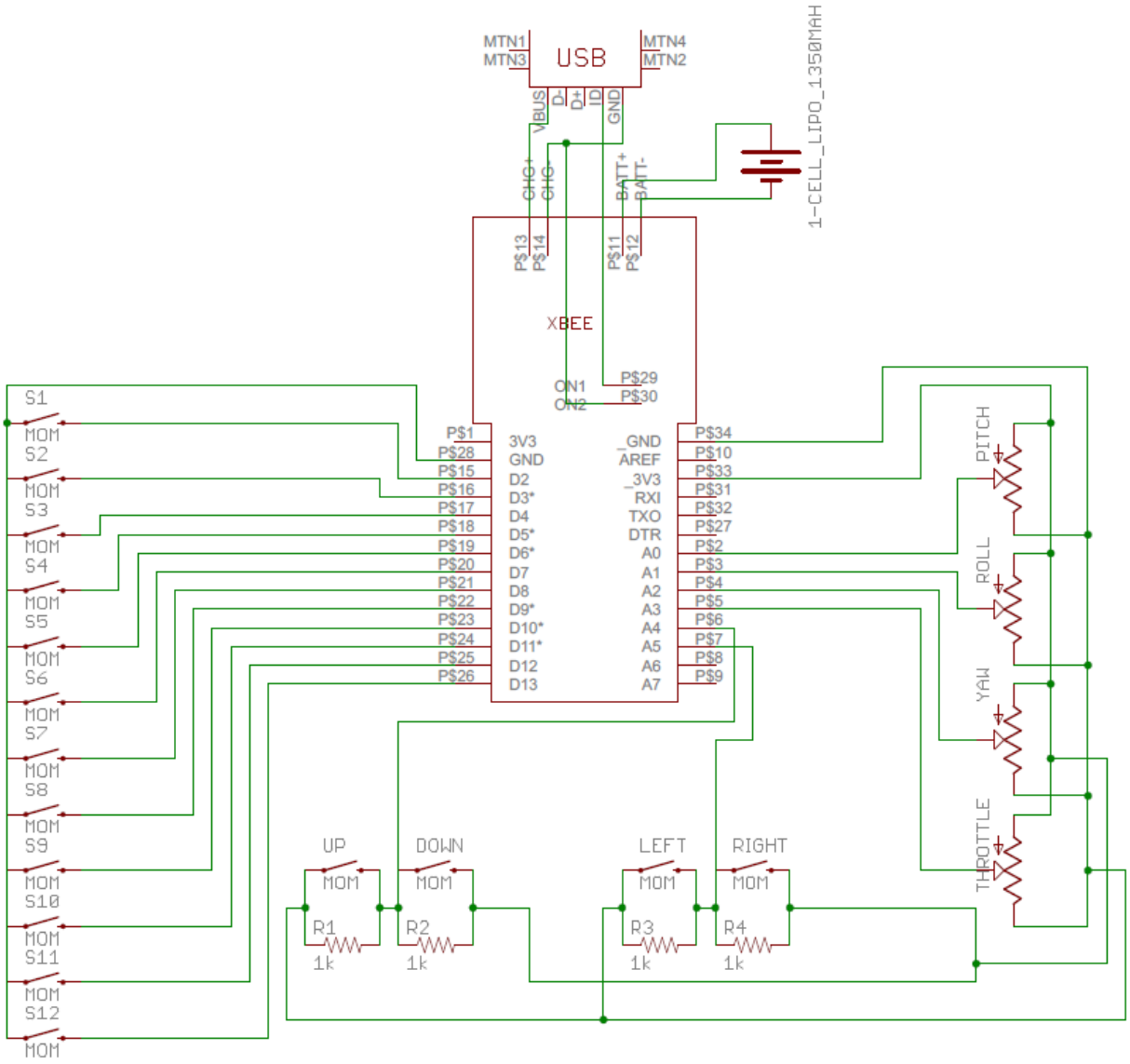
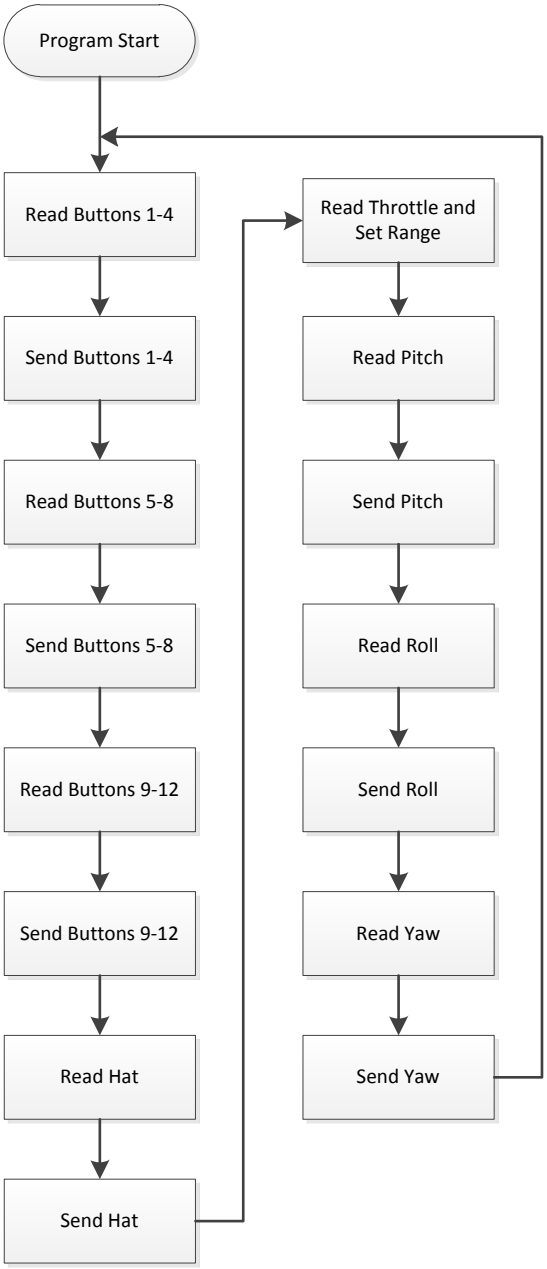
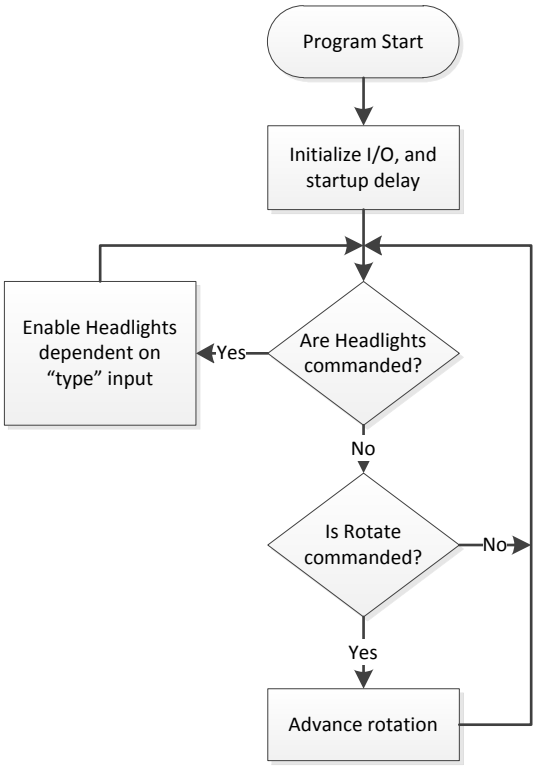


Figure 12: Wiring Diagram for the Joystick.

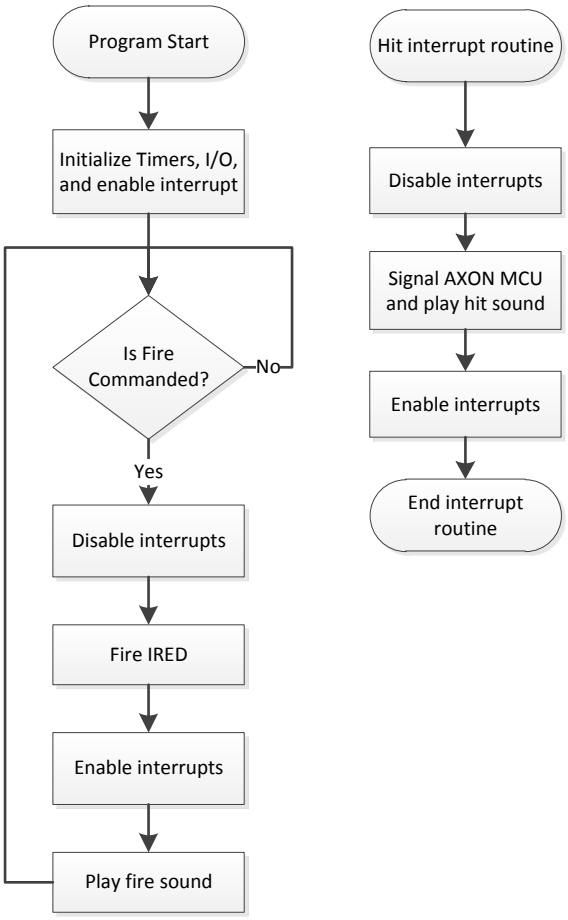
Joystick FIO MCU



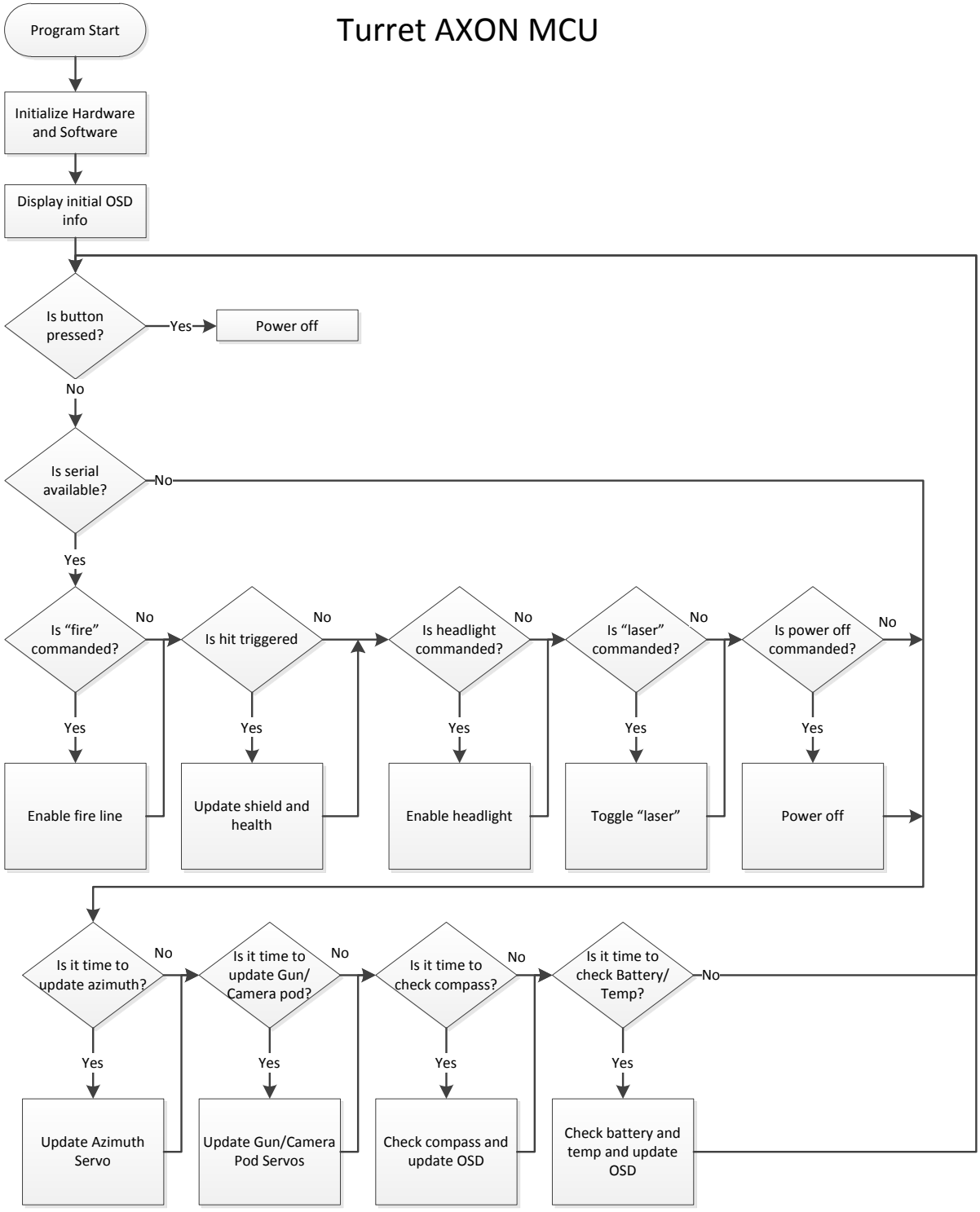
Turret PIC MCU



Turret MSP430-F-2012 MCU



Turret AXON MCU



```
/*******************************************************************************
************************Main function for tagger unit***************************
Port 1:
0: IRED (IRED) [non-negotiable, depends on hardware of P1.0]
1: Trigger input (TRIG) [active high]
2: Hit output signal (HIT)
3: Target input      (TARG)
4: Speaker output (SPKR)
5: Not Used
6: Not Used
7: Not Used

Port 2:
Not used

(PARENTHETICAL) tags are labels used for pins, see below.

*********************************Program flow**********************************
Software loop to poll for fire command
Fire: fires once per trigger pulse with delay for max rate
Hit interrupt: fast compute then calls output update and sound so that
               it is ready if it gets interrupted again
*******************************************************************************
*****************************End block comment*********************************/

#include <msp430x20x2.h>

//------------------------------Pin definitions--------------------------------
#define IRED 0x01
#define HIT  0x04
#define TRIG 0x02
#define TARG 0x08
#define SPKR 0x10

//-----------------------------------Macros--------------------------------------

int DT;
#define Delay(x); TAR = 0; DT = TAR; DT += x; while(TAR != DT);      //delay function

#define whileON while(!(P1IN&TARG) && TAR<BIT_1+DIFF+1)       //while target line is high
#define whileOFF while((P1IN&TARG) && TAR<BIT_P+DIFF)

int RT;
#define recvBit(x); TAR = 0; RT = TAR; whileON; x = TAR - RT; //times how long a single
                                                              //high pulse is

#define IR_ON P1DIR |= IRED       //IR led "on" (32 kHz signal)
#define IR_OFF P1DIR &= ~IRED     //IR led off
//---------------------------------Constants-------------------------------------
#define RATE 50      //maximum rate of fire delay
#define GUN_ID 2     //bit pattern to send (tuned to provide best reliability over the
                     //widest conditions)

#define BIT_0 10     //lengths of delays
#define BIT_1 20
```

```
#define BIT_P 10
#define BIT_S 30
#define DIFF 5

#define FIRE 1
#define HITS 2

//-----------------------------------Globals-------------------------------------
char byte;
char valid;
char mode;

//---------------------------------Prototypes------------------------------------
void fire();
char recv();
void send(char);
void sound(char);

//-------------------------------------Main--------------------------------------
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer

  P1DIR = 0xFF;             //All output
  P1DIR &= ~(TRIG|TARG);    //set inputs
  P1REN |= TRIG;            //set resistors
  P1OUT = 0x00;             // turn off P1

  P1OUT |= TARG;            //set target resistor
  P1REN |= TARG;

  P1SEL |= IRED;            //set IR output to ACLK
  P1DIR &= ~IRED;           //turn off IR output
  TACTL = TASSEL_1 + MC_2;  //ACLK, cont mode

  Delay(10);                //start-up delay

  P1IE |= TARG;             // P1.4 interrupt enable
  P1IES &= ~TARG;           // set P1.4 to Hi/Lo edge
  P1IFG &= ~TARG;           // clear P1.4 IFG (interrupt flag)

  _BIS_SR(GIE);             // set CPU to receive interrupts

  while(1)
  {
    if(P1IN & TRIG)         //if trigger is high, fire
      fire();
  }
}
//-------------------------------Hit routine-------------------------------------
#pragma vector=PORT1_VECTOR
__interrupt void Port_1()
{
  IR_OFF;                   //turn off IR
  valid = 1;

  TAR = 0;
```

```
  while(!P1IN & TARG);
  byte = recv();            //receive packet

  if(byte != 2)             //check validity
    valid = 0;

  if(valid)
  {
    P1OUT |= HIT;           //tell AXON
    sound(HITS);            //play sound
    Delay(100)
    P1OUT &= ~HIT;
  }
  P1IFG &= ~TARG;           //clear interrupt flag

}
//-------------------------------------recv()-------------------------------------
char recv()
{
  int t;
  char rb = 0;

  for(int i=0; i<2; i++)    //received pulse must conform to bit length pattern
  {
    rb = rb<<1;
    TAR = 0;
    whileOFF;
    recvBit(t);
    if((t > BIT_1-DIFF) && (t < BIT_1+DIFF))
    {
      rb |= 0x01;
    }
    else if((t > BIT_0-DIFF) && (t < BIT_0+DIFF))
      rb |= 0x00;
    else
      valid = 0;
  }
  return rb;
}
//---------------------------------send(char)---------------------------------
void send(char b)
{
  for(int i=0; i<2; i++)    //send bit pattern
  {
    if(b & 0x02)
    {
      IR_ON;
      Delay(BIT_1);
      IR_OFF;
    }
    else
    {
      IR_ON;
      Delay(BIT_0);
      IR_OFF;
    }
    b = b << 1;
    Delay(BIT_P);
```

```
  }
}

//-------------------------------------fire()-----------------------------------
void fire()   //sends one shot
{

    P1IE &= ~TARG;
    TAR = 0;
    IR_ON;
    Delay(BIT_S);
    IR_OFF;
    Delay(BIT_P);

    send(GUN_ID);

    P1IE |= TARG;

    sound(FIRE);

    while(P1IN & TRIG);     //waits until trigger signal goes low
      Delay(RATE);          //pause for any king of bounce, as well as max rate
}

//-------------------------------------sound()----------------------------------
unsigned long int wait;
void sound(char sNum)       //generates a sound
{
  if(sNum == HITS)          //sound when hit, a "beep"
  {
    int t;
    for(int j=0; j<65; j++)
    {
      P1OUT |= SPKR;
      t = TAR;
      while(TAR-t < 10);
      P1OUT &= ~SPKR;
      t = TAR;
      while(TAR-t < 10);
    }
  }
  else if(sNum == FIRE)     //fire sound, a rapidly increasing frequency, works well with
                            //these speakers
  {
    int d = 1;
    int t;
    for(int j=0; j<12; j++)
    {
      P1OUT |= SPKR;
      t = TAR;
      while(TAR-t < d);
      P1OUT &= ~SPKR;
      t = TAR;
      while(TAR-t < d);
      d += 7;
    }
  }
}
```

TURRET PROGRAM

```
#include "hardware.h"              //Webbot's hardware code

#define RET_START          192     //starting point for the reticule display
#define RET_CAT            0x80    //cat reticule first character
#define RET_DIAMOND        0x70    //diamond reticule first character
#define RET_SQUARE         0x60    //square reticule first character
#define COMP_CENTER        44      //center of compass display
#define BATT_START         130     //start of battery display
#define LINE_1_B_START     97      //start of battery warning line 1 display
#define LINE_1_T_START     95      //start of temperature warning line 1 display
#define LINE_2_START 126           //start of warning line 2 display
#define DAMAGE_START 26            //start of damage display


char pitchT;          //variable for constructing pitch value in from serial data
char pitch;           //pitch value from joystick
char rollT;           //variable for constructing roll value in from serial data
char roll;            //roll value from joystick
//char yawT;          //variable for constructing yaw value in from serial data
//char yaw;           //roll value from joystick
char lightMode;       //mode that headlights are in
char lightMode2;      //variable to wait until button is released before changing
                      //lightMode again
char retMode;         //mode that reticule is in
char retMode2;        //variable to wait until button is released before changing retMode
                      //again
char calMode;         //variable for calibrating compass
char temp;            //character received from serial
char battMode;        //level of the battery
char shields;         //"shields" level
char hull;            //"hull" level

int cameraPos;        //position of the camera/gun pod elevation
int gunPos;           //position of the gun pod (offset to line pods up)
int battLevel;        //raw reading from the battery meter

const int line1B[16] = {0x0c, 0x25, 0x38, 0x38, 0x29, 0x36, 0x3d, 0x00, 0x0d, 0x36, 0x2d,
0x38, 0x2d, 0x27, 0x25, 0x30};     //battery warning line 1

const int line1T[20] = {0x1e, 0x29, 0x31, 0x34, 0x29, 0x36, 0x25, 0x38, 0x39, 0x36, 0x29,
0x00, 0x0d, 0x36, 0x2d, 0x38, 0x2d, 0x27, 0x25, 0x30}; //temperature warning line 1

const int line2[18]  = {0x1d, 0x2c, 0x39, 0x38, 0x28, 0x33, 0x3b, 0x32, 0x00, 0x13, 0x32,
0x2d, 0x38, 0x2d, 0x25, 0x38, 0x29, 0x28};      //warning line 2


TICK_COUNT pitchTime;       //time that the pitch servo update was last called
TICK_COUNT rollTime;        //time that the roll servo update was last called
TICK_COUNT compassTime;     //time that the compass update was last called
TICK_COUNT battTime;        //time that the battery/temperature update was last called
TICK_COUNT shieldTime;      //time that last shield increase was called
TICK_COUNT hitTime;         //delay so that one hit only counts as one hit
```

```
COMPASS_TYPE bearing;       //compass bearing


bool pitchLow;       //true if the low byte of the pitch variable has been received
bool pitchNew;       //true if both bytes of the pitch variable have been received
bool rollLow;        //true if the low byte of the roll variable has been received
bool rollNew;        //true if both bytes of the roll variable have been received
//bool yawLow;       //true if the low byte of the yaw variable has been received
//bool yawNew;       //true if both bytes of the yaw variable have been received


int constrain(int val, int low, int high);      //constrains the val to low and high
char read(char Address);                        //reads address from OSD
char write(char Address, char Data);            //writes data to address on OSD
void displayReticule(int charStart, int dispStart);    //displays the reticule on the
                                                       //screen
void displayNumber(int numCenter, int number, bool brackets); //displays a number on the
                                                               //screen at numcenter
                                                               //possibly surrounded by
                                                               //brackets
void checkBatt();                  //checks the battery level
void displayBatt(int bm);          //displays the battery level
void displayWarning(bool batt);    //displays either the temperature or
                                   //battery level warning
void displayDamage();              //displays the shield and hull levels
void shutdown();                   //shuts the turret down


// Initialise the hardware
void appInitHardware(void) {
      initHardware();
}
// Initialise the software
TICK_COUNT appInitSoftware(TICK_COUNT loopStart)
{
      roll = 127;          //initialize recieved variables until first serial data
      pitch = 127;

      cameraPod.setConfig(1300, 450);    //set centers and ranges for servos
      gunPod.setConfig(1480, 165);
      azimuth.setConfig(1581, 30);

      cameraPos = 127;                   //initialize pods to lowest position
      cameraPos = constrain(cameraPos, -127, 127);
      gunPos = constrain(cameraPos-10, -127, 127);    //set gun position

      cameraPod.setSpeed(cameraPos);     //set servos
      gunPod.setSpeed(cameraPos);


      delay_ms(300);

      temp = read(0xEC);                 //enable OSD and display default reticule
      temp = temp & ~0x10;
      write(0x6C, temp);
      delay_ms(300);
      write(0x00, 0x08);
      displayReticule(RET_DIAMOND, RET_START);
```

```
	pitchLow = false;		//initialize variables
	pitchNew = false;
	rollLow = false;
	rollNew = false;

	battMode = 10;			//check battery
	checkBatt();
	battTime = 0;

	shields = 8;			//initialize damage levels
	hull = 8;
	displayDamage();

	return 0;
}




// This is the main loop
TICK_COUNT appControl(LOOP_COUNT loopCount, TICK_COUNT loopStart)
{
	if(button.pressed())		//on-board button acts as power off button
		shutdown();

if(!XBEE.isRxBufferEmpty())		//serial available from joystick, read data
	{
		temp = XBEE.read();

		if((temp & 0xF0) == 0x50)   //high nibble of pitch variable (originally
					    //designated as low nibble, hence the
					    //pitchLow name)
		{
			pitchT = temp & 0x0F;
			pitchLow = true;
		}
		else if(((temp & 0xF0) == 0x40) && pitchLow)    //low nibble of
								//pitch variable
		{
			pitchLow = false;
			pitchT = pitchT << 4;		//shift high nibble up
			pitchT = pitchT | (temp & 0x0F);   //insert low nibble
			pitch = pitchT;			//new pitch variable ready
		}
		else if((temp & 0xF0) == 0x70)    //high nibble of roll, same as pitch
		{
			rollT = temp & 0x0F;
			rollLow = true;
		}
		else if(((temp & 0xF0) == 0x60) && rollLow)     //low nibble of roll
		{
			rollLow = false;
			rollT = rollT << 4;
			rollT = rollT | (temp & 0x0F);
			roll = rollT;
		}
```

```
if((temp & 0xF2) == 0x12)          //power off button
{
       shutdown();

}
if((temp & 0xF1) == 0x01)          //trigger pressed
{
       LTfire.setPercent(10);
}
if((temp & 0xF1) == 0x00)          //trigger released
{
       LTfire.setPercent(0);
}


if((temp & 0xF2) == 0x02)          //button 2 pressed
{
       laser.on();
}
if((temp & 0xF2) == 0x00)          //button 2 released
{
       laser.off();
}


if((temp & 0xF4) == 0x04)          //headlight button pressed
{
       if(lightMode == 0)          //headlight configuration 1 selected
       {
              PIC1.high();
              PIC2.low();
       }
       else if(lightMode == 1)     //headlight configuratino 2 selected
       {
              PIC1.high();
              PIC2.high();
       }
}
else if((temp & 0xF4) == 0x00)     //headlight button released
{
       PIC1.low();
       PIC2.low();
}
if(((temp & 0xF8) == 0x28) && !lightMode2)//light change button pressed
{
       lightMode = lightMode + 1;  //change light mode (only 2 currently)
       if(lightMode == 2)
       {
              lightMode = 0;
       }
       lightMode2 = 1;

}
if((temp & 0xF8) == 0x20)   //light change button released
{
       lightMode2 = 0;
}
```

```
        if(((temp & 0xF4) == 0x24) && !retMode2) //reticule change button pressed
        {
                retMode = retMode + 1;      //change reticule
                if(retMode == 3)
                {
                        retMode = 0;
                }
                retMode2 = 1;

                if(retMode == 0)            //display new reticule
                {
                        displayReticule(RET_DIAMOND, RET_START);
                }
                else if(retMode == 1)
                {
                        displayReticule(RET_SQUARE, RET_START);
                }
                else if(retMode == 2)
                {
                        displayReticule(RET_CAT, RET_START);
                }
        }
        if((temp & 0xF4) == 0x20)          //reticule change button released
        {
                retMode2 = 0;
        }

        /*if(((temp & 0xF2) == 0x12) && !calMode)       //for calibrating compass,
                                                        //not available during
                                                        //normal operation
        {
                compassCalibrate.low();
                delay_ms(10);
                compassCalibrate.high();

                calMode = 1;
        }
        if((temp & 0xF2) == 0x10)
        {
                calMode = 0;
        }                                   //end calibrating compass */
}

if((shields < 8) && (loopStart - shieldTime > 5000000))       //regenerate shields
                                                              //one point
{
        shieldTime = loopStart;
        shields = shields + 1;
        displayDamage();
}

if(LTHit.isHigh() && loopStart - hitTime > 60000)      //One hit received
{
        shieldTime = loopStart;
        hitTime = loopStart;

        if(shields > 0)      //take points from shields first
                shields = shields - 1;
```

```
        else if(hull > 0)    //then remove from hull
                hull = hull - 1;

        else
                shutdown();   //"dead"

        displayDamage();
}

TICK_COUNT PT;              //time between pitch updates (changes speed of motion)
if(pitch > 127)
        PT = interpolate(pitch, 127, 255, 30000, 6000);

else if(pitch < 127)
        PT = interpolate(pitch, 127, 0, 30000, 6000);

else
        PT = 6000;

if(loopStart - pitchTime > PT)            //update pitch servos
{
        pitchTime = loopStart;

        cameraPos -= (pitch - 127)/4;     //set camera position

        cameraPos = constrain(cameraPos, -127, 127);
        gunPos = constrain(cameraPos+30, -127, 127);    //set gun position

        cameraPod.setSpeed(cameraPos);    //set servos
        gunPod.setSpeed(gunPos);
        //uart1.write(pitch);       //for debugging
}

if(loopStart - rollTime > 50)             //update azimuth servo
{
        rollTime = loopStart;
        azimuth.setSpeed(interpolate(roll, 0, 255, 127, -127));
}


if(loopStart - compassTime > 250000)      //get compass reading
{
        compassTime = loopStart;

        compass.read();
        COMPASS_TYPE temp = compass.getBearing();

        if(temp<90)   //adjust for rotation of compass from base
        {
                temp = temp + 360;
        }
        temp = temp - 90;

        if(temp != bearing)         //if bearing is new, then display
                                    //(cuts down on overhead when stationary)
        {
                bearing = temp;
```

```
                displayNumber(COMP_CENTER, bearing, true);
            }
        }


        if(loopStart - battTime > 7000000)       //check battery level and temperature
        {
            battTime = loopStart;
            checkBatt();                         //check battery and display level

            char temperature = a2dConvert8bit(temp1);       //check temperature
            if(temperature > 48)        //if overtemp, warn and shutdown
            {
                displayWarning(false);
                battMode = 0;
            }
            //displayNumber(COMP_CENTER + 30, temperature, false); //for calibrating
        }

        return 0;
}

void displayDamage() //displays the damage indicator
{
        write(0x05, 0x01);                  //display "S"
        write(0x06, DAMAGE_START);
        write(0x07, 0x1D);

        write(0x05, 0x01);                  //display start of shields symbol
        write(0x06, DAMAGE_START+1);
        write(0x07, 0xED);

        switch(shields)
        {
            case 0:       //empty
            write(0x05, 0x01);
            write(0x06, DAMAGE_START+2);
            write(0x07, 0xF2);

            write(0x05, 0x01);
            write(0x06, DAMAGE_START+3);
            write(0x07, 0xF2);
            break;

            case 1:       //0.125
            write(0x05, 0x01);
            write(0x06, DAMAGE_START+2);
            write(0x07, 0xF1);

            write(0x05, 0x01);
            write(0x06, DAMAGE_START+3);
            write(0x07, 0xF2);
            break;

            case 2:       //0.25
            write(0x05, 0x01);
            write(0x06, DAMAGE_START+2);
            write(0x07, 0xF0);
```

```
write(0x05, 0x01);
write(0x06, DAMAGE_START+3);
write(0x07, 0xF2);
break;

case 3:       //0.325
write(0x05, 0x01);
write(0x06, DAMAGE_START+2);
write(0x07, 0xEF);

write(0x05, 0x01);
write(0x06, DAMAGE_START+3);
write(0x07, 0xF2);
break;

case 4:       //0.5
write(0x05, 0x01);
write(0x06, DAMAGE_START+2);
write(0x07, 0xEE);

write(0x05, 0x01);
write(0x06, DAMAGE_START+3);
write(0x07, 0xF2);
break;

case 5:       //0.625
write(0x05, 0x01);
write(0x06, DAMAGE_START+2);
write(0x07, 0xEE);

write(0x05, 0x01);
write(0x06, DAMAGE_START+3);
write(0x07, 0xF1);
break;

case 6:       //0.75
write(0x05, 0x01);
write(0x06, DAMAGE_START+2);
write(0x07, 0xEE);

write(0x05, 0x01);
write(0x06, DAMAGE_START+3);
write(0x07, 0xF0);
break;

case 7:       //0.825
write(0x05, 0x01);
write(0x06, DAMAGE_START+2);
write(0x07, 0xEE);

write(0x05, 0x01);
write(0x06, DAMAGE_START+3);
write(0x07, 0xEF);
break;

default:      //full, also catches any errors
              //(there shouldn't be any, but it is good programming)
```

```
        write(0x05, 0x01);
        write(0x06, DAMAGE_START+2);
        write(0x07, 0xEE);

        write(0x05, 0x01);
        write(0x06, DAMAGE_START+3);
        write(0x07, 0xEE);
        break;
}

write(0x05, 0x01);                  //display end of shields symbol
write(0x06, DAMAGE_START+4);
write(0x07, 0xF3);



write(0x05, 0x01);                  //display "H"
write(0x06, DAMAGE_START+30);
write(0x07, 0x12);

write(0x05, 0x01);                  //display start of hull symbol
write(0x06, DAMAGE_START+31);
write(0x07, 0xED);

switch(hull)
{
        case 0:       //empty
        write(0x05, 0x01);
        write(0x06, DAMAGE_START+32);
        write(0x07, 0xF2);

        write(0x05, 0x01);
        write(0x06, DAMAGE_START+33);
        write(0x07, 0xF2);
        break;

        case 1:       //0.125
        write(0x05, 0x01);
        write(0x06, DAMAGE_START+32);
        write(0x07, 0xF1);

        write(0x05, 0x01);
        write(0x06, DAMAGE_START+33);
        write(0x07, 0xF2);
        break;

        case 2:       //0.25
        write(0x05, 0x01);
        write(0x06, DAMAGE_START+32);
        write(0x07, 0xF0);

        write(0x05, 0x01);
        write(0x06, DAMAGE_START+33);
        write(0x07, 0xF2);
        break;

        case 3:       //0.325
        write(0x05, 0x01);
```

```
write(0x06, DAMAGE_START+32);
write(0x07, 0xEF);

write(0x05, 0x01);
write(0x06, DAMAGE_START+33);
write(0x07, 0xF2);
break;

case 4:       //0.5
write(0x05, 0x01);
write(0x06, DAMAGE_START+32);
write(0x07, 0xEE);

write(0x05, 0x01);
write(0x06, DAMAGE_START+33);
write(0x07, 0xF2);
break;

case 5:       //0.625
write(0x05, 0x01);
write(0x06, DAMAGE_START+32);
write(0x07, 0xEE);

write(0x05, 0x01);
write(0x06, DAMAGE_START+33);
write(0x07, 0xF1);
break;

case 6:       //0.75
write(0x05, 0x01);
write(0x06, DAMAGE_START+32);
write(0x07, 0xEE);

write(0x05, 0x01);
write(0x06, DAMAGE_START+33);
write(0x07, 0xF0);
break;

case 7:       //0.825
write(0x05, 0x01);
write(0x06, DAMAGE_START+32);
write(0x07, 0xEE);

write(0x05, 0x01);
write(0x06, DAMAGE_START+33);
write(0x07, 0xEF);
break;

default:      //full, also catches any errors
              //(there shouldn't be any, but it is good programming)
write(0x05, 0x01);
write(0x06, DAMAGE_START+32);
write(0x07, 0xEE);

write(0x05, 0x01);
write(0x06, DAMAGE_START+33);
write(0x07, 0xEE);
break;
```

```
        }

        write(0x05, 0x01);                //display end of hull symbol
        write(0x06, DAMAGE_START+34);
        write(0x07, 0xF3);


        write(0x05, 0x00);   //housekeeping

}

void checkBatt()     //check battery level
{
        battLevel = a2dConvert8bit(battSense);
        //displayNumber(COMP_CENTER + 30, battLevel, false);   //for debugging
        if(battMode == 0)    //if level is critical and after 7-second delay, shutdown
        {
                shutdown();
        }
        else if(battLevel < 185)    //if critical, warn and prepare to shutdown
        {
                battMode = 0;
                displayWarning(true);
        }
        else if((battLevel > 187) && (battLevel < 190) && battMode != 1)     //empty
        {
                battMode = 1;
                displayBatt(battMode);
        }
        else if((battLevel > 192) && (battLevel < 200) && battMode != 2)     //0.25
        {
                battMode = 2;
                displayBatt(battMode);
        }
        else if((battLevel > 202) && (battLevel < 210) && battMode != 3)     //0.5
        {
                battMode = 3;
                displayBatt(battMode);
        }
        else if((battLevel > 212) && (battLevel < 220) && battMode != 4)     //0.75
        {
                battMode = 4;
                displayBatt(battMode);
        }
        else if(battLevel > 222 && battMode != 5)       //full
        {
                battMode = 5;
                displayBatt(battMode);
        }
}

void displayBatt(int bm)    //display battery level
{
        write(0x05, 0x01);                //display start of battery symbol
        write(0x06, BATT_START);
        write(0x07, 0xED);

        switch(bm)
```

```
{
        case 1:       //empty
        write(0x05, 0x01);
        write(0x06, BATT_START+1);
        write(0x07, 0xF2);

        write(0x05, 0x01);
        write(0x06, BATT_START+2);
        write(0x07, 0xF2);
        break;

        case 2:       //0.25
        write(0x05, 0x01);
        write(0x06, BATT_START+1);
        write(0x07, 0xF0);

        write(0x05, 0x01);
        write(0x06, BATT_START+2);
        write(0x07, 0xF2);
        break;

        case 3:       //0.5
        write(0x05, 0x01);
        write(0x06, BATT_START+1);
        write(0x07, 0xEE);

        write(0x05, 0x01);
        write(0x06, BATT_START+2);
        write(0x07, 0xF2);
        break;

        case 4:       //0.75
        write(0x05, 0x01);
        write(0x06, BATT_START+1);
        write(0x07, 0xEE);

        write(0x05, 0x01);
        write(0x06, BATT_START+2);
        write(0x07, 0xF0);
        break;

        default:      //full, also catches any errors
                      //(there shouldn't be any, but it is good programming)
        write(0x05, 0x01);
        write(0x06, BATT_START+1);
        write(0x07, 0xEE);

        write(0x05, 0x01);
        write(0x06, BATT_START+2);
        write(0x07, 0xEE);
        break;
}


write(0x05, 0x01);                //display end of battery symbol
write(0x06, BATT_START+3);
write(0x07, 0xF3);
```

```
	write(0x05, 0x00);   //housekeeping

}

void displayNumber(int numCenter, int number, bool brackets)  //displays a 3-digit number
{
	char Dtemp = number / 100;  //100's digit
	if(Dtemp == 0)
	{
		Dtemp = 0x0A;
	}
	write(0x06, numCenter-1);
	write(0x07, Dtemp);

	Dtemp = (number%100) / 10;  //10's digit
	if(Dtemp == 0)
	{
		Dtemp = 0x0A;
	}
	write(0x06, numCenter);
	write(0x07, Dtemp);

	Dtemp = (number%10); //1's digit
	if(Dtemp == 0)
	{
		Dtemp = 0x0A;
	}
	write(0x06, numCenter+1);
	write(0x07, Dtemp);

	if(brackets)  ///brackets if called for (compass heading only)
	{
		write(0x06, numCenter-2);
		write(0x07, 0x4A);

		write(0x06, numCenter+2);
		write(0x07, 0x4B);
	}

}

void displayReticule(int charStart, int dispStart)            //displays the reticule
{


	write(0x06, dispStart);     //first character
	write(0x07, charStart);

	delay_ms(5);

	write(0x06, dispStart+1);
	write(0x07, charStart+1);

	delay_ms(5);

	write(0x06, dispStart+2);
	write(0x07, charStart+2);
```

```
        delay_ms(5);

        write(0x06, dispStart+3);
        write(0x07, charStart+3);

        delay_ms(5);

        write(0x06, dispStart+30);
        write(0x07, charStart+4);

        delay_ms(5);

        write(0x06, dispStart+31);
        write(0x07, charStart+5);

        delay_ms(5);

        write(0x06, dispStart+32);
        write(0x07, charStart+6);

        delay_ms(5);

        write(0x06, dispStart+33);
        write(0x07, charStart+7);   //last character

        delay_ms(5);
}

void displayWarning(bool batt)     //displays batt or temp warning
{
        if(batt)      //battery first line
        {
                for(int i=0; i<16; i++)
                {
                        write(0x06, LINE_1_B_START+i);
                        write(0x07, line1B[i]);
                }
        }
        else          //temperature first line
        {
                for(int i=0; i<20; i++)
                {
                        write(0x06, LINE_1_T_START+i);
                        write(0x07, line1T[i]);
                }
        }

        for(int i=0; i<18; i++)           //second line
        {
                write(0x06, LINE_2_START+i);
                write(0x07, line2[i]);
        }

}

char read(char Address)           //reads a byte from OSD
{
```

```
	OSD.select(true);
	char Data = OSD.xfer(Address);
	OSD.select(false);
	return Data;
}

char write(char Address, char Data)		//writes a byte to OSD
{
	OSD.select(true);
	OSD.write(Address);
	OSD.write(Data);
	OSD.select(false);
	return Data;
}

void shutdown()		//shuts down the turret (requires hardware to turn back on)
{
	power.low();
}


int constrain(int val, int low, int high)//constrains a value to a range
{
	if(val < low)
		return low;

	else if(val > high)
		return high;

	else
		return val;
}
```

```
' Define configuration settings.
#CONFIG
__CONFIG _CONFIG1, _INTRC_IO & _PWRTE_ON & _MCLR_OFF & _LVP_OFF
#ENDCONFIG


DEFINE OSC 8  ' Set the internal oscillator frequency to 8 MHz.
OSCCON.4 = 1
OSCCON.5 = 1
OSCCON.6 = 1
ANSEL = 0       ' Turn off the analog to digital converters.

' Define Variables
led1 Var PORTB.0
I1 var word
I2 var word

I1 = 0              'Set initial count to 0.


TRISA = %11111111   ' Set port A as all inputs.
TRISB = $00         ' Set port B as all outputs.

PORTB = $01

' Main loop.
loop1:
  I1 = I1+1             ' Increase count by 1.

  if (PORTA.2 == 1 && PORTA.3 == 0) then    ' If button 3 is depressed LEDs 1, 3, 5, and 7
    PORTB = $99                             ' are high.
    while (PORTA.2 == 1)
    wend
    PORTB = 0
    pause 10
  endif

  if (PORTA.2 == 1 && PORTA.3 == 1) then    ' If button 3 is depressed after button 12 is
    PORTB = $55                             ' pressed LEDs 0, 2, 4, and 6 are high.
    while (PORTA.2 == 1)
    wend
    PORTB = 0
    pause 10
  endif

  if (PORTA.0 == 1) then                    ' When trigger is pressed LEDs cycle through using
   PORTB = PORTB << 1                       ' bit swapping.
  I1=0
```

```
   if ((PORTB == $10) || (PORTB == 0))then
      PORTB = $11
   endif

   pause 10
   endif

   if (I1 > 1000)then
      I1 = 0
      PORTB = 0
   endif
goto loop1

end
```

```
#define ROL  A0                 //pin definitions
#define PIT  A1
#define YAW  A2
#define THR  A3
#define UD   A4
#define LR   A5

const int SENDDEL = 5;          //delay between packets sent

byte switches;                  //byte to send
byte pitch;                     //joystick pitch position
byte roll;                      //joystick roll position
byte yaw;                       //joystick yaw position
byte range;                     //range for variables (set by throttle)
byte Max;                       //top end of range
byte Min;                       //bottom end of range
byte pitchDir;                  //stores normal or reverse pitch control setting

int temp;                       //variable for reading from analog inputs


void setup()
{
 pitchDir = 0;
 Serial.begin(57600);                   //initializes uart to 57600 baud

 for(int i=2; i<14; i++)                //sets all buttons to have pull-up resistors
   pinMode(i, INPUT_PULLUP);

}

void loop()
{
 getswitches();          //gets the button values and sends them
 //gethat();             //gets the 4-way hat and sends its data (not used currently)
 getaxes();              //gets the positions of the axes and sends them
}




void getswitches()      //gets the button values and sends them
{
 switches = 0x00;       //sets address nibble

 if(!digitalRead(2))
   switches = switches | B00000001;
 else
```

```
  switches = switches & B11111110;

if(!digitalRead(3))
  switches = switches | B00000010;
else
  switches = switches & B11111101;

if(!digitalRead(4))
  switches = switches | B00000100;
else
  switches = switches & B11111011;

if(!digitalRead(5))
  switches = switches | B00001000;
else
  switches = switches & B11110111;

Serial.write(switches);          //sends switches 1-4
delay(SENDDEL);                  //delay to prevent choking AXON uart
                                 //(can be removed or modified to increase update speed)


switches = 0x10;         //sets address nibble

if(!digitalRead(6))
  switches = switches | B00000001;
else
  switches = switches & B11111110;

if(!digitalRead(7))
  switches = switches | B00000010;
else
  switches = switches & B11111101;

if(!digitalRead(8))
  switches = switches | B00000100;
else
  switches = switches & B11111011;

if(!digitalRead(9))
{
  switches = switches | B00001000;

  if(pitchDir == 0)              //switches pitch direction and waits until button is released without
    pitchDir = 3;                 //blocking program
  else if(pitchDir == 1)
    pitchDir = 2;
}
else
{
```

```
  switches = switches & B11110111;

  if(pitchDir == 2)                  //locks in pitch direction when button is released
    pitchDir = 0;
  else if(pitchDir == 3)
    pitchDir = 1;
 }

 Serial.write(switches);            //sends switches 5-8
 delay(SENDDEL);


 switches = 0x20;          //sets address nibble

 if(!digitalRead(10))
  switches = switches | B00000001;
 else
  switches = switches & B11111110;

 if(!digitalRead(11))
  switches = switches | B00000010;
 else
  switches = switches & B11111101;

 if(!digitalRead(12))
  switches = switches | B00000100;
 else
  switches = switches & B11111011;

 if(!digitalRead(13))
  switches = switches | B00001000;
 else
  switches = switches & B11110111;

 Serial.write(switches);            //sends switches 9-12
 delay(SENDDEL);
}

void gethat()    //gets the 4-way hat and sends its data (not used currently)
 switches = 0x30;          //sets address nibble

 if(analogRead(UD)>823)
  switches = switches | B00000001;
 else
  switches = switches & B11111110;

 if(analogRead(UD)<200)
  switches = switches | B00000010;
 else
  switches = switches & B11111101;
```

```
 if(analogRead(LR)>823)
  switches = switches | B00000100;
 else
  switches = switches & B11111011;

 if(analogRead(LR)<200)
  switches = switches | B00001000;
 else
  switches = switches & B11110111;

 Serial.write(switches);  //sends 4-way hat info
 delay(SENDDEL);
}

void getaxes()  //gets the positions of the axes and sends them
{
 range = constrain(map(analogRead(THR),285, 980, 120, 0), 0, 120);          //sets range based on
                                                                            // throttle  level
 range = constrain(7 + range, 0, 127);

 Max = 128 + range;
 Min = 127 - range;

 temp = analogRead(PIT);
 if(temp < 440)
    pitch = constrain(map(temp,120, 440, Min, 127), Min, Max);
 else if(temp > 530)
  pitch = constrain(map(temp,530, 890, 127, Max), Min, Max);
 else
  pitch = 127;   //in dead zone

 if(pitchDir == 1 || pitchDir == 3)
  pitch = map(pitch, Min, Max, Max, Min);  //switch pitch direction

 switches = 0x40 | (0x0F & pitch);                  //sets address nibble and adds the lower data nibble
 Serial.write(switches);          //sends lower pitch nibble
 delay(SENDDEL);

 switches = 0x50 | (0x0F & (pitch>>4));          //sets address nibble and adds the upper data nibble
 Serial.write(switches);          //sends upper pitch nbble
 delay(SENDDEL);


 Max = constrain(Max, 145, 255);        //cuts off lower portion of range for the azimuth control
 Min = constrain(Min, 0, 110);          //to prevent the servo binding

 temp = analogRead(ROL);
 if(temp < 470)
  roll = constrain(map(temp,130, 470, Min, 127), Min, Max);
```

```
 else if(temp > 530)
  roll = constrain(map(temp,530, 940, 127, Max), Min, Max);
 else
  roll = 127;                //in dead zone

 switches = 0x60 | (0x0F & roll);                    //sets address nibble and adds the lower data nibble
 Serial.write(switches);          //sends lower roll nibble
 delay(SENDDEL);

 switches = 0x70 | (0x0F & (roll>>4));          //sets address nibble and adds the upper data nibble
 Serial.write(switches);          //sends upper roll nibble
 delay(SENDDEL);

 /*temp = analogRead(YAW);             //yaw not sent currently, but available for future projects
 if(temp < 480)
  yaw = constrain(map(temp,130, 480, Min, 127), Min, Max);
 else if(temp > 540)
  yaw = constrain(map(temp,540, 940, 127, Max), Min, Max);
 else
  yaw = 127;

 switches = 0x80 | (0x0F & yaw);                    //sets address nibble and adds the lower data nibble
 Serial.write(switches);          //sends lower yaw nibble
 delay(SENDDEL);

 switches = 0x90 | (0x0F & (yaw>>4));                    //sets address nibble and adds the upper
                                                         //data nibble
 Serial.write(switches);          //sends upper yaw nibble
 delay(SENDDEL);*/
}
```

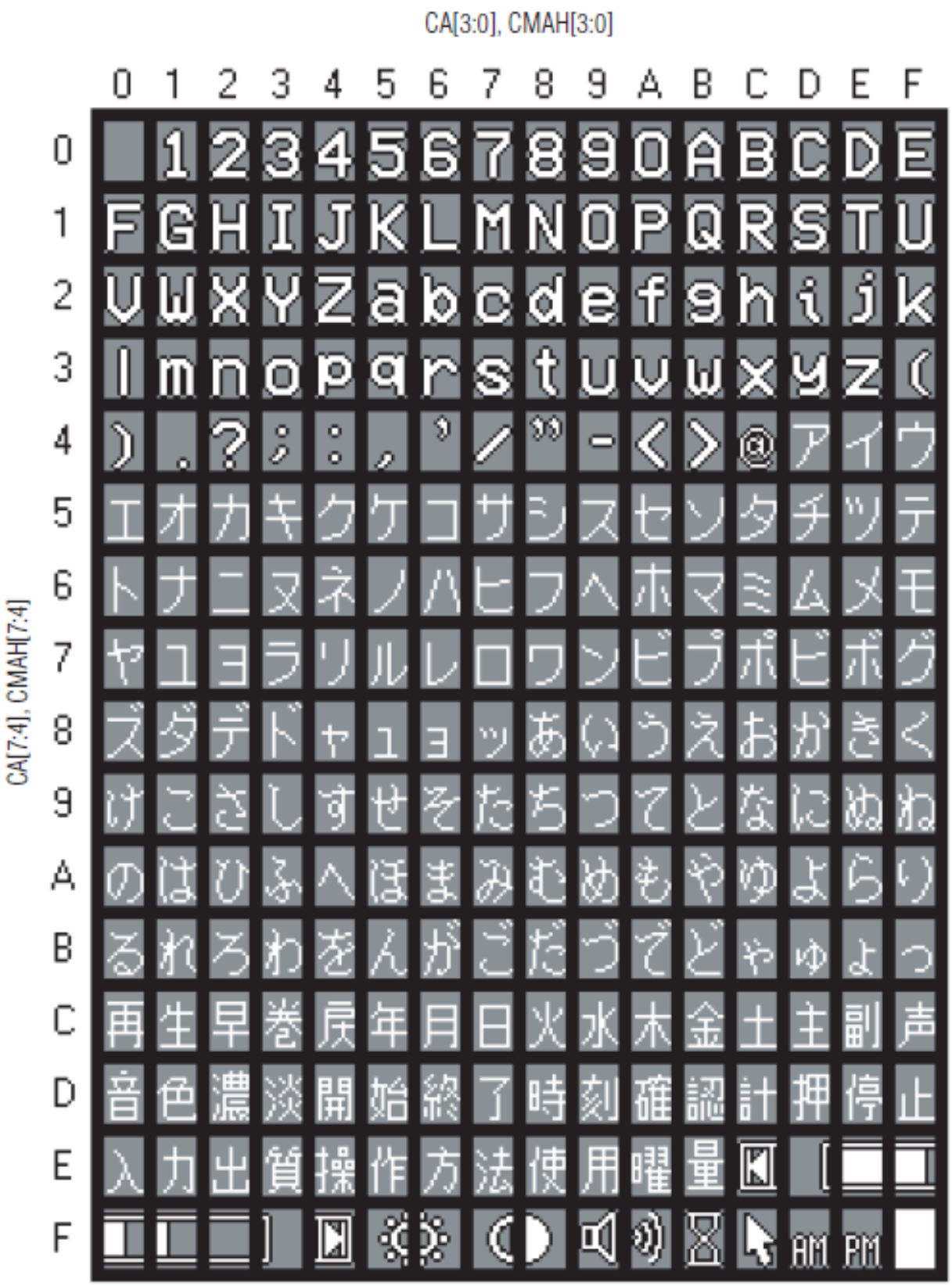
PIXEL COLUMN NUMBER

PIXEL ROW NUMBER	0	1	2	3	4	5	6	7	8	9	10	11	CHARACTER MEMORY ADDRESS LOW CMAL[5:0]
0	CDMI [7, 6]	CDMI [5, 4]	CDMI [3, 2]	CDMI [1, 0]	[7, 6]	[5, 4]	[3, 2]	[1, 0]	[7, 6]	[5, 4]	[3, 2]	[1, 0]	0, 1, 2
1	[7, 6]	[5, 4]	[3, 2]	[1, 0]	[7, 6]	[5, 4]	[3, 2]	[1, 0]	[7, 6]	[5, 4]	[3, 2]	[1, 0]	3, 4, 5
2	[7, 6]	[5, 4]	[3, 2]	**[1, 0]**	**[7, 6]**	**[5, 4]**	**[3, 2]**	**[1, 0]**	**[7, 6]**	**[5, 4]**	[3, 2]	[1, 0]	6, 7, 8
3	[7, 6]	[5, 4]	**[3, 2]**	[1, 0]	[7, 6]	[5, 4]	[3, 2]	[1, 0]	[7, 6]	[5, 4]	**[3, 2]**	[1, 0]	9, 10, 11
4	[7, 6]	**[5, 4]**	[3, 2]	[1, 0]	[7, 6]	[5, 4]	[3, 2]	[1, 0]	[7, 6]	[5, 4]	[3, 2]	**[1, 0]**	12, 13, 14
5	**[7, 6]**	[5, 4]	[3, 2]	[1, 0]	**[7, 6]**	**[5, 4]**	**[3, 2]**	**[1, 0]**	**[7, 6]**	[5, 4]	[3, 2]	**[1, 0]**	15, 16, 17
6	**[7, 6]**	[5, 4]	[3, 2]	**[1, 0]**	[7, 6]	[5, 4]	[3, 2]	[1, 0]	[7, 6]	**[5, 4]**	**[3, 2]**	[1, 0]	18, 19, 20
7	**[7, 6]**	[5, 4]	[3, 2]	**[1, 0]**	[7, 6]	[5, 4]	[3, 2]	[1, 0]	[7, 6]	[5, 4]	[3, 2]	[1, 0]	21, 22, 23
8	**[7, 6]**	[5, 4]	[3, 2]	[1, 0]	**[7, 6]**	**[5, 4]**	**[3, 2]**	**[1, 0]**	**[7, 6]**	[5, 4]	[3, 2]	[1, 0]	24, 25, 26
9	[7, 6]	**[5, 4]**	[3, 2]	[1, 0]	[7, 6]	[5, 4]	[3, 2]	[1, 0]	[7, 6]	**[5, 4]**	[3, 2]	[1, 0]	27, 28, 29
10	[7, 6]	[5, 4]	**[3, 2]**	[1, 0]	[7, 6]	[5, 4]	[3, 2]	[1, 0]	[7, 6]	[5, 4]	**[3, 2]**	[1, 0]	30, 31, 32
11	[7, 6]	[5, 4]	[3, 2]	**[1, 0]**	**[7, 6]**	**[5, 4]**	**[3, 2]**	**[1, 0]**	[7, 6]	[5, 4]	[3, 2]	**[1, 0]**	33, 34, 35
12	[7, 6]	[5, 4]	[3, 2]	[1, 0]	[7, 6]	[5, 4]	[3, 2]	[1, 0]	**[7, 6]**	[5, 4]	[3, 2]	**[1, 0]**	36, 37, 38
13	[7, 6]	**[5, 4]**	**[3, 2]**	[1, 0]	[7, 6]	[5, 4]	[3, 2]	[1, 0]	**[7, 6]**	[5, 4]	[3, 2]	**[1, 0]**	39, 40, 41
14	**[7, 6]**	[5, 4]	[3, 2]	**[1, 0]**	**[7, 6]**	**[5, 4]**	**[3, 2]**	**[1, 0]**	[7, 6]	[5, 4]	[3, 2]	**[1, 0]**	42, 43, 44
15	**[7, 6]**	[5, 4]	[3, 2]	[1, 0]	[7, 6]	[5, 4]	[3, 2]	[1, 0]	[7, 6]	[5, 4]	**[3, 2]**	[1, 0]	45, 46, 47
16	[7, 6]	**[5, 4]**	[3, 2]	[1, 0]	[7, 6]	[5, 4]	[3, 2]	[1, 0]	[7, 6]	**[5, 4]**	[3, 2]	[1, 0]	48, 49, 50
17	[7, 6]	[5, 4]	**[3, 2]**	**[1, 0]**	**[7, 6]**	**[5, 4]**	**[3, 2]**	**[1, 0]**	**[7, 6]**	[5, 4]	[3, 2]	[1, 0]	51, 52, 53

2-BIT PIXEL DEFINITION:

[x, y]	00 = BLACK
[x, y]	10 = WHITE
[x, y]	X1 = TRANSPARENT (EXTERNAL SYNC MODE) OR GRAY (INTERNAL SYNC MODE)

X = DON'T CARE


CA[3:0], CMAH[3:0]
0 1 2 3 4 5 6 7 8 9 A B C D E F
CA[7:4], CMAH[7:4]
0 1 2 3 4 5 6 7 8 9 A B C D E F
AM
PM

```
#include "hardware.h"       //Webbot's hardware file

#define RET_START    192    //start of reticle display
#define RET_CAT      0x80   //start of cat reticle in memory
#define RET_DIAMOND   0x70  //start of diamond reticle in memory
#define RET_SQUARE    0x60  //start of square reticle in memory

//color shortcuts for programming characters
#define B1 0x00
#define B2 0x00
#define B3 0x00
#define B4 0x00
#define BA 0x00

#define W1 0x80
#define W2 0x20
#define W3 0x08
#define W4 0x02
#define WA W1|W2|W3|W4

#define G1 0x40
#define G2 0x10
#define G3 0x04
#define G4 0x01
#define GA G1|G2|G3|G4


char read(char Address);    //reads a byte from the OSD
char write(char Address, char Data);     //writes a byte to the OSD
void displayReticule(int charStart, int dispStart);    //displays the reticule on
                                                       //screen

// Initialize the hardware
void appInitHardware(void) {
       initHardware();
}
// Initialize the software
TICK_COUNT appInitSoftware(TICK_COUNT loopStart)
{
       delay_ms(600);

       temp = read(0xEC);          //set OSD black level
       temp = temp & ~0x10;
       write(0x6C, temp);

       delay_ms(700);

       return 0;
}


// This is the main loop
TICK_COUNT appControl(LOOP_COUNT loopCount, TICK_COUNT loopStart)
{
```

```
////////////////////////////////////INSERT CHAR HERE/////////////////////////////////


////////////////////////////////END INSERT CHAR HERE/////////////////////////////////


	write(0x08, 0xA0);   //tells the OSD to write the character from shadow ROM to
character memory

	delay_ms(100);

	write(0x00, 0x08);   //turn on screen

	displayReticule(RET_CAT, RET_START);


	while(1)
	{
		if(button.pressed()) //power off
			power.low();
	}

	return 0;
}

void displayReticule(int charStart, int dispStart)     //displays the reticle
{


	write(0x06, dispStart);
	write(0x07, charStart);

	delay_ms(5);

	write(0x06, dispStart+1);
	write(0x07, charStart+1);

	delay_ms(5);

	write(0x06, dispStart+2);
	write(0x07, charStart+2);

	delay_ms(5);

	write(0x06, dispStart+3);
	write(0x07, charStart+3);

	delay_ms(5);

	write(0x06, dispStart+30);  //next row
	write(0x07, charStart+4);

	delay_ms(5);

	write(0x06, dispStart+31);
	write(0x07, charStart+5);

	delay_ms(5);
```

```
        write(0x06, dispStart+32);
        write(0x07, charStart+6);

        delay_ms(5);

        write(0x06, dispStart+33);
        write(0x07, charStart+7);

        delay_ms(5);
}



char read(char Address)     //reads a byte from the OSD
{
        OSD.select(true);
        char Data = OSD.xfer(Address);
        OSD.select(false);
        return Data;
}

char write(char Address, char Data)       //writes a byte to the OSD
{
        OSD.select(true);
        OSD.write(Address);
        OSD.write(Data);
        OSD.select(false);
        return Data;
}
```

```
write(0x09, 0x64);   //tell OSD which character to program

write(0x0A, 0);             //pixel set 1
write(0x0B, (G1 | W2 | W3 | W4)); //gray, white, white, white

write(0x0A, 1);             //pixel set 2
write(0x0B, (WA));          //all white

write(0x0A, 2);             //pixel set 3
write(0x0B, (WA));          //all white

write(0x0A, 3);             //pixel set 4
write(0x0B, (G1 | G2 | G3 | G4)); //all gray

write(0x0A, 4);             //pixel set 5
write(0x0B, (G1 | G2 | G3 | G4)); //all gray

write(0x0A, 5);             //pixel set 6
write(0x0B, (G1 | G2 | G3 | G4)); //all gray

write(0x0A, 6);             //pixel set 7
write(0x0B, (G1 | G2 | G3 | G4)); //all gray

//more of the same
```