A Soccer Playing Robot

Josh Apple
Torrey Frank
Matthew Saylor
Ted Siegel
Indiana University-Purdue University Fort Wayne

Department of Engineering

ENGR 410-ENGR411

Capstone Senior Design Project

Report #2

Project Title: A Soccer Playing Robot

Team Members: Mr. Josh Apple (ME)
               Mr. Torrey Frank (EE)
               Mr. Matthew Saylor (EE)
               Mr. Ted Siegel (ME)

Faculty Advisors: Dr. Yanfei Liu (ECE)
                  Dr. Jiaxin Zhao (ME)

Date: May 3, 2009
# TABLE OF CONTENTS

0. TABLE OF CONTENTS ........................................................................................................ 1  
1. ACKNOWLEDGEMENTS .............................................................................................. 4  
2. ABSTRACT ................................................................................................................... 5  
3. DETAILED FINAL DESIGN .......................................................................................... 6  
3.1. MASS ANALYSIS ...................................................................................................... 6  
3.2. ROBOTIC PLATFORM .............................................................................................. 6  
3.3. PNEUMATIC SYSTEM .............................................................................................. 8  
  3.3.1. PROJECTILE MOTION OF A SOCCER BALL .................................................. 8  
  3.3.2. SWINING KICKER VELOCITY ANALYSIS ...................................................... 8  
  3.3.3. CYLINDER SELECTION ................................................................................. 10  
  3.3.4. KICKER STRESS ANALYSIS ......................................................................... 13  
  3.3.5. PRESSURE TANK ANALYSIS ....................................................................... 14  
  3.3.6. COMPONENT ORFICE AND TUBING ANALYSIS ....................................... 16  
  3.3.7. MISCELLANEOUS PNUEMATIC COMPONENTS ............................................ 17  
3.4. MOTOR SYSTEM .................................................................................................... 18  
  3.4.1. MOTOR REQUIREMENTS ............................................................................. 18  
  3.4.2. HUB TO SHAFT ATTACHMENT .................................................................... 21  
3.5. WHEELS .................................................................................................................. 22  
3.6. MOTOR CONTROL .................................................................................................. 22  
3.7. DRIVERS .................................................................................................................. 25  
  3.7.1. MOTOR CONTROLLER ............................................................................... 26  
  3.7.2. SOLENOID DRIVER ...................................................................................... 27  
3.8. SENSORS ................................................................................................................ 28  
  3.8.1. PROXIMITY SENSOR .................................................................................. 28  
  3.8.2. HALL EFFECT SENSOR .............................................................................. 28  
  3.8.3. REED SWITCH ............................................................................................. 30  
3.9. MICROCONTROLLER ............................................................................................. 31  
  3.9.1. MICROCONTROLLER ................................................................................. 31  
  3.9.2. DEVELOPMENT BOARD ............................................................................. 32
1. ACKNOWLEDGEMENTS

We would like to thank Dr. Liu and Dr. Zhao for their comments and recommendations throughout the design and building process. We would like to thank our sponsors Raytheon and IPFW for providing funding for our project. Finally, we would like to thank Dewald Company Inc. for providing the pneumatic components for our kicking mechanism and Texas Instruments for providing the Development Board.
2. ABSTRACT

Our mission was to build a new robot for the RoboCup competition. Based on last year’s RoboCup project it was determined that the Pioneer 3-DX robot would be too slow to be competitive in competition. Our goal was to create an Omni-directional robot that will move at a competitive speed. Our team has designed new components as well as using components designed by the previous team. The solenoid driver and proximity sensing circuit designed by the previous team is utilized in our design. Several designs were proposed and evaluated for the platform, wheel orientation, kicking mechanism, power supply, drivers and microcontrollers. The final design is a combination of a bullet and pyramid shaped platform with four Omni-directional wheels that are controlled by single channel motor drivers. It has a pneumatic swinging kicking mechanism designed to propel the ball 4 meters through the air. The robot is designed to achieve a minimum velocity of 2 m/s. It uses a high level 32-bit microcontroller and is powered by a single 25.9V Li-Polymer battery. The robot uses sensors to enable kicking and report position. The robot is expandable with a CAN bus and a full speed USB port.
3. DETAILED FINAL DESIGN

3.1. MASS ANALYSIS

A major consideration in the selection of motors was the overall mass of the robot and the pneumatic kicker. The robot had not been fully designed at the time these components were being considered, and therefore the mass was an unknown. It was decided that despite this fact the mass could be estimated based on some known material properties and good estimations.

The group had decided on Aluminum 6061-T6 for the structural material and had some approximate data for the mass of many of the other components. The material properties were found from Matweb. The following table shows the estimated weight analysis used to aid in the selection of the motors.

<table>
<thead>
<tr>
<th>Component</th>
<th>Mass (kg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kicker</td>
<td>0.4</td>
</tr>
<tr>
<td>Cylinders</td>
<td>2.4</td>
</tr>
<tr>
<td>Air tank</td>
<td>1.81</td>
</tr>
<tr>
<td>Misc. Pneumatic components</td>
<td>1</td>
</tr>
<tr>
<td>Misc. Electrical components</td>
<td>1</td>
</tr>
<tr>
<td>Batteries</td>
<td>4.1</td>
</tr>
<tr>
<td>Platform</td>
<td>10</td>
</tr>
<tr>
<td>Motors (4)</td>
<td>4</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>25</strong></td>
</tr>
</tbody>
</table>

3.2. ROBOTIC PLATFORM

The design of the platform required careful attention to the limitations outlined in the Rules and Regulations provided by the league and stated in the problem statement. The design first started by taking into consideration the size of the wheels and motors selected for the robot. With this specific information along with the diameter of the ball, the desired wheel orientation, and the size constraints it was possible to create the base of the platform. It was decided that the motors would be mounted on the underside of the plate. A cut-out in the base plate had to be made in order for the ball to enter into the robot one third of its diameter. The dimensions of the ball itself were needed in order to establish how large this cavity should be. The ball was required to have a minimum diameter of 21.64cm. This represented the worst case scenario for the ball entering the convex hull. If the ball is closer to its maximum size, it may not enter the hull as far.

This compromise may result in the loss of some maneuverability of the robot with the ball, but was essential to remain within the rules. The ball was allowed to then enter the hull of the robot 7.21cm, and this was the distance used from the front edge of the plate towards the center. A
distance of 3.605cm on each side of the center line of the plate was then used for the horizontal distance of the required cutout. The ball was modeled and inserted into the drawing. Lines were then drawn tangential to the ball through the points described above which in turn created the angled portions of the cutout. It was later discovered that the distance of 7.21cm from the front edge of the plate was insufficient due to the length of the kicker in its extended state. It was discovered that the base plate with the wheels and the kicker extended would exceed the maximum allowable size. To reduce this overall size the cutout was extended into the center of the plate further, but the angles remained the same. The next task was to create a mounting structure for the kicker itself as well as an overall body for the robot. It was decided that due to the size of the nitrogen tank and the pneumatic components that the pyramid design would not provide an adequate amount of space and the proposed bullet design could have been too heavy. To remedy this situation a combination of the pyramid and bullet designs was utilized. This design allowed for more space and the open areas and kept the weight of the robot to a minimum. The overall height was kept below the maximum and left available vertical space for any vision system components that may have to be placed on top of the robot at a later time. The open areas of the robot were all designed to be small enough that the ball could not damage any components. The large mechanical components were modeled and added to the assembly to give a better visual idea of remaining space for all of the electrical components as well as to gain some rough dimensions of available space. There was plenty of room available for batteries, drivers, microcontrollers, and all of the rest of the components.
3.3. PNEUMATIC SYSTEM

3.3.1. PROJECTILE MOTION OF A SOCCER BALL

The group needed to determine what velocity and angle of the ball would be required to achieve the group’s goal of 5m of travel. The projectile motion equations were utilized. It was known that an angle of 45º would provide the most efficient flight path and was the angle selected for the ball’s motion. With this information, the range equation could be implemented to determine the ball’s required velocity.

\[ R = \frac{v_o^2 \sin(2\theta)}{g} \]

- \( R \) = range
- \( v_o \) = initial velocity of the ball
- \( g \) = gravitational constant
- \( \theta \) = angle of projectile motion

The goal of 5m could be met if the ball traveled 4m horizontally in the air and bounced along for the other 1m for example. The required velocity of the ball was calculated as

\[ v_o = \sqrt{\frac{Rg}{\sin(2\theta)}} = 6.26 \text{ m/s} \quad R = 4 \text{ m} \]

3.3.2. SWINING KICKER VELOCITY ANALYSIS

After the required speed of the ball had been determined, the next step was to determine the velocity of the kicker, that a specified force provided by the cylinder could achieve. In order to determine this velocity the concepts of work and kinetic energy were utilized and are shown below.
\[ W = \int F_c \, ds = F_c \cdot s, \text{ where } F_c = \text{force from cylinder} \]
\[ s = \text{linear displacement of the cylinder} \]

\[ T = \frac{1}{2} m_b v^2 + \frac{1}{2} m_k v^2, \text{ where } m_b = \text{mass of soccer ball} \]
\[ m_k = \text{mass of kicker} \]
\[ v = \text{velocity of kicker and ball system} \]

It was assumed that the ball would only receive half of the energy put into the system by the cylinder due to frictional, impact losses, unaccounted pressure losses, the inertia of the cylinder piston, and the uncertainty of the exact location of the ball on the kicker.

\[ T = \frac{1}{2} W \]

The velocity of the cylinder was then found from the following equation, which was derived from the kinetic energy and work equation above

\[ v = \sqrt{\frac{F_c s}{m_b + m_k}} \]

The required stroke length had to be calculated next in order to obtain a velocity for the cylinder.

![Figure: 3.3 Stroke Length (note drawing not to scale)](image)

The value for the kicker’s displacement was calculated using Pythagorean’s Theorem. Using similar triangles, the value of \( s \) was calculated.

\[ \frac{1.5}{s} = \frac{10}{7.85} \quad s = 1.178'' (30\text{cm}) \]
This result meant that a cylinder with a minimum stroke of 30 mm was required. The mass of the kicker was determined by selecting dimensions that the group felt would be sufficient for the kicker to adequately kick the ball. The mass of the soccer ball was given in the rules and regulations and calculation of the kicker is in the following section. With all of this information, the required velocity of the cylinder was calculated.

\[ v = \sqrt{\frac{F_c s}{m_b + m_k}} = 6.35 m/s , \text{ where } F_c = 1000 N \]

\[ s = 0.03 m \]

\[ m_k = \text{mass of the kicker} = 0.2904 kg \]

\[ m_b = \text{mass of the ball} = 0.45 kg \]

It is important to mention that the force above was varied until the velocity calculation equaled a value greater than or equal to the required ball velocity calculated for the desired range of 4 meters.

### 3.3.3. CYLINDER SELECTION

After reviewing many data sheets from several cylinder suppliers, it became apparent that there were a few constraints that needed to be addressed. The first of which was the speed at which the cylinders could move. In order to determine the velocity the cylinder would be required to have, the following analysis was performed.

\[ \omega = v = 25.03 rad/s , \text{ where } r = \text{length of kicker} = 10 \text{ inches} \]

\[ v = 250.3 \text{ in/s} \]

The fastest velocity that was seen on any of the data sheet from the pneumatic suppliers was 1000mm/s = 39.4in/s.

Using this information as well as the knowledge of the required angular velocity the placement of the cylinder on the kicker was varied until the following result was obtained.

\[ v = \omega r = 955.25 mm/s (37.5 in/s) , \text{ where } r = \text{distance from hinge to center line of cylinder} = 1.5 \text{ inches} \]

The value for \( r \) above was varied and could have been made smaller, but it was known that a mounting bracket had to be attached to the kicker. The mounting bracket was required to secure the cylinder rod to the kicker itself. A minimum distance of 1.5” from the hinge to the cylinder rod would not be changed.

The load on the cylinder due to the ball also had to be taken into consideration and was determined using the following analysis.
Figure 3.4 Load Factor Chart (CKD USA Corporation)

Figure 3.5 Reference Velocity of Cylinder vs. Time

Figure 3.4 was utilized to find the load factor need to determine the required time the cylinder had to actuate. The load factor determined from the figure was 0.7 for the cylinder selected. It is important to note that the figure was selected from many available based on the operating pressure of our pneumatic system. The operating pressure designed for was 100psi which is approximately 7MPa. The bore size selected was 50mm and the load calculated was
approximately 225lb-f (1000N). Utilizing the following equation from CKD the required time was approximately 0.04 s.

\[ v_m = \frac{L}{t} \times \left(1 + \frac{1.5 \times \omega}{100}\right), \]

where \( v_m = \) cylinder velocity = .8255m/s  
\( L = \) stroke length = .03m  
\( t = \) time  
\( \omega = \) load factor = .25

If the cylinder takes longer in practice it is accounted for by the 50% loss of energy assumption as stated above.

According to the employees of Dewald Fluid Power Company, a local pneumatic supply distributor, the pressure change throughout the cylinders stroke is 15% of the applied pressure. This information was based on the standards produced by the National Fluid Power Association. The following analysis shows the results for the system.

\[ F_c = \frac{\pi d^2}{4} \times .85 P_c = 228.6lb - f, \]

where \( d = \) cylinder bore diameter = 1.575in  
\( P_c = 100 \text{ psi-g} \)

The following table shows an excel spreadsheet that was put together in order to easily vary design variables to find an achievable system.
<table>
<thead>
<tr>
<th>Table: 3.2 Cylinder Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desired Range (m)</td>
</tr>
<tr>
<td>4.0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mass of Ball (kg)</th>
<th>Length of Kicker (in)</th>
<th>Length of Kicker (m)</th>
<th>Width of Kicker (in)</th>
<th>Width of Kicker (m)</th>
<th>Thickness of Kicker (in)</th>
<th>Thickness of Kicker (m)</th>
<th>Volume of Kicker (m)</th>
<th>Density of Aluminum T-6 (kg/m³)</th>
<th>Mass of Kicker (kg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4500</td>
<td>10.0000</td>
<td>0.2540</td>
<td>3.5000</td>
<td>0.0889</td>
<td>0.1875</td>
<td>0.0048</td>
<td>0.0001</td>
<td>2700</td>
<td>0.2904</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Force of Cylinder (N)</th>
<th>Force of Cylinder (lb-f)</th>
<th>Kicker Displacement (m)</th>
<th>Velocity of Kicker (m/s)</th>
<th>Velocity of Kicker (in/s)</th>
<th>Angular Velocity of Kicker (rad/s)</th>
<th>Radius of hinge to cylinder (in)</th>
<th>Velocity of Cylinder (in/s)</th>
<th>Cylinder Displacement (m)</th>
<th>Cylinder Displacement (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000.0000</td>
<td>224.8000</td>
<td>0.1995</td>
<td>6.3575</td>
<td>250.2953</td>
<td>25.0295</td>
<td>1.5000</td>
<td>37.5443</td>
<td>0.0299</td>
<td>1.1781</td>
</tr>
</tbody>
</table>

Operating Pressure 100 psig<br>Delta P 15 psi<br>Bore 1.97 inches<br>Stroke 1.178 inches<br># of cylinders 1<br>

NOTE: Load < 228.604 pounds 224.8

### 3.3.4. KICKER STRESS ANALYSIS

Many dimensions of the kicker were predetermined by the projectile motion and size of the soccer ball. The thickness of the kicker however could be varied and was only dependent on the stress that the kicker would undergo. The following analysis was done to calculate the stress.

![Figure: 3.6 Kicker Free Body Diagram](image)
\[ F_{ball} = m_{ball} g \cos(45) = 3.12N, \text{ where } m_{ball} = 450g \]

\[ M_{ball} = Fr = .455N - m, \text{ where } r = .146m \]

\[ M_{cylinder} = F_{cylinder} r = 38.1N - m, \text{ where } r = 0.0381m \]

\[ F_{cylinder} = 1000N \]

\[ I = \frac{bh^3}{12} = 8.2 \times 10^{-10}, \text{ where } b = \text{width} = .0889m \]

\[ h = \text{thickness} = .0048m \]

\[ y = \frac{h}{2} = 0.0024 \text{ (location of maximum stress)} \]

\[ \sigma = \frac{My}{I} = 110 \times 10^6 = 110MPa, \text{ where } M = M_{cylinder} - M_{ball} = 37.6N-m \]

\[ \eta_s = \frac{\text{Yield Strength}}{\text{Calculated Stress}} = 2.5, \text{ where Yield Strength} = 270 \text{ MPa (from Matweb for Alum. T6)} \]

This result proved to the group that the thickness of the kicker was sufficient for the forces that it would undergo. The group would attempt to find Aluminum 6061-T6 stock with a thickness of 3/8”. This was roughly based on the result for the kicker stress analysis, and was believed to be sufficient to withstand any forces from a ball hitting the robot.

3.3.5. PRESSURE TANK ANALYSIS

Based on the findings of the 2007-2008 report #1, it was found that a nitrogen pressure tank provided the highest number of kicks when compared to air and carbon dioxide pressure tanks. The first point worth mentioning was that the air tank could provide consistent pressure throughout each kick as would the nitrogen tank. The biggest difference was the size of the air tank required to achieve multiple kicks. The following table illustrates their results

Table: 3.3 Tank Size and Number of Kicks for Air Tank

<table>
<thead>
<tr>
<th>Pressure Tanks Size (gal)</th>
<th>Pressure Tanks Size (ci)</th>
<th>Number of Kicks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>134</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>269</td>
<td>40</td>
</tr>
<tr>
<td>2.5</td>
<td>672</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>1344</td>
<td>200</td>
</tr>
</tbody>
</table>
Size considerations of the robot are of great importance in this project and the table above showed that the required size of an air tank was very large in comparison to the nitrogen tanks that are available. A 3000 psi-g Nitrogen tank was available in a pressure tank as small as 48 ci. A carbon dioxide tank could be found that was similar in size and pressure capacity as that of a nitrogen tank however, the carbon dioxide tanks had a drawback of their own. The carbon dioxide tanks are filled with a mixture of liquid and gaseous carbon dioxide. As the carbon dioxide is released into, the pneumatic system small amounts of liquid carbon dioxide can escape the tank. The result is that the carbon dioxide experiences a large change in pressure, which changes the state of the carbon dioxide from liquid to gas creating a large increase in the pressure passed through the pneumatic system. This can lead to damaged components and definitely effects the reliability of the pneumatic system’s output. Nitrogen tanks do not have this drawback due to the fact that the nitrogen in the pressure tank is also gaseous. The only criteria remaining for the selection of the nitrogen tank was between a 3000 psi-g tank and a 4500 psi-g tank. The price difference between a 3000 psi-g and a 4500 psi-g tank was approximately $80. To verify that the 3000 psi-g pressure tank would provide the system with a sufficient number of kick, the ideal gas law was required.

The first task was to calculate the initial mass in the nitrogen tank.

\[ m_i = \frac{ZPV}{RT} = 0.00027 \text{kg}, \text{ where } Z = \text{compressibility factor} = 1 \]

\[
\begin{align*}
P & = \text{pressure} = 20.684 \text{ MPa (initial tank pressure)} \\
R & = \text{gas constant} = 0.2986 \text{ kJ/kg-K} \\
T & = \text{temperature} = 300 \text{ K (approximately room temperature & the value for which the compressibility factor for nitrogen equaled 1)} \\
V & = \text{volume of tank} = 0.00118 \text{ m}^3
\end{align*}
\]

The volume of the cylinder, the tubing, the solenoids, and the pressure regulator had to be determined. It was assumed that 12” = 304.8mm in length would be used as an estimate for the volume of the pneumatic components not including the cylinder. It was determined that ¼” diameter pneumatic components would work for this system. The result was that the pneumatic components had a volume of 12.6*10^{-6} m. The cylinder had a volume of 58.8*10^{-6} m. Then,

\[ \Delta m = \frac{ZPV}{RT}, \text{ where } Z = \text{compressibility factor} = 1 \]

\[
\begin{align*}
P & = \text{pressure} = 1.38 \text{ MPa (100psi for extension & 100 psi for retraction)} \\
R & = \text{gas constant} = 0.2986 \text{ kJ/kg-K} \\
T & = \text{temperature} = 300 \text{ K (approximately room temperature & the value for which the compressibility factor for nitrogen equaled 1)}
\end{align*}
\]

The tables in the 2007-2008 report #1 were utilized in the above calculations. This analysis provided the mass lost from the pressure tank after each kick. The pressure inside the tank was then recalculated using the following equation
This result allowed for the determination of the pressure difference in the tank after each kick, which was equal to 0.065MPa. The pressure was plotted against number of kicks in order to determine the maximum number of kicks. The number of kick was plotted until the pressure inside the tank was close to 200 psi in order to ensure that the last kick performed operated at the required pressure. This same analysis was performed on a 3000 psi nitrogen tank with a volume of 48 ci as well.

![Pressure in Tank after each Kick](image)

The results of this analysis were that the 72 ci and 48 ci tanks could produce a maximum of 299 and 221 kicks respectively. The 48 ci tank would therefore provide enough mass that the cylinder could be actuated every 4 seconds for the entire duration of the 15 minute period of the soccer game. The 72 ci tank could actuate every 3 seconds. The smaller weight and size of the 48 ci tank in addition to number of kicks allowable made it clear that this tank would be used.

### 3.3.6. COMPONENT ORFICE AND TUBING ANALYSIS

The forces, pressure, velocity of the cylinder had been analyzed and were proven to be sufficient enough to achieve the group’s desired results. However the system would only function properly if the nitrogen could travel with a velocity equal to or greater than the required velocities. This fact directly influenced the selection of the tubing diameter, port diameters of the cylinder, and solenoid valve. Utilizing the charts provided by CKD, a pneumatic supplier, a cylinder with a bore size of 50 mm could travel approximately $175\text{m/s} = 6890\text{in/s}$ using a 4KB210 valve. This valve had a maximum working pressure of $0.70\text{Mpa} = 101.5\text{psi}$. The port sizes of these valves were $\frac{1}{4}''$ in diameter. This component dimension had then determined the tube diameter. The
issue remaining with the tubing was to determine whether or not the tubing could withstand the operating pressure of the system.

![Figure: 3.8 Allowable Working Pressure at Temperature (CKD USA Corporation)](image)

The graph above was obtained from Nycoil, a pneumatic tubing and fitting supplier. The operating temperature utilized throughout the pressure tank analysis was 300K = 80.3°F. The graph showed that the $\frac{1}{4}$" polyethylene tubing would withstand over 100psi (our operating pressure) over a range of 68°F-130°F. The tubing would have an approximate factor of safety of 1.5 under our expected operating conditions. All of the fittings were selected to have nominal diameter of $\frac{1}{4}$".

### 3.3.7. MISCELLANEOUS PNEUMATIC COMPONENTS

A quick exhaust valve was suggested by a local pneumatic distributor to ensure that the exhaust of the cylinder would in no way limit the speed capabilities of the cylinder. No formal analysis was performed, but the part was utilized since it was donated. Mufflers for the pneumatic system were also suggested by the supplier. The mufflers purpose was to eliminate some of the noise associated with the cylinder’s exhaust. A BM-28 muffler had the $\frac{1}{4}$” dimension our system required. There was some drawback in the use of these mufflers in the form of back pressure, but the compressibility of our system was so low (approximately 1) that the back pressure was much less than 1 psi as shown in the figure below.
3.4. MOTOR SYSTEM

3.4.1. MOTOR REQUIREMENTS

To insure that the velocity goal of the robot moving two meters per second is achieved in all directions, the angular velocity required from the motors must be determined for the special cases in which the greatest losses will occur. These special cases are when the robot is traveling in the 0°, 90°, 180°, and 270° directions. Equation 1 below shows how to determine the rpm from the velocity V.

\[
\text{Revolutions per minute} = \frac{v \times 60}{2 \times \pi \times r}
\]

Eq. 1

Where:

v = velocity
r = radius of the wheel
Taking into consideration that the needed velocity in the y direction is 2 m/s Equation 2 below can be used to determine the needed V in order to obtain a velocity in the y direction of 2 m/s.

\[ V_y = V \cdot \sin \theta \]  

Eq. 2

This value for V comes out to be 2.8 m/s with a \( \theta \) of 45°. Then this velocity can be put in equation one with a radius of 3 inches for the wheel. The result is an rpm value of 355.

Next, the torque requirements can be found by taking into consideration the free body diagram below. When determining the required torque, the acceleration needs to be determined first. Using Equation 3 below in junction with the time (t) of 2 seconds, the time that it should take the robot to reach the desired velocity, the required acceleration can be found. The velocity for the greatest loss situation of 2.8 m/s is again used, and the required acceleration is found to be 1.41 m/s². Next, the force of friction required for the robot (F_μ) required to achieve the acceleration can be found using Equation 4.

\[ V = a \cdot t \]  

Eq. 3
Next, the free body diagram of the wheel below in Figure 3.12 is used to determine the torque requirements (T) from the motor noting that the motion of the robot is propelled by two motors with one or two more correcting its direction of motion.

Using equations five, six, and seven below the torque required from each motor can be found.

\[
\sum F_x - F_\mu = m \cdot a
\]

Eq.4

\[
\sum F - F_\mu = F_m
\]

Eq.5

\[
T = \frac{F_\mu \cdot r \cdot C_\mu}{2} = \frac{m \cdot a \cdot r \cdot C_\mu}{2}
\]

Eq.7
Where
\[ r = \text{radius of wheel (3in)} \]
\[ C_\mu = \text{Correction factor of 20\% for any unaccounted for losses} \]
The required torque per motor (T) needed per motor to meet the requirements list above is found to 1.48Nm.

With the information found above the requirements for the motors needed are 355 RPM and 1.48Nm. The motor selected is manufactured by Groschopp Inc.

Figure: 3.13 Torque vs. Shaft Speed Diagram for the Motor

The figure above shows the motor manufacturer data showing that the motor chosen meets the requirements derived above.

### 3.4.2. HUB TO SHAFT ATTACHMENT

The shaft of the motor has a 0.500 to 4.9995 inch shaft with a 0.125 inch keyway. The hub manufacturer (andymark) provides a hub that will mate to this shaft within .004 inch max, and the keyway on the hub is also 0.125 inch. Therefore, the only modifications needed were to drill and tap the set screw holes as shown below.

Figure: 3.14 Hub Modifications
The motor mounting of the motor would consist of a square type U bolt with an inner measurement would be 3 and 11/16 inch.

### 3.5. WHEELS

The wheels will also be purchased from the [andymark](http://www.andymark.com) website, and come with a bolt pattern that will match up to the hub. The wheel to be purchased can be seen in the figure below. The fasteners for connecting the hub to the wheel will consist of six M5x0.8 bolts 35mm long per wheel/hub. The mating part for the bolt will be a 5mm tall, 8mm wide nylon inserted locknut. The nylon will help prevent loosening due to any vibrations.

![Figure: 3.15 Purchased Omni Wheel](image)

### 3.6. MOTOR CONTROL

The Omni directional control of the robot is based on the assumption that the robot can move at any angle without changing the orientation of the robot. This means the robot does not have to turn to go in a desired direction. To control the robot to move in any direction, the combination of motors, speed and directions are varied. These directions are given by angles relative to the front of the robot going counter clockwise around. The directions are broke down into eight zones, where two motors are at full speed and the other two are varied.
The modeling was done with the forces provided by each motor. Motors 1 and 3 can provide force in the same direction, and motors 2 and 4 can apply force in the same direction. When these forces are added together, the resulting vector is the direction in which the robot will move.

Motor 1 = $\vec{a}$
Motor 2 = $\vec{b}$
Motor 3 = $\vec{c}$
Motor 4 = $\vec{d}$

$\vec{f} = \vec{a} + \vec{b}$
$\vec{g} = \vec{c} + \vec{d}$
$\vec{f} = \sqrt{(\vec{f})^2 + (\vec{g})^2}$
$\theta = \tan^{-1} \frac{\vec{g}}{\vec{f}}$

$\varphi$ = desired direction
$\varphi = \theta - \alpha$

Table: 3.4 $\alpha$ Value
To minimize the power required to run the motors, some angles can be achieved more efficiently by using three motors instead of four. This means that each zone is divided into two regions. This occurs when the forces provided by two motors equals the force that could be supplied by a single motor.

To find at what angle this occurs the full force of three motors are used and theta is then calculated. This will determine which region is the robot operating in. Region A is using four motors and region B is using three motors.

<table>
<thead>
<tr>
<th>Zone</th>
<th>α</th>
<th>θ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>45</td>
<td>+θ</td>
</tr>
<tr>
<td>2</td>
<td>135</td>
<td>-θ</td>
</tr>
<tr>
<td>3</td>
<td>45</td>
<td>-θ</td>
</tr>
<tr>
<td>4</td>
<td>225</td>
<td>-θ</td>
</tr>
<tr>
<td>5</td>
<td>135</td>
<td>-θ</td>
</tr>
<tr>
<td>6</td>
<td>315</td>
<td>-θ</td>
</tr>
<tr>
<td>7</td>
<td>225</td>
<td>-θ</td>
</tr>
<tr>
<td>8</td>
<td>405</td>
<td>-θ</td>
</tr>
</tbody>
</table>

Table: 3.5 Motor Direction, Zone and Region

<table>
<thead>
<tr>
<th>Region</th>
<th>Motor 1</th>
<th>Motor 2</th>
<th>Motor 3</th>
<th>Motor 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zone 1</td>
<td>A 1 1 1</td>
<td>A 1 1 1</td>
<td>A 1 1 1</td>
<td>A 1 1 1</td>
</tr>
<tr>
<td>Zone 2</td>
<td>A 1 1 1</td>
<td>A 1 1 1</td>
<td>A 1 1 1</td>
<td>A 1 1 1</td>
</tr>
<tr>
<td>Zone 3</td>
<td>A 0 1 1</td>
<td>A 0 1 1</td>
<td>A 0 1 1</td>
<td>A 0 1 1</td>
</tr>
<tr>
<td>Zone 4</td>
<td>A 0 1 1</td>
<td>A 0 1 1</td>
<td>A 0 1 1</td>
<td>A 0 1 1</td>
</tr>
<tr>
<td>Zone 5</td>
<td>A 0 1 1</td>
<td>A 0 1 1</td>
<td>A 0 1 1</td>
<td>A 0 1 1</td>
</tr>
<tr>
<td>Zone 6</td>
<td>A 0 1 1</td>
<td>A 0 1 1</td>
<td>A 0 1 1</td>
<td>A 0 1 1</td>
</tr>
<tr>
<td>Zone 7</td>
<td>A 0 1 1</td>
<td>A 0 1 1</td>
<td>A 0 1 1</td>
<td>A 0 1 1</td>
</tr>
<tr>
<td>Zone 8</td>
<td>A 0 1 1</td>
<td>A 0 1 1</td>
<td>A 0 1 1</td>
<td>A 0 1 1</td>
</tr>
</tbody>
</table>

*The letter denotes the value is calculated*
Another special case is the robot moving in the directions of the wheel orientation, 45°, 135°, 225° and 315°. In these directions, only two motors can be used because the other two are perpendicular to the direction of movement. In the following example, a motor with a value 1 is full forward and a -1 is full reverse.

Example Calculation:

\[ f = \frac{e}{\tan(\theta)} \]

\[
\begin{align*}
\phi &= 10 \\
\alpha &= 45 \\
\theta &= \phi + \alpha \\
\theta &= 55 \\
a &= 1 \\
b &= 1 \\
c &= -1 \\
e &= |a| + |c|
\end{align*}
\]

\[
\begin{align*}
f &= 1.4 \\
d &= f - |b| \\
d &= 0.4 \\
g &= \sqrt{e^2 + f^2} \\
g &= 2.44
\end{align*}
\]

The example above shows that to move at a 10° angle, motors 1 and 2 are full forward, motor 3 is full reverse and motor 4 is forward at 0.4 of its full force. It also shows that the effective force in that direction is 2.44.
3.7.1. MOTOR CONTROLLER

The motor controllers are the Syren10 manufactured by Dimension Engineering. The motor controller can support up to 10A continuous and 15A peak, and support voltages from 6V to 24V with the absolute maximum of 30V. The driver has regenerative current, which means that it will return power to the batteries when the robot is decelerating or changing direction. Regenerative current returns the inductive energy from the motor to the battery instead of dissipating it as heat. It has over current and over temperature protection, to insure the user cannot damage the driver if too large of motor is connected. A dipswitch selects between four modes of operation. One mode of operation is a simplified serial mode that uses TTL level single byte serial commands. It can support about 2000 commands per second, with four different baud rates. The commands are a single byte, if a value of 0 is sent the motor will be full forward, a value of 127 is stop and a value of 255 is full reverse. The driver outputs a pulse width modulated signal at 32 kHz. Since the controller does the PWM there will be less overhead for the microcontroller. The quiescent current draw of the motor controller is about 60mA. With the supply voltage of 24V, the power draw of these devices is 1.44W, each.

Figure: 3.20 SyRen10 Motor Controller Circuit

M+: Motor Connection
M-: Motor Connection
B+: Positive Battery Connection
B-: Negative Battery Connection
S1: Primary Signal Input
S2: Secondary Signal Input
V+: 5VDC 10mA Output
V-: Ground
TxX: Transmit Pins of Microcontroller PIN2, PIN18, PIN25, PIN148
3.7.2. SOLENOID DRIVER

The IC used in the solenoid driver circuit is a DRV104 manufactured by Texas Instruments. This is the solenoid driver circuit used by the 2007-2008 RoboCup project team. The DRV104 is a high power efficiency chip, which is designed for harsh conditions and will be very adequate for our project. The driver will open and close the solenoid valve, which will actuate the kicker. This driver requires a 5V power signal, which will be provided by the microcontroller. The solenoid driver will have a maximum current drawn of 375mA and have a power draw of 1.875W.

PIN1: Duty Cycle Adjust
PIN2: Delay Adjust
PIN3: Oscillator Frequency Adjust
PIN4: Master
PIN5: BOOT
PIN6: Out1
PIN7: Out2
PIN8: Vps1
PIN9: Vps2
PIN10: +Vs
PIN11: GND
PIN12: SYNC
PIN13: Status KO Flag
PIN14: Input
Dout: Digital Output of Microcontroller PIN16

Figure: 3.21 DRV104 Chip interface Circuit
3.8. SENSORS

3.8.1. PROXIMITY SENSOR

There will be one capacitive proximity sensor used to detect the ball before the robot will execute a kick. This proximity sensor is the sensor utilized in the 2007-2008 RoboCup project. Automation Direct with the part number of CT1-AP-1A manufactures this sensor. It will be mounted in the front of the robot directly above the kick plate. The sensor can sense an object between 2-15 mm and operates with a voltage between 10V to 30V. The output of the sensor with an object within range will be VCC, which will need to be stepped down to TTL level in order to interface to the microcontroller. The previous RoboCup project designed a transistor circuit to allow a microcontroller to interface with the proximity sensor. The proximity sensor will operate at 24V and have a maximum current draw of about 8mA. This puts the power draw of the sensor at 192mW.

![Figure: 3.22 Proximity Sensor Interface](image)

Din1: Digital Input Pin of Microcontroller PIN6
Pwr: Power
Gnd: Ground
Sgn: Signal

3.8.2. HALL EFFECT SENSOR

There will be Hall Effect sensors used on the wheels of the robot, these sensors will give feedback to the microcontroller and will provide more stability in the control. The motors will be controlled to run for a specified number of counts, instead of a specified amount of time. This gives the robot control advantage and the performance will not degrade as the battery discharges. The Hall Effect sensors will be used as a gear tooth counter. A permanent magnet will be attached to the back of the sensor and when a metal object passes by the sensor, a pulse is formed. The magnets south pole must be mounted on the flat side of the sensor for a gear tooth counter and must be between 50mT and 400mT. Four MLX90217 Hall Effect cam sensors manufactured by Melexis will be used to detect the metal bolts on the sides of the wheels and
give a resolution of 14 counts per revolution. The permanent magnet will be a Neodymium magnet (B331) manufactured by K & J Magnetics, Inc. and is rated at 368mT. The sensor will be placed at the top of the wheel, perpendicular to the orientation of the wheel.

The sensors have built in analog to digital converters, a 10-bit ADC for the signal and a 4-bit ADC for fixed hysteresis. It uses a single Hall plate to detect the change in the magnetic field. The sensors will require a voltage of 5V and output a current of 25mA, so the power draw of the sensors will be 125mW each.

With a resolution of 14 counts, the robot has an accuracy of about 1.35 inches. This means that the robot could travel at the maximum of 1.35 inches between counts which should be acceptable given the size of the field.
\[ s = r \theta \]
s = arc length  
\[ r = \text{radius of wheel} \]
\[ \theta = \text{angle in radians} \]
\[ r = 3'' \]

\[ \frac{360°}{14} = 25.7° \]
\[ 25.7° \times \frac{\pi}{180°} = 0.4485 \]

\[ \theta = 0.4485 \]
\[ s = 0.4485 \times 3'' \]
\[ s = 1.35'' \]

Figure: 3.26 Hall Effect Sensor Circuit

PIN1: Power  
PIN2: Ground  
PIN3: Signal  
DINx: Digital Inputs of Microcontroller PIN66, PIN172, PIN173, PIN174

3.8.3. REED SWITCH

The cylinder used in the kicking mechanism is equipped with two reed switches installed by the manufacturer. These reed switches connected to ground and then to digital inputs of the microcontroller. The reed switch will allow the robot to know when the cylinder is in the
extended position and retracted position. The typical current draw of the reed switch is about 50mA, so the power draw will be about 250mW.

### 3.9. MICROCONTROLLER

#### 3.9.1. MICROCONTROLLER

The high level microcontroller, which we utilized for our design, is the TMS320F28335 Digital Signal Controller (DSC) produced by Texas Instruments. This microcontroller has the speed and architectural complexity to fully handle the immediate tasks of motor command and sensor feedback but which would also be able to accommodate the complexity of real time digital image processing, wireless communication and all decision making tasks required for a fully functional autonomous robot capable of playing RoboCup MSL soccer in real time. It will able to easily integrate with other peripherals including a Laptop or Single Board computer or any microcontroller network through a CAN bus.

The TMS320F28335 is one of the most powerful DSC’s in industry today incorporating the intelligence of a comprehensive microcontroller design with the speed and computational capacity of a Digital Signal Processor (DSP). This device would provide all the functionality needed for our project as well as provide a great platform for other RoboCup teams to build upon. Also, the strong suit of this family of DSC’s is motor control. Very fast processing speed was a key consideration in selecting this DSC. Some of the specific features, which make the chip attractive, are:

- **High-Performance Static CMOS Technology**
  - Up to 150 MHz (6.67-ns Cycle Time)
- **On-Chip Memory**
  - F28335/F28235: 256K × 16 Flash, 34K × 16 SARAM
- **Enhanced Control Peripherals**
  - Up to 18 PWM Outputs
  - Up to 6 HRPWM Outputs With 150 ps MEP Resolution
  - Up to 6 Event Capture Inputs
  - Up to 2 Quadrature Encoder Interfaces
- **Serial Port Peripherals**
  - Up to 2 CAN Modules
  - Up to 3 SCI (UART) Modules
  - Up to 2 McBSP Modules (Configurable as SPI)
  - One SPI Module
- **12-Bit ADC, 16 Channels**
  - 80-ns Conversion Rate
- **Development Support Includes**
  - ANSI C/C++ Compiler/Assembled/Linker
  - Code Composer Studio™ IDE
  - DSP/BIOS™
  - Digital Motor Control and Digital Power Software Libraries
3.9.2. DEVELOPMENT BOARD

In order to utilize the Microcontroller and develop the necessary code to run the robot the eZdspF28335 development board has been chosen to be used. The development board is integrated into the design of the robot. This allows the access to two CAN ports and one serial port. It also allows the design team to develop the robot without designing a PCB to support the microcontroller. The development board needed to boot from flash to allow the robot to function after a power up. Switch 1 must be configured to boot from flash, which is (off, off, off, off).
3.10. USB PORT INTERFACE

The USB port interface will serve many of the same purposes as the serial port. However, one of the key advantages that the USB port will have compared with the DB 9 serial connector port is speed. This is already over 100 times faster than serial communication. However, our USB circuit is designed using an FT232R USB UART IC created by Future Technology Development International Ltd. A high speed port was designed so that it may be possible to download and process real-time digital images when a vision system is integrated into the design in the future. The USB port was chosen over other options like Firewire or Gigabit Ethernet because:

- It was difficult to find a simple gigabit Ethernet or Firewire controller interface
- USB interfaces are very common place in PCs
- A one chip interface solution could be found
Figure: 3.28 USB Port Interface

PIN 1: TXD
PIN 2: DTR#
PIN 3: RTS#
PIN 4: VCCIO
PIN 5: RXD
PIN 6: RI#
PINS 7, 18, 21: GND
PIN 8, 24: No Internal Connection
PIN 9: DSR#
PIN 10: DCD#
PIN 11: CTS#
PIN 12: CBUS4
PIN 13: CBUS2
PIN 14: CBUS3
Clear.Input: PIN74
PIN 13: CBUS2
PIN 15: USBDP
PIN 16: USBDM
PIN 17: 3V3OUT
PIN 19: RESET#
PIN 20: VCC
PIN 22: CBUS1
PIN 23: CBUS0
PIN 25: AGND
PIN 26: TEST
PIN 27: OSC1
PIN 28: OSCO
Async.Output: PIN63
Async.Input: PIN62
Request.Output: PIN75
3.11. POWER SUPPLY

For our design, we elect to employ a 25.9 V Lithium Polymer Ion battery since it has a very high energy efficiency and capacity; and it has a much longer recharge cycle lifespan than many other rechargeable batteries. It is also much more environmentally friendly and it is much less heavy than a traditional lead-acid alternative. We require a power supply that can accommodate the power needs of every component and part in our design. Most power will be drawn from our motors but there are several other components that must be considered. The following table lists the parts of the robot, which will have to be supplied with electrical power.

Table: 3.6 Power Consumption

<table>
<thead>
<tr>
<th>Device</th>
<th>Voltage (V)</th>
<th>Current (mA)</th>
<th>Power (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motor driver</td>
<td>25.9</td>
<td>60</td>
<td>1.554</td>
</tr>
<tr>
<td>Motor driver</td>
<td>25.9</td>
<td>60</td>
<td>1.554</td>
</tr>
<tr>
<td>Motor driver</td>
<td>25.9</td>
<td>60</td>
<td>1.554</td>
</tr>
<tr>
<td>Motor driver</td>
<td>25.9</td>
<td>60</td>
<td>1.554</td>
</tr>
<tr>
<td>Solenoid Driver</td>
<td>25.9</td>
<td>86.5</td>
<td>2.24035</td>
</tr>
<tr>
<td>Proximity Sensor</td>
<td>25.9</td>
<td>8</td>
<td>0.2072</td>
</tr>
<tr>
<td>Reed Switch</td>
<td>5</td>
<td>50</td>
<td>0.25</td>
</tr>
<tr>
<td>Motor</td>
<td>25.9</td>
<td>3500</td>
<td>90.65</td>
</tr>
<tr>
<td>Motor</td>
<td>25.9</td>
<td>3500</td>
<td>90.65</td>
</tr>
<tr>
<td>Motor</td>
<td>25.9</td>
<td>3500</td>
<td>90.65</td>
</tr>
<tr>
<td>Motor</td>
<td>25.9</td>
<td>3500</td>
<td>90.65</td>
</tr>
<tr>
<td>Microcontroller</td>
<td>5</td>
<td>300</td>
<td>1.5</td>
</tr>
<tr>
<td>Solenoid valve</td>
<td>25.9</td>
<td>0.1544</td>
<td>4</td>
</tr>
<tr>
<td>Hall Effect Sensor</td>
<td>5</td>
<td>4.5</td>
<td>0.0225</td>
</tr>
<tr>
<td>Hall Effect Sensor</td>
<td>5</td>
<td>4.5</td>
<td>0.0225</td>
</tr>
<tr>
<td>Hall Effect Sensor</td>
<td>5</td>
<td>4.5</td>
<td>0.0225</td>
</tr>
<tr>
<td>Hall Effect Sensor</td>
<td>5</td>
<td>4.5</td>
<td>0.0225</td>
</tr>
<tr>
<td><strong>Total Power (W)</strong></td>
<td></td>
<td></td>
<td><strong>377.10</strong></td>
</tr>
<tr>
<td><strong>Total Current (A)</strong></td>
<td></td>
<td></td>
<td><strong>14.70</strong></td>
</tr>
</tbody>
</table>

The total Power and Current above is calculated given one (1) hour of continuous component operation. We will design such that the robot will sustain power for a total of 45 minutes. This time was determined by the design team as being sufficient since the length of a match game play is 30 minutes.

Therefore:

Total Power Capacity required, \( P = 377.10 \times 0.75 = 282.825 \text{ Wh} \)

Total Current Capacity required, \( I = 14.70 \times 0.75 = 11.025 \text{ A} \)
The power supply we chose to satisfy these needs is a Polymer Li-Ion Battery Module. This module is a 25.9 V source and has a rating of 12.6Ah. The total watt-hour rating is 326.34 Wh and the maximum discharge rate is 18A rate. This battery module is made of seven (7) 3.7V cells connected in series and the entire module is wrapped in heavy duty heat shrink tubing. Within the battery is installed a PCM with equilibrium function to balance charging in each cell and to protect the module from overcharging above 29.6V and over-discharging below 16.1V. This power supply will serve all our power needs and at 1.585kg will help to minimize the weight of our robot.

![Figure: 3.29 Polymer Li-On single cell](image1.png)

![Figure: 3.30 Polymer Li-On Battery Module](image2.png)

The charger we will use to recharge this battery is the Smart Charger for the 25.9V Li-Ion/Polymer Rechargeable Battery Pack (Standard Female Tamiya Plug). This charger recharges with a current of 3A and a full charge will be completed after 6.3 hrs. A red LED will show the battery charging and the green LED will indicate when the charge is complete.

![Figure: 3.31 Polymer Li-On Battery Charger](image3.png)

3.12. **EMERGENCY STOP SWITCH**

An emergency stop switch will be mounted onto our robot to immediately terminate all actuation in the event of a robot malfunction. The device we will utilize is a Thermal Circuit Breaker Rocker Switch that will be capable of sustaining a maximum 20A of continuous current at 24VDC. This switch will be connected between the power supply and the DC-DC converter. This switch is designed to automatically trip and disconnect the power source from the circuit if a current exceeding 20A is present. In addition, the switch can also be manually actuated in the
event that a robot malfunction is seen by the design team or any operator. This device therefore serves both the purposes of electrical fuse and emergency stop switch. A generic drawing as well as the actual switch we intend to use to implement emergency stop functionality is shown below.

![Emergency Stop Switch Diagram](image1)

Figure: 3.32 Emergency Stop Switch Diagram

![Circuit Breaker Rocker Switch](image2)

Figure: 3.33 Circuit Breaker Rocker Switch

### 3.13. PRINTED CIRCUIT BOARD

The PCB was completely designed using Multisim and Ultiboard software. This software combination allows easy creation and modification of electrical circuit designs and provides design error and validation checks to help ensure the accuracy and integrity of the PCBs designed.
Figure: 3.34 Multisim Schematic

Figure: 3.35 Original PCB Layout
4. BUILDING PROCESS

4.1. ROBOTIC PLATFORM

The building process of the robot began with building the platform. After the material was ordered from The Metal Supermarket, it was cut to print in the first floor of the ET building at IPFW. The bent sections were sent out to a local fabrication shop to be manufactured. Once the bent pieces where completed the building began by first erecting the three braces that mount to the base plate. Simple measurements were made to ensure centering and alignment. With the main beams up the side pieces were installed next. All of the holes put into the various components were made using either a hand drill or drill press and finally an enclosure was bolted to the inside of the platform to house the electrical components.

4.1.1. MODIFICATIONS

The main modification to the original design was the elimination of the angled front supports from the front of the robot, which were designed to keep the soccer ball from entering the robot. It was determined that the ball could not enter the robot due to the lack of clearance from the positioning of the side post and kicking assembly. The kicker support beam was originally drawn up as a single piece, however due to manufacturing concerns the beam was cut into three pieces and bolted together. Self-adhesive weather stripping was placed on the top side of the kicker support beam to cushion the enclosure for the electrical components. A square hole was placed in the top back side of the platform to house an emergency stop switch. L-Brackets were added to the platforms side supports to provide additional robustness, without the brackets the side posts were only attached with a single bolt.

4.1.2. FINAL ROBOTIC PLATFORM

The final robotic platform is shown in the figures below.

Figure: 4.1 Kicker Support
Figure: 4.2 Self-adhesive Weather Stripping Added to Support Enclosure

Figure: 4.3 Emergency Stop Switch

Figure: 4.4 Platform to Side Supports Reinforcements
4.2. KICKING MECHANISM

The hard stop for the kicking mechanism was made using the CNC machine in the IPFW engineering department. The cylinder was mounted parallel with the ground when attached to
the kick plate, which required us to manufacture an aluminum spacer. The cylinder was bolted
to the base plate and the spacer and a joint bracket was used to connect the cylinder to the kick
plate. Self-adhesive weather stripping was placed on the hard stop to dampen the sound of the
kick plate striking against the hard stop. The 2007-2008 RoboCup’s brace for the pressure tank
was utilized and was mounted to the platform and held the pressure tank. A steel braided gas
line was connected to the pressure tank and routed to a manual regulator. The manual regulator
had a pressure gage attached to it. A component called a duck bill was then attached to the
manual regulator to connect the ¼” tubing. The tubing was then routed to the solenoid and
separate ¼” lines were routed to the input and exhaust ports of the cylinder. Teflon tape was
used to seal all the connections in the pneumatic system. A door hinge was used to attach the
kick plate to the kicker support. The final positioning of the kicker assembly had to be moved
outward relative to the base plate approximately 3.5” to avoid interference with the front two
motors.

The solenoid driver chip is connected to the solenoid valve’s pins 1 and 2. Pin 3 does not need a
connection, and the polarity of pins 1 and 2 are not essential.

The final components in the kicking system are the sensors. These sensors include the reed
switches and the proximity sensor. The reed switches are mounted on the cylinder from the
manufacturer; however the proximity sensor required a bracket to be mounted. The proximity
sensor has a range of 15mm, so it is mounted 10mm above the ball.

4.2.1. MODIFICATIONS

The clevis on the backside of the cylinder was modified to reduce its size in order to make it fit
properly inside of the platform. The front clevis of the cylinder was ground down to avoid
hitting the kick plate when the cylinder was actuated.

The proximity sensor was added to the front of the platform. The original proximity sensor
circuit included an inverter, but the inverter was unnecessary since the microcontroller’s pins are
already driven high. The other modification is that the microcontroller chosen can accept only
3.3V max not 5V, so a voltage divider will be used to bring the voltage down to the correct level.

The connections to the microcontroller from the reed switches required modification. The
microcontroller’s input pins are internal driven high on the development board, so the reed
switches were connected to ground instead of the voltage source. Connecting the reed switches
to ground allow the microcontroller to detect when the switch is closed, it also allows the
switches to be used without consuming any power.

4.2.2. FINAL KICKING MECHANISM

The final kicking mechanism is shown in the figures below.
Figure: 4.7 Kick Plate and Hard Stop

Figure: 4.8 Joint Bracket and Self-adhesive Weather Stripping
Added to Hard Stop
Figure: 4.9 Cylinder with Clevises and Joint Bracket

Figure: 4.10 Solenoid, Manifold, and ¼” lines
Figure: 4.11 Nitrogen Tanks, Manual Regulator, Pressure Gage, and Exhausts

Figure: 4.12 Duckbill
Figure: 4.13 Pressure Tank Holder

Figure: 4.14 Aluminum Beam used for Ball Sensor
Figure: 4.15 Dimensional Drawing for Cylinder Spacer

Figure: 4.16 Joint Bracket
4.3. OMNI DIRECTIONAL MOVEMENT SYSTEM

The process started by modifying the hubs to incorporate a set screw that would apply pressure directly to the key in between the shaft and hub. This modification can be seen below in figure
4.20. One critical step in assembling the motors was to ensure that the bolts connecting the wheels to the hubs were already in the hub before they are fastened to the motor shaft. This was done to ensure enough clearance for installing and tightening these bolts. Once the hub was fastened to the motor shaft with the key and set screw the motors were ready to be mounted to the base platform of the robot.

This process began by first taking dimensions from the motor manufacturing specs on the bolt pattern of the motors and laying them out at the 90° angles on a print of the platform. The result of this layout can be seen below in figure 4.21. For this robot the holes where laid out by hand and were drilled oversized. The actual size needed for the bolts was 0.19 inch, but most holes were drilled oversized to 0.25 and washers were used on the bolts. This helped to overcome the human error and tolerances involved during the layout process. With these holes drilled the motors could then be attached to the bottom of the platform. Finally, the wheels could be attached on the hub using the bolts already installed in the hub prior to mounting the motors.

The motor drivers are installed on the electric shelf built in the robot. The drivers are connected to the SCI-C, the serial C port of the microcontroller.

The final component of the Omni directional system is the Hall Effect sensors. The Hall Effect sensors were mounted the underside of the base plate. The motors case provided a clamp to keep the sensors in position.

---

4.3.1. MODIFICATIONS

In the original design of the motor control the drivers were to utilize the simple serial mode of the driver. However, the microcontroller only has 3 asynchronous serial ports and the simple serial mode would require 4. The robot now uses the packetized serial mode. This packetized serial mode allows multiple drivers to be connected on a single serial line. To control the drivers in this mode each driver is configured to a different address. Another difference between the simplified serial mode and the packetized serial mode is that the power value ranges from 0 to 127 and require a direction byte, instead of 0 to 255 with an encoded direction. The following figure illustrates the DIP switch positions for each of the drivers.
In the packetized serial mode the following format must be used to send a packet of data to the driver: address byte, direction byte, motor power byte, check byte. The check byte consists of 
((address byte & direction byte & motor power byte) + 127).

While testing the Hall Effect sensor on the breadboard it was determined that the original configuration of the magnet mounted on the sensor would not work. The sensor was not sensitive enough to detect the targets on the wheels. Instead of installing larger targets and modifying the wheels, the magnets were mounted on the wheel with super glue instead of mounting the magnet on the Hall Effect sensor. The other modification done the reed switch circuit was the voltage output of the Hall Effect sensor is 5V, so again a voltage divider was employed to drop the voltage from 5V to about 3V.
4.3.2. FINAL OMNI DIRECTIONAL MOVEMENT SYSTEM

The figures below show the final omni directional movement system.
Figure: 4.22 Motor and Platform Completed Assembly

Figure: 4.23 Motor Placement
Figure: 4.24 Motor Drivers

*Note: PCB Not installed

Figure: 4.25 Hall Effect Sensor
4.4. CONTROL CIRCUIT

To command our robot, our entire electrical control unit includes the eZdsp F28335 development board, our fabricated printed circuit board and a 50VDC 20A circuit breaker / emergency stop switch.

The eZdsp F28335 development board comes with the Code Composer Studio DSK Tools and includes a ‘C’ compiler, Assembler, Linker and Debugger. The kit also includes Texas Instruments’ F28335 header files and example software, Texas Instruments’ Flash APIs to support the F28335 and the digital signal controller is compatible with Windows 2000 and XP. The F28335 development board is physically equipped with the following:

- TMS320F28335 Digital Signal Controller, in socket, operating at 150 MHz.
- On chip 32-bit floating point unit
- 68K bytes on-chip RAM
- 256K bytes off-chip SRAM memory
- 512K bytes on-chip Flash Memory
- On chip 12 bit Analog to Digital (A/D) converter with 16 input channels
- Multiple Expansion Connections (analog, I/O)
- On board RS-232 connector with line driver
- On board CAN 2.0 interface with line driver and connector
- Onboard embedded USB JTAG controller

This development kit allows us the flexibility, processing power and robustness needed to implement all the controls necessary for our robot.

4.4.1. MODIFICATIONS

The only modifications needed on the development board were to solder header pins to the P-8 header strip of the development board as well as a few locations on the P-10 strip. These headers were the point of interface between the development board and the PCB.

4.4.2. FINAL CONTROL CIRCUIT

Our final development board control assembly is shown in the figure below. The 176 pin digital signal controller chip can be seen in socket on the development board.
4.5. PRINTED CIRCUIT BOARD

The eZdsp F28335 development kit is directly interfaced with our fabricated printed circuit board. Our PCB primarily acts as a bridge or “go-between” unit that implements circuitry to connect a proximity sensor, two reed switches, a solenoid driver, and four hall-effect sensors to our digital signal controller development board where all the robot intelligence and processing of signals is done. In addition to these, our PCB connects a signal line from the F28335 development board and power lines to the motor controllers from the 25.9VDC lithium polymer battery. Our PCB implements a USB Full-speed port should there be a need to include any additional serial interfacing capability for the robot’s future functionality.

4.5.1. MODIFICATIONS

After implementation and testing, a few changes were made from the original PCB design.

Trace Width – One of the major issues encountered with the original PCB design was the need for a much larger trace width from the power supply to the motor driver connection terminal blocks to allow for currents of up to 10A from power source to motor controller. Even with some consultation, the original trace width at this point on the PCB was designed at only 40 mils. During testing, it was discovered that this width was very insufficient as we found that it would only accommodate approximately 1A maximum current. The redesigned trace was made with a width of 750 mils and could accommodate up to 10A assuming a temperature rise factor of 10 °C along the trace.

DRV104 input pull-down resistor – The original solenoid driver circuit implemented using a model from the previous year’s design team. After testing this circuit, we found that no matter the input, the driver was always caused to output high. After investigating the model used, we discovered that a pull-down resistor added to the final DRV104 circuit from the previous design group was omitted from our circuit. Our redesigned solenoid driver circuit includes a 330Ω
resistor at pin 14 of the DRV104 chip and this rectified the output behavior of the chip. The previous year’s team used a 1kΩ resistor at the input to pin 14 but we found that a 330Ω resistor was sufficient for our design.

**Holes** – Our first PCB built did not include mounting holes so that our board could stand in its hardware unit and also mount to the development board. We included these mounting holes to make this interface easier.

**Fan** – Our final PCB design also included a 2-pin terminal block for powering a 5V DC mini fan which we mounted inside the electrical hardware unit. This fan supplied cooling to all the electronic components within the hardware unit.

**Fly-Back Diode and 100uF Capacitor** – Since we could not find a fly-back diode or a 100uF capacitor with the footprints specified in the original PCB design, we changed these footprints to accommodate through hole parts.

### 4.5.2. FINAL PRINTED CIRCUIT BOARD

The final Printed Circuit Board design is shown below:

Figure: 4.27 Final PCB Design Layout
4.6. ELECTRICAL HARDWARE

The development board and PCB were mounted in an aluminum enclosure that mounted to the side of the robot. The dimensions of the enclosure are 11 X 8.5 X 3.5. The base of the aluminum enclosure gave sturdy support to the electrical hardware and also provided a heat sink for the motor controllers and DC-DC voltage converter that were used in the design.

4.6.1. MODIFICATIONS

**Fan** – A 5VDC fan was mounted to the wall of the enclosure to provide cooling of electrical components within the enclosure. The slot created for the fan was made using a Dremel tool.
Serial Port Access – A Dremel tool was used to create a slot in one side of the aluminum enclosure which provided an exhaust to the flow of hot air created by the fan as well as which gave easy access to the serial and CAN ports of the eZdsp development board.

The final electrical hardware unit with all interfacing connections is shown below.
The robot was programmed in C programming language. Many of the commonly used source and header files were provided by Texas Instruments. These files are support the features of the TMS320F28335 microcontroller such as: EPWM, SCI, SPI, ECAP, etc. The files developed by the team include: Main.c, LED.c, PIN_Setup.c, Serial.c, Movement.c, Sensors.c, and Kick.c. The design of the software is modular and each of files, except Main, defines functions. The other files needed for the project are: CodeStartBranch.asm, DelayUs.asm, Passwords.asm, SetDBGIER.asm, DSP2833x_GlobalVariableDefs.c, Flash.c, and SystemCtrl.c. The command files DSP2833x_Headers_nonBIOS.cmd and F28335_nonBIOS_flash.cmd are also required.

The LED source file is only used for testing; it maps output pins of the microcontroller to the LED’s and provides the functions LEDon(#) and LEDoff(#). The function LEDinit() is used to setup the GPIO pins as output pins for the LEDs.

The PIN_setup file sets up all the General Purpose Input Output pins used in the project. There is only one function in the PIN_setup source file PIN_setup().

The Serial file sets up the serial ports used in the project. The functions SciC_Init() and SciC_xmit(#) are used to setup the SCI-C serial port to transmit and then to transmit data.

The Movement file sets up all of the Omni directional movement commands used for moving the robot. There are three functions available for use in this file. The first is the Stop(#) command, which is used to stop the motor that is inputted into the command. The movement(#,#) command allows the user to specify the desired angle in which the robot will travel and the number of counts the wheel will travel. The final command is Rotate(#,#), which allows the user to choice clockwise rotation (1) or counterclockwise (2) and a speed (0-127).
The Sensors file sets up the sensors used by the robot. The functions included in this file are Reed(#), Hall(#) and Prox(). The Reed and Hall functions require the input of the number of the sensor the user wants to interface with. These sensor functions check the input pins of the associated function and returns a value of a 0 or a 1.

The Kick file has only one function, the Kick() function. The kick command checks the position of the cylinder by the reed switches and the checks if the proximity sensor senses an object. If the cylinder is retracted and the proximity senses an object the kick function sets the output kick pin high, and the cylinder will extend. After a kick has occurred the cylinder will automatically retract if the cylinder is fully extended.

The following steps are followed to program the robot in flash memory.

1. The first step in programming the robot is to open Code Composer studio.

2. Next the project must be opened by going to project, and then open the file Robocup_non_bios_flash.pjt.

Figure: 4.32 Open Project File
3. Once the project is loaded it is then compiled.

4. Once the release .out file has been built it must be loaded into the flash memory of the microprocessor.

5. First go to Debug and connect to the development board.

6. Next go to tools and selectF28xx On-Chip Flash programmer.
7. Ensure OSCCLK(Mhz) is set to 30.
8. Ensure the correct .out file is going to be loaded.

9. Finally select Execute Operation to erase, program and verify.

All of the source files are included in the appendix. The user defined header files are also included; all other header files can be downloaded from www.ti.com.

4.8. DIFFICULTIES ENCOUNTERED

There were many difficulties encountered in the design of this robot. The installation of the platform components into the base plate proved to be challenging due to the angled pieces not being bent to the print. Although they were close, some imperfections can be seen.
aluminum used was an extremely strong material, and the lack of tools available made it difficult to modify and drill into at times, when not working in the maintenance shop. It was also extremely difficult to fully design the robot, since there were so many sub systems. The size of some of the components was unknown ahead of time and not every component required could be accounted for during the design stage.

The biggest difficulty encountered in mounting the motors was the layout of the hole patterns on the base platform. Small human errors quickly add up to the point where some holes had to be reamed out more than .25 inch. Although this was fixed by the use of washers on the bolts it isn’t the ideal situation. In hind sight the ideal situation would have been to have the base platform cut in a CNC along with the hole layouts with very small tolerances. This would have helped to eliminate some of the human error, but this could have also driven up the cost.

One constant difficulty we encountered throughout the build process was being able to find and attain easy access to many of the tools needed for the project. We bought some tools for the project as time went on and others were brought in by the team members as needed. Often, some tools were utilized to perform functions for which they were not made. Having a dedicated tool box in the build room for the project would greatly alleviate some of these difficulties and help the team expedite the building process.

The software package provided its own unique challenges. One difficulty was missing header files. The header files must be in the same folder as the source files or there is an error. The other way around this error is to specify the directory where the header files are located in the build options. Another difficulty was learning the IDE Code Composer.
5. TESTING PROCESS

5.1. DESIGN PARAMETERS

System: Pneumatic System
- The pressure tank was 48 cubic inch in size.
- The tank was filled with nitrogen gas to a pressure of 3000 psi.
- The manual pressure regulator was set to 100 psi.
- The expected number of kicks from the tank was 221 kicks.
- The expected distance the ball would travel horizontally was at least 5 meters while in the air, bouncing, and rolling.
- The expected output of the Proximity sensor should be 3.3V.

System: Omni Directional Movement
- The omni wheels were 6” in diameter and made from plastic.
- The motors used could a minimum torque of 1.48 N-m and 355 rpm.
- The expected speed of the robot was to be at least 2m/s.
- The expected output of the Hall Effect sensor should be 3.3V.

System: Platform
- The entire platform was built using Aluminum 6061-T6.
- The bolts used were 8-32 x 1 inch and made from steel.
- The expected result was that the robot will be fully functional after all the impact testing was completed.

System: Power Supply
- The power supply is comprised of a 25.9V Lithium Polymer Battery.
- The voltage convertor converts the 25.9V to 5V.

5.2. ELECTRICAL SYSTEMS
Testing the Hall Effect circuit consists of three phases. The Hall Effect circuit must be tested to ensure proper wheel position, which enables the control of robot movements. The first phase was to test the Hall Effect circuit on a bread board. When the magnet is in front of the sensor the multi-meter read 3V and after the magnet is removed the meter read 0V. The second phase ensures that the combination of the wheel and sensor gives the proper output. After the Hall Effect sensor was mounted the wheel was rotated and again the voltage was checked after each magnet on the wheel passed the sensor. The final phase is testing the interface of the circuit to the microcontroller. The test was never successfully performed, when all of the sensors were connected only one had the correct output voltage.

The reed switches are used to let the microcontroller know the position of the cylinder. When the microcontroller has received the signal from the reed switch, the microcontroller illuminated a LED for both the extended and retracted positions.

The proximity sensor circuit test consisted of checking the output voltage of the sensor, sensor placement, and the interface between the sensor and the microcontroller. The first test the circuit was built on the breadboard and the voltage output was tested. The output with the ball in front of the sensor is 0V and after the ball is removed 3.3V was read on the multi-meter. The next test was performed after the sensor was mounted into the robot. The proximity sensor has a built in LED that indicates if the sensor senses an object. The proximity sensor was mounted 10mm above the ball on the ground. A visual inspection was performed to see if the LED illuminated when the ball was placed under the sensor. The final test was to see if the microcontroller received the signal from the proximity sensor. A LED was illuminated by the microcontroller after the proximity sensor senses the ball.

The first motor driver test was used to see if the driver can drive the motors in both forward and reverse. The motors can be tested by using a potentiometer connected to the V+ and 0 terminals and the tap on the potentiometer connected to S1 of the driver. The motor controller has to be in analog mode. The DIP switches must be in the following positions:

1 = on  
2 = on  
3 = either  
4 = on  
5 = on  
6 = on

This corresponds to an Analog Bi-direction mode. In this mode, a signal of 2.5V corresponds to stop. At 0V, the motor is full reverse and at 5V corresponds to full forward. The test began with the voltage at 2.5V the motor was stopped. The voltage was decreased until the voltage 0V and the motors gradually increased in speed in the reverse direction. The voltage was again set at 2.5V. The voltage was increased until 5V and the motor gradually increased in speed in the forward direction.

The second phase of testing was to see if the microcontroller could control the driver. When a value of 0 is sent the motor is stopped and a value of 127 is full speed. So a value of 0 was sent
the motor remained stopped, a value of 10 was sent and the motor moved slowly. The values were incremented up to 127, and the motor’s speed increased with each increase in value.

Since the solenoid driver circuit was developed by a previous RoboCup team the solenoid driver circuit will only be tested to see if the current components operate. The solenoid driver circuit was tested once the PCB was made. The first test was to see if a 3.3V signal could actuate the solenoid, which it did. The second test was to see if the microcontroller could actuate the solenoid and solenoid actuated after a kick command was sent by the microcontroller. The final test was the proximity sensor to solenoid driver interaction. The robot was set to perform a kick if an object was sensed by the proximity sensor. The ball was rolled toward the robot and it performed the kick.

The power supply was designed to supply full power to the robot for 45 minutes. The capacity of the power supply was tested by turning on all four motors and allowing them to run at full speed. Any degradation in performance of the motors was monitored. After allowing the motors to run for a total of three (3) hours, it was determined by visual inspection, that the performance of the motors was not at all affected after this length of time running. The results of this test are shown below.

<table>
<thead>
<tr>
<th>Time (hrs)</th>
<th>Motors running at full speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:00</td>
<td>Yes</td>
</tr>
<tr>
<td>0:15</td>
<td>Yes</td>
</tr>
<tr>
<td>0:30</td>
<td>Yes</td>
</tr>
<tr>
<td>0:45</td>
<td>Yes</td>
</tr>
<tr>
<td>1:00</td>
<td>Yes</td>
</tr>
<tr>
<td>1:15</td>
<td>Yes</td>
</tr>
<tr>
<td>1:30</td>
<td>Yes</td>
</tr>
<tr>
<td>1:45</td>
<td>Yes</td>
</tr>
<tr>
<td>2:00</td>
<td>Yes</td>
</tr>
<tr>
<td>2:15</td>
<td>Yes</td>
</tr>
<tr>
<td>2:30</td>
<td>Yes</td>
</tr>
<tr>
<td>2:45</td>
<td>Yes</td>
</tr>
<tr>
<td>3:00</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The above results show that the power supply fully powered the motors for more than three (3) times the time required. It should be noted that this test was carried out with the robot in place on a stool elevated off the ground. Consideration should be given to the fact that the weight of the robot on the motors was not allowed to factor into the performance test as it would be in a real game situation. Also, the robot was not made to perform kicks or undergo any other actuation that would contribute to power supply usage. However, given the outstanding performance of the power supply, we believe that even with these added factors, the power supply would be very capable of fully powering all systems of the robot for the required 45 minutes.
The DC-DC converter in our system is required to output a constant 5 VDC value for a DC voltage input of 25.9 V from the Li-On polymer battery. The output of the converter is rated to be 5 VDC for any input voltage between 18V-35V. This fits very well with the power supply chosen since it is rated to be at 29.6V at full charge and about 16.1V at minimum charge. Thus, for the vast majority of the power supply range, the output from the DC-DC converter is constantly at 5V. This helps to ensure stability in our system. The experimental DC-DC conversion range was verified as shown in the test below. The input to the converter was incremented from 16.5V to 35V and the output voltage at each step was recorded as shown.

Table: 5.2 DC to DC Conversion

<table>
<thead>
<tr>
<th>Voltage Input (VDC)</th>
<th>Voltage Output (VDC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16.5</td>
<td>5</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>25</td>
<td>5</td>
</tr>
<tr>
<td>30</td>
<td>5</td>
</tr>
<tr>
<td>35</td>
<td>5</td>
</tr>
</tbody>
</table>

The performance of the DC-DC converter was found to be very satisfactory and the device worked very well in our circuit design.

5.3. MECHANICAL SYSTEMS

System: Pneumatic

Objective: The purpose of this experiment was to determine the number of kicks that the pneumatic system could produce from a full nitrogen cylinder.

Procedure:
1. The first thing done was to place the robot on the ground at a stationary location.
2. The pneumatic system was then actuated. Throughout the course of the experiment the number of kicks was be tracked.
3. Step 2 was repeated 3 times.
4. The experimental value was compared with expected result.

Results:

The pneumatic system produced on average of 126 kicks per tank.

Table: 5.3 Number of Kicks

<table>
<thead>
<tr>
<th>Trial</th>
<th># of Kicks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>124</td>
</tr>
<tr>
<td>2</td>
<td>127</td>
</tr>
</tbody>
</table>
System: Pneumatic Kicker

Objective: The purpose of this experiment was to determine the total kicking distance of the pneumatic system.

Procedure:
1. The robot was placed in a stationary position.
2. Next, the soccer ball was placed in front of the robot, and the kicking mechanism was actuated.
3. The total kicking distance of air and rolling was measured and recorded.
4. Step 3 was repeated ten times.
5. The experimental values were then compared with the expected results.

Results:

The testing revealed that the robot was capable of kicking the ball well beyond the minimum total distance of 5 m. While not a specific requirement the pneumatic kicker was able to kick the ball in the air horizontally a distance of approximately 3 m.
System: Omni Directional Movement

Objective: The purpose of this experiment was to determine the speed of the robot.

Procedure:
1. First, a starting position was marked on a two meter testing area.
2. The robot was placed with its front just behind the first line.
3. The robot then accelerated for one second as programmed and the distance was measured.
3.b. The robot then accelerated for 8 seconds as programmed and the distance was measured.
4. With this information the speed was found and compared with the expected values
5. This process was repeated for several angles.

Results:
As shown in the tables the robot did achieve the 2m/s goal set by the problem statement for most angles. The procedure used in 3.b. was used for angles that proved to be better controlled. The longer runs give a better average velocity reading as shown below the average velocities on the longer runs were up to nearly 3m/s.
### Table: 5.5 Velocity Table

#### Trial (0 degree direction)
<table>
<thead>
<tr>
<th>Trial</th>
<th>Distance in 1 second (in)</th>
<th>Speed (m/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>86.50</td>
<td>2.20</td>
</tr>
<tr>
<td>2.00</td>
<td>83.50</td>
<td>2.12</td>
</tr>
<tr>
<td>3.00</td>
<td>83.75</td>
<td>2.13</td>
</tr>
<tr>
<td>4.00</td>
<td>84.00</td>
<td>2.13</td>
</tr>
<tr>
<td>5.00</td>
<td>83.75</td>
<td>2.13</td>
</tr>
<tr>
<td>6.00</td>
<td>85.50</td>
<td>2.17</td>
</tr>
<tr>
<td>7.00</td>
<td>85.50</td>
<td>2.17</td>
</tr>
<tr>
<td>8.00</td>
<td>85.00</td>
<td>2.16</td>
</tr>
<tr>
<td>9.00</td>
<td>87.00</td>
<td>2.21</td>
</tr>
<tr>
<td>10.00</td>
<td>85.75</td>
<td>2.18</td>
</tr>
<tr>
<td>Average</td>
<td>85.03</td>
<td>2.16</td>
</tr>
</tbody>
</table>

#### Trial (180 degree direction)
<table>
<thead>
<tr>
<th>Trial</th>
<th>Distance in 1 second (in)</th>
<th>Speed (m/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>82.00</td>
<td>2.08</td>
</tr>
<tr>
<td>2.00</td>
<td>77.00</td>
<td>1.96</td>
</tr>
<tr>
<td>3.00</td>
<td>77.00</td>
<td>1.96</td>
</tr>
<tr>
<td>4.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>78.67</td>
<td>2.00</td>
</tr>
</tbody>
</table>

#### Trial (45 degree direction)
<table>
<thead>
<tr>
<th>Trial</th>
<th>Distance in 1 second (in)</th>
<th>Speed (m/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>82.00</td>
<td>2.08</td>
</tr>
<tr>
<td>2.00</td>
<td>82.50</td>
<td>2.10</td>
</tr>
<tr>
<td>3.00</td>
<td>83.00</td>
<td>2.11</td>
</tr>
<tr>
<td>4.00</td>
<td>81.25</td>
<td>2.06</td>
</tr>
<tr>
<td>5.00</td>
<td>79.50</td>
<td>2.02</td>
</tr>
<tr>
<td>6.00</td>
<td>78.75</td>
<td>2.00</td>
</tr>
<tr>
<td>7.00</td>
<td>83.00</td>
<td>2.11</td>
</tr>
<tr>
<td>8.00</td>
<td>81.50</td>
<td>2.07</td>
</tr>
<tr>
<td>9.00</td>
<td>80.75</td>
<td>2.05</td>
</tr>
<tr>
<td>10.00</td>
<td>78.50</td>
<td>1.99</td>
</tr>
<tr>
<td>Average</td>
<td>81.08</td>
<td>2.06</td>
</tr>
</tbody>
</table>

#### Trial (215 degree direction)
<table>
<thead>
<tr>
<th>Trial</th>
<th>Distance in 1 second (in)</th>
<th>Speed (m/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>85.50</td>
<td>2.17</td>
</tr>
<tr>
<td>2.00</td>
<td>88.00</td>
<td>2.24</td>
</tr>
<tr>
<td>3.00</td>
<td>85.00</td>
<td>2.16</td>
</tr>
<tr>
<td>4.00</td>
<td>86.25</td>
<td>2.19</td>
</tr>
<tr>
<td>5.00</td>
<td>82.25</td>
<td>2.09</td>
</tr>
<tr>
<td>Trial (215 degree direction)</td>
<td>Distance in 1 second (in)</td>
<td>Speed (m/s)</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>1.00</td>
<td>70.00</td>
<td>1.78</td>
</tr>
<tr>
<td>2.00</td>
<td>72.00</td>
<td>1.83</td>
</tr>
<tr>
<td>3.00</td>
<td>76.00</td>
<td>1.93</td>
</tr>
<tr>
<td>4.00</td>
<td>66.50</td>
<td>1.69</td>
</tr>
<tr>
<td>5.00</td>
<td>71.00</td>
<td>1.80</td>
</tr>
<tr>
<td>6.00</td>
<td>74.50</td>
<td>1.89</td>
</tr>
<tr>
<td>7.00</td>
<td>74.50</td>
<td>1.89</td>
</tr>
<tr>
<td>8.00</td>
<td>69.00</td>
<td>1.75</td>
</tr>
<tr>
<td>9.00</td>
<td>74.00</td>
<td>1.88</td>
</tr>
<tr>
<td>10.00</td>
<td>73.00</td>
<td>1.85</td>
</tr>
<tr>
<td>Average</td>
<td>72.05</td>
<td>1.83</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trial (135 degree direction)</th>
<th>Distance in 1 second (in)</th>
<th>Speed (m/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>68.00</td>
<td>1.73</td>
</tr>
<tr>
<td>2.00</td>
<td>69.50</td>
<td>1.77</td>
</tr>
<tr>
<td>3.00</td>
<td>75.00</td>
<td>1.91</td>
</tr>
<tr>
<td>4.00</td>
<td>59.00</td>
<td>1.50</td>
</tr>
<tr>
<td>5.00</td>
<td>71.50</td>
<td>1.82</td>
</tr>
<tr>
<td>6.00</td>
<td>76.00</td>
<td>1.93</td>
</tr>
<tr>
<td>7.00</td>
<td>75.00</td>
<td>1.91</td>
</tr>
<tr>
<td>8.00</td>
<td>65.50</td>
<td>1.66</td>
</tr>
<tr>
<td>9.00</td>
<td>77.00</td>
<td>1.96</td>
</tr>
<tr>
<td>10.00</td>
<td>75.50</td>
<td>1.92</td>
</tr>
<tr>
<td>Average</td>
<td>71.20</td>
<td>1.81</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trial (90 Degree)</th>
<th>Distance in 1 second (in)</th>
<th>Speed (m/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>90.00</td>
<td>2.29</td>
</tr>
<tr>
<td>2.00</td>
<td>86.00</td>
<td>2.18</td>
</tr>
<tr>
<td>3.00</td>
<td>87.00</td>
<td>2.21</td>
</tr>
<tr>
<td>4.00</td>
<td>90.00</td>
<td>2.29</td>
</tr>
<tr>
<td>5.00</td>
<td>88.00</td>
<td>2.24</td>
</tr>
<tr>
<td>6.00</td>
<td>88.50</td>
<td>2.25</td>
</tr>
<tr>
<td>7.00</td>
<td>88.50</td>
<td>2.25</td>
</tr>
<tr>
<td>8.00</td>
<td>90.00</td>
<td>2.29</td>
</tr>
<tr>
<td>9.00</td>
<td>89.00</td>
<td>2.26</td>
</tr>
<tr>
<td>10.00</td>
<td>86.00</td>
<td>2.18</td>
</tr>
<tr>
<td>Average</td>
<td>88.30</td>
<td>2.24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trial (270 Degree)</th>
<th>Distance in 1 second (in)</th>
<th>Speed (m/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>73.50</td>
<td>1.87</td>
</tr>
</tbody>
</table>
System: Platform

Objective: The purpose of the experiment was to determine the durability characteristics of the robot. This experiment was pursued after the robot had been through all other testing.

Procedure:
1. The robot was stationary to begin the experiment.
2. The robot was then subjected to several soccer ball impacts.
3. After each impact the functionality of the robot consisting of mobility and kicking was tested and recorded.
4. Step 3 was repeated 10 times.

Results:
The ball was not able to pass through any opening in the platform, which meant that the ball could cause no harm to any of the components inside the robot. The robot was fully functional after all of the ball impacts.
6. EVALUATION AND RECOMMENDATIONS

6.1. EVALUATION

System: Platform

The final dimensions of the robot were 45.7 x 45.7 x 62.23 cm. The size constraints were 50 x 50 x 80 cm. The result of the height being so low was that a vision system could easily be added to the top of the robot without any need for concern involving the vertical size constraint. Due to an unidentified discrepancy in the positioning of the motors the width of the robot exceeded the allowable maximum width dimension. The most logical explanation for this happening was in the initial design. The original dimensional drawings were created using Solidworks. Unfortunately, access to this program was lost and the files created with that program had to be converted to AutoCAD files. The conversion process for whatever reason can and did alter some dimensions. The group attempted to recalculate any discrepancy in the dimensions by hand based upon the geometry observed. Unfortunately, the final result was that the robot did not meet the size specifications. This only occurs when the kicker is fully extended and resulted in the robot being 0.8 cm too large. This is issue is addressed in the recommendations section. The final weight of the robot was 31.1 kg. The weight constraint was 40 kg and again the visions system could be added without much concern for the weight constraint. During the design for the robustness of the robot, the forces created by the cylinder were one of the main concerns. The cylinder actuating caused no failures to the platform. Another consideration for the robustness was to ensure that the soccer ball could not enter into the platform at all. The testing revealed that the soccer ball did not enter into the robot. The other confirmation of robustness was during the speed and omni movement testing. The robot experienced many collisions with a cement wall at full speed. The wheels, motors, and platform all survived without any damage. The rules of competition specify that all robots should slow down before any collision can occur. The group therefore determined based on testing, operation, and observation that the robot was robust. The electrical enclosure on the other hand was an issue and will be addressed in the recommendations section.

System: Pneumatic System

The pressure system did not supply the number of kicks that was expected. The group expected to obtain approximately 221 kicks per tank and observed an average of 126. This resulted in an error of approximately 43%. One potential reason for the error could be that the manual regulator was difficult to keep exactly at 100 psi. Another issue was that the gage on the pressure regular was only accurate within approximately +/-10% psi. The assumption of using 12” (12.6*10^-6 m) of tubing to represent the volume of the tubing used and the solenoid/manifold was another source of error. The exact volume of the solenoid/manifold could not be obtained from the manufacturer and was underestimated. Despite this significant error the result of the
testing was that the robot could kick once approximately every 7.1 seconds per half. This is far
more than enough to be competitive based on the videos of competition observed by the group.

The minimum kicking distance was achieved without question. Every single kick performed
achieved 5m with the distance traveled in the air along with the bouncing and rolling. The total
distance in the air had been designed for and was tested once the requirement had been met. The
result was that the ball was kicked on average a horizontal distance of 2.9 m in the air. The
group expected the ball to be kicked 4 m in the air. This resulted in an error of approximately
25%. Possible sources of error were that the kicking angle of 45 degrees was not being met and
would result in the ball not traveling in the air as far as expected. The assumption of the change
in pressure across the cylinder being 15 psi could have been inaccurate. The other issue was that
the solenoid selected required that the cylinder actuate in .04 s. The assumption of the 50%
energy loss in the design process had to account for any discrepancies in this actuating time. The
actuation was so quick that the available testing equipment could not measure the time accurately
enough. It is therefore reasonable to assume that this may have impacted the results.

System: Omni Directional Movement

On the angles where the movement was controllable the robot did achieve the goal of 2m/s. The
angles that proved to be troublesome were the angles that caused the robot to not travel in a
straight line. Our method for testing measured a straight line distance. If the robot could not
travel in a straight line the distance recorded was actually shorter than the distance actually
travelled. This worked against us in the velocity calculation because the velocity only changed
based on the changing distances. In all reality the robot may have been able to travel at the
required speed, but without the control or a radar gun the group was not able to accurately
measure those speeds. Another issue with the testing was that the omni wheels spun
significantly at the start of each run. The robot also did not stop in a final location when the
program ended because the omni wheels allowed the robot to continue moving forward. It was
decided that the spin out and the slide would have to be assumed to cancel each other out. In an
effort to minimize this assumption the robot was also programmed to travel for 8 s. The distance
traveled was mesure and the average velocity was obtained. The results of this testing were that
the robot traveled as fast as 2.9 m/s. The design was proven to be accurate and any discrepancies
arose from the uncontrollable movement. However, this uncontrollable movement will be fixed
by future design teams with the integration of a vision and real time operating system.

System: Electrical Control Circuit

The eZdsp development board was very useful in enabling us to accomplish the goals of the
electrical robot control. The final PCB design largely worked as it was designed to. It served
multiple functionalities and although various modifications were made from the original layout,
utilizing the PCB as the interfacing unit between the development board and the robotic
components was a very good approach to our design. Other components chosen such as the
emergency stop switch, the DC-DC converter, the aluminum enclosure and the power supply
fully met the goals for these components in the design.
The first recommendation for this project would be that more students should have been provided to this task. The full in depth analysis that would have provided much more accurate results was not possible due to the work load. Building a robot from scratch that had multiple systems to be designed and analyzed required far more time than was available. The result was that inadequate analysis was performed in order to keep the project moving forward to meet the deadline. From a mechanical perspective the project required CAD designers, CAD Design checkers, FEA analysts, ME’s with pneumatic system experience, ME’s with DC motor design experience, Fabricators, an MET with machine shop experience, and ME’s to work on concept and design of all systems. The building process would have benefited from the ability to weld the platform together. This would have made the building process much easier by avoiding all the drilling and tapping that was required for bolts. This would have also eliminated the gap issues that were encountered when the platform was assembled.

Future groups would benefit from utilizing a different kicking system. The pneumatic system can provide an adequate number of kicks, but the kicks are not competitive with what was observed on the internet videos of competition. The issue with the kicker extended outside of the allowable size constraint could easily be addressed by placing an angled shim on to the hard stop to limit the distance that the kicker could swing. The kicking distance would be decreased somewhat, but would be very minimal and the minimum kicking distance could still be easily achieved.

The omni directional system will greatly benefit from the future vision system and real time operating system. These features will allow for the robots movement to be much more accurate by correcting for varying speeds of each motor. Basically, any errors from perfectly straight at any angle would be corrected. This would greatly assist in the testing of the speed of the robot in all directions. The wheels used on the robot should also be looked at in finding new omni wheels with a higher coefficient of friction. Another possibility would be to use two sets of the current wheels per motor. If these modifications are able to eliminate some of the slippage during the start of movement the control will benefit greatly. The housing for the electrical components needs to have more analysis. During collisions the electrical components encountered some failures. It would be recommended that some sort of shock absorbing components be added to the electrical enclosure. The electrical enclosure should also be considered for heat transfer analysis to ensure the longevity of the components inside.

Before the robot is actually placed in competition a dribbling mechanism should also be considered into its design. History keeping is another key recommendation. Keep detailed formalized track of all design work. It would almost be a good idea to keep a daily log of all design, activity, and build tasks. A lack of available tools was a bit of an issue. The more parts that can be fabricated at a machine shop the better. If students will continue to do the majority of the fabrication they should at least be provided with the following:

- Hand drill
- Set of drill bits
- Band saw
- Drill Press
- Hammers
- Screw driver set
- Allen Wrenches
- Files
- Clamps
- Hack Saw
- Socket wrenches (both Metric and SAE)
- Wire cutters
- Wire crimpers
- Snap wring pliers
- Soldering Iron
- Heat Gun

For the software to run effectively the system must be operate in real time. The microprocessor supports a real time operating system called BIOS. The software should be further developed to include the RTOS. The movement functions could be further developed to include the Hall Effect sensors to give position feedback and stop the motors.

As for the sensors included in the robot, again we are recommending changing the proximity sensor. The sensor used does not have enough sensitivity to the ball. A touch sensor could be used as effectively as the proximity sensor. The reed switches should have a similar circuit as the proximity sensor. This would allow the reed switches to work properly by applying power to the switches.

Having plastic screws and standoffs would greatly reduce the risk of unexpected power surges between the two main circuit boards and greatly help to prevent unwanted potential difference between the circuit boards in the enclosure and the frame of the robot.
Fully isolating the circuits in the aluminum enclosure by placing them in a plastic enclosure would further prevent the possibility of unwanted potential difference between the circuit boards and the robotic frame and this also adds greater safety for any operator since all live power will be safely concealed and insulated.

During our testing there were a number of power surges and short circuits which affected some our electronic components. We assume that one of these surges actually damaged the development board at one time during the testing process. If we were to place a fuse in line with the PCB (which provides the development board’s power) we may have avoided causing damage to the development board. Implementing a fuse may help to protect the development board and prevent any other damage in the future.
7. CONCLUSION

The main purpose of this project was to create a suitable platform for a robotic soccer player to compete in the Middle Size League that was able to move Omni directionally and replace the Pioneer robot. The robust design of the robot ensures that collisions with other robots will not cause critical damage to the robot. To be a soccer playing robot the main goals were to ensure the robot could kick a soccer ball effectively, move at a competitive velocity, and be able to operate in an autonomous mode. This version of the robot is capable of moving Omni-directionally at about 2 m/s by the way of four motors equipped with Omni wheels. It can kick a size 5 soccer ball 3m through the air before the ball touches the ground with the pneumatic kicker, which includes compressed nitrogen, solenoid and a pneumatic cylinder. The software written gives the robot the proper foundation to allow it to operate in an autonomous mode. The robot is controlled by a TMS320F28335 microcontroller. Once the real time operating system is enabled and the robot has incorporated a vision system with the integrated sensors, there is no reason it cannot be autonomous. The sensors included in the robot are reed switches to give cylinder position, proximity sensor locate a ball in front of the kicker and Hall Effect sensors to give position feedback of the wheels. A 25.9V/12.6Ah Lithium polymer battery supplies the power to run the robot for the duration of the soccer match.
8. REFERENCES


9. LIST OF FIGURES

Figure: 3.1 Robotic Platform.................................................................7
Figure: 3.2 Projectile Path ........................................................................8
Figure: 3.3 Stroke Length (note drawing not to scale) ............................9
Figure: 3.4 Load Factor Chart (CKD USA Corporation) .........................11
Figure: 3.5 Reference Velocity of Cylinder vs. Time ..............................11
Figure: 3.6 Kicker Free Body Diagram ................................................13
Figure: 3.7 Tank Pressure after Kick ....................................................16
Figure: 3.8 Allowable Working Pressure at Temperature (CKD USA Corporation).................................17
Figure: 3.9 Muffler Back Pressure (Adsens Technology Inc) ..................18
Figure: 3.10 Velocity Components of Wheel ........................................18
Figure: 3.11 Free Body Diagram of Robot ............................................19
Figure: 3.12 Free Body Diagram of One Wheel .....................................20
Figure: 3.13 Torque vs. Shaft Speed Diagram for the Motor ....................21
Figure: 3.14 Hub Modifications .............................................................21
Figure: 3.15 Purchased Omni Wheel ....................................................22
Figure: 3.16 Motor Orientation and Direction .......................................22
Figure: 3.17 Direction Zones ...............................................................22
Figure: 3.18 Zone 1 Vector Map ............................................................23
Figure: 3.19 Critical Angle Example .....................................................24
Figure: 3.20 SyRen10 Motor Controller Circuit .....................................26
Figure: 3.21 DRV104 Chip interface Circuit .........................................27
Figure: 3.22 Proximity Sensor Interface ..............................................28
Figure: 3.23 Sensor Placement .............................................................29
Figure: 3.24 Targets on the Wheel ........................................................29
10. LIST OF TABLES

Table: 3.1 Mass Estimation

Table: 3.2 Cylinder Selection

Table: 3.3 Tank Size and Number of Kicks for Air Tank

Table: 3.4 $\alpha$ Value

Table: 3.5 Motor Direction, Zone and Region

Table: 3.6 Power Consumption

Table: 5.1 Robot Operating Time

Table: 5.2 DC to DC Conversion

Table: 5.3 Number of Kicks

Table: 5.4 Kick Distance in Air

Table: 5.5 Velocity Table
WD_DISABLE .set 1 ; set to 1 to disable WD, else set to 0
.ref _c_int00
.def code_start

 Function: codestart section

 Description: Branch to code starting point

 .sect "codestart"
code_start:
.if WD_DISABLE == 1
   LB wd_disable ; Branch to watchdog disable code
.else
   LB _c_int00 ; Branch to start of boot.asm in RTS library
.endif

 ; end codestart section

 Function: wd_disable

 Description: Disables the watchdog timer

 .if WD_DISABLE == 1
   .text
   wd_disable:
   EALLOW ; Enable EALLOW protected register access
   MOVZ DP, #7029h>>6 ; Set data page for WDCR register
   MOV @7029h, #0068h ; Set WDDIS bit in WDCR to disable WD
   EDIS ; Disable EALLOW protected register access
   LB _c_int00 ; Branch to start of boot.asm in RTS library
.endif
;end wd_disable

*****************************************************************************

.end

; end of file CodeStartBranch.asm
void Delay(int number) {
    int i;
    number=number*100;
    for (i=0; i<=number; i++) {
        DelayUs(10000); //Delay in micro seconds
    }
}
* File: DelayUs.asm
* Devices: TMS320F2833x
* Author: David M. Alter, Texas Instruments Inc.
* History:
  * 12/18/07 - original (D. Alter)
**********************************************************************

WDKEY .set 0x7025

**********************************************************************

.* Function: DelayUs()
* Description: Implements a time delay
* DSP: TMS320F28335, TMS320F28334, TMS320F28332
* Include files: none
* Function Prototype: void DelayUs(unsigned int)
* Useage: DelayUs(Usec);
* Input Parameters: unsigned int Usec = time delay in microseconds
* Return Value: none
* Notes:
* 1) The execution time of this routine is based upon a 150 MHz
   CPUCLK. It also assumes that the function executes out of
   internal RAM. If executing out of internal flash or external
   memory, the execution speed will be slightly slower.
   However, the inner loop of this function is essentially
   invariant to the memory it is running in. Therefore, even
   when running in flash memory, the basic loop time will be
   only slightly longer than 1 us.
* 2) The outer loop of this function is interruptible (i.e., every
   1 us). The user should disable interrupts before calling the
   function if they need an exact delay time, as opposed to a
   minimum delay time.
* *
**********************************************************************

.def _DelayUs
.text

_DELAYUs:
  MOVB AH, #0 ;Zero AH
  PUSH ST1 ;Save ST1 to preserve EALLOW setting

DelayUs1: ;Outer loop

;Service the watchdog in case it is active
EALLOW
MOVZ DP, #(WDKEY>>6)
MOV @WDKEY, #0x0055
MOV @WDKEY, #0x00AA
EDIS

;Proceed with the inner loop
RPT #138 ;Inner loop
|| NOP

    SUBB ACC,#1 ;Decrement outer loop counter
    BF DelayUs1, GT ;Branch for outer loop

;Finish up
    POP ST1 ;Restore ST1
    LRETR ;Return

;end of function DelayUs()

**********************************************************************

.end
;end of file DelayUs.asm
#include "DSP2833x_Device.h" // DSP2833x Headerfile Include File

//---------------------------------------------------------------------------
// Define Global Peripheral Variables:
//
//---------------------------------------------------------------
#elif __cplusplus
#pragma DATA_SECTION("AdcRegsFile")
#else
#pragma DATA_SECTION(AdcRegs,"AdcRegsFile");
#endif
volatile struct ADC_REGS AdcRegs;

//---------------------------------------------------------------
#elif __cplusplus
#pragma DATA_SECTION("AdcMirrorFile")
#else
#pragma DATA_SECTION(AdcMirror,"AdcMirrorFile");
#endif
volatile struct ADC_RESULT_MIRROR_REGS AdcMirror;

//---------------------------------------------------------------
#elif __cplusplus
#pragma DATA_SECTION("CpuTimer0RegsFile")
#else
#pragma DATA_SECTION(CpuTimer0Regs,"CpuTimer0RegsFile");
#endif
volatile struct CPUTIMER_REGS CpuTimer0Regs;

//---------------------------------------------------------------
#elif __cplusplus
#pragma DATA_SECTION("CpuTimer1RegsFile")
#else
#pragma DATA_SECTION(CpuTimer1Regs,"CpuTimer1RegsFile");
#endif
volatile struct CPUTIMER_REGS CpuTimer1Regs;
#else
#pragma DATA_SECTION(CpuTimer1Regs,"CpuTimer1RegsFile");
#endif
volatile struct CPUTIMER_REGS CpuTimer1Regs;

//-------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("CpuTimer2RegsFile")
#else
#pragma DATA_SECTION(CpuTimer2Regs,"CpuTimer2RegsFile");
#endif
volatile struct CPUTIMER_REGS CpuTimer2Regs;

//-------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("CsmPwlFile")
#else
#pragma DATA_SECTION(CsmPwL,"CsmPwlFile");
#endif
volatile struct CSM_PWL CsmPwl;

//-------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("CsmRegsFile")
#else
#pragma DATA_SECTION(CsmRegs,"CsmRegsFile");
#endif
volatile struct CSM_REGS CsmRegs;

//-------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("DevEmuRegsFile")
#else
#pragma DATA_SECTION(DevEmuRegs,"DevEmuRegsFile");
#endif
volatile struct DEV_EMU_REGS DevEmuRegs;

//-------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("DmaRegsFile")
#else
#pragma DATA_SECTION(DmaRegs,"DmaRegsFile");
volatile struct DMA_REGS DmaRegs;

#ifdef __cplusplus
#pragma DATA_SECTION("ECanaRegsFile")
#else
#pragma DATA_SECTION(ECanaRegs,"ECanaRegsFile");
#endif
volatile struct ECAN_REGS ECanaRegs;

#ifdef __cplusplus
#pragma DATA_SECTION("ECanaMboxesFile")
#else
#pragma DATA_SECTION(ECanaMboxes,"ECanaMboxesFile");
#endif
volatile struct ECAN_MBOXES ECanaMboxes;

#ifdef __cplusplus
#pragma DATA_SECTION("ECanaLAMRegsFile")
#else
#pragma DATA_SECTION(ECanaLAMRegs,"ECanaLAMRegsFile");
#endif
volatile struct LAM_REGS ECanaLAMRegs;

#ifdef __cplusplus
#pragma DATA_SECTION("ECanaMOTSRegsFile")
#else
#pragma DATA_SECTION(ECanaMOTSRegs,"ECanaMOTSRegsFile");
#endif
volatile struct MOTS_REGS ECanaMOTSRegs;

#ifdef __cplusplus
#pragma DATA_SECTION("ECanaMOTORegsFile")
#else
#pragma DATA_SECTION(ECanaMOTORegs,"ECanaMOTORegsFile");
#endif
volatile struct MOTO_REGS ECanaMOTORegs;
//-------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECanbRegsFile")
#else
#pragma DATA_SECTION(ECanbRegs,"ECanbRegsFile");
#endif
volatile struct ECAN_REGS ECanbRegs;

//-------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECanbMboxesFile")
#else
#pragma DATA_SECTION(ECanbMboxes,"ECanbMboxesFile");
#endif
volatile struct ECAN_MBOXES ECanbMboxes;

//-------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECanbLAMRegsFile")
#else
#pragma DATA_SECTION(ECanbLAMRegs,"ECanbLAMRegsFile");
#endif
volatile struct LAM_REGS ECanbLAMRegs;

//-------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECanbMOTSRegsFile")
#else
#pragma DATA_SECTION(ECanbMOTSRegs,"ECanbMOTSRegsFile");
#endif
volatile struct MOTS_REGS ECanbMOTSRegs;

//-------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECanbMOTORegsFile")
#else
#pragma DATA_SECTION(ECanbMOTORegs,"ECanbMOTORegsFile");
#endif
volatile struct MOTO_REGS ECanbMOTORegs;

//-------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("EPwm1RegsFile")
#else
#pragma DATA_SECTION(EPwm1Regs,"EPwm1RegsFile");
#endif
```c
#include <stdio.h>

#define fputc cputc

int main(void)
{
    volatile struct EPWM_REGS EPwm1Regs;

    //-------------------------------
    #if defined __cplusplus
    #pragma DATA_SECTION("EPwm2RegsFile")
    #else
    #pragma DATA_SECTION(EPwm2Regs,"EPwm2RegsFile");
    #endif
    volatile struct EPWM_REGS EPwm2Regs;

    //-------------------------------
    #if defined __cplusplus
    #pragma DATA_SECTION("EPwm3RegsFile")
    #else
    #pragma DATA_SECTION(EPwm3Regs,"EPwm3RegsFile");
    #endif
    volatile struct EPWM_REGS EPwm3Regs;

    //-------------------------------
    #if defined __cplusplus
    #pragma DATA_SECTION("EPwm4RegsFile")
    #else
    #pragma DATA_SECTION(EPwm4Regs,"EPwm4RegsFile");
    #endif
    volatile struct EPWM_REGS EPwm4Regs;

    //-------------------------------
    #if defined __cplusplus
    #pragma DATA_SECTION("EPwm5RegsFile")
    #else
    #pragma DATA_SECTION(EPwm5Regs,"EPwm5RegsFile");
    #endif
    volatile struct EPWM_REGS EPwm5Regs;

    //-------------------------------
    #if defined __cplusplus
    #pragma DATA_SECTION("EPwm6RegsFile")
    #else
    #pragma DATA_SECTION(EPwm6Regs,"EPwm6RegsFile");
    #endif
    volatile struct EPWM_REGS EPwm6Regs;

    //-------------------------------
    return 0;
}
```
#ifdef __cplusplus
#pragma DATA_SECTION("ECap1RegsFile")
#else
#pragma DATA_SECTION(ECap1Regs,"ECap1RegsFile");
#endif
volatile struct ECAP_REGS ECap1Regs;

//--------------------------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECap2RegsFile")
#else
#pragma DATA_SECTION(ECap2Regs,"ECap2RegsFile");
#endif
volatile struct ECAP_REGS ECap2Regs;

//--------------------------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECap3RegsFile")
#else
#pragma DATA_SECTION(ECap3Regs,"ECap3RegsFile");
#endif
volatile struct ECAP_REGS ECap3Regs;

//--------------------------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECap4RegsFile")
#else
#pragma DATA_SECTION(ECap4Regs,"ECap4RegsFile");
#endif
volatile struct ECAP_REGS ECap4Regs;

//--------------------------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECap5RegsFile")
#else
#pragma DATA_SECTION(ECap5Regs,"ECap5RegsFile");
#endif
volatile struct ECAP_REGS ECap5Regs;

//--------------------------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECap6RegsFile")
#else
#pragma DATA_SECTION(ECap6Regs,"ECap6RegsFile");
#endif
volatile struct ECAP_REGS ECap6Regs;

//-------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("EQep1RegsFile")
#else
#pragma DATA_SECTION(EQep1Regs,"EQep1RegsFile");
#endif
volatile struct EQEP_REGS EQep1Regs;

//-------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("EQep2RegsFile")
#else
#pragma DATA_SECTION(EQep2Regs,"EQep2RegsFile");
#endif
volatile struct EQEP_REGS EQep2Regs;

//-------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("GpioCtrlRegsFile")
#else
#pragma DATA_SECTION(GpioCtrlRegs,"GpioCtrlRegsFile");
#endif
volatile struct GPIO_CTRL_REGS GpioCtrlRegs;

//-------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("GpioDataRegsFile")
#else
#pragma DATA_SECTION(GpioDataRegs,"GpioDataRegsFile");
#endif
volatile struct GPIO_DATA_REGS GpioDataRegs;

//-------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("GpioIntRegsFile")
#else
#pragma DATA_SECTION(GpioIntRegs,"GpioIntRegsFile");
#endif
volatile struct GPIO_INT_REGS GpioIntRegs;

//-------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("I2caRegsFile")
#else
#pragma DATA_SECTION(I2caRegsFile)
#endif
#pragma DATA_SECTION(I2caRegs,"I2caRegsFile");
#endif
volatile struct I2C_REGS I2caRegs;

//------------------------------------------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("McbspaRegsFile")
#else
#pragma DATA_SECTION(McbspaRegs,"McbspaRegsFile");
#endif
volatile struct MCBSP_REGS McbspaRegs;

//------------------------------------------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("McbspbRegsFile")
#else
#pragma DATA_SECTION(McbspbRegs,"McbspbRegsFile");
#endif
volatile struct MCBSP_REGS McbspbRegs;

//------------------------------------------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("PartIdRegsFile")
#else
#pragma DATA_SECTION(PartIdRegs,"PartIdRegsFile");
#endif
volatile struct PARTID_REGS PartIdRegs;

//------------------------------------------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("PieCtrlRegsFile")
#else
#pragma DATA_SECTION(PieCtrlRegs,"PieCtrlRegsFile");
#endif
volatile struct PIE_CTRL_REGS PieCtrlRegs;

//------------------------------------------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("PieVectTableFile")
#else
#pragma DATASECTION(PieVectTable,"PieVectTableFile");
#endif
struct PIE_VECT_TABLE PieVectTable;

//------------------------------------------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("SciaRegsFile")
#else
#pragma DATA_SECTION(SciaRegs,"SciaRegsFile");
#endif
volatile struct SCI_REGS SciaRegs;

//-------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ScibRegsFile")
#else
#pragma DATA_SECTION(ScibRegs,"ScibRegsFile");
#endif
volatile struct SCI_REGS ScibRegs;

//-------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ScicRegsFile")
#else
#pragma DATA_SECTION(ScicRegs,"ScicRegsFile");
#endif
volatile struct SCI_REGS ScicRegs;

//-------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("SpiaRegsFile")
#else
#pragma DATA_SECTION(SpiaRegs,"SpiaRegsFile");
#endif
volatile struct SPI_REGS SpiaRegs;

//-------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("SysCtrlRegsFile")
#else
#pragma DATA_SECTION(SysCtrlRegs,"SysCtrlRegsFile");
#endif
volatile struct SYS_CTRL_REGS SysCtrlRegs;

//-------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("FlashRegsFile")
#else
#pragma DATA_SECTION(FlashRegs,"FlashRegsFile");
#endif
volatile struct FLASH_REGS FlashRegs;
#ifdef __cplusplus
#pragma DATA_SECTION("XIntruptRegsFile")
#else
#pragma DATA_SECTION(XIntruptRegs,"XIntruptRegsFile");
#endif
volatile struct XINTRUPT_REGS XIntruptRegs;

//-------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("XintfRegsFile")
#else
#pragma DATA_SECTION(XintfRegs,"XintfRegsFile");
#endif
volatile struct XINTF_REGS XintfRegs;

//=================================
=======
// End of file.
//=================================
=======
extern void InitFlash(void);

void InitFlash(void)
{
    asm("EALLOW");

    FlashRegs.FPWR.bit.PWR = 3; // Pump and bank set to active mode
    FlashRegs.FSTATUS.bit.V3STAT = 1; // Clear the 3VSTAT bit
    FlashRegs.FSTDBYWAIT.bit.STDBYWAIT = 0x01FF; // Sleep to standby transition cycles
    FlashRegs.FACTIVEWAIT.bit.ACTIVEWAIT = 0x01FF; // Standby to active transition cycles
    FlashRegs.FBANKWAIT.bit.RANDWAIT = 5; // Random access waitstates
    FlashRegs.FBANKWAIT.bit.PAGEWAIT = 5; // Paged access waitstates
    FlashRegs.FOTPWAIT.bit.OTPWAIT = 8; // OTP waitstates
    FlashRegs.FOPT.bit.ENPIPE = 1; // Enable the flash pipeline
}

#pragma CODE_SECTION(InitFlash, "secureRamFuncs")
asm(" EDIS"); // Disable
EALLOW protected register access

// Force a complete pipeline flush to ensure that the write to the last register
// configured occurs before returning. Safest thing is to wait 8 full cycles.

asm(" RPT #6 || NOP");

} // end of InitFlash()

//-- end of file ---------------------------------------------------------------
// FILE: Kick.c
//
// TITLE: Kick
//
// DESCRIPTION:
//
//   This program waits for the cylinder to be retracted and the proximity
//   sensor to be set before it actuates the cylinder.
//
//

#include "Project.h"   // Project Headerfile
void Kick(void)
{
    if (Reed(1)==1 && Prox()==1)
    {
        GpioDataRegs.GPASET.bit.GPIO18 = 1;  // Kick
    }
    else if (Reed(2)==1) && Prox()==0
    {
        GpioDataRegs.GPACLEAR.bit.GPIO18 = 1;  // Retract
    }
}
//###########################################################################
// FILE:    LED.c
//
// TITLE:   LED output file
//
// DESCRIPTION:
//
// The program produces LED outputs.
//
//###########################################################################
// Matthew Saylor
// March 03, 2009
//@end

#include "Project.h"     // Project Headerfile
void LEDon(int number);
void LEDoff(int number);
void LED_init(void);

void LED_init(void)
{
    EALLOW;
    GpioCtrlRegs.GPCPUD.bit.GPIO79 = 0;  // Enable pullup on
    GpioDataRegs.GPCCLEAR.bit.GPIO79 = 1;  // Output 0
    GpioCtrlRegs.GPCMUX1.bit.GPIO79 = 0; // GPIO79 = GPIO79
    GpioCtrlRegs.GPCDIR.bit.GPIO79 = 1;   // GPIO79 = output
    
    GpioCtrlRegs.GPCPUD.bit.GPIO77 = 0;  // Enable pullup on GPIO77
    GpioDataRegs.GPCCLEAR.bit.GPIO77 = 1;  // Output 0
    GpioCtrlRegs.GPCMUX1.bit.GPIO77 = 0; // GPIO77 = GPIO77
    GpioCtrlRegs.GPCDIR.bit.GPIO77 = 1;   // GPIO77 = output
    
    GpioCtrlRegs.GPCPUD.bit.GPIO75 = 0;  // Enable pullup on GPIO75
    GpioDataRegs.GPCCLEAR.bit.GPIO75 = 1;  // Output 0
    GpioCtrlRegs.GPCMUX1.bit.GPIO75 = 0; // GPIO75 = GPIO75
    GpioCtrlRegs.GPCDIR.bit.GPIO75 = 1;   // GPIO75 = output
    
    GpioCtrlRegs.GPCPUD.bit.GPIO73 = 0;  // Enable pullup on GPIO73
    GpioDataRegs.GPCCLEAR.bit.GPIO73 = 1;  // Output 0
    GpioCtrlRegs.GPCMUX1.bit.GPIO73 = 0; // GPIO73 = GPIO73
    GpioCtrlRegs.GPCDIR.bit.GPIO73 = 1;   // GPIO73 = output
}
GpioCtrlRegs.GPCPUD.bit.GPIO71 = 0;  // Enable pullup on GPIO71
GpioDataRegs.GPCCLEAR.bit.GPIO71 = 1;  // Output 0
GpioCtrlRegs.GPCMUX1.bit.GPIO71 = 0; // GPIO71 = GPIO71
GpioCtrlRegs.GPCDIR.bit.GPIO71 = 1;   // GPIO71 = output

GpioCtrlRegs.GPCPUD.bit.GPIO69 = 0;  // Enable pullup on GPIO69
GpioDataRegs.GPCCLEAR.bit.GPIO69 = 1;  // Output 0
GpioCtrlRegs.GPCMUX1.bit.GPIO69 = 0; // GPIO69 = GPIO69
GpioCtrlRegs.GPCDIR.bit.GPIO69 = 1;   // GPIO69 = output

GpioCtrlRegs.GPCPUD.bit.GPIO67 = 0;  // Enable pullup on GPIO67
GpioDataRegs.GPCCLEAR.bit.GPIO67 = 1;  // Output 0
GpioCtrlRegs.GPCMUX1.bit.GPIO67 = 0; // GPIO67 = GPIO67
GpioCtrlRegs.GPCDIR.bit.GPIO67 = 1;   // GPIO67 = output

GpioCtrlRegs.GPCPUD.bit.GPIO65 = 0;  // Enable pullup on GPIO65
GpioDataRegs.GPCCLEAR.bit.GPIO65 = 1;  // Output 0
GpioCtrlRegs.GPCMUX1.bit.GPIO65 = 0; // GPIO65 = GPIO65
GpioCtrlRegs.GPCDIR.bit.GPIO65 = 1;   // GPIO65 = output

GpioCtrlRegs.GPBPUOD.bit.GPIO40 = 0; // Enable pullup on GPIO40
GpioDataRegs.GPBCLEAR.bit.GPIO40 = 1;  // Output 0
GpioCtrlRegs.GPBMUX1.bit.GPIO40 = 0; // GPIO40 = GPIO40
GpioCtrlRegs.GPBDIR.bit.GPIO40 = 1;   // GPIO40 = output

EDIS;
}

void LEDon(int number)
{
    switch(number)
    {
        case 0:
            GpioDataRegs.GPCSET.bit.GPIO79 = 1;
            break;

        case 1:
            GpioDataRegs.GPCSET.bit.GPIO77 = 1;
            break;

        case 2:
            GpioDataRegs.GPCSET.bit.GPIO75 = 1;
            break;

        case 3:
            GpioDataRegs.GPCSET.bit.GPIO73 = 1;
            break;
    }
break;

case 4:
GpioDataRegs.GPCSET.bit.GPIO71 = 1;
break;

case 5:
GpioDataRegs.GPCSET.bit.GPIO69 = 1;
break;

case 6:
GpioDataRegs.GPCSET.bit.GPIO67 = 1;
break;

case 7:
GpioDataRegs.GPCSET.bit.GPIO65 = 1;
break;

case 8:
GpioDataRegs.GPBSET.bit.GPIO40 = 1;
break;

void LEDoff(int number)
{
    switch(number)
    {
        case 0:
            GpioDataRegs.GPCCLEAR.bit.GPIO79 = 1;
            break;

        case 1:
            GpioDataRegs.GPCCLEAR.bit.GPIO77 = 1;
            break;

        case 2:
            GpioDataRegs.GPCCLEAR.bit.GPIO75 = 1;
            break;

        case 3:
            GpioDataRegs.GPCCLEAR.bit.GPIO73 = 1;
break;

case 4:
    GpioDataRegs.GPCCLEAR.bit.GPIO71 = 1;
    break;

case 5:
    GpioDataRegs.GPCCLEAR.bit.GPIO69 = 1;
    break;

case 6:
    GpioDataRegs.GPCCLEAR.bit.GPIO67 = 1;
    break;

case 7:
    GpioDataRegs.GPCCLEAR.bit.GPIO65 = 1;
    break;

case 8:
    GpioDataRegs.GPBCLEAR.bit.GPIO40 = 1;
    break;

}
#include "Project.h" //Project Headerfile

void LED_init(void);
void Test(int number);

void main(void)
{
  // Initialize System Control:

  InitSysCtrl();

  // Flash control registers must be copied from flash (its load address)
  // to RAM (its run address) at runtime. Or unexpected errors may occur.

  memcpy(&secureRamFuncs_runstart,
         &secureRamFuncs_loadstart,
         &secureRamFuncs_loadend - &secureRamFuncs_loadstart);

  InitFlash(); //Initialize Flash

  PIN_setup(); //Setup GPIO pins used in the robot
  LED_init();  //Initializes the LEDs used in testing

  Scic_init(); //Initializes SCI-C to transmit
DelayUs(10000); //Delay in micro seconds
DelayUs(10000); //Delay in micro seconds

Scic_xmit(170); // Autobauding Character sent to drivers
DelayUs(10000); //Delay in micro seconds
DelayUs(10000); //Delay in micro seconds

for(;;)
{

    if(GpioDataRegs.GPADAT.bit.GPIO5 == 0)
    {
        // Test 1  Movement test
        // Test 2  Sensor/Kicker test
        Delay(1);
        Test(1);
    }

    else if(GpioDataRegs.GPADAT.bit.GPIO5 == 1)
    {
        Test(2);
    }

}
#include "Project.h"  // Project Headerfile

// The following functions are used to calculate the speed and direction for each motor for the desired angle
char Motor_Phi(int phi);
char Motor_Power1(int phi,int z);
char Motor_Direct1(int phi);
char Motor_Power2(int phi,int z);
char Motor_Direct2(int phi);
char Motor_Power3(int phi,int z);
char Motor_Direct3(int phi);
char Motor_Power4(int phi,int z);
char Motor_Direct4(int phi);
void Motor(int motor, char dx, char mx);

char m1;
char d1;
char m2;
char d2;
char m3;
char d3;
char m4;
char d4;
char mout;
int z;
int i;

void Movement(int phi) //Used for all angles @ full power
{
    z=Motor_Phi(phi);
    m1=Motor_Power1(phi,z);
    d1=Motor_Direct1(phi);
    m2=Motor_Power2(phi,z);
    d2=Motor_Direct2(phi);
    m3=Motor_Power3(phi,z);
    d3=Motor_Direct3(phi);
    m4=Motor_Power4(phi,z);
    d4=Motor_Direct4(phi);

    Motor(1,d1,m1);
    Motor(2,d2,m2);
    Motor(3,d3,m3);
    Motor(4,d4,m4);
}

void Movement_1(int phi, int pwr) //Used for the 8 normal directions @ user defined power
{
    z=Motor_Phi(phi);
    m1=Motor_Power1(phi,z);
    d1=Motor_Direct1(phi);
    m2=Motor_Power2(phi,z);
    d2=Motor_Direct2(phi);
    m3=Motor_Power3(phi,z);
    d3=Motor_Direct3(phi);
    m4=Motor_Power4(phi,z);
    d4=Motor_Direct4(phi);

    switch (phi)
    {
    case 0:
        Motor(1,d1,pwr);
        Motor(2,d2,pwr);
        Motor(3,d3,pwr);
        Motor(4,d4,pwr);
        break;
    }
case 45:
    Motor(1,d1,pwr);
    Motor(2,d2,0);
    Motor(3,d3,pwr);
    Motor(4,d4,0);
    break;

case 90:
    Motor(1,d1,pwr);
    Motor(2,d2,pwr);
    Motor(3,d3,pwr);
    Motor(4,d4,pwr);
    break;

case 135:
    Motor(1,d1,0);
    Motor(2,d2,pwr);
    Motor(3,d3,0);
    Motor(4,d4,pwr);
    break;

case 180:
    Motor(1,d1,pwr);
    Motor(2,d2,pwr);
    Motor(3,d3,pwr);
    Motor(4,d4,pwr);
    break;

case 225:
    Motor(1,d1,pwr);
    Motor(2,d2,0);
    Motor(3,d3,pwr);
    Motor(4,d4,0);
    break;

case 270:
    Motor(1,d1,pwr);
    Motor(2,d2,pwr);
    Motor(3,d3,pwr);
    Motor(4,d4,pwr);
    break;

case 315:
    Motor(1,d1,0);
    Motor(2,d2,pwr);
    Motor(3,d3,0);
void Stop(int motor)
{
    char add;
    char check;

    switch (motor)
    {
        case 1:
            add=128;
            break;
        case 2:
            add=130;
            break;
        case 3:
            add=132;
            break;
        case 4:
            add=134;
            break;
    }
    check= (add & 127);
    Scic_xmit(add);
    Scic_xmit(0);
    Scic_xmit(0);
    Scic_xmit(check);
}

void Rotate(int rot, int spd)
{
    char dir;

    switch (rot)
    {
        case 1:
            dir=0;
            break;
        case 2:
dir=1;
break;

Motor(1,dir,spd);
Motor(2,dir,spd);
Motor(3,dir,spd);
Motor(4,dir,spd);

char  Motor_Power1(int phi,int z)
{
    if (phi<91 || phi>341){
        m1=127;
    }
    else if ((phi>90 && phi<109) ||(phi>314 &&phi<342)){
        m1=z;
    }
    else if (phi>108 && phi<162){
        m1=0;
    }
    else if ((phi>161&& phi<180)|| (phi>288 && phi<315)){
        m1=z;
    }
    else if (phi>179 && phi<289){
        m1=127;
    }
    return m1;
}

char  Motor_Direct1(int phi)
{
    if (phi<91 || phi>341){
        d1=0;
    }
    else if ((phi>90 && phi<109) ||(phi>314 &&phi<342)){
        d1=0;
    }
    else if (phi>108 && phi<162){
        d1=0;
    }
    else if ((phi>161&& phi<180)|| (phi>288 && phi<315)){
        d1=1;
    }
    else if (phi>179 && phi<289){
d1=1;
    
    return d1;

}

char Motor_Power2(int phi, int z)
{
    if((phi<19) || (phi>269)){
        m2=127;
    }
    else if((phi>18 && phi<46) || (phi>251 && phi<270)){
        m2=z;
    }
    else if((phi>45 && phi<72) || (phi>180 && phi<199)){
        m2=z;
    }
    else if(phi>71 && phi<181){
        m2=127;
    }
    else if(phi>198 && phi<255){
        m2=0;
    }
    return m2;
}

char Motor_Direct2(int phi)
{
    if((phi<19) || (phi>269)){
        d2=1;
    }
    else if((phi>18 && phi<46) || (phi>251 && phi<270)){
        d2=1;
    }
    else if((phi>45 && phi<72) || (phi>180 && phi<199)){
        d2=0;
    }
    else if(phi>71 && phi<181){
        d2=0;
    }
    else if(phi>198 && phi<255){
        d2=0;
    }
    return d2;
}

char Motor_Power3(int phi, int z)
if(phi<109)
    m3=127;
else if(phi>161 && phi<271)
    m3=127;
else if(phi>288 && phi<342)
    m3=0;
else if((phi>108 && phi<135) || (phi>341))
    m3=z;
    return m3;
}

char Motor_Direct3(int phi)
{
    if(phi<109)
        d3=1;
    else if(phi>161 && phi<271)
        d3=0;
    else if(phi>288 && phi<342)
        d3=0;
    else if((phi>108 && phi<135) || (phi>341))
        d3=1;
    else if((phi>270 && phi<289)||(phi>134 && phi<162))
        d3=0;
    return d3;
}

char Motor_Power4(int phi,int  z)
{
    if(phi>89 && phi<199)
        m4=127;
    else if(phi>11 && phi<71)
        m4=0;
    return m4;
char Motor_Direct4(int phi)
{
    if(phi>89 && phi<199) {
        d4=1;
    } else if(phi>18 && phi<71) {
        d4=0;
    } else if((phi>=1 && phi<19) || (phi>225 && phi<252)) {
        d4=0;
    } else if((phi>71 && phi<90) || (phi>198 && phi<226)) {
        d4=1;
    } else if(phi==0 || (phi>251 && phi<360)) {
        d4=0;
    }
    return d4;
}

void Motor(int motor, char dx, char mx)
{
    char add;
    char check;

    switch (motor)
    {
        case 1:
            add=128;
            break;
        case 2:
            add=130;
            break;
        case 3:
            add=131;
            break;
        case 4:
            add=132;
            break;
    }
break;
case 3:
    add=132;
    break;
case 4:
    add=134;
    break;
}
check= ((add+dx+mx) & 127);
Scie_xmit(add);
Scie_xmit(dx);
Scie_xmit(mx);
Scie_xmit(check);
}

char  Motor_Phi(int phi)
{
  int  z;
  switch (phi)
  {
    case 0:
      z=127;
      break;
    case 1:
      z=118;
      break;
    case 2:
      z=109;
      break;
    case 3:
      z=101;
      break;
    case 4:
      z=93;
      break;
    case 5:
      z=86;
      break;
    case 6:
      z=78;
      break;
    case 7:
      z=71;
      break;
case 8:
    z=64;
    break;
case 9:
    z=57;
    break;
case 10:
    z=50;
    break;
case 11:
    z=44;
    break;
case 12:
    z=37;
    break;
case 13:
    z=31;
    break;
case 14:
    z=25;
    break;
case 15:
    z=19;
    break;
case 16:
    z=13;
    break;
case 17:
    z=8;
    break;
case 18:
    z=5;
    break;
case 19:
    z=123;
    break;
case 20:
    z=118;
    break;
case 21:
    z=113;
    break;
case 22:
    z=107;
    break;
case 23:
z=102;
break;
case 24:
    z=97;
    break;
case 25:
    z=92;
    break;
case 26:
    z=87;
    break;
case 27:
    z=82;
    break;
case 28:
    z=77;
    break;
case 29:
    z=72;
    break;
case 30:
    z=68;
    break;
case 31:
    z=63;
    break;
case 32:
    z=58;
    break;
case 33:
    z=53;
    break;
case 34:
    z=49;
    break;
case 35:
    z=44;
    break;
case 36:
    z=40;
    break;
case 37:
    z=35;
    break;
case 38:
    z=31;
break;
case 39:
  z=26;
  break;
case 40:
  z=22;
  break;
case 41:
  z=17;
  break;
case 42:
  z=13;
  break;
case 43:
  z=8;
  break;
case 44:
  z=5;
  break;
case 45:
  z=0;
  break;
case 46:
  z=4;
  break;
case 47:
  z=8;
  break;
case 48:
  z=13;
  break;
case 49:
  z=17;
  break;
case 50:
  z=22;
  break;
case 51:
  z=26;
  break;
case 52:
  z=31;
  break;
case 53:
  z=35;
  break;
case 54:
    z=40;
    break;

case 55:
    z=44;
    break;

case 56:
    z=49;
    break;

case 57:
    z=53;
    break;

case 58:
    z=58;
    break;

case 59:
    z=63;
    break;

case 60:
    z=68;
    break;

case 61:
    z=72;
    break;

case 62:
    z=77;
    break;

case 63:
    z=82;
    break;

case 64:
    z=87;
    break;

case 65:
    z=92;
    break;

case 66:
    z=97;
    break;

case 67:
    z=102;
    break;

case 68:
    z=107;
    break;

case 69:
\[ z=113; \]
break;
case 70:
  \[ z=118; \]
break;
case 71:
  \[ z=123; \]
break;
case 72:
  \[ z=5; \]
break;
case 73:
  \[ z=8; \]
break;
case 74:
  \[ z=13; \]
break;
case 75:
  \[ z=19; \]
break;
case 76:
  \[ z=25; \]
break;
case 77:
  \[ z=31; \]
break;
case 78:
  \[ z=37; \]
break;
case 79:
  \[ z=44; \]
break;
case 80:
  \[ z=50; \]
break;
case 81:
  \[ z=57; \]
break;
case 82:
  \[ z=64; \]
break;
case 83:
  \[ z=71; \]
break;
case 84:
  \[ z=78; \]
case 85:
    z=86;
    break;
case 86:
    z=93;
    break;
case 87:
    z=101;
    break;
case 88:
    z=109;
    break;
case 89:
    z=118;
    break;
case 90:
    z=127;
    break;
case 91:
    z=118;
    break;
case 92:
    z=109;
    break;
case 93:
    z=101;
    break;
case 94:
    z=93;
    break;
case 95:
    z=86;
    break;
case 96:
    z=78;
    break;
case 97:
    z=71;
    break;
case 98:
    z=64;
    break;
case 99:
    z=57;
    break;
case 100:
    z=50;
    break;
case 101:
    z=44;
    break;
case 102:
    z=37;
    break;
case 103:
    z=31;
    break;
case 104:
    z=25;
    break;
case 105:
    z=19;
    break;
case 106:
    z=13;
    break;
case 107:
    z=8;
    break;
case 108:
    z=5;
    break;
case 109:
    z=123;
    break;
case 110:
    z=118;
    break;
case 111:
    z=113;
    break;
case 112:
    z=107;
    break;
case 113:
    z=102;
    break;
case 114:
    z=97;
    break;
case 115:
z=92;
break;
case 116:
    z=87;
    break;
case 117:
    z=82;
    break;
case 118:
    z=77;
    break;
case 119:
    z=72;
    break;
case 120:
    z=68;
    break;
case 121:
    z=63;
    break;
case 122:
    z=58;
    break;
case 123:
    z=53;
    break;
case 124:
    z=49;
    break;
case 125:
    z=44;
    break;
case 126:
    z=40;
    break;
case 127:
    z=35;
    break;
case 128:
    z=31;
    break;
case 129:
    z=26;
    break;
case 130:
    z=22;
break;
case 131:
    z=17;
    break;
case 132:
    z=13;
    break;
case 133:
    z=8;
    break;
case 134:
    z=5;
    break;
case 135:
    z=0;
    break;
case 136:
    z=5;
    break;
case 137:
    z=8;
    break;
case 138:
    z=13;
    break;
case 139:
    z=17;
    break;
case 140:
    z=22;
    break;
case 141:
    z=26;
    break;
case 142:
    z=31;
    break;
case 143:
    z=35;
    break;
case 144:
    z=40;
    break;
case 145:
    z=44;
    break;
case 146:
    z=49;
    break;
case 147:
    z=53;
    break;
case 148:
    z=58;
    break;
case 149:
    z=63;
    break;
case 150:
    z=68;
    break;
case 151:
    z=72;
    break;
case 152:
    z=77;
    break;
case 153:
    z=82;
    break;
case 154:
    z=87;
    break;
case 155:
    z=92;
    break;
case 156:
    z=97;
    break;
case 157:
    z=102;
    break;
case 158:
    z=107;
    break;
case 159:
    z=113;
    break;
case 160:
    z=118;
    break;
case 161:
z=123;
break;
case 162:
z=5;
break;
case 163:
z=8;
break;
case 164:
z=13;
break;
case 165:
z=19;
break;
case 166:
z=25;
break;
case 167:
z=31;
break;
case 168:
z=37;
break;
case 169:
z=44;
break;
case 170:
z=50;
break;
case 171:
z=57;
break;
case 172:
z=64;
break;
case 173:
z=71;
break;
case 174:
z=78;
break;
case 175:
z=86;
break;
case 176:
z=93;
break;
case 177:
    z=101;
    break;
case 178:
    z=109;
    break;
case 179:
    z=118;
    break;
case 180:
    z=127;
    break;
case 181:
    z=118;
    break;
case 182:
    z=109;
    break;
case 183:
    z=101;
    break;
case 184:
    z=93;
    break;
case 185:
    z=86;
    break;
case 186:
    z=78;
    break;
case 187:
    z=71;
    break;
case 188:
    z=64;
    break;
case 189:
    z=57;
    break;
case 190:
    z=50;
    break;
case 191:
    z=44;
    break;
case 192:
    z=37;
    break;
case 193:
    z=31;
    break;
case 194:
    z=25;
    break;
case 195:
    z=19;
    break;
case 196:
    z=13;
    break;
case 197:
    z=8;
    break;
case 198:
    z=5;
    break;
case 199:
    z=123;
    break;
case 200:
    z=118;
    break;
case 201:
    z=113;
    break;
case 202:
    z=107;
    break;
case 203:
    z=102;
    break;
case 204:
    z=97;
    break;
case 205:
    z=92;
    break;
case 206:
    z=87;
    break;
case 207:
z=82;
break;
case 208:
  z=77;
  break;
case 209:
  z=72;
  break;
case 210:
  z=68;
  break;
case 211:
  z=63;
  break;
case 212:
  z=58;
  break;
case 213:
  z=53;
  break;
case 214:
  z=49;
  break;
case 215:
  z=44;
  break;
case 216:
  z=40;
  break;
case 217:
  z=35;
  break;
case 218:
  z=31;
  break;
case 219:
  z=26;
  break;
case 220:
  z=22;
  break;
case 221:
  z=17;
  break;
case 222:
  z=13;
break;
case 223:
    z=8;
    break;
case 224:
    z=5;
    break;
case 225:
    z=0;
    break;
case 226:
    z=5;
    break;
case 227:
    z=8;
    break;
case 228:
    z=13;
    break;
case 229:
    z=17;
    break;
case 230:
    z=22;
    break;
case 231:
    z=26;
    break;
case 232:
    z=31;
    break;
case 233:
    z=35;
    break;
case 234:
    z=40;
    break;
case 235:
    z=44;
    break;
case 236:
    z=49;
    break;
case 237:
    z=53;
    break;
case 238:
    z=58;
    break;
case 239:
    z=63;
    break;
case 240:
    z=68;
    break;
case 241:
    z=72;
    break;
case 242:
    z=77;
    break;
case 243:
    z=82;
    break;
case 244:
    z=87;
    break;
case 245:
    z=92;
    break;
case 246:
    z=97;
    break;
case 247:
    z=102;
    break;
case 248:
    z=107;
    break;
case 249:
    z=113;
    break;
case 250:
    z=118;
    break;
case 251:
    z=123;
    break;
case 252:
    z=5;
    break;
case 253:
z=8;
break;
case 254:
    z=13;
    break;
case 255:
    z=19;
    break;
case 256:
    z=25;
    break;
case 257:
    z=31;
    break;
case 258:
    z=37;
    break;
case 259:
    z=44;
    break;
case 260:
    z=50;
    break;
case 261:
    z=57;
    break;
case 262:
    z=64;
    break;
case 263:
    z=71;
    break;
case 264:
    z=78;
    break;
case 265:
    z=86;
    break;
case 266:
    z=93;
    break;
case 267:
    z=101;
    break;
case 268:
    z=109;
break;
case 269:
    z=118;
    break;
case 270:
    z=127;
    break;
case 271:
    z=118;
    break;
case 272:
    z=109;
    break;
case 273:
    z=101;
    break;
case 274:
    z=93;
    break;
case 275:
    z=86;
    break;
case 276:
    z=78;
    break;
case 277:
    z=71;
    break;
case 278:
    z=64;
    break;
case 279:
    z=57;
    break;
case 280:
    z=50;
    break;
case 281:
    z=44;
    break;
case 282:
    z=37;
    break;
case 283:
    z=31;
    break;
case 284:
    z=25;
    break;
    
case 285:
    z=19;
    break;
    
case 286:
    z=13;
    break;
    
case 287:
    z=8;
    break;
    
case 288:
    z=5;
    break;
    
case 289:
    z=123;
    break;
    
case 290:
    z=118;
    break;
    
case 291:
    z=113;
    break;
    
case 292:
    z=107;
    break;
    
case 293:
    z=102;
    break;
    
case 294:
    z=97;
    break;
    
case 295:
    z=92;
    break;
    
case 296:
    z=87;
    break;
    
case 297:
    z=82;
    break;
    
case 298:
    z=77;
    break;
    
case 299:
z=72;
break;
case 300:
    z=68;
    break;
case 301:
    z=63;
    break;
case 302:
    z=57;
    break;
case 303:
    z=53;
    break;
case 304:
    z=49;
    break;
case 305:
    z=44;
    break;
case 306:
    z=40;
    break;
case 307:
    z=35;
    break;
case 308:
    z=31;
    break;
case 309:
    z=26;
    break;
case 310:
    z=22;
    break;
case 311:
    z=17;
    break;
case 312:
    z=13;
    break;
case 313:
    z=8;
    break;
case 314:
    z=5;
break;
case 315:
    z=0;
    break;
case 316:
    z=5;
    break;
case 317:
    z=8;
    break;
case 318:
    z=13;
    break;
case 319:
    z=17;
    break;
case 320:
    z=22;
    break;
case 321:
    z=26;
    break;
case 322:
    z=31;
    break;
case 323:
    z=35;
    break;
case 324:
    z=40;
    break;
case 325:
    z=44;
    break;
case 326:
    z=49;
    break;
case 327:
    z=53;
    break;
case 328:
    z=58;
    break;
case 329:
    z=63;
    break;
case 330:
    z=68;
    break;
case 331:
    z=72;
    break;
case 332:
    z=77;
    break;
case 333:
    z=82;
    break;
case 334:
    z=87;
    break;
case 335:
    z=92;
    break;
case 336:
    z=97;
    break;
case 337:
    z=102;
    break;
case 338:
    z=107;
    break;
case 339:
    z=113;
    break;
case 340:
    z=118;
    break;
case 341:
    z=123;
    break;
case 342:
    z=2;
    break;
case 343:
    z=8;
    break;
case 344:
    z=13;
    break;
case 345:
z=19;
break;
case 346:
z=25;
break;
case 347:
z=31;
break;
case 348:
z=37;
break;
case 349:
z=44;
break;
case 350:
z=50;
break;
case 351:
z=57;
break;
case 352:
z=64;
break;
case 353:
z=71;
break;
case 354:
z=78;
break;
case 355:
z=86;
break;
case 356:
z=93;
break;
case 357:
z=101;
break;
case 358:
z=109;
break;
case 359:
z=118;
break;
}
return z;
* File: Passwords.asm
* Devices: TMS320F2833x
* Author: David M. Alter, Texas Instruments Inc.
* History:
*   12/18/07 - original (D. Alter)
* Notes:
*  1) The section "passwords" contains the actual CSM passwords that get
   linked to the CSM password locations in flash. The user must know
   what these passwords are in order to unlock the CSM.
*  2) Link the section "passwords" to the memory PASSWORDS on page 0.
*  3) It is recommended that all passwords be left as 0xFFFF during code
   development. Passwords of 0xFFFF are dummy passwords, and do not
   lock the code security module (Dummy reads of CSM PWL registers
   will unlock the CSM). When code development is complete, the user
   can modify the passwords to activate the code security module.
*  4) The section "csm_rsvd" is required when using code security.
*  5) Link the section "csm_rsvd" to the memory CSM_RSVD on page 0.
***********************************************************************

.sect "passwords"
.int 0xFFFF  ;PWL0 (LSW of 128-bit password)
.int 0xFFFF  ;PWL1
.int 0xFFFF  ;PWL2
.int 0xFFFF  ;PWL3
.int 0xFFFF  ;PWL4
.int 0xFFFF  ;PWL5
.int 0xFFFF  ;PWL6
.int 0xFFFF  ;PWL7 (MSW of 128-bit password)
***********************************************************************

.sect "csm_rsvd"
.loop (33FFF5h - 33FF80h + 1)
.int 0x0000
.endloop
***********************************************************************

.end
; end of file passwords.asm
// FILE:    PIN_setup.c
//
// TITLE:   GPIO  Pin Setup
//
// DESCRIPTION:
//
// Configures PINS for TMS320F28335 EZDSP Target board
//
//
//###########################################################################
// Matthew Saylor
// March 3, 2009
//###########################################################################

#include "Project.h"     // Project Headerfile

void PIN_setup(void)
{
    EALLOW; //Allows Access to Protected Space

    // Sets input pins for Hall Effect Sensors on P8
    GpioCtrlRegs.GPAPUD.bit.GPIO0 = 0; // Enable pullup on GPIO0
    GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 0; // GPIO0 = GPIO0 (PIN 9)
    GpioCtrlRegs.GPADIR.bit.GPIO0 = 0;  // GPIO0 = input
    GpioCtrlRegs.GPAPUD.bit.GPIO2 = 0;  // Enable pullup on GPIO2
    GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 0; // GPIO2 = GPIO2 (PIN 11)
    GpioCtrlRegs.GPADIR.bit.GPIO2 = 0;  // GPIO2 = input
    GpioCtrlRegs.GPAPUD.bit.GPIO4 = 0;  // Enable pullup on GPIO4
    GpioCtrlRegs.GPAMUX1.bit.GPIO4 = 0; // GPIO4 = GPIO4 (PIN 13)
    GpioCtrlRegs.GPADIR.bit.GPIO4 = 0;  // GPIO4 = input
    GpioCtrlRegs.GPAPUD.bit.GPIO27 = 0; // Enable pullup on GPIO27
    GpioCtrlRegs.GPAMUX2.bit.GPIO27 = 0; // GPIO27 = GPIO27 (PIN 15)
    GpioCtrlRegs.GPADIR.bit.GPIO27 = 0; // GPIO27 = input

    // Sets input pins for Reed Switches on P8
    GpioCtrlRegs.GPAPUD.bit.GPIO7 = 0; // Enable pullup on GPIO7
    GpioCtrlRegs.GPAMUX1.bit.GPIO7 = 0; // GPIO7 = GPIO7 (PIN 21)
    GpioCtrlRegs.GPADIR.bit.GPIO7 = 0;  // GPIO7 = input
    GpioCtrlRegs.GPAPUD.bit.GPIO16 = 0; // Enable pullup on GPIO16
    GpioCtrlRegs.GPAMUX2.bit.GPIO16 = 0; // GPIO16 = GPIO16 (PIN 23)
    GpioCtrlRegs.GPADIR.bit.GPIO16 = 0; // GPIO16 = input

    // Sets input pins for Proximity Switch on P8
GpioCtrlRegs.GPAPUD.bit.GPIO13 = 0;  // Enable pullup on GPIO13
GpioCtrlRegs.GPAMUX1.bit.GPIO13 = 0;  // GPIO13 = GPIO13 (PIN 17)
GpioCtrlRegs.GPADIR.bit.GPIO13 = 0;  // GPIO13 = input

// Enable SCI-C For motor drivers on P10
GpioCtrlRegs.GPBPU1.bit.GPIO62 = 0;   // Enable pullup on GPIO62 (PIN 4)
GpioCtrlRegs.GPBQSEL2.bit.GPIO62 = 3; // Asynch input
GpioCtrlRegs.GPBMUX2.bit.GPIO62 = 1;  // GPIO62 = SCIRXDA
GpioCtrlRegs.GPBPU1.bit.GPIO63 = 0;   // Enable pullup on GPIO63 (PIN 3)
GpioCtrlRegs.GPBMUX2.bit.GPIO63 = 1;  // GPIO63 = SCITXDA

// Enable SCI-A on GPIO28 - GPIO29 For RS-232 on P12
GpioCtrlRegs.GPAPUD.bit.GPIO28 = 0;   // Enable pullup on GPIO28
GpioCtrlRegs.GPAQSEL2.bit.GPIO28 = 3; // Asynch input
GpioCtrlRegs.GPAMUX2.bit.GPIO28 = 1;  // GPIO28 = SCIRXDA
GpioCtrlRegs.GPAPUD.bit.GPIO29 = 0;   // Enable pullup on GPIO29
GpioCtrlRegs.GPAMUX2.bit.GPIO29 = 1;  // GPIO29 = SCITXDA

// Enable CAN-A on GPIO30 - GPIO31 For CAN A on P11
GpioCtrlRegs.GPAPUD.bit.GPIO30 = 0;   // Enable pullup on GPIO30
GpioCtrlRegs.GPAQSEL1.bit.GPIO30 = 3; // Asynch input
GpioCtrlRegs.GPAMUX1.bit.GPIO30 = 1;  // GPIO30 = CANTXA
GpioCtrlRegs.GPAMUX1.bit.GPIO31 = 3;  // Asynch input
GpioCtrlRegs.GPAMUX2.bit.GPIO31 = 1;  // GPIO31 = CANRXA

// Enable SCI-B on GPIO9 - GPIO11 For USB on P8
GpioCtrlRegs.GPAPUD.bit.GPIO11 = 0;   // Enable pullup on GPIO11 (PIN 31)
GpioCtrlRegs.GPAQSEL1.bit.GPIO11 = 3; // Asynch input
GpioCtrlRegs.GPAMUX1.bit.GPIO11 = 1;  // GPIO11 = SCIRXDB
GpioCtrlRegs.GPAPUD.bit.GPIO9 = 0;    // Enable pullup on GPIO9 (PIN 29)
GpioCtrlRegs.GPAMUX1.bit.GPIO9 = 1;   // GPIO9 = SCITXDB

    //Enable output on GPIO18 for Kicker on P8
GpioCtrlRegs.GPAPUD.bit.GPIO18 = 0;   // Enable pullup on GPIO18
GpioDataRegs.GPACLEAR.bit.GPIO18 = 1; // Output 0
GpioCtrlRegs.GPAMUX2.bit.GPIO18 = 0;  // GPIO18 = GPIO18 (PIN 25)
GpioCtrlRegs.GPADIR.bit.GPIO18 = 1;   // GPIO18 = output

/******************************************************************************/
    // Sets input pins for Start to Test on P8
GpioCtrlRegs.GPAPUD.bit.GPIO5 = 0;    // Enable pullup on GPIO13
GpioCtrlRegs.GPAMUX1.bit.GPIO5 = 0;   // GPIO13 = GPIO13 (PIN 17)
GpioCtrlRegs.GPADIR.bit.GPIO5 = 0;    // GPIO13 = input

EDIS;  //Disables access to protected space
#include "Project.h"  // Project Headerfile

void LEDon(int number);
void LEDoff(int number);

int Reed(int number)
{
    int r;

    switch (number)
    {
    case 1:
    {
        if (GpioDataRegs.GPADAT.bit.GPIO7 == 0) { //no voltage at reed switch
            // Turn LED on
            LEDon(4);
            r = 1;
        }
        else if (GpioDataRegs.GPADAT.bit.GPIO7 == 1) { //voltage at reed switch
            // Turn LED off
            LEDoff(4);
            r = 0;
        }
    }
    case 2:
    {  // Change the code here.
    }
if (GpioDataRegs.GPADAT.bit.GPIO16 == 0) {// no voltage at reed switch
extend
// Turn LED on
LEDOn(5);
r=1;
}
else if (GpioDataRegs.GPADAT.bit.GPIO16 == 1) {// voltage at reed switch extend
LEDoff(5);
r=0;
}
}
return r;

int Prox(void)
{
int p;
if (GpioDataRegs.GPADAT.bit.GPIO13 == 0) {// no voltage at prox 1v
// Turn LED on
LEDOn(6);
p=1;
}
else if (GpioDataRegs.GPADAT.bit.GPIO13 == 1) {// voltage at prox
LEDoff(6);
p=0;
}
return p;
}

int Hall(int number)
{
int h;
switch (number)
{
case 1:
{
if (GpioDataRegs.GPADAT.bit.GPIO0 == 0) {// no voltage at Hall #1
// Turn LED on
LEDOn(0);
h=1;
}
else if (GpioDataRegs.GPADAT.bit.GPIO0 == 1) { //voltage at Hall #1
    LEDoff(0);
    h=0;
}

}  //end of case 1

case 2:
{
    if (GpioDataRegs.GPADAT.bit.GPIO2 == 0) { //no voltage at Hall #2
        // Turn LED on
        LEDon(1);
        h=1;
    }
    else if (GpioDataRegs.GPADAT.bit.GPIO2 == 1) { //voltage at Hall #2
        LEDoff(1);
        h=0;
    }
}

}  //end of case 2

case 3:
{
    if (GpioDataRegs.GPADAT.bit.GPIO4 == 0) { //no voltage at Hall #3
        // Turn LED on
        LEDon(2);
        h=1;
    }
    else if (GpioDataRegs.GPADAT.bit.GPIO4 == 1) { //voltage at Hall #3
        LEDoff(2);
        h=0;
    }
}

}  //end of case 3

case 4:
{
    if (GpioDataRegs.GPADAT.bit.GPIO27 == 0) { //no voltage at Hall #4
        // Turn LED on
        LEDon(3);
        h=1;
    }
    else if (GpioDataRegs.GPADAT.bit.GPIO27 == 1) { //voltage at Hall #4
        LEDoff(3);
        h=0;
    }
}

}  //end of case 4

return h;
#include "Project.h"    // Project Headerfile

void Scia_init()  //Initialize SCI-A
{

    SciaRegs.SCIFFTX.all=0x8000;  // Reset FIFO's

    SciaRegs.SCICCR.all =0x0007;   // 1 stop bit, No loopback
        // No parity,8 char bits,
        // async mode, idle-line protocol
    SciaRegs.SCICRLB.all =0x0003; // enable TX, RX, internal SCICLK,

    SciaRegs.SCICTL2.all = 0x0000; // Diable Rx Interrupt, Tx Interrupt
    SciaRegs.SCICL2.bit.RXBKINTENA =1;

    SciaRegs.SCIHBAUD    =0x0001;   // 9600 baud @LSPCLK = 37.5MHz.
    SciaRegs.SCILBAUD    =0x00E7;

    SciaRegs.SCIHBAUD    =0x0000;
    SciaRegs.SCILBAUD    =0x00F3;  //19200 baud @LSPCK = 37.5MHz

    SciaRegs.SCIHBAUD    =0x0000;
    SciaRegs.SCILBAUD    =0x0079;  //38400 baud @LSPCK = 37.5MHz

void Scic_init() //Initialize SCI-C
{

    // Reset FIFO's
    ScicRegs.SCIFFTX.all=0x8000;
    ScicRegs.SCICCR.all =0x0007;   // 1 stop bit, No loopback
    // No parity, 8 char bits,
    // async mode, idle-line protocol
    ScicRegs.SCICTL1.all =0x0003;  // enable TX, RX, internal SCICLK,
    ScicRegs.SCICTL2.all = 0x0000; // Diable Rx Interrupt, Tx Interrupt
    ScicRegs.SCICTL2.bit.RXBKINTENA =1;

    // ScicRegs.SCIHBAUD   =0x0001; // 9600 baud @LSPCLK = 37.5MHz.
    // ScicRegs.SCILBAUD   =0x00E7;
    ScicRegs.SCIHBAUD    =0x0000;
    ScicRegs.SCILBAUD    =0x00F3;  // 19200 baud @LSPCK = 37.5MHz

    // ScicRegs.SCIHBAUD   =0x0000;
    // ScicRegs.SCILBAUD   =0x0079;  // 38400 baud @LSPCK = 37.5MHz

    ScicRegs.SCICTL1.all =0x0023;     // Relinquish SCI from Reset

    // Initalize the SCI FIFO
    ScicRegs.SCIFFTX.all=0xE040;
}

void Scia_xmit(char out)
{
    SciaRegs.SCITXBUF=out;
}

// Transmit a character from the SCI-A
SciaRegs.SCICTL1.all =0x0023;     // Relinquish SCI from Reset
// Initalize the SCI FIFO
SciaRegs.SCIFFTX.all=0xE040;
//ScicRegs.SCIFFTX.all=0x8000;
}

// Transmit a character from the SCI-C
void Scic_xmit(char mout)
{
    ScicRegs.SCITXBUF=mout;
}

* File: SetDBGIER.asm
* Devices: TMS320F2833x
* Author: David M. Alter, Texas Instruments Inc.
* History:
*   12/18/07 - original (D. Alter)

***********************************************************************

Function: SetDBGIER()
Description: Sets the DBGIER register (for realtime emulation)
DSP: TMS320F28335, TMS320F28334, TMS320F28332
Include files: none
Function Prototype: void SetDBGIER(unsigned int)
Useage: SetDBGIER(value);
Input Parameters: Uint16 value = value to put in DBGIER register
Return Value: none
Notes: none
***********************************************************************

.def _SetDBGIER
.text

_SetDBGIER:
    MOV  *SP++,AL
    POP  DBGIER
    LRETR

; end of function SetDBGIER()
***********************************************************************

.end
;end of file SetDBGIER.asm
#include "Project.h"

/* Function: InitSysCtrl()
 * Description: Initializes the F2833x CPU.
 *---------------------------------------------------------------------------*/
void InitSysCtrl(void)
{
  volatile Uint16 i;      // General purpose Uint16
  volatile int16 dummy;     // General purpose volatile int16

  asm(" EALLOW");       // Enable EALLOW

  //--- Memory Protection Configuration
  DevEmuRegs.PROTSTART = 0x0100;  // Write default value to protection
  start register
  DevEmuRegs.PROTRANGE = 0x00FF;  // Write default value to protection
  range register

  //--- Configure the PLL

  // Note: The DSP/BIOS configuration tool can also be used to initialize the PLL
  // instead of doing the initialization here.

  // Make sure the PLL is not running in limp mode
  if (SysCtrlRegs.PLLSTS.bit.MCLKSTS != 1) {
    PLL is not running in limp mode
    SysCtrlRegs.PLLSTS.bit.MCLKOFF = 1;       // Turn off
    missing clock detect before changing PLLCR
    SysCtrlRegs.PLLSTS.bit.DIVSEL = 0;        // DIVSEL
    must be 0 or 1 (/4 CLKIN mode) before changing PLLCR
    SysCtrlRegs.PLLCR.bit.DIV = 0x000A;       // PLLx10/4
    (because DIVSEL is /4)
// Wait for PLL to lock.
// During this time the CPU will run at OSCCLK/2 until the PLL is stable.
// Once the PLL is stable the CPU will automatically switch to the new PLL value.

// Code is not required to sit and wait for the PLL to lock. However,
// if the code does anything that is timing critical (e.g. something that
// relies on the CPU clock frequency to be at speed), then it is best to wait
// until PLL lock is complete. The watchdog should be disabled before this loop
// (e.g., as was done above), or fed within the loop.
while(SysCtrlRegs.PLLSTS.bit.PLLLOCKS != 1)  // Wait for PLLLOCKS bit to set
{
    SysCtrlRegs.WDKEY = 0x0055;     // Service the watchdog while waiting
    SysCtrlRegs.WDKEY = 0x00AA;     //   in case the user enabled it.
}

// After the PLL has locked, we are running in PLLx10/4 mode (since DIVSEL is /4).
// We can now enable the missing clock detect circuitry, and also change DIVSEL
// to /2. In this example, I will wait a bit of time to let inrush currents settle,
// and then change DIVSEL from /4 to /2. This is only an example. The amount of
// time you need to wait depends on the power supply feeding the DSP (i.e., how much
// voltage droop occurs due to the inrush currents, and how long it takes the
// voltage regulators to recover).
SysCtrlRegs.PLLSTS.bit.MCLKOFF = 0;    // Enable missing clock detect circuitry
DelayUs(20/2);
    // Wait 20 us (just an example). Remember we're running
    // at half-speed here, so divide function argument by 2.
    SysCtrlRegs.PLLSTS.bit.DIVSEL = 0x2;   // Change to /2 mode
}
else
{
    // PLL is running in limp mode
    // User should replace the below with a call to an appropriate function,
    // for example shutdown the system (since something is very wrong!).
    asm(" ESTOP0");
}

//-- Configure the clocks
SysCtrlRegs.HISPCP.all = 0x0000;  // Hi-speed periph clock prescaler, 
HSPCLK=SYSCLKOUT/1
SysCtrlRegs.LOSPCP.all = 0x0002;  // Lo-speed periph clock prescaler, 
LOSPCLK=SYSCLKOUT/4

SysCtrlRegs.PCLKCR3.bit.GPIOINENCLK = 1;  // GPIO input module is clocked
SysCtrlRegs.PCLKCR3.bit.XINTFENCLK = 1;  // XINTF module is clocked
SysCtrlRegs.PCLKCR3.bit.DMAENCLK = 1;  // SYSCLKOUT to DMA enabled
SysCtrlRegs.PCLKCR3.bit.CPUTIMER2ENCLK = 1;  // SYSCLKOUT to CPU Timer2 enabled
SysCtrlRegs.PCLKCR3.bit.CPUTIMER1ENCLK = 1;  // SYSCLKOUT to CPU Timer1 enabled
SysCtrlRegs.PCLKCR3.bit.CPUTIMER0ENCLK = 1;  // SYSCLKOUT to CPU Timer0 enabled

SysCtrlRegs.PCLKCR1.bit.EQEP2ENCLK = 1;  // SYSCLKOUT to eQEP2 enabled
SysCtrlRegs.PCLKCR1.bit.EQEP1ENCLK = 1;  // SYSCLKOUT to eQEP1 enabled
SysCtrlRegs.PCLKCR1.bit.ECAP6ENCLK = 1;  // SYSCLKOUT to eCAP6 enabled
SysCtrlRegs.PCLKCR1.bit.ECAP5ENCLK = 1;  // SYSCLKOUT to eCAP5 enabled
SysCtrlRegs.PCLKCR1.bit.ECAP4ENCLK = 1;  // SYSCLKOUT to eCAP4 enabled
SysCtrlRegs.PCLKCR1.bit.ECAP3ENCLK = 1;  // SYSCLKOUT to eCAP3 enabled
SysCtrlRegs.PCLKCR1.bit.ECAP2ENCLK = 1;  // SYSCLKOUT to eCAP2 enabled
SysCtrlRegs.PCLKCR1.bit.ECAP1ENCLK = 1;  // SYSCLKOUT to eCAP1 enabled
SysCtrlRegs.PCLKCR1.bit.EPWM6ENCLK = 1;  // SYSCLKOUT to ePWM6 enabled
SysCtrlRegs.PCLKCR1.bit.EPWM5ENCLK = 1;  // SYSCLKOUT to ePWM5 enabled
SysCtrlRegs.PCLKCR1.bit.EPWM4ENCLK = 1;  // SYSCLKOUT to ePWM4 enabled
SysCtrlRegs.PCLKCR1.bit.EPWM3ENCLK = 1;  // SYSCLKOUT to ePWM3 enabled
SysCtrlRegs.PCLKCR1.bit.EPWM2ENCLK = 1;  // SYSCLKOUT to ePWM2 enabled
SysCtrlRegs.PCLKCR1.bit.EPWM1ENCLK = 1;  // SYSCLKOUT to ePWM1 enabled
SysCtrlRegs.PCLKCR0.bit.ECANBENCLK = 1;  // SYSCLKOUT/2 to eCAN-B enabled
SysCtrlRegs.PCLKCR0.bit.ECANAENCLK = 1;  // SYSCLKOUT/2 to eCAN-A enabled
SysCtrlRegs.PCLKCR0.bit.MCBSPAENCLK = 1;  // LSPCLK to McBSP-A enabled
SysCtrlRegs.PCLKCR0.bit.MCBSPBENCLK = 1;  // LSPCLK to McBSP-B enabled
SysCtrlRegs.PCLKCR0.bit.SCIBENCLK = 1;  // LSPCLK to SCI-B enabled
SysCtrlRegs.PCLKCR0.bit.SCIAENCLK = 1;  // LSPCLK to SCI-A enabled
SysCtrlRegs.PCLKCR0.bit.SPIAENCLK = 1;  // LSPCLK to SPI-A enabled
SysCtrlRegs.PCLKCR0.bit.SCICENCLK = 1;  // LSPCLK to SCI-C enabled
SysCtrlRegs.PCLKCR0.bit.I2CAENCLK = 1;  // LSPCLK to I2C-A enabled
SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1;  // HSPCLK to ADC enabled

// The PCLKCR0.TBCLKSYNC bit is handled separately in InitEPwm() since
// it affects synchronization of the ePWM counters.

//--- Configure the low-power modes
SysCtrlRegs.LPMCR0.all = 0x00FC;  // LPMCR0 set to default value

//--- Finish up
asm(" EDIS");                   // Disable EALLOW protected register access

} // end InitSysCtrl()

//--- end of file --------------------------------------------------------------
#include "Project.h"     // Project Headerfile

void Test(int number);

void Test(int number)
{
    switch (number)
    {
    case 1:   // Test Movement
        // Movement(angle)  Used for any angle at full speed
        // Movement_1(angle, Power (0-127))  Used for 8 common angles at user specified speed
        Movement_1(0,64);  
        Delay(1);
        Stop(1);
        Stop(2);
        Stop(3);
        Stop(4);
        Delay(1);
        Movement_1(180,64);
        Delay(1);
        Stop(1);
        Stop(2);
Stop(3);
Stop(4);

Delay(1);

Movement_1(90, 64);
Delay(1);

Stop(1);
Stop(2);
Stop(3);
Stop(4);

Delay(1);

Movement_1(270, 64);
Delay(1);

Stop(1);
Stop(2);
Stop(3);
Stop(4);

Delay(1);

Movement_1(45, 64);
Delay(1);

Stop(1);
Stop(2);
Stop(3);
Stop(4);

Delay(1);

Movement_1(225, 64);
Delay(1);

Stop(1);
Stop(2);
Stop(3);
Stop(4);

Delay(1);
Movement_1(315,64);
Delay(1);
Stop(1);
Stop(2);
Stop(3);
Stop(4);
Delay(1);
Movement_1(135,64);
Delay(1);
Stop(1);
Stop(2);
Stop(3);
Stop(4);
Delay(1);
Rotate(1,64);
Delay(1);
Stop(1);
Stop(2);
Stop(3);
Stop(4);
Delay(1);
Rotate(2,64);
Delay(1);
Stop(1);
Stop(2);
Stop(3);
Stop(4);
Delay(1);
break;
case 2:       //Test Sensors and Kicker
Reed(1);
Reed(2);
Hall(1);
Hall(2);
Hall(3);
Hall(4);
Prox();
Kick();

break;

}
/*
 // TI File $Revision: /main/8 $ 
 // Checkin $Date: June 2, 2008 11:12:24 $
 //###########################################################################
 // FILE: DSP2833x_Headers_nonBIOS.cmd
 // TITLE: DSP2833x Peripheral registers linker command file
 // DESCRIPTION:
 // This file is for use in Non-BIOS applications.
 // Linker command file to place the peripheral structures used within the DSP2833x header files into the correct memory mapped locations.
 // This version of the file includes the PieVectorTable structure.
 // For BIOS applications, please use the DSP2833x_Headers_BIOS.cmd file which does not include the PieVectorTable structure.
 //###########################################################################
 // $TI Release: DSP2833x/DSP2823x Header Files V1.20 $ 
 // $Release Date: August 1, 2008 $
 //###########################################################################
 */

MEMORY
{
PAGE 0: /* Program Memory */

PAGE 1: /* Data Memory */

DEV_EMU : origin = 0x000080, length = 0x000180 /* device emulation registers */
FLASH_REGS : origin = 0x000A80, length = 0x000060 /* FLASH registers */
CSM : origin = 0x000AEO, length = 0x000010 /* code security module registers */

ADC_MIRROR : origin = 0x000B00, length = 0x000010 /* ADC Results register mirror */

XINTF : origin = 0x000B20, length = 0x000020 /* external interface registers */

CPU_TIMER0 : origin = 0x000C00, length = 0x000008 /* CPU Timer0 registers */
CPU_TIMER1 : origin = 0x000C08, length = 0x000008 /* CPU Timer0 registers (CPU Timer1 & Timer2 reserved TI use)*/
CPU_TIMER2 : origin = 0x000C10, length = 0x000008 /* CPU Timer0 registers (CPU Timer1 & Timer2 reserved TI use)*/
PIE_CTRL : origin = 0x000CE0, length = 0x000020 /* PIE control registers */
PIE_VECT : origin = 0x000D00, length = 0x000100 /* PIE Vector Table */
DMA : origin = 0x001000, length = 0x000200 /* DMA registers */
MCBSPA : origin = 0x000500, length = 0x000040 /* McBSP-A registers */
MCBSPB : origin = 0x000540, length = 0x000040 /* McBSP-B registers */

ECANA : origin = 0x0006000, length = 0x000040 /* eCAN-A control and status registers */
   ECANA_LAM : origin = 0x0006040, length = 0x000040 /* eCAN-A local acceptance masks */
   ECANA_MOTS : origin = 0x0006080, length = 0x000040 /* eCAN-A message object time stamps */
   ECANA_MOTO : origin = 0x00060C0, length = 0x000040 /* eCAN-A object time-out registers */
   ECANA_MBOX : origin = 0x0006100, length = 0x000100 /* eCAN-A mailboxes */

ECANB : origin = 0x0006200, length = 0x000040 /* eCAN-B control and status registers */
   ECANB_LAM : origin = 0x0006240, length = 0x000040 /* eCAN-B local acceptance masks */
   ECANB_MOTS : origin = 0x0006280, length = 0x000040 /* eCAN-B message object time stamps */
   ECANB_MOTO : origin = 0x00062C0, length = 0x000040 /* eCAN-B object time-out registers */
   ECANB_MBOX : origin = 0x0006300, length = 0x000100 /* eCAN-B mailboxes */

EPWM1 : origin = 0x0006800, length = 0x000022 /* Enhanced PWM 1 registers */
EPWM2 : origin = 0x0006840, length = 0x000022 /* Enhanced PWM 2 registers */
EPWM3 : origin = 0x0006880, length = 0x000022 /* Enhanced PWM 3 registers */
EPWM4 : origin = 0x00068C0, length = 0x000022 /* Enhanced PWM 4 registers */
EPWM5 : origin = 0x0006900, length = 0x000022 /* Enhanced PWM 5 registers */
EPWM6 : origin = 0x0006940, length = 0x000022 /* Enhanced PWM 6 registers */

ECAP1 : origin = 0x006A00, length = 0x000020 /* Enhanced Capture 1 registers */
ECAP2 : origin = 0x006A20, length = 0x000020 /* Enhanced Capture 2 registers */
ECAP3 : origin = 0x006A40, length = 0x000020 /* Enhanced Capture 3 registers */
ECAP4 : origin = 0x006A60, length = 0x000020 /* Enhanced Capture 4 registers */
ECAP5 : origin = 0x006A80, length = 0x000020 /* Enhanced Capture 5 registers */
ECAP6 : origin = 0x006AA0, length = 0x000020 /* Enhanced Capture 6 registers */

EQEP1 : origin = 0x006B00, length = 0x000040 /* Enhanced QEP 1 registers */
EQEP2 : origin = 0x006B40, length = 0x000040 /* Enhanced QEP 2 registers */
GPIOCTRL : origin = 0x006F80, length = 0x000040 /* GPIO control registers */
GPIODAT : origin = 0x006FC0, length = 0x000020 /* GPIO data registers */
GPIOINT : origin = 0x006FE0, length = 0x000020 /* GPIO interrupt/LPM registers */
SYSTEM : origin = 0x007010, length = 0x000020 /* System control registers */
SPIA : origin = 0x007040, length = 0x000010 /* SPI-A registers */
SCI A : origin = 0x007050, length = 0x000010 /* SCI-A registers */
XINTRUPT : origin = 0x007070, length = 0x000010 /* external interrupt registers */
ADC : origin = 0x007100, length = 0x000020 /* ADC registers */
SCIB : origin = 0x007750, length = 0x000010 /* SCI-B registers */
SCIC : origin = 0x007770, length = 0x000010 /* SCI-C registers */
I2CA : origin = 0x007900, length = 0x000040 /* I2C-A registers */

CSM_PWL : origin = 0x33FFF8, length = 0x000008 /* Part of FLASHA. CSM password locations. */

PARTID : origin = 0x380090, length = 0x000001 /* Part ID register location */

SECTIONS
{
PieVectTableFile : > PIE_VECT, PAGE = 1

/*** Peripheral Frame 0 Register Structures /***/
DevEmuRegsFile : > DEV_EMU, PAGE = 1
FlashRegsFile : > FLASH_REGS, PAGE = 1
CsmRegsFile : > CSM, PAGE = 1
AdcMirrorFile : > ADC_MIRROR, PAGE = 1
XintfRegsFile : > XINTF, PAGE = 1
CpuTimer0RegsFile : > CPU_TIMER0, PAGE = 1
CpuTimer1RegsFile : > CPU_TIMER1, PAGE = 1
CpuTimer2RegsFile : > CPU_TIMER2, PAGE = 1
PieCtrlRegsFile : > PIE_CTRL, PAGE = 1
DmaRegsFile : > DMA, PAGE = 1

/*** Peripheral Frame 3 Register Structures /***/
McbspaRegsFile : > MCBSPA, PAGE = 1
McbspbRegsFile : > MCBSPB, PAGE = 1

/*** Peripheral Frame 1 Register Structures /***/
ECanaRegsFile : > ECANA, PAGE = 1

162
<table>
<thead>
<tr>
<th>File Name</th>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECanaLAMRegsFile</td>
<td>ECANA_LAM</td>
<td>1</td>
</tr>
<tr>
<td>ECanaMboxesFile</td>
<td>ECANA_MBOX</td>
<td>1</td>
</tr>
<tr>
<td>ECanaMOTSRegsFile</td>
<td>ECANA_MOTS</td>
<td>1</td>
</tr>
<tr>
<td>ECanaMOTORegsFile</td>
<td>ECANA_MOTO</td>
<td>1</td>
</tr>
<tr>
<td>ECAnbRegsFile</td>
<td>ECANB</td>
<td>1</td>
</tr>
<tr>
<td>ECAnbLAMRegsFile</td>
<td>ECANB_LAM</td>
<td>1</td>
</tr>
<tr>
<td>ECAnbMboxesFile</td>
<td>ECANB_MBOX</td>
<td>1</td>
</tr>
<tr>
<td>ECAnbMOTSRegsFile</td>
<td>ECANB_MOTS</td>
<td>1</td>
</tr>
<tr>
<td>ECAnbMOTORegsFile</td>
<td>ECANB_MOTO</td>
<td>1</td>
</tr>
<tr>
<td>EPwm1RegsFile</td>
<td>EPWM1</td>
<td>1</td>
</tr>
<tr>
<td>EPwm2RegsFile</td>
<td>EPWM2</td>
<td>1</td>
</tr>
<tr>
<td>EPwm3RegsFile</td>
<td>EPWM3</td>
<td>1</td>
</tr>
<tr>
<td>EPwm4RegsFile</td>
<td>EPWM4</td>
<td>1</td>
</tr>
<tr>
<td>EPwm5RegsFile</td>
<td>EPWM5</td>
<td>1</td>
</tr>
<tr>
<td>EPwm6RegsFile</td>
<td>EPWM6</td>
<td>1</td>
</tr>
<tr>
<td>ECap1RegsFile</td>
<td>ECAP1</td>
<td>1</td>
</tr>
<tr>
<td>ECap2RegsFile</td>
<td>ECAP2</td>
<td>1</td>
</tr>
<tr>
<td>ECap3RegsFile</td>
<td>ECAP3</td>
<td>1</td>
</tr>
<tr>
<td>ECap4RegsFile</td>
<td>ECAP4</td>
<td>1</td>
</tr>
<tr>
<td>ECap5RegsFile</td>
<td>ECAP5</td>
<td>1</td>
</tr>
<tr>
<td>ECap6RegsFile</td>
<td>ECAP6</td>
<td>1</td>
</tr>
<tr>
<td>EQep1RegsFile</td>
<td>EQEP1</td>
<td>1</td>
</tr>
<tr>
<td>EQep2RegsFile</td>
<td>EQEP2</td>
<td>1</td>
</tr>
<tr>
<td>GpioCtrlRegsFile</td>
<td>GPIOCTRL</td>
<td>1</td>
</tr>
<tr>
<td>GpioDataRegsFile</td>
<td>GPIODAT</td>
<td>1</td>
</tr>
<tr>
<td>GpioIntRegsFile</td>
<td>GPIOINT</td>
<td>1</td>
</tr>
<tr>
<td>SysCtrlRegsFile</td>
<td>SYSTEM</td>
<td>1</td>
</tr>
<tr>
<td>SpiaRegsFile</td>
<td>SPIA</td>
<td>1</td>
</tr>
<tr>
<td>SciaRegsFile</td>
<td>SCIA</td>
<td>1</td>
</tr>
<tr>
<td>XIntruptRegsFile</td>
<td>XINTRUPT</td>
<td>1</td>
</tr>
<tr>
<td>AdcRegsFile</td>
<td>ADC</td>
<td>1</td>
</tr>
<tr>
<td>ScibRegsFile</td>
<td>SCIB</td>
<td>1</td>
</tr>
<tr>
<td>ScicRegsFile</td>
<td>SCIC</td>
<td>1</td>
</tr>
<tr>
<td>I2caRegsFile</td>
<td>I2CA</td>
<td>1</td>
</tr>
<tr>
<td>CsmPwlFile</td>
<td>CSM_PWL</td>
<td>1</td>
</tr>
</tbody>
</table>

/*** Peripheral Frame 2 Register Structures ***/

/*** Code Security Module Register Structures ***/

/*** Device Part ID Register Structures ***/
PartIdRegsFile : > PARTID, PAGE = 1

}

/*
//==================================
=====
// End of file.
//==================================
=====
*/
/**********************************************************************
* File: f28335_nonBIOS_flash.cmd -- Linker command file for non-DSP/BIOS
* code with DSP in Boot to Flash boot mode.
*
* History: 09/18/07 - original (D. Alter)
***********************************************************************/

MEMORY
{
    PAGE 0: /* Program Memory */
        BEGIN_M0 : origin = 0x000000, length = 0x000002     /* Part of M0SARAM. Used for
"Boot to M0" bootloader mode. */
        L012SARAM : origin = 0x008000, length = 0x004000     /* SARAM, L0 through L3
combined. CSM secure */
        FLASH_ABCDEFGH : origin = 0x300000, length = 0x03FF7F     /* On-chip FLASH */
        CSM_RSVD : origin = 0x33FF80, length = 0x000076     /* Part of FLASH Sector A.
Reserved when CSM is in use. */
        BEGIN_FLASH : origin = 0x33FFF6, length = 0x000002     /* Part of FLASH Sector A.
Used for "Jump to flash" bootloader mode. */
        PASSWORDS : origin = 0x33FFF8, length = 0x000008     /* Part of FLASH Sector A.
CSM password locations. */
        ADC_CAL : origin = 0x380080, length = 0x000009     /* ADC_cal function in Reserved
memory */
        OTP : origin = 0x380400, length = 0x000400     /* 1Kw OTP */
        IQTABLES : origin = 0x3FE000, length = 0x000B50     /* Part of Boot ROM */
        IQTABLES2 : origin = 0x3FEB50, length = 0x00008C     /* Part of Boot ROM */
        FPUTABLES : origin = 0x3FEBDC, length = 0x0006A0     /* Part of Boot ROM */
        BOOTROM : origin = 0x3FF27C, length = 0x000D44     /* 8Kw Boot ROM */
        RESET : origin = 0x3FFFC0, length = 0x000002     /* Part of Boot ROM */

    PAGE 1: /* Data Memory */
        M0SARAM : origin = 0x000002, length = 0x0003FE     /* 1Kw M0 SARAM */
        M1SARAM : origin = 0x000400, length = 0x000400     /* 1Kw M1 SARAM */
        L4SARAM : origin = 0x00C000, length = 0x001000     /* 4Kw L4 SARAM, DMA
accessible */
        L5SARAM : origin = 0x00D000, length = 0x001000     /* 4Kw L5 SARAM, DMA
accessible */
        L6SARAM : origin = 0x00E000, length = 0x001000     /* 4Kw L6 SARAM, DMA
accessible, 1 WS prog access */
        L7SARAM : origin = 0x00F000, length = 0x001000     /* 4Kw L7 SARAM, DMA
accessible 1 WS prog access */
}

SECTIONS
} 

/*** Compiler Required Sections ***/

/* Program memory (PAGE 0) sections */
.text : > FLASH_ABCDEFGH, PAGE = 0
.cinit : > FLASH_ABCDEFGH, PAGE = 0
.const : > FLASH_ABCDEFGH, PAGE = 0
.econst : > FLASH_ABCDEFGH, PAGE = 0
.pinit : > FLASH_ABCDEFGH, PAGE = 0
.reset : > RESET, PAGE = 0, TYPE = DSECT /* We are not using the .reset section */
.switch : > FLASH_ABCDEFGH, PAGE = 0

/* Data Memory (PAGE 1) sections */
.bss : > L4SARAM, PAGE = 1
.ebss : > L4SARAM, PAGE = 1
.cio : > L4SARAM, PAGE = 1
.stack : > M1SARAM, PAGE = 1
.sysmem : > L4SARAM, PAGE = 1
.esysmem : > L4SARAM, PAGE = 1

// Added
// ramfuncs : > L5SARAM, PAGE = 1
// END

/*** User Defined Sections ***/
codestart : > BEGIN_FLASH, PAGE = 0 /* Used by file CodeStartBranch.asm */
csm_rsvd : > CSM_RSVD, PAGE = 0 /* Used by file passwords.asm */
internalMemFuncs : > FLASH_ABCDEFGH, PAGE = 0 /* Used by file Xintf.c. */
Link to internal memory */
.passwords : > PASSWORDS, PAGE = 0 /* Used by file passwords.asm */

/* Section secureRamFuncs used by file SysCtrl.c. */
secureRamFuncs : LOAD = FLASH_ABCDEFGH, PAGE = 0 /* Should be Flash */

RUN = L0123SARAM, PAGE = 0 /* Must be CSM secured RAM */

LOAD_START(_secureRamFuncs_loadstart),
LOAD_END(_secureRamFuncs_loadend),
RUN_START(_secureRamFuncs_runstart)

}

/******************* end of file ************************/

166
#ifndef Kick_H
#define Kick_H

extern void Kick(void);

#endif // end
#ifndef Movement_H
#define Movement_H

extern void Movement(int phi);
extern void Stop(int motor);
extern void Rotate(int rot, int spd);

// Test Cases

extern void Movement_1(int phi, int pwr);
extern void Movement_2(int phi, int dis, int pwr);

#endif // end
// FILE:   PIN_setup.h
// TITLE:  PIN Headerfile

#ifndef PIN_setup_H
#define PIN_setup_H

extern void PIN_setup(void);

#endif  // end

#ifndef PROJECT_H
#define PROJECT_H

#include "DSP2833x_Device.h"     // DSP2833x Headerfile Include File
#include "Movement.h"           //Movement function Headerfile
#include "Serial.h"             //Serial Headerfile
#include "Sensors.h"            //Sensor Headerfile
#include "Kick.h"               //Kick Headerfile
#include "PIN_setup.h"          // PIN Headerfile
#include <string.h>
#include "DSP2833x_DefaultIsr.h"  // ISR definitions

extern void DelayUs(Uint16);
extern void Delay(int number);
extern void InitFlash(void);
extern void InitPieCtrl(void);
extern void InitSysCtrl(void);

extern Uint16 secureRamFuncs_loadstart;
extern Uint16 secureRamFuncs_loadend;
extern Uint16 secureRamFuncs_runstart;

extern const struct PIE_VECT_TABLE PieVectTableInit; // Pie vector table (non-BIOS only)

#endif // end of DSP28x_PROJECT_H definition
#ifndef Sensors_H
#define Sensors_H

extern int Reed(int number);
extern int Prox(void);
extern int Hall(int number);

#endif // end
//###########################################################################
// FILE:   Serial.h
// TITLE:  Serial Headerfile
//###########################################################################
// April 12, 2009
//###########################################################################

#ifndef Serial_H
#define Serial_H

extern void Scic_init(void);
extern void Scic_xmit(char mout);

extern void Scia_init(void);
extern void Scia_xmit(char out);

#endif  // end

#endif  // end