RC Controller

Final Project Report

December 12, 2002

E155

Gene Lee & Elizabeth Lee-Su

Abstract

Often times one might want to resurrect RC toys with broken or lost transmitters, or to enhance their functionality. The goal of this project is to control a RC toy by creating a wireless RC controller. This can be done by utilizing wireless transmitter/receiver modules and sending the appropriate signals to drive the motors inside the toy. This project will control a RC Nissan XTerra toy car, while enhancing its steering by using a servo motor instead of the built in dc motor to drive its steering mechanism. Digital logic and the electronic parts of this project are emphasized, while the mechanical parts are present only to implement the project design.

1. Introduction

This final project is aimed to create a RC controller that can control any given RC toy. For the purposes of the project, a RC Nissan XTerra toy car was bought. The user will input a command which will then be transmitted to the car. The car will receive the command, and then the motors inside will respond accordingly. This project seeks to enhance the steering of the car by replacing the dc motor in the steering mechanism with a servo motor. As a result, not only will the car be remotely controlled, but its performance will be improved as well.

2. New Hardware

2. 1 Servo Motor

The Futuba FP-S148 servo motor is used for the steering in this prototype. Servo motors are controlled via a pulse-width-modulation (PWM) signal. The S148 takes in 3 inputs: power, ground, and a control signal. An example of a PWM signal is shown in Figure 1. The high time of the PWM signal determines the direction that the motor will turn.

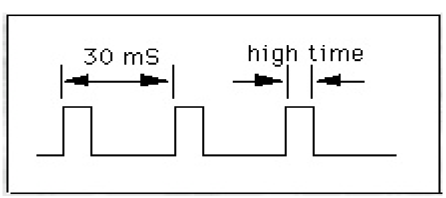

Figure 1: PWM Signal

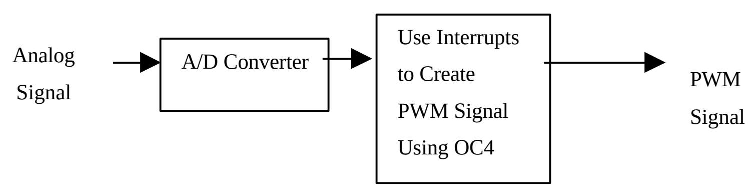

Figure 2: Block Diagram for HC11 Program

In order to determine the high times of PWM signals that correspond to motor movements, a program (see Appendix A) was written for the Motorola M68HC11 microcontroller using interrupts to pulse-width-modulate port A bit 4 using output compare OC4. The period of the PWM output is 30 ms long, which works well with the S148 and is typical of most servo motors. An analog voltage is read at PE7 which then goes through an analog to digital converter in which the result (between 0 and 256) will be the high time of the PWM signal. The user can then send PWM signals to the S148 and see how it reacts. A block diagram of the HC11 program is shown on Figure 2. By sending the servo a variety of PWM signals, the user can see which signals stimulate specific movements. The test results for the S148 are shown in Table 1. Although the S148 can be driven to other positions, for the purposes of this project, only the positions below will be considered.

High-Pulse (mS)	0.1	0.65	1.2	1.75	2.3
Position	-90	-45	0	45	90

Table 1: High Times Corresponding to Servo Positions

2.2. DC Motor

The DC motor that came with the car will be used to control forward and backward movements. The DC motor takes as input two signals: power and ground. If power and ground are reversed, the motor will turn CCW, otherwise, the motor will turn CW. In order to have the motor turn in reversible directions, a reversible drive circuit is needed.

The H-bridge is a simple, reversible drive circuit (see Figure 3) that will be used in this project to drive the DC motor forward and backwards. A basic H-Bridge has 4 switches, transistors, or other means of completing a circuit to drive a motor. In the above diagram, the switches are labeled A1, A2, B1 and B2. Since each of the four switches can be either open or closed, there are $2^4 = 16$ combinations of switch settings. The combinations of switch settings relevant to this project are shown in Figure 4.

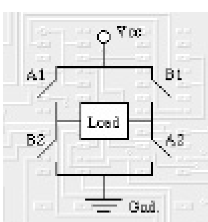

Figure 3: Connection Diagram for L293D Motor Driver

Closed switches	Polarity	Effect
A1 & A2	forward	motor spins forward
B1 & B2	reverse	motor spins backward
None	free	motor floats freely

Table 2: Table for H-bridge Switches

H-bridges are so common and useful that a number of commercially available integrated circuits (ICs) are made that combine all of the discrete components together. The IC that will be used for this project is the L293D (see Figures 5 and 6) from SGS-Thomson Microelectronics.

The L293 is a four channel motor driver IC that can be used for simultaneous, bi-directional control of two small motors. This IC can be used as 2 H-bridges, but for this project only one DC motor will be controlled, so pins 10-15 will be left empty (they are able to float). The L293 is limited to an output of 600 mA per channel and comes in a standard 16-pin, dual-in line integrated circuit package.

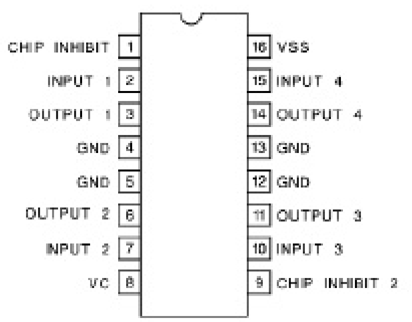

Figure 5: Diagram and Pin Placement for L293D

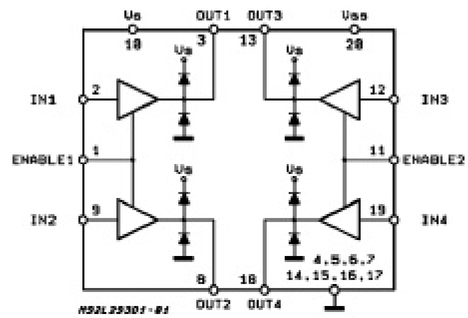

Figure 6: Schematics of L293D

2.3 Transmitter/Receiver Module

Any hope of salvaging the wireless transmitter and receiver that came with the car to use for control purposes, was lost when their circuitry could not be followed. Both transmitter and receiver were extremely compact and consisted of numerous capacitors, resistors, transistors, and an encoding/decoding “SuperChip” IC. Signals were observed from both the transmitter and receiver on an oscilloscope when controlling the car, but sense could not be made out of the waveforms. For this reason, wireless transmitters and receivers were purchased and the FPGA was used for encoding/decoding signals.

Research and budget constraints led to the purchase of the TXM-433-LC radio frequency (RF) transmitter and the RXM-433-LC-S RF receiver (Figure 7), which operate at 434 MHz and feature a transmission range in excess of 300 feet. A digital signal sent to the transmitter is modulated to be transmitted at the carrier frequency of 434 MHz. This signal is received by the receiver and then demodulated to the original signal. Below are schematics (Figure 11) of the transmitter and receiver with power supply noise filter circuits connected to V_{cc}. Eight digital signals with varying high times were encoded to correspond to eight different motor directions that the user could input via keypad. A transmitter FPGA would create the encoded signal depending on what key was inputted, and then output the signal to the transmitter. A receiver FPGA mounted on the car would receive the signal, and then decode it and send the motors the respective signals. The transmitter/receiver modules were tested, and it was discovered that as the transmitter and receiver were further apart, the high times transmitted were shorter. The cause of this is probably due to noise encountered during transmitting. That problem was accounted for by having a range of high times that corresponded to each command instead one specific value. For example, if a signal with a high time of 200 μS was sent, then the receiver would recognize a signal with a high time of 100-250 μS as 200 μS. With the new ranges of high times (see Table 3), signals can be transmitted to the car within the micro processor's lab and even out in the hallway, but as previously mentioned, the clarity of the signals are lost as the transmitter and receiver are further apart. Since this is a prototype, an economical transmitter/receiver module is used instead of a higher quality one that will not catch as much noise. For the purposes of this project, this pair of transmitter and receiver modules operated with reasonable errors that were able to be accounted within the receiver FPGA.

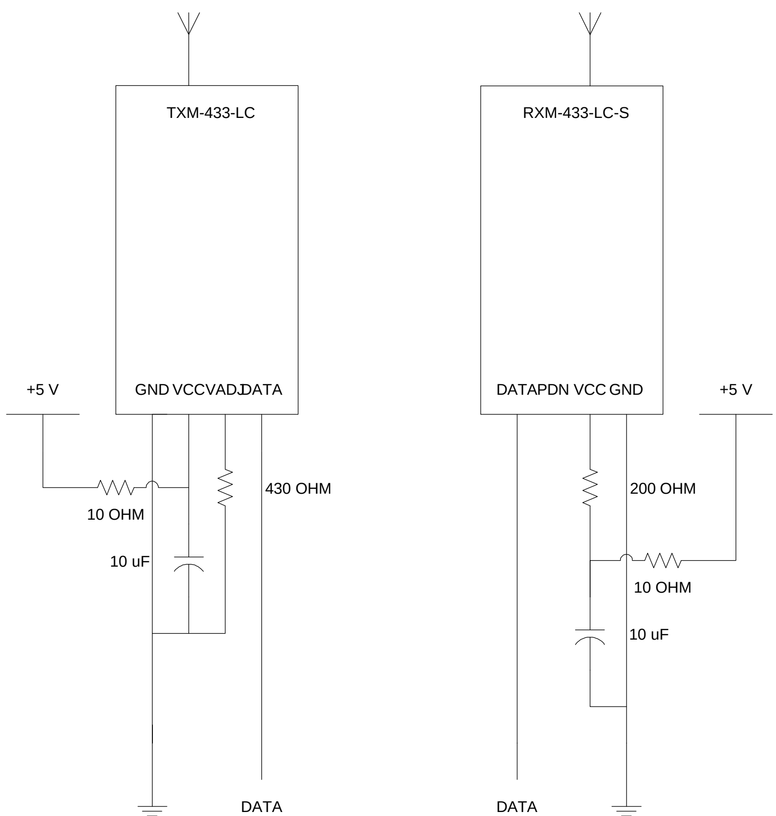

Figure 7: Schematics of Transmitter and Receiver

Key Pressed	Direction 1 (DC Motor)	Direction 2 (Servo Motor)	High Time sent (µS)	High Time Received (µS)
4	Forward	Left 45^o	200	100-250
5	Forward	Middle 0^o	400	300-450
6	Forward	Right 45^o	600	500-650
7	Forward	Left 90^o	800	700-850
8	Forward	Right 90^o	1000	900-1050
A	Backward	Left45^o	1200	1100-1250
0	Backward	Middle 0^o	1400	1300-1450
B	Backward	Right 45^o	1600	1500-1650

Table 3: Keypad Number Corresponding with DC/Servo Motor Controls and High Times

3. Electrical Elements

3.1 Encoding Module

The car is remotely controlled using a matrix keypad, taking into account budget constraints and simplicity in the decision making process. The matrix keypad is a four by four matrix of crossed wires, representing 16 different keys. For the duration of a key press, an electrical connection is created between the row and column wires that intersect underneath the key. To detect key presses, a Verilog module (Appendix B) is written that scans the keypad (see Figure 8 for the Finite State Machine) by sending one column wire at a time to a low logic state, while the others are held high. The idle states represent which column is being scanned for key presses.

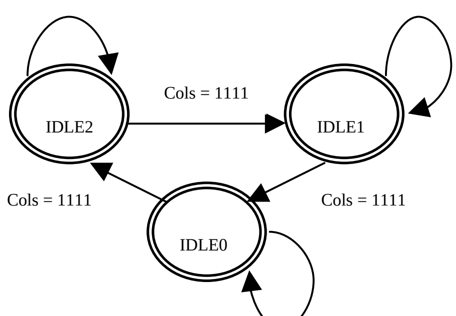

Figure 8: Finite State Machine of the Scanner

The row wires are pulled up with 47 kO resistors, so they will be in a high logic state if undisturbed. When a key is pressed, the corresponding row wire will be pulled low only when the corresponding column is sent low by the scanning circuitry, which will then freeze until the key is released. The combination of low row and column wires uniquely identifies which key has been pressed, which can then be encoded as an output. There are 8 different outputs because 8 keys on the keypad control the car. See Table 3 above for output signals corresponding with each key press. Figures 8 and 9 below are the block diagrams and schematics of this module.

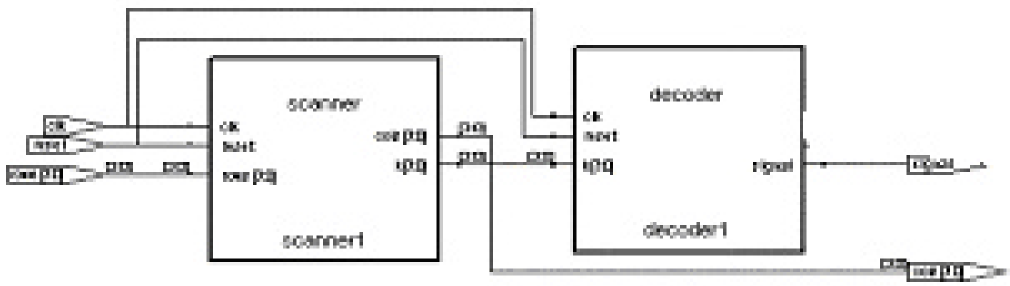

Figure 8: Block Diagram of Encoding Module

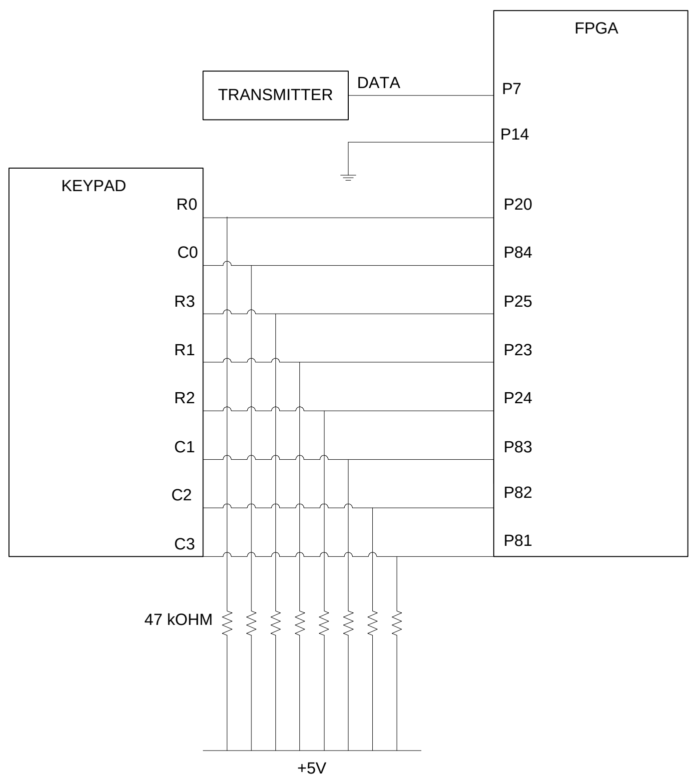

Figure 9: Schematic of Encoding Module on Breadboard

3.2 Decoding Module

From the encoding module, a signal (out of 8 possible signals) will be transmitted to the receiver and go through a decoding module that will drive the motors. Another Verilog module (Appendix C) was written that takes in the signal outputted by the encoding module and outputs corresponding signals to drive the dc and servo motors. The decoder module decodes the signal sent to it and recognizes which key has been pressed by the user. Within the decoder module, there are five other Verilog modules (Appendix C) that code for five pulse-width-modulated signals that control the Futaba FP-S148 servo motor to turn left or right 45°, left or right 90°, or to center itself. For the eight signals that outputs were assigned to, the decoder module will send low or high signals to the two leads on the DC motor and a specific PWM signal to the servo motor. A block diagram of this module can be seen below at Figure 10.

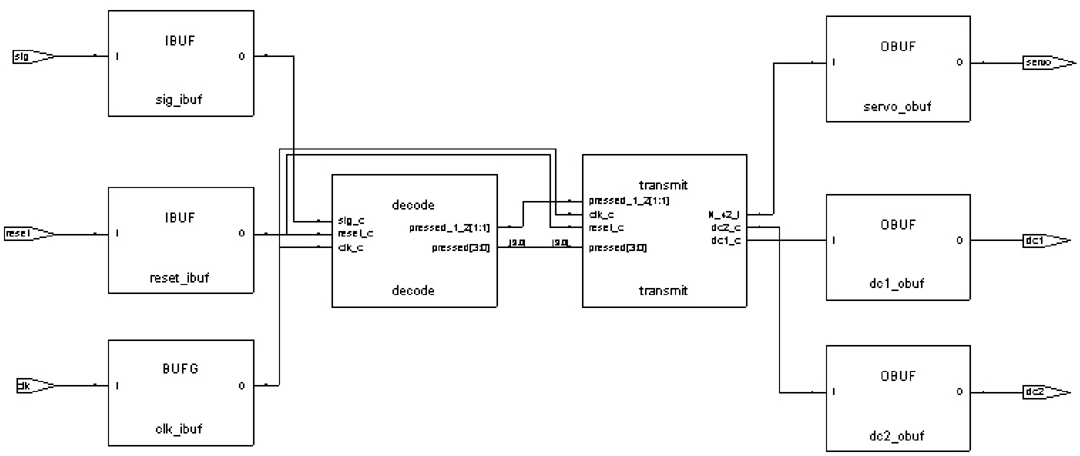

Figure 10: Block Diagram of the Decoding Module

Schematics of the decoding module can be seen on Figure 11. The signals (2) outputted to the dc motor will first go through an H-bridge because the dc motor needs a bidirectional circuit to drive it.

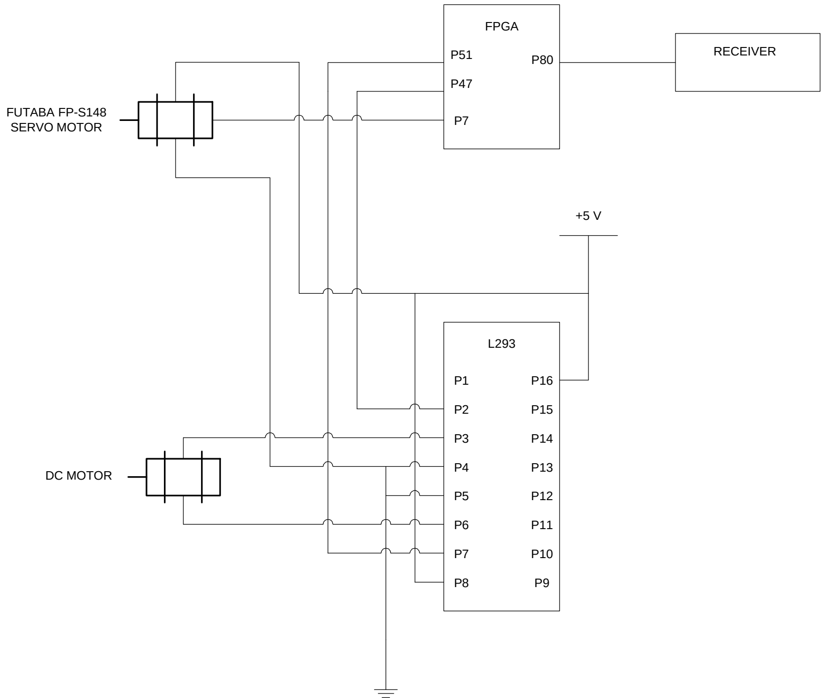

Figure 11: Schematics of Decoding Module

3.3 Power Supply

Three voltage regulators are used to supply 5V of DC voltage to the electrical components of the project which includes the FPGA, H-bridge, transmitter, and receiver. Voltage regulators are used because constant 5V DC voltages are needed to drive the electrical components. Two of the voltage regulators are placed on the breadboard on the car and the other one is placed on the breadboard on the control panel. The voltage regulator used was the 7805s (see Figure 12). The input is 7.2V or greater of DC voltage, the common is grounded, and the output voltage is 5V of DC voltage with 1.5A of current. Six 1.5V batteries were used to supply the voltage needed for the input. Wires and electrical tape are used to connect the batteries/battery packs together to make three portable power supplies (one for the transmitter FPGA and 2 for the receiver FPGA and DC motor) to input to the voltage regulators.

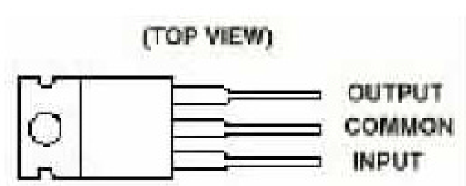

Figure 12: Diagram of 7805 Voltage Regulator

The problem that was encountered with the power supply was that it did not supply enough current to be able to drive the dc motor, which is driven by approximately 150mA. Basically two power supplies are used on the breadboard of the decoding module. One of the power supplies is used to power up the FPGA and servo motor and another power supply is used to drive the DC motor through the H-bridge. Since the FPGA can only output 12mA, not having a lot of current from the batteries is not a problem. However, the L293 H-bridge can out put up to 1.2A of current, not having enough current became a problem. The motor did not have enough current to drive it that the forward/backward motion is very slow and eventually dies. If the project went on longer, different batteries would be used that output more current.

4. Mechanical Elements

Since the project focus is on the electrical elements of RC cars, the only mechanical element looked at was the steering mechanism for the car. Originally the car was driven with a DC motor steering mechanism. Because a more precise control of the steering is desired, steering of the car is controlled with a servo motor. The existing steering mechanism in the car was modified to accommodate the FP-S148. Figure 12 shows a diagram of the steering mechanism of the servo motor, which consists of two Lego gears. The forward/backward movement mechanism, driven by a DC motor, was able to be incorporated in the final design without modification.

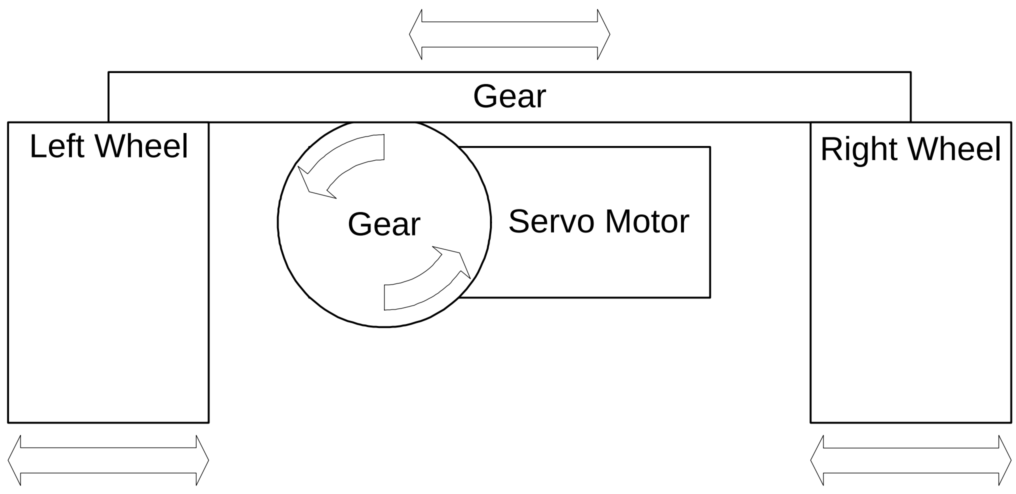

Figure 12: Diagram of Steering Mechanism for Servo Motor

5. Results

At the end of this project, a RC controller was built that controlled the Nissan XTerra toy car. The car can be controlled to go forwards, backwards, both straight and with different degrees of left and right steering. Everything worked as planned except that the battery packs made were very unreliable and the batteries that were used did not output enough current. The lessons learned from this project is that the testing process is just as important as the design and research, therefore enough time should be allotted for both.

6. References

[1] Futuba Corporation (California), (949) 455-9888

[2] Mouser Electronics, http://www.mouser.com

[3] Reynolds Electronics, http://www.reynoldselectronics.com

[4] Tower Hobbies, http://www.towerhobbies.com

7. Parts List

Part	Source	Price*
1.5V AA Batteries (10)	Walmart	5
7805 Voltage Regulator (2)	Jameco Electronics	0.7
Futuba S148 Servo Motor	Tower Hobbies	15
Gears for S148	Tower Hobbies	5
L293D H-Bridge	Mouser Electronics	2
RC Toy Car	Toys 'R Us	50
TXM/RXM-443-LC Transmitter/Receiver Module	Reynolds Electronics	35
*Prices do not included shipping/handling	Total:	112.7

APPENDIX A: Code to generate PWM signals

```
                             **this program creates a PWM output
                             **proportional to the analog voltage
                             **on PE7 using output compare OC4 and
                             **interrupts
                             *******************************************

                             *Port Addresses
1000                         REG      EQU     $1000   *base address of registers
1000                         PORTA    EQU     $1000
1004                         PORTB    EQU     $1004
100a                         PORTE    EQU     $100A
101c                         TOC4     EQU     $101C
1020                         TCTL1    EQU     $1020
1022                         TMSK1    EQU     $1022
1023                         TFLG1    EQU     $1023
1030                         ADCTL    EQU     $1030
1039                         OPTION   EQU     $1039
1031                         ADR1     EQU     $1031

                             *Masks
0010                         OC4F     EQU     %00010000
0004                         BIT2     EQU     %00000100

                             *Local variables
0020                          ORG     $20
0020 08 00                   PWMLO    FDB     $0800   *low time for pulse width modulation
0022 08 00                   PWMHI    FDB     $0800   *high time for pulse width modulation

                             *Interrupt vector
00d6                          ORG     $00D6           *output compare 4 interrupt service routine
00d6 7e c1 30                 JMP     OC4ISR          *hardwired to $00D6, where we put jump to our routine

                             *Main program
c100                          ORG     $C100

                             *Initialize
c100 86 80                    LDAA    #%10000000      *start a/d charge jump
c102 b7 10 39                 STAA    OPTION
c105 86 08                    LDAA    #%00001000      *set OC2 to set output pin low
c107 b7 10 20                 STAA    TCTL1
c10a 86 10                    LDAA    #%00010000      *enable OC4 interrupt
c10c b7 10 22                 STAA    TMSK1
c10f 0e                       CLI                     *turn on interrupts

                             *Sample A/D converter and adjust high/low times while waiting
c110 86 aa                    LDAA    #$AA
c112 b7 10 31                 STAA    ADR1
c115 bd c1 54                LOOP     JSR     CONVERT        *sample output
c118 0f                       SEI                     *disable interrupts while modifying the low and high times
c119 f6 10 31                 LDAB    ADR1                   *read result of a/d conversion
c11c f7 10 04                 STAB    PORTB
c11f 4f                       CLRA                           *set high bits of double accumulator to 0
c120 05                       ASLD                           *multiply delay by 16
c121 05                       ASLD
c122 05                       ASLD
c123 05                       ASLD
c124 dd 22                    STD     PWMHI
c126 cc ea 60                 LDD     #$EA60          *compute low time = 1000-high time
c129 93 22                    SUBD    PWMHI           *we want the period to be 30 ms
c12b dd 20                    STD     PWMLO
c12d 0e                       CLI                     *reenable interrupts
c12e 20 e5                    BRA     LOOP
```

```

                           *Interrupt service routine for OC4
c130 ce 10 00              OC4ISR   LDX     #REG
c133 1f 23 10 1c            BRCLR   TFLG1-REG,X OC4F RTOC4 *ignore other interrupts
c137 86 10                  LDAA    #OC4F                 *store 1 to clear flag
c139 a7 23                  STAA    TFLG1-REG,X           *zeros do nothing
c13b 1e 20 04 0b            BRSET   TCTL1-REG,X BIT2 LASTHI
c13f 1c 20 04               BSET    TCTL1-REG,X BIT2
c142 ec 1c                  LDD     TOC4-REG,X            *increment output compare time
c144 d3 20                  ADDD    PWMLO
c146 ed 1c                  STD     TOC4-REG,X
c148 20 09                  BRA     RTOC4

c14a 1d 20 04              LASTHI   BCLR    TCTL1-REG,X BIT2       *set OC4 to set pin low
c14d ec 1c                  LDD     TOC4-REG,X                     *set wait time
c14f d3 22                  ADDD    PWMHI
c151 ed 1c                  STD     TOC4-REG,X

c153 3b                    RTOC4    RTI                            *return from the interrupt

                           *a/d converter subroutine
                           *reads analog voltage on PE7
                           *leaves result in ADR1 register
                           *trashes accumulator

c154 86 07                 CONVERT  LDAA    #$07    *scan, mult = 0; select challen PE7
c156 b7 10 30               STAA    ADCTL           *begin conversion of channel PE7

c159 b6 10 30              CLOP     LDAA    ADCTL   *wait for conversion to complete
c15c 84 80                  ANDA    #%10000000      *look at conversion complete flag
c15e 27 f9                  BEQ     CLOP            *wait until done
c160 39                     RTS
```

Appendix B: Code for Keyboard Encoding Module

```
module rc(clk,reset,rows,cols,signal);

    input      clk;

    input      reset;

    input      [3:0] rows;

    output     [3:0] cols;

    output     signal; //one of 8 signals outputted to represent the key pressed

    wire       [3:0] k;

    scanner scanner1(clk,reset,rows,cols,k); // scans keypad for key press

    decoder decoder1(clk,reset,k,signal);    // encodes key press and outputs to transmitter

endmodule


module scanner(clk,reset,rows,cols,k);
       input                         clk;
       input                         reset;
       input          [3:0]   rows;
       output  [3:0]   cols;
       output  [3:0]   k;

       reg            [3:0]   cols;
       reg            [3:0]   nextcols;
       reg            [3:0]   k;

parameter IDLE2 = 4'b1011; //3rd column
parameter IDLE1 = 4'b1101; //2nd column
parameter IDLE0 = 4'b1110; //1st column
parameter L2   = 4'b1110;  //1st column
parameter R2   = 4'b1011;  //3rd column
parameter LL   = 4'b1110;  //1st column
parameter RR   = 4'b1011;  //3rd column
parameter L0   = 4'b1110;  //1st column
parameter R0   = 4'b1011;  //3rd column
parameter U           = 4'b1101;  //2nd column
parameter D           = 4'b1101;  //2nd column

always @(posedge clk or posedge reset)
       if (reset) cols <= IDLE2;
       else cols <= nextcols;

always @(cols or rows) //scans the columns
       case (cols)
       IDLE2: if (&rows)                            nextcols <= IDLE1;
                      else                          nextcols <= IDLE2;

       IDLE1: if (&rows)                            nextcols <= IDLE0;
                      else                          nextcols <= IDLE1;

       IDLE0: if (&rows)                            nextcols <= IDLE2;
                      else                          nextcols <= IDLE0;

       L2:      if (&rows)                          nextcols <= IDLE2;
                      else                          nextcols <= IDLE0;

       R2:      if (&rows)                          nextcols <= IDLE1;
```

```
                        else                            nextcols <= IDLE2;

        LL:     if (&rows)                              nextcols <= IDLE2;
                        else                            nextcols <= IDLE0;

        RR:     if (&rows)                              nextcols <= IDLE1;
                        else                            nextcols <= IDLE2;

        L0:     if (&rows)                              nextcols <= IDLE2;
                        else                            nextcols <= IDLE0;

        R0:     if (&rows)                              nextcols <= IDLE1;
                        else                            nextcols <= IDLE2;

        U:              if (&rows)                      nextcols <= IDLE0;
                        else                            nextcols <= IDLE1;

        D:              if (&rows)                      nextcols <= IDLE0;
                        else                            nextcols <= IDLE1;

        default:                                        nextcols <= IDLE2;

endcase

always @(cols or rows)
        case({cols, rows})
        8'b10111011:    k <= 'h1;       // UR
        8'b10111110:    k <= 'h2;       // DR
        8'b11101011:    k <= 'h3;       // UL
        8'b11101110:    k <= 'h4;       // DL
        8'b10111101:    k <= 'h5;       // RR
        8'b11101101:    k <= 'h6;       // LL
        8'b11011011:    k <= 'h7;       // U
        8'b11011110:    k <= 'h8;       // D

        default:        k <= 'h0; // IDLE
endcase

endmodule

module decoder(clk,reset,k,signal);
        input           clk;
        input           reset;
        input           [3:0]   k; //key pressed
        output  signal; //signal that represent the key pressed

parameter UL = 'h1;     // up left
parameter DL = 'h2;     // down left
parameter UR = 'h3;     // up right
parameter DR = 'h4;     // down right
parameter LL = 'h5;     // hard left
parameter RR = 'h6;     // hard right
parameter U = 'h7;      // up
parameter D = 'h8;      // down

reg ul45f;
reg uf;
reg ur45f;
reg ul90f;
reg ur90f;
reg dl45f;
reg df;
reg dr45f;
//calls functions for each of the 8 signals
left45 left45(clk,reset,ul45);
dleft45 dleft45(clk,reset,dl45);
left90 left90(clk,reset,ul90);
up up(clk,reset,u);
down down(clk,reset,d);
right45 right45(clk,reset,ur45);
```

```
dright45 dright45(clk,reset,dr45);
right90 right90(clk,reset,ur90);

always @(k)     // depending on key press, one output signal is to be sent
        case (k)
                UL: begin
                                ul45f <= 1;
                                uf <= 0;
                                ur45f <= 0;
                                ul90f <= 0;
                                ur90f <= 0;
                                dl45f <= 0;
                                df <= 0;
                                dr45f <= 0;
                        end
                UR: begin
                                ul45f <= 0;
                                uf <= 0;
                                ur45f <= 1;
                                ul90f <= 0;
                                ur90f <= 0;
                                dl45f <= 0;
                                df <= 0;
                                dr45f <= 0;
                        end
                DL: begin
                                ul45f <= 0;
                                uf <= 0;
                                ur45f <= 0;
                                ul90f <= 0;
                                ur90f <= 0;
                                dl45f <= 1;
                                df <= 0;
                                dr45f <= 0;
                        end
                DR: begin
                                ul45f <= 0;
                                uf <= 0;
                                ur45f <= 0;
                                ul90f <= 0;
                                ur90f <= 0;
                                dl45f <= 0;
                                df <= 0;
                                dr45f <= 1;
                        end
                LL: begin
                                ul45f <= 0;
                                uf <= 0;
                                ur45f <= 0;
                                ul90f <= 1;
                                ur90f <= 0;
                                dl45f <= 0;
                                df <= 0;
                                dr45f <= 0;
                        end
                RR: begin
                                ul45f <= 0;
                                uf <= 0;
                                ur45f <= 0;
                                ul90f <= 0;
                                ur90f <= 1;
                                dl45f <= 0;
                                df <= 0;
                                dr45f <= 0;
                        end
                U:  begin
                                ul45f <= 0;
                                uf <= 1;
                                ur45f <= 0;
                                ul90f <= 0;
```

```
                              ur90f <= 0;
                              dl45f <= 0;
                              df <= 0;
                              dr45f <= 0;
                        end

               D:  begin
                              ul45f <= 0;
                              uf <= 0;
                              ur45f <= 0;
                              ul90f <= 0;
                              ur90f <= 0;
                              dl45f <= 0;
                              df <= 1;
                              dr45f <= 0;
                        end
       default: begin
                              ul45f <= 0;
                              uf <= 0;
                              ur45f <= 0;
                              ul90f <= 0;
                              ur90f <= 0;
                              dl45f <= 0;
                              df <= 0;
                              dr45f <= 0;
                              end
       endcase
//mux to determine which signal to send
assign signal = ul45f ? ul45 : (uf ? u : (ur45f ? ur45 : (ul90f ? ul90 : (ur90f ? ur90 : (dl45f ?
dl45 : (df ? d : (dr45f ? dr45 : 0)))))));

endmodule


module dleft45(clk,reset,dl45);//signal creator
    input clk;
    input reset;
    output dl45;

    parameter stopping = 1200;   //1.2 ms high time
        parameter stopping2 = 30000; //30 ms period

    reg [14:0] count; //15 bit counter
        reg dl45;


always @(posedge clk or posedge reset)
begin

       if (reset)
       begin
                dl45 <= 0;
                count <= 0;
       end
       else if (count <= stopping)
       begin
               dl45 <= 1;
               count <= count + 1;

       end
       else if (count >= stopping2)
       begin
               dl45 <= 0;
               count <= 0;

       end
       else
       begin
               dl45 <= 0;
               count <= count + 1;
```

```
        end
end
endmodule

module down(clk,reset,d);
    input clk;
    input reset;
    output d;
    parameter stopping = 1400;   //1.4 ms high time
    parameter stopping2 = 30000; //30 ms period
         reg [14:0] count; //15 bit counter
         reg d;
always @(posedge clk or posedge reset)
begin
        if (reset)
        begin
                 d <= 0;
                 count <= 0;
        end
        else if (count <= stopping)
        begin
                d <= 1;
                count <= count + 1;
        end
        else if (count >= stopping2)
        begin
                d <= 0;
                count <= 0;
        end
        else
        begin
                d <= 0;
                count <= count + 1;
        end
end

endmodule
module dright45(clk,reset,dr45);
    input clk;
    input reset;
    output dr45;

    parameter stopping = 1600;   //1.6mS high time
         parameter stopping2 = 30000; //30mS period

         reg [14:0] count; //15 bit counter
         reg dr45;

always @(posedge clk or posedge reset)
begin
        if (reset)
        begin
                 dr45 <= 0;
                 count <= 0;
        end
        else if (count <= stopping)
        begin
                dr45 <= 1;
                count <= count + 1;
        end
        else if (count >= stopping2)
        begin
                dr45 <= 0;
                count <= 0;
        end
        else
        begin
                dr45 <= 0;
                count <= count + 1;
        end
```

```
end

endmodule

module left45(clk,reset,ul45);
    input clk;
    input reset;
    output ul45;

    parameter stopping = 200;   //.2 ms high time
         parameter stopping2 = 30000; //30 ms period

    reg [14:0] count; //15 bit counter
         reg ul45;

always @(posedge clk or posedge reset)
begin

        if (reset)
        begin
                 ul45 <= 0;
                 count <= 0;

        end
        else if (count <= stopping)
        begin
                ul45 <= 1;
                count <= count + 1;

        end
        else if (count >= stopping2)
        begin
                ul45 <= 0;
                count <= 0;

        end
        else
        begin
                ul45 <= 0;
                count <= count + 1;
        end
end

endmodule
```

```
module left90(clk,reset,ul90);
    input clk;
    input reset;
    output ul90;

    parameter stopping = 800;   //.8 ms high time
    parameter stopping2 = 30000; //30 ms period

    reg [14:0] count; //15 bit counter
    reg ul90;

always @(posedge clk or posedge reset)
begin
       if (reset)
       begin
                ul90 <= 0;
                count <= 0;
       end
       else if (count <= stopping)
       begin
               ul90 <= 1;
               count <= count + 1;
       end
       else if (count >= stopping2)
       begin
               ul90 <= 0;
               count <= 0;
       end
       else
       begin
               ul90 <= 0;
               count <= count + 1;
       end
end

endmodule

module right45(clk,reset,ur45);
    input clk;
    input reset;
    output ur45;

    parameter stopping = 600;   //.6 ms high time
         parameter stopping2 = 30000; //30ms period

         reg [14:0] count; //15 bit counter
         reg ur45;

always @(posedge clk or posedge reset)
begin
       if (reset)
       begin
                ur45 <= 0;
                count <= 0;
       end
       else if (count <= stopping)
       begin
               ur45 <= 1;
               count <= count + 1;
       end
       else if (count >= stopping2)
       begin
               ur45 <= 0;
               count <= 0;
       end
       else
       begin
               ur45 <= 0;
               count <= count + 1;
       end
```

```
end

endmodule

module right90(clk,reset,ur90);
    input clk;
    input reset;
    output ur90;

    parameter stopping = 1000;   //1.0 ms high time
    parameter stopping2 = 30000; //30 ms period
    reg [14:0] count; //15 bit counter
    reg ur90;

always @(posedge clk or posedge reset)
        if (reset)
        begin
                ur90 <= 0;
                count <= 0;
        end
   else if (count <= stopping)
        begin
               ur90 <= 1;
               count <= count + 1;
        end
        else if (count >= stopping2)
        begin
               ur90 <= 0;
               count <= 0;
        end
        else
        begin
               ur90 <= 0;
               count <= count + 1;
        end

endmodule

module up(clk,reset,u);
    input clk;
    input reset;
    output u;

    parameter stopping = 400;   //.4 ms high time
         parameter stopping2 = 30000; //30 ms period

         reg [14:0] count; //15 bit counter
         reg u;

always @(posedge clk or posedge reset)
begin
        if (reset)
        begin
                u <= 0;
                count <= 0;
        end
        else if (count <= stopping)
        begin
               u <= 1;
               count <= count + 1;
        end
        else if (count >= stopping2)
        begin
               u <= 0;
               count <= 0;
        end
        else
        begin
               u <= 0;
               count <= count + 1;
```

```
          end
end
```

Appendix C: Code for Signal Decoding Module

```
module receive(clk,reset,sig,servo,dc1,dc2);
    input clk;
    input reset;
    input sig;
    output servo; //servo signal
    output dc1; //dc signal
    output dc2; //dc signal

         wire [3:0] pressed;

         decode decode(clk, reset, sig, pressed);    // decodes signal received by receiver
         transmit transmit(clk, reset, pressed, servo, dc1, dc2); // signals to be sent to motors

endmodule

module decode(clk,reset,sig, pressed);
    input clk;
    input reset;
    input sig; //signal received
    output [3:0] pressed; //key pressed

         reg [14:0]    clk_count; //counts clk cycles
         reg [14:0]    sig_count; //counts high times
         reg [14:0]    out; //final high time
         reg [3:0]     pressed; //key pressed

         parameter stopp = 30000; // 30 ms period
         // range of high times to determine which signal is being sent
         parameter ULupper = 650;//500-650 range encodes for up left 45 signals
         parameter ULlower = 500;
         parameter DLupper = 1650;//1500-1650 down left 45
         parameter DLlower = 1500;
         parameter URupper = 250;//100-250 up right 45
         parameter URlower = 100;
         parameter DRupper = 1250;//1100-1250 down right 45
         parameter DRlower = 1100;
         parameter LLupper = 1050;//900-1050 up left 90
         parameter LLlower = 900;
         parameter RRupper = 850;//700-850 up right 90
         parameter RRlower = 700;
         parameter Uupper  = 450;//300-450 up
         parameter Ulower  = 300;
         parameter Dupper  = 1450;//1300-1450 down
         parameter Dlower  = 1300;

         always @(posedge clk or posedge reset) //counts number of high times in 30 ms period
         begin
                if (reset)
                begin
                       clk_count <= 0;
                       sig_count <= 0;
                       out <= 0;
                end

                else if (sig & (clk_count < stopp))
                begin
                       clk_count <= clk_count + 1;
                       sig_count <= sig_count + 1;
           end

                else if (clk_count == stopp)
                begin
                       clk_count <= clk_count + 1;
                       out <= sig_count;
                end
```

```
                else if(sig & (clk_count > stopp))
                begin
                        clk_count <= 1;
                        sig_count <= 1;
                end

                else if(clk_count > stopp)
                begin
                        clk_count <= 1;
                        sig_count <= 0;
                end

                else
                        clk_count <= clk_count + 1;
        end

always @(out)  // decodes received signal into what key has been pressed
        if (out >= URlower & out < URupper) pressed <= 'h3;
        else if (out >= Ulower & out <= Uupper) pressed <= 'h7;
        else if (out >= ULlower & out <= ULupper) pressed <= 'h1;
        else if (out >= RRlower & out <= RRupper) pressed <= 'h6;
        else if (out >= LLlower & out <= LLupper) pressed <= 'h5;
        else if (out >= DRlower & out <= DRupper) pressed <= 'h4;
        else if (out >= Dlower & out <= Dupper) pressed <= 'h8;
        else if (out >= DLlower & out <= DLupper) pressed <= 'h2;
        else pressed <= 'h0;
endmodule

module transmit(clk,reset,pressed,servo,dc1,dc2); // decodes key press into motor actions
    input clk;
    input reset;
    input [3:0] pressed;

    output servo;
    output dc1;
    output dc2;

parameter UL = 'h1;    // UL up left 45
parameter DL = 'h2;    // DL
parameter UR = 'h3;    // UR
parameter DR = 'h4;    // DR
parameter LL = 'h5;    // LL up left 90
parameter RR = 'h6;    // RR
parameter U = 'h7;     // U
parameter D = 'h8;     // D

reg lsig;
reg rsig;
reg llsig;
reg rrsig;
reg dc1;
reg dc2;
//calls signals for servo
left45 left45(clk,reset,l45);
left90 left90(clk,reset,l90);
middle middle(clk,reset,mid);
right45 right45(clk,reset,r45);
right90 right90(clk,reset,r90);
//determines which signals are to be sent to motors
always @(pressed)
        case (pressed)
                UL:     begin
                                lsig <= 1;
                                rsig <= 0;
                                llsig <= 0;
                                rrsig <= 0;
                                dc1 <= 1;
                                dc2 <= 0;
                        end
```

```
            UR:     begin
                          lsig <= 0;
                          rsig <= 1;
                          llsig <= 0;
                          rrsig <= 0;
                          dc1 <= 1;
                          dc2 <= 0;
                    end
            DL:     begin
                          lsig <= 1;
                          rsig <= 0;
                          llsig <= 0;
                          rrsig <= 0;
                          dc1 <= 0;
                          dc2 <= 1;
                    end
            DR: begin
                          lsig <= 0;
                          rsig <= 1;
                          llsig <= 0;
                          rrsig <= 0;
                          dc1 <= 0;
                          dc2 <= 1;
                    end
            LL:     begin
                          lsig <= 1;
                          rsig <= 0;
                          llsig <= 1;
                          rrsig <= 0;
                          dc1 <= 1;
                          dc2 <= 0;
                    end
            RR:      begin
                          lsig <= 0;
                          rsig <= 1;
                          llsig <= 0;
                          rrsig <= 1;
                          dc1 <= 1;
                          dc2 <= 0;
                    end
            U:      begin
                          lsig <= 0;
                          rsig <= 0;
                          llsig <= 0;
                          rrsig <= 0;
                          dc1 <= 1;
                          dc2 <= 0;
                    end

            D:       begin
                          lsig <= 0;
                          rsig <= 0;
                          llsig <= 0;
                          rrsig <= 0;
                          dc1 <= 0;
                          dc2 <= 1;
                    end
       default:     begin
                                          lsig <= 0;
                                          rsig <= 0;
                                          llsig <= 0;
                                          rrsig <= 0;
                                          dc1 <= 0;
                                          dc2 <= 0;
                    end
       endcase
//mux to determine which servo signal to send
assign servo = lsig ? (llsig ? l90 : l45) : (rsig ? (rrsig ? r90 : r45) : mid);

endmodule
```

```
module left45(clk,reset,l45); // create servo signal
    input clk;
    input reset;
    output l45;

    parameter stopping = 650;   //.65 ms high time
    parameter stopping2 = 30000; //30 ms period

    reg [14:0] count; //15 bit counter
    reg l45;


always @(posedge clk or posedge reset)
begin
       if (reset)
       begin
                l45 <= 0;
                count <= 0;
       end
else if (count <= stopping)
       begin
               l45 <= 1;
               count <= count + 1;
       end
       else if (count >= stopping2)
       begin
               l45 <= 0;
               count <= 0;
       end
       else
       begin
               l45 <= 0;
               count <= count + 1;
       end
end

endmodule

module left90(clk,reset,l90);
    input clk;
    input reset;
    output l90;

    parameter stopping = 100;   //.1 ms high time
    parameter stopping2 = 30000; //30 ms period

    reg [14:0] count; //15 bit counter
    reg l90;

always @(posedge clk or posedge reset)
begin
       if (reset)
       begin
                l90 <= 0;
                count <= 0;
       end
   else if (count <= stopping)
       begin
               l90 <= 1;
               count <= count + 1;
       end
       else if (count >= stopping2)
       begin
               l90 <= 0;
               count <= 0;
       end
       else
       begin
               l90 <= 0;
```

```
               count <= count + 1;
       end
end

endmodule

module middle(clk,reset,mid);
    input clk;
    input reset;
    output mid;

    parameter stopping = 1200;   //1.2 ms high time
    parameter stopping2 = 30000; //30 ms period

    reg [14:0] count; //15 bit counter
    reg mid;

always @(posedge clk or posedge reset)
begin
       if (reset)
       begin
                mid <= 0;
                count <= 0;
       end
       else if (count <= stopping)
       begin
                mid <= 1;
                count <= count + 1;
       end
       else if (count >= stopping2)
       begin
                mid <= 0;
                count <= 0;
       end
       else
       begin
               mid <= 0;
               count <= count + 1;
       end
end

endmodule

module right45(clk,reset,r45);
    input clk;
    input reset;
    output r45;

    parameter stopping = 1750;   //1.75ms high time
    parameter stopping2 = 30000; //30ms period

    reg [14:0] count; //15 bit counter
    reg r45;

always @(posedge clk or posedge reset)
begin
       if (reset)
       begin
                r45 <= 0;
                count <= 0;
       end
       else if (count <= stopping)
       begin
                r45 <= 1;
                count <= count + 1;
       end
       else if (count >= stopping2)
       begin
                r45 <= 0;
                count <= 0;
```

```
        end
        else
        begin
                r45 <= 0;
                count <= count + 1;
        end
end

endmodule

module right90(clk,reset,r90);
    input clk;
    input reset;
    output r90;

    parameter stopping = 2300;   //2.3 ms high time
    parameter stopping2 = 30000; //30 ms period
    reg [14:0] count; //15 bit counter
    reg r90;

always @(posedge clk or posedge reset)
        if (reset)
        begin
                 r90 <= 0;
                 count <= 0;
        end
        else if (count <= stopping)
        begin
                r90 <= 1;
                count <= count + 1;
        end
        else if (count >= stopping2)
        begin
                r90 <= 0;
                count <= 0;
        end
        else
        begin
                r90 <= 0;
                count <= count + 1;
        end

endmodule
```