EE 2361 Final Project: Chrome Dino Autobot

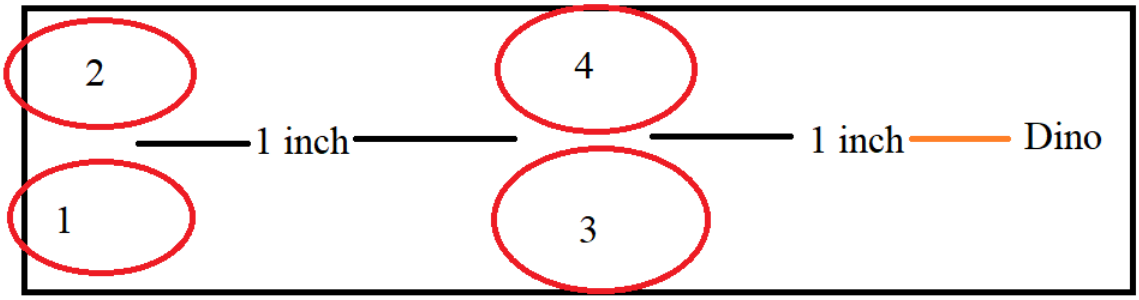
Spring 2020

Andrea Smith and Ferris Henning

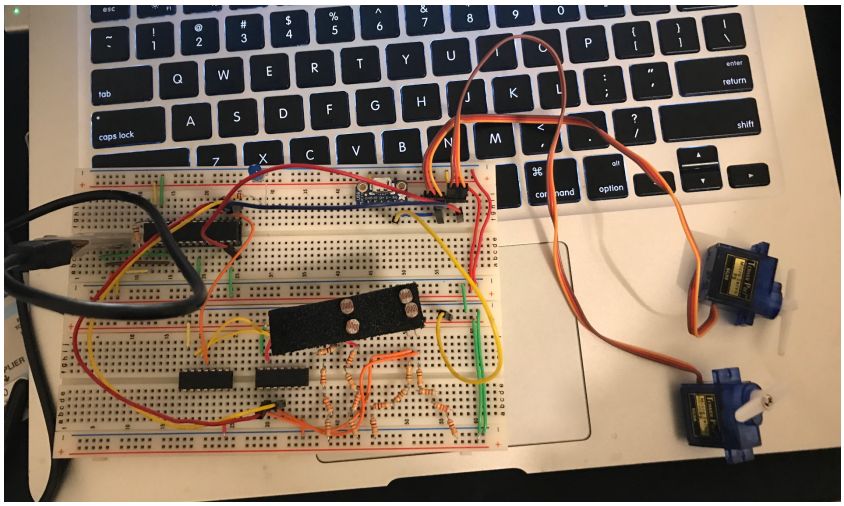
Intro:

This library is intended to work with a circuit with four photoresistors that enable interrupt service routines, timers, and servos to play Chrome's offline dinosaur game. It is inspired by an Arduino project that is only capable of having the dinosaur jump as a response to the photoresistor detecting an obstacle. [That project can be found here.]

The photoresistors are set up as follows:



Foam to hold the photoresistors



There is one servo that presses the spacebar to jump, and one servo that presses the down arrow key to duck. The two far left photoresistors initially detect obstacles, and there are three possible cases that must be considered to decide what the servos should do.

1. The bottom photoresistor fires

2. The top photoresistor fires, but not the bottom photoresistor
3. Both photoresistors fire

In the case where the bottom photoresistor goes off, the bot will always trigger the jump servo to press the spacebar, because no matter what, the dinosaur needs to jump over an obstacle. If only the top photoresistor fires, but the bottom does not, the obstacle is a bird and the dinosaur needs to duck. If both photoresistors fire, it is the same case as number 1, and the jump servo will trigger.

The dinosaur game's speed increases with time, so in order to jump or duck on time, a second set of photoresistors is needed. These "back" photoresistors use the timers triggered by the "front" photoresistors to calculate the speed of the obstacle and decide how quickly to trigger the servo so that the dinosaur clears the obstacle in time.

Hardware Description:

x1 PIC24FJ64GA002

x4 PDV-P8008 Photocell 80k-240k Ohm 5.1 mm photoresistor

x4 19k ohm resistor

x1 SN74HC86N (Logical XOR)

x1 SN74HC04N (Logical inverter)

x2 Tower Pro 9G Micro Servo

x1 USB Micro Header

Documentation:

```
void setup() {

    __builtin_write_OSCCONL(OSCCON & 0xbf); // unlock PPS
   RPINR0bits.INT1R = 8;
   RPINR1bits.INT2R = 9;
   RPOR3bits.RP6R = 18; // RP6 for OC1
   RPOR2bits.RP5R = 19;
   __builtin_write_OSCCONL(OSCCON | 0x40); // lock   PPS


   CLKDIVbits.RCDIV = 0;
   AD1PCFG = 0x9fff;
   TRISB = 0b111000000;
   TRISA = 0b0; // RA0 output
```

This portion of the setup() function sets the input and output pins. The input pins RP7 and RP8 receive the signals the top and bottom "front" photoresistors, which are first fed through a logical inverter to clean up the signal and get a crisp "jump" up. RP9 receives the signal from the "back" photoresistors, which are run through an XOR gate to combine the data due to a limited number of pins. This works because it does not matter whether the top or bottom photoresistor fires, either way the obstacle speed will be calculated.

```
INTCON2 = 0; // INTCON2 reg controls external interrupts
   IFS0bits.INT0IF = 0;

   // Set up ISR 0
   _INT0EP = 0; // External interrupt 0 occurs on rising edge
   _INT0IE = 1; // Enable

   // Set up ISR 1
   _INT1EP = 0;
   _INT1IE = 1; // Enable

   // Set up ISR 2
   _INT2EP = 0;
   _INT2IE = 1; // Enable
```

This portion of setup() simply initializes the three external interrupts used to decide whether to jump or duck. The interrupts all occur on the rising edge of the signal.

```
// Timer 1 setup
   T1CON = 0;
   TMR1 = 0;
   T1CONbits.TON = 0; // Off for now
   T1CONbits.TCKPS = 0b11; // 256 prescaler
   PR1 = 0;

   // Timer 2 setup
   T2CON = 0;
   TMR2 = 0;
   T2CONbits.TON = 0;
   T2CONbits.TCKPS = 0b11;
   PR2 = 0;

   // Timer 3 setup
   T3CON = 0;
   TMR3 = 0;
   T3CONbits.TON = 0;
   T3CONbits.TCKPS = 0b01;
   PR3 = 39999;
   _T3IF = 0;
   T3CONbits.TON = 1;
```

This portion of setup() initializes the Timers 1 to 3. Timer 1 and 2 both have a 256 prescaler so they can count to the largest possible value to avoid overflow. Timer 3 controls the servos, which have a 20 ms duty cycle, so the prescaler is the smallest clock division possible to give it the best accuracy possible.

```
void __attribute__((__interrupt__,__auto_psv__)) _INT0Interrupt(void)
{
   T1CONbits.TON=1;
   _INT0IF = 0;
   dinoAction=0;
   }
```

Interrupt 0 enables Timer 1, and sets the bot into “jump” mode by setting dinoAction to 0.

```
void __attribute__((__interrupt__,__auto_psv__)) _INT1Interrupt(void) // Duck
{
   if(T1CONbits.TON == 0){
      dinoAction=1;
   }
   T1CONbits.TON=1;
   _INT1IF = 0;
}
```

Interrupt 1 checks if Timer 1 is already on. If it is, the dinosaur must jump regardless, so it does nothing to allow the Interrupt 0 settings run and resets the interrupt flag.

```
void __attribute__((__interrupt__,__auto_psv__)) _INT2Interrupt(void) // When to jump
{
   if(T1CONbits.TON==1){
   PR2= TMR1 * DISTANCE_MULTY  - JUMP_OFFSET;
   T2CONbits.TON=1;
   T1CONbits.TON=0;
   TMR1 = 0;
   the_next_move=dinoAction;
   dinoAction=3;
   OC2RS = 3000 +  UNPRESSED_ANGLE;
   OC1RS = 3000 +  UNPRESSED_ANGLE;
   }
   _INT2IF = 0;
}
```

Interrupt 2 checks if Timer 1 is already on. If it is, it calculates the speed of the obstacles and decides how quickly the dinosaur action should happen by updating the PR2 value, and starting Timer 2.

```
void __attribute__((__interrupt__,__auto_psv__)) _T2Interrupt(void) // When to jump
{
   action(the_next_move);
   IFS0bits.T2IF = 0;
   the_next_move=3;

}
```

The Timer 2 interrupt is what actually triggers the servo movement by calling action() with the arguments determined by Interrupt 2, then resets the Timer 2 interrupt flag.

```
void action(int move){
   if(move==0){
      OC1RS = 3000 +  PRESSED_ANGLE; // Jump servo
   } else{
      OC2RS = 3000 +  PRESSED_ANGLE; // Duck servo
   }
}
```

The function action() takes an integer argument, *move*, that will either set the output on the OC1 register to jump, or on the OC2 register to duck.

```
int main(void) {
   setup_dino();
   while(1);
   return 1;
}
```

The main function only serves to call setup_dino().

Basic Usage Example

Basic usage of this library would be to use it as an object detection and timing calculator for the game. The User would then input jumps and ducks as the PIC outputted them to LEDs.

Advanced Usage Example

Advanced use would be to fully automate playing of the Dino game at higher seeps. This library can use the information it gathered by the photoresistors to determine the speed of the objects and modify when it would jump to anticipate the objects movement in front of the dinosaur. Theoretically this library can overflow the score counter.