# Collectorbot

Kramer STRAUBE & Nadia M'TARRAH

E115 Final Project Report

December 11th, 2009

## ABSTRACT

This project was to build a robot by using a Radio-Controlled (RC) car which was controllable through either an autonomous or a radio control mode. This robot could drive around, detect walls with a distance sensor and avoid them in autonomous mode. In radio control mode, the user could bump and object to grab it and deposit it into a container located on the back of the RC car, provided the object met certain weight and size criteria. This prototype was using the PIC, FPGA, two H-Bridges, a EZTEC Silverado 1:15 RC Car, one LEGO light sensor which was used for two functions : an ambient light and a distance sensor, one LEGO touch sensor, and two LEGO motors to actuate the two axes of the claw.

# INTRODUCTION

Radio-Controlled cars provide a cheap and widespread platform for robotic applications but they generally begin with only the low level capabilities of movement via radio control. We sought to add new features to the commercial RC car to demonstrate its worth as part of a robotic platform. The central added feature was the ability of the RC car to grab objects and deposit them into a container located on the top of the RC car. Another feature was to illuminate LEDs on the car body when the ambient lighting was sufficiently lacking. Lastly, the RC car was to be capable of moving autonomously using a distance sensor.

A LEGO light sensor is multiplexed as both an ambient light sensor and as a distance sensor (in conjunction with an LED in the sensor). The distance sensor operates by shining an LED on the objects and measuring the reflected light. This method only works well for very short distances (<30 centimeters) and has differing outputs based on the reflectivity of the object. When the distance sensor measures a certain threshold of reflected light in autonomous mode, the robot is programmed to turn. The ambient light sensor connects to the PIC so that when the ambient light is low enough, a control bit is sent to the FPGA which illuminates the LEDs.

A LEGO touch sensor is used to determine when the robot bumped into an object during radio control mode and activates the claw. The claw activation is controlled by the PIC and consists of four stages: closing the claw to grab the object, lifting the claw and object, opening the claw to drop the object in the container and dropping the claw back down into its original position.

## PIC Design

The PIC (PIC p18f452) is used for various functions in this project. First, the PIC takes in directional inputs and a stop input and uses simple logic to determine which RC Car motors to drive. The motors are driven through an H-Bridge SN 754410 to step up the voltage from 3.3 V out of the PIC to 6 V needed to properly drive the motors. The PIC also took a collection input that indicated that the claw collection routine should begin. When this input is high, the PIC uses a series of timer values and interrupts involving TMR0 to accurately perform the four stages of collection.

Another use for the PIC is to perform A/D conversions on the input from the analog LEGO light sensor. The LEGO light sensor is multiplexed as an ambient light sensor and a distance sensor by having the PIC control the LED on the LEGO light sensor. Then, the analog input to the PIC is converted to digital bits. These digital bits are either analyzed to determine whether the lighting is sufficient or the eight lowest bits are inverted and sent to the FPGA.

## FPGA Design

The FPGA is used to control mode selection, RC input, sensor analysis, and the autonomous mode. The FPGA takes in the input from a LEGO touch sensor to determine when an object is bumped and sends the appropriate signal to the PIC to initiate the collection routine. The FPGA controls which mode is being used (radio control or autonomous) by reading a value from one of the DIP switches on the HarrisBoard.

The radio control receiver module drives several inputs on the FPGA. The FPGA reads these values to determine appropriate actions. When in autonomous mode, if the ‘up’ radio control input is high, then the stop output is powered to make the car completely stop moving. In

radio control mode, the radio control receiver inputs are passed directly through to the PIC as directions.

The FPGA interprets the information sent to it by the PIC to decode what the sensors imply. The PIC drives one input on the FPGA which indicates whether there is sufficient ambient light. When there is not and the robot is in radio control mode, the FPGA illuminates LEDs on the robot. When the robot is in autonomous mode, the LEDs are always lit as an indicator of the autonomous mode. The PIC sends the distance information from the LEGO light sensor to the FPGA. If the value is considered to be low enough and the robot is in autonomous mode then FPGA will send the PIC the signal to turn the car for a given duration to avoid the wall.

The autonomous mode is controlled by the FPGA. The FPGA uses the distance inputs from the PIC to control when to turn. When the distance is large enough, the robot will just go forward. This makes for a simple left-turn algorithm.

The FPGA is used to run the autonomous mode of which the mode of selection is controlled by one of the dip switches on the board. Then the FPGA will output signals indicating the directions (forward, backwards, left and right) to move and the PIC will control the motors appropriately.

## New Hardware

Six new pieces of hardware are used in the design and implementation of this robot: the radio-control receiver, the RC car motors, the LEGO motors, the LEGO touch sensor, the LEGO light sensor and the H-Bridge.

The radio-control receiver is treated as an input bus to the FPGA. These inputs need to be connected to pull-down resistors to prevent charge from staying on the wire and holding inputs high. The power for all of the hardware except the LEGO motors comes from the 6 V of AA batteries in the RC car body and is pulled from a pin on the radio-control receiver.

The RC car motors are controlled through an H-Bridge to step up the voltages from the 3.3 V output from the PIC to the 6 V required. These motors are 6 V DC motors. The RC car motors only need to be connected at the two terminals for forward voltage and reverse voltage. They are controlled at either full forward voltage, full reverse voltage or no voltage. The direction and magnitude of the voltage applied control the direction and magnitude of the speed of the wheel.

The LEGO motors are driven by an output from the PIC that runs through an H-Bridge to step the voltage up from 3.3 V to 9 V needed to run the motors. The LEGO motors are PWM motors with shaft encoders but in this design are only connected at the two voltage terminals. Like the DC motors, the direction and magnitude of the voltage applied control the direction and magnitude of the speed of the driveshaft. In this design, the LEGO motors are only run at full voltage, in forward or reverse, or at no voltage.

The LEGO touch sensor acts like a switch and is connected in a voltage divider circuit to the PIC. When the sensor is depressed, the switch is connected and the PIC input is high. Otherwise, the input is low.

The LEGO light sensor is connected with a ground, a 4.3 V logic input, an analog output with a 10 kOhm pull-up resistor and an input pin to control the light. The 4.3 V logic input is used to determine whether the input pin is high or low. The input pin is connected to a PIC output. When this pin is high, the light is on. Otherwise, the light is off. The analog output is connected to the PIC on pin RA0 where an A/D conversion occurs inside the PIC. This output is larger when the light sensor gets less light and smaller in the absence of light.

The H-Bridge is a more complicated piece of hardware. The H-bridge controls up to 4 outputs based on one input per output and one enable per two outputs. The H-Bridge also takes two voltages in: 6 V, to use for logic comparisons with the inputs and enables, and 6 V or 9 V depending on which H-Bridge is under inspections. The 6 V source is used to draw more current from the batteries so the PIC does not directly drive the motors at its lesser voltage and lower current capabilities. The 9 V source is used to drive the LEGO motors. The 3.3 V outputs from the PIC are read as high values on the H-Bridge with a logic voltage of 6 V. The specific H-Bridge in use is the SN754410. When the enable is high, the output will equal the input. However, the enable and input are considered high at the logic voltage (VCC1) while the output is driven high at the output voltage (VCC2). Now since the motors are connected with a voltage terminal on two different H-Bridge outputs, the output pairs can cause specific behaviors. If both outputs are low, then the motor can freely spin. If one output is high and the other is low, the

motor spins based on that output. If both outputs are high, then the motor resists any movement, effectively acting like a brake.

## Mechanical Design

The mechanical design of the robot proved quite difficult. Two different sensors burnt out during the construction (the Sharp GP2D12 and the Kemo B045 Photo Sensor Kit). This forced us to use the LEGO light sensor and to multiplex it. Further difficulties were found when constructing the claw. The claw needed the ability to apply large forces on the inside of the claw and to be lifted effectively. It also needed to be able to grip objects. The claw was made from LEGOs but was connected in key places many times over to ensure rigidity. The inside of the claw was covered with sandpaper to provide grip. The container was simply the aesthetic plastic car shape, which we removed from the RC car, turned over and connected to the car body. Many pull-down resistors were added because they removed many issues with motors such as continues inputs or H-Bridge inputs being forced a voltage when they were off.

## Results

The entire design functioned on the bench over many test runs. However, once the pieces were combined on the car body, the PIC did not function properly and seemed to enter an endless loop when placed into a ‘run’ mode. When in an ‘animate’ mode, the PIC would run but interrupts and loading timer values did not work. Since this began

# Schematics/Block Diagram

The schematic is very hectic due to the large number of inputs and outputs. The inputs and outputs are labeled on the HarrisBoard and other larger hardware pieces.

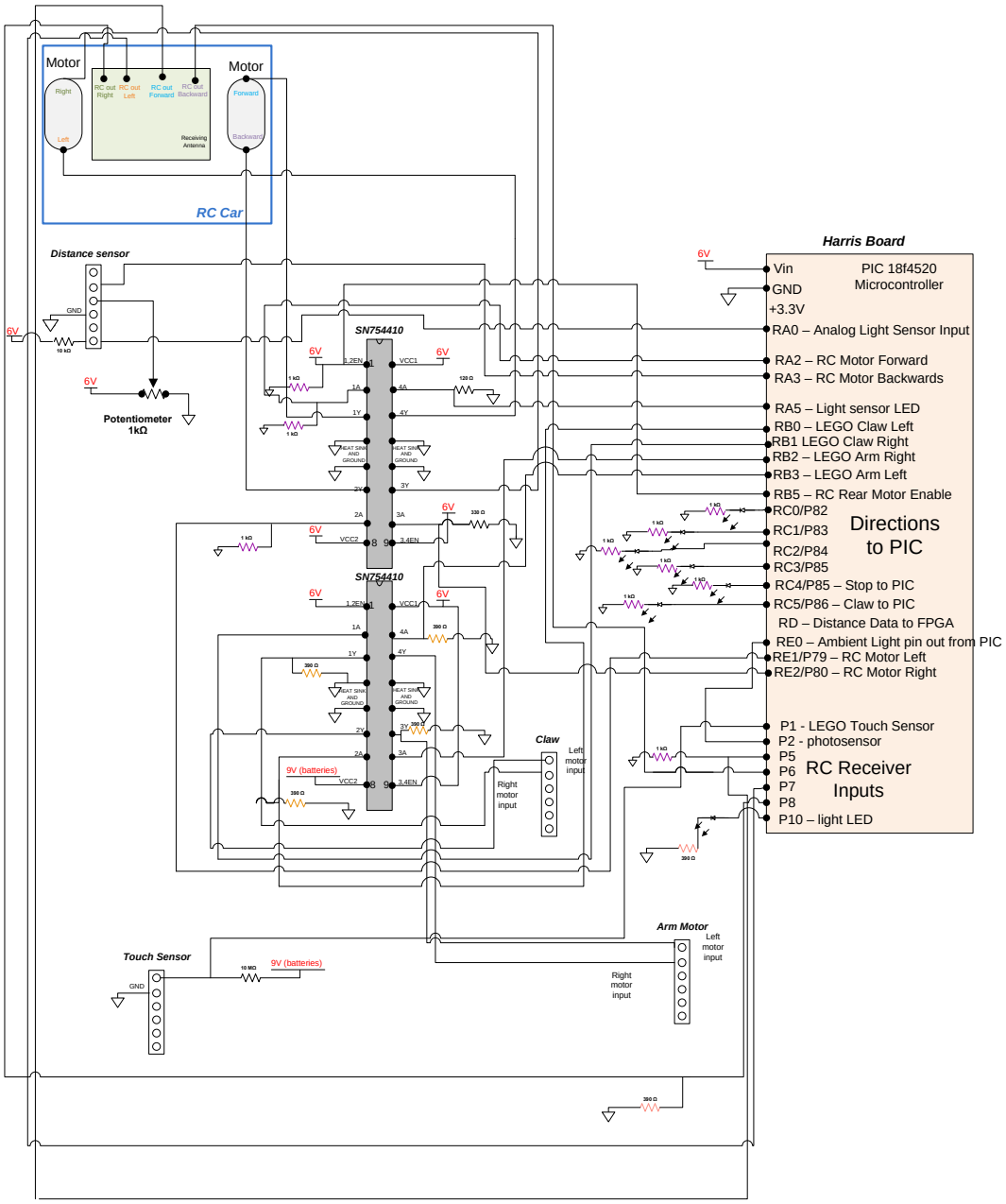


***Figure 1.*** *Schematics of the breadboard*

# References

*Sn754410*. (n.d.). Retrieved from http://www3.hmc.edu/~harris/class/e155/sn754410.pdf

*Mindstorms*. (n.d.). Retrieved from http://mindstorms.lego.com/en-us/Default.aspx

Evans, Tony and James Steele. "Radio-Controlled Tank" Retrieved from http://www3.hmc.edu/~sharris/class/e155/Projects_2007/RCTank.pdf

# Parts List

| Item Name | Source | Cost |
|---|---|---|
| 1 EZTEC Chevy Silverado 1:19 RC (Blue) (Model #: 21119) | Radioshack | $19.99 |
| 2 LEGO Motors + cables | Stockroom | - |
| 2 H-Bridges (*SN754410*) | Lab | - |
| 1 LEGO NXT Distance/Light Sensor | Stockroom | - |
| 1 HarrisBoard *(PIC 18F4520)* | Lab | - |
| 1 Operational Amplifier (*UA471*) | Lab | - |
| 1 LEGO NXT pieces kit | Stockroom | - |
| 7 LEDs | Lab | - |
| 1 LEGO touch sensor + cable | Stockroom | - |
| 11 Resistors of 1k**Ω** | Lab | - |
| 7 Resistors of 390**Ω** | Lab | - |
| 1 Resistor of 10M**Ω** | Lab | - |
| 1 Resistor of 30k**Ω** | Lab | - |
| 1 Resistor of 46k**Ω** | Lab | - |
| 1 Resistor of 7k**Ω** | Lab | - |
| 1 Potentiometer 1k**Ω** | Lab | - |
| Duct Tape | Kramer Straube | - |
| Cable Ties | Kramer Straube | - |

# Appendix

## PIC Code

```
#include <p18f452.h>


 /**
 Kramer Straube and Nadia M'Tarrah
 MicroP's Final Project PIC code
 12/8/2009

 Function: This code takes in inputs from the FPGA and interprets those
  into appropriate motor movements. It also performs A/D conversion
  on a light sensor which is also used with a light as a distance sensor.

 Notes:
    TMR0 values were found through testing and readjusting
    Interrupts likely failing due to RF interference (also causes claw to fail)
    Only works in Animate in Debug mode in nearly all functionality
    Run allows for RC car motor running but not much else
 **/
/** method prototypes **/
void main(void);
void claw(void);
void isr(void);
void setupAD1(void);
void setupAD2(void);
char stage;
char ADsource;

#pragma code high_vector = 0x08
void high_interrupt(void){
_asm
GOTO isr
_endasm
}

#pragma code
#pragma interrupt isr
```

```
void isr(void){                         //handles all the interrupts
   INTCONbits.GIE = 1;                    //reset the global interrupt
if (INTCONbits.TMR0IF == 1){
   INTCONbits.TMR0IF = 0;                 //clear the flag
                              //based on cycle number set the outputs accordingly
   if (stage == 1){
      TMR0H = 0x5A;
      PORTBbits.RB0 = 0;                  //stop closing claw
      PORTBbits.RB2 = 1;                  //raise arm
      TMR0L = 0x97;                       //set tmr0 value
   }
   if (stage == 2){
      TMR0H = 0xE8;
      PORTBbits.RB2 = 0;                  //turn off raise arm
      PORTBbits.RB1 = 1;                  //open claw
      TMR0L = 0x97;                       //set tmr0 value
   }

   if (stage == 3){
      TMR0H = 0x5A;
      PORTBbits.RB1 = 0;                  //turn off the open claw
      PORTBbits.RB3 = 1;                  //drop the arm down
      TMR0L = 0x97;                       //set tmr0 value
   }
   if (stage == 4){

      PORTB= 0;                          //clear portB
   }
   TMR0H = 0xFF;
   TMR0L = 0xFF;
   stage = stage +1;                      //increment the stage
}

   if (PIR1bits.ADIF == 1){
      PIR1bits.ADIF = 0;    //clear the flag
      if (ADsource){
                              //move the A/D bits needed to the PIC output pins
         PORTD = 0xFF-ADRESL;             //output the distance value to PORTD
         ADsource = 0;                 //set to ambient light mode
         PORTAbits.RA1 = 0;              //turn off the light
```

```
        }
        else{
          if(ADRESL<0xF0){
            PORTEbits.RE0 = 0;
          }
          else{
            PORTEbits.RE0 = 1;
          }
          //PORTEbits.RE0 = !();    //output whether the light is sufficient or not
          PORTEbits.RE0 = ((&ADRESH)[6]||(&ADRESH)[7]);//send an ADbit to FPGA
          ADsource = 1;               //set the mode to distance sensor
          PORTAbits.RA1 = 1;            //turn on the light
          //setupAD1();
        }
        //ADCON0bits.GO = 1;              //turn A/D back on
    }
}

#pragma code
void main(void){
    TRISB = 0;                      //Preparing all of the ports
    TRISD = 0;                      //prepare PORTD
    PORTD = 0;                      //clear PORTD
    TRISC = 0xFF;
    TRISA = 1;
    TRISE = 0;
    PORTB = 0;
    LATB = 0;
    INTCON = 0;                     //clear this first because we had problems setting it
    T0CON = 0b11000111;                 //need to set this first to enable 16 bit

    PIR1 = 0b00000000;                  //all zeroes to clear all peripheral flags
    PIE1 = 0b01000000;                  //0 (interrupt not needed), 1 (A/D interrupt set)
    IPR1 = 0b01000000;                  //the only one is making the A/D interrupt priority
high
    RCONbits.IPEN = 1;                  //Allows low priority interrupts (just in case)
    INTCON = 0xE0;                  //111 (GIE, PEIE,TMR0IE) set interrupts up, rest 0
    INTCON2 = 0xF4;                     //1(PORTB pullups off), 111(interrupts on rising
edge),
                            //0(NC),1(TMR0 interrupt high priority),
```

```
                        //00 (NC, No PORTB interrupt poriority)

  T0CON = 0b10000111;           //This sets up a 16bit TMR0 with a 256 prescale
  setupAD1();                   //setup the A/D Controls
  ADsource = 0;                 //start with the ambient light sensor
  stage = 5;                    //make the stage beyond what I care about in case
                                //it goes through the isr for the A/D
  //ADCON0bits.GO = 1;                  //Start the A/D
                                //FPGA comes in on PORTC
                                //A/D in is RA0
  while (1){
    if (PORTCbits.RC4 == 0){            //stop bit implemented
                                //swap pins
      PORTBbits.RB5 = 1;            //enable these motors
      PORTAbits.RA2 = PORTCbits.RC0;    //drive the RCcar motors
      PORTEbits.RE1 = PORTCbits.RC1;
      PORTEbits.RE2 = PORTCbits.RC3;
      PORTAbits.RA5 = PORTCbits.RC2;}
    else{
                                //drive motors = 0;
      PORTAbits.RA2 = 0;            //Stop motors
      PORTEbits.RE1 = 0;
      PORTEbits.RE2 = 0;
      PORTAbits.RA5 = 0;
    }

    if (ADCON0bits.GO == 0){
      ADCON0bits.GO = 1;
    }
      //ADCON0bits.GO = 1;                //make sure A/D is still running
      if(PORTCbits.RC5){                //if I am told to collect
        claw();                     //run the claw
      }

      if (PIR1bits.ADIF == 1){
        isr();                      //catch failing interrupts...
      }
  }
}
```

```
//subroutine to control the claw lifting motion and timing
void claw(void){
   INTCONbits.TMR0IF = 0;
   TMR0H = 0xE8;
                              //load and run timer0
   stage = 1;                       //set the stage number
   TMR0L = 0x97;
   //TMR0H = 0xCA;
   T0CONbits.TMR0ON = 1;                  //makes usre the timer is on
                              //set initial outputs for LEGO motors
   PORTBbits.RB0 = 1;                  //start closing the claw
   while(stage<5){                  //after the 4 stages the claw actuation is done
                              //wait as interrupts do the work detailed below
      if(INTCONbits.TMR0IF == 1){            //extra catch because interrupts don't seem to
work well
         isr();
      }
   }
   //T0CONbits.TMR0ON = 0;
   PORTB = 0;                        //clear portB afterwards
   //PORTBbits.RB5 = 0;
}

//sets up the A/D control registers
void setupAD1(void){
   ADCON0 = 0b11000001;                  //11(use F rc), 000 (use RA0), 001(no GO, NC,
Power on)
   ADCON1 = 0b10001110;                   //1(right justify),0(use F rc),00(NC),
                              //1110 (RA0 is only analog input and V+ = Vcc, V- = Gnd,
                              // rest of PORTA is digital I/O)
}
```

# FPGA Code:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Kramer Straube and Nadia M'Tarrah
//
// Create Date:    22:00:57 12/04/2009
// Design Name:
// Module Name:    code_ks
// Project Name:
// Target Devices:
// Tool versions:
// Description: This code takes in some sensor data from both switches, sensors,
//     the RC controller and the PIC. It outputs the appropriate directions to move
//     to the PIC along with a stop bit (for stop overrides) and a claw collection
//       bit. It also illuminates LEDs when appropriate.
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module code_ks(
   input clk,
   input [7:0] distance,
   input [3:0] rccommands,
   input aiSwitch,
   input dark,
   input bump,
    input fakeAI,
   output reg [3:0] directions,
   output reg stop,
   output reg leds,
   output reg collect
   );
```

```
reg [7:0] turningCounter;                    //8 bit counter to help with
                                             //turning for a certain time

always @(*)                                  //combinational logic for
                                             //collection,the stop command from remote
                                             //and non-AI movement
begin
if (aiSwitch)
   begin
   //directions = 4b'1001; //to be fixed for AI
   //leds = 1;
   leds = dark;                              //when I get the dark indicator
                                             //from the PIC, turn on the LEDS
   if (rccommands[0])                        //if a specific RC command is read
      stop = 1;                              //make the PIC stop driving
                                             //the RCcar motors
   else
      stop = 0;                              //else just let it go

   if (bump)                                 //if the touch sensor is not hit
      collect = 0;                           //send the PIC a 0 for claw
   else
      collect = 1;                           //otherwise tell it to pick the item up

   end
else
   begin
   stop = 0;                                 //don't send a stop bit
   leds = dark;                              //when I get the dark indicator
                                             //from the PIC,
                                             //turn on the LEDS
   if (bump)                                 //if the touch sensor is not hit
   collect = 0;                              //send the PIC a 0 for claw
   else
   collect = 1;                              //otherwise tell it to pick
                                             //the item up
   end

end
```

```
always @(posedge clk)                          //use sequential for AI
                                              //movement logic
  begin
    if (aiSwitch)                              //if I am in AImode
    begin
                                              //simple turn-left before walls AI
      if (distance > 8'h08 | | fakeAI)          //if my distance sensor sees a
                                              //wall (or the fake input is used)
        begin
        turningCounter = 8'hFF;               //setup counter for turning time
                                              //(may need to adjust)
        directions = 4'b1001;                 //turn left
        end
      else
        begin
        if (turningCounter == 0)              //if I am not turning
          begin
          turningCounter = 0;                 //make sure the counter stays @ 0
          directions = 4'b0001;               //just go forward
          end
        else
          begin
          turningCounter = turningCounter -1; //decrement the counter
          directions = 4'b1001;               //keep turning left to assure
                                              //no wall side scrapes (kinda)
          end
        end
    end
    else
    begin
      directions = rccommands;                //otherwise pass the commands
                                              //through
    end
  end
endmodule
```