A Laser Guided RC Vehicle

Final Project Report
December 7, 2006
E155

Mike Chan and Matt Williams

Abstract

Remote control of vehicles or small robots is often accomplished through radio signals. Another compelling possibility for data transmission is light. This project prototypes a simple vehicle which can be maneuvered by a laser pointer. The control system for the vehicle uses solar panels to sample light intensity, the PIC to determine which panel the laser is incident upon, and the FPGA to control the movement of the vehicle. If the user directs the laser upon a solar cell, the vehicle rotates to face the user and travels in their direction. If no input is given, the vehicle remains stationary.

Overview

This project involves controlling the movement of a RC vehicle by directing laser light upon solar panels. The solar panels are mounted on the RC vehicle. When the laser pointer excites a panel, the detection system determines which direction to move the vehicle, either forward, left, or right. The vehicle remains idle when no laser light is incident upon its panels. Signal detection is done by the PIC microcontroller. The FPGA receives the direction from the PIC and moves the vehicle.

Motivation

Autonomous vehicles are crucial for exploring hazardous areas, such as abandoned mines or the surface of distant planets. While artificial intelligence will allow these vehicles to collect data and perform functions on their own, there will be times where manual control will become necessary. When these vehicles are distant from the user, a way to propagate commands through long distances becomes a necessity. Due to potential interference or security concerns, radio frequency control may not be the ideal solution. Instead, a localized control mechanism, such as a laser, may be a more suitable means of control. Although there are limitations to this method, this project will explore one possibly of using light to control a distant object.

System Partitioning

This project is comprised of two autonomous systems. The first is a laser pulsing system that is responsible for modulating the laser intensity to send a signal that is detectable by the vehicle control system mounted on the vehicle. The laser control system is primarily a PIC microcontroller acting as a timer. The vehicle control system consists of analog circuits, a PIC, and a FPGA.

The vehicle control system must detect laser light, amplify and digitize the signal, process the signal to extract the laser from extraneous light sources, decide upon a direction of travel, and use that information to maneuver the vehicle. Figure 1 shows the division of these tasks between each piece of hardware.

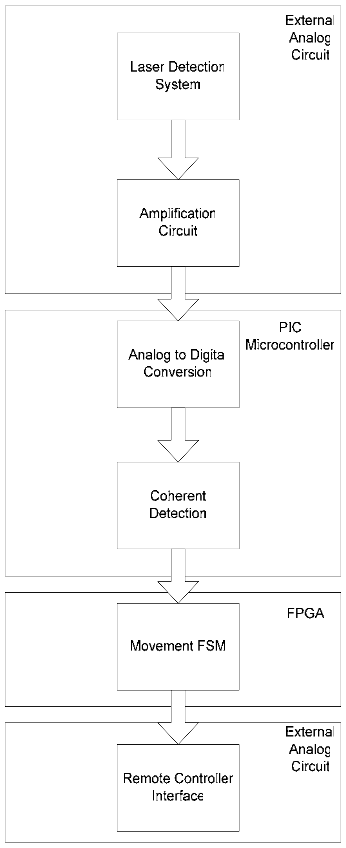


Figure 1: Block diagram for the vehicle control system.

External analog circuits are required to sample the response of the solar panels and amplify the signal to a level suitable for analog-to-digital conversion. In addition, a second analog circuit is required as an interface between the FPGA and the vehicle's remote controller. The PIC digitizes the input signal, performs a coherent detection algorithm, and sets a threshold to determine which panel, if any, the laser is stimulating. The PIC polls the solar cells to collect the needed information. Lastly, the FPGA implements a finite state machine (FSM) which controls the vehicle's direction of travel and prevents anomalous events from inducing motion.

Analog Circuitry

As shown in Figure 1, the analog circuits detect light, act as a signal amplifier, and provide an interface between the FPGA and RC controller. Figure 2 shows the breadboard schematic of the analog components.

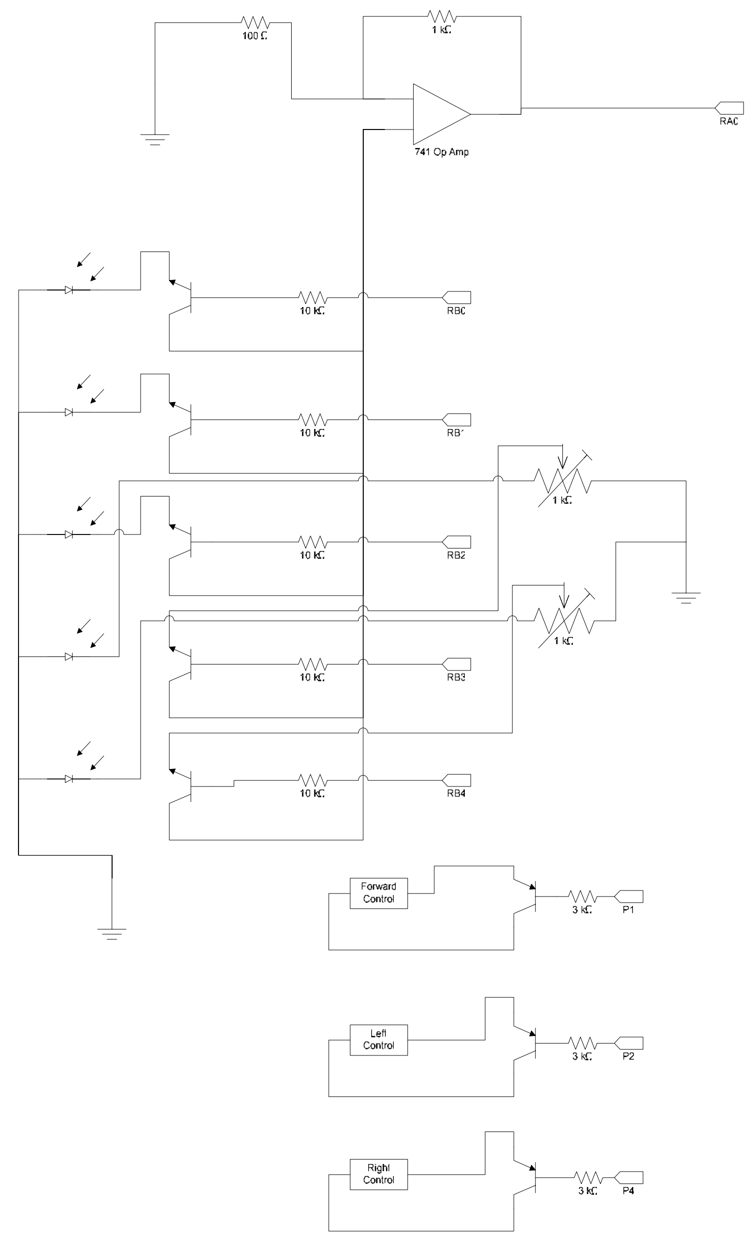


Figure 2: Schematic of breadboard circuits.

Laser Detection System

The laser detection system is at its simplest the combination of a solar cell and an amplifier. Solar cells are used because of their large surface area and relatively low cost. These cells detect the 610 Hz intensity modulated signal from the pulsing laser. This mitigates many drawbacks of solar cells, such as their limited sensitivity, since a low frequency and high intensity source is used. However, the large surface area of a solar cell creates the need for a slightly more complicated laser detection system because the signal from background light becomes significant in magnitude. While fluorescent lights have a lower intensity than a laser, their ubiquity combined with the size of a panel means they quickly dominate the signal from the laser if they are not filtered out.

Common red cellophane placed over the solar cells acts as an effective filter of ambient light. The continuous spectrum of a fluorescent light means most of the power from the light is absorbed by the cellophane. The transmitted light is of far lower intensity than the laser. This allows the signal from the laser to be the dominant signal detected by the solar cells regardless of the solar cells' sizes. This alleviates most problems. However, there is the issue of differences in the sensitivity of the solar cells. A problem the team encountered is attenuated light detected by a particularly sensitive cell creates a larger signal than the signal a less sensitive panel creates with the laser incident upon it. To balance the sensitivities of the panels, the team used potentiometers as voltage dividers.

Each solar cell is connected to the op-amp through a NPN transistor. The bases of these transistors are controlled by the PIC. Since the output for the PIC is in one-hot encoding, the PIC can choose which signal will be amplified by the op-amp. The op-amp is wired as a non-inverting amplifier with a gain of eleven and amplifies whichever solar cell is accessible. The output of the op-amp feeds into the analog to digital converter of the PIC and is digitized before being processed by the PIC and FPGA.

Remote Control Interface

The last element in the control of the vehicle is issuing commands to the vehicle. The team accomplishes this by interfacing with the vehicle's remote control. The vehicle remote operates by sending a pulsing signal to the radio transmitter. A series of pushbuttons determine whether or not a particular signal reaches the transmitter. The team bypasses the pushbuttons on the controller with PNP transistors. Once the system determined a direction to travel, the appropriate PNP base is brought to ground closing the circuit and allows the proper signal to be sent to the transmitter. With this very simple method, the team was able to unobtrusively control the vehicle and utilize the preexisting control mechanisms in the remote.

Microcontroller Design

The microcontroller has by far the most important role in this project. The microcontroller takes analog data from the solar panels, A/D converts the data, and determines which solar panel the laser is incident upon using coherent detection.

The overall structure of the microcontroller is divided into two portions. The first segment samples, processes, stores the analog input. The other segment generates the functions for the coherent detection, polls the solar panels, determines laser direction, and sends the result to the FPGA.

In order to differentiate ambient light from the laser, the team decided to pulse the laser at 610 Hz. 610 Hz was chosen due to sheer convenience, since it correlated one of the timer bits. It is also a much higher frequency than the 120 Hz signal from indoor lighting, and it is under Nyquist frequency of the PIC's A/D converter. Since the PIC is sampling data rapidly, the team decided to apply a coherent detection algorithm in the form of a modified version of Fourier series transform to determine the 610Hz frequency component of the input data. While Fourier series use sines and cosines as basis functions, the team realized that any orthogonal basis signals will work. For computational simplicity, the team used orthogonal square waves instead. The PIC code has variables A and B which map to the values of the square waves. Two variables are needed because the waves generated by A and B are 90 degrees out of phase. Each sample data will be added to a variable called sum1 when A is high and subtracted from sum1 when A is low. The process is repeated with variable B and sum2. This addition takes place in the main module of the C code and runs until an interrupt occurs.

The interrupt has several functions. The first function is to change the values of A and B. This is done by interrupting every quarter period of the 610Hz square wave. Every two interrupts the values of A and B are inverted. However, B is always inverted one quarter period before A, creating a 90 degree phase difference between the pair.

The next part of the interrupt stores the L1 norm of sum1 and sum2 into a storage array after sampling for four periods for a given solar panel. The interrupt also determines which solar panel to poll and ensures each panel is polled for four periods. This is controlled by driving pins RB0-RB4 either high or low. After cycling through all the different pins once, the interrupt's next function is to send the index of the element with the highest value to the FPGA using one-hot encoding. To filter out ambient light, the team creates a threshold by right shifting the largest value 8 bits. If that value is zero, then the output to the FPGA will be zero and the tank will cease to move. In either case, the output to the FPGA is done via the serial port. To prevent the FSM on the FPGA from entering illegal states, pin RE0 is used as an enable signal. This signal goes high when the PIC is transmitting data and is low otherwise.

With ideal transistors, a steady high intensity signal will not generate large Fourier coefficients at 605 Hz. With real transistors, the edge effects, when the transistor is on but not yet saturated, produces a spike that may cause the calculated Fourier coefficient for that panel to be large. To circumvent this effect, the team disregards data from the first and last periods when the transistor may not be fully saturated.

The PIC which controls the laser must pulse it at 610 Hz. This is also done by mapping the output pin RD0 to the 14th bit of the timer. Pin RD0 is tied to a PNP transistor that controls the power to the laser pointer. This makes the output of the laser a 610 Hz square wave.

FPGA Design

The FPGA is used in this system to create a finite state machine that controls the vehicle's direction of movement. The FPGA takes in an external clock, an enable signal, and a serial input – all from the PIC. The FPGA outputs a three-bit direction and a three-bit state used for debugging. In support of the finite state machine, there are two additional modules, a decoder – which maps the solar cell number to one of the three possible directions – and a serial to parallel converter.

The serial input to the FPGA from the PIC is a binary number from zero to five. That number represents which solar cell the PIC has determined the laser was incident upon. If the result is zero, no panel exceeded the threshold so the laser is assumed not to be incident upon the vehicle. While sending data, the PIC also sends a clock, which is used for the serial to parallel conversion, and an enable flag. That enable is set high when the PIC is sending data. Upon completion of the transmission, the enable signal is set low. This enable signal serves as the clock for the FSM that is the core of the system.

The finite state machine takes in direction information from the decoder. It accepts four signals – a stop signal, a forward signal, a left signal, and a right signal. This machine differs from typical FSMs because it triggers upon the negative edge of the enable signal. This ensures the state is updated after the serial input has been completely passed. The decoder determines which signal the output from the PIC corresponds to. The purpose of the FSM is to determine which direction the vehicle should move in. The FSM diagram is shown in Figure 3.

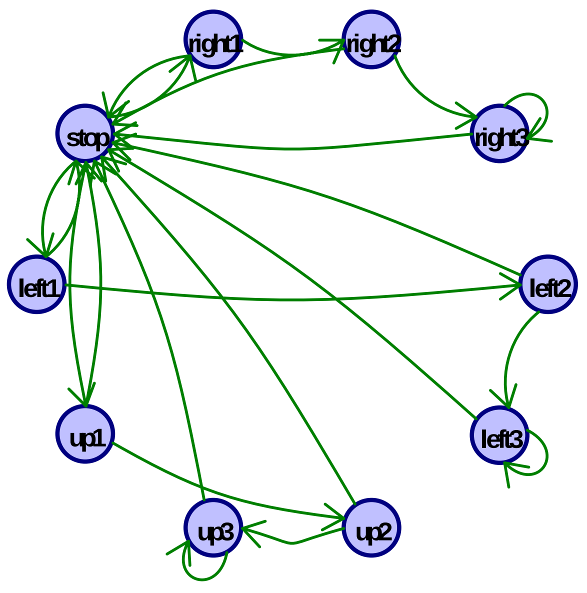


Figure 3:Finite state machine diagram for control of the vehicle.

The default state for this FSM is the stop state. In this state or any state that is not up3, left3, or right 3, the vehicle is stationary. In the stop state, the FSM will accept all of the directional inputs. However, in any other state only one directional input will not send the machine back to the stop state. For instance, in the state up1 any input but up will send the machine back to the stop state. In the third state, up3, left3 and right3 will the vehicle move. In that state, a similar input will keep the machine in the moving state and all others will reset it back to the stop state. This is to ensure the vehicle is receiving a consistent stimulus from one direction before it moves.

The output of the finite state machine is a three bit number which represents the direction in one-cold encoding. This is the output both to the LEDs and to the remote control interface circuit.

Results

The laser controlled RC vehicle works as proposed. The vehicle is stationary when no laser is incident upon the detectors and moves in the appropriate direction when the laser excites a detector. The most difficult part of the project design was to negate the effects of ambient light on the vehicle. The team tried many methods of eliminating the background signal such as normalizing the sensitivity of each solar panel and creating minimum thresholds for the Fourier coefficients. After several failed attempts to eliminate the background through software, the team tried to perform the same function through hardware. From physics, the team realized one way to diminish ambient light is to run the light through a red filter before hitting the panels. This method will essentially let the laser pointer through since the laser is red, but will block most light of other colors.

Even with the cellophane wrap, high intensity light can still trigger the vehicle to move. Through trial and error, the team discovered that the signal from the high intensity light is caused by the non-ideal behavior of the transistors. The team addressed the issue by building in breaks in data acquisition before and after turning a transistor on or off.

The scope of the project was redefined during the duration of the project. The initial project proposal involved having the RC vehicle follow light reflected off the ground. Due to hardware and budget constraints, the team was unable to obtain detectors with enough sensitivity to detect the reflected light. Therefore, the team altered the project so the laser has to be incident upon the photo-detectors, which allows the use of inexpensive photo-detectors.

Parts List

Part	Source	Price
RC Tiger Tank	Hobbytron.com	$42
Laser Pointer	Office Depot	$20
741 Op Amp	Stockroom	
NPN Transistor	Stockroom	
Breadboard	Fry's Electronics	$30
Solar Panels	Electronics Goldmine	$30

Appendix A: PIC C Code (Laser Pointer)

```
// Author: Mike Chan and Matt Williams
// Date 12/3/06
// Purpose: Circuit to control pulsing of the laser.

#include<p18f452.h>
#include<timers.h>

void main(void);

void main(void)
{
        char out, temp;
        T0CON = 0x88;  //start clock, no prescale
        TRISD = 0;

        while(1)
        {
        temp = TMR0L;
        out = (TMR0H >> 5);
        PORTD = out;
        }

}
```

Appendix B: PIC C Code (Vehicle Control)

```
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        Authors: Mike Chan and Matt Williams
        Laser_Poll_Trans.c
        Date: November 12th
        Purpose: Code which polls a given number of solar panels,
                        A/D converts the input, impliments a coherent
                        detection algorithm, and outputs the index of the
                        largest value via the serial port.
        Notes: This code is designed to poll NPN transistors
                  and will not function using PNP transistors.
*/ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#include <p18f452.h>
#include <timers.h>

// Number of Solar Cells attached
#define NUM_LED 5


// ~~~~~~~~~~~~~~~~~~~ Global Variables ~~~~~~~~~~~~~~~~~~~~~~~~
long storage[NUM_LED];  // Stores coherent detection values
long sum1 = 0; // Stores coherent dection sum during computation
long sum2 = 0; // Stores coherent dection sum from pi/2 shifted function
unsigned long totalsum = 0; // stores the L1 norm of sum1 and sum2

// Function generator variables
int A = 0;
int B = 0;
int i = 0;
int j = 1;

int k = 0; // Counts number of quarter periods elapsed

int poll = 0; // Index of polled solar cell


// ~~~~~~~~~~~~~~~~~~~ Function Prototypes ~~~~~~~~~~~~~~~~~~~~~~
void main(void);
void dft(void);
long abs(long x);
int max(long x[NUM_LED]);

// Timer Interrupt Code
```

```
#pragma code high_vector = 0x08
void low_interrupt(void)
{
        _asm
                GOTO dft
        _endasm
}
#pragma code


void main(void)
{
        long AtoD_Storage;

        poll = 0;  // Polls 1st solar panel



        // ~~~~~~~~~~~~~~~~ I/O ports ~~~~~~~~~~~~~~~~~~~~~~
        // Notes: PORTA used for A/D conversion
        //                      PORTB used for polling solar cells
        //        PORTC used for serial output
   //                   PORTE used to send enable flag
        TRISA = 1;  //Input
        TRISB = 0;  // Output ports
        TRISC = 0;
        TRISE = 0;

        //Set up Serial Port for Master Mode output
        SSPCON1 = 0x20;
        SSPSTAT = 0x40;

        // ~~~~~~~~~~ Timer0 Initialization  ~~~~~~~~~~~~~~~
        // Creates a 16 bit unscaled timer with interrupt
        // to count 1/4 th of a period.
        // Functions with a 40 MHz clock.
        TMR0H = 0xF0;
        TMR0L = 0xEE;
        T0CON = 0x88; // Sets up timer
        INTCON = 0xA0; // Sets up Interrupt


        // ~~~~~~~~~~ A/D Converter Intialization ~~~~~~~~~~~
        // Intializes and turn on A/D converter with input at RA0.
        // Vdd and GND are Vrefs.
        ADCON1 = 0xC0;
```

```
        ADCON0 = 0x80;
        ADCON0bits.ADON = 1;

        // ~~~~~~~~~~ Main Program Loop ~~~~~~~~~~~~~~~~~~~~~
        while(1)
        {

                // Start A/D converter
                ADCON0bits.GO = 1;

                // Extracts digitized value from previous run
                AtoD_Storage = ADRES ;

                // Calculates Coefficients for basis functions
                // (Sum1 and Sum2 represent square waves Pi/2 out of phase)
                if((k > 3) && (k < 8)) // Throws away first and last period
                {
                        if (A == 0x01)
                                sum1 += AtoD_Storage;
                        else
                                sum1 -= AtoD_Storage;
                        if (B == 0x01)
                                sum2 += AtoD_Storage;
                        else
                                sum2 -= AtoD_Storage;
                }

//              // Infinite Loop while A/D finishes
                while(ADCON0bits.GO == 1)
                {
                }

        }
}

// ~~~~~~~~~~ Function Generation and Analysis ~~~~~~~~~~
#pragma interrupt dft
void dft(void)
{

        INTCONbits.TMR0IF = 0; // Resets interrupt
        TMR0H = 0xF0;  // Resets timer
        TMR0L = 0xEE;

        // Generates Square waves.  A and B change every 2 interrupts
        // which creates two square waves Pi/2 out of phase
```

```
if (i < 2)
{
        A = 0x01;
        i++;
}
else
{
        A = 0x00;
        if (i==2)
                i++;
        else
                i = 0;
}
if (j < 2)
{
        B = 0x01;
        j++;
}
else
{
        B = 0x00;
        if (j==2)
                j++;
        else
                j = 0;
}




//sends signal to FPGA after 4 periods
if (k == 15)
{

        // L1 norm of values
        totalsum = abs(sum1) + abs(sum2);
        storage[poll] = totalsum;

        // Sends data when all solar cells have been polled
        if(poll == (NUM_LED-1))
        {
                PORTE = 1; // Sends enable flag
                SSPBUF = max(storage);
                // loop till writing is finished
                while(SSPCON1bits.WCOL==1)
                {
                }
            }

            // Shift to new solar cell
            if(poll < (NUM_LED - 1))
            {
                poll++;
                PORTB<<=1;
            }
            else  // Resets to the start
            {
                poll = 0;
                PORTB = 0x01;
            }


            // Resets coeffs and counters
            totalsum = 0;
            sum1 = 0;
            sum2 = 0;
            k = 0;
            PORTE = 0;

        }
        else
            k++;  // Increment counter

        return;
}

// Absolute value function, drop in replacement for "abs()" which
// the PIC doesn't have
long abs(long x)
{
        if (x < 0)
            return -x;
        else
            return x;

}

// Finds the maximum value in an array
int max(long x[NUM_LED])
{
        int i;
        int index = 0;
        for(i=1; i < NUM_LED; i++)
                if (x[i] > x[index])
                {

                        index = i;
                }

        if( (x[index] >>8) == 0)
                return 0;
        else
                return (0x01<<index);

}
```

Appendix C: Verilog Code

Top Level Module:

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Engineer: Mike Chan and Matt Williams
// Date: 12/3/06
//        Purpose: Creates hardware to accept serial input from a PIC, a decoder to
//                          convert the input, and a FSM to control the tank.
// Inputs:        ext_clk: Clock from the PIC sent during serial transmission
//                                enable: Signal sent from PIC during serial transmission
//                                serial: Serial data
// Outputs:      direction_out: inverted output to control PNP transistors
//                                state: Debug information sent to the LEDs
////////////////////////////////////////////////////////////////////////////////
module Tank_Main(ext_clk, enable, serial, direction_out, state);
        input ext_clk;
    input enable;
    input serial;
    output [2:0] direction_out;
        output [2:0] state;

        wire [7:0] led_dir;
        wire [2:0] temp;
        wire [2:0] direction;
        wire [1:0] translate;

        shift_reg max_dir(ext_clk, serial, led_dir);
        decoder decode_me(led_dir, translate);
        Tank_Control control(enable, translate, direction[0], direction[1], direction[2], temp);

        // Inverts output and displays debug info on LEDs
        assign state = direction_out;
        assign direction_out = ~direction;

endmodule
```

Decoder:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Mike Chan and Matt Williams
// Date: 12/3/06
// Purpose: Decoder to assign panel number to direction
//////////////////////////////////////////////////////////////////////////////////
module decoder(led_dir, dir);
    input [7:0] led_dir;
    output reg [1:0] dir;

        parameter stop = 8'b0;
        parameter up1 = 8'b1;
        parameter left1 = 8'b10;
        parameter left2 = 8'b1000;
        parameter right1 = 8'b100;
        parameter right2 = 8'b10000;

        always @( * )
                case(led_dir)
                        stop: dir = 2'b11;
                        up1:  dir = 2'b00;
                        left1: dir = 2'b01;
                        left2: dir = 2'b01;
                        right1: dir = 2'b10;
                        right2: dir = 2'b10;
                        default: dir = 2'b11;
                endcase


endmodule
```

Shift Register:

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Engineer: Mike Chan and Matt Williams
// Date: 12/3/06
//        Purpose: Serial to parallel shifter
//
////////////////////////////////////////////////////////////////////////////////
module shift_reg(clk, serial, parallel);
    input clk;
    input serial;
    output [7:0] parallel;

            reg [7:0] registers;

            // Shifts
            always@(posedge clk)
                        registers <= {registers[6:0], serial};


            // Assigns outputs
            assign parallel = registers;


endmodule
```

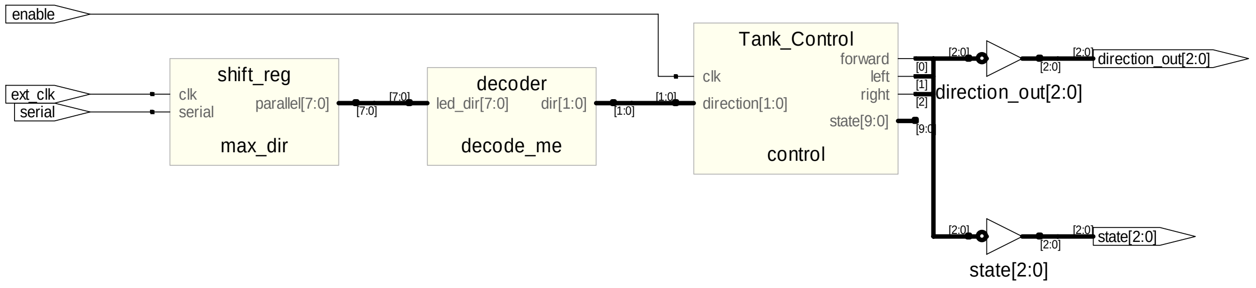
Tank Control FSM:

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Mike Chan and Matt Williams
// Date: 12/3/06
// Purpose: FSM to control the tank
////////////////////////////////////////////////////////////////////////////////
module Tank_Control(clk, direction, forward, left, right, state);
    input clk;
    input [1:0] direction;
    output reg forward;
    output reg left;
    output reg right;
             output reg [9:0] state;

            parameter stop = 10'b1;
            parameter up1 = 10'b10;
            parameter up2 = 10'b100;
            parameter up3 = 10'b1000;
            parameter left1 = 10'b10000;
            parameter left2 = 10'b100000;
            parameter left3 = 10'b1000000;
            parameter right1 = 10'b10000000;
            parameter right2 = 10'b100000000;
            parameter right3 = 10'b1000000000;

            parameter go_up = 2'b00;
            parameter go_left = 2'b01;
            parameter go_right = 2'b10;

             reg [9:0] next_state;


             // state change
             always@(negedge clk)
                        state <= next_state;

            // Next state logic
             always@ ( * )
                        case( state )
                                    stop: if(direction == go_up)
                                                            next_state <= up1;
                                                else if(direction == go_left)
                                                            next_state <= left1;
```

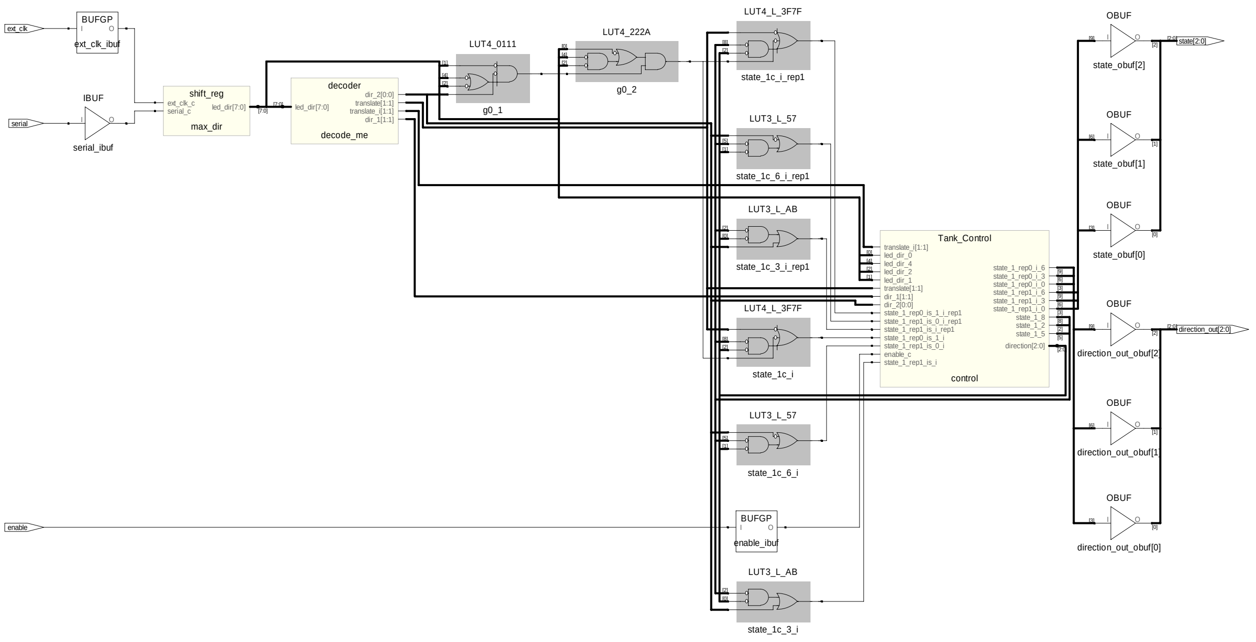
```
                else
                        next_state <= right1;
up1: if(direction == go_up)
                        next_state <= up2;
                else
                        next_state <= stop;

up2: if(direction == go_up)
                        next_state <= up3;
                else
                        next_state <= stop;

up3: if(direction == go_up)
                        next_state <= up3;
                else
                        next_state <= stop;
left1: if(direction == go_left)
                        next_state <= left2;
                else
                        next_state <= stop;

left2: if(direction == go_left)
                        next_state <= left3;
                else
                        next_state <= stop;

left3: if(direction == go_left)
                        next_state <= left3;
                else
                        next_state <= stop;

right1: if(direction == go_right)
                        next_state <= right2;
                else
                        next_state <= stop;

right2: if(direction == go_right)
                        next_state <= right3;
                else
                        next_state <= stop;

right3: if(direction == go_right)
                        next_state <= right3;
                else
                        next_state <= stop;
                default: next_state <= stop;
        endcase


        //Output logic
        always@ ( * )
                case(state)
                        up3: begin
                                        forward <= 1;
                                        left <= 0;
                                        right <=0;
                                        end

                        left3: begin
                                        forward <= 0;
                                        left <= 1;
                                        right <=0;
                                        end

                        right3: begin
                                        forward <= 0;
                                        left <= 0;
                                        right <=1;
                                        end

                        default: begin
                                        forward <= 0;
                                        left <=0;
                                        right<=0;
                                        end
                        endcase

endmodule
```

Appendix D: FPGA Schematics

Top Level RTL:



Top Level Tech:



ext_clk
BUFGP
ext_clk_ibuf
IBUF
serial
serial_ibuf
shift_reg
ext_clk_c
serial_c
led_dir[7:0]
max_dir
decoder
led_dir[7:0]
dir_2[0:0]
translate[1:1]
translate_i[1:1]
dir_1[1:1]
decode_me
LUT4_0111
g0_1
LUT4_222A
g0_2
LUT4_L_3F7F
state_1c_i_rep1
LUT3_L_57
state_1c_6_i_rep1
LUT3_L_AB
state_1c_3_i_rep1
LUT4_L_3F7F
state_1c_i
LUT3_L_57
state_1c_6_i
BUFGP
enable_ibuf
LUT3_L_AB
state_1c_3_i
enable
Tank_Control
translate_i[1:1]
led_dir_0
led_dir_4
led_dir_2
led_dir_1
translate[1:1]
dir_1[1:1]
dir_2[0:0]
state_1_rep0_is_1_i_rep1
state_1_rep1_is_0_i_rep1
state_1_rep1_is_i_rep1
state_1_rep0_is_1_i
state_1_rep1_is_0_i
enable_c
state_1_rep1_is_i
state_1_rep0_i_6
state_1_rep0_i_3
state_1_rep0_i_0
state_1_rep1_i_6
state_1_rep1_i_3
state_1_rep1_i_0
state_1_8
state_1_2
state_1_5
direction[2:0]
control
OBUF
state_obuf[2]
OBUF
state_obuf[1]
OBUF
state_obuf[0]
state[2:0]
OBUF
direction_out_obuf[2]
OBUF
direction_out_obuf[1]
OBUF
direction_out_obuf[0]
direction_out[2:0]