6.115 Final Project Report
May 14, 2014
Michael Everett

Introduction

This project is related to my favorite topics, which are controls and robotics. Air hockey is a fun game that everyone knows how to play, so it is easy to explain the goal of the project. I have some solutions to this problem already: one was a fairly serious research project featured in IEEE 2013, and another was posted on Reddit earlier this year. However most people use fancy cameras to track the puck and complicated motor systems.

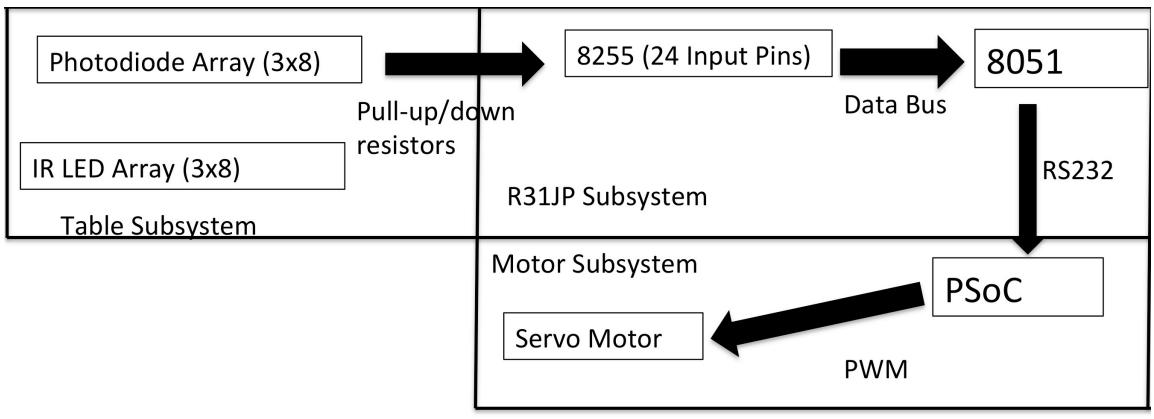
In this project I built a robotic goalie for air hockey that will only use common electronic components: photodiodes, LEDs, and a servo motor. This is a creative project because it involves a relatively inexpensive solution to a classic arcade game. An array of LEDs is mounted facing upwards through holes in the table. An array of photodiodes is opposite the LEDs, facing down. Then as a puck crosses in between the two systems, the photodiodes change signals, which can be measured with a 8255 and then to a microcontroller.

Hardware Description

The three main systems in my project are:

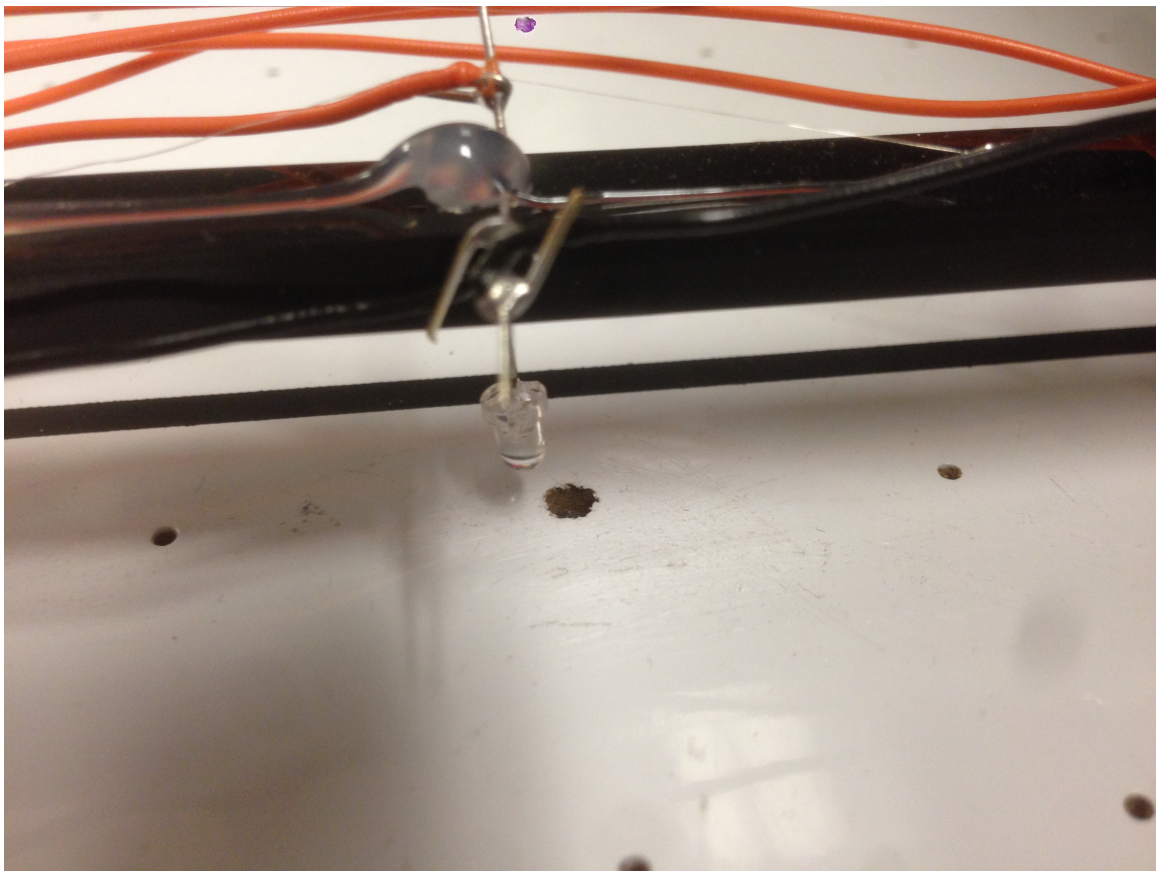
- Table + Sensors
- R31JP (8051 + essential circuits) + Lab Kit
- PSoC 5 (Cypress Programmable System on a Chip)

They each communicate with each other as follows:

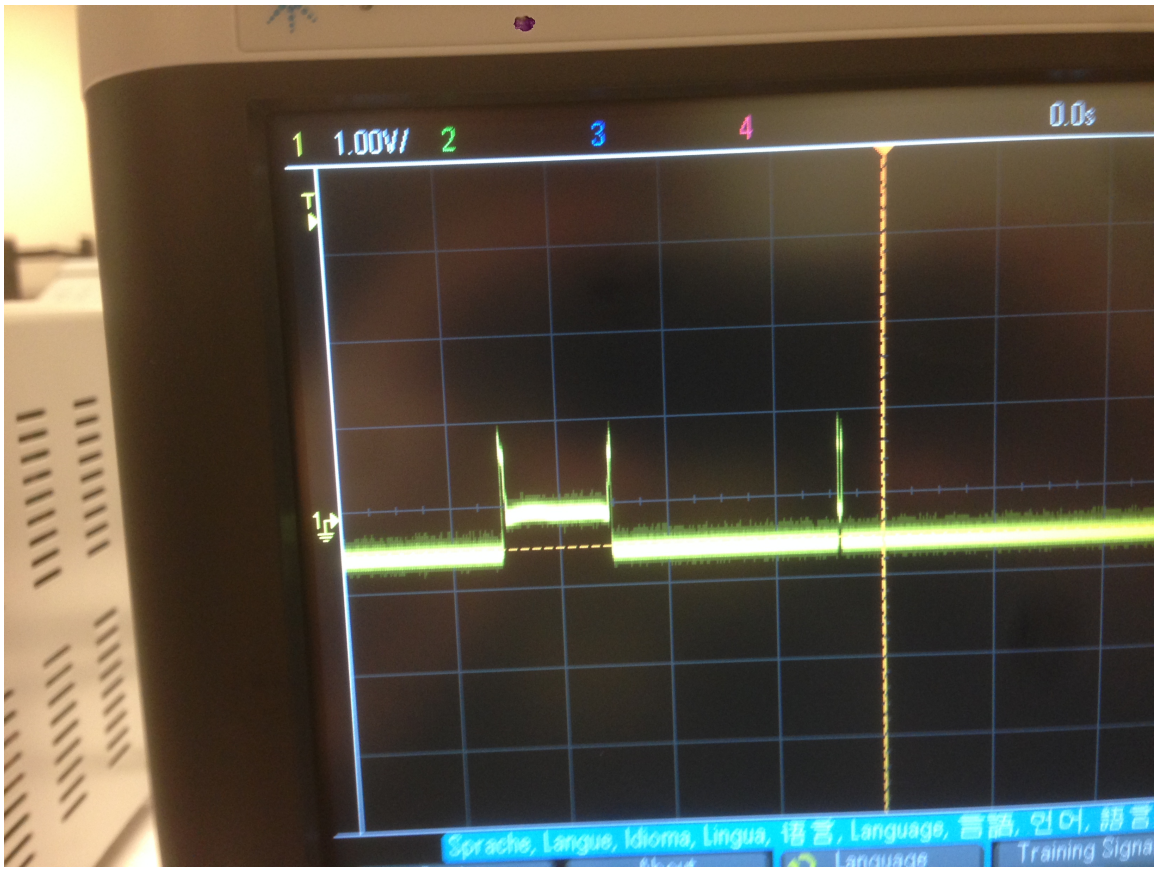


I will walk through the hardware events associated with a moving puck leading to a servo motor instruction.

This is a photodiode (Digikey # 1080-1140-ND) mounted on a bar which hangs above the table. It is optimized for IR wavelengths ~900nm. It is connected to a pull-down resistor so its natural voltage reading is low when it is illuminated. Pointing up out of the big hole is an IR LED (Digikey # 754-1600-ND) which is also ~900nm. The LED has a current-limiting resistor of 1.1K and is wired between +12V and GND. The same current could have been applied with a smaller resistor value and +5V instead. You can see the schematic for the diodes later in this report.



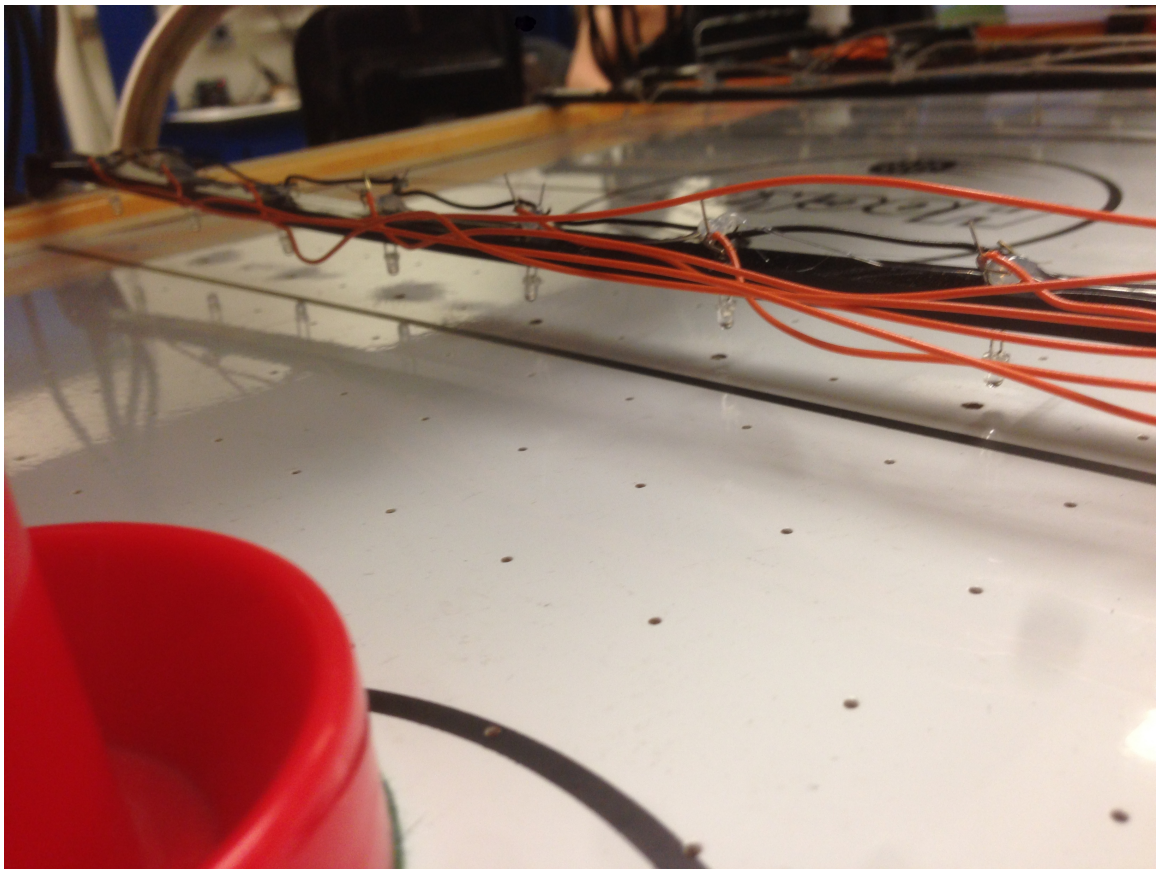
When the puck goes between the LED and Photodiode, a spike of ~2V occurs then it settles to a value ~0.5V above low. This switch is enough for the 8255 to sense a logic change. I tried using bigger resistance for bigger voltage swings, but the time constant of the system then comes into play. Since the diode has some capacitance, tau ~ R. I picked the value of R experimentally. This scope shot shows a puck crossing under, holding, leaving, then zooming through one diode.



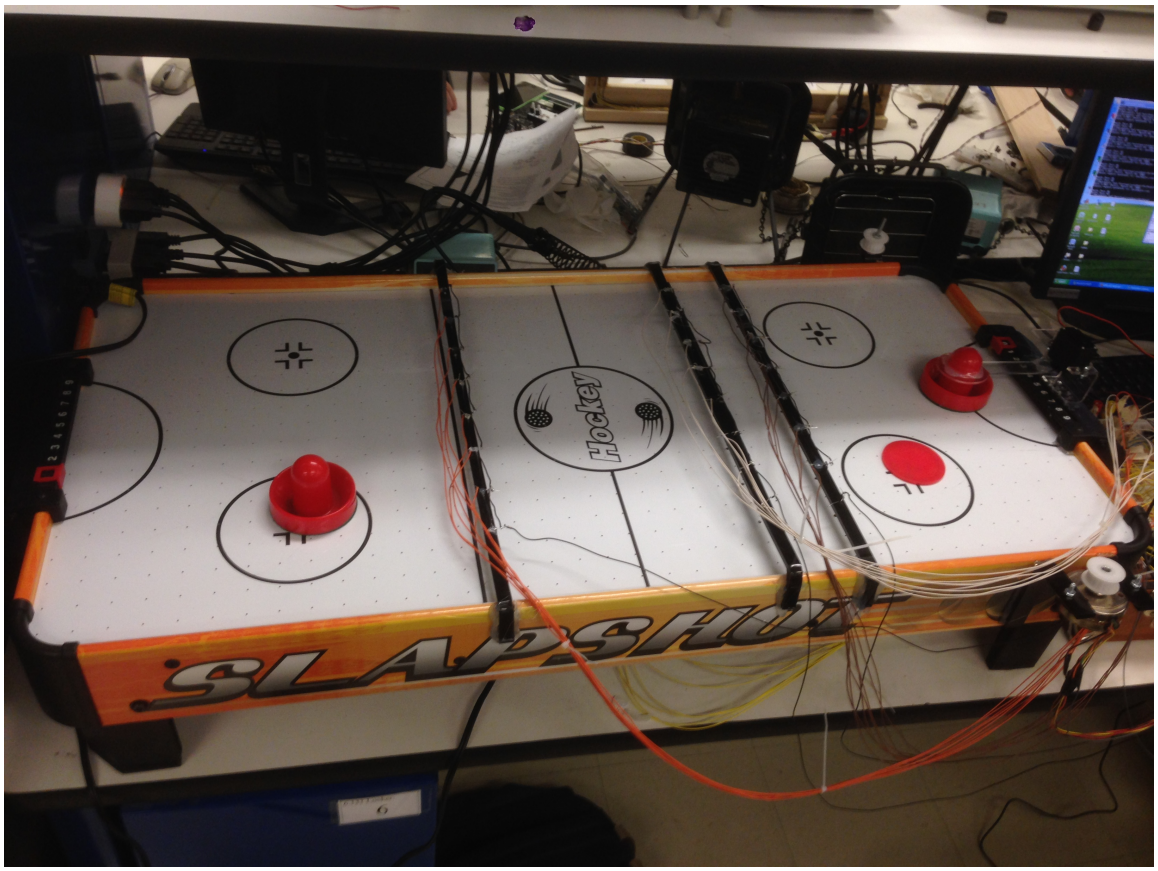
|__No Puck__|_Puck_|__No Puck___|___No Puck_____|
| | | | |
^ ^ ^

Puck enters — Puck leaves — Puck goes through quickly

Here is a picture of one bar. The gap between the table and diode is just enough for a puck to squeeze through. The orange wires hold the 8 signals and are pulled up on the lab kit, and the black wires are commonly grounded.

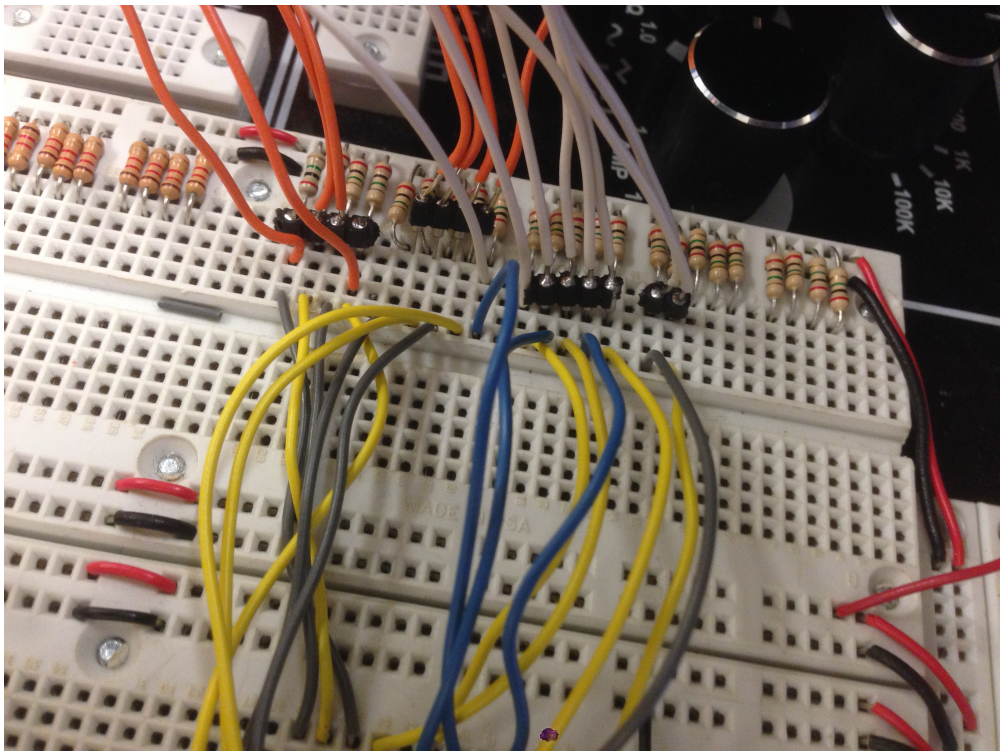


The locations for the bars were picked strategically. 2 points define a line, but I had to account for bounces. Therefore I needed 3 bars to predict the puck's end path, assuming the puck only travels in a straight line.



The 24 photodiodes are pulled up then connected to the 8255.

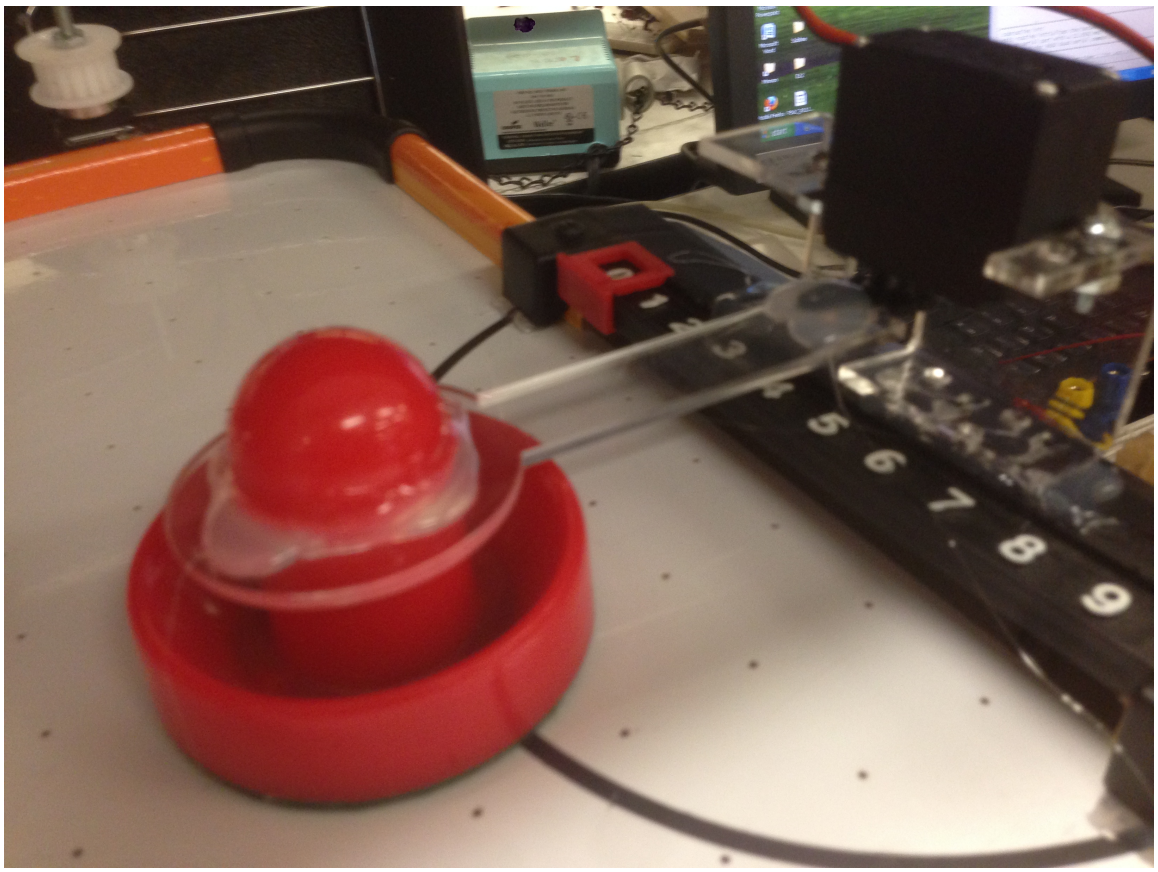
Photodiodes



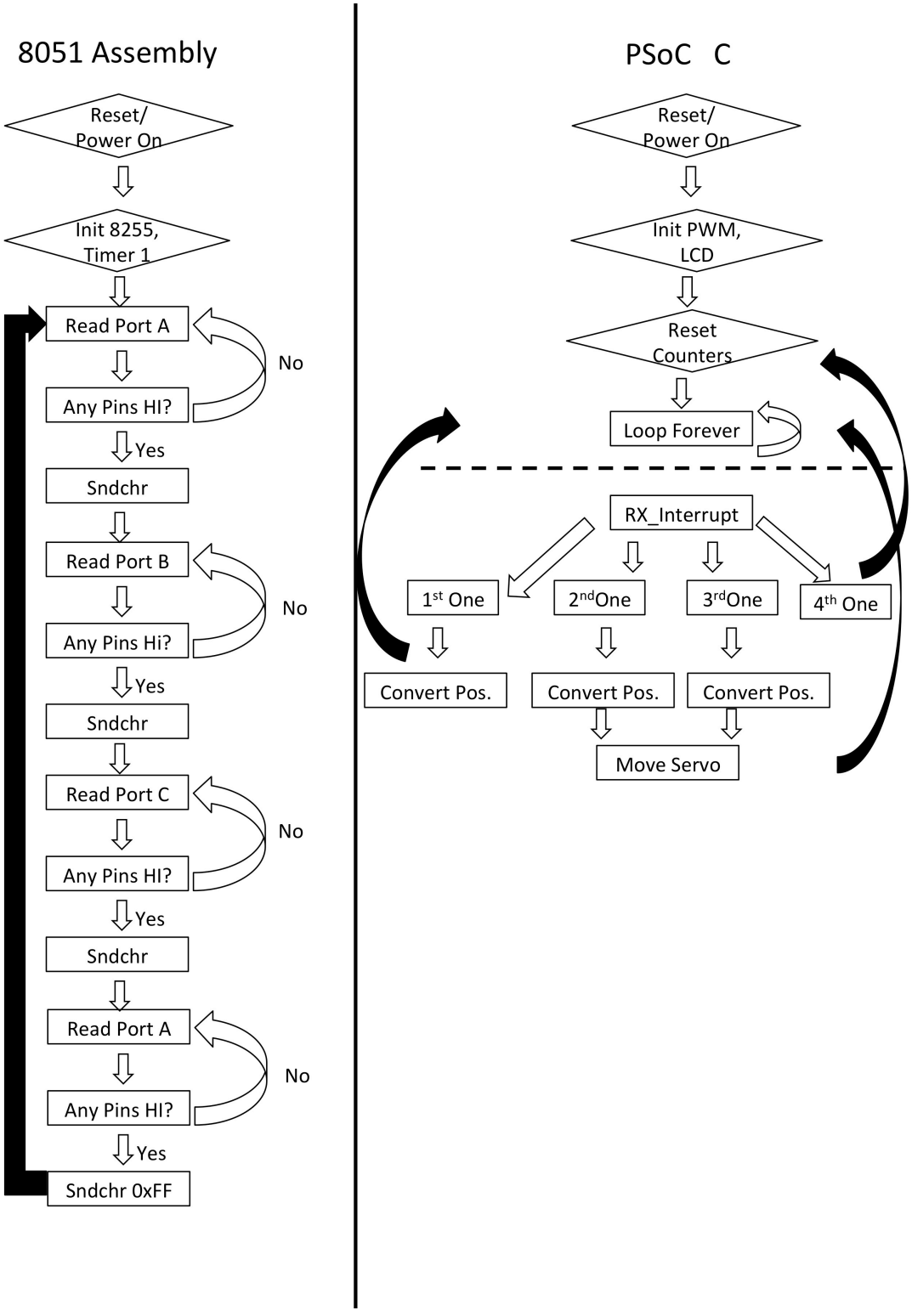
8255

The 8255 connects to the 8051 via the data bus. That connects to the PSoC via RS232. The PSoC connects to the servo motor via Pin 4[7] which generates a PWM to move the servo.

The motor drive system is a servo which has internal closed-loop feedback for position accuracy. I command the location based on a PWM signal. A 5ms period with pulses ranging from 1-2ms encode the desired position. I controlled the PWM with the PSoC.



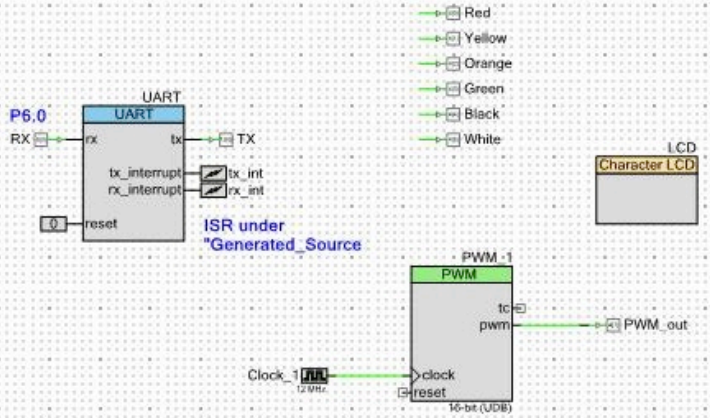
Software Description



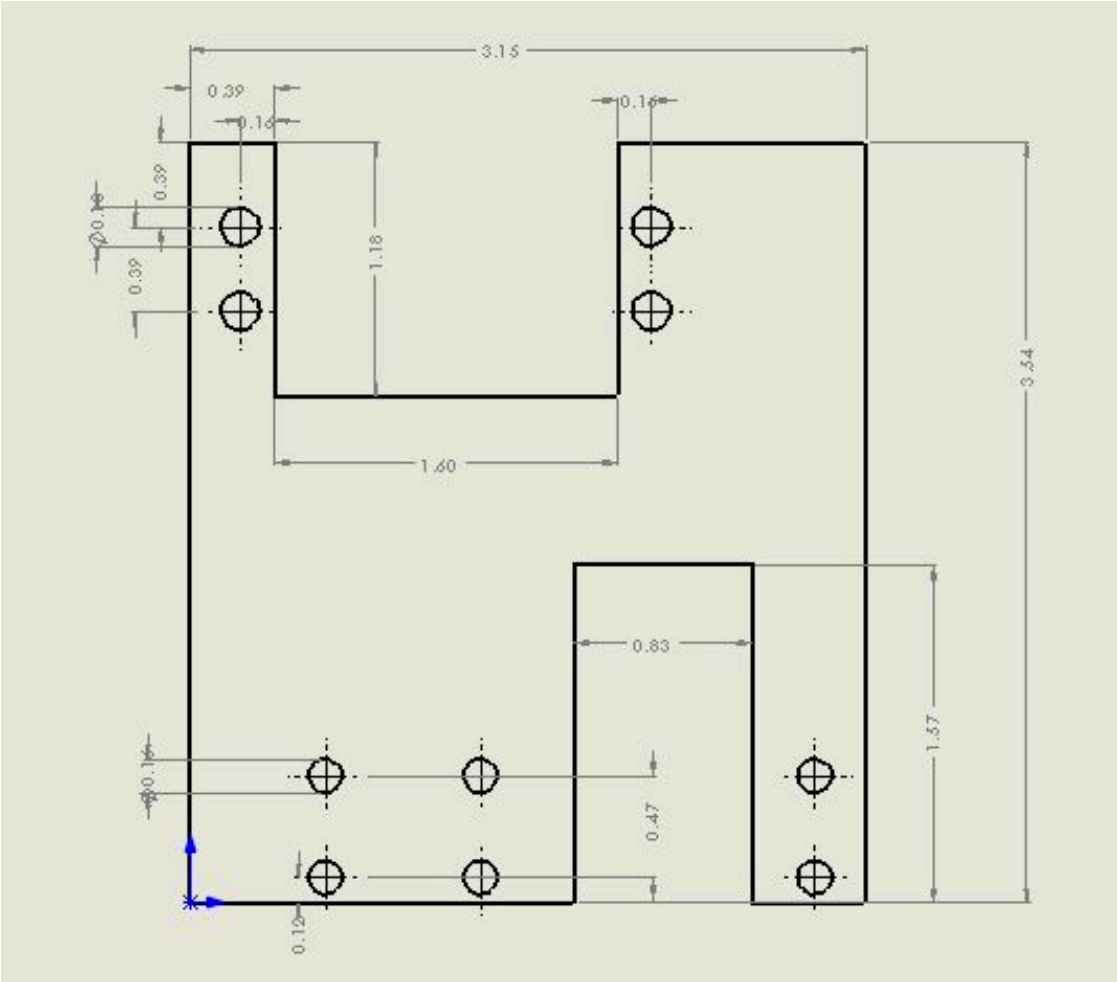
PSoC Top Design

Air Hockey Table

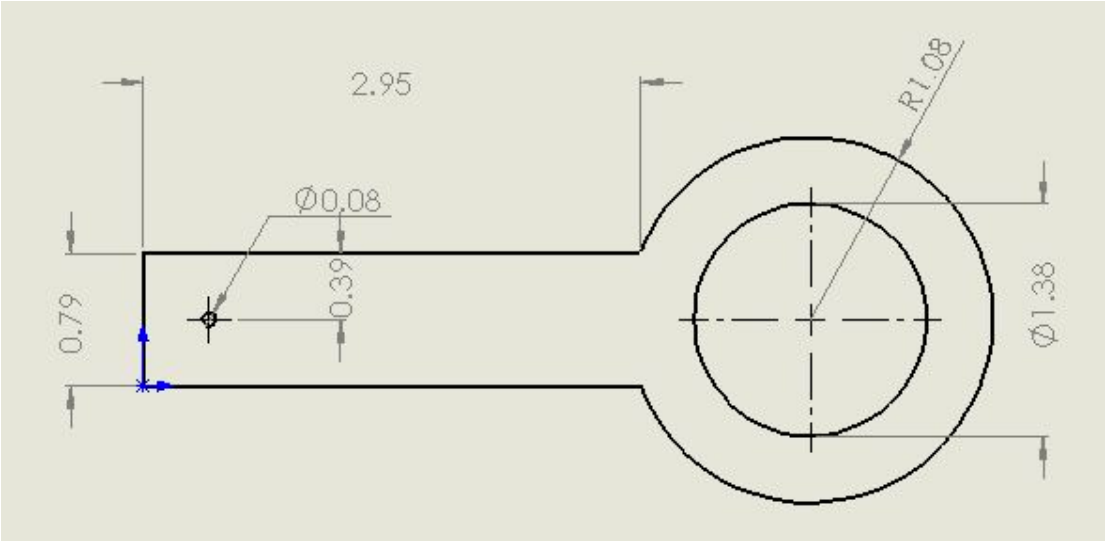
6.115 Final Project
Michael Everett
May 14, 2014



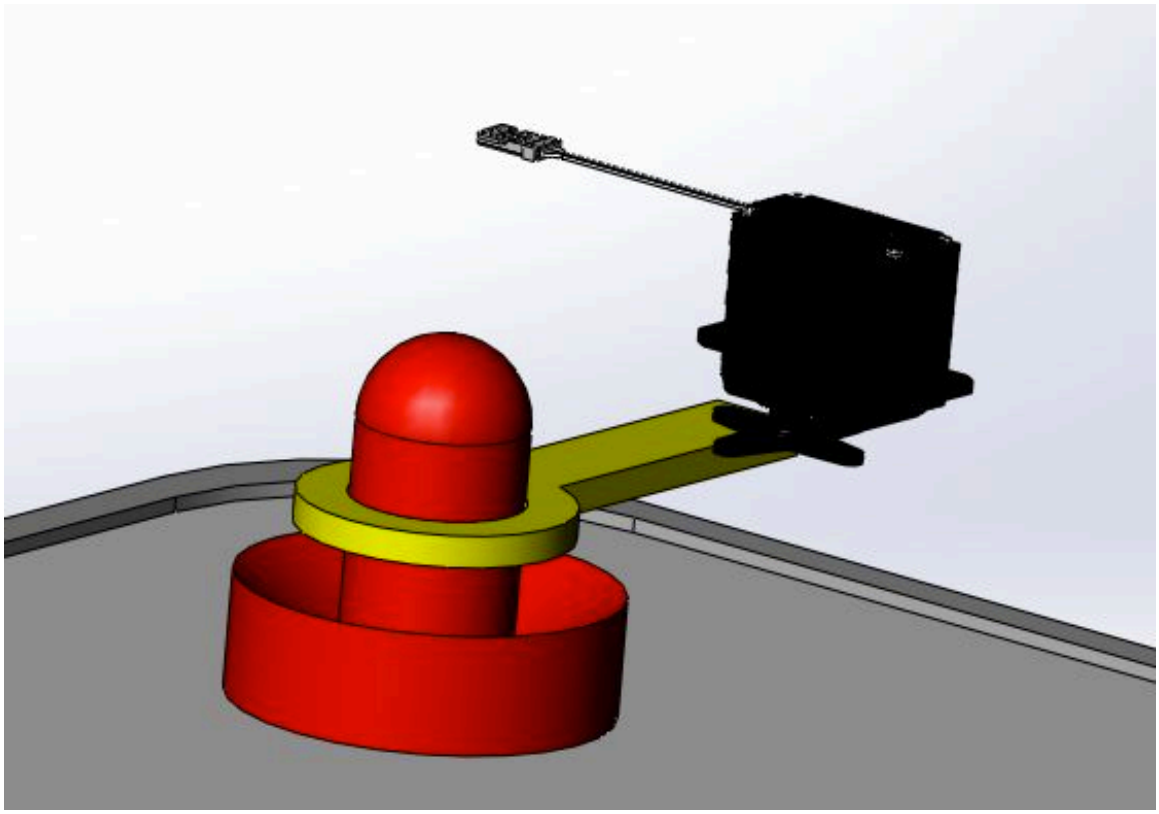
Hardware/Mechanical Schematics



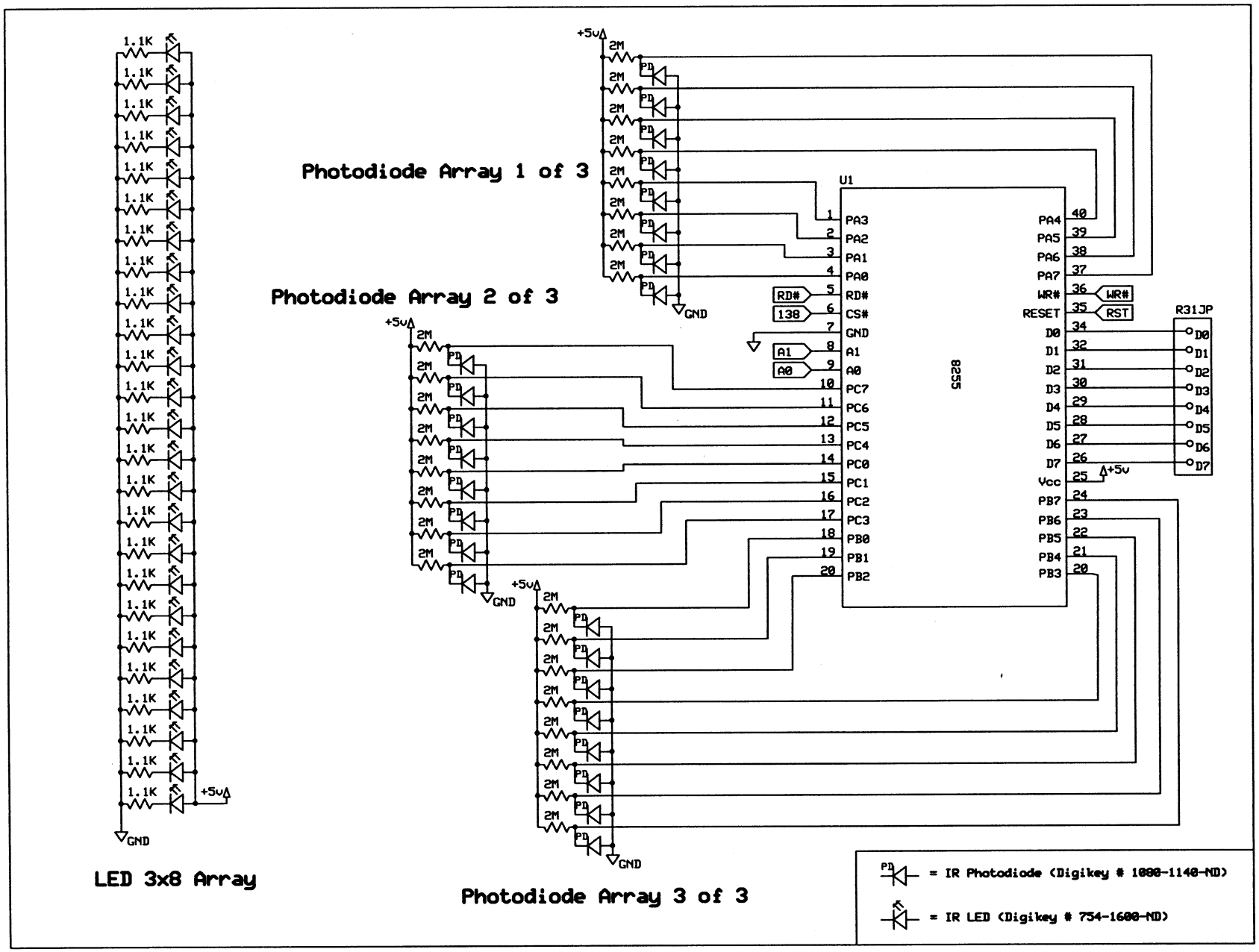
Laser Cut Drawing for Servo-Table Attachment (acrylic)



Laser Cut Drawing for Servo-Paddle Attachment (acrylic)



SolidWorks 3D Model of Servo Paddle Movement



MIT 6.115 Final Project		
Air Hockey Table Electronics		
Michael Everett	Rev 1.0 5/13/2014	Page # 1 of 1

E:\mfe_final_project_2014.sch - 8255+LM18293

Appendix A: Commented Code

```
/* ========================================
 *
 * Air Hockey Table: PSoC C Program
 *
 * Michael Everett
 * May 14, 2014
 * 6.115 Final Project
 *
 * This program is built using PSoC Creator 3.0
 * on the lab computers. Then it's programmed to
 * the PSoC 5.
 *
 * It contains code to read 4 serial inputs via
 * the UART and logically determine the puck's path.
 * Basically it assumes the puck will always travel in a
 * straight line (fair assumption) and uses the difference
 * in position 1&2, then 2&3 to guess. It then points the
 * servo to the proper position.
 *
 * I discretized the number of servo locations to the 8
 * photodiodes plus the 7 spaces in between, so 15 possible
 * locations.
 *
 * ========================================
*/

#include <device.h>
int count_num_senses = 0;
int current_step;
int current_pos = 1;
int desired_pos;
int delay = 400;
int pos_1;
int pos_2;
int pos_3;
int bad_byte = 0;
uint8 hi = 1;
uint8 lo = 0;
char8 star = 42;

void go_to_pos(int desired);
void servo_go_to_pos(int desired);
int calc_desired_pos(int a, int b);
int des_pos_1;
int des_pos_2;


CY_ISR(RX_INT)
{
      // RX_ISR: This function is called anytime the UART RX line
      // is triggered. It processes the serial data from the 8051.

      char8 byte = UART_ReadRxData();
```

```
      char8 rollover = 255;               // 8051 sends #0FFh if it's
done with a cycle


      // This section turns a 8255 reading and turns it into a
position,
      // so '08' => '7' and '03' => '2'
      char8 byte_pos = byte;
      char8 one = 1;
      char8 three = 3;

      int pos = 2;
      while ((byte_pos != three) && (pos < 16)) {
            // keep bitwise shifting until we get a 03h. This handles
            // the case where 2 diodes were triggered.
            // Even positions are in between diodes (i.e. 1,3,5 =
diodes 1,2,3)
            byte_pos = byte_pos >> 1;
            pos = pos + 2;
      }
      if (pos > 14) {
            // if the byte never equaled 3, it means only one bit
            // was set so that means the puck was only under one diode.
            // this is the most common scenario.
            pos = 1;
            byte_pos = byte;
            while ((byte_pos != one) && (pos < 16)) {
                  byte_pos = byte_pos >> 1;
                  pos = pos + 2;
            }
      }



      // This is when the puck bounces back toward the human,
      // so we want to reset our prediction data.
      if (byte == rollover) {
            count_num_senses = 0;
            LCD_ClearDisplay();
      }

      // A byte that corresponds to position 1,2, or 3:
      else {
            if (count_num_senses == 0) {
                  LCD_Position(0,0);
                  if (pos < 16) { pos_1 = pos; }
                  // if pos > 15, there was a problem.
            }
            else if (count_num_senses == 1) {
                  LCD_Position(0,3);
                  if (pos < 16) { pos_2 = pos; }
                  des_pos_1 = calc_desired_pos(pos_1,pos_2);
                  servo_go_to_pos(des_pos_1);
                  // rather than waiting for all 3 data points,
                  // we can assume no bounce and if
                  // necessary, we'll correct it when we get the 3rd
                  // position.
```

```
            }
            else if (count_num_senses == 2) {
                  LCD_Position(0,6);
                  if (pos < 16) { pos_3 = pos; }
            }


            // this prints the byte on the LCD screen. It's equivalent
            // to running 'prthex' on the 8051 instead of 'sndchr'
            // but way easier to write in C than Assembly.
            char8 high_byte = (byte >> 4) + 48;
            char8 low_byte = (byte & 0b00001111) + 48;
            LCD_PutChar(high_byte);
            LCD_PutChar(low_byte);


            // this section is mostly for debugging. It prints the 3
positions
            // (translated from byte to 1-15 already) and the two
predicted
            // positions to ensure the motor is doing the right stuff.
            if (count_num_senses == 2) {
                  char8 pos_1_ascii = pos_1 + 48;     // 48+ a number =
Ascii code
                  char8 pos_2_ascii = pos_2 + 48;     // e.g. 48+5 = 53
= 0x35 = '5'
                  char8 pos_3_ascii = pos_3 + 48;
                  des_pos_2 = calc_desired_pos(pos_2,pos_3);
                  char8 des_pos_1_ascii = des_pos_1 + 48;
                  char8 des_pos_2_ascii = des_pos_2 + 48;
                  LCD_Position(0,9);
                  LCD_PutChar(pos_1_ascii);
                  LCD_PutChar(pos_2_ascii);
                  LCD_PutChar(pos_3_ascii);
                  LCD_Position(1,0);
                  LCD_PutChar(des_pos_1_ascii);
                  LCD_PutChar(des_pos_2_ascii);
                  //go_to_pos(des_pos);

                  servo_go_to_pos(des_pos_2);
                  // this accounts for bouncing
            }


            count_num_senses = count_num_senses+1;


      }
}

int calc_desired_pos(int a, int b) {
      // given two positions, it finds the predicted next position.
      // delta = b-a.
      // b + delta = new position = 2b-a
      int pred = (2*b)-a;

      // if it's out of bounds, assume the bounced back as far as it
```

```
	// went out of bounds.
	if (pred > 15) { pred = 30 - pred; }
	if (pred < 1) { pred = 2 - pred; }
	return pred;
}

void servo_go_to_pos(int pos) {
	uint16 compare1;
	// all of these constants are somewhat experimentally
	// determined, but correspond to the number of machine
	// cycles for the PWM duty cycle.
	// the total count = 50,000 (= 5ms period)
	if (pos == 1) { compare1 = 14000; }
	else if (pos == 2)  { compare1 = 15500; }
	else if (pos == 3) { compare1 = 17000; }
	else if (pos == 4) { compare1 = 18000; }
	else if (pos == 5) { compare1 = 19000; }
	else if (pos == 6) { compare1 = 20000; }
	else if (pos == 7) { compare1 = 21000; }
	else if (pos == 8) { compare1 = 21500; }
	else if (pos == 9) { compare1 = 22000; }
	else if (pos == 10) { compare1 = 22500; }
	else if (pos == 11) { compare1 = 23000; }
	else if (pos == 12) { compare1 = 23500; }
	else if (pos == 13) { compare1 = 24000; }
	else if (pos == 14) { compare1 = 25000; }
	else if (pos == 15) { compare1 = 26000; }

	PWM_1_WriteCompare(compare1);
	// this command overwrites the 'compare1' value using the
	// PWM API.
}

void stepper_motor(int position) {
	if (position == 0) {
		Red_Write(hi);
		Yellow_Write(lo);
		Orange_Write(lo);
		Green_Write(hi);
		Black_Write(lo);
		White_Write(hi);
		CyDelay(delay);
	}
	else if (position == 1) {
		Red_Write(hi);
		Yellow_Write(lo);
		Orange_Write(hi);
		Green_Write(hi);
		Black_Write(lo);
		White_Write(lo);
		CyDelay(delay);
	}
	else if (position == 2) {
		Red_Write(hi);
		Yellow_Write(lo);
		Orange_Write(lo);
		Green_Write(hi);
```

```
            Black_Write(hi);
            White_Write(lo);
            CyDelay(delay);
      }
      else if (position == 3) {
            Red_Write(hi);
            Yellow_Write(hi);
            Orange_Write(lo);
            Green_Write(hi);
            Black_Write(lo);
            White_Write(lo);
            CyDelay(delay);
      }
      current_step = position;
}

void spin_init() {
      stepper_motor(0);
}

void spin_forever() {
      while (1) {


            Red_Write(hi);
            Yellow_Write(lo);
            Orange_Write(lo);
            Green_Write(hi);
            Black_Write(lo);
            White_Write(hi);
            CyDelay(delay);

            Red_Write(hi);
            Yellow_Write(lo);
            Orange_Write(hi);
            Green_Write(hi);
            Black_Write(lo);
            White_Write(lo);
            CyDelay(delay);

            Red_Write(hi);
            Yellow_Write(lo);
            Orange_Write(lo);
            Green_Write(hi);
            Black_Write(hi);
            White_Write(lo);
            CyDelay(delay);

            Red_Write(hi);
            Yellow_Write(hi);
            Orange_Write(lo);
            Green_Write(hi);
            Black_Write(lo);
            White_Write(lo);
            CyDelay(delay);

      }
```

```
}


void spin(int steps) {
      int steps_left = steps;
      int next_step;
      if (steps > 0) {
            while (steps_left != 0) {
                  if (current_step == 0) { next_step = 3; }
                  else { next_step = next_step - 1; }
                  stepper_motor(next_step);
                  steps_left--;
            }

//          while (steps_left != 0) {
//                if ((current_step == 0) && (steps_left != 0)) {
//                      stepper_motor(1);
//                      steps_left = steps_left+direction;
//                }
//                if ((current_step == 1)  && (steps_left != 0)) {
//                      stepper_motor(2);
//                      steps_left = steps_left+direction;
//                }
//                if ((current_step == 2) && (steps_left != 0)) {
//                      stepper_motor(3);
//                      steps_left = steps_left+direction;
//                }
//                if ((current_step == 3) && (steps_left != 0)) {
//                      stepper_motor(0);
//                      steps_left = steps_left+direction;
//                }
//          }
      }

      else if (steps < 0) {
            while (steps_left != 0) {
                  if (current_step == 3) { next_step = 0; }
                  else { next_step = next_step + 1; }
                  stepper_motor(next_step);
                  steps_left++;
            }
//          while (steps_left != 0) {
//                if ((current_step == 0) && (steps_left != 0)) {
//                      stepper_motor(3);
//                      steps_left = steps_left+direction;
//                }
//                if ((current_step == 3)  && (steps_left != 0)) {
//                      stepper_motor(2);
//                      steps_left = steps_left+direction;
//                }
//                if ((current_step == 2) && (steps_left != 0)) {
//                      stepper_motor(1);
//                      steps_left = steps_left+direction;
//                }
//                if ((current_step == 1) && (steps_left != 0)) {
```

```
//                        stepper_motor(0);
//                        steps_left = steps_left+direction;
//                }
//          }

      }
}

void go_to_pos(int desired) {
      int steps_needed = 4*(desired - current_pos);
      spin(steps_needed);
      current_pos = desired;
}







void main()
{
      LCD_Start();                                    // initialize lcd


    CyGlobalIntEnable;
    rx_int_StartEx(RX_INT);             // start RX interrupt (look for
CY_ISR with RX_INT address)
                                        // for code that writes
received bytes to LCD.

    UART_Start();                       // initialize UART
    UART_ClearRxBuffer();
      LCD_ClearDisplay();
      LCD_Position(0,0);

      Clock_1_Start();
      PWM_1_Start();

      //spin_init();


      while (1) {

      }




}




/* [] END OF FILE */
```

```
;=============================================================
; Air Hockey Table: Assembly Program
;
; Michael Everett
; May 14, 2014
; 6.115 Final Project
;
; This code is assembled into a .hex file and uploaded to the
; R31JP with SecureCRT via a serial cable, and begins running
; when the user moves the switch into RUN mode.
;
; The code sends 4 bytes to the PSoC each cycle, which is a hex
; value corresponding to the location of the photodiode which
; was triggered by the puck.
;=============================================================

.org 00h
ljmp main

.org 100h
main:                           ; To get started, initialize the
8255
   lcall init                   ; and Timer 1 for Serial
Communication
far:
      mov dptr, #0fe11h       ; Constantly check Port B to see if a
      movx a, @dptr             ; puck crossed under the furthest
away bar.
      jz far                          ; Once it does, send the
location and move on.
      lcall sndchr
      mov R0, a                 ; One of the LEDs died so this
stores point 1.

middle:
      mov dptr, #0fe12h         ; Similarly wait for Port C
      movx a, @dptr
      jz middle
      lcall sndchr              ; Upon puck, send 2nd point

      mov R1, a                 ; The dead LED is in bar 3, point
1, so
      cjne R0, #01h, close    ; if both bar 1 and bar 2 were point 1,
      cjne R1, #01h, close    ; we can assume bar 3 would have been
      mov a, #01h               ; also, and just send the
character.
      lcall sndchr
      ljmp middle2


close:                                  ; Similarly wait for Port A
      mov dptr, #0fe10h
      movx a, @dptr
      jz close
      lcall sndchr
```

```
middle2:
      mov dptr, #0fe11h             ; Now the puck has started bouncing
back
      movx a, @dptr                 ; so start worrying about its
return
      jz middle2
      lcall delay                   ; This delay is for 'debounce'
issues
      mov a, #0ffh                  ; Let the PSoC know a full cycle
has
      lcall sndchr                  ; occurred.
      ljmp far                      ; Start over!



;===============================================================
; subroutine init
; this routine initializes the hardware
; set up serial port with a 11.0592 MHz crystal,
; use timer 1 for 9600 baud serial communications
; and starts the 8255 as 24 input pins.
;===============================================================
init:
      mov   tmod, #20h       ; set timer 1 for auto reload - mode 2
      mov   tcon, #41h       ; run counter 1 and set edge trig ints
      mov   th1,  #0fdh      ; set 9600 baud with xtal=11.059mhz
      mov   scon, #50h       ; set serial control reg for 8 bit data
                             ; and mode 1
      mov dptr, #0fe13h          ; Initialize the 8255 for 24 input
pins.
      mov a, #9bh
      movx @dptr, a

    ret

;=============================================================
; subroutine sndchr
; this routine takes the chr in the acc and sends it out the
; serial port.
;=============================================================
sndchr:
   clr  scon.1            ; clear the tx  buffer full flag.
   mov  sbuf,a            ; put chr in sbuf
txloop:
   jnb  scon.1, txloop    ; wait till chr is sent
   ret

delay:
      mov R0, #0ffh
      mov R1, #0ffh
      loop2:
            djnz R1, loop2
      mov R1, #0ffh
      djnz R0, loop2
ret
```