CSE3120 – Digital Systems Project

Soccer Robot for RoboCup Competition

Second Semester Report

Integration Group

Mentor: Dr. Bin Qiu

Subject Leader: Mr. Gordon Lowe

Integration Members:
Ozcan Huseyin
   Lida Lu
   Bao Ngu
   Tze-Vernn Yong

Submission Date: 10-11-1999
# TABLE OF CONTENT

- **2ND SEMESTER PROJECT** .................................................................................................................. 3
- **ACKNOWLEDGMENT:** ....................................................................................................................... 3
- **ABSTRACT:** ........................................................................................................................................ 4
- **INTRODUCTION:** ............................................................................................................................... 5
- **METHODOLOGY:** .............................................................................................................................. 5
  - **SENSORS** ........................................................................................................................................ 6
  - **COMMUNICATION:** ........................................................................................................................ 7
  - **SOFTWARE:** .................................................................................................................................... 8
  - **MOTION CONTROL:** ......................................................................................................................... 9
  - **FIELD POSITIONING SYSTEM (FPS):** .............................................................................................. 10
  - **ARTIFICIAL INTELLIGENCE:** .......................................................................................................... 12
  - **MECHANICAL:** ............................................................................................................................... 12
  - **INTEGRATION:** ............................................................................................................................... 13
- **CONCLUSION:** ................................................................................................................................. 17
Acknowledgment:

We would like to acknowledge and thank our Mentor, Dr Bin Qui, for his support and guidance throughout the year. He kept the group on track and made sure we managed our tasks accordingly.

We would like to thank QNX and PE Electronics for their sponsorship of the SBC, additional hardware components and the QNX operating system for our project. Their prompt service and technical guidance are greatly appreciated.

We appreciate tremendously RMIT’s support and supplying us with the relevant information to making this possible. Thanks to Andrew Jenning ‘s for his expertise in the field of soccer robots.

We also give thanks to our Project leader, Gordon Lowe, for his supervision of the overall project and leadership management of our whole project group.

We would like to greatly acknowledge Gary Evans for his tremendous help and technical instructions for all groups in the project. Thanks Gary!

And finally last but not least we want to thank everyone who participated in the soccer robot project. We appreciate everyone for making this all possible.
Abstract:

Following the first semesters research activities, the Hardware groups role changed to that of Integration. This meant that the Integration group had to redefine their roles in the overall project. The definition had to include the details of the work that was required from an integration perspective. Primary our role was to communicate and become intermediaries between the various groups that required some form of integration. It was our job to detail the interfaces between the groups on both a software and hardware perspective.

Continuing from last semesters work, the groups already had a solid understanding of the tasks that needed to be completed by the end of 2nd semester. Details of the work performed by each of the groups are as follows:

**Software:** The software group was responsible for the selection and maintenance of their selected operating system. They were also responsible for ensuring programming standards to make programs written by other groups easier to read and understand.

**Sensors:** Continuing from last semesters work, the sensors group was responsible for the vision system of the robot. Their tasks involved were grabbing frames from a frame grabber card and the image processing software to determine distances and distinguish various objects on the field.

**Motion Control:** They are in control of the robots movement on and of the field. They are required to write software to communicate with FPS so initiate controlled movement.

**Field Positioning System:** An integral part of the system that requires a lot of communication with other groups. System components include compass and comm port communication systems. Their other responsibilities include calculating, keeping track of the location and positioning of the robot.

**Communications:** Continuing from their task last semester, the communications group had to develop the wireless communication/control system for the soccer robot. Their system components included a car alarm modified in various ways to produce a select amount of states (i.e. stop, go and reset) responsible for the overall control of the robot.

**Artificial Intelligence:** They will provide orchestrated movements to control the various sections of the soccer robot in a very abstract way. This includes developing code to map the field and make decision on the robot orientation, movement and tactics.

**Mechanical:** This group is responsible for fitting together all the components of the robot. Building from their work from first semester, their task include installing the motors on the chassis, purchasing of the batteries and finding locations for the components to build the robot.

This semester, our role as the Integration group in the project is provide support on integration needs for other groups. Since most of the system components have already been purchased, the integration group had a relatively well defined role. Our initial task was to develop a Gantt that summarized our requirements for the second semester of the project. Having completed this, the huge task remaining for the integration’s group was to collect integration specifications from each group. Having collected this information, we needed to inform the various groups of the interfaces that each of the groups have decided on. This ensures that all groups will have a simplified integration’s exercise, thereby reducing the time and problems associated with the integration process.

Our other task for the semester was the purchasing decision for the SBC (Small Board Computer). This decision has to coincide with the software groups selected operation system called QNX (pronounced Q-nix). Changes were made from our original SBC decision due to unforeseen complications with QNX and manufacturers supplies. This lead to the purchasing of the POS-560 Pentium Processor based Control Board.
Introduction:

In the second semester, our task in the project had changed from a hardware perspective to an integration perspective. This resulted in a change in project aim. Initially, our role was to research and locate the best solution to the hardware problem associated with the soccer robot. However, once QNX had formally sponsored the project, a group of QNX engineers had undertaken to find a board that best suited our specifications. Having become redundant in the project, we were allocated the task of system integration. This seemingly daunting task required a lot of task redefinition. This made our task difficult, as most of the groups had already defined a specification based on their 1st semesters work. Having said this, our task was to collect information based on these interfaces and the subsequent policing of these restrictions.

Details of the other groups interfaces and the overall system connection are included in the report. These diagrams will include system connections from abstract system connections through specific group connections. Once these have been designed and agreed upon, the next task was to police the development of these systems.

Another section of study was the design of a development guideline for the other groups in the project. Our requirements were to initiate development by other groups that should essentially prove the operation and integration of specific sections of the robot. Each group that required some complex integration exercise had a development guideline to follow (designed by the integration group) so to make the integration a smooth and modular process.

The other major role in the integration’s group was the setting up and maintenance of the SBC (once it was purchased). Some of the tasks included setting up of the SBC, researching the various operations of the SBC components and installing QNX where by providing a platform for further development.

Methodology:

Initially we had to redefine our task at hand to fit our role as integration group. Our first role is to develop an Gantt chart that best predicted our roles through out the semester. This proved to be a difficult and time consuming task as it required a lot of information gathering from various groups. The information gathered was to provide a detail knowledge into the operation and design of the various systems. This knowledge was to provide our group with enough data to design a global integration strategy. Having completed the Gantt chart, we had to use the information gathered from the other groups to document a overall systems connection as per the integration group. This information had to be agreed upon by all interacting groups that will be affected by the integration exercise.

Figure 1: Abstract interaction diagram.
The kind of information that we required from the groups was data that were essentially being passed between groups. The Integration groups set out to collect this data by bringing forward questions to groups regarding their specifications at this point. We reviewed this information, provided feedback to the group, and lead them onto the correct path for the integration exercise. This particular task gave us an overview and a thorough understanding of the individual interactions between groups.

Once the information had been gathered, the next task was to compile individual interface designs to produce a benchmark for each system. By knowing the interfaces and interactions of each group, we could provide a better “custom make” integration solution to each of the groups. The following section of the report will be the detailed analysis of each group interface.

**Sensors**

The interface of the sensors software is a global “bucket array” used to hold information about the various objects around the robot. The sensors program will use the frame grabber to sample the immediate environment and collect information from a matrix of pixels. Their software has been specifically designed to threshold, measure and calculate distances from objects. Once these objects have been calculated, the information is stored in a global array that is visible to anyone who wishes to access the information. Information is guaranteed to be valid at the frequency of the frame acquisition. The bucket array is illustrated per Figure 2.

![Figure 2: Sensors Global Bucket Array](image)

This information is available to all the groups that require the object information. The primary user of this information is Artificial Intelligence. They use the information to plan a relative path for the soccer robot. This information has been acknowledged by A.I as standard method of object information passing.

The program that will be calculating this bucket array is the primary algorithm used by sensors. The frame grabbing software will also be required to allow a software controlled frame grabbing process, i.e. pictures of the environment will be acquired when they are requested. Using this method, sensors will be able to have more control over the frame grabbing process, thus a more efficient implementation. At this point of the report, inter-process communication and the benefits this method provides for the purpose of modular design will be discussed. Using the previous abstract software model (figure 1), inter-process communication is essential for simultaneous program execution, whereby each block in the diagram represent an executable program. Each of the separate program requires spawning by some other program. The program code that spawns a process is as follows:

```
PID = spawnl (P_NOWAIT , "<directory>" , arg0 , arg1 , ... , argN , mode);
```

Once a process has been spawned, inter-process communication must be used to pass arguments to and from each process. This can be seen in Figure 1 as arrows, going from one block to the other block.
The method which must be employed is by first send and receive functions (use to perform inter-process communication). To send a message to a process, the send function that carries these argument are used.

\[ \text{Send (PID, "<message>", length of message, flags);} \]

Following the send function, the process that was intended to receive the sent message requires the Receive function call.

\[ \text{Receive (PID, message [], message length);} \]

Using these complimentary function calls, inter-process communication is a viable solution to the modular design for the software of the soccer robot.

**Communication:**

The comm’s group was responsible for the communication system of the soccer robot. Their task was well defined and relatively set out. Their aims over the semester were well defined which made their integration with the soccer robot simple and straightforward. A car alarm system was used as the medium on which some form of communication will be made. The system includes a smart node that is connectable to a COM port on a computer. Packets of data are sent through the serial port to the communications device (car alarm), which then control the car alarm in a specific manner. The states that were used in their design were GO, STOP, RESET and COMM ERROR. These states are used by FPS to ascertain the state of the robot and allow some form of wireless control. Having this system is very beneficial to the system, as control of the robot can be acquired from a substantial distance. During the testing phase of the overall system, the communications device could be used to stop the robot when it is out of control.

Their task was to complete a communications program that implemented a packet driven communications protocol. This had to be written in ANSI C and compiled using the Watcom compiler so the device can be controlled in QNX. Their system connection is as follows:

![Diagram of the communication system](image)

**Figure 3: Communication’s Wireless System.**

The state machine above shows the block model of the communications systems. The various states in the design of the communications system are described below:

- **RESET** – This state is responsible for the re-initialization of the soccer robot. Once the reset state has been activates, the entire system will undergo a re-initialization. This is discussed further in the report.
- **ERROR** – The error state is an indication of a corrupt transmission of packet data. During the error state, the packet will be resent.
- **STOP** – During the stop state, the soccer robot will be idle, waiting for a GO command. Therefore, the stop state is a mild reset.
• GO – In the GO state, the robot will be initialized (if not already done) and the robot will be playing soccer by trying to locate the ball and moving towards the ball.

Software:

QNX was the software groups selected Operating System. QNX is a real time operating system. The differences between a non-real time operating system, such as Windows 95/98, and QNX is the kernel. QNX’s kernel has been designed so that the process management is one that conforms to a real time system. Real time performance of an Operating System is highly advantageous. Its reaction to specific conditions must be performed in a time critical manner. Without this time-critical performance, certain functions could not be guaranteed execute in the appropriate manner. This could cause severe problems in the system performance. This operating system was deployed onto two machines during the software development process so all aspects of QNX could be investigated and learned.

Software revision control was the other major area of interest to the software group. Their tasks were to keep track of the software development process and to organize some guidelines for other groups to follow. The guidelines included:

• **Commenting and documentation of the program code.** This is a very important aspect of software development as the information will be used by students in years to come. This was also one of our very important roles. It was one that the entire integration’s group spent a lot of time looking at other code created by groups such as FPS, Sensors, AI and Communications groups.

• **Program structure and readability.** This is also a very important section of code development. Each individual has their own means of structuring their program. Our task in this section of the project was to keep a standard of programming by changing other students program code to the standard that we were trying to deploy. An example of this is as follows:

Some students types the following:

```c
void lots_of_args(char* string, int num1, int num2, float decimal, int num3, double num4)
```

This would have been better written as:

```c
Void lots_of_args(char* string,
int num1,
int num2,
float decimal,
int num3,
double num4);
```

This is one of many standards that was enforced by the integration’s group to ensure readable and portable code.

• **Good error handling.** This is an essential aspect of the programming code. All memory allocations must be checked for NULL and provide a graceful handling of these conditions. This also includes all file opening, that was a very common aspect of students code.

• **Freeing of memory.** If memory has been allocated, it is essential to free the memory when the functions have completed using the variable.

• **Naming Variables and Functions.** The names of global variables and functions in a program serve as a comment of some sort. Therefore, the choice of the variables and function names need to be selected so that they give useful information about the meaning of the variable or function. Local variable names can be shorter, because they are used only within one context, where (presumably) comments explain their purpose.

Software group had taken on the role of “help desk” with regards to QNX. Their priorities were the operation and maintenance of the QNX operating system. Their knowledge of the operating system proved very useful to all groups developing code and using QNX for some purpose.
Integration Group

**Motion Control:**

Motion control is the least abstract group in the whole project. Their task was very well defined, giving them a definite task to have completed by the end of the second semester. Continuing with the micro mouse board, they used the 68HC11 to control the actuators (motors) of the soccer robot. Motion Control have developed program code to respond to given signals from FPS to move in the appropriate manner given commands across a serial connection. The program that they develop will need to allow the robot to move left, right and forward as well as accelerate and decelerate.

The microprocessor system that the Motion control group is using for their system is interfaced to the SBC through a serial COM port. The Motion control program will send and receive instruction data to and from FPS, one instruction at a time. These instruction signals are represented by an 8-bit opcode where only the bit being sent is active high. There are 7 instruction signals that are used to interface FPS and Motion Control as shown below.

**Signals sent by Motion Control:**

<table>
<thead>
<tr>
<th>BIT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5. INST</td>
<td>signals that the current instruction has been completed by Motion control and is ready for the next the instruction.</td>
</tr>
<tr>
<td>4. STR</td>
<td>signals FPS that the straight-line motion part of the instruction is complete.</td>
</tr>
<tr>
<td>7. ENC</td>
<td>signals an encoder count transmission. It will only transmit encoder count update for the straight-line motion part of an instruction.</td>
</tr>
<tr>
<td>6. INT</td>
<td>signals a Motion interrupt by the Motion control hardware.</td>
</tr>
</tbody>
</table>

![Figure 4: 8 bit opcode of instruction signals sent by Motion Control to FPS.](image)

Bits 0 through 3 are unused and are available for further system expansion. These instructions will be used by FPS as acknowledgment signals and therefore keep track of the operations performed by motion control.

Motion Control also needs to accept some signals from FPS as input to allow a command based design to be implemented. A command base design allows Motion Control to react to a given set of stimulus signals by performing the task that is required. This is done across a serial cable. AI will be responsible for the compilation and calculation of commands that will be sent to FPS. FPS is then responsible for the mechanism for organizing the signals into a bit pattern and then send the information down the serial line to Motion Control. The byte of information that is to be sent to motion control is as follows:

**Signals received by Motion Control**

<table>
<thead>
<tr>
<th>BIT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2(\rightarrow 0). VELx</td>
<td>Velocity instruction required for straight line motion. Has a selection of 8 possible speed settings.</td>
</tr>
<tr>
<td>3. BALL</td>
<td>Represents the ball flag. This means that the ball is currently in the possession of the soccer robot</td>
</tr>
<tr>
<td>4. F/B</td>
<td>Signifies the forward or backward motion of the current movement</td>
</tr>
<tr>
<td>5. L/R</td>
<td>Signifies the current rotation direction i.e. left or right.</td>
</tr>
<tr>
<td>6. NOT USED</td>
<td></td>
</tr>
<tr>
<td>7. INT</td>
<td>Signifies and FPS interrupt to motion control. This instruction is used to force motion control to stop the current instruction, come to a complete halt, and report status back to FPS.</td>
</tr>
</tbody>
</table>

![Figure 5. 8-bit opcode of instruction signals set to Motion control from FPS](image)
Integration Group

Motion control is required to communicate across a serial cable. The serial communications interface is controlled by the 68HC11 operating system called BUFFALO. BUFFALO has routines that can be called using the 68HC11-instruction set to fetch data from the buffer allocated to the serial port. Once some data is written to the serial port, an interrupt is set and BUFFALO will respond by grabbing and filling the buffer with the data streamed down the serial line. This data needs to be latched and stored into memory for further analysis. Therefore, a serial communication program must also be written by the motion control group to intercept the data coming down from the serial line. The program code needs to be interrupt driven. Interrupts are more beneficial to the design in this case due to the real time characteristics required from the motion control group.

Motion Controls initialization routine is by far the most complicated to implement of all the groups. Once the 68HC11 have had the power disconnected, the memory of the micro-controller needs to be reinitialized, i.e. reloaded with the motion control program. For this reason, motion control needs to be reinitialized every time the power is disconnected from the soccer robot, or the soccer robot is reset. The re-initialization process is as follows:

- Power up the 68HC11 microprocessor when the power is applied to complete system
- Establish the communications with the 68HC11 via the serial interface.
- Download the S-Record down the serial line to complete the program loading into the memory of the 68HC11.
- Send the command “GO 4000” to run the program on the 68HC11. At this stage the motion control object is waiting form some input form FPS.

This completes the initialization of the motion control object. The S-Record of the file is stored in an array within a program that is responsible for the initialization of the motion control object. Program is compiled with the up-to-date S-Record of the motion control (which is hard coded into the program object). This makes the FPS module a more complete program.

Field Positioning System (FPS):

FPS is the group with the most interfaces in the entire project. Their interaction with the complete system is the most complicated and the one that needs the most amount of integration. Sensors, Artificial Intelligence, Communications and Motion Control are all required to integrate with the FPS object. Therefore, this section of the project needed a great deal of analysis and design. Initially, the most important section of their project was the completion of a “skeleton” that illustrated the structure and design of the FPS main module. This main module will be accepting data and information from most of the groups in the project. Some of this information is also used in calculations and passed to other modules in the system.

The skeleton of the FPS system was the first requirement. Designing the skeleton was an important section of the FPS integration step, due to the FPS module being a major integration exercise. All of the information that the FPS module uses is from other systems. Therefore, an intelligent design is required so to avoid complications at a later stage. The FPS module was selected to be split into two sections: The Main Module and the Instruction Queue Module. A diagram of the interaction of the two modules and the interaction of the FPS system with all communicated information can be seen in Figure 6. The majority of the design was completed by the FPS group with integration’s group as the consultants. With the FPS and integration’s teams working together the final design was agreed upon and the skeleton implementation had begun. At this point the modular design (as discussed earlier in the report) was agreed upon, the group began the arduous task of learning inter-process communication, and the problems associated with such a design. However, this was discussed to be a viable solution to the problem and one that FPS must stick to. This methodology introduced design constraints on the other groups. Inter-process communication requires the spawning of other programs by a hierarchically superior process. Because FPS was the most interface intensive process, it was up to FPS to spawn all the processes at a specific time in the initialization process.

At this section of the project, the initialization process will be discussed. Initialization is a very important section of the design and need to be incorporated in to the skeleton of the FPS main module. Initially, the FPS module will be executed. Once the execution has begun, the various modules will be
Figure 6: Main module and Instruction module of overall interaction with FPS. Spawned and initialized according to the requirement of the process. List of the processes that need to be spawned are as follows:

1. Instruction Queue Module for the sending of movement instructions.
2. Motion control program that is stored in the S-Record.
3. FPS compass module to allow direction information.
4. Sensors Object that is responsible for the image processing.
5. The Sensors Object will then spawn the Grab Module that is responsible for the image acquisition via the frame grabber card.
6. Communications object will then be spawned.
7. After all the processes have been spawned, the Artificial Intelligence module will need to be spawned to allow complete control over all the processes via the abstract FPS interface.

The overall controlling process will be the AI process, although the FPS process controls the entire initialization procedure. This level of abstraction is required to control the FPS module through the sending of simple *global* instructions. The FPS module/object will then react to the software signals sent by AI in such a manner so to control all the movement tasks of the robot. FPS will be also managing the compass information and using this information with the AI commands to pass information to the instruction module. To complement this information, the FPS module will be using the bucket array data compiled by the sensors group as information about the surrounding environment. Therefore, the FPS group will be compiling information from all the section of the robot and using the information is some form so to present the data to AI. This point leads onto the integration of AI and FPS.
**Integration Group**

**Artificial Intelligence:**

AI will be the simplest group to integrate into the complete soccer robot system (with respect to software interface). The group’s task throughout the entire project was to complete a soccer simulator and path planner. These two software programs will be used to test the game playing strategy and tactics that will be used during the game. Their only interface lies with FPS. Therefore, their integration is a relatively simple and straightforward one. Having said this, both FPS and AI need to describe their interfaces to the integration’s group to complete an interface design.

Inter-process communication is required between the two processes when they are executing dynamically. This is achieved by the use of the send( ) and receive( ) functions. Artificial intelligence requires the following information to calculate a path plan:

1. **Objects in the immediate environment of the soccer robot.** This information is gathered by the sensors group and is made available globally. Therefore, the AI group can have direct access to the information to the data once it is made available. FPS instructs sensors to grab and analyze an image, therefore, the image capture/refresh rate relies on the FPS module.
2. **Current location of the soccer robot.** Even though AI have a running calculation of the current position of the soccer robot with respect to AI’s internal map, FPS will still be the primary information source for field location coordinates.
3. **Movement completion information.** Once an instruction is supplied by AI and fed to motion control via the instruction queue, motion control should complete the designated command and halt. Once the motion control command is completed, an acknowledgment should be sent back to the AI group to allow them to confirm the new coordinates. If the command could not be completed by motion control for some reason, AI should not be allowed to update their internal map.

All of the above information needs to be supplied to the AI group for the successful calculation of tactics.

The tactics (better known as task planner behaviors) will be programmed to allow different cases of behavior during the game, depending on the surroundings and current state of the game. These tactics are move robot, find the ball, acquire ball, attack with the ball, kick goal, and tackle other robot. With these states of game-play, the artificial intelligence will be able to play a game of soccer. Therefore, it is obvious that the AI component of the soccer robot will need the appropriate information supplied to it during the game in order to perform the calculation required to enter each state of game play (mentioned previously).

**Mechanical:**

The mechanical group’s task was to plan, design, and build the overall physical appearance of the soccer robot. Their tasks included; designing the chassis according to the official specification, researching motor types and performance, kicker and the battery for power supply to robot. Throughout the second semester, they also worked on building the entire soccer robot by connecting all the components together.

Placement of all of the hardware (i.e.: the SBC, the motion control board and the kicker relay board) components needed to be well planned. The components needed to be mounted in a way to provide easy access to ports and pin connectors for testing and integration needs. All the components in the system needed to fit into a limited space.

The mechanical group used an aluminum cage as the main body of the robot. This allowed ample room for the components if the soccer robot. However, certain problems were still persistent when integrating all the components into the chassis. They were as follows:

1. **The aluminum box is metal so it can conduct electricity.** The group had to make sure that the SBC does not touch any other electrical components or the chassis itself.
2. **Dc - Dc converters.** Had to be mounted on the exterior of the chassis. This could be a problem if the other robots accidentally contact the robot. Any form of damage could potentially disconnect or damage the power converters.
3. **The tower for the camera must be tough and stable.** Shaky tower will cause the mirror to move resulting in inaccurate frame grabbing.
4. **Impact of the kicker.** A strong kicker could make the soccer robot to jolt backwards.
5. *Motors over-heating.*
6. *The weight of the soccer robot.* Excess weight will put strain on the motors, which could cause premature overheating and excessive power consumption.
7. *Accessibility of components.* Having all the components accessible is a major advantage. If any components were to be damaged the replacement of the components will be simple and quick.

A diagram on how the components are placed is as below:

![Diagram of Mechanical components placement of the soccer robot.](image)

*Figure 7: Mechanical components placement of the soccer robot.*

*Integration:*

In this section of the report, we will list our achievement in the project. Although the collection and management of the information already presented in the report was part of our role definition, we also performed numerous other tasks throughout the project.

One of our major tasks throughout the semester was the purchasing and configuration of the SBC that was purchased by the department. The model of the SBC was POS-560. It included a variety of components that would best suit an embedded application. These components were very useful to the overall project. All of the groups were given the required resources allowing a very flexible design routine.

Once the SBC was purchased, the initial task was the installation of QNX operating system (after we mounted the board onto a temporary housing facility). We decided to install QNX onto the Disk On Chip for extra speed, increased performance and reliability. The Disk On Chip model was the M-Systems Disk On Chip 2000. M-Systems has developed a new technology for writing information onto the DOC (Disk On Chip) that has virtually eliminated the volatile nature of the flash disk. Having the added benefit of QNX compatibility also helped the operating system installation. All the appropriate documentation was available on the supplied disk and on the Internet. The order of the tasks to install QNX onto the DOC is as follows:

1. Unpack the required software from the Utilities disk onto a temporary hard drive.
2. Create a link to the Disk On Chip driver by typing the following:
   ```
   >> ln -f ./Fsyst.diskonchip /bin/Fsys.diskonchip
   ```
3. Format the DOC by typing the following command: (This will erase any other information on the DOC).
   ```
   >> dformat
   ```
4. Once the device has been formatted, we need to set the DOC as a raw device by typing the command:
5. Partitioning the DOC is the next stage of the QNX installation. To partition the device, we need to type in the following commands:

```bash
>> fdisk /dev/tffs0 add -f 1 QNX ALL
>> fdisk /dev/tffs0 boot QNX
>> fdisk /dev/tffs0 loader
```

6. Having completed all of the formatting and partitioning, we can proceed with the installation of QNX. This requires a CD-ROM because the QNX software comes on a CD-ROM medium.

7. The QNX installation is similar to that of any other operating system such as Windows 98. However, certain functionality’s of QNX had to be removed due to DOC size restrictions. The components that were not included were:
   - Photon Micro GUI
   - Watcom Compilers
   - Help and documentation

Having installed QNX, we were required to test the installation and for various assurances. The integration’s group was required to prove the operation of specific drivers such as serial ports (and other peripherals, especially the Digital I/O ports), video card and floppy disk controller. This completed the integration of the operating system with the SBC.

After the installation of QNX on the SBC, we concentrated on integrating the kicker system. The mechanical group had spent time researching the various ball propulsion systems. The best solution that they offered was a pneumatic piston controlled by an electric pulse. The electric pulse switches a piston inside kicking mechanism that allowed compressed air to push a rod at high velocity. The system required two components, one is the container that holds the compressed air and the other is the kicker (piston).

Another essential component in the kicker system is the relay circuitry that is responsible for supplying sufficient current to switch the piston. The relay circuitry is controlled by the digital I/O pins on the SBC. This required research into the operation and usage of the digital I/O pins. This information was gathered from the technical documentation provided by the SBC supplier. The pin layout of the digital I/O is as follows:

```
  | QND | In 3 | In 2 | In 1 | In 0 |
---|------|------|------|------|------|
 9 |      |      |      |      |      |
 7 |      |      |      |      |      |
 5 |      |      |      |      |      |
 3 |      |      |      |      |      |
 1 |      |      |      |      |      |
 2 |      |      |      |      |      |
 10|      |      |      |      |      |
```

There are two high-drive digital output pins and four digital input pins. The requirement for the kicker is only one digital output. This output pin will be used as a software-controlled trigger that provides current to the piston device. When testing the digital I/O pins, the voltage swing on the output was only 0.2 volts (where it should have been 5 volts). The reason for the poor voltage swing was the missing pull up resistor. Once we connected the pull up resistor to the digital I/O pins 2 and 8, the 5-volt output swing was apparent. Now the pins of the digital output were ready to be connected to the relay circuit board. The complete wiring diagram of the kicking system can be seen illustrated in figure 9.
The complete code for the program that controls the kicking mechanism can be seen in appendix #1. This code will be included in AI’s module, as they are the system that will be making the decisions on when to kick the ball (or to the opponent!).

One of the issues that were raised in class was the large scale of the programming exercise of the soccer robot. Most groups require some software module to perform or control some aspect of the robot. As the size of the code increases, so does the level of difficulty with respect to error detection or tracking. When discussing this potential problem in class, the integration’s group came up with the idea of a logging system. By logging, each software module is required to have some extra lines of code that are embedded in strategic locations to inform the user of potential problems or problems that have occurred in the system.

This was a challenging problem. The functionality of the logging system needs to be dynamic and changeable. The definition of the problem and solution are as follows:

Problem Definition:

- All logging statements need to be within a file.
- The logging system has an option to be activated or disabled.
- Levels of logging are also required to achieve a level of abstraction. By having a “level” system, the user can turn off/on specific levels according to the required logging information. This allows for a more fine-tuned logging system where the user can request more information from the logging code, or decrease the level of information.
- All groups require individual files for which the logging statements will be stored in.

Problem Solution:

- Have a system where there is 4 levels of abstraction in terms of logging statements. The levels of the logging have been defined as:
LEVEL 1 – INITIALIZATION AND EXIT FUNCTIONS. Logging statements require to be placed after every initialization. An example of an initialization is the spawning of a process or opening of a file.

LEVEL2 – FILE WRITING AND FILE DELETING. Once a file has already been opened (level 1) any output to a file needs to be logged. Also any files that are deleted during the execution process of the code needs to be logged.

LEVEL 3 – INTER-PROCESS MESSAGE PASSING. Because inter-process message passing is a freely used function and one that is of great importance, (i.e. the user needs to know if a message HAS been successfully passed between two processes), we need to place logging statements after every message passing function call.

LEVEL 4 – PROCESS SPECIFIC DATA. Any information that is specific to the process in which the data is located in. An example of this is values of function variables.

- Each group will need to have a tailor made solution to their integration of the logging system. This means an integration group member will have to be present at all times during the integration process.
- The interface for the logging will be as follows:

  \[ \text{Log}_{\text{GROUP}}(\text{LEV}x, \_\_\text{FILE}\_\_, \_\_\text{LINE}\_\_, \text{"message"}); } \]

  The \text{GROUP} for the above interface the group name, i.e. SENSORS. The \text{LEV}x is the level of logging that will be activated. The \_\_\text{FILE}\_\_ places the file name of which that logging statement is located in. The \_\_\text{LINE}\_\_ will indicate the line number of the logged call. And finally, the “message” is the user defined message that will be logged in the logging file. Each group will have their individual \text{GROUP.log} file that is stored in the common directory named “/home/global/logs”. To activate the various levels of logging, the user must specify the level required at \textit{compilation time}. An example of this is as follows:

  Assuming the code is in the file named “code_file.c” and the levels of logging required are 1, 2, 4. The command line for the compilation is as follows:

  \[ \text{gcc code_file.c -o code_file -DDEBUG -DLEVEL1 -DLEVEL2 -DLEVEL4} \]

  The only drawback of using a compilation system to switch the levels of logging on and off was the compilation required at every time the levels needed to be changed. If we had some extra time for the project, a more dynamic system would have been better. During the testing of the logging program, we realized that the logging system was very effective.

  Policing the logging system was the main part of our task when it was up and running. We had to tell all the other groups how the logging file was to be used and where it was to be placed. Thus what we did was just sit with the groups and tell them when or where they should place the logging code. This will help them to find problems that they feel is useful when debugging.

  We also had to police when other groups should want to use the SBC. This means at some stage there is more than 2 group would want to use the SBC at one time. Thus, our group would have to differentiate the priority of each group when they required the services of the SBC. Higher priority groups will be given the SBC first. We made a logbook on who was last o use it. Groups that used the SBC had to sign in and sign out, whilst mentioning what they did.

  Once all the system connections were completed and the soccer robot took shape, the selection of the SBC usefulness became apparent. The POS-560 SBC was a very versatile system. Its setup time was very short and its compatibility with existing hardware was very good. By not having to worry about the compatibility issues (that would have been a problem if another SBC had been selected) the integration time of the computer was short. This was advantageous to the overall system especially when QNX was installed at a lightning pace. The system speed was very impressive and the size of the RAM proved very useful. Our image processing requirements needed a very fast microprocessor system. The extra RAM on the SBC system was advantageous when the frame grabbing program was introduced to the system. Seeing as the system did not have a hard drive, the need for extra storage room was solved by the creation of a 10 mega byte RAM disk. All the images that were captured by the frame grabbing software was stored on the RAM disk for analysis by the image processing code.
Having extra serial ports also proved very advantageous for a dynamic resource allocation. Some of the groups changed their specifications throughout the project, and having extra ports meant their changed designs were not restricted in any way. We could offer each of the groups any resource they needed because of the vast resources available in the SBC system. Digital I/O’s are an example of this. When the mechanics group decided to have a kicking mechanism (as specified earlier in the report), the availability of the digital I/O pins meant the kicker could be interfaced via the most efficient mechanism. Otherwise, the kicker would have needed a smart node and extra software to provide the appropriate signals.

To complete the methodology, the final system connections as recorded by the integration’s group is as follows:

![Complete system connection diagram](image)

**Figure 10. Complete system connection diagram**

**Conclusion:**

This section of the report will be used to sum the contributions of each group members. Following is brief indication of the task undertaken by all the group members in the integration’s group.

**Ozcan Huseyin**

1. Researched the components of the POS-560.
2. Communicated with all the other groups to learn their interfaces.
3. Worked with the Sensors group in the frame grabbing software.
4. Loaded QNX onto the Disk On Chip.
5. Designed the Logging system used by all groups in the project.
6. Integrated the logging system with the groups.
7. Designed and implemented the kicker system for the soccer robot.
8. Policed the standards set by the integration’s group with respect to software interface.
9. Performed Project management skills over the semester.
10. Defined the role of the Integration’s group

**Lida Lu**

1. Researched the components of the POS-560.
Integration Group

2. Communicated with all the other groups to learn their interfaces.
3. Helped to load QNX onto the Disk On Chip.
4. Continuously reviewed the Gantt chart and updated areas of weakness.
5. Compiled a list of requirements needed by other teams with respect to communication methods.
6. Helped design the logging system.
7. Policed the standards set by the integration’s group with respect to software interface.

*Bao Ngu*

1. Researched the components of the POS-560.
2. Communicated with all the other groups to learn their interfaces.
3. Helped determine the new tasks that the integration group had to perform.
4. Policed the standards set by the integration’s group with respect to software interface.
5. Compiled a list of requirements needed by other teams with respect to communication methods.

*Tze-Vernn Yong*

1. Researched the components of the POS-560.
2. Communicated with all the other groups to learn their interfaces.
3. Helped to load QNX onto the Disk On Chip.
4. Researched the DC-Converters.
5. Mounted the SBC into the robot chassis with help of the mechanics group.
6. Provided assistance in the design of the logging system.
7. Performed all the small task needed by all the groups in the project.

After reaching the deadline of the project and not being able to complete the desired task, we were forced to re-visit the project stages and summarize the successes and failures. The integration’s group feels that the project was well designed but the implementations were not sufficient. The majority of the groups left their tasks too late or did not design their organization well. As a consequence, the most important section of the project was delayed and eventually not completed. This section was the motion control. If this section of the project was to have been completed in the time frame, other groups would have had ample testing time and integration time to complete the final project. This would have given the project a very good chance of success. However, this is a lesson for the following years. We predict that future students should be working heavily on this section of the project.

For future reference, we believe that the project has been designed very well in terms of the software model and hardware devices. The communications devices will need to be upgraded as more robots are added to the field. This will rely more heavily on the Artificial Intelligence aspect of the project.

As a final remark, the whole of the integration’s group had a very fun and enjoyable time with the project. The interaction with all the group members was very interesting and rewarding. The digital systems group grew closer as a team and have forged friendships that will last forever.
#include <stdio.h>

#define DEBUG 1 /* turn on the logging */
#define LEVEL1 1 /* turn LEVEL 1 logging */
#define LEVEL2 0
#define LEVEL3 0
#define LEVEL4 0
#define LEV1 1 /* turn LEVEL 1 logging */
#define LEV2 2
#define LEV3 3
#define LEV4 4

/* function prototypes */
int hello1(char h[]);
int log(int level, int line, char[40]);

int main()
{
    int hello;
    int i;

    for(i=0 ; i < 10 ; i++)
        printf("%d ", i+1);

    printf("\n");

    printf("__LINE__ = %d\n", __LINE__);
    printf("__FILE__ = %s\n", __FILE__);
    printf("__DATE__ = %s\n", __DATE__);
    printf("__TIME__ = %s\n", __TIME__);
    printf("__STDC__ = %d\n\n\n", __STDC__);

    log(LEV1,__FILE__, __LINE__, "HELLO WORLD");

}

int testing(char h[])
{
    printf("Have called the function testting\n");
    printf("%s\n", h);
}

int log(int level, int line, char string[]) {
    
    #ifdef DEBUG
    printf("LOGGING Activate\n");

    #ifdef LEVEL1
    if(level==1)
        printf("Level ONE logging activated: \n ** %s ** \nON LINE: %d\n", string, line);
    #endif

    #ifdef LEVEL2
    if(level==2)
        printf("Level TWO logging activated\n");
    #endif

    #ifdef LEVEL3
    if(level==3)
        printf("Level THREE logging activated\n");
    #endif

    #ifdef LEVEL4
    if(level==4)
        printf("Level FOUR logging activated\n");
    #endif
    #endif

}

APPENDIX #2

#include <stdio.h>
#include <conio.h>
#define DIG_IO 0x220
#define OUT1 0x01
#define OUT2 0x02

void main() {
    int choice;
    while(1){
        printf("Kicker program\n\n");
        printf("0 - kick 0\n\n");
        printf("Choice: ");

        scanf("%d", &choice);

}
if(choice == 0)
{
    // apply a pulse to the dig1 output line
    outp(DIG_IO, OUT1);
    // dissable the pulse to the dig2 output line
    outp(DIG_IO, OUT2);
}
}