Daisy II

By: Steve Rothen
EEL5666
Spring 2002

Table of Contents

Abstract………………………………………………………..…… 3
Executive Summary……………………………………………. 4
Introduction……………………………………………………….. 4
Integrated System……………………………………………… 5
Mobile Platform………………………………………………... 8
Actuation……………………………………………….……...9
Sensors ……………………………………………………..……. 10
Behaviors ……………………………………………………..…. 13
Experimental Layout and Results……………………….……. 13
Conclusion……………………………………………………….16
Documentation…………………………………………….......17
Appendices………………………………………………..…...18

Abstract

Daisy II is an autonomous robot that has the ability to play fetch with itself. The basic function of Daisy II is to locate a ball, fetch the ball, then bring the ball back to a specified location. Daisy 2 is a mobile robot that will navigate using an IR beacon to locate the ball. IR and bump sensors will provide the necessary object avoidance required for motion. The unique aspect of Daisy II is the controller. She uses an FPGA chip made by Altera. This is a logic chip so Daisy II's actions and behaviors are all logic based.

Executive Summary

Daisy II is a ball retrieving robot constructed around the Altera FPGA. The field programmable gate array (FPGA) is a logical device which can simulate and emulate any logical device. This FPGA is the only controller for the robot. All the necessary signals are provided through a second board. Using this board complicated this robot since some of the features necessary for robot development are not available. Daisy II is built completely around logic, meaning every signal is analyzed at every clock pulse and the proper actions are performed based off these inputs. The added hardware is essential to Daisy II. This board provides Daisy II with obstacle avoidance and object location information.

Introduction

The main idea behind Daisy II is my dog Daisy. She is a little bit over two year of age and loves to play with her tennis ball. When she wants to play and I am not interested she will play with herself. She will throw the ball then retrieve the ball. This inspired me to design a robot that will serve the same function. Daisy II will have the ability to locate and retrieve the ball while incorporating object avoidance techniques. Daisy II will be built around a single Field Programmable Gate Array (FPGA). This will provide a logic basis to dictate behaviors. The FPGA is integrated onto an Altera (University Program) board. The FPGA used on this board is the EPF10K20RC240-4. Any necessary documentation can be at www.altera.com.

Integrated System

The overall design of the robot will be centered on the FPGA and a second hardware board. This will be the brain of the robot. The board used on the robot has several drawbacks. The board has no extra built in components that are essential to robot design. The board does not have any built in A/D converters or any motor driving circuitry. It more then makes up for it in capabilities. This board has the ability to simulate any logic gate along with any memory applications. The Altera board has around 170 assignable I/O pins. These will be utilized to talk to external comparator networks and motor drivers. It also has a 25.175 MHz crystal oscillator so any necessary timing operations can be performed.

The software design for the robot will be built around a basic state machine. The FPGA can be programmed in VHDL. This versatile language can be used to design a state machine and any homemade component that the robot requires. The most convenient aspect of using this board is the ability to simulate the processor and having the ability to see what every signal is in every state. The main purpose of using this board was to test my software skills.

The design for the robot was a central state machine controlling all aspects of motion and behavior. This state machine takes in all the necessary signals and outputs the proper control signals for motion. The logic behind the controller is simple. The signals are all prioritized, that is obstacle avoidance takes precedence over everything. At every clock pulse the signals are examined if an obstacle IR detector gives a true signal the robot moves away from the object. The hierarchy is as follows:

Possible	IR Detectors			Beacon		Motor Control	
Conditions	IR_1	IR_2	IR_3	1	2	Left	Right
1	1	1	0	X	X	R	R
2	1	0	0	X	X	R	H
3	0	1	0	X	X	H	R
4	0	0	1	X	X	F	F
5	1	0	1	X	X	F	R
6	0	1	1	X	X	R	F
7	1	1	1	X	X	R	F
8	0	0	0	X	X	Beacon Control	
9	X	X	X	1	1	F	F
10	X	X	X	1	0	H	F
11	X	X	X	0	1	F	H
12	X	X	X	0	0	F	R

1 – TRUE 0 – FALSE X – Don't care
F – Forward R – Reverse H - Hold

These 12 states cover all possible signal states that might be important. The first seven are strictly for obstacle avoidance. The sensors network will provide a true signal when an obstacle is detected. The same principle is applied to the beacon network. When the beacon is in front of the sensor the network will output a true signal. If the three IR detectors are false the motor control is handed over to the beacon signals. If no signal is found then Daisy II spins clockwise. These signals were derived from a comparator network between a GP1D12 and an adjustable voltages source. This allows for calibration of the IR detectors for different environments.

The code for the motor controller was written entirely in VHDL. The necessary control signal are derived from an external; hardware board. The Altera board used has numerous assignable I/O pins. The software used to program my board is MaxPlus II. This software package allows for traditional programming along with a graphical interface tool. Components can be designed with structure and behavior and represented using only the components I/O pins. This is the method I chose to develop my program. The component I used in programming Daisy II were all designed with there own

structure and behavior. The main controller unit handles all the motor control. The design for the components along with the code behind the devices can be found in the appendix. This isn't the only component used. I added a counter network that controls a register. This register takes in the motor control signals from the controller and outputs them to the motor driver network. This register is used to prevent the motors from switching directions too rapidly. The counter controls when the register gets updated.

Additional components were required to handle and interpret the signals properly. A de-bouncer was added to hold the signal in true longer then a single clock pulse. Since the motor driver get updated only every half second it was necessary to hold the IR and beacon signals longer. The de-bouncer was constructed using several D flip-flops. These were connected in series to simulate a shifter right. The outputs at each flip-flop are ORed together.

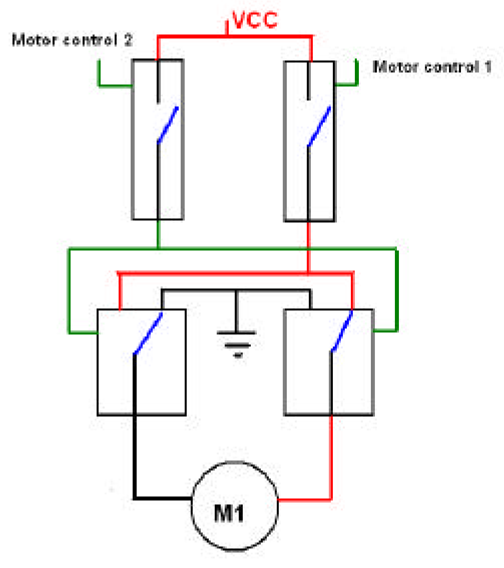
The only remaining component is a 4 by 2 switch. This switch is controlled by the retrieval mechanism switch. This switch is true when a break beam is broken. It is only broken by the ball which tells the controller the ball is in the holding area. This switch controls which beacon is the target beacon. The beacon will either be the ball or I. This switch controls which beacon signals are considered in the guidance component.

Mobile Platform

The actual platform will be constructed from airplane-wood frame. It will be circular in design and have the ability to hold along the electronics along with the necessary motors and sensors. The circular design will allow the robot to rotate itself 360 deg without any worries hitting anything. The two drives wheels will be located on two opposite corners and a third balancing leg will be situated at a perpendicular to the two wheels. The forth side is the front and will contain the retrieval mechanism. The sensors will be located throughout the robot base. The main two IR and bump sensors will be located on the front looking portion of the robot, with a third IR and bump located at the rear. The platform is 10 in. in diameter to accommodate the Altera board. The add-on hardware board and the Altera board run parallel to each other. They are mounted to the board using L bracts and screws. The cables connecting all the components were made long enough to ensure the robot can function when dissembled from the platform. Since the hardware board was constructed using standard components and lots of soldier the ability to take the board apart is essential. The only other unique platform design is the ball retrieval mechanism. This is a simple design using wood in the shape of a Y. This is used to guide the ball into the holding area. The design of the mechanism will be further looked at in the sensor section.

Actuation

The main actuation of the robot will be in the form of rotation motion i.e. wheeled movement. The basic robot function will be ball retrieval. So wheel movement is the only actuation for Daisy II. This may seem every simple but it is a lot more complicated then it first appears. I want to control the motors using basic logic. To achieve this, a second hardware setup was needed. This network takes in two logic signals and moves the motor is either direction. This network was created using to DPST and two SPST relays. The connections are given below:



Control Lines		Motor
1	1	Reverse
1	0	Forward
0	1	Off
0	0	Off

The second relay was needed since the current necessary to flip the two DPST relays simultaneously was not provided by the Altera board. This setup ensures that the power driving the motors is completely separate from the Altera's power connection. This prevents the motor from drawing too much current, damaging the FPGA. This

method is nice but doesn't allow me to change speeds, only direction. For my application this was an acceptable trade-off.

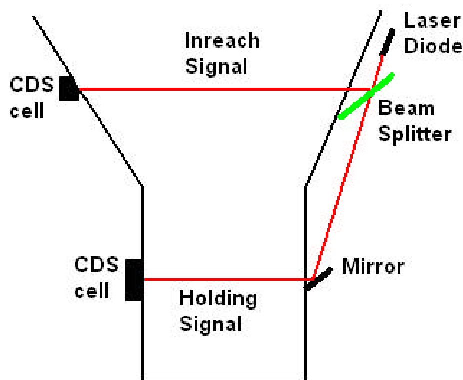
Sensors

The main motion related sensors will be the standard IR emitter/detector setup along with the bump sensor. These IR combinations are provides using the GP2D12. This is a nice little device which will output a voltage based of the amount of IR detected. This is usually taken into an A/D and sampled. This value can be used in the code to determine when an obstacle is located in front of the sensor. Using the Altera board I do not have the luxury of the A/D converter. I achieved the functionality of the A/D converter by using a comparator network. The part used will be the LM339 which is a quad comparator. The principle behind the comparator is simple. It is made up of four op-amps, when the voltage level on one pin is higher then the other a true signal will be outputted. The main thing you need to realize is the output is an open collector output. These means when the signal should be true it can only be seen as +5V when a pull up network is added. This was unknown to me and I thought the comparator would output a +5V signal. The output on the other hand is undeterminable until I added the pull-up network. This provided me with a +5V true signal. I attempted to read the signal from the board and was unsuccessful. It seems the signal is stable enough to be calculated by the Altera board. I solved this problem by taking the signals though a hew inverter (74_04.) This did invert the signal but that was ok since I can change my code to look at the inverse. This inverter acted as a buffering system stabilizing the signal. This network was all built on the hardware board. The comparator takes in two voltages: the sensor

voltage and an adjustable voltage source. This adjustable voltage source is essential to ensure the device will work in a wide variety of environments. The voltage source was created using a voltage divider network with a potentiometer. When testing the robot small adjustment can be made to this voltage source to set when the IR detector will sense an object. The output voltage of the sensor is between 1.5-2.5V so the adjustable voltage source has the same swing. These sensors handle the obstacle avoidance aspect for Daisy II.

A similar network is used for ball location. The ball itself has a 56.8 kHz clock driving an IR beacon. This modulated IR is different from the GP2D12 co they will not interfere with each other. The robot is equipped with several hacked IR detectors modulated for the same frequency. These output a voltage similar to then GP2D12 and will be handled in the same manor. This will allow the robot to locate the ball and retrieve it. A similar beacon will be placed by itself as the home base. Switching between the beacons will be handled by the code and a single control signal. This control signal will provided by the retrieval mechanism.

The retrieval mechanism is based off a break beam sensor. A laser beam is spitted into two beams and directed into two CDS cells. The cells are standard CDs cell which change resistance based off light intensity. The retrieval mechanism is located on the bottom of the robot so surrounding light is minimized. The layout for the mechanism is given below:



The laser diode was obtained through Radio Shack and the beam splitter was purchased through Edmund Scientific. The beam splitter is a thin piece of glass which allows 50% if the beam to pass through and the reaming 50% to be reflected. The CDS cells vary resistance based off different lighting conditions. There are two CDS cells provide the controller with two different signals. The first signal is the inreach signal. This provides the robot with the location of the ball when it is under the platform. The platform will block the direct IR signals so this inreach signal will tell the robot to proceed forward. The second signal informs the robot when the ball is in the holding area. The signal Testing was used to determine the best condition to provide myself with the greatest resistance swing.

Resistance seen Vs Possible Lighting Conditions

		Laser	
Lighting		On	Off
Overhead	no Cover	1.1k	2.8k
	Cover	2.1k	35k
Front	no Cover	1.2k	6k
	Cover	1.5k	44k
None		1.5k	.7M

When fully assembled a 1.5 volt swing occurred when an object was obstructing the laser beam. These signals were taken into a comparator and out through an inverter similar to the IR detectors. The remaining sensors include an IR beacon. I constructed

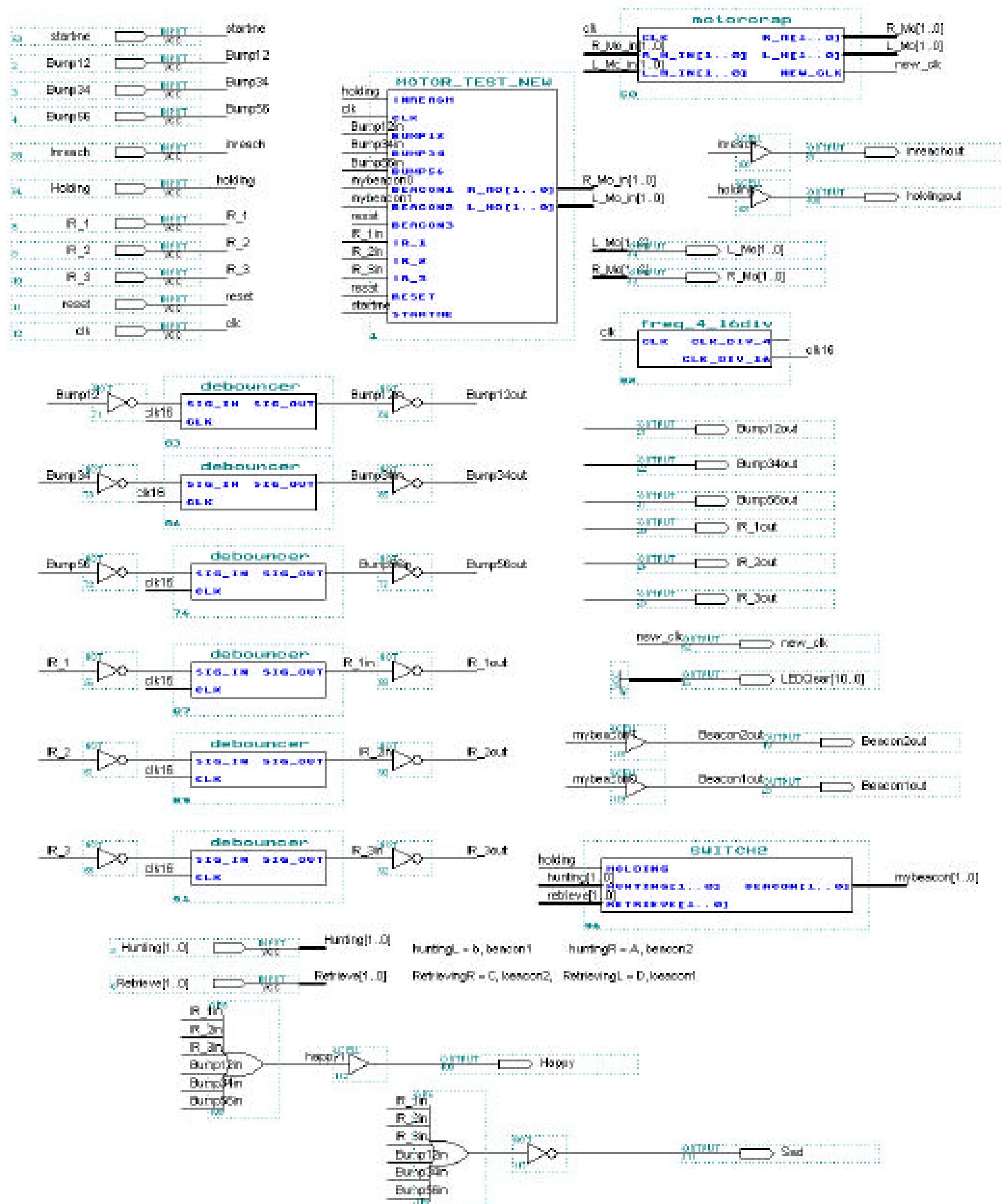
an IR beacon running at 40 kHz and placed it inside the ball. This will act as the target object. When retrieved the target will be a second beacon located by myself. Both are modulated at the same frequency and are read by Radio Shack IR detector cans. These were hacked allowing an analog signal out.

Behaviors

Daisy II will exhibit several behavioral actions. The two most significant will be a facial expression showing what she is thinking. She will show a smile when she is fetching and a frown when she is faced with an obstacle. This was achieved using several LEDs and a 5V reed relay. The control signal for the smile is provided by the Altera board. The IR and bump signals are Ored together so if any of the obstacles are detected a frown is shown. The smile is created in the same fashion using an inverter for the control signal. The main function of Daisy II is also a behavior. She will locate a ball and bring it back to me.

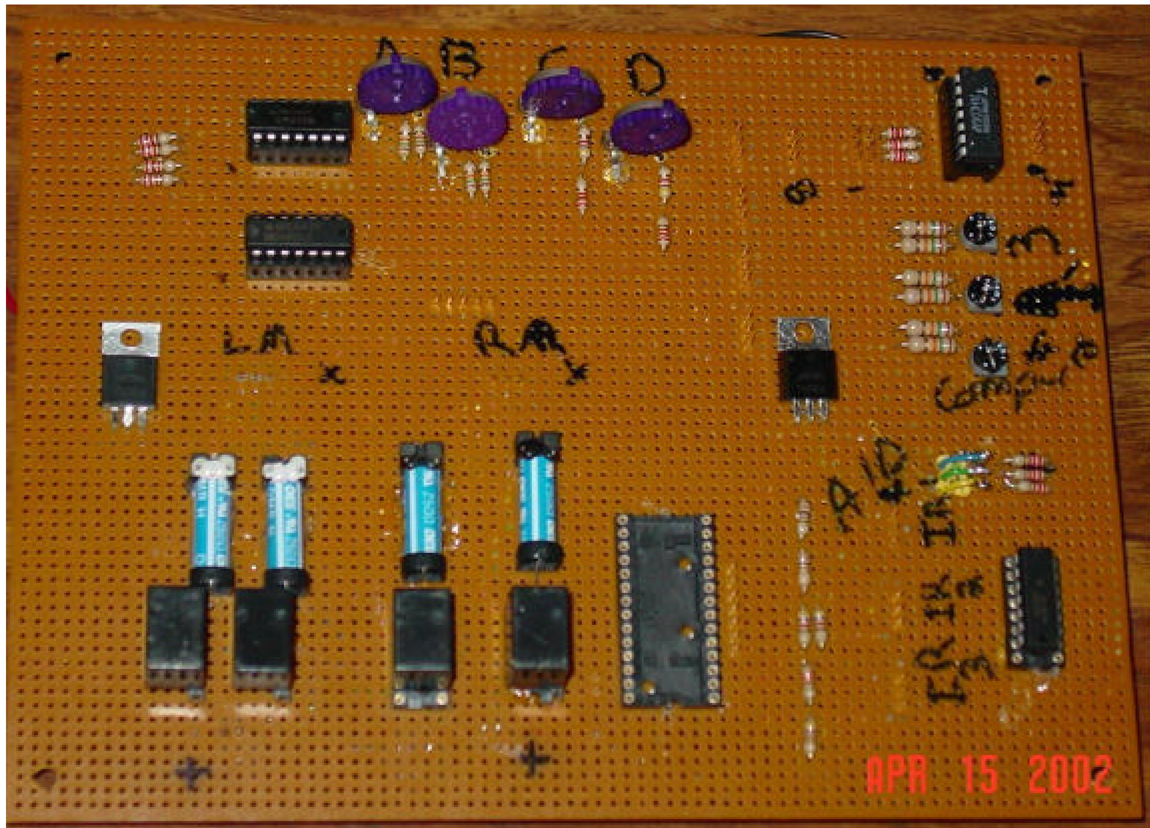
Experimental Layout and Results

The software design for Daisy II is pretty straight forward. She was built around one main controlling unit. MaxPlus II and VHDL make design work very simple. Below is a graphical representation for the program code. The left starts with the declaration of the inputs and the right contains the outputs. The components are made on a need to have basis. MaxPlus II gives the programmer the ability to design their own components. These components have their own structure and behavior. The code behind all the components can be found in the appendix along with some timing analysis.



The coding is only half the robot. In order for the robot to interpret the signals coming in addition circuitry was required.

Below is a picture of the hardware add-on board.



The bottom left contains all the motor driver circuitry. The control pins for the motors and out to the board are the male pins on both sides of the components. On the bottom next to the motor driver is the external A/D I implemented and later discarded. Following around to the right is the obstacle avoidance circuitry. The pins out to the IR modules supply the power and data lines to and from the modules. The potentiometers on the right are for calibration of the IR detectors. The top part of the circuitry is to handle the IR beacon interface. They also have there own potentiometer network. The six pins in the center of the board (up and to left) are the connections to the Bump sensors and the break beam sensors. There are two different 5V regulators to keep the two power systems completely separate. The only thing I didn't mention is probably the most

important. In order to make the circuit to work with the Altera board like designed the two boards must share a common ground. This is the most important part. The only thing I did was to run a wire from the ground out of the Altera board to the ground for the add-on board. This seems like common sense but didn't occur to me at the time. I accidentally found this out. I was using a logic probe powered by the Altera board. It has a second ground to connect to the circuit to be tested. This probe provided the key bridge between the two circuits. When removed it didn't work. This led me to find that both circuits need to share a ground.

Conclusion

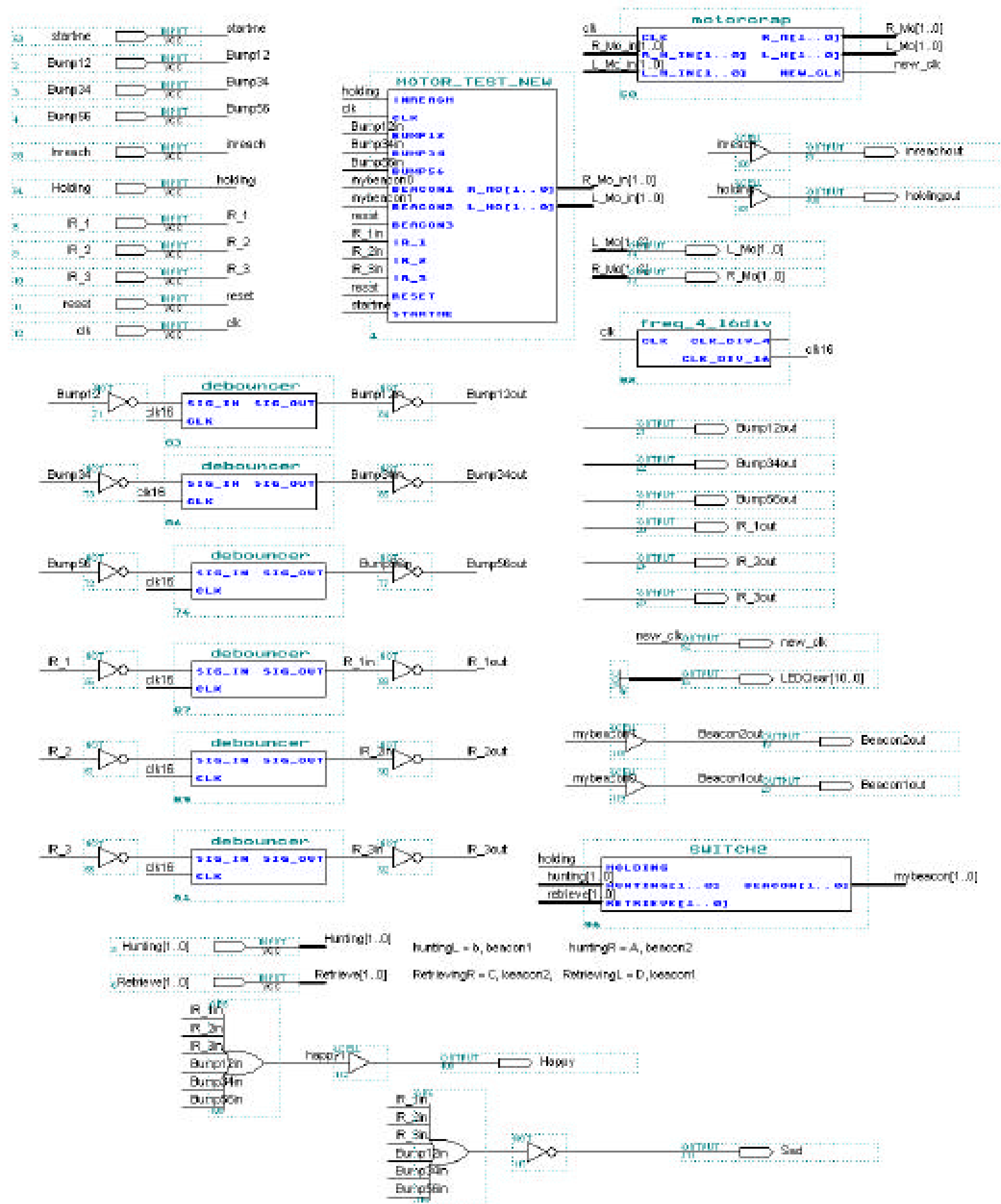
Well I don't really know where to start so I will just say it was a success. Daisy II performs all actions she was supposed to do. She can track a ball, hunt the ball, then bring the ball back to a single beacon. This projects was more challenging the first anticipated. Don't get me wrong I knew it would be hard has heck but the sheer amount of time spent on the robot was unanticipated. I am satisfied at the outcome of the robot. There are several areas of improvement only because I didn't see them until the robot was working. I followed the strict rule, when working don't tweak. I am glad I chose to use the Altera board strictly because the hands on circuit work I was able to do. At first I thought using this board would force me to write some tricky code in order to get it to work properly. That was not the case. I have the most straight forward code. The hardware support was the knife in my side. Overall this robot has taught me more in one semester then I have learned in the past couple of years. I feel this robot was a success and I am very pleased with the outcome.

Documentation

All documentation for the FPGA can be found at www.altera.com. This includes pin assignments and timing diagrams. The reaming component's documentation can be found by searching for the parts spec sheet.

Appendices

Overall Layout



VHDL Code for the Motor Driver Network

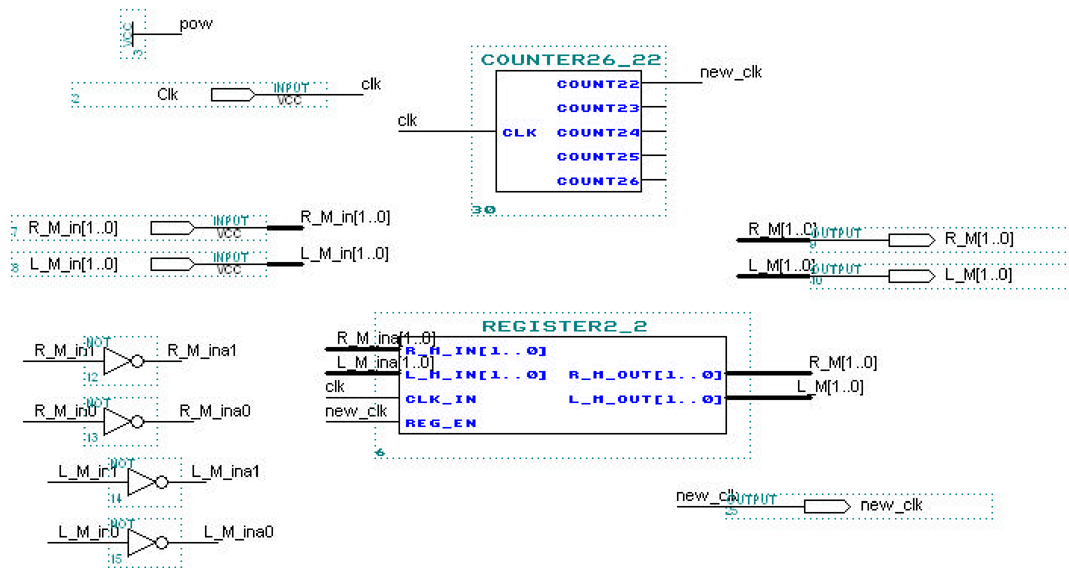
```
-- Steve Rothen

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
ENTITY motor_test_new IS
PORT(
	Inreach : IN	STD_LOGIC;
	clk		: IN	STD_LOGIC;
	Bump12  : IN	STD_LOGIC;
	Bump34  : IN	STD_LOGIC;
	Bump56  : IN	STD_LOGIC;
	Beacon1 : IN	STD_LOGIC;
	Beacon2 : IN	STD_LOGIC;
	Beacon3 : IN	STD_LOGIC;
	IR_1	: IN	STD_LOGIC;
	IR_2	: IN	STD_LOGIC;
	IR_3	: IN	STD_LOGIC;
	reset	: IN	STD_LOGIC;
	startme : IN	STD_LOGIC;
	R_Mo	: OUT	STD_LOGIC_VECTOR(1 DOWNTO 0);
	L_Mo	: OUT	STD_LOGIC_VECTOR(1 DOWNTO 0)
	);
END motor_test_new ;

ARCHITECTURE behavior OF motor_test_new IS
	TYPE state IS (reset1, Start, Crusin);
	SIGNAL present_state : state;
	SIGNAL next_state : state;
	Signal R_Mo_int:	STD_LOGIC_VECTOR(1 downto 0);
	Signal L_Mo_int:	STD_LOGIC_VECTOR(1 downto 0);
	Signal Forward	:STD_LOGIC;
	--signal count:	STD_LOGIC_VECTOR(7 downto 0);

BEGIN
	PROCESS (present_state,inreach, startme, IR_1, IR_2, IR_3, Bump12, Bump34, Bump56, Beacon1,
Beacon2, Beacon3)
	BEGIN
		CASE present_state IS

		When reset1 =>
			next_state <= Start;

		When Start =>
			R_Mo_int <= "00";
			L_Mo_int <= "00";
			if (startme= '1') then
			next_state <=Start;
			else
			next_state <= Crusin;
			end if;

		When Crusin =>
--Obstacle Avoidance
			--Case 1
			if (
				((IR_1 = '1') or (Bump12 = '1')) and
				((IR_2 = '1') or (Bump34 = '1')) and
```

```
		not((IR_3 = '1') or (Bump56 = '1'))
		) then
		--IR_1 & IR_2 & !IR_3 => reverse both
		L_Mo_int <= "11";
		R_Mo_int <= "11";
else
--Case 2
if (
		((IR_1 = '1') or (Bump12 = '1')) and
		not((IR_2 = '1') or (Bump34 = '1')) and
		not((IR_3 = '1') or (Bump56 = '1'))
		) then
		--IR_1 & !IR_2 & !IR_3 => reverse left hold right
		L_Mo_int <= "11";
		R_Mo_int <= "00";
else
--Case 3
if (
		not((IR_1 = '1') or (Bump12 = '1')) and
		((IR_2 = '1') or (Bump34 = '1')) and
		not((IR_3 = '1') or (Bump56 = '1'))
		) then
		--!IR_1 & IR_2 & !IR_3 => reverse right hold left
		L_Mo_int <= "00";
		R_Mo_int <= "11";
else
--Case 4
if (
		not((IR_1 = '1') or (Bump12 = '1')) and
		not((IR_2 = '1') or (Bump34 = '1')) and
		((IR_3 = '1') or (Bump56 = '1'))
		) then
		--!IR_1 & !IR_2 & IR_3 => forward both
		L_Mo_int <= "10";
		R_Mo_int <= "10";
else
--Case 5
if (
		((IR_1 = '1') or (Bump12 = '1')) and
		not((IR_2 = '1') or (Bump34 = '1')) and
		((IR_3 = '1') or (Bump56 = '1'))
		) then
		--IR_1 & !IR_2 & IR_3 => forward left reverse right
		L_Mo_int <= "10";
		R_Mo_int <= "11";
else
--Case 6
if (
		not((IR_1 = '1') or (Bump12 = '1')) and
		((IR_2 = '1') or (Bump34 = '1')) and
		((IR_3 = '1') or (Bump56 = '1'))
		) then
		--!IR_1 & IR_2 & IR_3 => left reverse, right forward
		L_Mo_int <= "11";
		R_Mo_int <= "10";
else
--Case 7
if (
		((IR_1 = '1') or (Bump12 = '1')) and
		((IR_2 = '1') or (Bump34 = '1')) and
		((IR_3 = '1') or (Bump56 = '1'))
		) then
```

```
				--IR_1 & IR_2 & IR_3 => screwed
				L_Mo_int <= "11";
				R_Mo_int <= "10";
			else
--Case inreach
			if (
				not((IR_1 = '1') or (Bump12 = '1')) and
				not((IR_2 = '1') or (Bump34 = '1')) and
				not((IR_3 = '1') or (Bump56 = '1')) and
				(inreach = '0')
				) then
				--!IR_1 & !IR_2 & !IR_3 => Object forward
				L_Mo_int <= "10";
				R_Mo_int <= "10";
			else

--Finding Object
			--Case 8
			if (
				not((IR_1 = '1') or (Bump12 = '1')) and
				not((IR_2 = '1') or (Bump34 = '1')) and
				not((IR_3 = '1') or (Bump56 = '1')) and
				((Beacon1 = '0') and (Beacon2 = '0'))
				) then
				--!IR_1 & !IR_2 & !IR_3 => Object forward
				L_Mo_int <= "10";
				R_Mo_int <= "10";
			else
			--Case 9
			if (
				not((IR_1 = '1') or (Bump12 = '1')) and
				not((IR_2 = '1') or (Bump34 = '1')) and
				not((IR_3 = '1') or (Bump56 = '1')) and
				(Beacon1 = '0')
				) then
				--!IR_1 & !IR_2 & !IR_3 => Object left, hold left, right forward
				L_Mo_int <= "00";
				R_Mo_int <= "10";
			else
			--Case 10
			if (
				not((IR_1 = '1') or (Bump12 = '1')) and
				not((IR_2 = '1') or (Bump34 = '1')) and
				not((IR_3 = '1') or (Bump56 = '1')) and
				(Beacon2 = '0')
				) then
				--!IR_1 & !IR_2 & !IR_3 => Object right, hold right, left forward
				L_Mo_int <= "10";
				R_Mo_int <= "00";
			else
			if (
				not((IR_1 = '1') or (Bump12 = '1')) and
				not((IR_2 = '1') or (Bump34 = '1')) and
				not((IR_3 = '1') or (Bump56 = '1')) and
				(Beacon3 = '0')
				) then
				--!IR_1 & !IR_2 & !IR_3 => Object right, hold right, left forward
				L_Mo_int <= "10";
				R_Mo_int <= "11";
			else
```

```
                                        L_Mo_int <= "11";
                                        R_Mo_int <= "10";
                                end if;
                                end if;
                                end if;
                                end if;
                                end if;
                                end if;
                                end if;
                                end if;
                                end if;
                                end if;
                                end if;
                                end if;
                next_state <= Crusin;
                                        END CASE;
                END PROCESS;

states: process (clk)
                begin
                if (reset = '0') then
                  present_state <= reset1;
                else
                        if (clk'event and clk = '1') then
                                present_state <= next_state;
                        else
                                present_state <= present_state;
                        end if;
                end if;
end process states;

                R_Mo <= R_Mo_int;
                L_Mo<= L_Mo_int;


                END behavior;
```

The Layout for the Motor Driver Interface



VHDL Code for the Register and the Counter

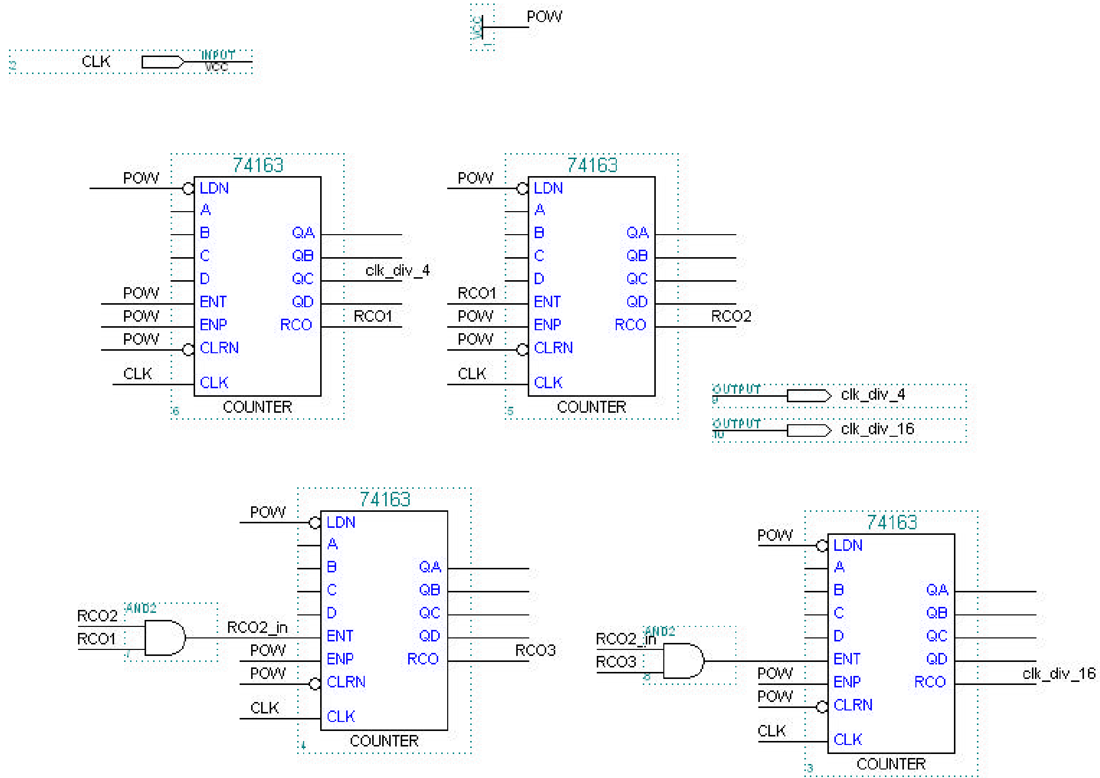
```
--Steve Rothen

library ieee;
use ieee.std_logic_1164.all;


-- ++++++++++++++++++++++++++++++++++++++++++++
-- + Entity section                                                  +
-- ++++++++++++++++++++++++++++++++++++++++++++
entity REGISTER2_2 is port(
        R_M_in   : in std_logic_vector(1 downto 0);
        L_M_in   : in std_logic_vector(1 downto 0);
        Clk_in   : in std_logic;
        reg_en   : in std_logic;
        R_M_out : out std_logic_vector(1 downto 0);
        L_M_out : out std_logic_vector(1 downto 0)
);
end REGISTER2_2;

--------------------------------------------------------------
-- Structure
--------------------------------------------------------------
ARCHITECTURE behavior of register2_2 is

--signal LE1: std_logic;
begin
        --LE1 <= reg_en;
        proc: process(Clk_in, reg_en)
                begin
                if (Clk_in'event and Clk_in ='1') then
                  if (reg_en='1') then
                        R_M_out <= not R_M_in;
                        L_M_out <= not L_M_in;
                  end if;
                end if;
        end process proc;
end behavior;
```

Counter

```
--Steve Rothen
library ieee;
use ieee.std_logic_1164.all;
Use ieee.std_logic_unsigned.all;


ENTITY counter26_22 IS
                PORT(
                        clk                     : IN        STD_LOGIC;
                        count22         : OUT       STD_LOGIC;
                        count23         : OUT       STD_LOGIC;
                        count24         : OUT       STD_LOGIC;
                        count25         : OUT       STD_LOGIC;
                        count26         : OUT       STD_LOGIC

                        );
                END counter26_22 ;

architecture seq of counter26_22 IS
signal count: std_logic_vector(25 downto 0);
Begin
```

```
        Process(clk)
        Begin
                IF (clk'event and clk = '1') Then
                        count <= count + '1';
                ELSE
                        count <= count;
                END IF;
        END Process;
        count26 <= (count(25) and count(24) and count(23) and count(22) and count(21) and count(20)
                        and count(19) and count(18) and count(17) and count(16) and count(15) and count(14)
                        and count(13) and count(12) and count(11) and count(10) and count(9) and count(8)
                        and count(7) and count(6) and count(5) and count(4)
                        and count(3) and count(2) and count(1) and count(0));

        count25 <= (not count(25) and count(24) and count(23) and count(22) and count(21) and count(20)
                        and count(19) and count(17) and count(17) and count(16) and count(15) and count(14)
                        and count(13) and count(12) and count(11) and count(10)and count(9) and count(8)
                        and count(7) and count(6) and count(5) and count(4)
                        and count(3) and count(2) and count(1) and count(0));

        count24 <= (not count(25) and not count(24) and count(23) and count(22) and count(21)
                        and count(20) and count(19) and count(18) and count(17) and count(16) and
                        count(15) and count(14) and count(13) and count(12) and count(11) and count(10)
                        and count(9) and count(8) and count(7) and count(6) and count(5) and count(4)
                        and count(3) and count(2) and count(1) and count(0));


        count23 <= (not count(25) and not count(24) and not count(23) and count(22) and count(21)
                        and count(20) and count(19) and count(18) and count(17) and count(16) and count(15)
                        and count(14) and count(13) and count(12) and count(11) and count(10)
                        and count(9) and count(8) and count(7) and count(6) and count(5) and count(4)
                        and count(3) and count(2) and count(1) and count(0));


        count22 <= (not count(25) and not count(24) and not count(23) and not count(22) and
                        count(21) and count(20) and count(19) and count(18) and count(17) and count(16)
                        and count(15) and count(14) and count(13) and count(12) and count(11) and count(10)
                        and count(9) and count(8) and count(7) and count(6) and count(5) and count(4)
                        and count(3) and count(2) and count(1) and count(0));


End seq;
```

Layout for the Clock Divider



Layout for the Debouncer