CRITICAL DESIGN REPORT

1. OVERALL SYSTEM
   1.1 Overall System Design
   1.2 Overall Timing Analysis
   1.3 Overall Visual Feedback
   1.4 Interface Control
   1.5 Overall Test Plan
   1.6 Overall Requirements Analysis
   1.7 Overall Risk Assessment
   1.8 Overall Budget Analysis
   1.9 Overall Management Plan

2 USER INTERFACE/CONTROLLER BLOCK
   2.2 Sub-System Specifications
   2.3 Test Plan
   2.4 Requirements Analysis
   2.5 Risk Assessment

3 NETWORKING BLOCK
   3.2 Sub-System Specifications
   3.3 Test Plan
   3.4 Requirements Analysis
   3.5 Risk Assessment

4 LOW-LEVEL BLOCK
   4.2 Sub-System Specifications
   4.3 Test Plan
   4.4 Requirements Analysis
   4.5 Risk Assessment

5 MANAGEMENT & WORK BREAKDOWN STRUCTURE (ATTACHMENT)

6 APPENDIX/SCHEMATICS
   Appendix A: Bill of Materials
   Appendix B: Schematics
1. **OVERALL SYSTEM**

1.1 **Overall System Design**

The Lafayette Automated Rapid Transit Control System (LART-CS) is a versatile and user-friendly system that controls the Lafayette Rapid Transit system. The Lafayette Rapid Transit system is a model train system that controls multiple trains over the Lafayette College campus and has stations at various locations.

*Figure 1.1 – Top-level System Diagram*
The system is composed of three main parts: UI/Controller (Located on the MasterPC), Network (Implemented via PIC Microcontrollers & RS-232), and Low-Level hardware (interfaces with existing infrastructure & cabling).

The MasterPC shall communicate to each station via RS-232 standards, using the packet structures defined in section 3.1. To avoid collisions of using a single wire as the ether, the communication shall be done in parallel via the Octopus 550 (described in section 2.1).

The information coming from the MasterPC shall be analyzed via the PICs at each station. These PICs shall not only control switching, but also send the rail & speed information to secondary SlavePICs specialized in generating unique PWM signals. Furthermore, the MasterPIC shall collect sensor information periodically via a polling scheme, and send information back to the MasterPC.

The power shall be supplied from a 120V wall plug, which feeds into a surge protector. The output of this feeds both the MasterPC and the A/C-D/C converter (located one per station) which will be used to supply the rest of the system with the desired power levels.
1.2 Overall Timing Analysis

![Overall Timing Diagram](image)

**Figure 1.3 – Overall Timing Diagram**

- **Delay 1:** FlipFlop Delay = 1msec
- **Delay 2:** MasterPIC Delay = 41usec
- **Delay 3:** PIC-to-PC Delay = 2.50msec
- **Delay 4:** PC Delay = 10msec
- **Delay 5:** PC-to-PIC Delay = 9.17msec
- **Delay 6:** PIC-to-PIC Delay = 5.83msec
- **Delay 7a:** SlavePIC delay = 20usec
- **Delay 7b:** MasterPIC Delay = 5.6msec
- **Delay 8a:** H-Bridge Delay = 1.1usec
- **Delay 8b:** Switch C. Delay = 5.7msec

Path-a Delay = Delay1 + Delay2 + Delay3 + Delay4 + Delay5 + Delay6 + Delay7a + Delay8a
= 1 + 0.041 + 2.5 + 10 + 9.17 + 5.83 + 0.020 + 0.0011
= 28.5621msec

Path-b Delay = Delay1 + Delay2 + Delay3 + Delay4 + Delay5 + Delay6 + Delay7b + Delay8b
= 1 + 0.041 + 2.5 + 10 + 9.17 + 5.83 + 5.6 + 5.7
= 38.841msec → Critical Path

According to our tests, the maximum speed of the train is 50cm/sec. The most critical point on the layout is the 5cm distance between a sensor and the start of a switching rail. This gives us 0.1sec = 100msec to react after a train passed that sensor, and to activate the switch so the train can turn. Thus, we can conclude that we are well below the critical reaction time.
1.3 Visual Feedback

Each of the 3 sub-groups will have visual feedback where it is most necessary:

The UI group will have the GUI that provides all necessary feedback for the operator. Details of this can be seen in Figure 1.3

The Networking Group will have LEDs on TxD and RxD pins of PIC-PC connection, and on the single ether of the PIC-to-PIC communication. Details can be seen in Figures N-9 and N-11.

The Low-Level Group will have LEDs on all the output PWM signals of the H-Bridge. Details can be seen in Figures A-13 and A-14.

1.4 Interface Control

Interface control is attached as a separate document. This document includes the wiring structure, control signal accounting, as well as the labeling for each station. Also included are the connections from the PICs to the 50 pin grey cable and from the 50 pin grey cable to the flip flops. Finally this document includes the RS-232 communication inputs and outputs.

For further details refer to the Interface Control Document.

1.5 Overall Test Plan

Refer to the System Acceptance Test Plan for more detail.
### 1.6 Overall Requirements Analysis

#### Technical Requirements as defined in the Statement of Work

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Software</th>
<th>Network</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>R001 rail switch control</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>R002 engine power control</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>R003 train proximity monitoring</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>R004 expandability and adaptability</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>R005 control and monitoring speed</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>R006 applications programming interface</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R007 maintenance user interface</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R008 demonstration application</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>R009 modifications of the CFE layout</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>R010 power input</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### General Project Requirements as defined in the Statement of Work

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Software</th>
<th>Network</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPR001 documentation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR002 environmental</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR003 EMI/EMC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR004 hazmats</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR005 safety and good practice</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR006 reliability</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR007 maintainability</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR008 sourcing sustainability</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR009 global sustainability</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR010 ethics report</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>GPR011 project demonstration</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR012 final disposal of projects</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

#### Ethical Requirements as defined in the Ethics Report

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Software</th>
<th>Network</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER001 trains will not crash</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ER002 train acceleration</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

#### Reliability Requirements (defined below)

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Software</th>
<th>Network</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>RR001 Mean Time Between Failure</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>RR002 System life</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Maintainability Requirement (defined below)

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Software</th>
<th>Network</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR001 Mean Time To Repair</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

RR001: Mean Time Between Failure
A system wide MTBF will be greater than 100 hours over the system lifetime

RR002: System Life
The system life shall be 5 years

MR001: Mean Time To Repair
A system wide MTTR will be less than 1 week over the system lifetime.

Detailed descriptions of how each of the individual requirements will be met by each group can be found in sections 2.4, 3.4 and 4.4.
1.7 Overall Risk Assessment

The riskiest subsystems of the project have been designed but there still remains an amount of uncertainty as to whether or not they will perform as expected. This is due to inexperience with the technology and limitations of certain elements that are hard to predict without test results. Many of these issues deal with timing issues. The speed with which data is sent between the PC and station boards is fast compared with how long it takes to make the appropriate changes at a station. On the other end, with five different inputs into the PC on different ports, the API must be able to work quickly enough to interpret this load of data without losing information. The API has an important job of sending and receiving on all five ports. The switch controlling could create timing issues because of the relatively long amount of time an output needs to be set in order to discharge the capacitor.

The polling subsystem is also a potentially problematic system because of necessary intervals in which the sensors must be checked. If a polling routine is interrupted too often or fails in some other way the trains will not be seen. On the PC the maintenance mode is risky because of the inexperience with XML code.

Lower risk subsystems include the hardware found at each station. These circuits are much easier to analyze at a design phase and problematic situations can be better avoided. The GUI interface is relatively risk free. The rail powering at each station is a fairly simple design as well because of the separate slave PICs solely for creating the PWM signals. The RS-232 link also a fairly simple system because the PICs are designed for this type of communication.

<table>
<thead>
<tr>
<th>Subsystems</th>
<th>Calculated Risk Factor</th>
<th>Risk Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>UI/API Group: Maintenance/Demo User Interface</td>
<td>0.199</td>
<td>Low</td>
</tr>
<tr>
<td>UI/API Group: Maintenance/Demo Control</td>
<td>0.484</td>
<td>Medium</td>
</tr>
<tr>
<td>UI/API Group: API</td>
<td>0.397</td>
<td>Medium</td>
</tr>
<tr>
<td>Networking Group: RS-232Communication</td>
<td>0.377</td>
<td>Medium</td>
</tr>
<tr>
<td>Networking Group: Sensor Polling</td>
<td>0.531</td>
<td>High</td>
</tr>
<tr>
<td>Networking Group: Rail control</td>
<td>0.370</td>
<td>Medium</td>
</tr>
<tr>
<td>Networking Group: Switch control</td>
<td>0.599</td>
<td>High</td>
</tr>
<tr>
<td>Low-Level Group: Overall system power</td>
<td>0.3778</td>
<td>Medium</td>
</tr>
<tr>
<td>Low-Level Group: Rail power</td>
<td>0.3778</td>
<td>Medium</td>
</tr>
<tr>
<td>Low-Level Group: Rail Switching</td>
<td>0.377</td>
<td>Medium</td>
</tr>
<tr>
<td>Low-Level Group: Sensors</td>
<td>0.398</td>
<td>Medium</td>
</tr>
</tbody>
</table>
1.8 Overall Budget Analysis

1.8.1 Direct Costs:

Materials used in the project include the Octopus-550, PICs, PCBs, resistors, capacitors, cables, heat sinks, etc. totaling $1,433.53. For a detailed list of all the materials used in the project and their associated costs, please see the Bill of Materials in Appendix A. Additionally, the cost of fabrication of the boards is a direct cost. This is estimated to be $600.00

*Total direct costs: $2,033.53*

The ECE department has allotted $4,000 for the project. Our current estimate is approximately 50% of this figure, so any cost overrun we experience should be manageable.

1.8.2 Indirect Costs:

Labor (2590 Hours)

*Total indirect costs: $51,800.00*

To date we are spending approximately 185 hours per week on the project. We expect this average to continue for the second half of the semester, so that the total number of hours worked during the project will be 2590. If we are paid $20.00, the cost of labor will be $51,800 for the entire project.

*Total costs for project: $53,833.53*

The total cost for the project is just under $53,833.53, with the vast majority of this money coming from the indirect cost of labor. The total direct cost to the ECE department is estimated to be just over $2,000. Even if the direct costs were to overrun by 50%, the project would still cost less than the $4,000.00 that the ECE department is able to devote to the project.
1.9 Overall Management Plan

Management of the project up to the Critical Design Review (CDR) has been based on tasks being modularized and being assigned to the three major design groups UI/Controller, Networking and Low-level. Each of these teams has a team leader Austin, Serdar and Alex respectively who is responsible for their group’s tasks and meeting deadlines. The overall management of the schedule and top-level tasks were done by Taha. Technical documentation and CDR documentation was divided among each of the groups and a group of people assigned to compiling and reviewing the documents (Shrijan, Taha, Alex). Weekly status letters were divided up among each of the team members. Other project documentation was assigned among pairs of team members with relevant deadlines for their completion so that they can be reviewed.

Post CDR, the management structure will be reorganized so that communication, task assignment are more streamlined. There will be a management team consisting of (Shrijan, Taha, Alex) responsible for updating the task list and schedule on a weekly basis, writing up the weekly status letters and communicating tasks and deadlines to their respective teams. Tasks will still be assigned based on a group basis.

Completion of the remainder of the project will be based around the major milestones of the purchasing deadline, manufacturing deadline and project completion. The aim of the project team is to complete all networking design before spring break, send out the first batch of PCB fabrication before spring break so that it can be fully tested before the second complete batch of PCB is sent out for fabrication. As far as the software components of the system, the aim of the team is to complete all software design by the manufacturing deadline so that all system-level testing and integration can begin after that, up to project completion.

Please review the project schedule attached in Section 5.0.
2. **User Interface / Controller Block**

### 2.1 UI/Controller Sub-System Specifications

Top Level Diagram:

![Figure 1.1 API Diagram](image-url)

*Figure 1.1 API Diagram*
The GUI and all the coding will be programmed in JAVA and AWT/Swing.

2.1.1 **USER INTERFACE:**
The maintenance GUI is for a train operator who would use it to monitor the trains and to control their movements and speed. It will also include an emergency stop button that the operator could press if there is potential for disaster. The maintenance GUI will include combo boxes to control the speed of the train on each rail segment. In addition it will include buttons for each switch and radio buttons for detecting sensor action. The demonstration GUI includes a log to inform the user of the positions of the trains. In addition there are buttons to start the demonstration and to stop the trains.

2.1.2 **DEMONSTRATION CONTROL:**
The demonstration control logic will choose which stations the trains will be running, what speed they will be running at, and the direction they will be moving. The logic will also have a failsafe, which would detect an error if for example, trains are going too fast, and then the whole system will pause. The operator can also at his discretion stop the whole system. This also sends the sensor signals to the GUI.

2.1.3 **MAINTENANCE CONTROL:**
The maintenance control logic will send method call backs to control the track layout, including switches, sensors, and speed control at each track segment.

2.1.4 **API:**

The ControllerMain class will parse the XML document to get characteristics for each station. This will create a map for each station. It sends the values for each station to the control classes. It receives data from the BuilderDecoder class which sends sensor values to the control classes. In addition it sends information from the XML document to the Station class.

The Station class gets information from the Switch, Sensor, and Rail class to build each station.

The BuilderDecoder class organizes information from the ControllerMain class about the switches and rails and transmits it through the serial port. In addition it receives sensor information from the SensorListener Interface and sends it to the ControllerMain class.

2.1.5 **OCTOPUS 550:**
The Octopus 550 is a Peripheral Component Interconnect (PCI) to Serial-RS232 port expansion card. It will be placed in the Master PC and supports up to 8-serial ports. This card will be used to connect to each of the stations in parallel via serial cables.
Attached UML DIAGRAM
Attached MAINTENANCE AND DEMONSTRATION GUIS
Express Logic

If (1.2) then stop and wait
If (1.3) then next stop Skillman
If(3.2) then approaching Skillman station
If(3.3) then stop and wait
If(3.4) then next stop Metzgar
If(5.2) then approaching Metzgar
If(5.3) then stop and wait

If(5.2) then next stop Skillman
If(3.4) then approaching Skillman
If(3.3) then stop and wait
If(3.2) then next stop 3rd street
If(1.3) then approaching 3rd street
If(1.2) then stop and wait

Train 1 Local

If(1.0) then stop and wait and next stop Sullivan and switch 1
If(2.6) then approaching Sullivan
If(2.7) then stop and wait and next stop Skillman
If(3.6) then approaching Skillman
If(3.7) then stop and wait and next stop Farinon and switch 6
If(3.5) then approaching Farinon and switch 9
If(4.6) then stop and wait and next stop Metzgar
If(4.7) then approaching Metzgar and switch 11
If(5.1) then stop and wait and next stop Farinon

If(4.2) then approaching Farinon
If(4.1) then stop and wait and next stop Skillman
If(3.2) then approaching Skillman
If(3.1) then stop and wait and next stop Sullivan
If(2.2) then approaching Sullivan
If(2.1) then stop and wait and next stop 3rd street
If(1.1) then approaching 3rd street

Train 2 Local

If(5.1) then stop and wait and next stop Farinon
If(4.2) then approaching Farinon
If(4.1) then stop and wait and next stop Skillman
If(3.2) then approaching Skillman
If(3.1) then stop and wait and next stop Sullivan
If(2.2) then approaching Sullivan
If(2.1) then stop and wait and next stop 3rd street
If(1.1) then approaching 3rd street
If(1.0) then stop and wait and next stop Sullivan and switch 1
If(2.6) then approaching Sullivan
If(2.7) then stop and wait and next stop Skillman
If(3.6) then approaching Skillman
If(3.7) then stop and wait and next stop Farinon and switch 6
If(3.5) then approaching Farinon and switch 9
If(4.6) then stop and wait and next stop Metzgar
If(4.7) then approaching Metzgar and switch 11
2.2 UI/Controller Test Plan

The software tests for the system will be a multi step process, where we test all software modules from the lowest level to the highest. The first step in the process will be to individually test all classes in the API. For most of the classes these will be unit tests written using the JUnit testing framework. For the BuilderDecoder class, we will write a simple test program and look at the serial port either using HyperTerminal on a different computer or on the oscilloscope to ensure the packets we are sending are correct. To do this our test program will create a new instance of a Station class and we can then test various cases. For example, we can call the build method and tell all rails to stop and all switches to remain the same. In this case we would expect to see a packet on the oscilloscope that had 10 bytes transmitted, each with the value 00000000. We can then call the setRailValue method to set a new value for one rail and the resulting packet would be 10 bytes, and only the one byte corresponding to the rail we changed would be non-zero. Finally, we can call the setSwitchValue method to change the value of one switch. For this packet we would expect one bit of the first byte to be asserted corresponding to the switch we changed, as well as the same rail byte from before (because we never set this rail value back to zero).

The next step in the testing process will be to test the implementation of the maintenance user interface. To do this we will perform a series of operations on the interface itself such as changing individual rail and switch values. This will essentially be the same as when we did unit testing on the BuilderDecoder, except now the commands will be coming from the maintenance interface rather than from a test program. Once we know the BuilderDecoder can correctly form and send out packets then we can simply print out the byte stream that will be sent and ensure that it is correct. Alternately, we can connect the serial port to an oscilloscope to make sure the correct data is being transmitted. Finally, we will create a test program to simulate changing sensor values and watch the screen to make sure the values get updated as we expect.

Testing the demo application will be a little more difficult because this interface accepts very little user input. We can use the same testing program to simulate the incoming sensor data. We will ensure the control logic can correctly handle the events by watching the printouts on the interface, as well as watching the resulting packets being sent out on the oscilloscope.

Once we are confident in all of the software modules, we will move on to integration testing. We can start by testing that we can receive incoming sensor data from the PICs by having them run a test program that simply sends new data after a fixed time interval. After that, we can move on to a full system test using the maintenance interface. This will allow us to comprehensively test every part of the system one-by-one. We can use the interface to issue commands and watch the physical layout to make sure all the commands were received and interpreted accurately. Finally, once we are sure all parts of the system are behaving correctly, we can test our control logic for the demo application by using the physical system and monitoring the results.
### 2.3 UI/Controller Requirements Analysis

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Maintenance</th>
<th>Demonstration</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>R001 rail switch control</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>R002 engine power control</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>R003 train proximity monitoring</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>R004 expandability and adaptability</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>R005 control and monitoring speed</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>R006 applications programming interface</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>R007 maintenance user interface</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>R008 demonstration application</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>R009 modifications of the CFE layout</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>R010 power input</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Maintenance</th>
<th>Demonstration</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPR001 documentation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR002 environmental</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR003 EMI/EMC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR004 hazmats</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR005 safety and good practice</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR006 reliability</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR007 maintainability</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR008 sourcing sustainability</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR009 global sustainability</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR010 ethics report</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR011 project demonstration</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR012 final disposal of projects</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Maintenance</th>
<th>Demonstration</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER001 trains will not crash</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ER002 train acceleration</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**R001 Rail Switch Control:**
The maintenance and demonstration control will use the API to send to the hardware to specify positions for rail switches.

**R002 Engine Power Control:**
The maintenance and demonstration control will use the API to send signals telling the hardware to control the engine power.

**R003 Train Proximity Monitoring:**
The hardware will communicate train locations with the computer through the API. The UI will display locations on the screen and if the system is in demonstration mode, the controller will monitor their positions.
**R004 Expandability and Adaptability:**
All four sections are running on a PC. If anything in the system needs to be expanded the software will be able to handle it with minimal redesign using the API and XML document. Extensive recompilation of the software won’t be necessary.

**R005 Control and Monitoring Speed:**
The PC and hardware will communicate very quickly compared to the train speeds so all control and monitoring will be accurate.

**R006 Applications Programming Interface:**
The fully-documented API has the capabilities that require an applications programmer to write software applications that control, monitor and engine all power interfaces. The scope is sufficient to support both low level debugging applications and high-level automated applications.

**R007 Maintenance User Interface:**
Our user interface will allow a user to control and monitor all the track switches, train proximity sensors, and engine speed control of each track segment. The maintenance user interface conforms to the API and runs on a suitable hardware platform. Maintenance procedures will be outlined in the maintenance plan document.

**R008 Demonstration Application:**
Our demonstration control will allow anybody to demonstrate the function of the system. In the interface, there will be a log so informational messages on the position of the train can be notified to the user. The control will demonstrate 2 local and 1 express trains and our control logic will provide a tolerance for possible mistakes by the user and potential disasters, such as collisions and system damage, will not be able to happen. The demonstration user interface conforms to the API and runs on a suitable hardware platform.

**R009 Modifications of the CFE Layout:**
All four sections are running on a PC and will not need any modification to the overall layout.

**R010 Power Input:**
The computer can run from a standard wall outlet.

---

**GPR001 Documentation:**
The API is well documented and in addition, block diagrams, UML diagrams, and detailed system description. Also all the code and executable binaries will be in the commonly accepted formats.

**GPR002 Environmental:**
All the sections are running on the PC and the PC meets the temperature requirements.

**GPR003 EMI/EMC:**
All the sections are running on the PC and the PC meets the radiation requirements.

**GPR004 Hazmats:**
All the sections are running on the PC and the PC meets the hazard requirements.

**GPR005 SAFETY AND GOOD PRACTICE:**
All the sections are running on the PC and we won’t be putting massive voltages into the computer. Also in terms of coding, we will be commenting before writing the code, which meets the good practice requirement.

**GPR006 RELIABILITY:**
All the sections are running on the PC and the PC meets the reliability requirements. The system wide MTBF is greater than 100,000 hours over the system lifetime.

**GPR007 MAINTAINABILITY:**
All the sections are running on the PC and the PC meets the maintainability requirements. The system wide MTTR is less than 1 day over the system lifetime.

**GPR008 SOURCING SUSTAINABILITY:**
The code can be programmed and used in different IDE’s and a PC from any PC maker can be used.

**GPR009 GLOBAL SUSTAINABILITY:**
All four sections are running on a PC and will not need any modification to the overall layout.

**GPR010 ETHICS REPORT:**
Not related to UI Group.

**GPR011 PROJECT DEMONSTRATION:**
The demonstration control will be designed for a continuous, unattended display that would excite everyone who attends the demonstration. The demonstration will be able to deactivate automatically after some short delay, if the demonstration is activated by someone.

**GPR012 FINAL DISPOSAL OF PROJECTS:**
All source code and documents will be zipped and stored in a separate folder.

---

**ER001 TRAIN WILL NOT CRASH:**
In the demonstration control, we will have failsafe coded so that such a situation isn’t possible.

**ER002 TRAIN ACCELERATION:**
Not related to UI group.
2.4 UI/Controller Risk Assessment

2.4.1 MAINTENANCE/Demo User Interface

These interfaces are designed for the user to control all signals (speed and direction) that will be sent out to the stations.

<table>
<thead>
<tr>
<th>Methodology Maturity</th>
<th>Methodology Complexity</th>
<th>Software Maturity</th>
<th>Software Complexity</th>
<th>Dependency of Schedule on External Functional Groups</th>
<th>Dependency of Performance on External Functional Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Potential for Failure Factor = (0.1 + 0.1 + 0.1 + 0.1 + 0 + 0)/6 = 0.067

<table>
<thead>
<tr>
<th>Degradation of Technical/Operational Performance at System Level</th>
<th>Schedule Impact</th>
<th>Cost Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9</td>
<td>0.3</td>
<td>0</td>
</tr>
</tbody>
</table>

Consequence for Failure Factor = (0.9 + 0.3 + 0)/3 = 0.4

**Risk Factor**: \( P+C - (P*C) = 0.067 + 0.04 - (0.067 * 0.4) = 0.199 \) low risk

2.4.2 MAINTENANCE/Demo Control

These control modules monitor the maintenance and demonstration operations to make sure the correct signals going out to the stations.

<table>
<thead>
<tr>
<th>Methodology Maturity</th>
<th>Methodology Complexity</th>
<th>Software Maturity</th>
<th>Software Complexity</th>
<th>Dependency of Schedule on External Functional Groups</th>
<th>Dependency of Performance on External Functional Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.7</td>
<td>0.1</td>
<td>0.3</td>
<td>0.1</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Potential for Failure Factor = (0.1 + 0.7 + 0.1 + 0.3 + 0.1 + 0.1)/6 = 0.23

<table>
<thead>
<tr>
<th>Degradation of Technical/Operational Performance at System Level</th>
<th>Schedule Impact</th>
<th>Cost Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7</td>
<td>0.3</td>
<td>0</td>
</tr>
</tbody>
</table>

Consequence for Failure Factor = (0.7 + 0.3 + 0)/3 = 0.33

**Risk Factor**: \( P+C - (P*C) = 0.23 + 0.33 - (0.23 * 0.33) = 0.484 \) medium risk
2.4.3 API

The API is the underlying framework of the lower level methods that communicate between serial port and the PC.

<table>
<thead>
<tr>
<th>Methodology Maturity</th>
<th>Methodology Complexity</th>
<th>Software Maturity</th>
<th>Software Complexity</th>
<th>Dependency of Schedule on External Functional Groups</th>
<th>Dependency of Performance on External Functional Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Potential for Failure Factor = \( \frac{0.1 + 0.1 + 0.1 + 0.3 + 0 + 0}{6} = 0.1 \)

<table>
<thead>
<tr>
<th>Degradation of Technical/Operational Performance at System Level</th>
<th>Schedule Impact</th>
<th>Cost Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9</td>
<td>0.1</td>
<td>0</td>
</tr>
</tbody>
</table>

Consequence for Failure Factor = \( \frac{0.9 + 0.1 + 0}{3} = 0.33 \)

**Risk Factor:** \( P + C - (P \times C) = 0.1 + 0.33 - (0.1 \times 0.33) = 0.397 \) medium risk
3. NETWORKING BLOCK

3.1 Networking Sub-System Specifications

The overall networking system is composed of two main parts: the MasterPIC and 3 SlavePICs. The MasterPIC is in charge of communicating with the MasterPC as well as collecting sensor information and powering the switches. The MasterPIC receives rail speed & polarity and switch information and sends back sensor information. The SlavePICs are in charge of providing PWM signals and polarity information to H-Bridges, in order to power the rails. The duty cycles for PWM signals and the polarity information are sent to the appropriate slaves by the MasterPIC each time a packet arrives from the MasterPC.

(Refer to Figure 1.2 for detailed station diagram.
Refer to Appendix-N2 for layout details & pin connections)
3.1.1 **MASTERPIC**

The MasterPIC has 3 main tasks:
1) Receiving packets with rail speed & polarity and switch information from MasterPC, and distributing them to the SlavePICs (*Packet reception & Distribution*)
2) Powering the switches as commanded by the MasterPC (*Switch Control*)
3) Collecting sensor information and sending packets to the MasterPC (*Sensor Polling*)

### 3.1.1a RS-232 Communication

Both PC communication (PC-to-PIC & PIC-to-PC) and PIC-to-PIC communication is done via RS-232 protocol. The speed of the RS-232 communication is currently set to the default 9600Kbs. If we need to increase communication speed to meet strict timing requirements, we can do so later on in the schedule, as changing speed does not affect any hardware. The current speed of 9600KBS is chosen due to its well balanced speed & error trade-off.

**PIC-to-PC station packet** length is 2bytes, consisting of 1byte of information about the MasterPIC’s station number and 1 byte of CRC. The station byte is only sent once during power-up so the MasterPC can determine which stations lies on which COM ports. Only the first 3bits of the first byte are used to encode the station number. Details of this packet can be seen in Figure 3.2b.

After that, any 1byte packet that is sent by the MasterPIC to the MasterPC or vice-versa will be treated as an error byte, forcing the sender to re-transmit the last packet. Details of an error byte can be seen in figure 3.2c.
PC-to-PIC standard packet length is 11 bytes, consisting of 9 rail bytes, 1 switch byte, and 1 CRC byte. Each rail byte contains the necessary speed & polarity information for a given rail pair. The switch byte contains the necessary information to manage 4 switches. Thus the packet enables us to control all stations, since any given station has maximum 9 rail pairs and 4 switches. Details of a single rail and switch byte can be seen in figures 3.3a and figure 3.3b.

In the switch byte, every 2 bits contain the information for a single switch. The even numbered bits control the Left coil (0 = no power, 1 =power) and the odd numbered bits control the Right coil. A 1-1 combination would not be acceptable as it would damage the switch, and will be watched out for by the MasterPIC, in case such information arrives due to errors.

In the rail byte, the first 4 bits contain the speed information for the rail pair, enabling the encoding of 16 different speeds ranging from 0-F. Bits 4 & 5 contain the polarity information for the two rails in the pair. Bit 4 is for type-A Rails and bit 5 is for type-B Rails (Refer to ICD for naming schemes). Bit6 is used for ‘Emergency Stop’, and if it’s ‘1’, then the PWM will not undergo a gradual change while stopping. The last bit is currently unused.

Every time a packet arrives, it creates an interrupt in the MasterPIC. By receiving packets on an interrupt basis as opposed to polling, the MasterPIC has more time to attend to other tasks. In this interrupt service routine, the MasterPIC puts the switch byte in a queue (as described in 3.1.1c) and distributes the 9 rail bytes to the 3 slaves in the following way:

- **R0-R1-R2 → Slave0**
- **R3-R4-R5 → Slave1**
- **R6-R7-R8 → Slave2**

Because PIC-to-PIC communication is done on single wire ether, each PIC needs their own ID in order to correctly exchange information. IDs 0-3 are reserved for Slaves (currently 0-2 is used) and ID 4 is the MasterPIC. Whenever there’s a packet on the ether, each SlavePIC will check if the ‘to’ address of the packet matches their ID. It accepts the packet if the address matches, and rejects it otherwise.

PIC-to-PIC standard packet length is 7 bytes long made of 3 preliminary bytes, 3 rail bytes, and a CRC byte. It contains the following:

1. 1 byte ‘to’ data, denoting which PIC the packet comes from
2. 1 byte ‘from’ data, denoting which PIC the packet is going to
3. 1 byte ‘length’ data, denoting the length of the packet
4. 3 bytes of ‘rail’ data, containing the exact same information as the original packet from the MasterPC
5. 1 byte of ‘CRC’ data, used to check for errors in transmission
Figure 3.4a – PIC-to-PIC Standard Packet Structure

Figure 3.4b – PIC-to-PIC Error Packet Structure

Even if the station to be controlled does not have 9 rail pairs, the MasterPIC will still assume that it does, and divide the original 9 byte packet into three 3-byte packets, and distribute them to three SlavePICs. (Why we have chosen to use 3 SlavePICs will be discussed in the Section 3.2.2). For nonexistent rails, the outputs from the slaves will be left unconnected. Furthermore, to keep all packet structures standard, the MasterPC will also send 9 bytes of rail information to all stations, even if that station does not have 9 pairs of rails. Details of this packet can be seen in Figure 3.4a.

By doing so, we are able to have one standard station board and standard packets, with the same Master and Slave PICs on all of them, thereby cutting down design and manufacturing costs, improving expandability and flexibility, and making debugging easier and more straightforward.

**PIC-to-PIC error packet** length is 5 bytes long made of 3 preliminary bytes, 1 error byte, and a CRC byte. It contains the following:

1) 1 byte ‘to’ data, denoting which PIC the packet comes from
2) 1 byte ‘from’ data, denoting which PIC the packet is going to
3) 1 byte ‘length’ data, denoting the length of the packet
4) 1 bytes of “error” data, containing the exact same information as the original packet from the MasterPC
5) 1 byte of ‘CRC’ data, used to check for errors in transmission

Since only SlavePICs are receiving information, only they can send error packets to the MasterPIC. Thus, practically the ‘to’ address is always the ID of the MasterPIC. Once the MasterPIC receives this packet, it will re-send the last packet to the slave that requested it, denoted by the ‘from’ byte. The structure of this packet can be seen in figure 3.4b.
3.1.1b Collecting Sensor Information:

The MasterPIC will constantly be polling the 9 sensor inputs periodically, scanning changes in sensor data. We have decided to use polling rather than generating interrupts, since the latter proved to be more costly and complex as it required complex external circuitry to implement. We have seen that polling every 1msec is more than enough resolution to catch any train passing over a sensor, since our tests have shown that a train can take a minimum of 40msec to pass over a sensor going at the 100% speed. Furthermore, it is computationally easy to do within the MasterPIC due to its high clock speed of 40 MHz.

The polling is generated by using an internal timer of the MasterPIC. When the timer reaches a count denoting a 1msec time-lapse, an interrupt occurs. Within this interrupt service routine, the MasterPIC first outputs a rising edge on the pin which is connected to the 9-bit register as its clock. It waits for 10usec for the register outputs to stabilize, and then looks at the values of all 9 pins connected to de-bounced sensor outputs. It compares the current values to the previous values in memory, and notes which sensors have become active from a non-active state. If at least 1 sensor became active from a non-active state, the MasterPIC compiles a 3-byte packet to be sent to the MasterPC, where the last byte is for CRC, and each bit of the remaining 2 bytes encodes whether a sensor is active or not. Details of this packet can be seen in figure 3.5:

![Sensor Byte Structure](image)

*Figure 3.5 – Sensor Byte Structure (Bits 9-15 are unused)*

It is important to note that we are only sending packets when a train arrives at a sensor, and not when it leaves, since doing the former provides enough information for the MasterPC to update its information fully.

3.1.1c Switch Control:

The MasterPIC will take the switch byte (last byte) of the packet sent from the PC and output low values (the FET switches are active low) on either the Red or the Green coils for all the switches that need to be powered. This operation needs to last until at least the capacitors of the low-level circuitry discharge to give the coils enough energy. This time is given as 5.6msec, and is the width of the low pulse that must be output.

This poses a problem, since 5.6msec is a long time to do nothing but output pulses, as the MasterPIC must still poll sensors every 1msec, and accept further packets from the computer.
Furthermore, if another packet arrives commanding another switch to be powered while we are outputting pulses, then we cannot handle that event, unless we keep it in memory.

To solve this problem, we will use the MasterPIC’s 2 unused 16-bit timers, and assign each timer a task queue. This task will simply be the PC packet’s switch byte. Once a timer counts up to 5.6msec, it will move on to the next task. During this time, all the interrupts will be active so that we can poll, and receive further packets from the PC. Every time a packet arrives with a switch byte other than 0 (at least 1 switch is being powered) the MasterPIC will place it to the first empty queue. If both queues are full, then it will assign it add it to any of the queues. Both queues will be 2 tasks deep, as 4 tasks should be enough to handle all possible scenarios.

### 3.1.2 SlavePIC:

The SlavePIC has 1 main task:

1) Receiving packets with rail speed & polarity and switch information from MasterPIC, and generate three 8Hz PWM signals with correct duty cycles (*Rail Control*)

#### 3.1.2a Rail Control:

Because the maximum number of rail pairs in any given station is 9 and the maximum number of unique PWMs that can be output via a PIC182431 are 3, we had to use 3 SlavePICs. This way, we are able to control the speed and polarity of each individual rail pair at any given time. (*Refer to Appendix-N3,N4,N5 for layout details & pin connections*)

Each SlavePIC will accept 3 bytes of rail information, each byte encoding speed & polarity information for a single rail pair. Upon receiving a packet from the MasterPIC, the SlavePIC will look at the speed bits and polarity for all 3 rail pairs, and the polarity bits, and output a PWM signal with a duty cycle corresponding to the required speed, and the required polarity signals. The duty cycle ranges from 36% to 96% at 4% increments, and the polarity signals can be 0-0, 0-1, 1-0, and 1-1.

Furthermore, to comply with requirement ER002, the duty cycle needs to be incremented slowly until the required duty cycle is achieved. To do this, the SlavePICs will use 3 of their unused timers for each PWM to periodically increment the duty cycle by 1% starting from its current duty cycle until the desired duty cycle is reached. This period should be long enough for a comfortable speed-up, but not so long that it delays scheduling and causes problems. Thus, any change in speed will be handled on an incremental basis, making it comfortable for passengers.
3.2 Networking Test Plan

3.2.1 RS-232 COMMUNICATION

Test Requirement 1: Verify that the correct frame is sent to the PC from a PIC.

Testplan:
- a. Wrote code that sends out a specific frame to HyperTerminal verify that the frame sent was received correctly.
- b. Attach multiple PICs to different ports on the Octopus and verify that they can be accepted and distinguished from one another.

Test Requirement 2: Verify that a frame is correctly received from the PC.

Testplan:
- a. Send out a packet frame via HyperTerminal and write code to send, the received frame back to HyperTerminal. Verify that the correct packet is received by comparing the hexadecimal values with the original.
- b. Connect multiple PICs to different ports on the Octopus and verify that the PC can send to specific PICs by having them send back the packets to different ports on HyperTerminal.

3.2.2 SENSOR POLLING

Test Requirement 1: Verify the correct operation of the debouncing GAL

Testplan:
- a. Connect a GAL to a train sensor and run a train over it.
- b. View the output of the GAL on an oscilloscope to see if the sensor has been debounced.
- c. Verify that the PIC can see this GAL output as TTL logic.

Test Requirement 2: Verify the correct operation of polling subroutines.

Testplan:
- a. Use the lab equipment to generate sensor outputs one sensor at a time.
- b. Run these outputs through the debouncing GAL.
- c. Route the GAL outputs to the sensor input pins on the PIC and verify that the PIC does not miss any sensor pulses even when multiple pulse occur in short succession on separate pins.
3.2.3 Rail Control

Test Requirement 1: Verify that the proper signaling for the power in different sections of the rail is assigned as determined by the byte stream received from the user.

Testplan:
   a. Check to see that the correct PWM’s are created on the correct slave PIC pins when a packet is sent to the MASTER PIC from the PC.
   b. Also check the rail polarity pins are all correctly set.

Test Requirement 2: Verify that the Pulse Width Modulation (PWM) signals to control the speed of the motor is being assigned the correct frequency and duty cycle for one out of the sixteen pre-determined speeds. Do a worst case scenario where all PWM signals need to be turned on upon a single received packet.

Testplan:
   a. Measure the PWM output pins of the PIC via an oscilloscope and verify that the frequency and duty cycle is being assigned properly.
   b. Check that the slave PICs are able to generate 3 genuinely different duty cycles at any given time.

3.2.4 Switch Control

Test Requirement 1: Verify the desired allocation of power to either the Red or Green coil depending on the information sent by the PC.

Testplan:
   a. Check that if a switch is ordered to be turned by the PC, the correct outputs come out of the master PIC.
   b. Check that the values coming out of the GAL are the correct time and value to activate a coil using an oscilloscope.
3.3 Networking Requirements Analysis

<table>
<thead>
<tr>
<th>Requirements</th>
<th>RS-232 Communication</th>
<th>Sensor Polling</th>
<th>Rail Control</th>
<th>Switch Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>R001 rail switch control</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>R002 engine power control</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>R003 train proximity monitoring</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R004 expandability and adaptability</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>R005 control and monitoring speed</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>R006 applications programming interface</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>R007 maintenance user interface</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R008 demonstration application</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R009 modifications of the CFE layout</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>R010 power input</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirements</th>
<th>RS-232 Communication</th>
<th>Sensor Polling</th>
<th>Rail Control</th>
<th>Switch Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPR001 documentation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR002 environmental</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR003 EMI/EMC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR004 hazmats</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR005 safety and good practice</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR006 reliability</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR007 maintainability</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR008 sourcing sustainability</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR009 global sustainability</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR010 ethics report</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR011 project demonstration</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPR012 final disposal of projects</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirements</th>
<th>RS-232 Communication</th>
<th>Sensor Polling</th>
<th>Rail Control</th>
<th>Switch Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER001 trains will not crash</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ER002 train acceleration</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirements</th>
<th>RS-232 Communication</th>
<th>Sensor Polling</th>
<th>Rail Control</th>
<th>Switch Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>RR001 Mean Time Between Failure</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>RR002 System life</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirements</th>
<th>RS-232 Communication</th>
<th>Sensor Polling</th>
<th>Rail Control</th>
<th>Switch Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR001 Mean Time To Repair</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
R001 Rail Switch Control:
The RS-232 communicates between the PC and a master PIC. The master PIC controls the switch control.

R002 Engine Power Control:
The RS-232 communicates between the PC and a master PIC. The master PIC then outputs signals to three slave PICs. Each slave PIC outputs 3 PWM signals, this gives us a total of 9 signals to control 9 rail switches (the maximum number of rail pairs at one station).

R003 Train Proximity Monitoring:
Sensor polling will monitor all the sensors at a particular station. Timing analysis was performed to check for the worst case scenario. The two closest sensors are _______ cm, the train travels at ____ at the fastest speed. This gives ____ seconds for the master PIC to poll all sensors. Based on timing analysis, it will take 35 ms to poll all sensors and thus this requirement will be met. Included in the timing analysis is the time requirement to communicate over the RS-232.

R004 Expandability and Adaptability:
The RS-232 communication will use an octopus so that 8 serial ports will be available, this allows for multiple trains to operate simultaneously. The sensor polling timing analysis also allows for multiple trains operating simultaneously. Rails are labeled to enable more rails to be added easily for expandability. Switches _______ Expandability and Adaptability?!?!!

R005 Control and Monitoring Speed:
Based on the timing analysis, there will not be any adverse delays, contention, or missed events. The timing analysis shows that all sensors will be able to be polled within the allowable time. The critical path for timing analysis allows time for a sensor to be polled, information sent over RS-232 to the PC and then interpreted and sent back to the PIC and still enough time for the PIC will send information to control the rails and switches.

R006 Applications Programming Interface:
Not related to Network group.

R007 Maintenance User Interface:
Not related to Network group.

R008 Demonstration Application:
Not related to Network group.

R009 Modifications of the CFE Layout:
Modifications to the CFE layout will not be necessary, hardware will utilize the CFE layout.
**R010 Power Input:**
Not related to Network group. Power for the network hardware will be created by the low-level group.

**GPR001 Documentation:**
Circuits are documented with schematics.

**GPR002 Environmental:**
All parts meet environmental requirement for temperature range and RH, non-condensing (refer to BOM).

**GPR003 EMI/EMC:**
All parts meet requirement US CFR Title 47 Part 15 subpart B, and there are no intentional radiators.

**GPR004 Hazmats:**
There are no hazardous materials in this design.

**GPR005 Safety and Good Practice:**
Network group follows safety and good practice requirements including color coded wiring, clear labeling, and correct labeling on boards.

**GPR006 Reliability:**
Network group will meet additional reliability requirements.

**GPR007 Maintainability:**
Network group will meet additional maintainability requirements.

**GPR008 Sourcing Sustainability:**
All parts meet sourcing sustainability requirement (refer to BOM).

**GPR009 Global Sustainability:**
Production will consider global sustainability. Further details to be provided in sustainability report.

**GPR010 Ethics Report:**
Network group will meet additional ethical requirements

**GPR011 Project Demonstration:**
All completed projects were or will be demonstrated to for review by the ECE faculty.

**GPR012 Final Disposal of Projects:**
All parts can either be stored for future work, placed on display, or discarded. Project will not include hazardous materials.

**ER001 TRAIN WILL NOT CRASH:**
Software will include a failsafe mode which will be communicated over the RS-232.

**ER002 TRAIN ACCELERATION:**
Train acceleration and deceleration will not cause illness to passengers. Train acceleration and deceleration will be controlled by the master PIC.

**RR001 MEAN TIME BETWEEN FAILURE:**
Testing will be performed to be sure system has an overall mean time between failure of 100 hrs.

**RR002 SYSTEM LIFE:**
The system life will be 5 years. Each part will have a life of at least 5 years.

**MR001 MEAN TIME TO REPAIR:**
The mean time to repair will be less than 1 week.
3.4 Networking Risk Assessment

3.4.1 RS-232 Communication

The communication has been shown to work between one PIC and a single COM port on the computer. There still needs to be confirmation that we can have multiple COM ports through the Octopus part.

<table>
<thead>
<tr>
<th>Methodology Maturity</th>
<th>Methodology Complexity</th>
<th>Hardware/Software Maturity</th>
<th>Hardware/Software Complexity</th>
<th>Dependency of Schedule on External Functional Groups</th>
<th>Dependency of Performance on External Functional Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.1</td>
<td>0.3</td>
<td>0.1</td>
<td>0.5</td>
<td>0</td>
</tr>
</tbody>
</table>

Potential for Failure Factor = \( \frac{0.1 + 0.1 + 0.3 + 0.1 + 0.3 + 0}{6} = 0.15 \)

<table>
<thead>
<tr>
<th>Degradation of Technical/Operational Performance at System Level</th>
<th>Schedule Impact</th>
<th>Cost Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.5</td>
<td>0.3</td>
</tr>
</tbody>
</table>

If the RS-232 does not work we need to find a different method of data transmission. This would mean redesigning logic inside the PIC and possibly needing completely different hardware. It would delay the low level hardware progress as well as the software development.

Consequence for Failure Factor = \( \frac{0 + 0.5 + 0.3}{3} = 0.267 \)

**Risk Factor:**

\[ P+C - (P*C) = 0.15 + 0.267 - (0.15 \times 0.267) = 0.377 \text{ med risk} \]

3.4.2 Sensor Polling

The sensor interrupts have been researched and preliminary code has been written and designed to run these polling routines. No problems have been encountered yet with the design of this subsystem.

<table>
<thead>
<tr>
<th>Methodology Maturity</th>
<th>Methodology Complexity</th>
<th>Hardware/Software Maturity</th>
<th>Hardware/Software Complexity</th>
<th>Dependency of Schedule on External Functional Groups</th>
<th>Dependency of Performance on External Functional Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Potential for Failure Factor = \( \frac{0.3 + 0.3 + 0.3 + 0.3 + 0.3 + 0.5}{6} = 0.333 \)

<table>
<thead>
<tr>
<th>Degradation of Technical/Operational Performance at System Level</th>
<th>Schedule Impact</th>
<th>Cost Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Consequence for Failure Factor = \( \frac{0.3 + 0.3 + 0.3}{3} = 0.3 \)

**Risk Factor:**

\[ P+C - (P*C) = 0.3 + 0.33 - (0.3 \times 0.33) = 0.531 \text{ high risk} \]
3.4.3 Rail Control

Design of code that will send out packets to slave PICs where PWM’s will be created has been designed. There are no problems that have been noticed for the communication between the PICs or the creation of separate PWM signals.

<table>
<thead>
<tr>
<th>Methodology Maturity</th>
<th>Methodology Complexity</th>
<th>Hardware/Software Maturity</th>
<th>Hardware/Software Complexity</th>
<th>Dependency of Schedule on External Functional Groups</th>
<th>Dependency of Performance on External Functional Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>0.1</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Potential for Failure Factor = \( (0.3 + 0.1 + 0.3 + 0.3 + 0.3 + 0.3)/6 = 0.267 \)

<table>
<thead>
<tr>
<th>Degradation of Technical/Operational Performance at System Level</th>
<th>Schedule Impact</th>
<th>Cost Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>0.1</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Consequence for Failure Factor = \( (0.3 + 0.1 + 0.5)/3 = 0.3 \)

Risk Factor:
\[ P+C - (P*C) = 0.3 + 0.1 - (0.3 * 0.1) = 0.37 \text{ medium risk} \]

3.4.4 Switch Control

We have designed code for outputting the correct digital signals for the low level hardware to use. The design has some issues that could arise from timing. The amount of time that we output a rail switching signal must be long enough for the capacitor to discharge. If there are too many requests in a short time the PIC might lose some of this information. Testing will reveal these problems.

<table>
<thead>
<tr>
<th>Methodology Maturity</th>
<th>Methodology Complexity</th>
<th>Hardware/Software Maturity</th>
<th>Hardware/Software Complexity</th>
<th>Dependency of Schedule on External Functional Groups</th>
<th>Dependency of Performance on External Functional Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.3</td>
<td>0.5</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Potential for Failure Factor = \( (0.5 + 0.3 + 0.5 + 0.3 + 0.3 + 0.3)/6 = 0.367 \)

<table>
<thead>
<tr>
<th>Degradation of Technical/Operational Performance at System Level</th>
<th>Schedule Impact</th>
<th>Cost Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>0.3</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Consequence for Failure Factor = \( (0.3 + 0.3 + 0.5)/3 = 0.367 \)

Risk Factor:
\[ P+C - (P*C) = 0.367 + 0.367 - (0.367 * 0.367) = 0.599 \text{ high risk} \]
4. Low-Level Group

4.1 Low Level Sub-System Specifications

The Low Level Subsystem is comprised of five (5) main components. These subsystems will allow:

- Trains to switch tracks at any speed (R001)
- The trains to be powered for various speeds (R002)
- Protection from a potential short between the rails (R002)
- Monitoring of the trains location through the use of the magnetic reed switches on the track (R003)
- Power to be transformed from a 120V AC, 60Hz source into various DC voltages to power both the digital and analog components (R010)

The Sub-Systems that will take care of these tasks are the Rail Switching Circuit (Drawing #14), the Rail Power System (Drawings #12 and #XX), the Sensor Circuit (Drawing #8) and the Power Board (Drawings #2 and #3).
4.1.1 **Rail Switching Circuit (Drawing #14)**

Control of the train’s path is taken care of by this circuit. By charging a large capacitor and then discharging it across the rail switch, an inductor with an internal resistance 4.6 ohms. Depending on the side of the switch (red or green) across which the capacitor is discharged the direction in which the train will travel is toggled. While the capacitor is always connected to the 22V source the charge only flows through to the rail switch when a switch is flipped to close that branch of the circuit. By using this design the switch becomes a slightly less critical part in that if it remains closed for a little too long it will not damage the rail switch (inductor). The resistor and capacitor combination chosen for this circuit allow for each station to switch a rail in a little less than four (4) seconds.

This switch, which is actually a transistor, is toggled between open and closed by a signal from the PIC. The decision was made to use a transistor to prevent the timing and other issues a traditional relay would have presented.

4.1.2 **Rail Power System (Drawings #12 and #XX)**

This system will amplify a Pulse Width Modulated (PWM) signal that is passed into the system by the PIC. The modified PWM signal output has the same duty cycle as the signal input by the PIC, however its amplitude is now 18V, enough to power the trains. Additionally, Light Emitting Diodes (LEDs) have been integrated into this system. One (1) LED has been included for each rail to be powered. These LEDs will allow trouble shooting to go more quickly by providing output showing which rails are and are not working. In the case that debris causes a short between the two rails, the system has been designed in such a way so that no major damage will be sustained.

4.1.3 **Sensor Circuit (Drawing #8)**

This circuit will interface with the reed switch imbedded in the track as well as the PIC to let the user know where the train is. Using a 5V source this circuit has been designed to output TTL voltage levels to the PIC indicating either the presence or absence of the train. When the train passes over a reed switch the switch acts as a short, causing little or no voltage drop across it. When the train is not present the reed switch appears as an open circuit and thus all the voltage drop occurs across it. This voltage drop is what is output to the PIC.

4.1.4 **Power Circuit (Drawings #2 and #3)**

This system will take the 120V AC, 60Hz source and transform it. First it will be changed into a 24V DC / 1A signal. Next this DC signal will create four (4) separate voltage sources: an 18V DC source, a 22V DC source, and two (2) 5V DC sources. The 22V source will be used to power the Rail Switching Circuit. The 18V source will be used by the Rail Power System. The 5V will be split and used for different applications. The first will power the Rail Switching Circuit and the Cooling Fan. The other 5V source will be used to power the PIC chips at the station.
4.2 Low Level Test Plan

4.2.1 Speed Control (H-Bridge) Testing

The speed control component of the track was tested in two ways.

Test One: Circuit tested without integration.
1. Using DC power supplies 18V were input to the Vss and Vss1 pins and 5V was input on input 1.
2. Using a function generator a 5V PWM signals was input on the shared enable 1 and 2 pin.
3. An oscilloscope was hooked up to the two output 1 and 2, and output was measured across the pins. Output = 18V PWM => desired output.

Test Two: Circuit tested with integration.
1. H-Bridge hooked up to PIC, with both 5 volt PIC output tied to input one and the other tied to input 2 respectively. The PWM signal from the PIC connected to the shared enable 1, 2 pin on the H-Bridge.
2. The 18V supply came from a voltage regulator.
3. Output pins 1 and 2 connected to oscilloscope. Output = 18 volt PWM => desired output.
4. Output pins connected to rails with train. Result = proper movement of train with different speeds based on PWM signal from PIC, and direction based on whether output from PIC was high on input pin 1 on H-Bridge or input pin 2 on H-Bridge. Test passed, train operated as expected.

Inputs: 18V DC, 5V DC, 5V PWM
Outputs: PWM signal
Components: H-Bridge

4.2.2 Power Circuit Testing

Four tests were conducted on the topology shown below. The purpose of the topology was to give a clean programmed voltage on the output. Tests one through three are individual circuit tests, and test four is an integrated test with all regulator circuits coming off of one 24 V input.
Test One:
1. Vin = 24 Volts, R1 = 1K ohm, R2 = 13K ohm.
2. Desired output = 18V => test passed.

Test Two:
1. Vin = 24 Volts, R1 = 1K ohm, R2 = 1.6K ohm.
2. Desired output = 5V => test passed.

Test Three:
1. Vin = 24 Volts, R1 = 1K ohm, R2 = 16K ohm.
2. Desired output = 22V => test passed.

Test Four:
1. Vin = 24 Volts, input feeds to all inputs of previous three circuits (same node)
2. Output => all outputs matched same as previous three circuits => test passed
4.2.3 **Switching Circuit Testing**

Testing of the switching circuit was conducted both in hardware and simulated in pspice. Testing in hardware was carried out without the transistor switch as the part is not yet available. The circuit was able to move the switches on the train tracks depending on which wire (red or green) the capacitor was connected to. The circuit was also simulated in software using a different transistor in the simulation as the spice deck for the NTE196 which we are using wasn’t available. Below is a diagram of the simulation results.

![Diagram of simulation results](image)

The capacitor takes five (5) seconds to completely charge while the discharge is nearly instantaneous, as it is discharged over a small resistance of the switch coil of 4.6ohm. Since these tests have proved successful, the next step is to get the actual hardware and test them individually. The next phase of testing would be to test them integrated with one another, then test them after integrating it with the signals coming out of the pic from the networking group.
4.2.4 **SENSOR CIRCUIT TESTING**

Testing of the sensor circuits took place with the use of a bread board / prototyping board. The circuit was connected to the sensors on the track, one at a time, and verified that the voltage output was the desired logic level to be transmitted to the PIC. Another concern that has been examined through testing is the bounce of the reed switch. We had to verify that the bouncing of the switch was not too great for the PIC to handle.

Test One: Circuit tested without integration.
1. Using DC 5V the sensor circuit was powered.
2. Manually operating the trains, they were passed over the sensor under test
3. An oscilloscope was hooked up to the output and the voltage output was examined to ensure proper operation.

Test Two: Circuit tested with integration.
1. Using DC 5V the sensor circuit was powered.
2. Manually operating the trains, they were passed over the sensor under test
3. The output was connected to the PIC and then examined on the oscilloscope to verify that the bouncing of the switch was within tolerable limits
### 4.3 Low Level Requirements Analysis

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Power Conversion</th>
<th>Direction/Speed</th>
<th>Rail Switching</th>
<th>Sensors/Proximity</th>
</tr>
</thead>
<tbody>
<tr>
<td>R001</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R002</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>R003</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R004</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>R005</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>R006</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R007</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R008</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>R009</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>R010</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Power Conversion</th>
<th>Direction/Speed</th>
<th>Rail Switching</th>
<th>Sensors/Proximity</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPR001</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>GPR002</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>GPR003</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>GPR004</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>GPR005</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>GPR006</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>GPR007</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>GPR008</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>GPR009</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>GPR010</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>GPR011</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>GPR012</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Power Conversion</th>
<th>Direction/Speed</th>
<th>Rail Switching</th>
<th>Sensors/Proximity</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER001</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>ER002</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
</tbody>
</table>

**R001 – Rail Switch Control**
The rail switching circuit will take the power that it needs from a charged capacitor. The charging and discharging of the capacitor is controlled by transistors switches. The transistors get their signals from the PIC.

**R002 – Engine Power Control**
The direction/speed or rails circuit will receive their signal from the H-Bridge which will output a PWM signal for speed control and will only power one rail out of a pair for direction. PWM and polarity is determined by the network and interpreted by the PIC which gives the signal to the H-Bridge. There is also a safety circuit in place that will protect against a fault current of a short circuit across the rails.

**R003 – TRAIN PROXIMITY MONITORING**
The proximity circuit will provide feedback of train location and allow independent programmed monitoring of all train proximity sensors. The signal given from the sensor will feed to the PIC for interpretation and messaging the network.

**R004 – EXPANDABILITY AND ADAPTABILITY**
All circuits will interface directly with the train and supports expandability and adaptability. If the project were to expand it would only be necessary to increase the amount of low level circuits and now new designs would be needed.

**R009 – MODIFICATIONS OF THE CFE LAYOUT**
The train layout will not be modified as well as the tracks, track-switches, train sensors and trams themselves. The wiring connecting directly to the switches and sensors will also not be modified but the busses will be jumpered and augmented with additional hardware.

**R010 – POWER INPUT**
The power conversion circuit will take the power directly from the wall (120V AC, 60Hz), and convert that power to any voltage needed in the system with multiple outputs, meeting this requirement. The power board outputs 3 different voltages using regulators. It outputs 18V, 22V, and 5V sources.

**GPR001: Documentation**
All circuits are well documented with schematics. All schematics have a system block diagram, subsystem specifications, principals of operations, system acceptance, and subsystem test reports. A users manual is also present for all parts.

**GPR002: Environmental**
All circuits demonstrate reliable and normal function in ambient temperatures. All circuits also tolerate a storage environment of 0 to 60 degrees C and 5% to 95% RH.

**GPR003: EMI/EMC**
All circuits meet requirement US CFR Title 47 Part 15 subpart B, and there are no intentional radiators.

**GPR004: Hazmats**
There are no hazardous materials in any of the circuit designs.

**GPR005: Safety and Good Practice**
All circuits will have reference designators, color coded wiring, clear labeling of of all controls and indicators, socket fuses, and access panels on enclosures.

**GPR006: Reliability**
All circuits are not run off of any clocks, so mean time between failure only depends on parts not breaking. All parts are protected from short circuits and will not blow making the low-level systems very reliable. Parts are also easy to replace.

**GPR007: Maintainability**
All systems show at least a system wide MTTR less than 1 week over the system lifetime.

**GPR008: Sourcing Sustainability**
All circuits are made with parts that can be supplied by multiple vendors.

**GPR010: Ethics Report**
All circuits are taken into account when thinking about ethical principles. This is discussed in further detail in the ethics report.

**GPR011: Project Demonstration**
All completed projects were or will be demonstrated to for review by a member of the ECE faculty.

**GPR012: Final Disposal of Projects**
All circuits are printed on one board and will be easily stored or put on display. If discarded all circuits contain no hazardous and agree with the hazmat requirement and can be discarded in a safe manor.

**ER001: Trains Will Not Crash**
The speed/direction, switching, and train proximity circuits will respond to any inputs from the network. As long as the user uses the system in a safe fashion, or the system is running in demonstration mode the trains will not crash.

**ER002: Train Acceleration**
The speed and direction circuit will give the desired speed to the train. The natural weight of the train and variable resistance in the motor allows for a gradual increase in speed and decrease in speed, given one desired speed.
4.4 Low Level Risk Assessment

4.4.1 Overall System Power:

This module is responsible for supplying power to all components of the system, including the computers, the PICs, and the logic chips. It will be necessary to convert AC power from the wall to the appropriate DC voltage levels for each component requiring power.

<table>
<thead>
<tr>
<th>Methodology Maturity</th>
<th>Methodology Complexity</th>
<th>Hardware Maturity</th>
<th>Hardware Complexity</th>
<th>Dependency of Schedule on External Functional Groups</th>
<th>Dependency of Performance on External Functional Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Degradation of Technical/Operational Performance at System Level

<table>
<thead>
<tr>
<th>Schedule Impact</th>
<th>Cost Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.1</td>
</tr>
</tbody>
</table>

\[
P(F) = (0.1 + 0.1 + 0.1 + 0.1 + 0 + 0) / 6 = 0.0667
\]

\[
C(F) = (0.9 + 0 + 0.1) / 3 = 0.333
\]

Risk Factor = 0.0667 + 0.333 – (0.0667 * 0.333) = 0.3778 (Medium Risk)

4.4.2 Rail Power:

Rail power will be controlled by varying the duty cycle of a PWM signal coming from the PIC. This module will be responsible for amplifying the PWM signal and transferring it to the rails to control the speed and direction of the train.

<table>
<thead>
<tr>
<th>Methodology Maturity</th>
<th>Methodology Complexity</th>
<th>Hardware Maturity</th>
<th>Hardware Complexity</th>
<th>Dependency of Schedule on External Functional Groups</th>
<th>Dependency of Performance on External Functional Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Degradation of Technical/Operational Performance at System Level

<table>
<thead>
<tr>
<th>Schedule Impact</th>
<th>Cost Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.1</td>
</tr>
</tbody>
</table>

\[
P(F) = (0.1 + 0.1 + 0.1 + 0.1 + 0 + 0) / 6 = 0.0667
\]

\[
C(F) = (0.9 + 0 + 0.1) / 3 = 0.333
\]

Risk Factor = 0.0667 + 0.333 – (0.0667 * 0.333) = 0.3778 (Medium Risk)
4.4.3 Rail Switching:

This module is used to control the path that the train will take at each intersection of rails. A voltage supplied to a coil will change the position of the rails at each junction.

<table>
<thead>
<tr>
<th>Methodology Maturity</th>
<th>Methodology Complexity</th>
<th>Hardware Maturity</th>
<th>Hardware Complexity</th>
<th>Dependency of Schedule on External Functional Groups</th>
<th>Dependency of Performance on External Functional Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Degradation of Technical/Operational Performance at System Level</th>
<th>Schedule Impact</th>
<th>Cost Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9</td>
<td>0</td>
<td>0.1</td>
</tr>
</tbody>
</table>

P(F) = (0.1 + 0.1 + 0.1 + 0.1 + 0 + 0) / 6 = 0.0667  
C(F) = (0.9 + 0 = 0.1) / 3 = 0.333

Risk Factor = 0.0667 + 0.333 – (0.0667 * 0.333) = 0.377 (Medium Risk)

4.4.4 Sensors:

The sensors are used to determine the position of the trains on the track. When the switch is activated, a short circuit exists across the terminals. Otherwise, an open circuit exists across the terminals. This module will be responsible for outputting a digital signal indicating whether or not a sensor is activated.

<table>
<thead>
<tr>
<th>Methodology Maturity</th>
<th>Methodology Complexity</th>
<th>Hardware Maturity</th>
<th>Hardware Complexity</th>
<th>Dependency of Schedule on External Functional Groups</th>
<th>Dependency of Performance on External Functional Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Degradation of Technical/Operational Performance at System Level</th>
<th>Schedule Impact</th>
<th>Cost Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9</td>
<td>0</td>
<td>0.1</td>
</tr>
</tbody>
</table>

P(F) = (0.1 + 0.1 + 0.1 + 0.1 + 0 + 0) / 6 = 0.0667  
C(F) = (0.9 + 0 +0.1) / 3 = 0.333

Risk Factor = 0.0667 + 0.3667 – (0.0667 * 0.3667) = 0.398 (Medium Risk)
5. **Management & Work Breakdown Structure**

Please see the schedule printout.
6. Appendices