CALVIN COLLEGE

TEAM 5: STORBOT

Engineering 340: Senior Design

Matthew Lubbers, Ryan Mejeur, David VandeBunte, David VanKampen

5/18/2009

STORBOT™

EXECUTIVE SUMMARY

Standard factories today lose many man-hours to warehouse management tasks. Employees leave their positions on the assembly line to go retrieve a small box of parts; they must contact a warehouse employee to help them find what they need; and when they finally get what they want, they have to make sure they are properly updating the parts database with what they have taken.

This series of simple tasks is an obvious opportunity for automation, and robots are making their way into warehouses to perform tasks that are too simple for human employees. A related problem faced in small warehouses and small stores is the cost barrier to put up one of these automated solutions. Companies are still striving to make their processes more streamlined and efficient, but there are few feasible low-cost solutions on the market.

These systems, generally referred to as “Automatic Storage and Retrieval Systems” (AS/RS), are making their way into warehouses in an incredible push for increased robot autonomy. STORBOT (Storage, Transfer, Organization, and Retrieval Robot) is a prototype that demonstrates the potential for a reliable, inexpensive, and practical solution to this problem.

The team met these design objectives during the course of the project:

1) Used an FPGA running the open-source GNU/Linux operating system for the robotic control unit.
2) Developed an I/O address port to allow CPU access to motors through addressing
3) Designed a line-following scheme to replace the typical AS/RS track system.
4) Designed a mobile base with wheels powered by electric stepper motors.
    a. Base is capable of moving weight of entire device and load.
5) Design an arm lift and extension modules
6) Must be capable of lifting specified load weight of 5 pounds (up to 30 lbs.).
7) Design a gripper unit with pressure sensitivity or optical alignment capabilities (vacuum sensor to test pressure, vacuum motor to apply pressure)
8) Design a Graphical User Interface (GUI) for sending commands to STORBOT.

The included report summarizes the design decisions, design patterns, and work plan completed for the project.

TABLE OF CONTENTS

TABLE OF TABLES

TABLE OF FIGURES

INTRODUCTION

According to the Bureau of Economic Analysis (BEA), the warehousing and storage industry grossed 48.6 billion dollars in 2006, accounting for approximately 0.92% of the United States' total gross domestic product (GDP*).* Improvements are constantly being made in the warehousing and storage industry to streamline processes, increase efficiency, and maximize profitability. These improvements often involve automated equipment from high-speed pallet shrink wrappers to autonomous forklifts. Our design is aimed at providing a further level of automation to the warehouse industry, especially small and medium sized warehouses.

Today's warehouses require sophisticated control systems due to the enormous size of a modern warehouse and the volume of products moving through it. Our design attempts to solve the problem of controlling goods while increasing the efficiency of warehouses by providing an inventory database along with a robotic retrieval and storage system. In principle, the system will be controlled by a host computer that will interface with both the robot and a goods control server.

The host computer would be password protected and physically accessible to limited personnel. The host computer will track items coming in and out of the warehouse, as well as send updates to a goods control server as needed. The host computer will communicate with the retrieval robot through a wireless standard to request that goods be moved, retrieved, or stored in the warehouse. On the robot, a microcontroller or custom logic configuration (referred to here as the controller) will handle all requests from the host computer, but will be able to operate independently of the host computer while executing a command.

The robotic system will be directed by the controller through an amplification stage, which amplifies the power and control signals to operate the robot and its arm movement. The robot will be able to store and retrieve goods by using a three axis arm connected to a mobile base, which moves to the desired item location in the warehouse using a removable tape or color system. The robotic arm may include servo motors, DC motors, and pneumatics. The robotic retrieval system will be designed for small items weighing up to fifty pounds, such as: plastic parts, screw boxes, nail boxes, gaskets, small motors, etc. The complete system design will be flexible, with the goal of easy installation into existing warehouses.

The team's project was to design such a prototype robotic system while staying in a $1500 budget. The functional prototype is capable of being fully incorporated into a functioning warehouse system and demonstrates a reasonable level of flexibility, providing a platform for future development.

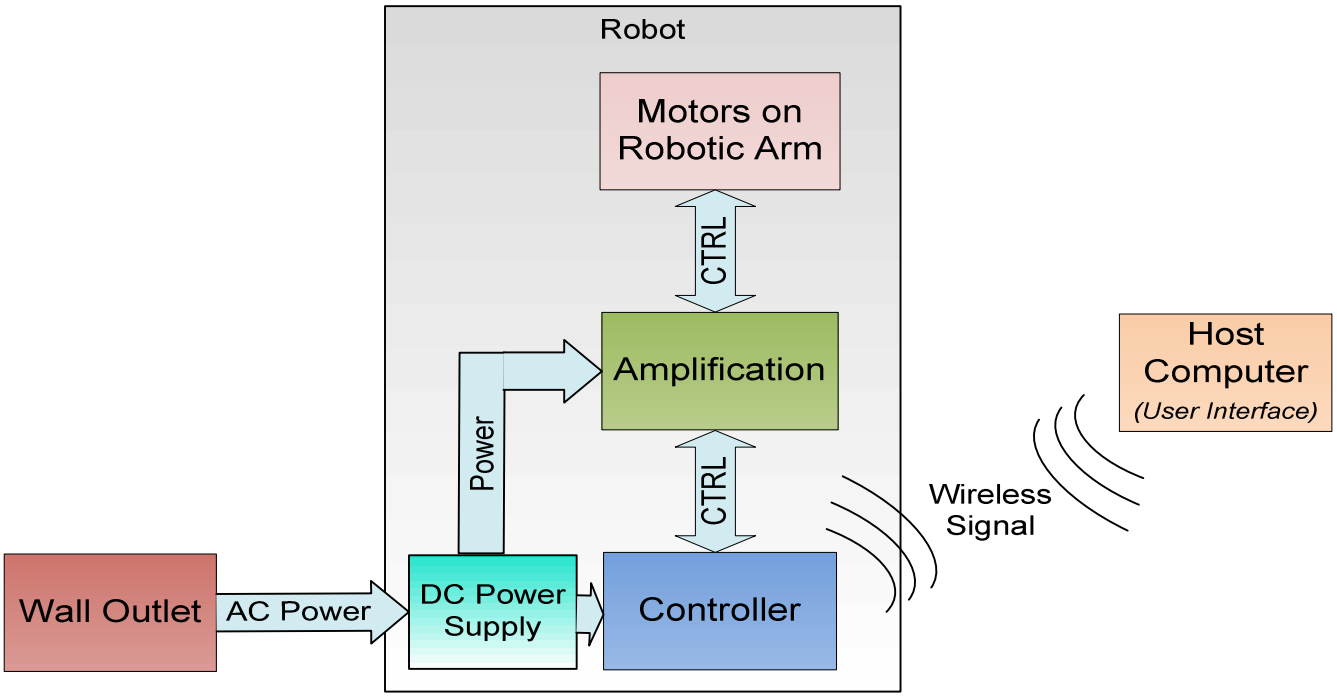


Figure 1: System Block Diagram

An associated goal was to also design a user friendly system that is intuitive to operators running and maintaining it. The team made every effort to adhere to the codes and regulations of governing safety agencies, particularly by implementing an accessible power cut (i.e. emergency stop) and equipping the system with sensors to reduce risk to personnel operating the equipment. The remainder of this report describes the technical and functional aspects of the design, as well as the feasibility of the project from an economic and technical perspective.

Acronyms and Terms

Table 1: Acronyms

ADC	Analog to Digital Converter
ASIC	Application Specific Integrated Circuit
ASRS	Automated Storage and Retrieval System
CISC	Complex Instruction Set Computer
FPGA	Field Programmable Gate Array
Ft-lb	Foot pound
GDP	Gross Domestic Product
GNU	GNUs Not Unix
GPL	General Public License
GUI	Graphical User Interface
In-lb	Inch pound
ISA	Instruction Set Architecture
Nm	Newton Meter
OS	Operating System

PCB	Printed Circuit Board
PPC	Power PC
PPFS	Project Proposal and Feasibility Study
RISC	Reduced Instruction Set Computer
TBD	To Be Determined/ Done
UART	Universal Asynchronous Receiver Transmitter
VHDL	VHSIC Hardware Description Language

Table 2: Terms

DC Motor	Motor that forces electric current through a coil to produce mechanical torque.
Firmware	A computer program embedded in hardware
Hardware	The physical components of a computer or robotics system
Operating System	System Responsible for the management and coordination of activities for computer resource sharing
Servo Motor	Motor with error feedback for precise position control, incapable of multiple full rotations.
Software	Computer programs and procedures that perform tasks on computer systems
Stepper Motor	Motor with multiple “toothed” electromagnets that allow position encoding without a feedback mechanism

TEAM MEMBERS

INTRODUCTION

MATTHEW LUBBERS

Matt Lubbers is from St. Joseph, Michigan; he is pursuing a major in Electrical Engineering, and is already Automotive Service Excellence Certified. Matt has worked in and is interested in electrical applications in the automotive industry. He has accepted a position at Cook Nuclear in Bridgman, MI following graduation, and will also be pursuing a graduate degree at Western Michigan.

RYAN MEJEUR

Ryan Mejeur is from Kalamazoo, Michigan; he is working on a major in Electrical Engineering and a minor in Biochemistry. Ryan is interested in pharmaceutical process machinery and is going into the US Army National Guard ROTC after graduation. He is interested in both the mechanical and electrical sides of engineering problems.

DAVID VANDEBUNTE

David VandeBunte is from Grand Rapids, Michigan; he is pursuing a major in Electrical Engineering and a minor in Computer Science and Optics, and expects to graduate with honors. Dave is interested in computer hardware design and embedded systems; he currently works at DornerWorks Ltd and has accepted a position there after graduation.

DAVID VANKAMPEN

David VanKampen is from Portage, Michigan; he is working on a major in Electrical Engineering. Dave is interested in embedded systems and hardware design and currently works at DornerWorks Ltd. He is enaged to be married to fellow engineering student Becca Sheler in October 2009, and will also be attending graduate school at Grand Valley State University after graduation, continuing study in electrical and computer engineering.

PROJECT ROLES

MATT LUBBERS

Matt worked diligently throughout both semesters making sure the mechanical work stayed on track. He devoted most of his time second semester to the elevator design, making it stronger and more reliable.

RYAN MEJEUR

Ryan spearheaded the mechanical design of the base as well as the rotator unit for the top of the robot. This task involved the physical layout and arrangement of all components, mechanical strength tests, and calibration testing.

DAVID VANDEBUNTE

David VandeBunte spent the large majority of his time refining the digital system. From an early point in the project, he prepared the hardware and software in parallel with mechanical work to be ready to all come together in the end. He became familiar with the Xilinx tool suite: Base System Builder, Xilinx Platform Studio, and the Xilinx EDK Shell. All of these tools were used to setup the peripheral hardware around the PowerPC processor, written in VHDL.

He also wrote a large part of the software used in the final design. Although he was away for spring break, he spent his time reading about Linux Device Drivers, which were used in the final design.

DAVID VANKAMPEN

David VanKampen served as the team leader and glue that held the project together. He made sure the team met all required class deadlines, and kept the website up to date as well.

The main role Dave filled, however, was preparing and integrating STORBOT's sensors. The robot needed to be able to sense its environment to operate properly. David researched viable options, put in orders for sensors, wrote VHDL for the on the Altera board, and helped David VandeBunte add them into the final PowerPC system. At the end of the project, he performed a good deal of the wiring and debugging for the board in general, and the wheel controllers specifically.

PROJECT DESCRIPTION

STORBOT is composed of four electrical engineering students, though the projects mechanical nature might suggest otherwise. The robot unit designed by Team 5 is intended to *S*tore, *T*ransfer, *O*rganize, and *R*etrieve stocked items in a factory or warehouse. In many warehouses, employees with the lowest pay rates are often assigned the job of retrieving items from the warehouse. Warehouse "gophers", as they are called, waste significant amounts of time running around the warehouse retrieving light items. Team 5 believes this task can be automated, thus saving factories time and money by reducing wasted time and effort.

The STORBOT robot was designed and constructed from the ground up by the members of Team 5. The control system was developed on a PowerPC processor using the open source Linux operating system. Several different areas of engineering design found their way into this project. Due to the robot being constructed from scratch, additional knowledge of mechanical systems was required. Digital systems programming and design was also prevalent because precise motor control involves both hardware and software elements. Feedback loops and sensors play a big role in giving the robot autonomy, presenting the team with many control system challenges.

Team 5 also took the time to use their connection with the Computer Science Department to recruit a CS student as a consultant assisting the development of the Graphical User Interface, as well as a serial port driver, for allowing a host PC to communicate with the robot.

PROJECT REQUIREMENTS

The team had several goals they wished to achieve through the duration of this senior design project. They are listed as follows. These are the basic requirements that, if met, would allow the team to consider the project a success. Other outcomes can be achieved, but they are extemporaneous to the following list.

1. Use a free, open source VHDL processor core to serve as a robotic control unit

2. Develop an I/O address port to allow CPU access to motors through addressing

3. Design a track system

4. Design a moveable base with wheels powered by electric motors capable of moving weight of entire device and load

5. Design an arm lift and extension modules

    a. Must be capable of lifting specified load weight of 15 pounds

    b. Must be capable of carrying said weight away from its place on shelf

6. Design a gripper unit with pressure sensitivity or optical alignment capabilities

7. Design a Graphical User Interface for sending commands and monitoring performance

8. If time allows, incorporate a Database Server to monitor the stock and location of items

DISCUSSION

PATENT SEARCH

In the process of building the prototype, team members kept in mind the possibility of getting a patent on components of the design. The team maintained a project patent notebook to document which parts of the robot might be most patentable.

As part of the patent search, the team met with Mike Harris, the director of the Enterprise Center, which was created as a department at Calvin College to protect professor's and student's intellectual property rights. The center also provides advice in associated legal areas, particularly for students or professors considering patents. Mike Harris enjoyed seeing the robot run and is looking into how Calvin can help in the patent process. Dave VanKampen will be working with Mike on any further patent work.

MARKET STUDY

The team performed a full business plan for a school competition, available at the project website. Three companies stood out as competition within STORBOT's target market.

Most of the existing warehousing automation companies today develop large scale products. The focus is on having warehouses run autonomously from the ground up, with large palletizing robots that can handle near car-size loads. However, these systems cost millions of dollars to install and run, need significant space to maneuver, and result in long factory downtime during installation. However, any of these large businesses could reduce the scale of their current products if they believed the small-scale market was growing. The matrix shown in Table 3 outlines the current competition in the small scale market only.

The strength of the existing competition lies within their experience and existing customer base. They also have insight into the difficulties involved in bringing a robot from concept to reality, and from prototype to the market.

However, the weaknesses of the existing competition suggest potential for companies focused on robots similar to the team's prototype. Few companies offer a low cost and quick turnkey solution for small-scale automation. Many of the current companies require ground-up integration, requiring time and money commitments most companies are unable to make.

Table 3: Competitive Matrix

FACTOR	Automated Warehouse Solutions, Inc.	FATA Automation	Kiva Robots (Amazon)	Westfalia
Low Price	5	1	3	3
Superior Quality	5	4	4	4
Customizable Product	5	3	2	4
Unique Features	4	5	4	4
Rapid Product Delivery	4	3	3	4
Customer Service	5	2	4	3
Total	28	18	20	22

CHRISTIAN PERSPECTIVE – DESIGN NORMS

Design norms look at the project's design from a moral, rather than physical and technical, perspective. A design norms study looks at how this project will affect society and the environment in which the system will operate. This team is dedicated to creating a finished product that is morally and ethically edifying to those using it and those affected by it. According to Professors Gayle Ermer and Steven VanderLeest, "Normative design attempts to balance design trade-offs not only among technical constraints but also among ethical constraints. Designing to such norms forces the engineer to consider the broader impact of the design on the society in which it will be embedded. Design norms include concepts such as cultural appropriateness, transparency, stewardship, integrity, justice, and caring [TBD: Reference]." This design team is also committed to making the robotic warehouse system culturally appropriate.

TRANSPARENCY

This design of the robot will be as transparent as possible. We want the robotic warehouse system to be aesthetically pleasing to those operating and working around the equipment. The system will be reliable so that those who operate the equipment will be effective and efficient. We would like the equipment to operate in the background and not require excessive movement or work by the robotic machinery. Our design will fully consider stewardship of the environment and humanity.

CARING

This system will make the warehouse storage industry less physically demanding. If warehouse jobs become less physically demanding, job applicants with physical limitations will be able to contribute in more warehouse operations. We would also like the system to be caring to those who aren't technically literate, by making the GUI as user friendly as possible. Lastly, we believe that the design won't cut any jobs, but rather supplement employees already working in the warehouse, allowing them to work in other ways (equipment technicians or maintenance personnel).

STEWARDSHIP

We will try to ensure the design is appropriate for warehouse applications and does not interfere with the culture it is implemented in. This minimal impact will be accomplished by creating a robot that is aesthetically pleasing and does not interfere with the warehouse personnel's activities. Although we will not have enough time during the senior design to create multiple language features in the GUI, we would like those developing the system beyond senior design to create multi-language capabilities for the host computer interface. The system will be user friendly and easy for warehouse operators to learn. The design will be efficient, allowing personnel to operate independently from the system which should optimize their time.

The design will be energy efficient and made from recyclable materials to minimize the impact on the environment. The power consumption is a concern because of the cost impact and more importantly the carbon emission impact that the system will have on the global environment. The system will also have a small footprint to optimize space; this will allow warehouse owners to be better stewards of the land that they own. Also, by creating a more efficient inventory

system, we hope that the loss of inventory due to expiration and misplacement will be reduced so the industry will produce less waste.

INTEGRITY

In this project we hope to maintain integrity and promote integrity for those who will use our product. Hundreds of millions of dollars are stolen each year in merchandise by those working in warehouses and stores. This system hopes to reduce that high dollar loss by creating electronic accountability for the employees. We will accomplish this through an electronic inventory database that will track items coming to and from the warehouses. Lastly, we hope to maintain integrity with the design by not eliminating jobs, but rather make the people more efficient at their jobs. We hope the design will be caring to all who use the equipment.

INITIAL PROTOTYPE

An initial prototype was designed and assembled during the month of January in order to perform initial tests and investigate the feasibility of the overall design. While the prototype tests proved the feasibility of the overall design, it uncovered several problems that would need to be addressed by a modified design.

Design issues learned from the prototype:

- Height travel not large enough to reach multiple levels of shelves.
- Guide rods design only able to be supported on the top and bottom of the tower, creating a lot of chatter due to the flexing of the guide rods.
- Guide rods had no linear bearing or bushing and therefore created metal to metal contact when moving up and down. While this approach worked acceptably for the prototype, it created a lot of unnecessary friction and resistance in the system, and was very prone to wearing.
- The side and top arm bearings were machined with intentional slots to allow for adjustability, however they soon became a nuisance due to unwanted movement .
- Arm stepper motor was programmed to move by "guessing" the optimal stepping rate, which caused a very jerky and inefficient way of driving the motor.
- Arm motor was unable to generate enough torque to pull a box of 5 lbs effectively.
- Arm motor seized to work consistently.

Replacement arm motor with sufficient torque is much larger and must be mounted on the base of the elevator assembly.

Slide extensions for shelves were located far from the center of support and began to sag due to weight distribution.

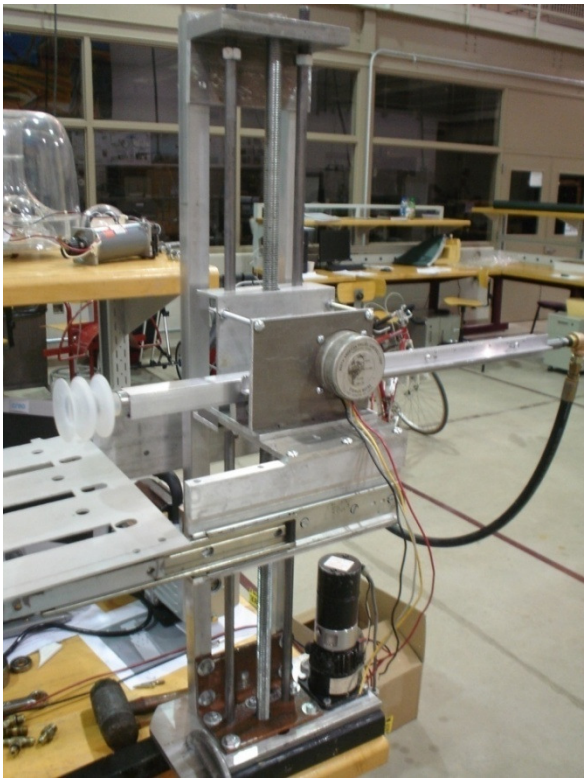


Figure 2: Initial Elevator

-
- Guide rods and support rods to close together to create adequate support for the load.
- Could not have cross-member support between guide rod support bars due to interference issues.
- Rotating base was not big enough to provide stability on the base for such a tall and heavy elevator system.

After the construction and testing of the prototype, the final elevator assembly began to be designed and constructed while keeping the issues of the prototype close at hand.

One of the major issues of the prototype was that the guide rods were merely steel rods with a clearance hole in the metal to allow “sliding” of the elevator. This caused a large amount of chatter, resistance, and shaking of the elevator as it moved up and down. To address this issue, linear guide rods and 4 linear bearing blocks were used to allow the elevator to glide smoothly up and down. By separating the guide rods and placing the majority of the weight near their center point, the load has a much more even distribution. Also, by cutting clearance sections in the lower plate of the elevator, cross-members can attach the guide rods to the support rods throughout the entire height of the tower. This keeps the guide rods from bending, and therefore provides a smooth raising and lowering of the elevator assembly. Additionally, the linear guide rods are 50” in height, and therefore the height in which the robot can pick parts is almost double.

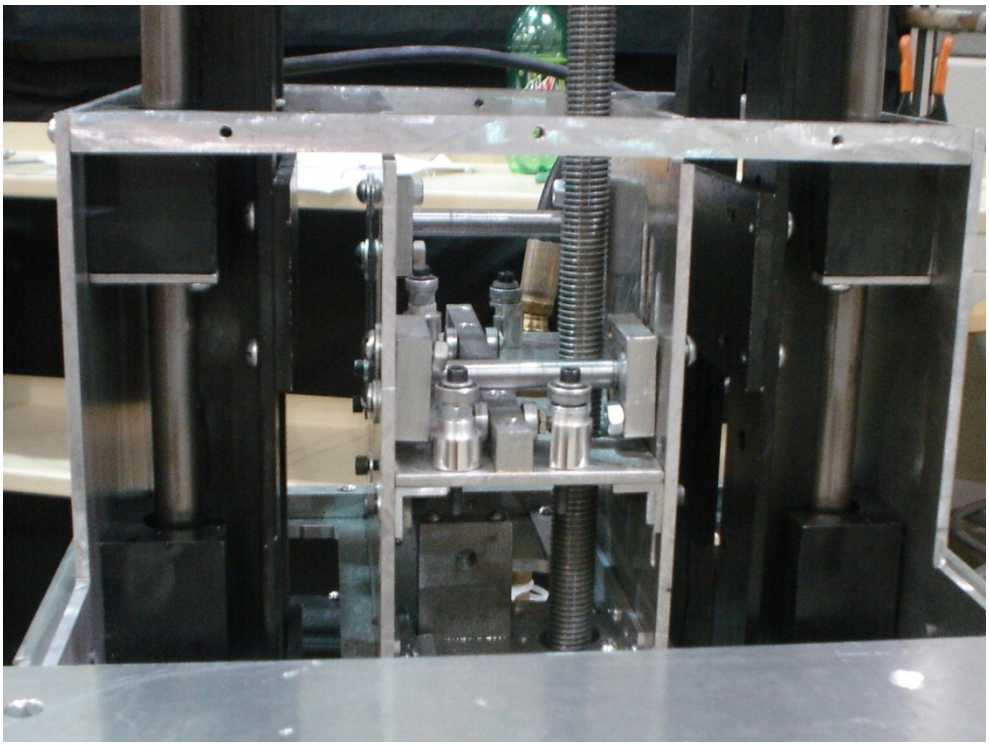


Figure 3: Arm Rack Assembly

Another problem with the initial prototype was that bearings which supported the arm were slotted and were prone to come loose and become sloppy. The bearings on the final design

MECHANICAL HARDWARE SYSTEM DESIGN

ELEVATOR DESIGN

The elevator system includes the elevator, arm, shelf, and tower assemblies. The requirement of the elevator assembly is to travel up and down the tower system, while the arm and shelf assemblies extend and retract, allowing for products to be retrieved from the shelf.

PROTOTYPE ISSUES ADDRESSED

While the prototype proved to be useful for specific tests and affirm the feasibility of the design, there were many issues that needed to be addressed. Therefore, after the construction and testing of the prototype, the final elevator assembly began to be designed and constructed to overcome these obstacles and achieve the overall goals of the project.

ELEVATOR ASSEMBLY

One of the biggest issues of the prototype was that the guide rods were merely steel rods with clearance holes in the metal to allow the elevator assembly to sliding up and down. This resulted in a large amount of chatter, resistance, and shaking of the elevator as it moved up and down. To address this issue, linear guide rods and 4 linear bearing blocks were used to allow the elevator to glide smoothly up and down. An example of a linear bearing and block can be seen to the right in Figure 4. A linear bearing uses a sleeve comprised of small ball bearings to allow fluid motion along a cylindrical surface. This linear bearing is

Figure 4: Linear Bearing Block

then held in a bearing block, which allows the bearing to be mounted to a flat surface, and traps the bearing from moving.

Another issue with the design of the prototype was that the guide rods were too close to each other to support an adequate load. The final elevator design was therefore completely redesigned to separate the linear bearings in both the horizontal and vertical direction. The center of the load on the product plate is centered in the middle of the four linear bearings to provide further stability and a more even weight distribution. Furthermore, the shelves are mounted to the sides of the elevator assembly, and the resulting load bearing of the product is near the center of the assembly, eliminating the sag due to uneven distribution which caused problems in the prototype.

Another problem with the initial prototype was that the bearings which supported the arm were slotted and were prone to come loose and become sloppy. The bearings on the final design have been calculated for the right amount of tolerance to allow a tight fit while allowing the arm to move without much drag resistance. Once the right tolerance values were determined, the holes for the bearings on the final design could be drilled and tapped in the correct location, eliminating the problem of them coming loose and becoming sloppy.

ELEVATOR TOWER

The initial prototype was built with the intention of being a small scale design which would aid in tests and feasibility of design. For this reason, the elevator tower measured about 30 inches in height, allowing for approximately 20 inches of travel. This allowed for an adequate prototype for testing, however it did not allow for elevation to multiple levels of product shelves. This problem was addressed by increasing the elevator tower to 52 ½ " in height, which resulted in an increase in elevator travel.

Another major concern with the prototype tower was that due to the lack of bearings or bushings between the guide rods and elevator, the guide rods could only be supported on the top and bottom. This lack of support in the middle of the guide rods created a lot of flex in the rods themselves when the elevator assembly traveled up and down the tower. This flex resulted in unnecessary resistance and chatter, and an overall unstable system. Since the linear bearings are formed in the shape of a 'U' (see Figure 4), it allows for a supporting piece directly mounted to the back of the guide rod. This prevents flex throughout the entire range of motion of the elevator assembly.

A similar problem with the prototype was that the guide rods could only be connected to the guide rod support bars at the top and bottom of the tower. To overcome this issue and provide a great deal of stability, clearance sections were cut in the bottom plate of the elevator assembly to allow for cross-members to connect the guide rods and the support bars along the entire height of the tower.

ARM

The initial arm motor used to power the rack and pinion was a small stepper motor. To power the stepper motor, a VHDL program was written to pulse the individual windings in the correct sequence. However, because the motor was found in storage and provided no specific datasheet or information on the amount of pulse steps per second, it was very hard to determine the optimal stepping rate. The resulting arm operation was jerky and inefficient.

In addition, the arm motor was unable to generate enough torque to pull a 5 lb box effectively. Although the use of a stepper motor controller may have allowed the motor to achieve an acceptable amount of torque, stepper controllers tend to be very expensive. Also, the stepping motor was designed to have a high amount of precision and relatively low torque with high speed. In the application of the final design, moderate accuracy, lower speed, and high torque were needed. For these reasons it was decided to purchase a geared DC motor with the correct speed and torque ratings.

While the new arm motor proved to work exceptionally well, it was much longer in physical length than the previous stepping motor, and therefore must be relocated from the side of the assembly to the bottom.

FINAL DESIGN

ELEVATOR ASSEMBLY

As mentioned previously, to achieve the goal of a 10 lb box and evenly distribute the load of the product, the linear guide bearings must be separated over a large distance, with the center of the load near the center of support. In addition, the original goal box was to be 12x12x12 inches in dimension. Therefore, the product plate was designed to be 14x14 inches to accommodate this objective. The shelves were then mounted on the outer sides of the assembly, while the linear bearings were mounted on the inner portion of the sides. Similarly, the elevator assembly height was determined by allowing for as much

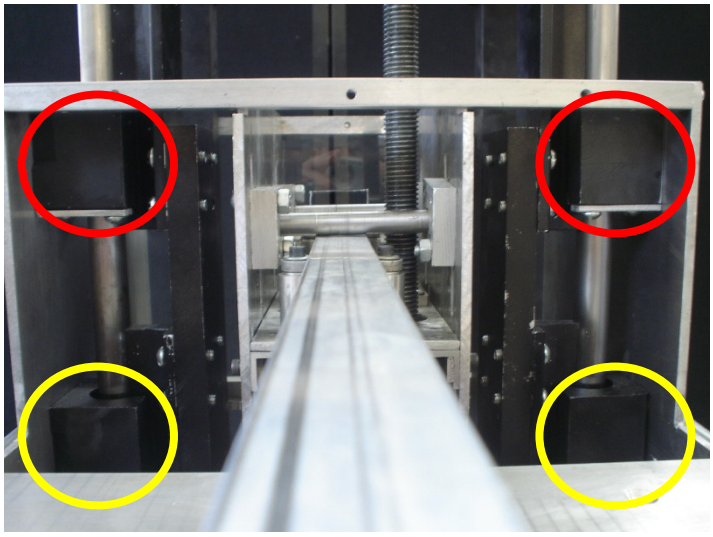


Figure 5: Linear Bearing Locations

vertical separation of the upper and lower linear bearings without making the assembly so tall that it greatly reduced the amount of travel along the elevator tower. The linear bearing locations can be seen above in Figure 5. The upper linear bearings are indicated by red circles, while the lower bearings are indicated in yellow.

Due to the product plate height in relation to the elevator, the arm height must be approximately in the middle of the elevator assembly. To do this, a suspended middle level was created to create the correct arm level height and to house the arm and bearings. The middle level is attached to the inner elevator vertical support plates through angle brackets. A picture of the middle level can be seen in Figure 6.

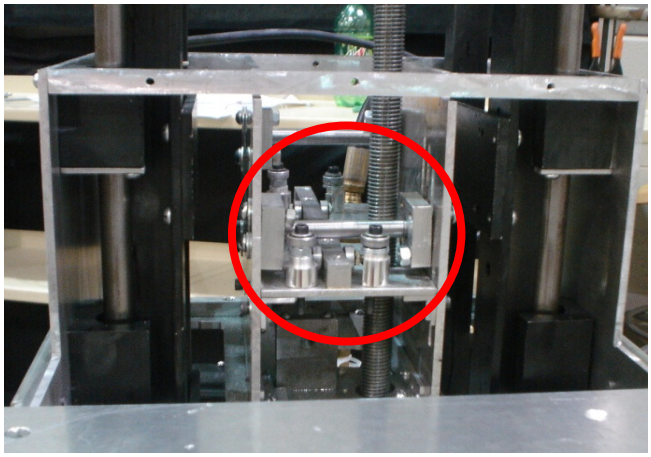


Figure 6: Middle Arm Supporting Layer

The spacing between the bottom of the suction cup attached to the arm and the top of the product plate is critical in order to achieve functionality. Due to the current inaccurate method of determining shelf heights, the spacing between the suction cup and product plate must be relatively large in order to allow for a large range of error. To allow for a large amount of error

and eventual calibration of the arm height, slots were machined in the inner elevator support plates to allow for adjustment in arm height. These slots also allow the use of the larger suction cup, which would allow for more pulling torque of the arm.

The elevator assembly is powered through the use of a 24V DC motor attached to a ¾"-10 threaded rod through timing gears and gearbelt. Figure 7 below shows the linkage between the elevator and threaded rod .When the final prototype was initially constructed the gearing ratio between the elevator motor and threaded rod was nearly 1:1. However, after the monitored current draw of the motor reached nearly 7 amps while under load, it was decided to purchase a larger gear for the rod. While this reduced the speed of elevation, it greatly increased the torque generated for the rod, which resulted in a lower current draw from the motor and an increase in load capacity.

Figure 7: Elevator Motor and Rod Connection

The final gear ratio between the motor and rod was 1:2.25, with a motor current draw under full load of 4 amps. The continuous current recommendation for the elevator motor is 5.3 amps, and therefore the decision to purchase a larger gear was made based on safety precautions of the motor.

By powering the elevator motor, the threaded rod spins, allowing the elevator assembly to move up and down through the use of a tapped block of steel which is attached to the bottom of the assembly. As the rod spins, the threads push up on the assembly or pull the assembly down, depending on the intended direction of motion. Figure 8 below shows a picture of

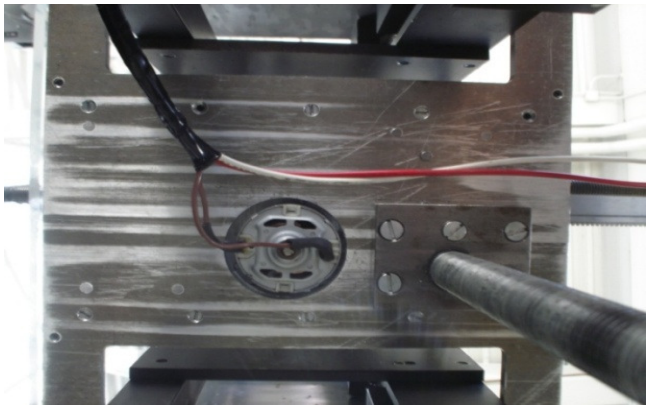


Figure 8: Bottom of Elevator Assembly

the threaded rod and tapped block located on the bottom of the elevator assembly.

ARM

The arm is powered through the use of a 24V, 2.2 amp geared motor operating at a speed of 75 RPM. A pinion gear is attached to the shaft of the motor, and when powered, it drives the arm forward or backward through the use of a rack attached to the arm. The rack was machined in house through the use of the mill in the machine shop. It was a very tedious process, taking about 6 hours to machine, however it works exceptionally

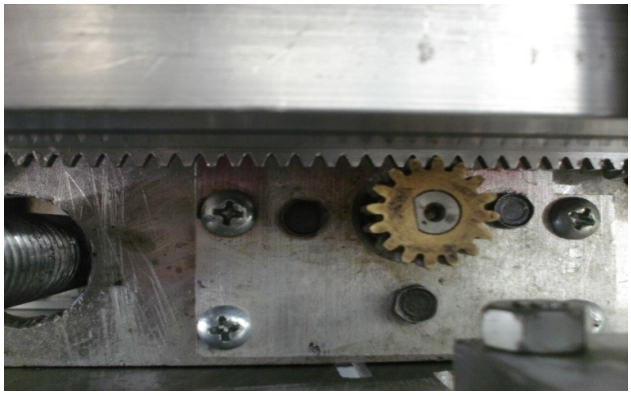


Figure 9: Arm Rack and Pinion

well and was well worth the invested time. The circumference of the pinion gear is approximately 1 inch, meaning the arm will extend and retract at a speed of about 75 inches per minute, or 1.25 inches per second. This motor was selected due to its appropriate speed and torque capabilities. The rack and pinion system can be seen in Figure 9 above.

To insure that the arm is prevented from moving when extended or under load, several bearings are required to keep it in the correct location. A top down view of the elevator middle section housing the arm, bearings, and motor can be seen to the right in Figure 11. Bearings to prohibit side to side movement are located at the front and back of the plate, and are circled in red. These

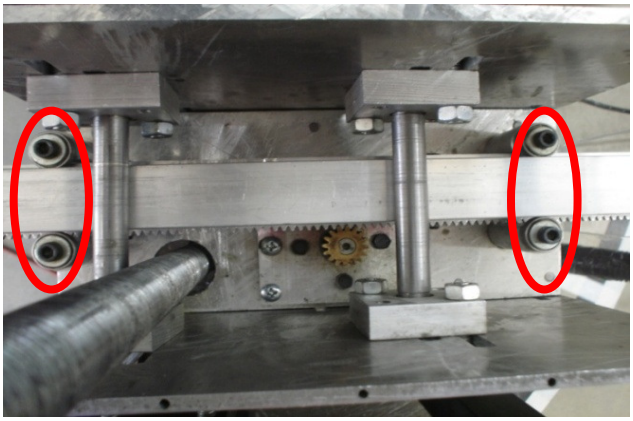


Figure 10: Arm Bearings and Housing

bearings are located as far apart as possible in order to create horizontal

stability of the arm. In addition to the side bearings, two rollers are located on the top of the arm to provide vertical stability. Since the arm and middle level height are adjustable, the top rollers must also be adjustable to roll along the top of the arm at every height. Therefore, slots for

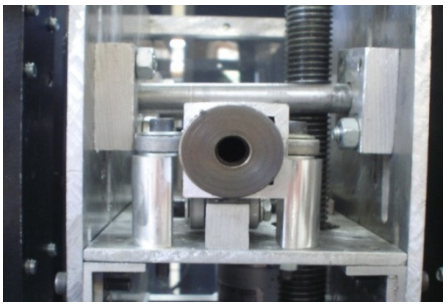


Figure 11: Cup Housing

the roller bearing blocks are machined into the

inner elevator support plates to allow for the rollers to be set to the same range of

heights as the arm. Finally, two bearings are mounted on the bottom of the arm in both the front and back of the plate to allow the arm to roll smoothly along the plate. These bottom bearings also aid in the vertical support of the arm. The side, top, and bottom bearings can be seen in Figure 11 above.

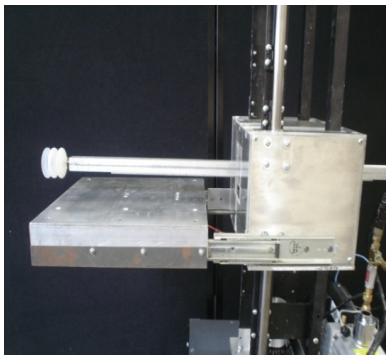


Figure 12: Extended Arm Position

The arm is 34” in length, with 25” inches of extension. It takes approximately 8 seconds for the arm to travel from the fully retracted position to the fully extended position. When the arm is fully extended, the arm is approximately 16” past the edge of the wheelbase. This allows for plenty of extension to retrieve small objects, objects deep within the shelf, or even multiple items placed within the depth of the shelf. Figure 13 above shows the arm fully retracted, while Figure 12 shows the arm fully extended.

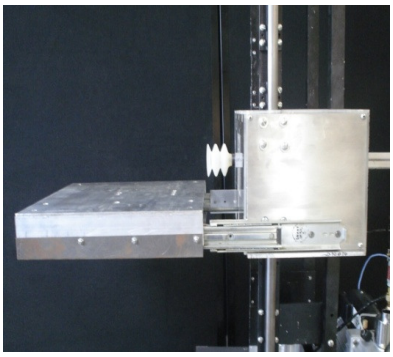


Figure 13: Retracted Arm Position

SHELF

In addition to a moveable arm, the shelf is similarly capable of extending and retracting. The shelf system is composed of standard drawer slides which are mounted to the elevator sides and support the load of the box through the use of a 14” x 14” product plate. The shelf is capable of movement through a 24V 2.2 amp DC motor attached to a threaded rod through a coupling. When the shelf must be extended, the threaded rod pushes the shelf assembly away from the tapped block attached to the elevator. The motor can then easily be turned in reverse to retract the shelf assembly. The shelf motor has an operational optical encoder

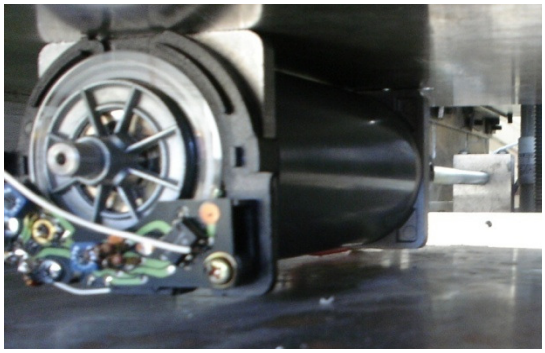


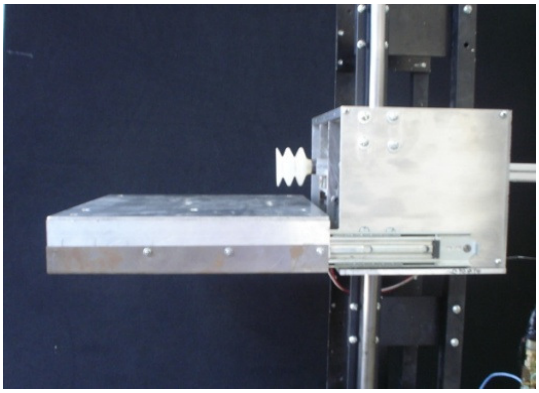
Figure 14: Shelf Motor

wheel attached to the shaft, however, shelf position did not need to be extremely accurate, and therefore the encoder was not used. A picture of the shelf motor, threaded rod, and tapped block can be seen in Figure 14 above.

The purpose of the moveable shelf assembly is largely due to the range of error in the line following capability of the robot. Due to "wobble" in the robot base as it followed the line along the shelving system, the base and therefore the arm did not line up perfectly perpendicular to the product shelf. To retrieve the product, the product shelf must be within a relatively close distance to the shelf of the item to insure that the product can be brought onto the plate smoothly. However, due to the range of error in the line sensing of the base, if the product shelf were to be fully extended and fixed in position, it would likely collide with the product shelving system as it moved along the line. Therefore, an extendable shelf providing a few inches of travel was necessary. In addition, the original design was to incorporate a rotating function. In order to achieve this functionality, the shelf must be able to retract from the product shelf before rotation can occur.

Figure 15 below shows the shelf in the retracted position, while

Figure 16 pictures the shelf fully extended.



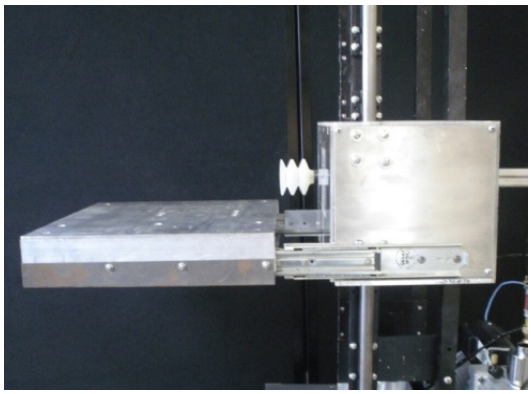


Figure 15: Retracted Shelf Position

Figure 16: Extended Shelf Position

ELEVATOR TOWER

As previously mentioned, the elevator tower was redesigned using support bars and linear guide rods which the linear bearings traveled along. These guide rods and support bars had several cross-member plates attached along the entire height of the tower to insure its stability. Support angle brackets anchored the guide rods and support bars to the top and bottom tower plates as well as to each other. The bottom plate of the tower, also referred to as the rotator plate, can be seen in Figure 17 above.

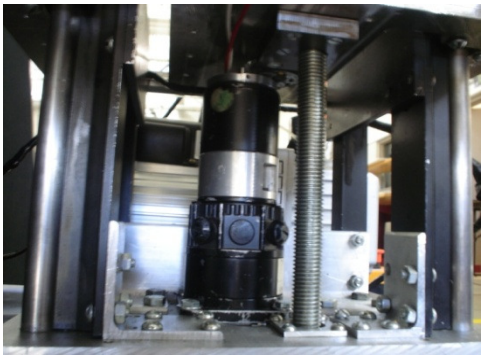
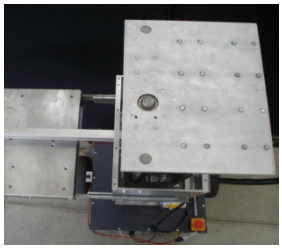


Figure 17: Bottom Tower Plate

The threaded rod is approximately 53” in length, and when it is driven at about 900 RPM it begins to wobble. If this wobble were to be eliminated completely by fixing the rod in place, it could lead to undesired binding. Therefore, a homemade rubber isolation mount was created to allow minimal movement of the rod, while still



having the rod secured in position. This was accomplished by simply using a piece of rubber hose fit on the rod and inserted into a bearing in the top plate. This component did an exceptional job of keeping the rod

Figure 18: Top Tower Plate

in the desired position, while allowing a small amount of movement. The rubber fitting can be seen in Figure 18.

The height of the elevator tower was greatly increased to allow for elevation to multiple levels of product shelves. This problem was addressed by increasing the elevator tower to a height of 52 ½ inches. Additionally, the elevator assembly travel was increased from 20 inches to approximately 31 inches, allowing for multiple levels of shelves. The elevator tower is mounted on the rotator which moves along the top of the base plate. The top of the elevator assembly is capable of reaching within 3 inches of the top of the tower, and therefore the highest shelf that the current prototype is capable of reaching is 56 ½ inches from the ground. Figure 19 shows the elevator at the highest available shelf height. Due to interference of

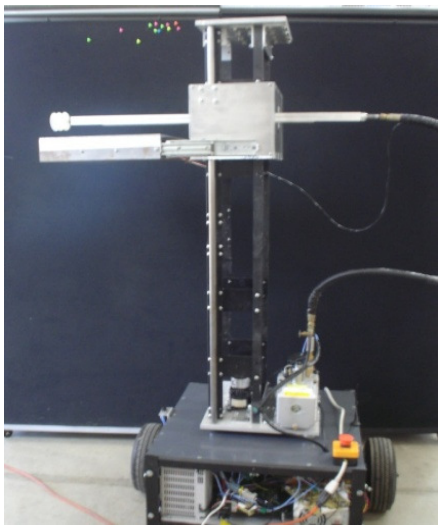


Figure 19: Highest Shelf Position

the bottom of the elevator assembly with the top of the elevator motor, the elevator assembly is unable to reach the bottom of the elevator tower. However, the lowest shelf it is capable of reaching is 25 inches from the ground.

Figure 20 to the right shows the elevator at its lowest height position. This provides a significant improvement of overall elevator travel over the initial prototype, and it should be noted that the current elevator height was restricted by the material available, and could easily be changed to reach even higher shelf heights if the direct application required it.

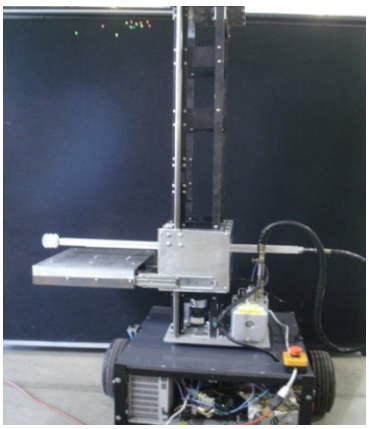


Figure 20: Lowest Shelf Position

VACUUM SYSTEM DESIGN

The vacuum pump system is shown in the drawing below. The system uses an Anver made double bellowed silicon suction cup attached to a 5.5 CFM rotary vane two stage deep vacuum pump. The deep vacuum pump is capable of 75 microns which is basically 0 Atm. The suction cups come in two (2) sizes: the small cup is 2.5 inches in diameter and is capable of 13 PSI of lift, and the large cup is 3.5 inches in diameter and is capable of 36 PSI of lift. This design allows for smaller objects (less than 40 lb) to be retrieved with the small cup and larger objects (less than 100 lb) to be retrieved with the large cup.

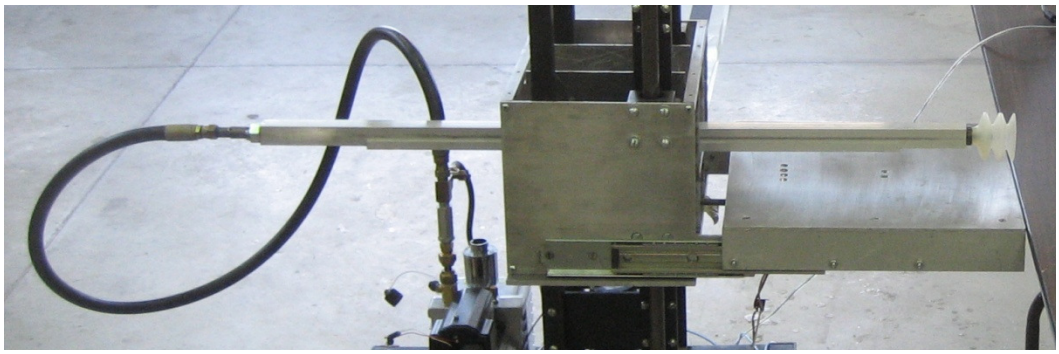


Figure 21: Vacuum Assembly

The two (2) suction cups were selected for the designed load (small cup) and an oversized load (large cup). The suction cups were both fitted with compression fittings and threaded with the same sizes making the connection point to the arm universal. The bellows style cup was selected for its strength and flexibility. An object can be off center and angled with relation to the suction cup, but the suction cup will still be able to attach to the target item. The universal connection fitting is connected to a pipe that is encased within the arm structure and attaches to a hydraulic hose on the opposing end of the pipe. The pipe has both a mechanical thread for attaching it to the arm structure, and a pressure thread fitting for attaching the vacuum hose. The hose is a flexible connection between the pipe and the vacuum pump.

The vacuum pump is attached to the base plate and acts as a counter balance to the elevator structure. This vacuum pump was selected for several reasons: vacuum power, durability, and ease of use/adaption for the design.

The horizontal vacuum design was chosen over the following designs:

- Mechanical hand
- Electromagnetic hand lift
- Vertical suction cup lift

The mechanical hand gripper design would require space on either side of the box to grip the item eliminating useful space to store other items. Furthermore, the gripper would need to be

very spatially accurate to effectively grab the box and hold it without crushing the box with too firm a grip. The Electormagnetic hand could be used instead of the suction cup design, but the materials would need to be ferromagnetic for this to work effectively. The vertical suction hand design is feasible, but this will require space to be left unoccupied above the box and very accurate placement of the suction cup. If the suction cup was off by even a little bit, the box would shift and possibly cause a seal breach resulting in a dropped box.

Ultimately the horizontal vacuum design was chosen because this design allowed for minimal, if any, loss of space through parts separation. This design also accommodates boxes of all sizes and materials because the suction cup is flexible and very powerful for the area it is required to attach to. Additionally, the weight of the box is held up by the plate, explained in the hand design section, instead of constantly being supported by servo drives or continuous suction resulting in wasted energy.

Control of the vacuum system is very simple. The FPGA controls the vacuum pump via a relay (phototriac) because the pump is a 1/3 hp AC motor. The FPGA sends a 5 V signal to turn the pump on and a 0 V signal to leave the pump off. The FPGA will detect good suction through a vacuum pressure sensor. When there is no attachment, the sensor will have a lower pressure reading, not exceeding the threshold voltage. When the suction cup finally is attached, the pressure reading will jump, exceeding the threshold voltage indicating there is an attachment. This sensor is called a manifold absolute pressure (MAP) sensor and was found on a GEO Metro. This sensor was easily adapted to our design through a barbed t valve and flexable suction hose. The sensor uses an analog signal that is measured by the GPIO pins of the develoopment board.

This design shows a lot of potential and has patent potential. Currently, the team is pursuing a utility patent on this design. Documentation has been put together detailing the design and a meeting was conducted with a person from the Calvin Enterprise Center to go over the path forward in obtaining a patent for this design.

ROTATOR DESIGN

The rotator's (turn-disc) purpose is to allow the entire upper assembly (elevator) to turn 90° degrees to the left and 90° degrees to the right. This would allow the robot to access items on both sides of the isle. As described in the base design section, there are two bearing blocks, one on the upper base plate and one on the lower base plate. These blocks are aligned so that the turn-disc shaft is at perpendicular to the plates. The shaft going through the bearing blocks was made from ¾" iron round stock and machined on a lathe to fit the gear drive to the bottom of the

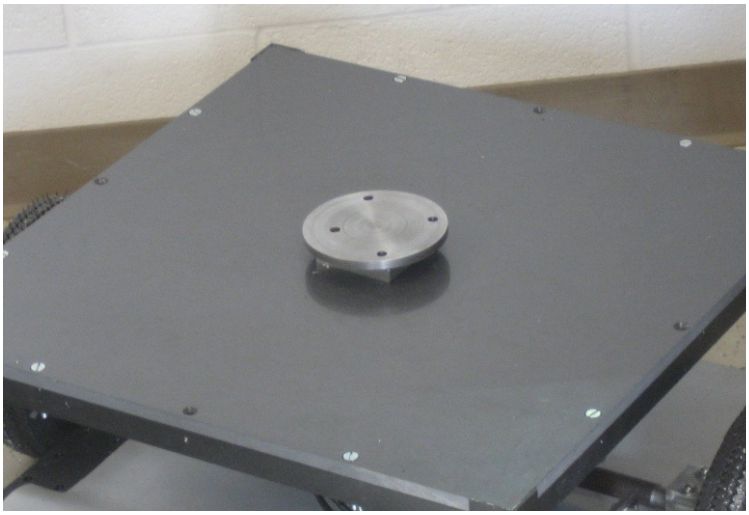


Figure 22: Plastic Deck

shaft and weld the rotator plate to the top of the shaft (assembly shown in the picture above). The ¼" iron rotator plate was then attached to the aluminum elevator assembly with 3/8" screws threaded into the bottom of the elevator plate. This allowed for the rotator to be separated from the elevator with only four (4) screws. Additionally, the elevator plate has four (4) roller bearing that support the weight of the entire upper assembly (roller bearing shown in picture below text); these roller bearing transfer the weight of the elevator to the upper deck rather than putting strain on the shaft and bearing system. This roller bearing design proved to be very robust and allowed for easy assembly/disassembly, provided the proper lubrication mechanism for very smooth rotational movement of the elevator assembly, and it created the proper gap distance between the elevator plate and the base plate to fit the drive system for the elevator screw motor.

Figure 23: Roller Bearing

Although the structural design of the rotator was quite robust and stable, there were several problems with the mechanics of the design.

The first design used an L style belt with two (2) 20 teeth timing gears; one on the motor shaft and the other on the turn-disc shaft with a keyway. The timing belt was connected under the bottom of the bottom aluminum base plate, this allowed for a direct drive to power the

rotation. The team used a geared DC motor with a speed of 5 rev/min; this allowed the design to have a lot of torque and higher precision. The control for this design was going to be a contact switch or an optical encoder. Unfortunately, the original rotator design had several flaws when we tested the prototype. First, the belt had a lot of backlash and slop which caused a large amount of error at the end of the arm due to the distance of the arm from the base and error source. After doing the calculations, the team discovered that one degree of error equates to over ½" of error at the arm. The original design had at least +/- 10^{o} degrees of error resulting in 10" or more inches of error at the arm; this was not acceptable because it did not meet the teams design specification (retrieve a box of 4" dimensions). The team decided to get a new motor and try using different gear sizes and belts to correct this problem. Secondly, the tension on the motor bushing was too much for the motor to handle and caused unwanted binding of the motor shaft. Ultimately, this motor was unsalvageable because of the binding problems.

The team purchased another stepper motor for the rotator and changed the belt design to an XL style with two (2) larger 48 tooth timing gears. The team felt that the larger gears combined with the smaller tooth resolution would produce less belt backlash and the stepper motor would provide better motor accuracy for the rotation resolution. Also, the motor came with an optical encoder that could be used for positioning. The new system was setup the exact same way as the previous design and was tested for performance. The results turned out better, but still they were unacceptable. The rotator system did not turn consistently and we discovered that the motor still might be under powering the rotation. The error in positioning was not as close as it needed to be, and the belt still had too much backlash. Fortunately, this element of the design was not critical to the overall functionality of the robot and could be temporarily fixed to get the robot operational and prove concept of design.

The team feels that it would be possible to get this option working with more design time and prototype testing. There are three designs that the team feels could get the rotator functioning and operational to an acceptable level:

- A detent mechanism that acts as a mechanical positioning system for the rotator to "fall into" when in the proper position. This would be combined with the existing designs and an optical encoder that would help the motor position the assembly to the proper position.
- A meshed gear system with low backlash would be a very accurate way to position the rotator. This design would require a lot of very precise machining and design, but the team feels that this would work very well given more time to complete the design and construct a prototype.
- A four-bar linkage mechanism would also be a suitable design for the rotator. This, like the meshed gear design, would produce a directly driven system with little to no backlash. This design would require a new high torque motor for driving the rotator and would also require the redesign of a rotator shaft. Unfortunately, this design would also require more time to design and construct.

Due to time constraints on the design prototype completion, the team decided to go fix the rotator into position with a bolt. This allowed the prototype to be operational and the turn-disc design and construction could be completed at a later date.

BASE DESIGN

The base design was required to support the weight of the elevator and arm structures as well as the item being retrieved. The base was required to operate at speeds up to 6 mph and maintain traction and stability up to this maximum speed. The base needed to have the turn disc mounted atop the upper decking and provide stability of the turn disc by some means, namely a bearing system. The base needed to have room for all the components to be mounted within it providing protection from outside elements. Most importantly, the base was required to provide stability while the robot was pulling the item from the shelf and returning the item to the operator.

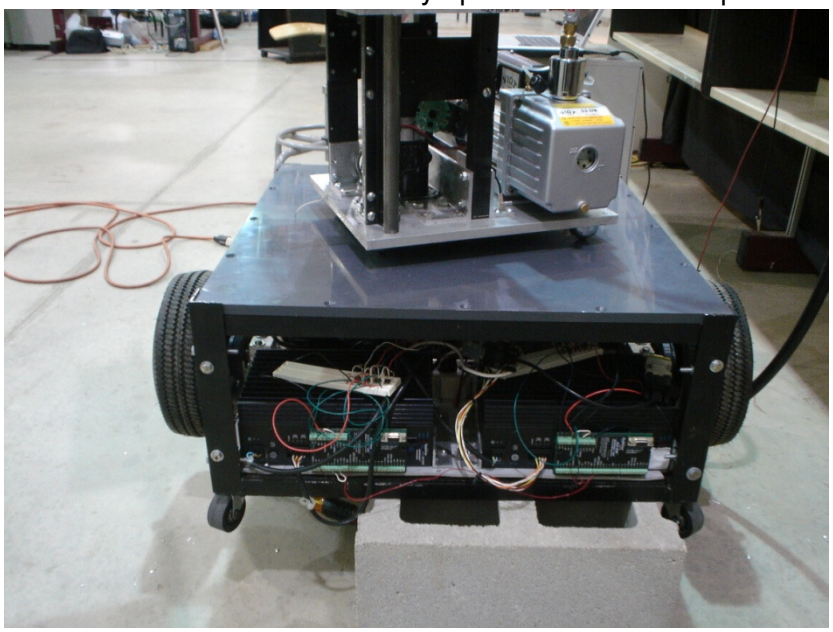


Figure 24: Base

The base structural design is 24” x 24” x 10” and is made completely from aluminum. The frame skeleton is 1” square aluminum tubing welded together in two separate pieces and connected using screws and angel brackets for easy disassembly. The bottom plate is 3/16” aluminum for mounting the internal components and the bottom bearing assembly. The top plate is ½” high density polyethylene which supports the entire weight of the elevator and plate assembly. Both of these plates are fastened to the frame with 12 ¼” bolts. The base has several systems and components housed within and on the structure.

The drive and stability system has a solid, unmoving axel for mounting the drive wheels on. These wheels are driven by stepper motors through a direct drive timing belt at a 1:4.8 gear ratio. The original design was for a 1:8 gear ratio, but due to financial and construction material constraints the team decided to go with the lower gearing system. This drive system consists of a ten (10) tooth gear mounted through a specially design and machined coupling component to the stepper motor shaft. The motor drives the 48 tooth timing gear mounted on the two drive wheels through a ½ inch XL timing belt. The XL style was used to allow for very precise movement and reduced slop and backlash in the belt. The gear mounted on the drive wheels is

welded directly onto the wheel rims through another specially designed and machined coupling piece. The wheels on the drive system are pneumatic so the traction and height can be adjusted through air pressure; additionally the pneumatic wheels provide some damping for the base vibrations and platform rock. The drive wheels are supplemented by 4 caster wheels mounted at all four base corners to provide the necessary stability for the base system. These caster wheels are modified from their original state to provide the necessary rotational speeds required by the design specifications. The original wheels only incorporated a plastic walled rotating surface which would easily melt when the base was at top speed (caster wheel RPM > 1000). The wheels and axel/bearing system was changed out for modified rollerblade wheels. The rollerblade wheels were lathed down to the proper height and incorporated a more robust axel/bearing system.

As mentioned above, the motors used for the drive system are stepper motors. The stepper motors are Slo-Syn 200 step/rev motors rated each at 350 oz-in of initial torque and capable of 3000 rev/min. These motors were mounted to the frame using designed and machined steel angle brackets and bolts. These motor mount brackets are attached to the frame skeleton and the aluminum plate using ¼" bolts and nuts to provide the proper stability and strength to drive the entire robot forward and backwards and for ease of assembly and disassembly. The two (2) drive motors are powered and driven by the stepper motor controllers both mounted to the bottom plate.

There are three (3) sensors mounted to the base frame: the ultrasonic distance sensor, the line tracking reflectance sensors, and the IR proximity sensor. The ultrasonic distance sensor is mounted on an angle bracket which is attached to the top left side of the base pointed upwards at the bottom of the plate arm assembly. This placement gives the distance from the top of the base to the bottom of the arm plate. The reflectance sensors are attached to the bottom of the base on the front frame tubing using two threaded rods and an adjustable plate mount. This design needed to be adjustable on two axes because the reflectance sensors needed to be adjustable height wise due to sensor sensitivity issues and also needed to be adjusted width wise for line following control damping function optimization. Finally, the IR proximity sensor is mounted to the top front part of the frame as a safety measure for people or objects in the way of the robot's path of travel. Structurally the sensor is unable to be adjusted, but through software the sensitivity can be adjusted to optimize this safety feature.

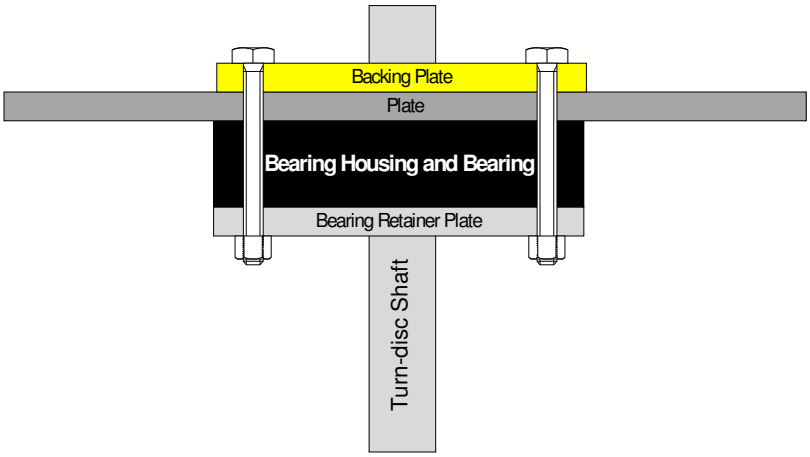


Figure 25: Bearing Block

To accommodate the turn-disc, the base has two bearing blocks that fit a ¾" shaft. The bearing blocks are mounted dead center on both plates to allow the turn-disc to mount in the middle of the robot's base. Each bearing block is mounted to each plate with four ¼" bolts and two ¼" dowels to keep the proper 90o alignment of the turn-disc shaft. The bearing blocks house the bearings in ½" machined aluminum plates with bearing retainer plates machined from 1/8" steel plate. The bearing blocks are held to the plate with through-hole bolts and abacking plate opposite the bearing block and base plates. This can be visualized with the drawing of the top plate bearing block above. This design and implementation was chosen because it allowed for hand fitting instead of precise machining. Precise machining would normally be preferred in this situation, but could not be reasonably accomplished due to the limits of shop equipment size and error propagation in the base structure tolerances. This design is accurate to +/- ¼" from the base to the top of the elevator assembly.

The turn-disc motor is a stepper motor and is mounted to the aluminum base plate with ¼" bolts which thread into the aluminum case of the stepper motor. This motor is controlled by a stepper motor controller mounted above the left drive motor stepper controller. This stepper motor is used to rotate the turn-disc to the left and right through a timing belt drive. This design will be explained in more detail in the rotator design section of this report.

The AC and DC power components of the design are shown in the diagram below in red. The AC power strip is the source for all onboard electricity on the robot. The power strip has six (6) plug connections and is connected through an emergency switch that disconnects all power to the robot in an emergency situation. Three (3) of the six (6) power connections are used for the stepper motor controllers, two (2) are used for the DC power supplies, and one (1) is used for the AC powered vacuum pump. The adapted computer power supply is mounted to the aluminum plate in the right rear part of the base and powers the FPGA and some small control circuitry, and the 24 V power supply powers the regular DC motors through the H-bridge and other control circuitry. The 24 V supply is connected to the frame with a DIN rail for easy removal and is located in the left rear portion of the base. The H-bridge is not mounted on any structure, instead it has an insulation layer applied to the exposed connections and is laid on top of the computer power supply. The H-bridge relay along with the two (2) transistor switches (for the vacuum pump relay and the H-bridge relay) are located in the area below shown in yellow. These small circuits are only prototyped on a breadboard, but could easily be made into printed circuit boards (PCBs). Lastly, the development board is mounted in the middle rear of the robots base. The development board was placed here because it provided easy access and could easily be connected to the computer power supply. This position for the development board also provided the best place to have all the wire connections; wires coming from the back would not interfere with the turn-disc movement. Unfortunately, we were unable to provide the development board with any shielding from the powerful motor magnetic fields surrounding the board. It would be advisable to have some kind of chassis for the development board to be mounted inside of for long term use. This action would reduce potential bit errors and increase longevity of the development board. A hole was drilled through the top plastic plate to connect the wires for the elevator assembly's motors, the vacuum pump, and the MAP sensor. This

gave the team better wire control and would allow for the sides to be enclosed with plastic or metal pending a ventilation system design to cool the heat producing components in the base. Lastly, an electrostatic mat was placed on the aluminum base plate to prevent electrostatic buildup and discharge of the plate on the electronic circuitry. This measure is purely precautionary as we did not have any problems with electrostatic circuitry damage.

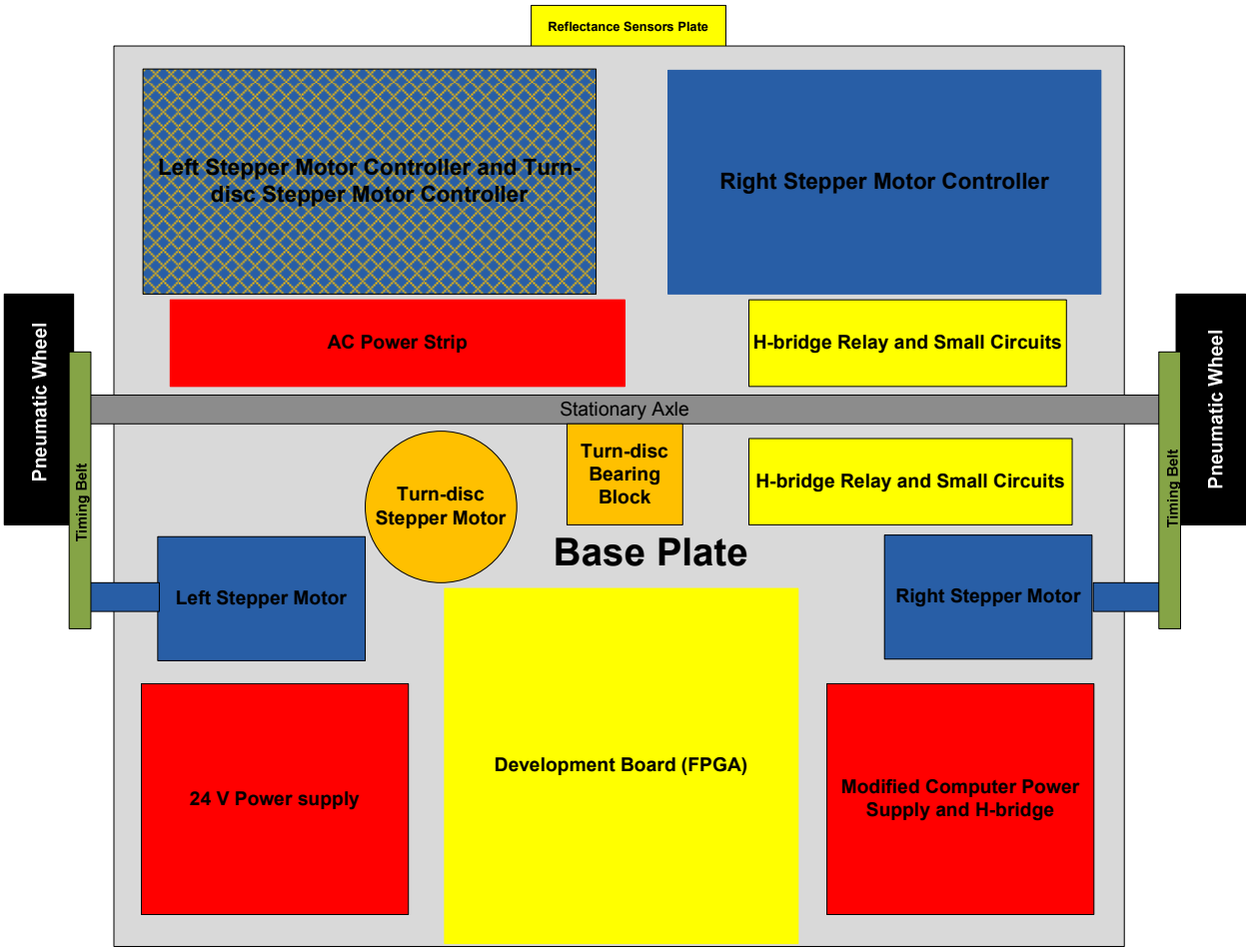


Figure 26: Base Layout

In the diagram above: you can see all the power circuitry is shaded in red, the control and data components in yellow, the drive motors and controllers in blue, the turn-disc motor and controller and bearing system in orange, and the remaining colors for the drive system. The base (lower assembly) design and prototype were a complete success in the scope of this project. The base is able to travel at speeds of 6 mph and greater, it can carry the entire weight of the robot's upper assembly, it has excellent stability and controllability, and it is able to house all of the components necessary for the robot to operate properly.

Surprisingly, the team did not have any problems during the design and construction of the base prototype. Only minor design changes were made to the base design as the prototype went from a concept idea and drawing to a working concrete manifestation. The team did experience some minor problems with the drive system; the belt tension is not easily adjustable

and sometimes slips timing cogs when accelerating and decelerating. To fix this problem, the team created slots in the motor mounting bracket to allow adjustability in the belt's tension. That being said, the team was very happy with the base design and prototype and feel very confident that all conditions for success—only specific to the base—were met.

DIGITAL HARDWARE DESIGN

SENSOR DESIGN

The sensor design was integral to the success of this project. The sensors give the device the capability to feel its way around its environment, rather than running around blindly. There are three human sense categories that the STORBOT sensors fall into: touch, sight, and sound.

TOUCH

To prevent the mobile unit from running into the shelves or into people, the last stage of "defense" is a series of touch sensors. Lever switches perform this role, attached to a hinged metal plate. The metal plate is pitched slightly backwards. As the bottom (leading) edge comes into contact with a surface, it compresses the lever switches under it, which send a signal to the processor, which in turn shuts down the motors sending the device in that direction.

SIGHT

There are two sensor types used to control the robot's sight. The first of these is reflectance. Reflectance is a useful sense for multiple reasons. First, reflectance is not a sense that stands out particularly much to humans. Therefore, an environment can be constructed with very high variance in reflectance, and a human in that environment will not be inconvenienced or troubled by the change. Reflectance is also useful because it allows for a rapid, stark change between inputs without requiring mechanical movement or extensive analog circuitry. The switch from reflective to non-reflective actually can be read on any digital I/O port, so these sensors are ideal for quick integration into a prototype robotics system.

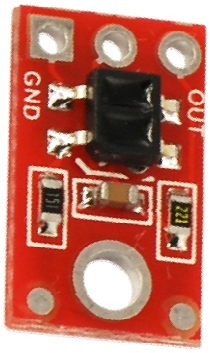


Figure 27: Reflectance Sensor

One goal the team had for this project was to integrate the entire electronics into an existing processor system through extensive use of VHDL. A sample is shown below in Listing 1 outlining the use of a tri-state buffer for the reflectance sensors. The output line had to be charged for at least 10 microseconds, then read for up to 1 millisecond. Once the output line dropped below the FPGA's threshold voltage, a logic zero was read. During this reading cycle, a register was incremented while that input line was high, and held constant when it was low

and during that charging time. Thus, the register could have a range of values, corresponding to the reflectance of the material the sensor was over.

This was used to implement the line-following functionality of the robot. By following a reflective/ non-reflective line, the robot leaves no permanent footprint in the warehouse, like if it needed a mechanical track.

Listing 1: Reflectance Sensor VHDL

```
counter_proc: process (Bus2IP_Clk, Bus2IP_Reset) is
    begin
        if Bus2IP_Reset = '1' then
            internal_counter <= 0;
        elsif rising_edge(Bus2IP_Clk) then
            internal_counter <= internal_counter + 1;
            if internal_counter = 100000 then -- after 1 ms, we have
done enough sampling, and can start over
                internal_counter <= 0;
                read_value <= '0';
            elsif internal_counter = 99999 then
                register_out <= register_incrementer;
            elsif internal_counter = 1000 then -- after 10 us, we can
start reading
                read_value <= '1';
            end if;
        end if;
    end process counter_proc;

    buffer_proc: process (Bus2IP_Clk, read_value) is
    begin
        if Bus2IP_Reset = '1' then
            Sensor_IO_T <= '0';
        else
            if read_value = '0' then
                Sensor_IO_T <= '0';
                Sensor_IO_O <= '1';
            else
                Sensor_IO_T <= '1';
            end if;
        end if;
    end process buffer_proc;

    sensor_read_proc: process (Bus2IP_Clk, Bus2IP_Reset) is
    begin
        if Bus2IP_Reset = '1' then
            sensor_read_val <= '0';
        elsif rising_edge(Bus2IP_Clk) then
            if read_value = '1' then
                sensor_read_val <= Sensor_IO_I;
            end if;
        end if;
```

```
    end process sensor_read_proc;

    --test_sensor_read_val <= sensor_read_val;
    --test_read_value <= read_value;

    output_vectorizing_process: process(Bus2IP_Clk, Bus2IP_Reset)
    begin
        if Bus2IP_Reset = '1' then
            register_incrementer <= (others => '0');
        elsif rising_edge(Bus2IP_Clk) then
            if (read_value = '1') then
                if (sensor_read_val = '1') then
                    register_incrementer <= register_incrementer + 1;
                else
                    register_incrementer <= register_incrementer;
                end if;
            else
                register_incrementer <= (others => '0');
            end if;
        end if;
    end process output_vectorizing_process;
```

Another sensor that employed the sense of sight was optical encoders. Optical encoders actually employ a similar technique used by reflectance sensors. They have a series of very small black and white (or black and clear) lines arcing from a concentric point pass under a beam-break sensor or a reflectance sensor. The output of that sensor is a close-to digital signal; high when black, low when white/clear. If you watch the output and count the number of rising edges, and also know the number of tick marks for the whole circle, you know the rate of rotation. If that disk bearing the tick marks is then mounted on the shaft of your motor, you can know how far a motor

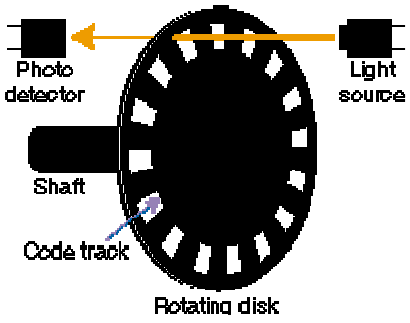


Figure 28: Optical Encoder

has rotated. If there is no slip assumed, this then can be directly correlated to how far the motor has moved its load.

SOUND

The sound sense is enacted through two ultrasonic distance sensors mounted on the front and back of the base unit. These are used in redundancy with the absolute position monitoring software on the slo-syn motors to determine position along the shelves, and distance from "home." The sensors (MaxBotix, P/N LV-EZ1), provide a serial, analog, and pulse width

Figure 29: Ultrasonic Distance Sensor

modulated output. STORBOT incorporates the PWM output. It is accurate to the inch for a range of about 21 feet. The pulse width varies by 147 microseconds per inch.

The final use of these sensors was extremely integral to the success of the project. Due to their accuracy as well as their precision, and the loss of multiple optical encoders, it was decided to use the sensors as the method for determining elevator height. A sensor was mounted on the side of the robot, above the wheel, facing straight up. From this position, it was able to bounce its ultrasonic waves off the bottom of the shelf, a very flat surface clear of obstruction. The returned waves were thus very clean, and an accurate height could be obtained.

Figure 30: Ultrasonic Sensor Mounting

One issue that arose here was that the accuracy of that height value turned out to be only reliable to a resolution of one inch. Since the shelves we used were about 0.5 inches tall, and we needed to end right in the middle of them, some debugging was necessary. The final software performs some time averaging, as well as range finding to get a precise height. We ensured that it would always approach every shelf from the bottom, and the rise up. This seemed to resolve that problem acceptably well.

SYSTEM INTEGRATION

The FPGA at the core of STORBOT permitted digital hardware to change throughout the project. The team used Xilinx FPGAs because of their wide support, but an initial learning curve needed to be conquered to understand Xilinx software.

TOOLS AND SETUP

The following sections outline the digital hardware and software tools that were used to control STORBOT's peripherals.

Xilinx University Program Virtex-II Pro Development Board

A "Xilinx University Program Virtex-II Pro" (XUPV2P) development board was STORBOT's digital platform. This board was also chosen because of its wide support base – many tutorials are available online about how to use it for different purposes (several different tutorials were used for different sections of the project).

Xilinx ISE Foundation

The team used the "Xilinx ISE Foundation" software package in conjunction with the XUPV2P development board. This licensed software is available for several thousand dollars directly from Xilinx, but a 60-day trial version is available online. Given the project's budget constraints, the team decided to renew the trial version every 60 days for the duration of the project.

The software was installed on a host computer and reinstalled twice during the time the project's duration. The Xilinx version of 'make' conflicts with recent installations of cygwin, so the version of the 'make' utility needed to be monitored and occasionally changed

The team made several attempts at getting the Xilinx software packages running on Ubuntu Linux, but were not successful. Although it would have saved some development time to have all software and utilities on one operating system, the time that it would take to figure out how to run this software to Ubuntu Linux may have been even more time consuming.

XILINX HARDWARE SETUP

Tutorials on how to use Xilinx software are available at several sites on the internet. The team created one tutorial about setting up software libraries, adding peripherals, and a few other minor functions, included in Appendix A. Other important sources of information are listed in Table 4.

Table 4: Sources of Information for Linux on Xilinx boards

Source	Description
http://www.fpgadeveloper.com/	This website provided information about creating basic hardware setups, adding peripherals, creating peripherals, and basic techniques with the XUPV2P. Only simple C programs could be written, compiled directly to PowerPC assembly language and downloaded by USB to the board (no operating system information).
https://rm-rfroot.net/xupv2p/	Jonathan Donaldson's personal website outlines the steps to put Montavista Linux 2.4 on the XUPV2P (as part of his master's thesis). It provides information on setting up a cross-compiler and building a root file system. However, Montavista Linux 2.4 is not actively supported, and the latest versions did not fit in the team's budget.
http://git.xilinx.com	Near the beginning of 2008, Xilinx set-up this website to consolidate information related to putting Linux on different versions of their boards. Although the kernel branch that is maintained at this site is meant to be used with either the Virtex-4 or Virtex-5, the kernel is general enough that it also works with the Virtex-II Pro. Instructions are available for compiling the kernel and generating a "device tree file" (.dts) for the kernel from a Xilinx project.

ANALOG AND MIXED SIGNAL CIRCUITS

The team relied on several circuits to provide the interface between digital control signals and devices being controlled. The H-bridge, opto-isolation, and relay circuits were the most important of these circuits and are summarized here.

H-BRIDGE

An H-bridge circuit provided the interface between the digital signals coming from the board and DC motors for STORBOT's arm, shelf, and elevator. The four inputs to the H-bridge were properly paired so that two incoming signals low-voltage signals could control the direction of the DC motors on the other side of the H-Bridge. The circuit schematic is shown in Figure 31.

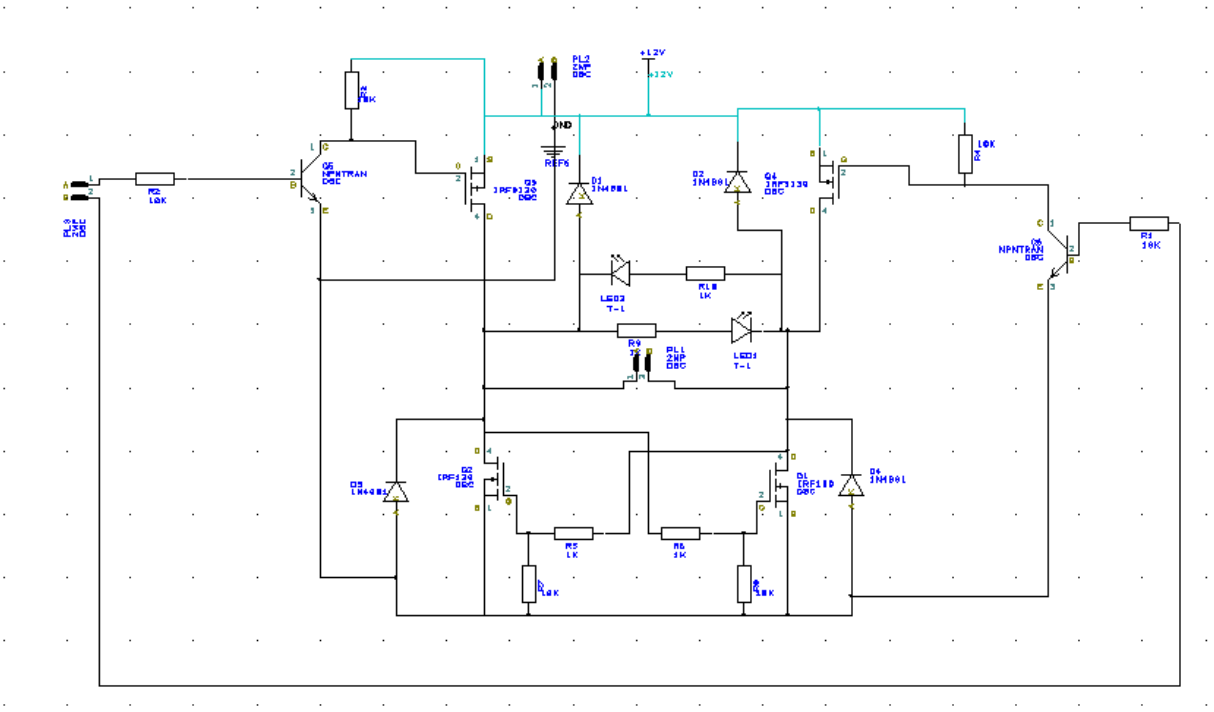


Figure 31: H-Bridge Schematic

The H-bridge circuits were prototyped on a breadboard and then compacted into a two-sided PCB for being placed in STORBOT's base. The high-voltage lines on the H-bridge were properly protected from grounded cases and other signals with non-conducting tape. The H-bridge PCB is shown in Figure 32.

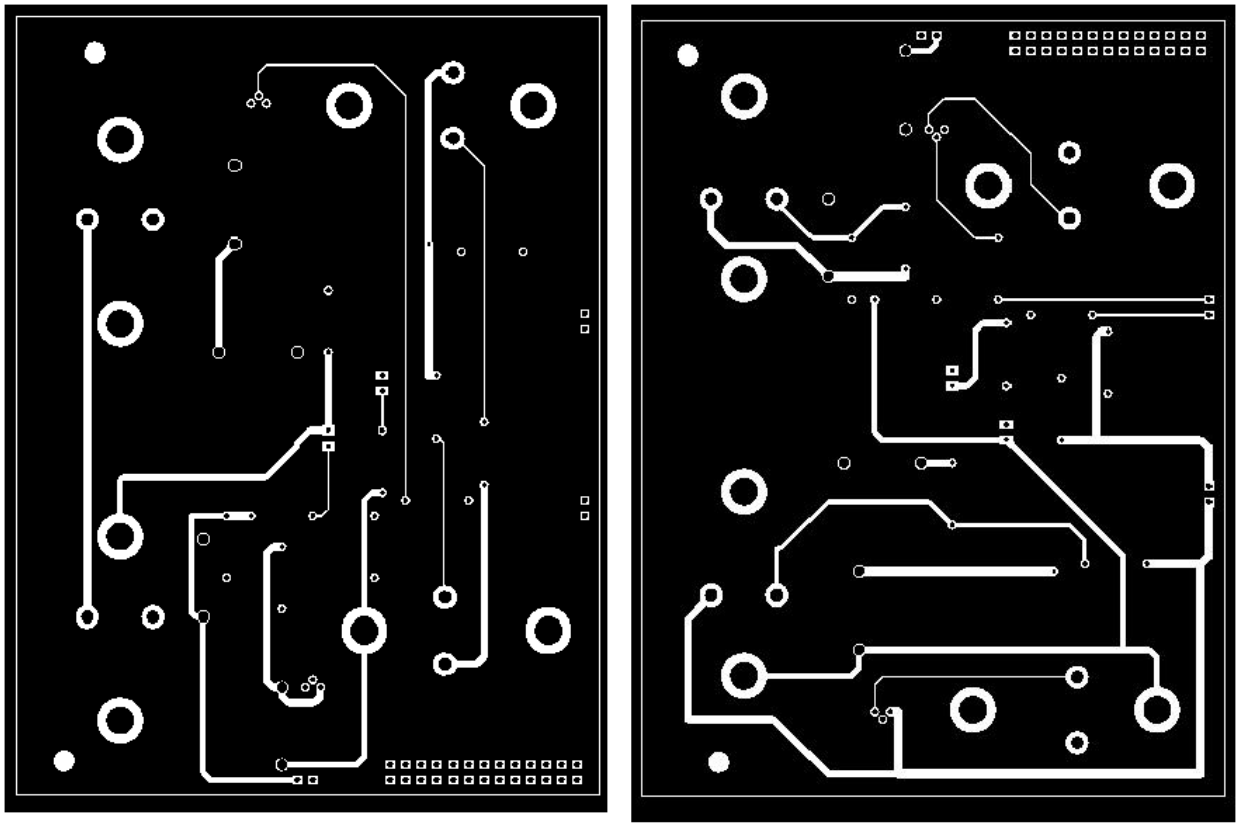


Figure 32: H-Bridge PCB Layout, Top and Bottom Sides.

The lines that left the far side of the H-bridge for the set of DC motors were carefully thought out to prevent problems that arise in high current lines. In STORBOT's base, 22 AWG wires were the standard for all power and signal lines. The H-bridge to DC motor lines, however, had thicker 16 AWG to accommodate the higher currents expected (trading off wire flexibility). Resettable fuses (Positive temperature coefficient, PTC, thermistors) were also soldered into these lines to prevent any undesirable feedback to the H-bridges and, if something went wrong, limit current to the DC motors.

OPTO-ISOLATION

Opto-isolation circuits between the FPGA and the Slo-syn motor controllers provided protection from back-emfs generated by the motor controllers during turn-on and power-off. The 4N33 opto-isolators were configured in their default test configuration, as shown in Figure 33 below.

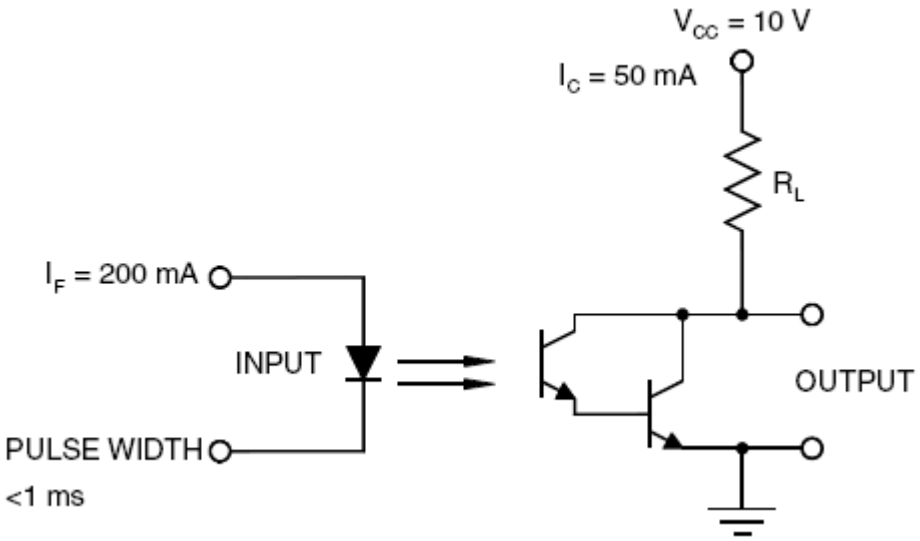


Figure 33: Opto-Isolation Circuits

Although the motor controllers advertised internal opto-isolation, practice with them revealed that not all pins were properly protected (though some may have been). The team lost a custom-made circuit board of eight voltage steppers and twelve development board pins to back emfs from the motor controller.

This external opto-isolation circuit protected the board, and also provided a voltage step through the two photo-sensitive transistors. This extra feature allowed the motor controllers to operate with 12V logic pin inputs, their default configuration.

RELAY

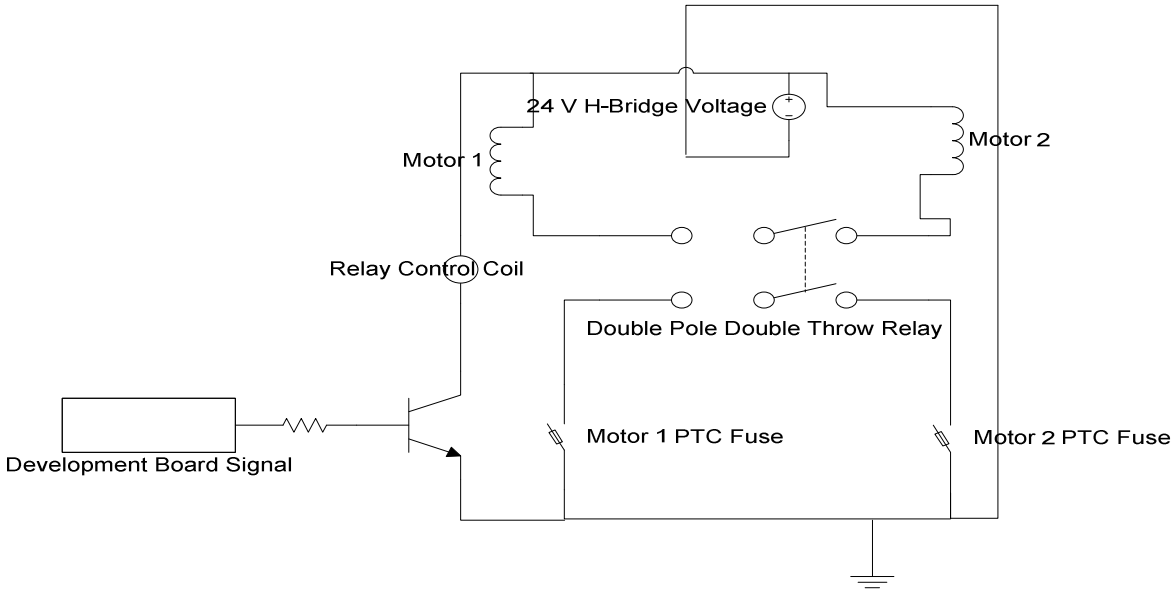


Figure 34: Relay Circuitry

The relay circuit (shown above) allows for the design and connection of only two H-bridges for the robot's DC motors. The relay is a double pole, double throw (DPDT) relay that allows one of the H-bridges to switch between the plate motor and the elevator motor since these motors never need to operate at the same time. The H-bridge is connected to the input terminals of the relay and powers the elevator motor when in the normally closed (NC) position. When the H-bridge needs to be switched to the plate motor, a signal is sent from the development board to a transistor switch that powers the 24 V relay into the normally open (NO) position. In the NO position the plate motor is powered by the same H-bridge as the elevator. Both of these motors are connected in line with PTC fuses to protect the motors from being overdriven. This design saved the implementation of one (1) GPIO pin and saved time and resources by eliminating the need for a third H-bridge PCB.

SOFTWARE DESIGN

OPERATING SYSTEM

Although the project could have proceeded without an operating system, the advantages associated with an operating system (high-level languages, interrupt management, CPU and process management, among other services) were considered worth the extra time investment to set up an OS. The team used the most widely-known and supported free operating system, a simple embedded version of Linux.

RELATIONSHIP TO HARDWARE

Hardware and software are integrally tied together, with the latter dependent on the former in many ways. STORBOT's design, furthermore, often required daily hardware changes. Device tree files (.dts) were the link that kept the operating system aware of any new hardware changes. Although a .dts file could be manually written from a detailed understanding of our hardware system, Xilinx provided a "Device-Tree Generator" utility that automatically created a .dts file from a Xilinx hardware project. An example .dts file for STORBOT is given in Appendix D.

The Linux build tree provides locations for .dts files in several architecture branches. The build system detects a new or modified .dts file and uses it to get a hint for what may be coming in hardware when it boots. Linux also probes for other new hardware on boot and during operation.

CROSS-COMPILER

A cross-compiler is required to build a kernel for the PowerPC processor (unless a PowerPC computer is available, an unnecessary extra cost). The most common way to build a cross-compiler is using Dan Kegel's 'crosstool' utility, available at http://www.kegel.com/crosstool/. This utility was downloaded and installed on our development system (Ubuntu Linux). The team was able to successfully build a cross-compiler after careful configuration and experimentation with gcc and glibc versions.

DEVICE DRIVERS

Putting Linux on the XUPV2P is possible thanks to the wide device driver support for Xilinx devices on Linux. Many of these device drivers were written by Xilinx and so are highly reliable. The site http://git.xilinx.com provided one site to download the Linux kernel with all of these devices included. Our own device drivers could be included through the Linux build system already in place.

DEVELOPMENT AND BUILD

With the cross-compiler set up and a correct .dts file in place, the Linux kernel was ready to build. To configure a kernel, type:

```
$> make menuconfig
```

at the root of the build tree. In this menu, the team specified which drivers to include and where to locate the .dts file, among other things. Figure 35 shows the kernel menu used to include STORBOT device drivers.

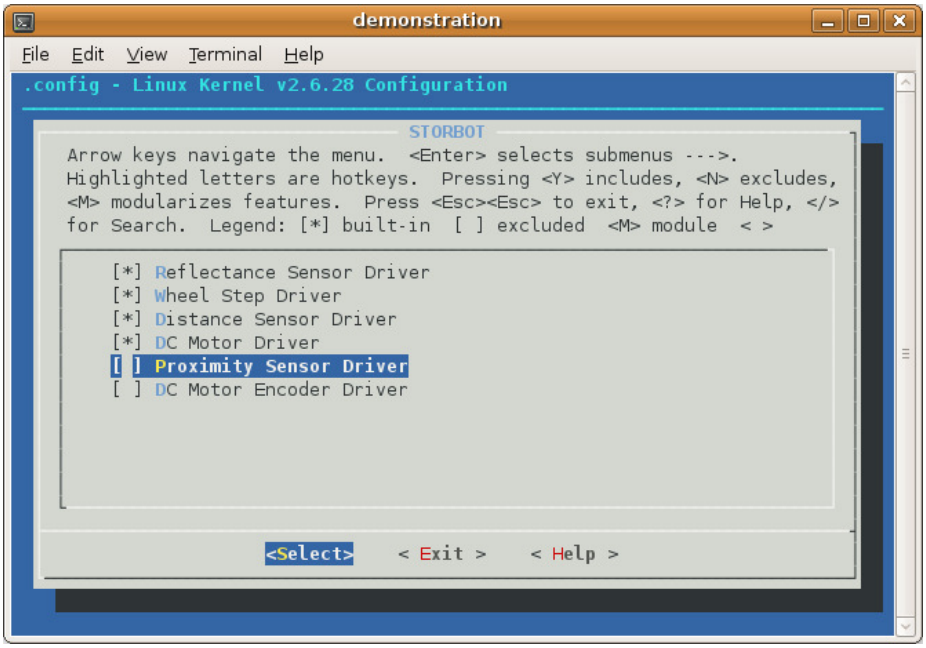


Figure 35: Menu to select STORBOT's device drivers

To build the kernel, type:

```
$> make ARCH=powerpc CROSS_COMPILE=powerpc-405-linux-gnu- zImage
```

This specifies the architecture and cross-compiler that will be used to build the kernel. The kernel is available in the build tree at arch/powerpc/boot/.

The kernel is useless without hardware to run on, and without a system to start it. Xilinx provides the 'Impact' program to combine an executable with hardware, and we used this program indirectly to build an .ace file. An .ace file can be used by the SystemACE CF card manager to automatically (on the boot of the board) load hardware into the FPGA and then load the executable kernel to the first location run by the processor in SDRAM (essentially functioning as a boot loader). To create the SystemACE file we specified several build parameters in an .opt file, for example:

Listing 2: Example of .opt file used to generate .ace file

```
-jprog
-board user
-target ppc_hw
-hw implementation/system.bit
-elf kernel/simpleImage.virtex405-ml405.elf
-configdevice devicenr 1 idcode 0x1127e093 irlength 14 partname xc2vp30
-debugdevice devicenr 1 cpunr 1
-ace acefile/storbot.ace
```

…and executed (for an .opt file name xupGenace.opt):

```
$> xmd -tcl genace.tcl -opt xupGenace.opt
```

The operation results in an .ace file named storbot.ace. We moved this file to the first partition of the CF card, and on startup the board loaded the appropriate hardware and kernel from the .ace file.

ROOT FILE SYSTEM

The setup provided by Xilinx's "SystemACE" CF card manager allows the system designer to use system .ace effectively as a boot loader. However, a kernel without a file system to mount has little to no purpose (In Linux, everything is a file), and the kernel would have to be carefully changed to make this possible without a kernel panic. Fortunately, Xilinx's "SystemACE" CF card manager also allows the designer to treat a CF card as a hard drive (for example, by booting off of /dev/xsa2). Linux first boots off an initrd or initramfs (small file systems included in the kernel, in RAM) and then mounts the real file system on the CF card.

MKROOTFS

A root file system for Linux can include less than ten files, but to perform any real functions we needed to provide more files to the operating system. Initially, the development process focused on a short script written by Wolfgang Klingauf, available at http://www.klingauf.de/v2p/. Through the project most of this script was deleted and replaced by the team's own code, and the script became significantly longer. The script is included in Appendix E. The essential features of this script (the executable name is mkrootfs) are:

1) Re-create the file system from the ground up (mkdir {bin, dev} etc.)
    - This effectively protects the root file system from errors on the board. When the power to the XUPV2P is turned off (without properly shutting down) stale file handles begin to corrupt the file system. These errors would otherwise make us slowly lose the file system we had built.
2) Install BusyBox and associated libraries
    - BusyBox is a collection of minimal Linux utilities collected into a single executable (such as ls, cd, mkdir, rmdir, less, etc.). BusyBox can be downloaded from http://www.busybox.net. This program is used in many embedded systems

all over the world. BusyBox requires glibc libraries that are also copied to the file system by mkrootfs.

3) Install native PowerPC compiler.
    - A native PowerPC 405 compiler was cross-compiled using a Desktop PowerPC computer (for a similar project with an XUPV2P) and is available online at http://www.kdvelectronics.eu/ml403/ppc405_native_gcc.html. The engineer who performed the task, Koen De Vleeschauwer, personally helped the team add the native compiler into the file system. Because the native compiler requires significant number of extra libraries, it is only included in some file system builds.
4) Install /home, /etc, and compile C and C++ source code.
    - Although other folders can be recreated from the ground up, /home and /etc must be stored on the host computer and are copied to the file system every time it is created. Any C files in /home are automatically compiled and the main set of code for STORBOT, written in C++ and located in /home/controller, is compiled and added to the file system.
5) Add custom device drivers to the file system
    - The device drivers that are compiled into the kernel are only available to user space through files in /dev that specify their major and minor numbers.
6) Un-mount CF card attached to host computer
    - After the file system has been copied to the second partition of the CF card, it must be un-mounted before it can be removed from its holder and put in the XUPV2P. Failure to do this often results in stale file handles on the card.

KERNEL BOOT

On kernel boot, the first user space process (named init) reads files in /etc for banners, runlevels, and other instructions. The BusyBox version of init by default runs /etc/init.d/rcS when no inittab file is present in /etc. In this script file, the designer can specify any activities they would like to happen at kernel boot. The rcS file used with STORBOT is included in Appendix F. This script was originally provided by Wolfgang Klingauf, but almost completely re-modified for STORBOT's purposes. This script

1) Mounts pseudo file systems (/tmp, /proc)
2) Starts the system log daemon
3) Populates /dev using the BusyBox equivalent of udev, called mdev
4) Starts a shell immediately (rather than wait for a login to start a shell)

DEVICE DRIVERS

The digital hardware peripherals on STORBOT interfaced to the operating system through a small set of device drivers. These device drivers abstracted the physical address, device-specific operations, and direct register access from the peripherals. The example user space interface code and driver code listed below demonstrates the most important functions of the device drivers.

PERIPHERAL INTERFACES

This section attempts to outline the team's motivation for using device drivers in STORBOT's design, with the wheel step peripheral as an example. At the hardware level, the wheel step peripheral is five simple pins that are generally assigned to drive one of the motor controllers running STORBOT's wheels. Without a driver, the method for controlling these pins from user space would be relatively complex. The drivers provide a cleaner, standardized interface to the peripheral and saved the team from error prone and time-consuming copying and pasting of code.

No Device Driver

Without the device driver, controlling the wheel step peripheral would have to be performed by manually writing to locations in SDRAM. Listing demonstrates the convoluted interface to the wheel step peripheral using this strategy.

Listing 3: User Space Interface to Peripheral - No Device Driver

```
int main() {
        int memfd;
        void *mapped_base_wheel_step, *mapped_dev_base_wheel_step;
        off_t dev_base_wheel_step = 0xC7C00000;
        memfd = open("/dev/mem", O_RDWR | O_SYNC);
        if (memfd == -1) {
                printf("Can't open /dev/mem.\n");
                exit(0); }
        // Map one page of memory into user space such that the device is in that page,
        // but it may not be at the start of the page
        mapped_base_wheel_step = mmap(0, (4096UL), PROT_READ | PROT_WRITE, MAP_SHARED, memfd,
        dev_base_wheel_step & ~(4096UL-1));
        if (mapped_base_wheel_step == (void *) -1) {
                printf("Can't map the Wheel Step memory to user space.\n");
                exit(0); }
        // get the address of the device in user space - an offset from the page base
        mapped_dev_base_wheel_step = mapped_base_wheel_step + (dev_base_wheel_step & (4096UL-1));
        *((unsigned long *) (mapped_dev_base_wheel_step)) = 0x00000002;  // write to pin 2
        // unmap the memory before exiting
        if ((munmap(mapped_base_wheel_step, (4096UL)) == -1) || (munmap(mapped_base_wheel_step,
        4096UL) == -1)) {
                printf("Can't unmap memory from user space.\n");
                exit(0); }
        close(memfd);
        return 0;  }
```

The code in Listing represents the minimum amount of work that needs to be done to use the peripheral. If this technique were used for all of STORBOT's peripherals, a significant amount time would be lost copying and pasting code rather than solving real problems. There are several features of this code that made writing device drivers reasonable:

1) Hard-coded memory addresses
    a. By hard-copying physical addresses of peripherals into our code, we commit ourselves to updating these values whenever the hardware we are working with changes.
    b. The kernel is already aware of the physical addresses of our peripherals through the .dts file (Device Tree File, see Appendix A). It is redundant to include these values a second time.
2) Error-prone Copying and Pasting

a. A significant amount of code would need to be copied and pasted from one peripheral to another. Copying, pasting, and understanding this much code would be difficult for new users (some team members began coding later in the project).
b. The team member who describes the hardware for the peripheral may understand that writing "0x00000002" to the register at the base of the peripheral sets pin two of five high, but team members who aren't familiar with the hardware would have to learn this. Bit-by-bit micro-management is always dangerous.

3) Don't Repeat Yourself
    a. It is standard software design practice to minimize redundant code. The code in Listing includes three error checking blocks and fourteen instances of the text "wheel step" in one form or another, each of which needs to be changed for a new peripheral. All this code should be written once for each peripheral.

Device Driver Interface

The wheel step device driver greatly simplifies its user space interface. The code in Listing 4 demonstrates the interface to the device driver:

Listing 4: User Space Interface to Peripheral – Including Device Driver

```
int main() {
        // open the device's driver interface
        fd0 = open("/dev/wheel_step_0", O_RDWR);
        if (fd0 == -1) {
                perror("open failed\r\n");
                rc0 = fd0;
                exit(-1); }

        // ioctl sends specific instruction to the driver
        rc0 = ioctl(fd0, WHEEL_STEP_WRITE_PIN_2, 1);
        if (rc0 == -1) {
                perror("read failed\r\n");
                close(fd0);
                exit(-1); }

        close(fd0);
        return 0;
}
```

This approach solves many of the problems with writing and reading directly to memory. The interface is significantly simpler, meaning it is easier to repeatedly use the code. Unfortunately, there are still two error checking blocks and two instances of the term "wheel step" in one form or another. In the interest of not repeating this code as well, we introduced wrapper classes for all our peripherals in our controller code (see Software Design).

WHEEL STEP DEVICE DRIVER

The wheel step device driver is listed separately in Appendix G. Several additional features to notice in the device driver are:

1) Automatic Peripheral Detection
    - At boot time, the "_init" function finds the number of wheel step peripherals in the kernel automatically (up to two in this case). Not only does the device driver

save users from hard-coding physical address, it takes care of finding the number of devices in the kernel.

2) Logical Pin-Setting
    - The device driver hard-codes the bit patterns that must be “OR”ed or “AND”ed with the peripheral’s control register to set certain pins. Remembering some of these patterns would be difficult even for the team member who wrote the hardware.

For other device drivers, like the DC_Motor, ioctl() supplied specific functions (forward and backward).

CONTROLLER CODE

At the highest level of abstraction, STORBOT was controlled by a large program written in C++. This program was broken down into the most important STORBOT functions.

DRIVER WRAPPERS

Every device driver that is involved in STORBOT requires error-checking, opening, closing, and reading functions. In the interest of minimizing redundant code, every device driver was wrapped in a C++ class (this also provides a cleaner interface to driver functions). As an example, the source code for the reflectance sensor class is shown in Listing .

Listing 5: Reflectance Sensor Class Wrapper

```
Refl_Sens::Refl_Sens(char* dev_identifier) {
        fd = open(dev_identifier, O_RDWR);
        if (fd == -1) {
                perror("refl_sens open failed\r\n");
                exit(-1); } }

Refl_Sens::~Refl_Sens() { close(fd); }

bool Refl_Sens::value() {
        rc = read(fd, rd_buf, 4);
        if (rc == -1) {
                perror("refl_sens read failed\r\n");
                close(fd);
                exit(-1); }
        if ((*(rd_buf)) < 50000) {
                return true;
        } else {
        return false; } }
```

With this setup, only the constructor and the value() functions are required to get all the information that is needed from the reflectance sensor.

HIGH-LEVEL FUNCTIONS

At the top-level of abstraction, STORBOT was controlled by a series of functions, one for each major task. The main controller code is included in Appendix I. The tasks that this program tackled were following a line, moving to a box height, and the overall function of getting a box. This controller code is meant to be as human-readable as possible.

UTILITY PROGRAM

During development, the team needed a quick solution for moving STORBOT's components out of the way or into a certain place for mechanical work or other goals. Rather than writing small programs to perform these tasks, one program provided general access to most of STORBOT's moving mechanical components. The program that provided this function was named '`mv_mot`' (move_motor). For example, to move the suction cup arm back for one second, one could type:

```
$> mv_mot -m arm -d 0 -s 1
```

…at the terminal. This program's source code is included in Appendix J.

GRAPHICAL USER INTERFACE

The graphical user interface, or GUI, was a very important part of this project. It served as a "barrier" of sorts between the end user, a factory or warehouse worker, and the internals of the robot control system. Since the robot was running Linux, and the serial port served as the terminal, any Linux commands sent over the serial port could control the robot. This allowed us to embed controls inside the code of the GUI (see Appendix H), and use it as a front end for the serial port. Thus, when the right sequence of buttons is clicked, and Linux formatted command is sent to the robot. The robot parses the command line for switches, as any Linux program does. In our case, the switches were shelf and row numbers. These were all tied in to each part using PartsList.txt, our inventory file. That file is also in Appendix H.

One important goal that was achieved through the GUI was the integration of other majors into the success of the project. Computer Science major Josh Vroom was a big help for the team. He put together the skeleton for the GUI, getting the general shape and function set up. After he handed it off to us before spring break, the team took it from there, extending the functionality of the serial port, adding a configuration and parts list file, as well as an inventory status bar.

All of this was fully functional by the end of the project. Another program was developed

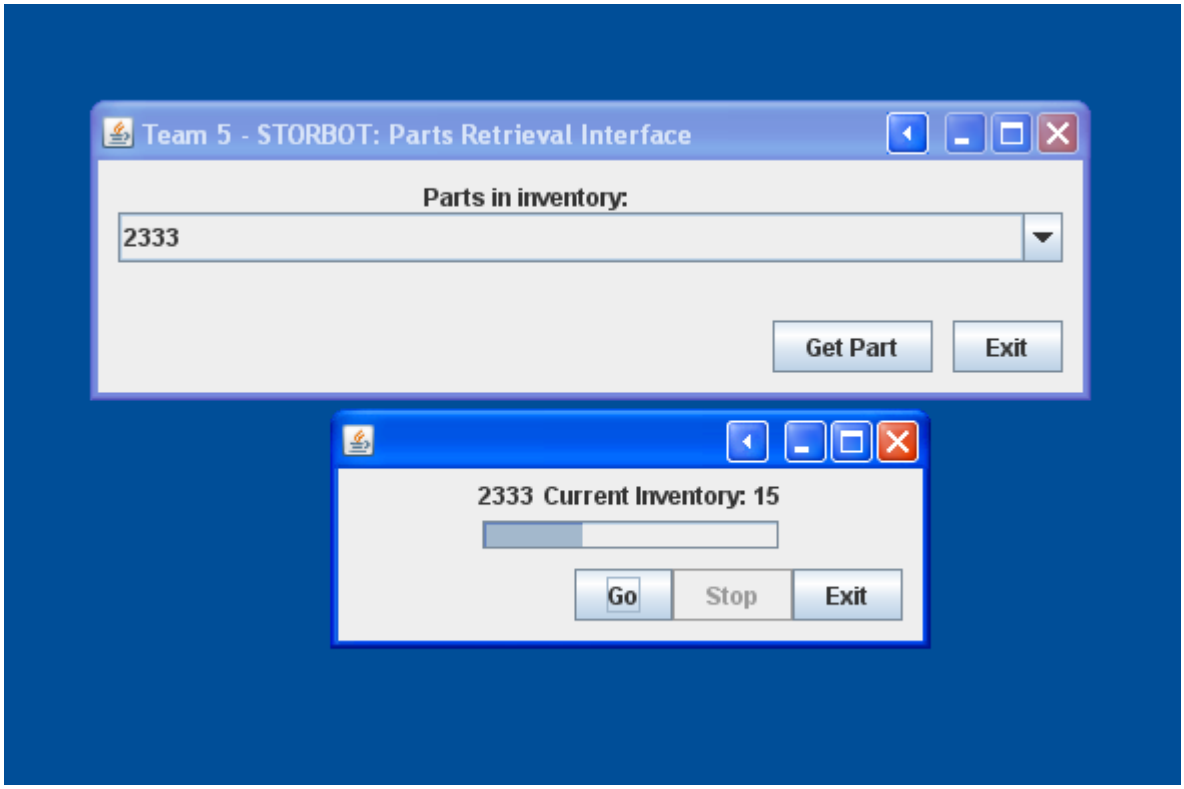


Figure 36: Graphical User Interface

from it for storing boxes, which simply increments the inventory value rather than decrementing it. There is area for continued work here, as the original idea was an isolated inventory

database server, which the robot host computer would be able to communicate with. Using this system, any number of robots could be working in the same warehouse or factory, and while they are all operating simultaneously, they will not interfere with each other with regards to inventory, since they are all working off the same database.

This leads into another area for expansion that might get touched on elsewhere in this report, and that is the addition of barcode scanners to the arm or shelf device on the robot. If each shelf had an associated barcode on it that described the items on the shelf, or if each box had a barcode sticker on it, then, while the robot is moving along the track and/ or up the elevator, it can be scanning all these barcodes. This serves two purposes. The first is the verification of the box it is going to pick up. If the barcode it reads at the location it is going to does not match up with the barcode value for the part it is expecting, it recognizes an error and aborts. Also, it serves as inventory checking and verification, as it can scan the barcodes of all boxes on the way out and the way back as it passes them.

DESIGN ALTERNATIVES

There were quite a few options that were investigated over the course of the project as alternatives to the final design. Furthermore, for high-volume production of the prototype, other cost-saving design changes would need to be made. For instance:

CONTROL ALTERNATIVES

For a high-volume run, the development board in the base of STORBOT would need to have its logic converted into a custom circuit, probably running on a microcontroller (and perhaps still using custom logic in an FPGA). The design could also make use of a PLC, if the programming done in C and C++ could be properly converted to ladder logic.

ELECTRICAL DESIGN ALTERNATIVES

An important change that would decrease the cost of the entire robot would be getting a new motor controller in the base of STORBOT. The Slo-syn motor controllers were very old (about 15 years old) and were high-end models when they were released. The vacuum pump on STORBOT's side could also be optimized for cost with further research, since it would be functional as long as it could still apply vacuum.

The team would also need to use a battery in the complete design, and design the relevant power circuitry for that configuration. Each component's power consumption would have to be minimized.

The sensors in the final design would need to be significantly improved. The distance sensor that determined the shelf heights would have to be replaced by an encoder on the elevator motor to get more accurate information on the elevator position. More safety components would have to be added all around the robot.

MECHANICAL DESIGN ALTERNATIVES

For a production prototype of STORBOT, the team would probably want to maximize the height of the robot, to reach as many shelves as possible. A telescoping arm design would also decrease the width of the robot as it traveled down aisles. The line following components at the bottom of the robot would need to be improved (to be more robust) and the rotation of the top of the robot would need to be fixed.

BUDGET

The team was able to estimate what funds were available early in the year, and planned to save costs in several areas to make the project feasible. Some production costs can be estimated from the team's prototype costs.

COSTS INCURRED

The budget breakdown for this project is shown in Table 5. After contributions at the beginning of the year from the Calvin BizPlan Program, Calvin Engineering Department, and DornerWorks Ltd., a local embedded engineering company, the team was able to complete a budget based on $1500 of available funds.

Table 5: STORBOT Project Expenses

Sept.	Oct.	Nov.	Dec.	Jan.	Feb.	Mar.	Apr.	
$300.00	$0.00	$0.00	$200.00	$0.00	$0.00	$0.00	$0.00	
$0.00	$0.00	$1,000.00	$0.00	$0.00	$0.00	$0.00	$0.00	
$300.00	$0.00	$1,000.00	$200.00	$0.00	$0.00	$0.00	$0.00	**$1,500.00**

pending

Nov 3: DW Donation Received

Sept.		Oct.		Nov.		Dec.		
XUP Virtex-II Pro Dev. System:	$299.00			*Pressure Sensor*	$7.35			
Protective Plexiglass Cover:	$19.95							
PmodCLS - 2x16 LCD:	$22.95							
Shipping - Dev. Board:	$15.00							
	$356.90		**$0.00**		**$7.35**		**$0.00**	**$364.25**

Jan.		Feb.		Mar.		Apr.		
Threaded Rod	$11.00	Vacuum Pump	$161.59	Digkey Misc.	$80.01	Wireless RS-232	$88.00	
Kingston 256 MB RAM	$14.99	Suction Cups	$51.30	Two Geared DC Motors	$110.00	More Digikey Misc.	$55.27	
512 MB CF Card	$8.88	Wheel Motor	$40.93	Gearing for Drivetrane	$34.68	Plexiglass	$57.02	
				Linear Bearings	$32.36	Electrostatic Mat	$18.95	
				Misc. Metal & Plastic	$29.20	Pump Upgrade	$50.00	
				Caster Wheels	$18.15	Wheels	$35.06	
						Timing Belts	$8.16	
						Wheels and Bearings	$7.41	
						Felt Paper and Duct tape	$39.71	
						Nuts and Pipe fittings	$11.64	
						Roller Bearings	$26.31	
						CCD Barcode Scanners	$9.35	
						SparkFun Order Misc.	$33.12	
						Map Sensor	$10.90	
						E-Stop Button	$15.00	
						Turndisc Pulleys	$30.03	
						Turndisc Belt/Pulley	$11.40	
	$34.87		**$253.82**		**$304.40**		**$507.33**	**$1,100.42**

Balance:	$35.33

However, a large percentage of STORBOT's true cost is hidden in donated or borrowed materials/components.

ESTIMATED TOTAL COST

Estimating the cost of donated materials and components is difficult, given that many components may have lost value over time or were worth less when the team obtained them as scrap. According to Chuck Holwerda, the slo-syn stepper controllers are currently worth about $1000 each, and the motors associated with them are worth about $200 each. There is a new 24V, 10A power supply in the base worth $239, as well as a $70 computer power supply.

Your shopping cart contains the following items:

Stock Number	Item Description	Length	Quantity	Status	Price Each	Totals	Change Quantity/Size
P114	1/4 inch THICK A36 Steel Plate	1 X 4 Ft.	1	In Stock	$55.32	**$55.32**	Change
R134	3/4 inch Dia. Round Bar Hot Rolled A-36 Steel Round	10.0 Ft.	1	In Stock	$16.80	**$16.80**	Change
P3316T6	.190 (3/16) thick 6061-T651 Aluminum Plate	2 X 4 Ft.	1	In Stock	$162.48	**$162.48**	Change
P312	.500 (1/2) thick 6061-T651 Aluminum Plate	1 X 1 Ft.	1	In Stock	$49.00	**$49.00**	Change
P312	.500 (1/2) thick 6061-T651 Aluminum Plate	1 X 1 Ft.	1	In Stock	$49.00	**$49.00**	Change
F4182	1/8 X 2 6061 Aluminum Flat	8.0 Ft.	1	In Stock	$14.08	**$14.08**	Change
T3118	1 x 1 x 1/8 wall 6063 Aluminum Square Tube	8.0 Ft.	1	In Stock	$24.40	**$24.40**	Change
A3114	1 X 1 X 1/4 Aluminum Angle 6061-T6 Aluminum Structural Angle	8.0 Ft.	1	In Stock	$24.48	**$24.48**	Change
T3118	1 x 1 x 1/8 wall 6063 Aluminum Square Tube	21.0 Ft.	1	In Stock	$38.43	**$38.43**	Change
P312	.500 (1/2) thick 6061-T651 Aluminum Plate	1 X 2 Ft.	1	In Stock	$98.00	**$98.00**	Change
P3316T6	.190 (3/16) thick 6061-T651 Aluminum Plate	2 X 2 Ft.	1	In Stock	$81.24	**$81.24**	Change
Quantity Discounts Saved You! $58.02 (Learn More)				Total:		$613.23	Empty Cart
Hot Tip! For maximum discount put all items in Cart before you Calculate Shipping!	CALCULATE SHIPPING			<< CONTINUE SHOPPING			CHECK OUT >>

Figure 37: Metals Depot Cost Estimate

All of the metal that was obtained from the shop is worth around $600, according to Metals Depot (see Figure 37), a large-scale online metal vendor. The total cost of the prototype is approximated in Table 6.

Table 6: Prototype Cost Estimate

Line Item	Cost
Budget	$1,500.00
Power Supplies	$300.00
Slo-Syn Controllers	$2,000.00
Motors	$500.00
Metal	$600.00
Total:	$4,900.00

Production Model Estimates

If STORBOT were to go into full scale production, many of these costs would drop, but the system would be redesigned to such a large degree that the prototype would be a bad base for estimating total cost. Factors that would change in a production model:

1) Custom circuit board rather than development board (perhaps still including an FPGA).
2) Cheaper motor controllers in the base (Slo-syns were quite old, and very high-end)
3) Battery Pack and Power Circuitry
4) Telescoping Arm Design
5) Smaller Vacuum Pump

Acknowledgments

STORBOT would like to thank the following people and groups for their many contributions to the success of this project throughout the academic year.

Robert Bossemeyer: Team Advisor. Professor Bossemeyer was patient with the team through a less than productive first semester, and provided good pointers for keeping the plan on track and keeping things organized.

Steven Vanderleest: Industrial Mentor. Steve, a professor at Calvin College and a vice-president at DornerWorks, Ltd. served as both a liason to the donor and a valuable industrial mentor. He heavily influenced the direction of the project in its early development, pushing it in a more mechanical and "ground-up" direction, which has proved to be a very educational and exciting experience.

Nicholas Goote: Industrial Mentor. Nick, another employee of DornerWorks, provided guidace in the early planning stages of the project, as well as advice with regards to the development board chosen for this prototype.

DornerWorks: Donor. DornerWorks generously donated funding to the team for extra materials the project may require, and it was greatly appreciated, and came in very handy.

Phil Jasperse: Shop attendant. Phil was a very valuable asset to the team throughout the entire second semester, as the real construction began. He was always willing to listen as we bounced ideas off of him, and also willing to kindly point out some of the potential weaknesses in the mechanical design.

Mike Barber: Industrial Consultant. Mr. Barber met with the team in the middle of the first semester to review the commercial viability of the project and suggest areas for improvement when it came to financial justification of the project idea.

Xilinx University Program: Donor. The Xilinx University Program donated a Virtex-5 development board to the team. Though it was not used in the final product, it helped guide the design choices for which processor and operating system would be best for our project.

Chuck Holwerda: Electronics Shop attendant. Chuck made his shop equipment readily available to the team, and provided access and instruction for making the PCBs the team used in the final product. He also equipped us with two slo-syn motor controllers we used to propel the base.

BIBLIOGRAPHY

Yaghmour, Karim. Building Embedded LINUX Systems. Sebastopol, CA: O'Reilly & Associates, Inc, 2003.

Paul, Richard P. SPARC Architecture, Assembly Language Programming, and C. 2nd ed. Upper Saddle River, NJ: Prentice Hall Inc, 2000.

Uni-directional DC Motor Controllers. Ocean Controls. 10 Nov. 2008 <X:EngineeringTeamsTeam5websitesdc_motor_controllers.htm>.

Robotic Gripper Sizing: The Science, Technology and Lore. 1996. ZAYTRAN, Inc. 3 Nov. 2008 <X:EngineeringTeamsTeam5websitesgripperinfo.htm>.

Sharp IR Rangers Information . 10 Oct. 2008. Acroname Robotics. 3 Nov. 2008 <X:EngineeringTeamsTeam5websitesirsensors.html>.

Robot Sensors. 2004. Summerour Robotics Corp. 4 Nov. 2008 <X:EngineeringTeamsTeam5websitessensorsandcontrollers.htm>.

Baskiyar, S. "A survey of contemporary real-time operating systems." Informatica 29.2 (2005): 233-40.

Fowler, Kim. "Part 2: Real-time operating systems (RTOS)." IEEE instrumentation & measurement magazine 5.2 (2002): 48-49.

Stankovic, John A. "Real-Time Operating Systems." Real-Time Systems 28.2-3 (2004): 237-53.

Shaw, Alan C. Real-Time Systems and Software. New York: John Wiley, 2001.

Wikipedia. 2 Dec. 2008. 2 Dec. 2008 <www.wikipedia.org>.

PowerPC Licensing. IBM. 2 Dec. 2008 <http://www-03.ibm.com/technology/power/licensing/index.html>.

OpenSPARC UInivserity Program. 8 July 2008. Sun Microsystems. 2 Dec. 2008 <http://www.opensparc.net/edu/university-program.html>.

XUP V2P Development Board. Digilent, Inc. 2 Dec. 2008 <http://digilentinc.com/Products/Detail.cfm?NavTop=2&NavSub=453&Prod=XUPV2P>.

ISE Design Suite 10.1. Xilinx. 2 Dec. 2008 <http://www.xilinx.com/products/design_resources/design_tool/>.

Ermer, Gayle and Vanderleest, Steven H. Using Design Norms to Teach Engineering. June 2002. Calvin College. 2 Dec. 2008 <http://www.calvin.edu/academic/engineering/faculty/svleest/abstracts/asee02.htm>.

Automated Storage & Retrieval System. Westfalia USA. 2 Dec. 2008 <http://westfaliausa.com/>.

Kiva Systems. Kiva Systems. 2 Dec. 2008 <http://www.kivasystems.com/>.

The GNU General Public License. 29 June 2007. GNU. 2 Dec. 2008 <http://www.gnu.org/copyleft/gpl.html>.

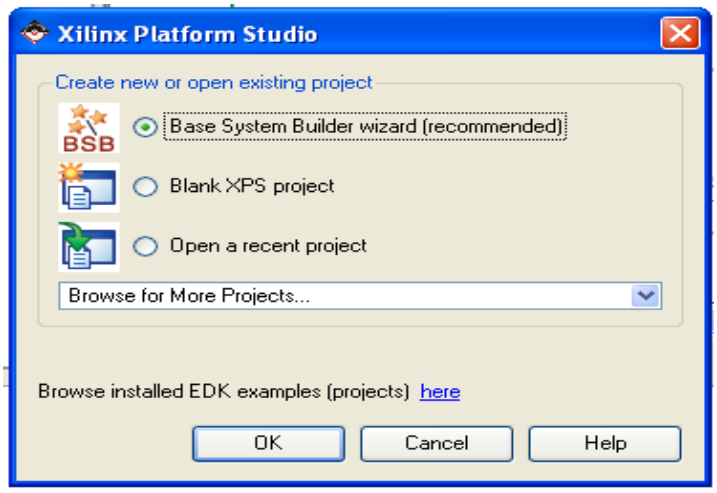
Bureau of Economic Analysis. The U.S. Government. 2 Dec. 2008 <http://www.bea.gov/>.

U.S. Department of Labor. The U.S. Government. 2 Dec. 2008 <http://www.dol.gov/dol/topic/wages/index.htm>.

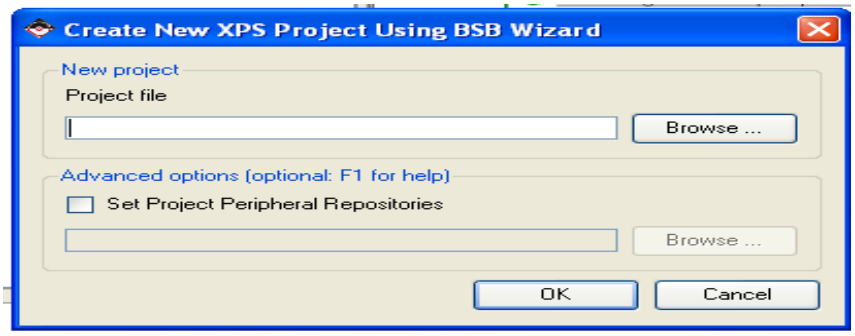
APPENDIX A –SETTING UP XILINX PROJECT FOR LINUX TUTORIAL

HARDWARE SETUP

1) Start 'Xilinx Platform Studio' (XPS) on your desktop.
   - After a short wake-up, you should get the following display:

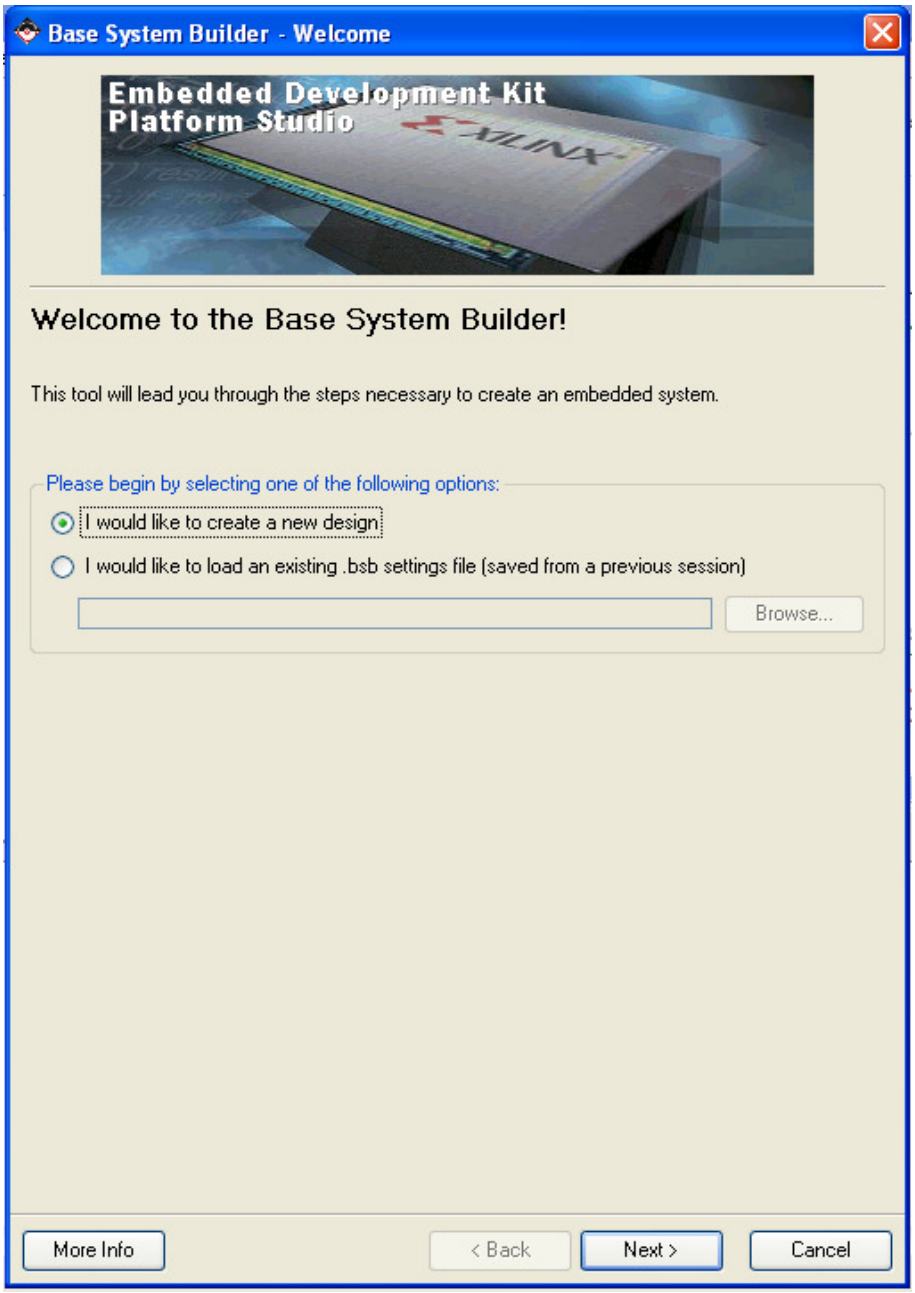


   - You will want to use 'Base System Builder' (BSB) to get the base components into your system, and add custom peripherals later.
2) Click on OK.
   - The following prompt will appear:



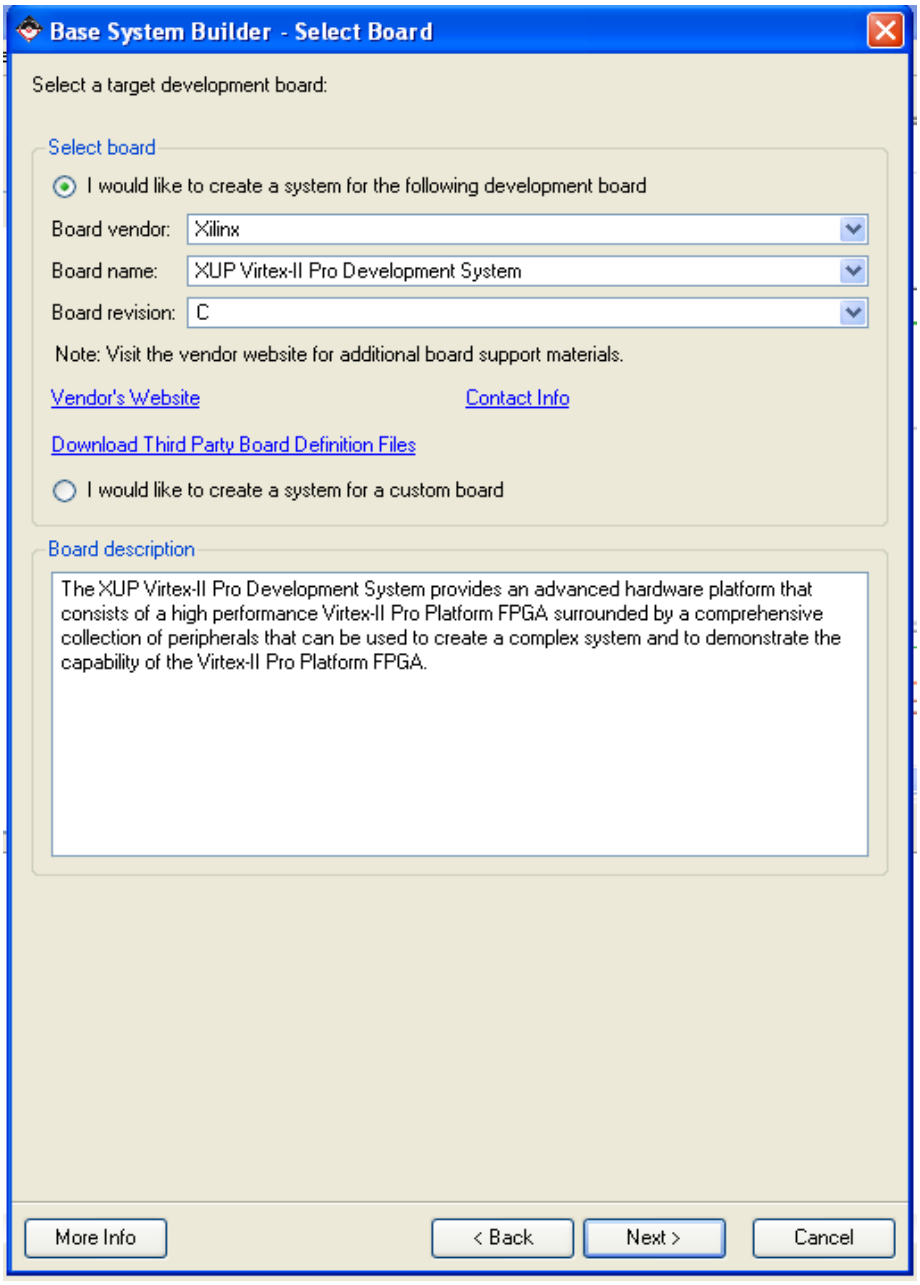
   - Click 'Browse…' without naming the project file. In the file browser, choose a location for your new project. Create a new folder in this location, your choice for the name. Save the default .xmp file (system.xmp) in this folder. All the project's Xilinx files will be generated in this folder, next to system.xmp.
   - Check the "Set Project Peripheral…" box. Choose the board definition files for the XUPV2P, available at http://www.xilinx.com/univ/xupv2p.html. You must choose the lib folder within these files. NOTE: there can be NO spaces in the path to this file, or the board will not show up in dropdown menu in the step below. There can also not be (or [, and probably many other symbols are bad, too. I would recommend that the path to your project is also clean of these things.

3) With both fields in the prompt above filled, hit OK.
   - The following prompt will appear:



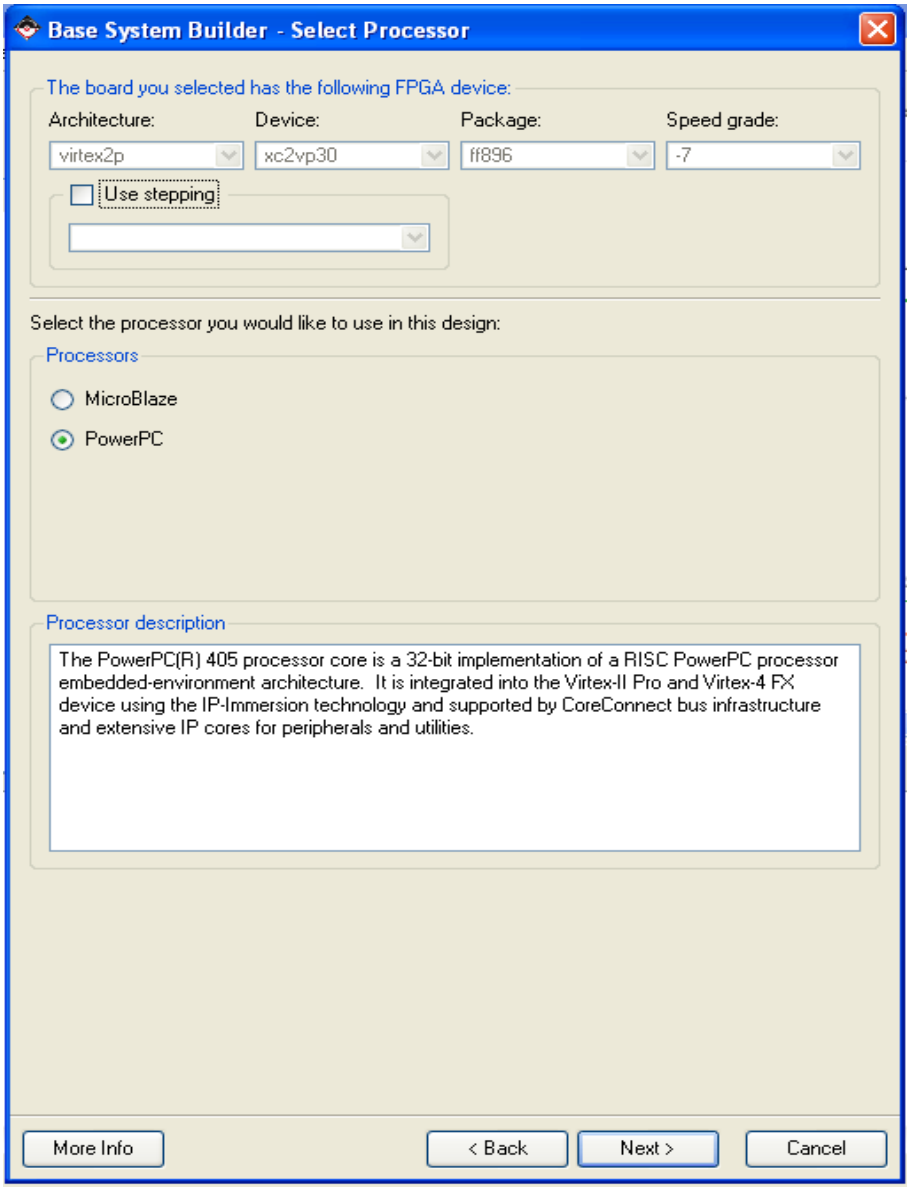
   - If you don't want to go through the entire BSB every time you create a similar project, use the existing .bsb option here.

4) Hit "Next" to continue:
   - The following prompt will appear:



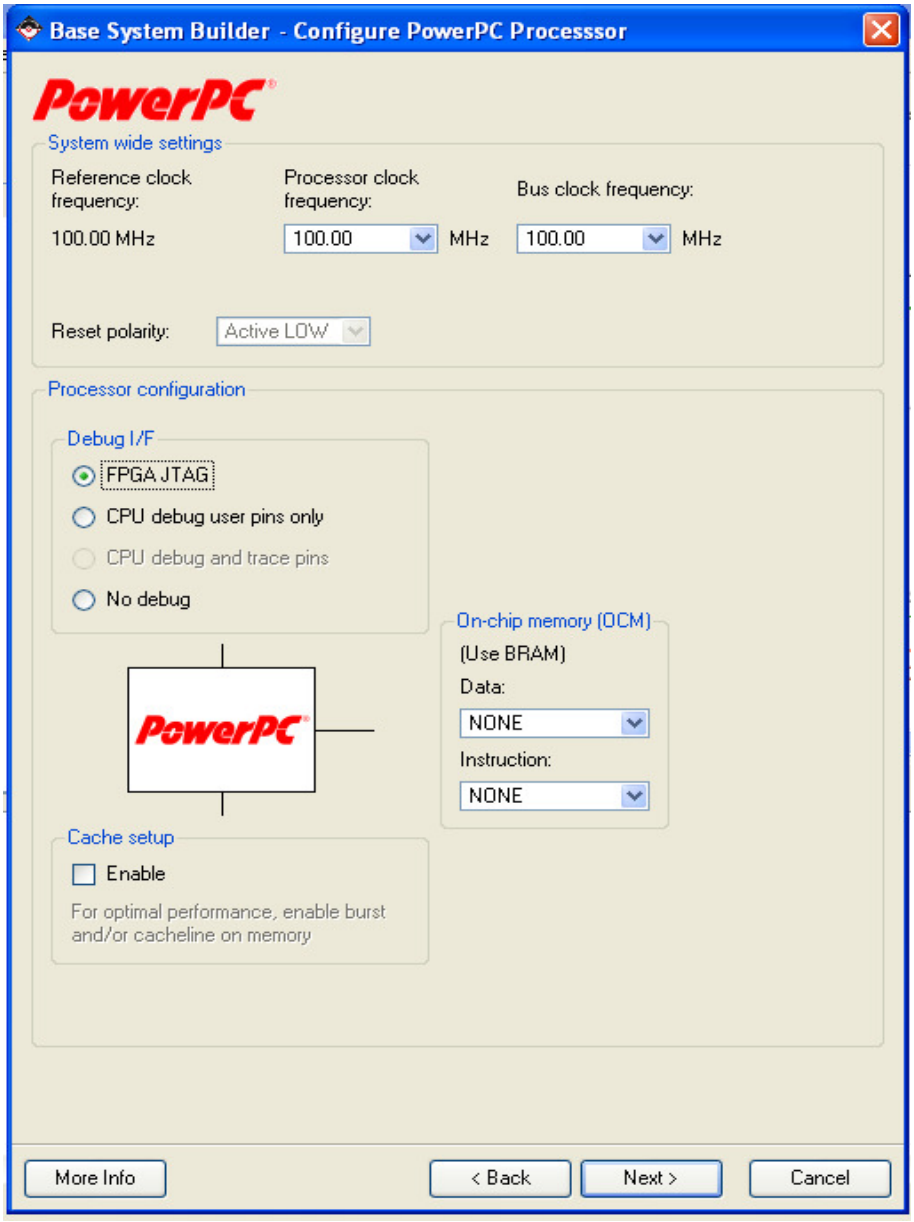
- As I have done, select “Xilinx”, then “XUP Virtex …”

5) Hit “Next” to continue:
   - The following prompt will appear:



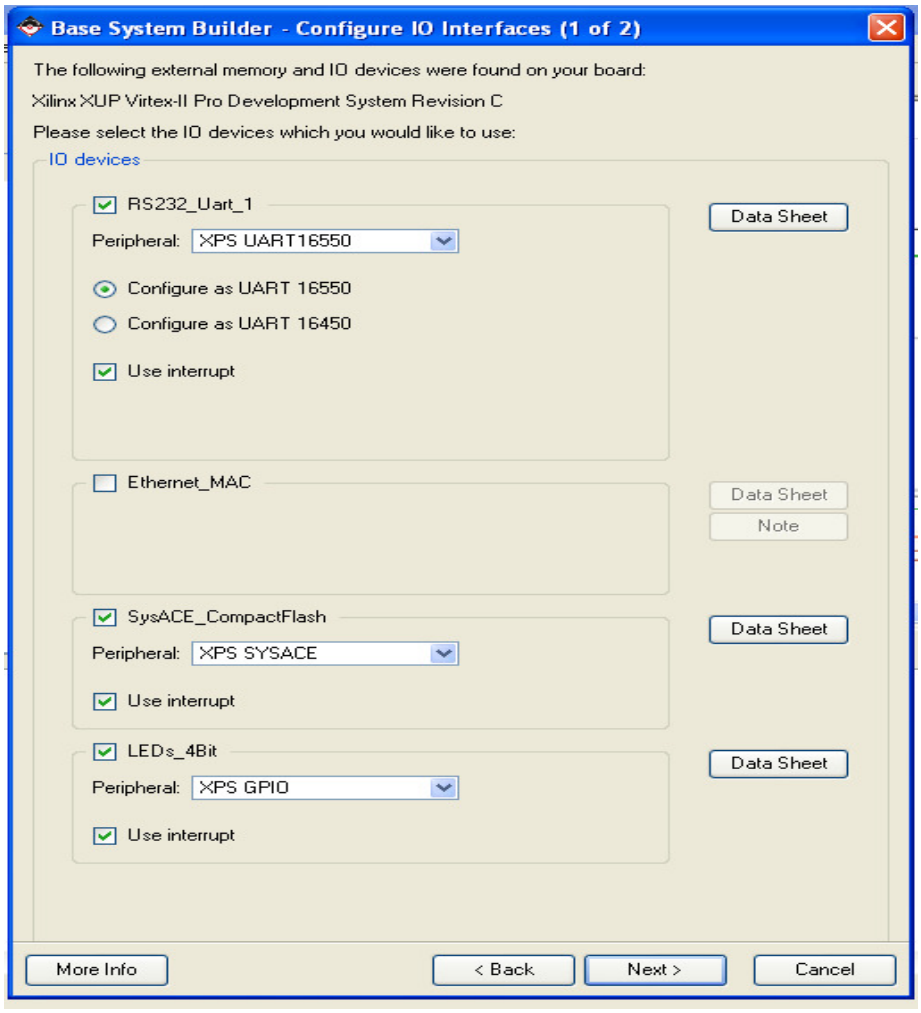
- Select the PowerPC processor.

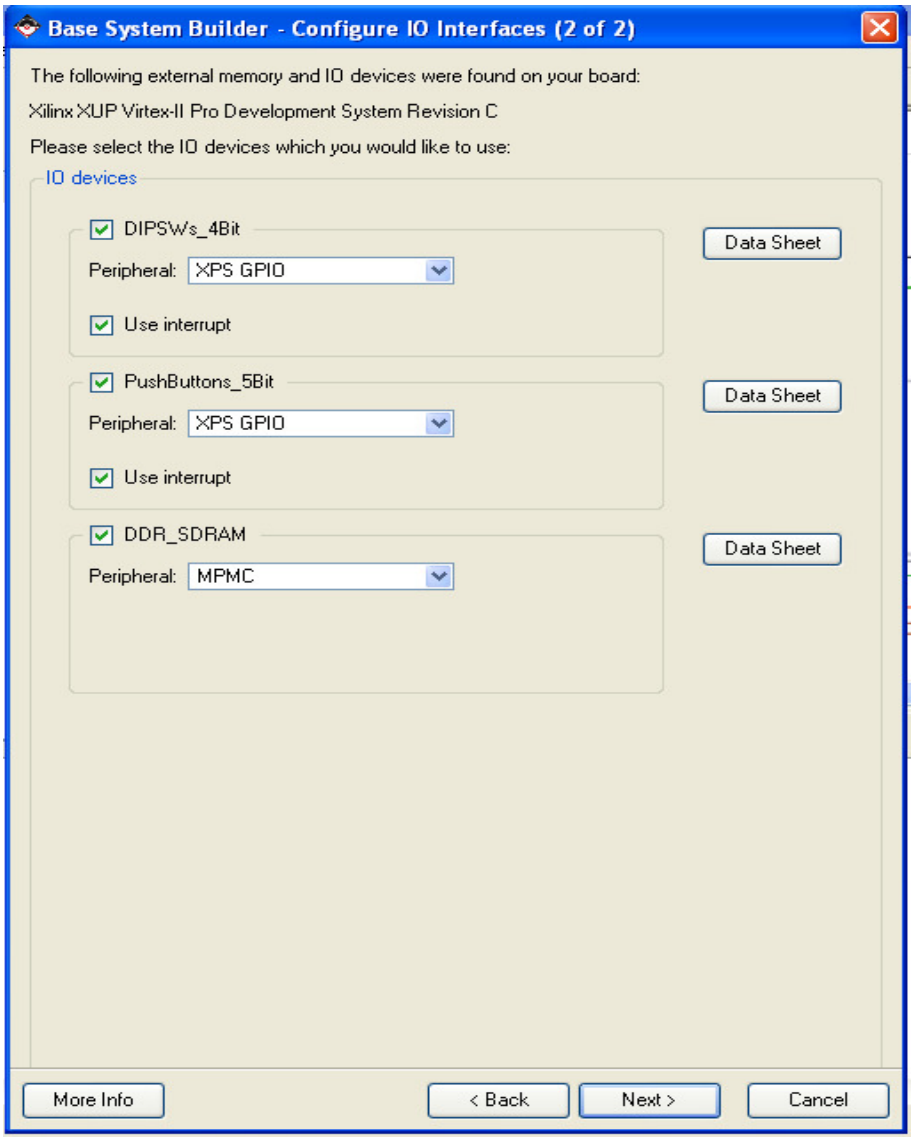
6) Hit “Next” to continue:
   - The following prompt appears:



- Leave all settings as they are.

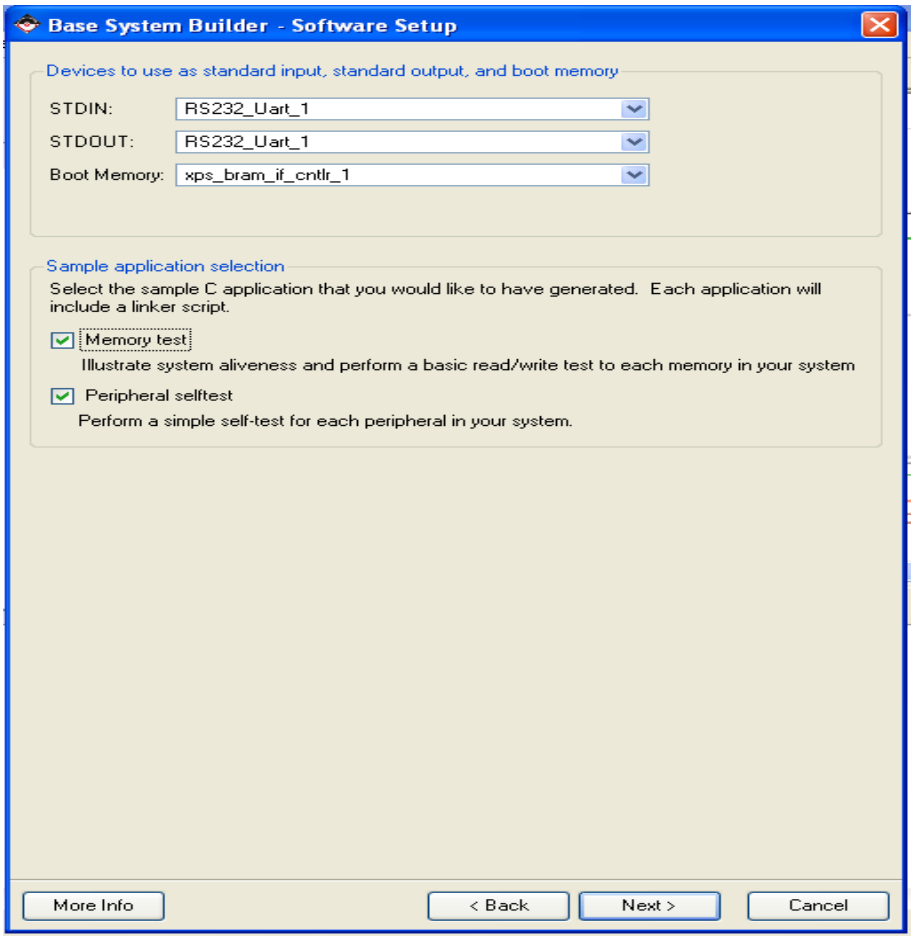
7) Hit “Next” to continue:
   - The following prompts appear:


Base System Builder - Configure IO Interfaces (1 of 2)
The following external memory and IO devices were found on your board:
Xilinx XUP Virtex-II Pro Development System Revision C
Please select the IO devices which you would like to use:
IO devices
RS232_Uart_1
Peripheral: XPS UART16550
Configure as UART 16550
Configure as UART 16450
Use interrupt
Data Sheet
Ethernet_MAC
Data Sheet
Note
SysACE_CompactFlash
Peripheral: XPS SYSACE
Use interrupt
Data Sheet
LEDs_4Bit
Peripheral: XPS GPIO
Use interrupt
Data Sheet
More Info
< Back
Next >
Cancel



- Set the peripherals and interrupts as shown above.
- The RS232_Uart_1 was changed to a UART16550 (it is the default UART in the kernel configuration, so we will use it rather than trying to switch to the UARTLITE).
- Interrupts need to be added to all peripherals if they are going to be recognized by Linux (likely through a probing process).

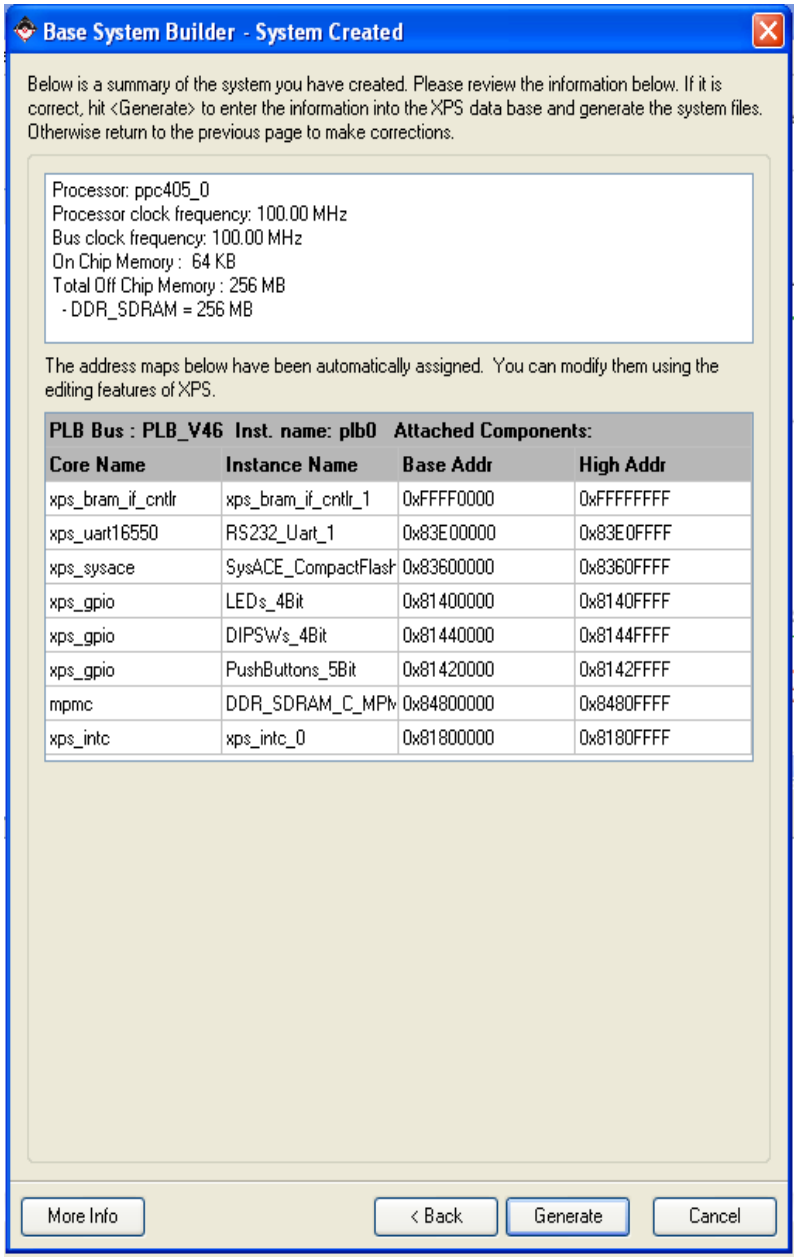
8) Hit “Next” (through both windows) to continue:
   - The next window has only one pull-down menu; choose 64 KB of Block RAM.
9) Hit “Next” to continue:
   - The following prompt appears:



- De-select both the memory test and the peripheral test (these only show off the features of the peripherals and demonstrate how to write a C program for the board w/o Linux).

10) Hit “Next” to continue:

- The summary prompt will appear:

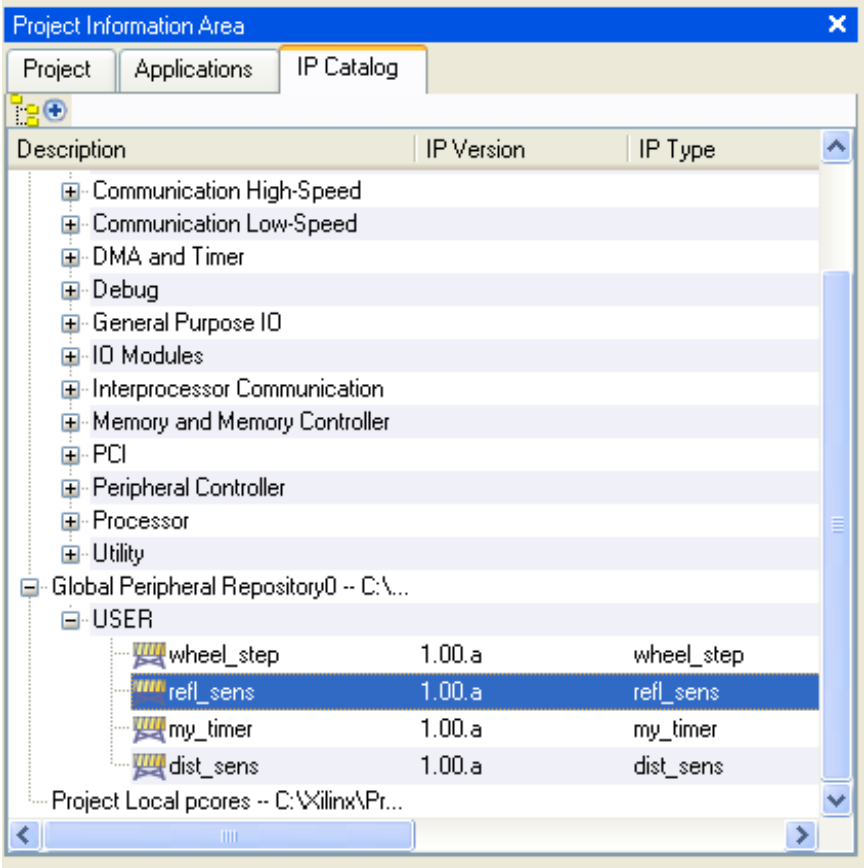


11) Click “Generate” to build your system.

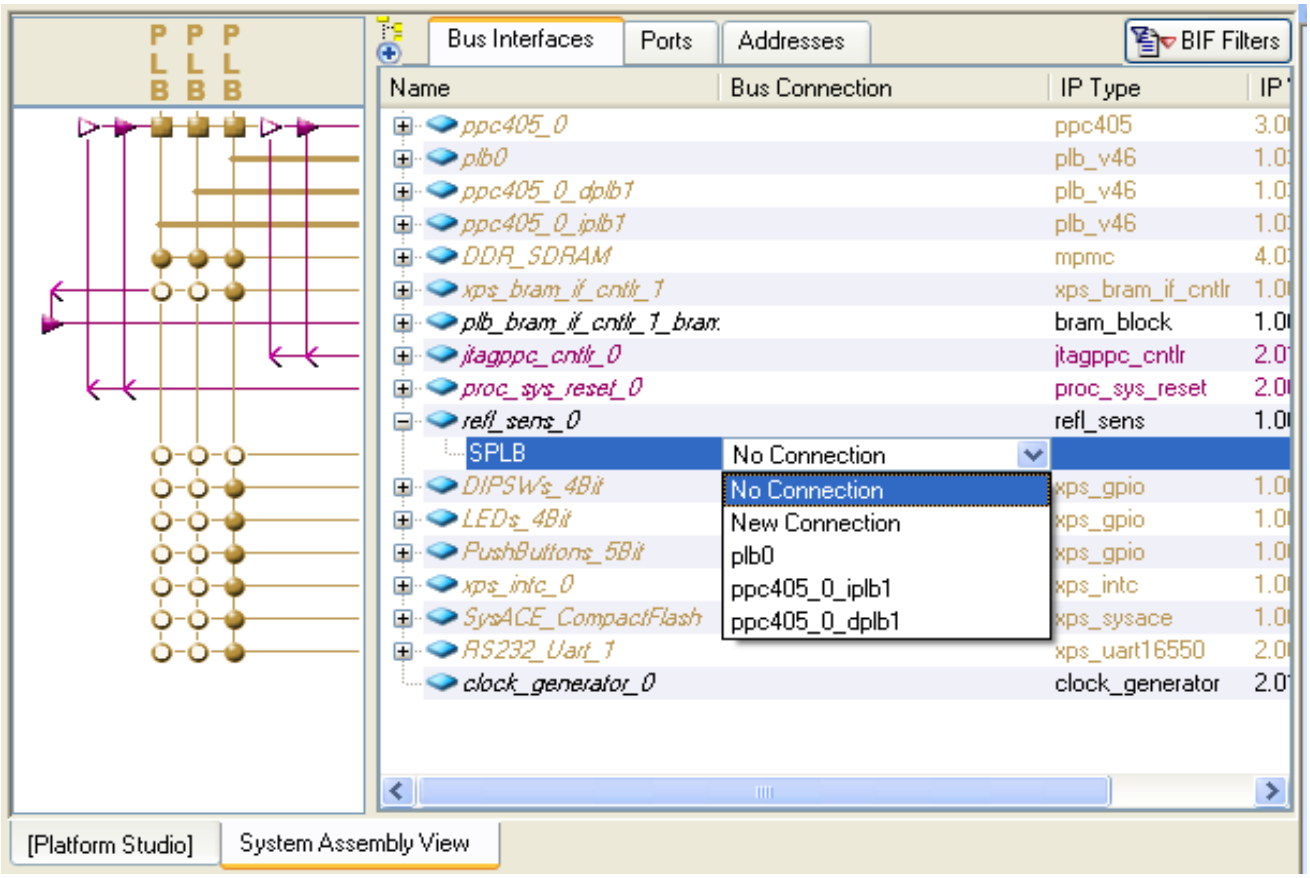
- A final prompt appears asking where to save the .bsb file and describing where files were put.

12) Hit “Finish.”

- You now enter the main XPS program. You should add any peripherals you would like to the project at this point. Note that this is completely optional. We will demonstrate adding the custom refl_sens peripheral.
- Find the project information area:



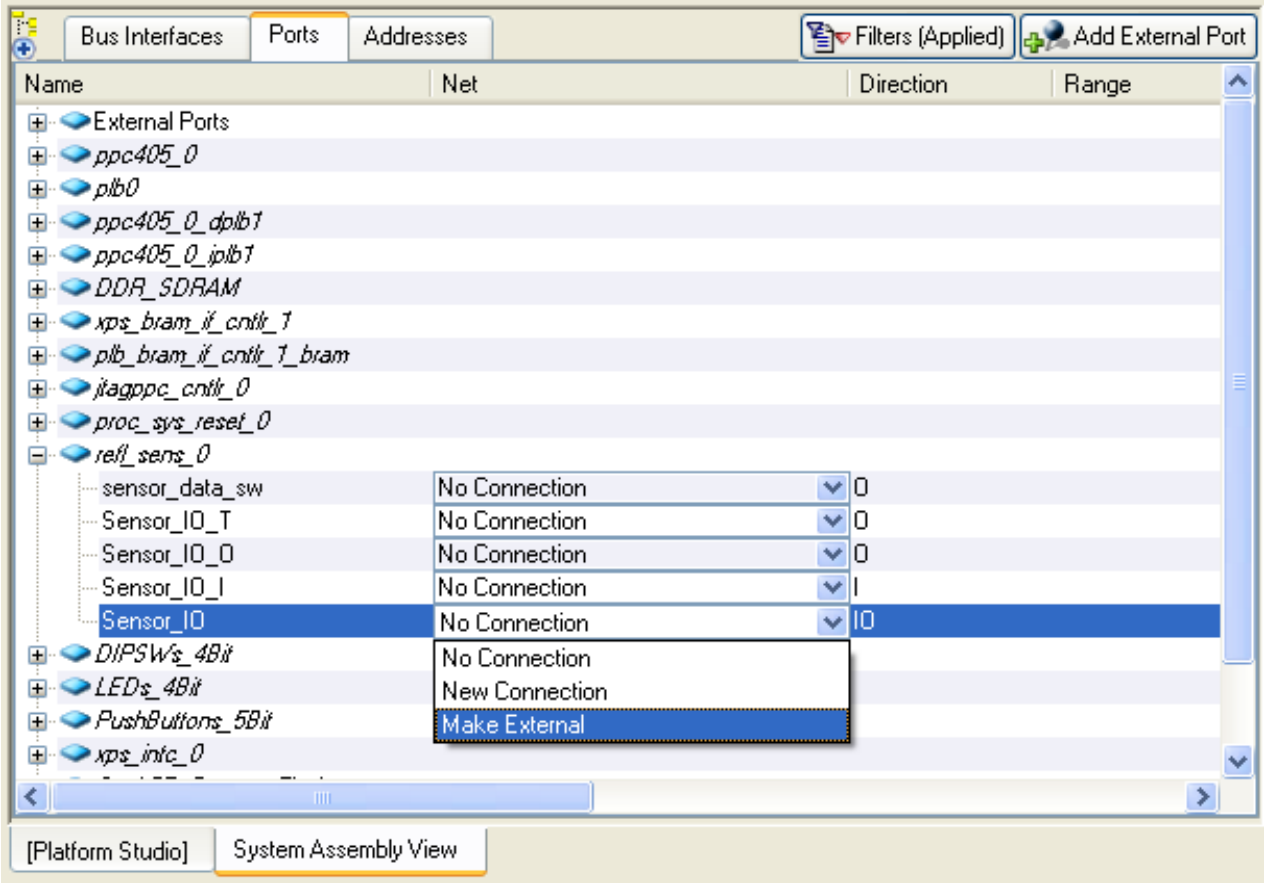
13) Hit the “IP Catalog” tab in the “Project Information Area.” Under “Global Peripheral Repository” find the refl_sens peripheral and double-click to add it to the project.
    - The refl_sens will appear on the “Bus Interfaces” tab in the “System Assembly View” tab in the area to the right:



14) From the drop-down list that is shown, select plb0 to attach the peripheral to our processor bus.

15) Click the “Ports” tab, next to the “Bus Interfaces” tab.

- Find the refl_sens_0 in the list:



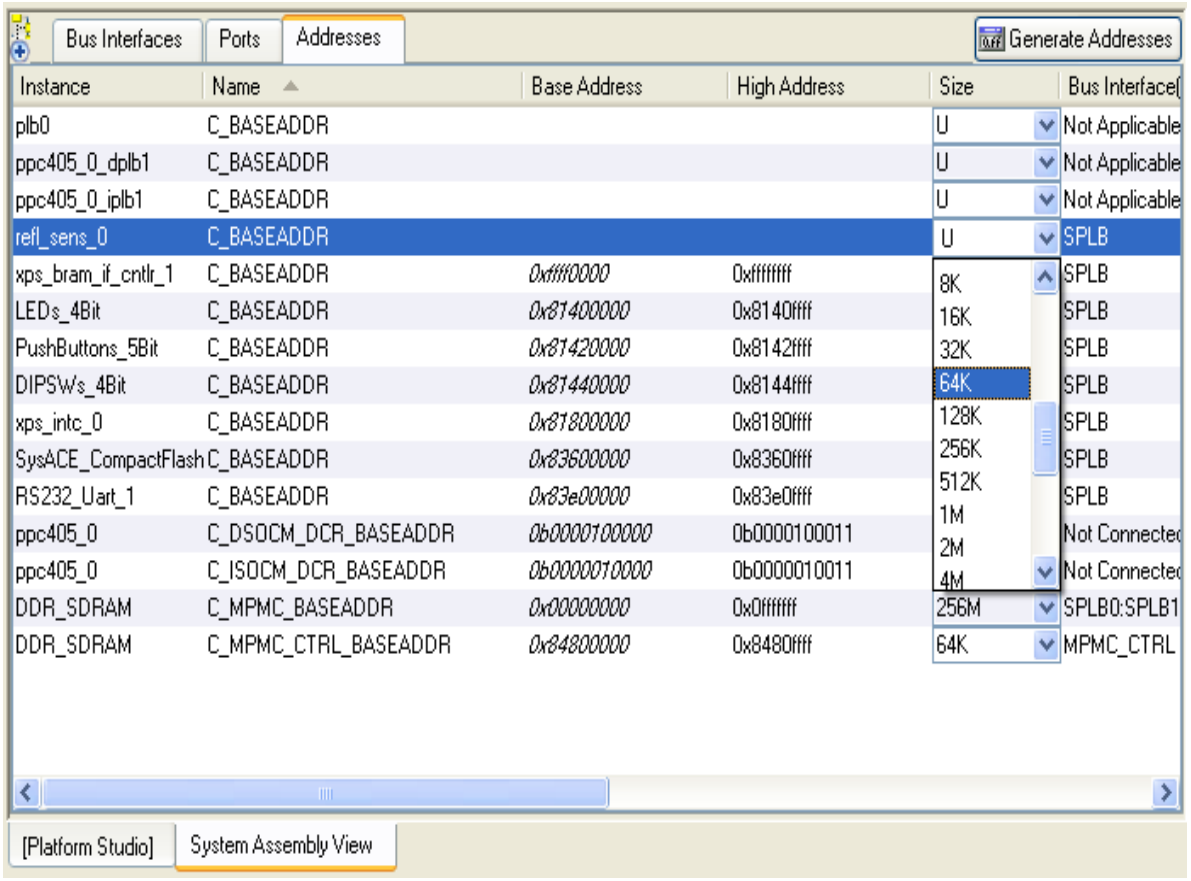
- At this point we are deciding which signals on this peripheral we want to attach to other places in the system. We could name a signal here and connect it to another peripheral; in our case, we would like to make signals go completely out of the board.

16) Choose “Make External” from the drop-down list for the Sensor_IO signal.

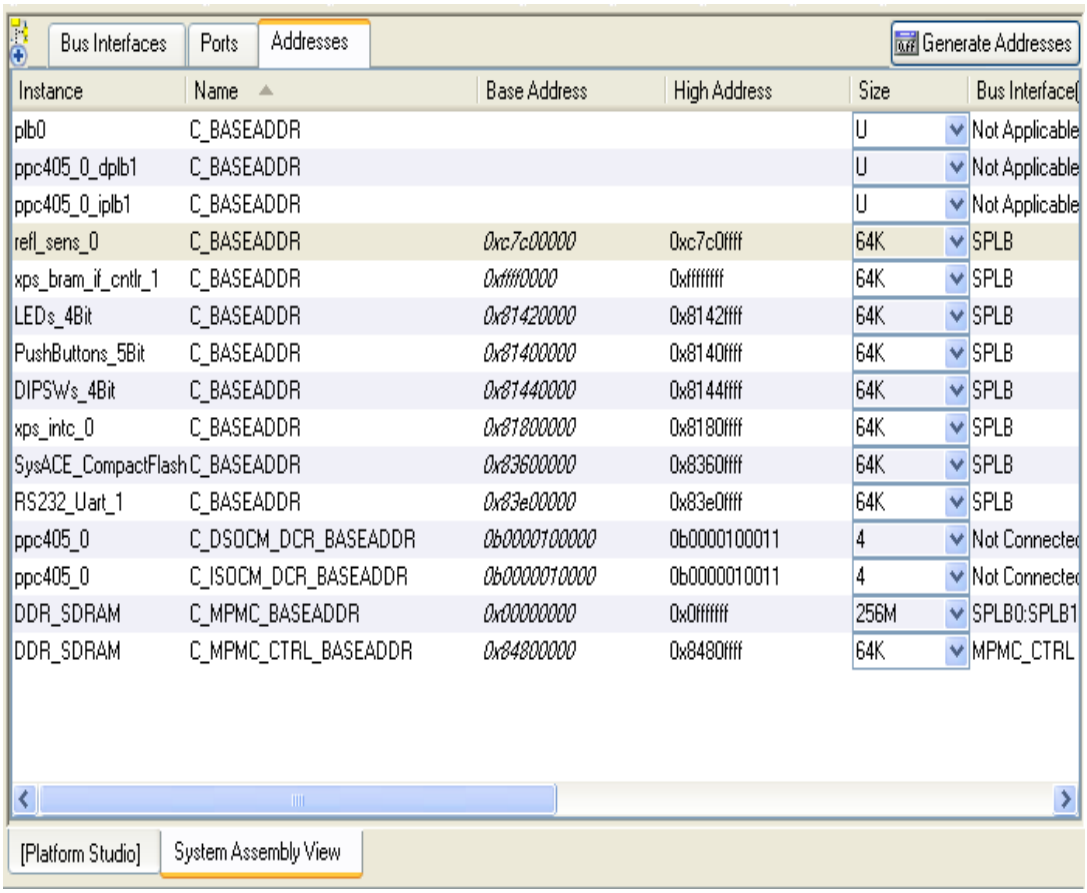
- We could also choose to send the other pins out of the system / out of the board, but in our case none of the other signals are necessary for the refl_sens to work (the signals labeled with _T, _O, and _I are always created alongside an inout pin, which Sensor_IO is).
- Once the signal is external, it will appear at the top of this window under “External Ports.” You have specified you want the signal to leave the system, but not where you want to go to specifically (i.e., which metal pin?). We will deal with this in a few steps.

17) Click the “Addresses” tab.

- Find the refl_sens:

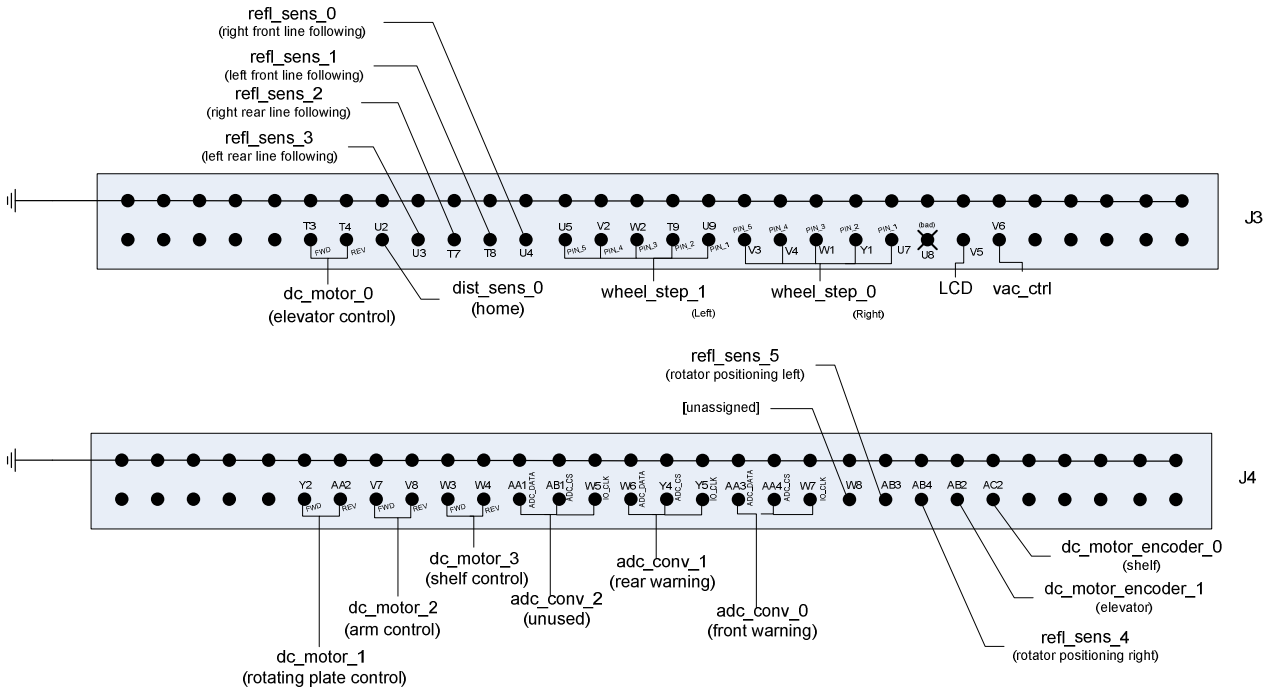


18) For the size of the refl_sens, choose 64K (this is the space it occupies in the system memory map).
19) Click “Generate Addresses” to have XPS automatically choose where this 64K will be and move the other peripheral appropriately.
    - Our final system memory map:

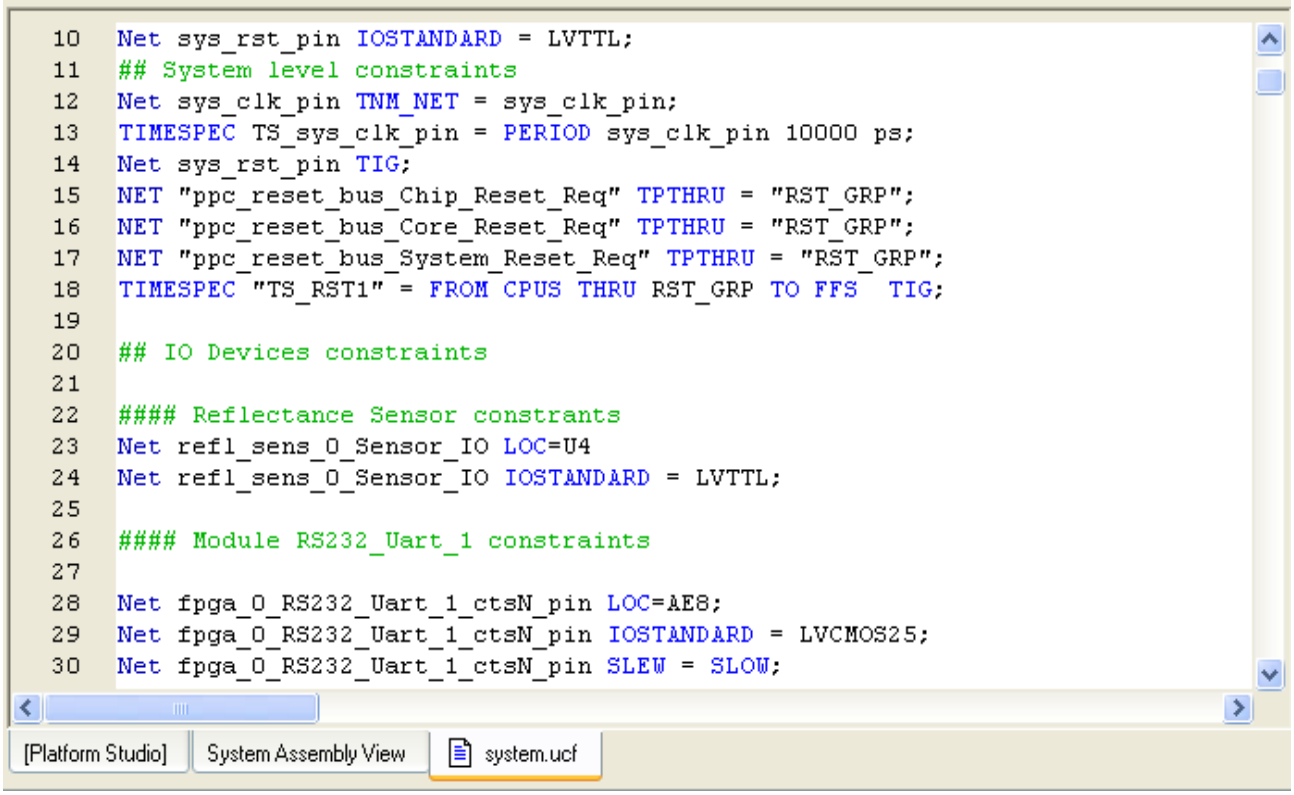


20) You assigned external pins on your relf_sens; but where are those pins physically on the board, that you could read with a DMM? We need to tell XPS where to put the signals we have (only Sensor_IO in this case).

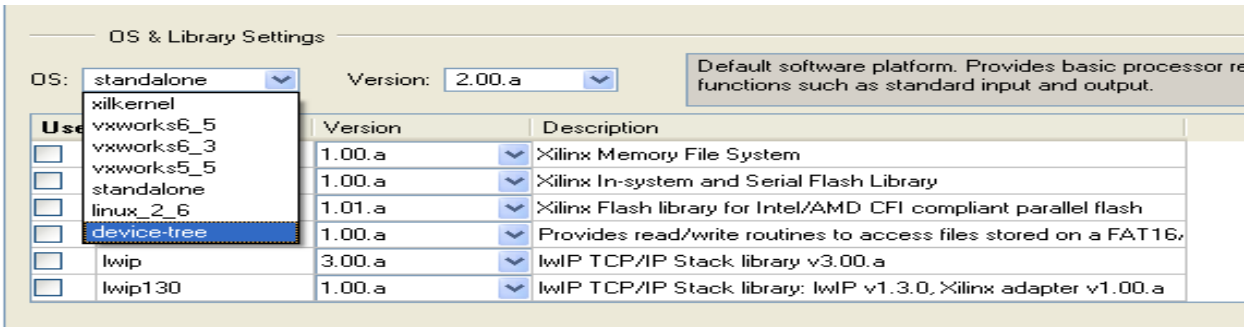
- The schematics for the XUPV2P are available at http://www.xilinx.com/univ/xupv2p.html. You can read into these schematics to find all the places you could put your signals; but for now we will assume that you are using either J3 or J4. Below are the pins that my group is using to control our robot, STORBOT. It includes the FPGA pins (T3, T4, U2 …) that we need to tie to here in XPS.



21) Under the “Project” tab in “Project Information Area” to the left, under Project Files, double-click on “UCF File: data/system.ucf”
    - In this file we will tie our “External Ports” to physical pins. We have only have one pin to tie.
    - Here is how I changed my .ucf file:

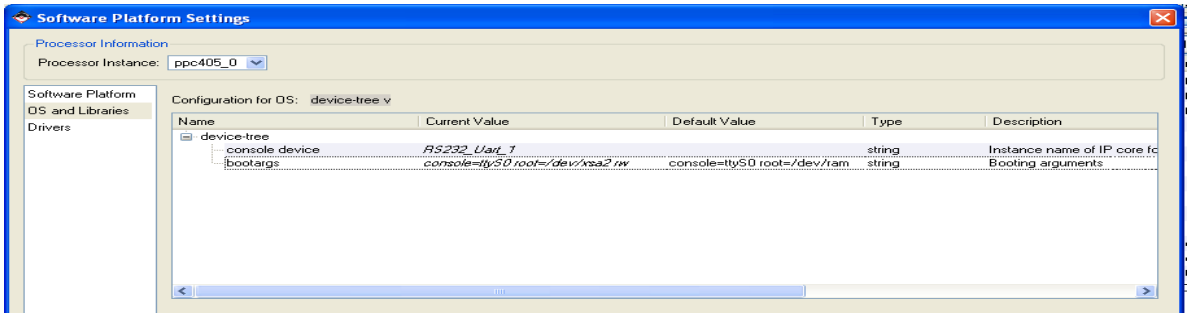


22) Type these changes into your .ucf file.
    - We have everything setup that XPS will need to generate the hardware for our system. However, Linux needs to know about the hardware that we are generating here, and some parameters of the system. XPS has software in place to generate this hardware description, a .dts file.
23) Go to http://xilinx.wikidot.com/device-tree-generator to read about the device tree generator, if you would like. Then go to http://git.xilinx.com/ and (unless you want to use git) at the bottom of the page, under the Project Name device-tree.git choose the link “tree.” Then, at the top of the next page hit “snapshot” to download the device tree generator code.
24) Run tar –xzvf on the downloaded file (in cygwin if on a Windows machine) and move the folder “bsp” in the unzipped files to your project folder (at the same level you put system.xmp).
25) Close XPS, then restart XPS and reopen your project (Open a recent project -> hit OK -> find your system.xmp file).
26) In XPS, under the Software drop-down menu, choose “Software Platform Settings…”
27) Under OS & Library Settings, choose device-tree:



28) On the left, hit the OS and Libraries button/tab.

- The following prompt will appear:



- The “Current Value” for the console device and bootargs will be empty for you.

29) Type in RS232_Uart_1 for the console device.

30) Under bootargs type in “console=ttyS0 root=/dev/xsa2 rw”

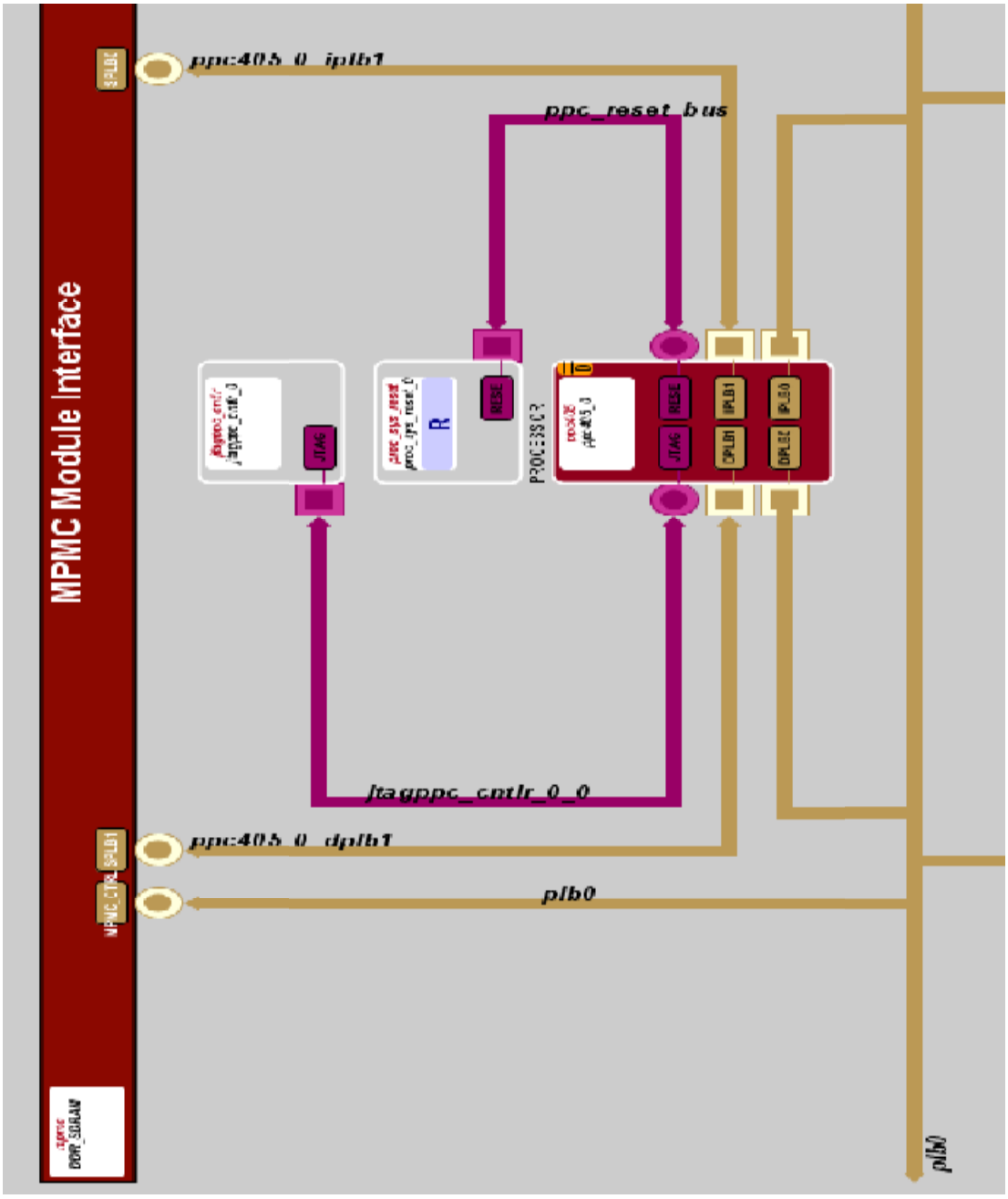
31) Hit ‘OK’ to close the window.

32) Select Software -> Generate Libraries and BSPs.

- This step should go relatively fast – we are getting just one file we need from it.

33) We have described all of our hardware for XPS – it is ready to put all these together into a .bit file, the summary of the hardware on our board. To generate this file, choose Hardware -> Generate Bitstream (this takes 10 min - but you can do software work in parallel with it).

APPENDIX B – XILINX SYSTEM BLOCK DIAGRAM



APPENDIX C – EXAMPLE SET OF SYSTEM SLAVES

APPENDIX D – EXAMPLE .DTS FILE

```
/*
 * Device Tree Generator version: 1.1
 *
 * (C) Copyright 2007-2008 Xilinx, Inc.
 * (C) Copyright 2007-2008 Michal Simek
 *
 * Michal SIMEK <monstr@monstr.eu>
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License as
 * published by the Free Software Foundation; either version 2 of
 * the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston,
 * MA 02111-1307 USA
 *
 * CAUTION: This file is automatically generated by libgen.
 * Version: Xilinx EDK 10.1.03 EDK_K_SP3.6
 *
 * XPS project directory: STORBOT_Final_0
 */

/dts-v1/;
/ {
       #address-cells = <1>;
       #size-cells = <1>;
       compatible = "xlnx,virtex405", "xlnx,virtex";
       model = "testing";
       DDR_SDRAM: memory@0 {
              device_type = "memory";
              reg = < 0x0 0x10000000 >;
       } ;
       chosen {
              bootargs = "console=ttyS0 root=/dev/xsa2 rw";
              linux,stdout-path = "/plb@0/serial@83e00000";
       } ;
       cpus {
              #address-cells = <1>;
              #cpus = <0x1>;
              #size-cells = <0>;
              ppc405_0: cpu@0 {
                     clock-frequency = <100000000>;
                     compatible = "PowerPC,405", "ibm,ppc405";
                     d-cache-line-size = <0x20>;
                     d-cache-size = <0x4000>;
                     dcr-access-method = "native";
```

```
                dcr-controller ;
                device_type = "cpu";
                i-cache-line-size = <0x20>;
                i-cache-size = <0x4000>;
                model = "PowerPC,405";
                reg = <0>;
                timebase-frequency = <100000000>;
                xlnx,dcr-resync = <0x0>;
                xlnx,deterministic-mult = <0x0>;
                xlnx,disable-operand-forwarding = <0x1>;
                xlnx,fastest-plb-clock = "DPLB0";
                xlnx,generate-plb-timespecs = <0x1>;
                xlnx,mmu-enable = <0x1>;
        } ;
} ;
plb0: plb@0 {
        #address-cells = <1>;
        #size-cells = <1>;
        compatible = "xlnx,plb-v46-1.03.a", "simple-bus";
        ranges ;
        DDR_SDRAM: mpmc@84800000 {
                #address-cells = <1>;
                #size-cells = <1>;
                compatible = "xlnx,mpmc-4.03.a";
                reg = < 0x84800000 0x10000 >;
        } ;
        DIPSWs_4Bit: gpio@81440000 {
                compatible = "xlnx,xps-gpio-1.00.a";
                interrupt-parent = <&xps_intc_0>;
                interrupts = < 1 2 >;
                reg = < 0x81440000 0x10000 >;
                xlnx,all-inputs = <0x1>;
                xlnx,all-inputs-2 = <0x0>;
                xlnx,dout-default = <0x0>;
                xlnx,dout-default-2 = <0x0>;
                xlnx,family = "virtex2p";
                xlnx,gpio-width = <0x4>;
                xlnx,interrupt-present = <0x1>;
                xlnx,is-bidir = <0x1>;
                xlnx,is-bidir-2 = <0x1>;
                xlnx,is-dual = <0x0>;
                xlnx,tri-default = <0xffffffff>;
                xlnx,tri-default-2 = <0xffffffff>;
        } ;
        LEDs_4Bit: gpio@81420000 {
                compatible = "xlnx,xps-gpio-1.00.a";
                interrupt-parent = <&xps_intc_0>;
                interrupts = < 2 2 >;
                reg = < 0x81420000 0x10000 >;
                xlnx,all-inputs = <0x0>;
                xlnx,all-inputs-2 = <0x0>;
                xlnx,dout-default = <0x0>;
                xlnx,dout-default-2 = <0x0>;
                xlnx,family = "virtex2p";
                xlnx,gpio-width = <0x4>;
```

```
        xlnx,interrupt-present = <0x1>;
        xlnx,is-bidir = <0x0>;
        xlnx,is-bidir-2 = <0x1>;
        xlnx,is-dual = <0x0>;
        xlnx,tri-default = <0xffffffff>;
        xlnx,tri-default-2 = <0xffffffff>;
    } ;
    PushButtons_5Bit: gpio@81400000 {
        compatible = "xlnx,xps-gpio-1.00.a";
        interrupt-parent = <&xps_intc_0>;
        interrupts = < 0 2 >;
        reg = < 0x81400000 0x10000 >;
        xlnx,all-inputs = <0x1>;
        xlnx,all-inputs-2 = <0x0>;
        xlnx,dout-default = <0x0>;
        xlnx,dout-default-2 = <0x0>;
        xlnx,family = "virtex2p";
        xlnx,gpio-width = <0x5>;
        xlnx,interrupt-present = <0x1>;
        xlnx,is-bidir = <0x1>;
        xlnx,is-bidir-2 = <0x1>;
        xlnx,is-dual = <0x0>;
        xlnx,tri-default = <0xffffffff>;
        xlnx,tri-default-2 = <0xffffffff>;
    } ;
    RS232_Uart_1: serial@83e00000 {
        clock-frequency = <100000000>;
        compatible = "xlnx,xps-uart16550-2.00.b", "ns16550";
        current-speed = <9600>;
        device_type = "serial";
        interrupt-parent = <&xps_intc_0>;
        interrupts = < 4 2 >;
        reg = < 0x83e00000 0x10000 >;
        reg-offset = <0x1003>;
        reg-shift = <2>;
        xlnx,family = "virtex2p";
        xlnx,has-external-rclk = <0x0>;
        xlnx,has-external-xin = <0x0>;
        xlnx,is-a-16550 = <0x1>;
    } ;
    SysACE_CompactFlash: sysace@83600000 {
        compatible = "xlnx,xps-sysace-1.00.a";
        interrupt-parent = <&xps_intc_0>;
        interrupts = < 3 2 >;
        reg = < 0x83600000 0x10000 >;
        xlnx,family = "virtex2p";
        xlnx,mem-width = <0x10>;
    } ;
    adc_conv_0: adc-conv@ce200000 {
        compatible = "xlnx,adc-conv-1.00.a";
        reg = < 0xce200000 0x10000 >;
        xlnx,family = "virtex2p";
        xlnx,include-dphase-timer = <0x0>;
    } ;
    adc_conv_1: adc-conv@ce220000 {
```

```
        compatible = "xlnx,adc-conv-1.00.a";
        reg = < 0xce220000 0x10000 >;
        xlnx,family = "virtex2p";
        xlnx,include-dphase-timer = <0x0>;
} ;
adc_conv_2: adc-conv@ce240000 {
        compatible = "xlnx,adc-conv-1.00.a";
        reg = < 0xce240000 0x10000 >;
        xlnx,family = "virtex2p";
        xlnx,include-dphase-timer = <0x0>;
} ;
dc_motor_0: dc-motor@cac00000 {
        compatible = "xlnx,dc-motor-1.00.a";
        reg = < 0xcac00000 0x10000 >;
        xlnx,family = "virtex2p";
        xlnx,include-dphase-timer = <0x0>;
} ;
dc_motor_1: dc-motor@cac40000 {
        compatible = "xlnx,dc-motor-1.00.a";
        reg = < 0xcac40000 0x10000 >;
        xlnx,family = "virtex2p";
        xlnx,include-dphase-timer = <0x0>;
} ;
dc_motor_2: dc-motor@cac60000 {
        compatible = "xlnx,dc-motor-1.00.a";
        reg = < 0xcac60000 0x10000 >;
        xlnx,family = "virtex2p";
        xlnx,include-dphase-timer = <0x0>;
} ;
dc_motor_3: dc-motor@cac20000 {
        compatible = "xlnx,dc-motor-1.00.a";
        reg = < 0xcac20000 0x10000 >;
        xlnx,family = "virtex2p";
        xlnx,include-dphase-timer = <0x0>;
} ;
dc_motor_encoder_0: dc-motor-encoder@c1a40000 {
        compatible = "xlnx,dc-motor-encoder-1.00.a";
        reg = < 0xc1a40000 0x10000 >;
        xlnx,family = "virtex2p";
        xlnx,include-dphase-timer = <0x0>;
} ;
dist_sens_0: dist-sens@c7a00000 {
        compatible = "xlnx,dist-sens-1.00.a";
        reg = < 0xc7a00000 0x10000 >;
        xlnx,family = "virtex2p";
        xlnx,include-dphase-timer = <0x0>;
} ;
refl_sens_0: refl-sens@c7c40000 {
        compatible = "xlnx,refl-sens-1.00.a";
        reg = < 0xc7c40000 0x10000 >;
        xlnx,family = "virtex2p";
        xlnx,include-dphase-timer = <0x0>;
} ;
refl_sens_1: refl-sens@c7c60000 {
        compatible = "xlnx,refl-sens-1.00.a";
```

```
                reg = < 0xc7c60000 0x10000 >;
                xlnx,family = "virtex2p";
                xlnx,include-dphase-timer = <0x0>;
        } ;
        refl_sens_2: refl-sens@c7c00000 {
                compatible = "xlnx,refl-sens-1.00.a";
                reg = < 0xc7c00000 0x10000 >;
                xlnx,family = "virtex2p";
                xlnx,include-dphase-timer = <0x0>;
        } ;
        refl_sens_3: refl-sens@c7c20000 {
                compatible = "xlnx,refl-sens-1.00.a";
                reg = < 0xc7c20000 0x10000 >;
                xlnx,family = "virtex2p";
                xlnx,include-dphase-timer = <0x0>;
        } ;
        wheel_step_0: wheel-step@c1a20000 {
                compatible = "xlnx,wheel-step-1.00.a";
                reg = < 0xc1a20000 0x10000 >;
                xlnx,family = "virtex2p";
                xlnx,include-dphase-timer = <0x0>;
        } ;
        wheel_step_1: wheel-step@c1a00000 {
                compatible = "xlnx,wheel-step-1.00.a";
                reg = < 0xc1a00000 0x10000 >;
                xlnx,family = "virtex2p";
                xlnx,include-dphase-timer = <0x0>;
        } ;
        xps_bram_if_cntlr_1: xps-bram-if-cntlr@ffff0000 {
                compatible = "xlnx,xps-bram-if-cntlr-1.00.a";
                reg = < 0xffff0000 0x10000 >;
                xlnx,family = "virtex2p";
        } ;
        xps_intc_0: interrupt-controller@81800000 {
                #interrupt-cells = <0x2>;
                compatible = "xlnx,xps-intc-1.00.a";
                interrupt-controller ;
                reg = < 0x81800000 0x10000 >;
                xlnx,num-intr-inputs = <0x5>;
        } ;
    } ;
    ppc405_0_dplb1: plb@1 {
        #address-cells = <1>;
        #size-cells = <1>;
        compatible = "xlnx,plb-v46-1.03.a", "simple-bus";
        ranges ;
    } ;
} ;
```

APPENDIX E – SCRIPT TO CREATE ROOT FILE SYSTEM (MKROOTFS)

```
#!/bin/sh
#
# mkrootfs creates a root filesystem for the
# Xilinx XUPV2P embedded development kit.
#
# Original Author
# (C) Wolfgang Klingauf 2003, 2004  [ wolfgang@klingauf.
#
# Re-written for the XUPV2P board for the linux-2.6 kernel
# (C) David VandeBunte 2009
# -- with some guidance from a similar script by Jonathan Donaldson --

# gcc & glibc versions chosen for PPC Linux
GCC_VER=gcc-4.2.0
GLIBC_VER=glibc-2.3.6

# Position to create embedded rootfs.  This will be sudo removed so watch
out.
RFS=/home/davidvandebunte/rootfs_dev/xupv2p_rfs

# Cross-compiler prefix
CC=powerpc-405-linux-gnu

# location of glibc tool-chain binaries which will be copied to the RFS.  All
the *.so
# files in the root file system come from these binaries.
TARGET_PREFIX=/opt/crosstool/${GCC_VER}-${GLIBC_VER}/${CC}/${CC}

export TARGET=powerpc-405-linux-gnu
export PREFIX=/opt/crosstool/gcc-4.2.0-glibc-2.3.6/${TARGET}

# update path to contain the actual location of the powerpc405 cross-compiler
gcc (and related)
export PATH=${PREFIX}/bin:${PATH}

# cross build tools directory
BUILD_TOOLS=/home/davidvandebunte/crosstool/crosstool-0.43/build/powerpc-405-
linux-gnu/gcc-4.0.2-glibc-2.3.6

# embedded linux kernel sources
PPC_KERNEL=/home/davidvandebunte/xilinx_kernel/linux-2.6-xlnx

# embedded linux kernel version
PPC_KERNEL_VERSION=2.6

# location of mounted cf card partitions
ACE_PART=/dev/sdb1
RFS_PART=/dev/sdb2

#
# No need to change anything below this line!
#
```

```
DEV_DIR=`pwd`

echo "mkrootfs v0.5 - (C) David VandeBunte 2009"
echo

if [ "$1" = "-h" ] || [ "$1" = "--help" ]; then
        echo "usage: $0 [options]"
        echo
        echo "options:"
        echo "  -h  | --help - show this help"
        echo "  -ct | - add (c)ompiler (t)ools to root file system"
        echo "  -nm | - (n)o file system (m)ount or copy to compact flash"
        exit
fi

COMPILER_TOOLS=0
NO_MOUNT=0

# DVB 04/08/09
# extract variables from command line.
for command in "$@"
do
        case ${command} in
        "-ct")
                COMPILER_TOOLS=1
                ;;
        "-nm")
                NO_MOUNT=1
                ;;
        *)
                echo "Command switch not recognized!  Exiting script..."
                exit
                ;;
        esac
done

if [ $NO_MOUNT -eq 0 ]; then
        echo " (1) Do you have the CF card attached to the computer?"
        echo " (2) ...and is the ACE partition in ${ACE_PART} and the RFS
partition in ${RFS_PART}?"
fi
echo "Press a key to proceed or CTRL-C to abort..."
read

cd ${DEV_DIR}

# remove old rootfs, if exist
if [ -a ${RFS} ]
        then echo "Removing old rootfs..."; sudo rm -rf ${RFS}
fi

# create directories
echo Creating directory structure...
mkdir -p ${RFS}/{bin,boot,dev/{pts,shm},etc/opt,home,lib,mnt,proc}
mkdir -p ${RFS}/{root,sbin,tmp,usr/local,var,opt}
for dirname in ${RFS}/usr ${RFS}/usr/local
        do
```

```
        mkdir $dirname/{bin,etc,include,lib,sbin,share,src}
        ln -s share/{man,doc,info} $dirname
        mkdir $dirname/share/{dict,doc,info,locale,man}
        mkdir $dirname/share/{nls,misc,terminfo,zoneinfo}
        mkdir $dirname/share/man/man{1,2,3,4,5,6,7,8}
done
mkdir ${RFS}/var/{lock,log,mail,run,spool}
mkdir -p ${RFS}/var/{tmp,opt,cache,lib/misc,local}
mkdir ${RFS}/opt/{bin,doc,include,info}
mkdir -p ${RFS}/opt/{lib,man/man{1,2,3,4,5,6,7,8}}
chmod 0750 ${RFS}/root
chmod 1777 ${RFS}/tmp ${RFS}/var/tmp
# for syslogd
ln -s /tmp/messages ${RFS}/var/log/messages

# copy glibc
echo Copying glibc...
cd ${TARGET_PREFIX}/lib
# cp *-*.so ${RFS}/lib
cp *.so ${RFS}/lib
cp *.so ${RFS}/usr/lib
cp *.so ${RFS}/usr/local/lib
cp *.so ${RFS}/var/lib
cp *.so.[*0-9] ${RFS}/lib
cp *.so.[*0-9] ${RFS}/usr/lib
cp *.so.[*0-9] ${RFS}/usr/local/lib
cp *.so.[*0-9] ${RFS}/var/lib
cp -d *.so.[*0-9] ${RFS}/lib
cp libSegFault.so libmemusage.so libpcprofile.so ${RFS}/lib
echo "glibc files installed. To reduce the size of installed libraries, use
strip /lib/*.so"

# DVB 02/17/09, 03/18/09, 03/20/09, 03/28/09
if [ $COMPILER_TOOLS -eq 1 ]; then
        echo "Installing compiler tools..."
        rm -r ${RFS}/usr/*
        sudo cp -r ${DEV_DIR}/usr/* ${RFS}/usr
        sudo chmod 777 ${RFS}/usr/bin
        sudo chmod 777 ${RFS}/usr/sbin
        echo "gcc and related tools installed."
else
        echo "Compiler tools not installed."
fi

# install nsswitch.conf from build-tools/glibc/nss/
echo Installing /etc/nsswitch.conf
cp ${BUILD_TOOLS}/glibc-2.3.6/nss/nsswitch.conf ${RFS}/etc

# install kernel modules
echo "Install kernel modules..."
cd ${PPC_KERNEL}
cp -a images/modules-${PPC_KERNEL_VERSION}/* ${RFS}

# install /etc
echo "Installing /etc..."
cd ${DEV_DIR}
cp -a etc ${RFS}
```

```
# DVB 02-14-09, 03-18-09
# Compiling all c programs in home for the powerpc-405.
cd ${DEV_DIR}/home
echo "Compiling 'Hello World!' and other .c files..."
for file in `ls | grep .c | sed 's/\(.*\)\..*/\1/'`
do
${CC}-gcc -o ${file} ${file}.c
done

# DVB 04/15/09
# Compile the main controller program and add to the CF card.
cd ${DEV_DIR}/home/controller
echo "Adding main controller code..."
make

# DVB 03-18-09
# install /home
echo "Installing /home..."
cp -r ${DEV_DIR}/home ${RFS}
cd ${RFS}/bin
ln -s /home/controller/goGetBox goGetBox

# clean up the /home directory in development directory for future use.
cd ${DEV_DIR}/home
for file in *
do
        if [ ! -d ${file} ]; then
                if [ -x ${file} ]; then
                        rm $file
                fi
                if [ ! -z "`echo "${file}" | grep "~$"`" ]; then
                        rm $file
                fi
        fi
done

# done
echo "Your rootfs has been created."
echo
echo "Now installing busybox to get a running system."
cd ${DEV_DIR}
cd busybox-1.2.0

# Adjust .config for installing the busybox targets properly.
mv .config .config.OLD
cat .config.OLD | sed /^PREFIX/d > .config.OLD2 && rm -f .config.OLD
echo "PREFIX=\"$RFS\"" > .config
cat .config.OLD2 >> .config && rm -f .config.OLD2

echo "Busybox output sent to busybox.txt"
make dep > busybox.txt
make >> busybox.txt
make install >> busybox.txt

# DVB 02-07-09
cd ${RFS}/dev
```

```
sudo mknod console c 5 1
sudo mknod full c 1 7
sudo mknod mem c 1 1
sudo mknod mtd c 90 0
sudo mknod mtdblock b 31 0
sudo mknod mtdr c 90 1
sudo mknod null c 1 3
sudo mknod ram b 1 0
sudo mknod rtc c 10 135
sudo mknod tty c 5 0
sudo mknod ttyS c 4 64
sudo mknod zero c 1 5
sudo mknod refl_sens_0 c 234 0
sudo mknod refl_sens_1 c 234 1
sudo mknod refl_sens_2 c 234 2
sudo mknod refl_sens_3 c 234 3
sudo mknod wheel_step_0 c 235 0
sudo mknod wheel_step_1 c 235 1
sudo mknod dist_sens_0 c 236 0
sudo mknod dist_sens_1 c 236 1
sudo mknod prox_sens_0 c 237 0
sudo mknod prox_sens_1 c 237 1
sudo mknod prox_sens_2 c 237 2
sudo mknod dc_motor_0 c 238 0
sudo mknod dc_motor_1 c 238 1
sudo mknod dc_motor_2 c 238 2
sudo mknod dc_motor_3 c 238 3

# DVB 02-14-09, 03-28-09
# Copy files over to CF card.
if [ $NO_MOUNT -eq 0 ]; then
        echo
        echo "Moving files to CF card..."
        echo "Mounting paritions to appropriate locations."
        if [ -s /media/disk ]; then
                sudo umount /media/disk
        fi
        if [ -s /media/disk-1 ]; then
                sudo umount /media/disk-1
        fi
        sudo mkdir /media/{rfs_part,ace_part}
        sudo mount ${RFS_PART} /media/rfs_part
        sudo mount ${ACE_PART} /media/ace_part
        CF_CARD_RFS=/media/rfs_part
        CF_CARD_ACE=/media/ace_part
        sudo rm -r ${CF_CARD_RFS}/*
        echo "Copying new files to CF card."
        sudo cp -r ${RFS}/* ${CF_CARD_RFS}
        echo "RFS transfer complete!  Unmounting volumes..."
        sudo umount /media/rfs_part
        sudo umount /media/ace_part
        sudo rmdir /media/{rfs_part,ace_part}
fi

# DVB 01-30-09
# [-R]ecursively change permissions in rootfs.  Make root the owner (whether
# here or on the board) and make it so that everyone can execute all files.
```

```
echo
echo "Changing permissions and ownerships"
sudo chown -R root ${RFS}
sudo chmod -R a+x ${RFS}
sudo chmod 755 ${RFS}/etc/init.d/rcS

echo
echo "Done."
```

APPENDIX F – RCS FILE (RUN AT KERNEL BOOT)

```
#!/bin/ash

echo
echo Welcome to XUPV2P powerpc linux 2.6.  \(etc/init.d/rcS\)
echo
echo Starting system...

# First mount /proc!
echo -n "Mounting /proc: "
mount -n /proc /proc -t proc
echo "done."

# Make sure / is rw
echo -n "Mounting '/' read-write: "
mount -n -o remount,rw /
echo "done."

# Setup the ram filesystem setup
echo -n "Mounting /tmp: "
mount -n /tmpfs /tmp -t tmpfs
echo "done."

# Mount /sys (device files, system files).
echo -n "Mounting /sys: "
mkdir /sys
mount sysfs /sys -t sysfs -n

touch /tmp/utmp
touch /tmp/wtmp
touch /tmp/lastlog
touch /tmp/messages
touch /tmp/thttpd.log

# Start syslogd/klogd
echo -n "Starting syslogd: "
syslogd
echo "done."
echo -n "Starting klogd: "
klogd
echo "done."

# fill up /dev using busybox utility mdev
mdev -s

# run shell
echo "logging in as root..."
exec /bin/ash

echo System started.
```

APPENDIX G – WHEEL STEP DEVICE DRIVER

```
/*
 *  wheel_step.c - driver for the wheel motor controllers connected to our board.
 */
#include <linux/module.h>  /* Needed by all linux kernel driver modules */
#include <linux/kernel.h>  /* Needed for KERN_INFO and other printk stuff */
#include <linux/cdev.h>    /* provides cdev struct, how the kernel represents char
   devices internally */
#include <linux/spinlock.h>/* provides the spinlock type, and functions. */
#include <asm/uaccess.h>   /* copy_to_user functions, other functions so we don't have
                                    to dereference user-space pointers */
#include <linux/errno.h>   /* provides all of the error numbers like EFAULT */
#include <linux/fs.h>      /* has the file data structure */
#include <linux/slab.h>    /* kmalloc and kfree */

// #ifdef CONFIG_OF
// For open firmware.
#include <linux/of_device.h>
#include <linux/of_platform.h>
// #endif

#define MAJOR_NUM 235
#define DEVICE_NAME "wheel-step"

/* Use '0xC6' as our magic number, based on look at ioctl-numbers.txt */
#define WHEEL_STEP_MAGIC 0xC6

// define all of our commands
#define WHEEL_STEP_IOCRESET         _IO(WHEEL_STEP_MAGIC, 0)
#define WHEEL_STEP_WRITE_PIN_1      _IOW(WHEEL_STEP_MAGIC, 1, int)
#define WHEEL_STEP_WRITE_PIN_2      _IOW(WHEEL_STEP_MAGIC, 2, int)
#define WHEEL_STEP_WRITE_PIN_3      _IOW(WHEEL_STEP_MAGIC, 3, int)
#define WHEEL_STEP_WRITE_PIN_4      _IOW(WHEEL_STEP_MAGIC, 4, int)
#define WHEEL_STEP_WRITE_PIN_5      _IOW(WHEEL_STEP_MAGIC, 5, int)
#define WHEEL_STEP_MAXNR   6


// wheel_step registers.  used for ioctl call.
typedef struct {
         unsigned int  control_reg; // each bit of the register controls a pin for the
                                             // motor controller
} wheel_step_hw_map;

static struct cdev wheel_step_cdev[4];
static wheel_step_hw_map *wheel_step_registers;
unsigned long* wheel_step_virt_addr[4];

/* FILE SYSTEM OPERATIONS */
static int wheel_step_open(struct inode *inode, struct file *filp) {
         unsigned int* minor_num = kmalloc(1 * sizeof(unsigned int), GFP_KERNEL);
         *(minor_num) = iminor(inode);
         filp->private_data = minor_num;
         return 0;
}

static int wheel_step_release(struct inode *inode, struct file *filp) {
         kfree(filp->private_data);
         return 0;
}

static int wheel_step_ioctl(struct inode *inode, struct file *filp, unsigned int cmd,
unsigned long arg) {
         unsigned int minor_num = *((unsigned long*) (filp->private_data));
         switch(cmd) {
                  case WHEEL_STEP_WRITE_PIN_1:
```

```
                        if (arg == 0) {
                                *(wheel_step_virt_addr[minor_num]) &= 0xFFFFFFFE;
                        } else if (arg == 1) {
                                *(wheel_step_virt_addr[minor_num]) |= 0x00000001;
                        } else {
                                printk(KERN_ERR "Must choose 0 or 1 for the pin!");
                                return -ENOTTY;
                        }
                        break;
                case WHEEL_STEP_WRITE_PIN_2:
                        if (arg == 0) {
                                *(wheel_step_virt_addr[minor_num]) &= 0xFFFFFFFD;
                        } else if (arg == 1) {
                                *(wheel_step_virt_addr[minor_num]) |= 0x00000002;
                        } else {
                                printk(KERN_ERR "Must choose 0 or 1 for the pin!");
                                return -ENOTTY;
                        }
                        break;
                case WHEEL_STEP_WRITE_PIN_3:
                        if (arg == 0) {
                                *(wheel_step_virt_addr[minor_num]) &= 0xFFFFFFFB;
                        } else if (arg == 1) {
                                *(wheel_step_virt_addr[minor_num]) |= 0x00000004;
                        } else {
                                printk(KERN_ERR "Must choose 0 or 1 for the pin!");
                                return -ENOTTY;
                        }
                        break;
                case WHEEL_STEP_WRITE_PIN_4:
                        if (arg == 0) {
                                *(wheel_step_virt_addr[minor_num]) &= 0xFFFFFFF7;
                        } else if (arg == 1) {
                                *(wheel_step_virt_addr[minor_num]) |= 0x00000008;
                        } else {
                                printk(KERN_ERR "Must choose 0 or 1 for the pin!");
                                return -ENOTTY;
                        }
                        break;
                case WHEEL_STEP_WRITE_PIN_5:
                        if (arg == 0) {
                                *(wheel_step_virt_addr[minor_num]) &= 0xFFFFFFEF;
                        } else if (arg == 1) {
                                *(wheel_step_virt_addr[minor_num]) |= 0x00000010;
                        } else {
                                printk(KERN_ERR "Must choose 0 or 1 for the pin!");
                                return -ENOTTY;
                        }
                        break;
                default:
                        printk(KERN_ERR "you can't choose this option");
                        return -ENOTTY;
        }
        return 0;
}

static ssize_t wheel_step_read(struct file *filp, char __user *buf, size_t count, loff_t
*f_pos) {
        ssize_t retval = 0;
        unsigned long mem_buf;
        unsigned int minor_num = *((unsigned long*) (filp->private_data));
        // note that this switch statement exists only so this driver can handle the
// default case.
        switch(minor_num) {
                case 0:
                        mem_buf = *(wheel_step_virt_addr[0]);
                        break;
```

```
                case 1:
                        mem_buf = *(wheel_step_virt_addr[1]);
                        break;
                default:
                        printk(KERN_ERR "unrecognized minor number, wheel_step read\n");
                        mem_buf = 0;
                        break;
        }
        if (copy_to_user(buf, &mem_buf, count)) {
                retval = -EFAULT;
                return retval;
        }
        retval = count;
        *f_pos += count;
        return retval;
}

struct file_operations wheel_step_fops = {
        .owner = THIS_MODULE,
        .open  = wheel_step_open,
        .release  =       wheel_step_release,
        .read  = wheel_step_read,
        .ioctl = wheel_step_ioctl,
};

/* LOADING/UNLOADING DRIVER FUNCTIONS */
static int __init wheel_step_init(void) {
        struct device_node *np = NULL;
        int result;
        struct resource resource;
        unsigned int startAddr;
        int i;

        for(i = 0; i < 2; i++) {
                np = of_find_node_by_name(np, "wheel-step");
                if (!np) {
                        if (i == 0) {
                                printk(KERN_ERR "wheel-step: can't find a compatible
node in"
                                  + " this kernel build\n");
                                return -ENODEV;
                        } else {
                                return 0;
                        }
                } else {
                        result = of_address_to_resource(np, 0, &resource);
                        if (result < 0) {
                                return result;
                        }
                        startAddr = (unsigned int)resource.start;

                        cdev_init(&wheel_step_cdev[i], &wheel_step_fops);
                        if (cdev_add(&wheel_step_cdev[i], MKDEV(MAJOR_NUM, i), 1) ||
                                (register_chrdev_region(MKDEV(MAJOR_NUM, i), 1, "wheel-
step")) < 0) {
                                printk(KERN_ERR "wheel-step-%i: cannot register
/dev/wheel-step\n", i);
                                return -ENODEV;
                        }
                        // map the physical address of the timer into the virtual
address
                    // space
                        wheel_step_virt_addr[i] = of_iomap(np, 0);
                        printk(KERN_INFO "wheel_step_%i at 0x%08X: registered\n", i,
startAddr);
                }
        }
```

```
        return 0;
}

static void __exit wheel_step_exit(void)
{
        printk(KERN_INFO "wheel_step_exit!  Never gotten to this code...\n");
}

module_init(wheel_step_init);
module_exit(wheel_step_exit);

MODULE_AUTHOR("Senior Design Team 5");
MODULE_LICENSE("GPL");
```

APPENDIX H – GRAPHICAL USER INTERFACE SOURCE CODE

INVENTORYFORM.JAVA

```
//*************************************************************
// Code: Inventory_form.java
//
// Author: Joshua Vroom
//
//
// Modification History
//
// Author: Josh Vroom
// Date: March 15, 2009
// Version: 1.0
// Modifications: Initial Version
//
//
// Summary: The inventory form is a user notification box that pops up every
// time a product is selected.  The inventory form informs the user of the
// product they selected, as well as the remaining quantity of that product.
//
//
//
//*************************************************************

package gui_code;



import java.awt.BorderLayout;
import java.io.*;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.RandomAccessFile;

import app.Com;
import app.Parameters;
import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JProgressBar;
import javax.swing.SwingConstants;


public class Inventory_form extends JFrame {
        static Integer KEYSTART = 0;
        static Integer KEYSTOP = 6;
        static Integer QTYSTART = 7;
        static Integer QTYSTOP = 9;
        static Integer QTYTOTSTART = 10;
        static Integer QTYTOTSTOP = 12;
        static Integer SHELFSTART = 13;
        static Integer SHELFSTOP = 14;
```

```
	static Integer RACKSTART = 15;
	static Integer RACKSTOP = 16;
	private static final long serialVersionUID = 1L;
	public String CONFIGHEADER;
	 public String CONFIGCOMPORT;
	 public String CONFIGPLPATH;

	public  Inventory_form(String selectedItemString, RandomAccessFile raf) {
			final String selectedItemStringFinal = selectedItemString;
			final RandomAccessFile rafFinal = raf;

			getConfigData();

			JButton exit = new JButton("exit");
	//create the "STOP MOVEMENT" button and add it's action listener
			final JButton stopButton = new JButton("Stop");
			stopButton.setEnabled(false);
	//create the "STOP MOVEMENT" button and add it's action listener
			final JButton goButton = new JButton("Go");
	//create the "STOP MOVEMENT" button and add it's action listener
			final JButton exitButton = new JButton("Exit");

			JPanel exitPane = new JPanel();
			exitPane.setLayout(new BoxLayout(exitPane, BoxLayout.LINE_AXIS));
			exitPane.setBorder(BorderFactory.createEmptyBorder(0, 10, 10, 10));
			exitPane.add(Box.createHorizontalGlue());
			exitPane.add(exit);


	// Lay out the buttons from left to right.
			JPanel buttonPane = new JPanel();
			buttonPane.setLayout(new BoxLayout(buttonPane, BoxLayout.LINE_AXIS)); // changes axis of buton
alignment from x to y (page vs line)
			buttonPane.setBorder(BorderFactory.createEmptyBorder(0, 10, 10, 10));
			buttonPane.add(Box.createHorizontalGlue()); // likely causes right alignment
			buttonPane.add(Box.createRigidArea(new Dimension(10, 0)));
			buttonPane.add(goButton);
			buttonPane.add(stopButton);
			buttonPane.add(exitButton);

	// Put everything together, using the content pane's BorderLayout.

			pack();


			String quantity = getCurrentQuantityData(selectedItemStringFinal, rafFinal);

			JLabel partLabel = new JLabel(selectedItemString.substring(0,4));
			partLabel.setHorizontalAlignment(SwingConstants.CENTER);
			if (quantity == " 0" ) goButton.setEnabled(false); // Will not allow you to get a box that isnt there
			String invQtyString = "Current Inventory: " + quantity ;
			final JLabel inventoryLabel = new JLabel(invQtyString);
			inventoryLabel.setHorizontalAlignment(SwingConstants.CENTER);

			final JProgressBar ProgBar = new JProgressBar();
			JPanel progressPane = new JPanel();
			progressPane.add(ProgBar, BorderLayout.SOUTH);
			progressPane.setBorder(BorderFactory.createEmptyBorder(0, 10, 10, 10));
			String totalQty = getTotalPossibleQuantity(selectedItemString, raf);
			ProgBar.setMaximum(Integer.decode(totalQty));
			ProgBar.setValue(Integer.decode(getCurrentQuantityData(selectedItemStringFinal, rafFinal)));
			ProgBar.setName("Current Inventory");

			JPanel inventoryPane = new JPanel();
			inventoryPane.add(partLabel, BorderLayout.NORTH);
			inventoryPane.add(inventoryLabel, BorderLayout.SOUTH);
```

```
                inventoryPane.add(ProgBar);
                inventoryPane.setSize(250,50);
                buttonPane.setSize(250,150);
                Container contentPane = getContentPane();
                contentPane.add(inventoryPane,-1);
                contentPane.add(buttonPane, BorderLayout.PAGE_END);
                setPreferredSize(new Dimension(300, 120));
                pack();
                setLocationRelativeTo(exit);
                setVisible(true);



        // Adding the stop action listener to send out a CTRL-Z on the serial port
                // since the robot is running a Linux program for each box retrieval process
                // a ctrl+z is the kill signal, so it ends the program by killing it, unix-style
                stopButton.addActionListener(new ActionListener(){
                        public void actionPerformed(ActionEvent e) {
                                //byte[] send = "crtl z".getBytes(); //TODO find ascii for ctrl-z (might be 0x1A)
                                //writeToSerialPort(send);
                                System.out.println("Stopping robot motion");
                                goButton.setEnabled(true);
                                stopButton.setEnabled(false);
                                String quantityNew = getNewIncrementedQuantityData(selectedItemStringFinal,
rafFinal);
                                inventoryLabel.setText("Current Inventory: " + quantityNew);
                                ProgBar.setValue(Integer.decode(getCurrentQuantityData(selectedItemStringFinal,
rafFinal)));
                                //inventoryLabel.setText("Current Inventory: " + quantity);

                        }

                });

                goButton.addActionListener(new ActionListener(){
                        public void actionPerformed(ActionEvent e) {
                                String robotCommand = "goGetBox -s " +  getShelfData(selectedItemStringFinal,
rafFinal) + " -r " + getRackData(selectedItemStringFinal, rafFinal)+ "\n\r"; // linux formatted command for box retrieval
                                try {
                                writeStringToSerialPort(robotCommand);
                                } catch (Exception d) {
                                        d.printStackTrace();
                                }
                                goButton.setEnabled(false);
                                stopButton.setEnabled(true);
                                String quantityNew = getNewDecrementedQuantityData(selectedItemStringFinal,
rafFinal);
                                inventoryLabel.setText("Current Inventory: " + quantityNew);
                                ProgBar.setValue(Integer.decode(getCurrentQuantityData(selectedItemStringFinal,
rafFinal)));
                                                        }

                });

                exitButton.addActionListener(new ActionListener(){
                        public void actionPerformed(ActionEvent e) {

                                setVisible(false);
                        }

                });
        }

        public String getCurrentQuantityData (String selectedItem, RandomAccessFile raf){
                String qtyString = null ;
                //Long previousFilePointer = null;
```

```
                try{
                        String record = null;
                        String recordKey = null;
                        raf.seek(0);
                        while ( (record=raf.readLine()) != null ) {
                                recordKey = record.substring(KEYSTART,KEYSTOP);                 // key in 1-4
                                if (selectedItem.equals(recordKey)) {
                                        qtyString = record.substring(QTYSTART,QTYSTOP);

                                                break;
                                                }
                        }
                }
                catch (IOException e) {
                        e.printStackTrace();
                }
                if (qtyString.charAt(0) == ' '){
                        qtyString = qtyString.substring(1);
                }
                return qtyString;

        }

        public  String  getNewDecrementedQuantityData(String selectedItem, RandomAccessFile raf){
                String quantityString = null ;
                Long previousFilePointer = null;
                try{
                        String record = null;
                        String recordKey = null;
                        raf.seek(0);
                        previousFilePointer = raf.getFilePointer() ;
                   while ( (record=raf.readLine()) != null ) {
                                recordKey = record.substring(KEYSTART,KEYSTOP);                 // key in 1-4
                                if (selectedItem.equals(recordKey)) {
                                        quantityString = record.substring(QTYSTART,QTYSTOP);
                                        // update the quantity in this record, due to matching key:
                                                quantityString = decrementQuantity(raf,
previousFilePointer,Integer.decode(quantityString.trim()));
                                                break;
                                                }
                                previousFilePointer = raf.getFilePointer() ;   // remember where 'next' record starts, in
case we need to update it
                                }
                }
                catch (IOException e) {
                        e.printStackTrace();
                }
                return quantityString;
        }

        public String decrementQuantity(RandomAccessFile raf, Long recordPointer, Integer oldQuantity) {

                Long quantityFilePointer = recordPointer + QTYSTART;   // known offset of the quantity in every record
                if (oldQuantity == 0) {
                        return " 0"; // very rudimentary way to keep us from never getting a part when we dont have it
                }
           Integer newQuantity  = oldQuantity - 1 ;              // decrement quantity
                String newQuantityString = newQuantity.toString();      // convert incremented integer back to string
                // be sure this String is " n" for values < 10:
                if (newQuantityString.length() == 1) newQuantityString = " " + newQuantityString; // convert "n" to " n"

                try {
                        // FILE UPDATE:
                        //    File pointer should be referring to the "quantity" field in the desired record (by .seeK())
                        //    we expect to write exactly TWO bytes from the 2-digit quantity string:
                        raf.seek(quantityFilePointer);                  // position to (2 byte) quantity field in desired record
```

```
                        raf.write(newQuantityString.getBytes());        // write the TWO bytes of the quantity string


                }catch (IOException e) {
                        e.printStackTrace();
                }
                return newQuantityString;                       // in case caller cares what the new quantity is
                }

        public  String  getNewIncrementedQuantityData(String selectedItem, RandomAccessFile raf){
                //String invData = "00";                // always 3 items: rack, shelf, quantity
                String quantityString = null ;
                Long previousFilePointer = null;
                try{
                        String record = null;
                        String recordKey = null;
                        raf.seek(0);
                        //System.out.println("did .seek");

                   previousFilePointer = raf.getFilePointer() ;
                   while ( (record=raf.readLine()) != null ) {
                                recordKey = record.substring(KEYSTART,KEYSTOP);                 // key in 1-4
                                if (selectedItem.equals(recordKey)) {
                                        quantityString = record.substring(QTYSTART,QTYSTOP);
                                        // update the quantity in this record, due to matching key:
                                                quantityString = incrementQuantity(raf,
previousFilePointer,Integer.decode(quantityString.trim()));
                                                break;
                                                }
                                previousFilePointer = raf.getFilePointer() ;   // remember where 'next' record starts, in
case we need to update it
                                }
                }
                catch (IOException e) {
                        e.printStackTrace();
                }
                return quantityString;
        }

        public String incrementQuantity(RandomAccessFile raf, Long recordPointer, Integer oldQuantity) {

                Long quantityFilePointer = recordPointer + QTYSTART;   // known offset of the quantity in every record

           Integer newQuantity  = oldQuantity +1 ;              // decrement quantity
                String newQuantityString = newQuantity.toString();      // convert incremented integer back to string
                // be sure this String is " n" for values < 10:
                if (newQuantityString.length() == 1) newQuantityString = " " + newQuantityString; // convert "n" to " n"

                try {
                        // FILE UPDATE:
                        //    File pointer should be referring to the "quantity" field in the desired record (by .seeK())
                        //    we expect to write exactly TWO bytes from the 2-digit quantity string:
                        raf.seek(quantityFilePointer);                  // position to (2 byte) quantity field in desired record
                        raf.write(newQuantityString.getBytes());        // write the TWO bytes of the quantity string


                }catch (IOException e) {
                        e.printStackTrace();
                }
                return newQuantityString;                       // in case caller cares what the new quantity is
                }
        public String getShelfData (String selectedItem, RandomAccessFile raf){
                String shelfString = null ;
                try{
                        String record = null;
                        String recordKey = null;
```

```
                raf.seek(0);
                while ( (record=raf.readLine()) != null ) {
                        recordKey = record.substring(KEYSTART,KEYSTOP);          // key in 1-4
                        if (selectedItem.equals(recordKey)) {
                                shelfString = record.substring(SHELFSTART,SHELFSTOP);

                                        break;
                                        }
                        }
                }
                catch (IOException e) {
                        e.printStackTrace();
                }
                return shelfString;
        }


        public String getRackData (String selectedItem, RandomAccessFile raf){
                String rackString = null ;
                try{
                        String record = null;
                        String recordKey = null;
                        raf.seek(0);
                   while ( (record=raf.readLine()) != null ) {
                        recordKey = record.substring(KEYSTART,KEYSTOP);          // key in 1-4
                        if (selectedItem.equals(recordKey)) {
                                rackString = record.substring(RACKSTART,RACKSTOP);

                                        break;
                                        }
                        }
                }
                catch (IOException e) {
                        e.printStackTrace();
                }
                return rackString;
        }

        public String getTotalPossibleQuantity(String selectedItem, RandomAccessFile raf){
                String totalQtyString = null ;
                try{
                        String record = null;
                        String recordKey = null;
                        raf.seek(0);
                        while ( (record=raf.readLine()) != null ) {
                        recordKey = record.substring(KEYSTART,KEYSTOP);          // key in 1-4
                        if (selectedItem.equals(recordKey)) {
                                totalQtyString = record.substring(QTYTOTSTART,QTYTOTSTOP);

                                        break;
                                        }
                        }
                }
                catch (IOException e) {
                        e.printStackTrace();
                }
                return totalQtyString;
        }


public void writeStringToSerialPort( String command) throws Exception{
        /*Runtime rt = Runtime.getRuntime();
        Process p = null;
        String portname = "com3:";
         String command1[] = {
```

```
                                    "c:\\windows\\system32\\cmd.exe", "/c",
                                    "start", "/min",
                                    "c:\\windows\\system32\\mode.com", portname,
                                    "baud=9600", "parity=n", "data=8",
                                    "stop=1",
                                    };
                try {
                                     p = rt.exec( command1 );
                                            if( p.waitFor() != 0 ) {
                                               System.out.print("Error executing command: " + command1 );
                                               System.exit( -1 );
                                            }
                                    } catch (InterruptedException e) {
                                            // TODO Auto-generated catch block
                                            e.printStackTrace();
                                    } catch (IOException e){
                                            e.printStackTrace();
                                    }*/
                 //*** Open port COM1
                Parameters param = new Parameters();
        param.setPort("com"+CONFIGCOMPORT);
        System.out.println("com"+CONFIGCOMPORT);
        param.setBaudRate("9600");
        Com myCom = new Com(param);
                System.out.println(command);
                //byte commandBytes[] = command.getBytes();
                char commandXmit = 0;
                //String commandXmit2 = null;
                for (int i = 0; i < command.length(); i++){
                        commandXmit = command.charAt(i);
                        //commandXmit2 = Character.toString(commandXmit);
                        //commandByte = commandXmit2.getBytes();
                        myCom.sendSingleData(commandXmit);
                }
                //*** Close COM1
        myCom.close();
}

public void writeToSerialPort(byte[] part) {
                try {
                        String portname = "com3:";
                        FileOutputStream fos = new FileOutputStream( portname );
                        fos.write(part, 0, part.length );
                        System.out.print(part );
                        fos.close();
                } catch (FileNotFoundException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
}


public void getConfigData(){

                try{
                        RandomAccessFile raf = new RandomAccessFile("team5.cfg", "r");
                        String record = null;
                        while ( (record=raf.readLine()) != null ) {
                        if (record.substring(0,7).equals("header="))
                                {CONFIGHEADER = record.substring(7);}
                        if (record.substring(0,8).equals("comport="))
                                {CONFIGCOMPORT = record.substring(8);}
                        if (record.substring(0,7).equals("plpath="))
```

```
                        {CONFIGPLPATH = record.substring(7);}
        }

        } catch (IOException e){
                e.printStackTrace();
        }

}
}
```

USERFORM.JAVA

```
//*********************************************************
// Code: User_form.java
//
// Author: Joshua Vroom
//
//
// Modification History
//
// Author: Josh Vroom
// Date: March 15, 2009
// Version: 1.0
// Modifications: Initial Version
//
// Author: Dave Van Kampen
// Version: 1.1
// Date: April 8, 2009
// Modifications: Added code for setting serial port baud rate, closing serial port
//
//
// Summary: The User Form is the main window people running the robot will see
// they will see a drop down menu, an exit button, and a select button.
// an option we may want to add is a set of radio buttons
// they can choose from for which com port they want to write to,
// assuming the computer has more than 1.
// this will allow control of multiple units.
//
// Data:
//
//
//*********************************************************

package gui_code;
import javax.swing.*;

import java.awt.*;
import java.awt.event.*;
import java.io.*;

public class Userform extends JFrame {

       private static final long serialVersionUID = 1L;
       static JFrame frame;
       static Integer XRACK= 0;
       static Integer XSHELF= 1;
       static Integer XQTY= 2;
  static Integer KEYSTART = 0;
  static Integer KEYSTOP = 6;
  static Integer QTYSTART = 7;
  static Integer QTYSTOP = 9;
 static String PICKAPART="Pick a part...";
 public String CONFIGHEADER;
 public String CONFIGCOMPORT;
 public String CONFIGPLPATH;
       // the explicit constructor
       public Userform() {
```

```
            super("Team 5 - STORBOT: Parts Retrieval Interface");        // build title
for form



             final JButton button = new JButton("Pick a new part...");   //TODO explain
what this does
             RandomAccessFile raf = null;
                   try{
                          getConfigData();
                           raf = new RandomAccessFile(CONFIGPLPATH, "rw");
                          } catch (IOException e){
                           e.printStackTrace();
                   }
                         // TODO is this needed, or is there better way to do this
                   final RandomAccessFile rafFinal = raf ;

                   final JComboBox partsComboBox = getPartsComboBox(rafFinal);

             // appears to require Stream as 'final' to use in other functions
                           // TODO is this needed, or is there better way to do this
    partsComboBox.setSize(100,100);
            // create the "Get Part" button and add it's action listener
            JButton getPartButton = new JButton("Get Part");

            // create the "Exit" button and add it's action listener
            JButton exitButton = new JButton("Exit");

            // Create a container so that we can add a title around
            // the scroll pane. Can't add a title directly to the
            // scroll pane because its background would be white.
            // Lay out the label and scroll pane from top to bottom.
            JPanel listPane = new JPanel();
            listPane.setLayout(new BoxLayout(listPane, BoxLayout.PAGE_AXIS));
            JLabel label = new JLabel("Parts in inventory:");
            listPane.add(label);
            label.setAlignmentX(LEFT_ALIGNMENT);                // move a little left of
center, but not to left of form (?)
            listPane.add(Box.createHorizontalGlue());

            listPane.add(partsComboBox);
            listPane.add(Box.createRigidArea(new Dimension(0, 5)));
            listPane.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

            // Lay out the buttons from left to right.
            JPanel buttonPane = new JPanel();
            buttonPane.setLayout(new BoxLayout(buttonPane, BoxLayout.LINE_AXIS));
            buttonPane.setBorder(BorderFactory.createEmptyBorder(0, 10, 10, 10));
            buttonPane.add(Box.createHorizontalGlue());
            buttonPane.add(getPartButton);
            buttonPane.add(Box.createRigidArea(new Dimension(10, 0)));
            buttonPane.add(exitButton);


            // Put everything together, using the content pane's BorderLayout.
            Container contentPane = getContentPane();
            setPreferredSize(new Dimension(500, 150));
            contentPane.add(listPane, BorderLayout.NORTH);
            contentPane.add(buttonPane, BorderLayout.PAGE_END);


            pack();
            setLocationRelativeTo(button);   //TODO   explain why this is needed?
positions form on screen?

            getPartButton.addActionListener(new ActionListener() {
```

```
                        //final DataInputStream disLocal = dis;
                    public void actionPerformed(ActionEvent e) {
                    //JComboBox partsComboBox = new JComboBox();
                       Object selectedItem = partsComboBox.getSelectedItem();
                       String selectedItemString = selectedItem.toString() ;
                       //System.out.println("selected: " + selectedItemString);
                       if (selectedItemString.equals(PICKAPART)) {
                              //TODO action for picking the first ComboBox string - not
allowed
                       }
                                                  //Inventory_form(String
selectedItemString,RandomAccessFile raf)
                Inventory_form IF = new Inventory_form(selectedItemString, rafFinal);
                    }

                });

                exitButton.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                       System.out.print("exit the UF");
                       System.exit(0);
                    }
                });


        }

        public static void main(String[] args) {

                Userform UF = new Userform();
                UF.setVisible(true);

        }

        public void getConfigData(){

                 try{
                        RandomAccessFile raf = new RandomAccessFile("team5.cfg", "r");
                       String record = null;
                       while ( (record=raf.readLine()) != null ) {
                       if (record.substring(0,7).equals("header="))
                              {CONFIGHEADER = record.substring(7);}
                       if (record.substring(0,8).equals("comport="))
                              {CONFIGCOMPORT = record.substring(8);}
                       if (record.substring(0,7).equals("plpath="))
                              {CONFIGPLPATH = record.substring(7);}
                }

                 } catch (IOException e){
                        e.printStackTrace();
                 }

        }



        public JComboBox getPartsComboBox(RandomAccessFile raf){
                String[] pl= {PICKAPART};
                JComboBox comboBox = new JComboBox(pl);

        try{
                String record = null;
                while ( (record=raf.readLine()) != null ) {
                              comboBox.addItem(makeObj(record.substring(KEYSTART,KEYSTOP)));
// key is 1-4
                       }
```

```
        }
        catch (IOException e) {
                e.printStackTrace();
        }
  return comboBox;

        }

        private Object makeObj(final String item)  {
    return new Object() { public String toString() { return item; } };
  }

}
```

CONFIGURATION FILE (TEAM5.CFG):

```
header=Team 5 Project
comport=3
plpath=C:/PartsList.txt
```

PARTS LIST FILE:

```
ABCDEF 10 11 3 1
123456 27 45 3 2
MASKS  12 45 2 1
BOLTS  44 56 2 2
BOOKS  31 45 1 1
654321 10 11 1 2
FOOD   40 51 3 3
EMPTY  18 18 3 4
MONEY  15 17 2 3
FULL   52 52 2 4
WIRES  45 45 1 3
METAL  45 45 1 4
SCRAP  54 56 3 5
LIGHT  43 45 3 6
BELTS  44 45 2 5
HEAVY  32 33 2 6
HATS   88 88 1 5
DRINK  17 18 1 6
CANDY  22 22 3 7
SHOES  45 45 2 7
MISC   45 45 1 7
```

APPENDIX I – MAIN CONTROLLER CODE

```
//============================================================================
// Name        : controller.cpp
// Author      :
// Version     :
// Copyright   : Your copyright notice
// Description :
//============================================================================

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <iostream>
#include "Base.h"
//#include "Elevator.h"
```

```
#include "Prox_Sens.h"
#include "Wheel_Step.h"
#include "Dist_Sens.h"
#include "Refl_Sens.h"
#include "DC_Motor.h"
#include "Vac_ctrl.h"
#include "Switch_Reader.h"
using namespace std;

// variables
int err_check;
int myCommandLine[2];

// functions
int countStripes(int stripes);
//int countToHomeStripe(int stripes);
int moveToShelf(int shelf);
void parseCommandLine(int argc,char** argv);

// devices
Refl_Sens refl_sens_0 = Refl_Sens("/dev/refl_sens_0");
Refl_Sens refl_sens_1 = Refl_Sens("/dev/refl_sens_2");
DC_Motor arm_motor = DC_Motor("/dev/dc_motor_0");
DC_Motor elv_shelf_motor = DC_Motor("/dev/dc_motor_1");
DC_Motor sw_motor = DC_Motor("/dev/dc_motor_2");
Wheel_Step ws_0 = Wheel_Step("/dev/wheel_step_0");
Wheel_Step ws_1 = Wheel_Step("/dev/wheel_step_1");
Prox_Sens prox_sens = Prox_Sens("/dev/prox_sens_0");
Prox_Sens vac_sens = Prox_Sens("/dev/prox_sens_1");
Dist_Sens elv_level = Dist_Sens("/dev/dist_sens_0");
vac_ctrl vacCtrl = vac_ctrl("/dev/vac_ctrl_0");
Switch_Reader armSwitch = Switch_Reader("/dev/switch_reader_0");

int main(int argc, char* argv[]) {
        sleep(5);
        printf("take out the cord if you want to.\n");
        parseCommandLine(argc, argv);

        // stop the motor initially, make sure vacuum is off.
        ws_0.set_pin(2,0);
        ws_1.set_pin(2,0);
        //make sure the vacuum is off for starters
        vacCtrl.off();
        // make the elevator the standard for the h-bridge output
        sw_motor.noPower();

        // follow a line to the row we need
        countStripes(myCommandLine[0]);

        // raise elevator to height we need
        moveToShelf(myCommandLine[1]);

        //turn on the relay to switch control to the shelf motor
        sw_motor.forward();
        //start extending the shelf
        elv_shelf_motor.forward();
        usleep(850000);
        // turn off the shelf after .85 seconds
        elv_shelf_motor.noPower();
        //switch power back to the elevator, just to be safe, even though we're probably
retracting the shelf
        sw_motor.noPower();

        // turn on vacuum
        vacCtrl.on();

        long unsigned int i = 0;
```

```
        // send out the arm until a vacuum occurs
        arm_motor.forward();
        while (vac_sens.value() > 80 ) {
                usleep(100000);
                i++;
                if (i == 70) {
                break;
                }
        }
        //we have either achieved vacuum or totally missed the box.  either way, stop the
arm
        arm_motor.noPower();

        //ok, lets come back, the same amount of time we went out.
        arm_motor.reverse();
        while (i != 5) {
                usleep(100000);
                i--;
        }
        // we are far enough back, stop the arm again.  remember, the vacuum is on this
whole time
        arm_motor.noPower();

        vacCtrl.off();

        //turn on the relay to switch control to the shelf motor
        sw_motor.forward();
        //start extending the shelf
        elv_shelf_motor.reverse();
        usleep(850000);
        // turn off the shelf after .85 seconds
        elv_shelf_motor.noPower();
        //switch power back to the elevator, just to be safe, even though we're probably
retracting the shelf
        sw_motor.noPower();

        // go back to the shelf one height
        moveToShelf(1);

        // go back to the desk
        countStripes(8-myCommandLine[0]);

        //run up to the height of the operator's desk, hardcoded in as shelf #4
        vacCtrl.on();
        moveToShelf(5);

        // go out for six seconds to push the box off
        arm_motor.forward();
        sleep(1);
        vacCtrl.off();
        usleep(4500000);

        // give the operator 10 seconds to remove the box, maybe having to
        // to fanagle with the suction cup
        arm_motor.noPower(); // this is important, we forgot it once and the motor got
mad.
        sleep(3);
        // bring the arm back to a home-ish position, ready to start next time
        arm_motor.reverse();
        usleep(5400000);

        arm_motor.noPower();

        printf("done\n");

        return 0;
```

```
}

void parseCommandLine(int argc, char** argv) {
        int i;

        // parse the command line
        int shelf = 0;
        int row = 0;

        for (i = 1; i < argc; i++) {
                // check for what leading switch character
                if (argv[i][0] == '-') {
                        switch (argv[i][1]) {
                                case 's':       shelf = atoi(argv[++i]);
                                                break;

                                case 'r':       row = atoi(argv[++i]);
                                                break;

                                default: printf("error parsing input - no such
switch\n");
                                                exit(-1);
                        }
                } else {
                        printf("error parsing input - need to begin command with '-
'\n");
                        exit(-1);
                }
        }
        myCommandLine[0] = row;
        myCommandLine[1] = shelf;
}

int countStripes(int stripes) {
        // set so that we are going forwards
        ws_0.set_pin(3,1);
        ws_1.set_pin(3,1);
        usleep(100);
        // begin moving (set to run)
        ws_0.set_pin(1,1);
        ws_1.set_pin(1,1);
        // clear LCD screen
        printf("\e[j");
        printf("test 3");
        fflush(stdout);

        int num_stripes = 0;
        int edge_detect = 0;
        int edge_count = 0;
        // pin1 is for run, pin2 is for slowing down wheels, pin3 is for jogging backwards
        while (1) {
                if ((refl_sens_0.value() < 50000) && (refl_sens_1.value() >= 50000)) {
                        ws_0.set_pin(1,1);
                        ws_1.set_pin(1,1);
                        ws_0.set_pin(2,0);
                        ws_1.set_pin(2,1);
                        edge_detect = 1;
                        edge_count = 0;
                } else if ((refl_sens_1.value() < 50000) && (refl_sens_0.value() >=
50000)) {
                        ws_0.set_pin(1,1);
                        ws_1.set_pin(1,1);
                        ws_0.set_pin(2,1);
                        ws_1.set_pin(2,0);
                        edge_detect = 1;
                        edge_count = 0;
```

```
                } else if ((refl_sens_0.value() < 50000) && (refl_sens_1.value() <
50000)) {
                        if (edge_detect == 1) {
                                if (edge_count > 2750) {
                                        num_stripes++;
                                        printf("num_stripes: %i\n", num_stripes);
                                        fflush(stdout);
                                        if (num_stripes == stripes) {
                                                // stop the motors
                                                ws_0.set_pin(1,0);
                                                ws_1.set_pin(1,0);
                                                // set both motors to slow (just in
case)
                                                ws_0.set_pin(2,1);
                                                ws_1.set_pin(2,1);
                                                break;
                                        } else {
                                                // this is not our final stripe, so
                                                // we need to just go straight until
we are over it.
                                                do {
                                                        ws_0.set_pin(1,1);
                                                        ws_1.set_pin(1,1);
                                                        ws_0.set_pin(2,0);
                                                        ws_1.set_pin(2,0);
                                                } while ((refl_sens_0.value() <
50000) && (refl_sens_1.value() < 50000));
                                        }
                                        edge_detect = 0;
                                } else {
                                        edge_count++;
                                }
                        } else {
                                do {
                                        ws_0.set_pin(1,1);
                                        ws_1.set_pin(1,1);
                                        ws_0.set_pin(2,0);
                                        ws_1.set_pin(2,0);
                                } while ((refl_sens_0.value() < 50000) &&
(refl_sens_1.value() < 50000));
                        }
                }
                // if we are seeing two blacks just continue going at full speed.
                else {
                        ws_0.set_pin(1,1);
                        ws_1.set_pin(1,1);
                        ws_0.set_pin(2,0);
                        ws_1.set_pin(2,0);
                        edge_detect = 1;
                        edge_count = 0;
                }
                if (prox_sens.value() > 72 ) {
                        printf("someone got in the way.\n");
                        ws_0.set_pin(1,0);
                        ws_1.set_pin(1,0);
                        sleep(10);
                }
        }
        edge_detect = 0;
        edge_count = 0;

        // guarantee we have stopped the motor
        ws_0.set_pin(1,0);
        ws_1.set_pin(1,0);
        sleep(1);

        return 0;
```

```
}

int moveToShelf(int goal_shelf) {
        int goal_height_1 = 200000; // default to safe value.
        int goal_height_2 = 200000; // default to safe value.

        if (goal_shelf == 1) {
                goal_height_1 = 138000;
                goal_height_2 = 156000;
        } else if (goal_shelf == 2) {
                goal_height_1 = 322884;
                goal_height_2 = 352884;
        } else if (goal_shelf == 3) {
                goal_height_1 = 502000;
                goal_height_2 = 518000;
        } else if (goal_shelf == 4) {
                goal_height_1 = 189000;
                goal_height_2 = 153000;
        } else if (goal_shelf == 5) {
                goal_height_1 = 153000;
                goal_height_2 = 189000;
        } else if (goal_shelf == 6) {
                goal_height_1 = 167000;
                goal_height_2 = 152000;
        } else if (goal_shelf == 7) {
                goal_height_1 = 351000;
                goal_height_2 = 366000;
        } else if (goal_shelf == 8) {
                goal_height_1 = 546000;
                goal_height_2 = 531000;
        } else {
                printf("shelf number not recognized.");
                exit(-1);
        }

        int num_hits = 0;
        int current_value = 0;
        if (goal_height_1 != 200000) {
                while(1) {
                        current_value = elv_level.value();
                        if (abs(goal_height_1 - current_value) < 3750) {
                                num_hits++;
                                elv_shelf_motor.noPower();
                                if (num_hits > 2) {
                                        printf("goal shelf 1 reached: %i, %i\n",
goal_height_1, current_value);
                                        break;
                                }
                        } else if (current_value < goal_height_1) {
                                elv_shelf_motor.forward();
                        } else if (current_value > goal_height_1) {
                                elv_shelf_motor.reverse();
                        } else if (current_value < 135000) {
                                elv_shelf_motor.noPower();
                                printf("no power");
                        }
                }
        }

        // go to second height if need be
        num_hits = 0;
        if (goal_height_2 != 200000) {
                while(1) {
                        current_value = elv_level.value();
                        if (abs(goal_height_2 - current_value) < 3750) {
                                num_hits++;
                                elv_shelf_motor.noPower();
```

```
                                        if (num_hits > 6) {
                                                printf("goal shelf 2 reached: %i, %i\n",
goal_height_2, current_value);
                                                break;
                                        }
                                } else if (current_value < goal_height_2) {
                                        elv_shelf_motor.forward();
                                } else if (current_value > goal_height_2) {
                                        elv_shelf_motor.reverse();
                                } else if (current_value < 135000) {
                                        elv_shelf_motor.noPower();
                                        printf("no power");
                                }
                        }
                }

                if (goal_shelf == 3) {
                        elv_shelf_motor.forward();
                        usleep(400000);
                        elv_shelf_motor.noPower();
                } else if (goal_shelf == 2) {
                        elv_shelf_motor.reverse();
                        usleep(100000);
                        elv_shelf_motor.noPower();
                } else if (goal_shelf == 1) {
                        elv_shelf_motor.reverse();
                        usleep(150000);
                        elv_shelf_motor.noPower();
                } else if (goal_shelf == 5) {
                        elv_shelf_motor.forward();
                        usleep(200000);
                        elv_shelf_motor.noPower();
                }

                return 0;
}
```

APPENDIX J – MOVE MOTOR UTILITY PROGRAM

```
//============================================================================

// Name        : move_motor.cpp
// Author      :
// Version     :
// Copyright   : Your copyright notice
// Description :
//============================================================================

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <iostream>
#include "DC_Motor.h"
#include "Prox_Sens.h"
#include "Dist_Sens.h"
#include "Vac_ctrl.h"
#include "Wheel_Step.h"
#include "Refl_Sens.h"
using namespace std;

void parseCommandLine(int argc,char** argv);
int moveToShelf(int goal_shelf);
int testHeights();

// variables
int err_check;
//myCommandLine* commandline;

// parse the command line
char motor[5] = "";
int direction = 0;
unsigned int seconds = 0;
unsigned int usecs = 0;

// devices
DC_Motor arm_motor = DC_Motor("/dev/dc_motor_0");
DC_Motor elv_shelf_motor = DC_Motor("/dev/dc_motor_1");
DC_Motor sw_motor = DC_Motor("/dev/dc_motor_2");
Prox_Sens elv_prox_height = Prox_Sens("/dev/prox_sens_0");
Dist_Sens elv_dist_height = Dist_Sens("/dev/dist_sens_0");
vac_ctrl vacCtrl = vac_ctrl("/dev/vac_ctrl_0");
Wheel_Step rotatorMotor = Wheel_Step("/dev/wheel_step_2");
Refl_Sens refl_sens_0 = Refl_Sens("/dev/refl_sens_0");
Refl_Sens refl_sens_1 = Refl_Sens("/dev/refl_sens_2");
Dist_Sens elv_level = Dist_Sens("/dev/dist_sens_0");


int main(int argc, char* argv[]) {
        parseCommandLine(argc, argv);
        rotatorMotor.set_pin(1,0);
        rotatorMotor.set_pin(2,0);

        if (strcmp("elv", motor) == 0) {
                if ((direction) == 1) {
                        sw_motor.noPower();
                        elv_shelf_motor.forward();
                        printf("moving elevator up.\n");
                        fflush(stdout);
                        sleep(seconds);
                        usleep(usecs);
                        elv_shelf_motor.noPower();
                } else if ((direction) == 0) {
```

```
                sw_motor.noPower();
                elv_shelf_motor.reverse();
                printf("moving elevator down.\n");
                fflush(stdout);
                sleep(seconds);
                usleep(usecs);
                elv_shelf_motor.noPower();
        } else {
                printf("oh no error");
        }
} else if (strcmp("arm", motor) == 0) {
        if ((direction) == 1) {
                sw_motor.noPower();
                arm_motor.forward();
                printf("moving arm forward.\n");
                fflush(stdout);
                sleep(seconds);
                usleep(usecs);
                arm_motor.noPower();
        } else if ((direction) == 0) {
                sw_motor.noPower();
                arm_motor.reverse();
                printf("moving arm backward.\n");
                fflush(stdout);
                sleep(seconds);
                usleep(usecs);
                arm_motor.noPower();
        } else {
                printf("oh no error");
        }
} else if (strcmp("shelf", motor) == 0) {
        if ((direction) == 1) {
                sw_motor.forward();
                elv_shelf_motor.forward();
                printf("moving shelf forward.\n");
                fflush(stdout);
                sleep(seconds);
                usleep(usecs);
                elv_shelf_motor.noPower();
                sw_motor.noPower();
        } else if ((direction) == 0) {
                sw_motor.forward();
                elv_shelf_motor.reverse();
                printf("moving shelf backwards.\n");
                fflush(stdout);
                sleep(seconds);
                usleep(usecs);
                elv_shelf_motor.noPower();
                sw_motor.noPower();
        } else {
                printf("oh no error");
        }
} else if (strcmp("vac",motor) == 0) {
        if ((direction) == 1) {
                vacCtrl.on();
        } else if ((direction) == 0 ) {
                vacCtrl.off();
        }
} else if (strcmp("rot",motor) == 0 ) {
        if (direction == 1) {
                rotatorMotor.set_pin(1,1);
                usleep(250000);
                rotatorMotor.set_pin(1,0);
        } else if (direction == 0) {
                rotatorMotor.set_pin(2,1);
                usleep(250000);
                rotatorMotor.set_pin(2,0);
```

```
                }
        }
        printf("done\n");
        return 0;
}

void parseCommandLine(int argc, char** argv) {
        int i;
        for (i = 1; i < argc; i++) {
                // check for what leading switch character
                if (argv[i][0] == '-') {
                        switch (argv[i][1]) {
                                case 'm':       strcpy(motor, argv[++i]);
                                                break;

                                case 's':       seconds = atoi(argv[++i]);
                                                break;

                                case 'u':       usecs = atoi(argv[++i]);
                                                break;

                                case 'f':       moveToShelf(atoi(argv[++i]));
                                                exit(0);

                                case 'd':       direction = atoi(argv[++i]);
                                                break;

                                case 't':       testHeights();
                                                break;

                                case 'i':
                                                for (i = 0; i < 5; i++) {
                                                        printf("elv prox height sens
val:%i\n",elv_prox_height.value());

                                                        printf("elv dist height sens
val:%i\n",elv_dist_height.value());

                                                        printf("line follow refl
sens 0 val:%i\n",refl_sens_0.value());

                                                        printf("line follow refl
sens 1 val:%i\n",refl_sens_1.value());

                                                        usleep(1000);
                                                }
                                                exit(0);

                                default: printf("error parsing input - no such
switch\n");
                                                exit(-1);
                        }
                } else {
                        printf("error parsing input - need to begin command with '-
'\n");
                        exit(-1);
                }
        }
}

int moveToShelf(int goal_shelf) {
        int goal_height_1 = 200000; // default to safe value.
        int goal_height_2 = 200000; // default to safe value.

        if (goal_shelf == 1) {
                goal_height_1 = 138000;
                goal_height_2 = 156000;
        } else if (goal_shelf == 2) {
                goal_height_1 = 322884;
                goal_height_2 = 352884;
        } else if (goal_shelf == 3) {
```

```
                goal_height_1 = 502000;
                goal_height_2 = 518000;
        } else if (goal_shelf == 4) {
                goal_height_1 = 189000;
                goal_height_2 = 153000;
        } else if (goal_shelf == 5) {
                goal_height_1 = 153000;
                goal_height_2 = 189000;
        } else if (goal_shelf == 6) {
                goal_height_1 = 167000;
                goal_height_2 = 152000;
        } else if (goal_shelf == 7) {
                goal_height_1 = 351000;
                goal_height_2 = 366000;
        } else if (goal_shelf == 8) {
                goal_height_1 = 546000;
                goal_height_2 = 531000;
        } else {
                printf("shelf number not recognized.");
                exit(-1);
        }

        int num_hits = 0;
        int current_value = 0;
        if (goal_height_1 != 200000) {
                while(1) {
                        current_value = elv_level.value();
                        if (abs(goal_height_1 - current_value) < 3750) {
                                num_hits++;
                                elv_shelf_motor.noPower();
                                if (num_hits > 2) {
                                        printf("goal shelf 1 reached: %i, %i\n",
goal_height_1, current_value);
                                        break;
                                }
                        } else if (current_value < goal_height_1) {
                                elv_shelf_motor.forward();
                        } else if (current_value > goal_height_1) {
                                elv_shelf_motor.reverse();
                        } else if (current_value < 135000) {
                                elv_shelf_motor.noPower();
                                printf("no power");
                        }
                }
        }

        // go to second height if need be
        num_hits = 0;
        if (goal_height_2 != 200000) {
                while(1) {
                        current_value = elv_level.value();
                        if (abs(goal_height_2 - current_value) < 3750) {
                                num_hits++;
                                elv_shelf_motor.noPower();
                                if (num_hits > 6) {
                                        printf("goal shelf 2 reached: %i, %i\n",
goal_height_2, current_value);
                                        break;
                                }
                        } else if (current_value < goal_height_2) {
                                elv_shelf_motor.forward();
                        } else if (current_value > goal_height_2) {
                                elv_shelf_motor.reverse();
                        } else if (current_value < 135000) {
                                elv_shelf_motor.noPower();
                                printf("no power");
                        }
```

```
                }
        }

        if (goal_shelf == 3) {
                elv_shelf_motor.forward();
                usleep(400000);
                elv_shelf_motor.noPower();
        } else if (goal_shelf == 2) {
                elv_shelf_motor.reverse();
                usleep(100000);
                elv_shelf_motor.noPower();
        } else if (goal_shelf == 1) {
                elv_shelf_motor.reverse();
                usleep(150000);
                elv_shelf_motor.noPower();
        } else if (goal_shelf == 5) {
                elv_shelf_motor.reverse();
                usleep(200000);
                elv_shelf_motor.noPower();
        }

        return 0;
}

int testHeights() {
        int current_value = 0;

        sw_motor.noPower();
        elv_shelf_motor.forward();
        for (int i = 0; i < 15; i++) {
                current_value = elv_level.value();
                printf("state: %i, %i\n", i, current_value);
                usleep(30000);
        }
        elv_shelf_motor.noPower();

        return 0;
}
```