Software Quality Assurance Audit Report:

Date: April 6, 2008

Tested by: Shrijan Roy Rajkarnikar and Austin Robison

Extended Results of the Software Audit Report is attached in a separate sheet.

\[ P = \text{Pass} \quad F = \text{Fail} \]

<table>
<thead>
<tr>
<th>Test</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Station Class</strong></td>
<td></td>
</tr>
<tr>
<td>Test the constructor by creating a new Station.</td>
<td>P</td>
</tr>
<tr>
<td>Add a new rail to a Station and test for the size of rails associated with the particular station to be incremented by 1.</td>
<td>P</td>
</tr>
<tr>
<td>Add a new switch to a Station and test for the size of the switches associated with the particular station to be incremented by 1.</td>
<td>P</td>
</tr>
<tr>
<td>Add a new sensor to a Station and test for the size of the sensors associated with the particular station to be incremented by 1.</td>
<td>P</td>
</tr>
<tr>
<td>Assign a particular value to the rail number for a given rail in the Station and check to see the correct value returned by the getter method.</td>
<td>P</td>
</tr>
<tr>
<td>Assign STRAIGHT switch and TURN switch to two switch values and check to see that the correct switch values are assigned.</td>
<td>P</td>
</tr>
<tr>
<td><strong>Rail Class</strong></td>
<td></td>
</tr>
<tr>
<td>Test the constructor by creating a new Rail.</td>
<td>P</td>
</tr>
<tr>
<td>Assign the LEFT and RIGHT rails neighboring a given rail and check to see the correct values returned by the getter methods.</td>
<td>P</td>
</tr>
<tr>
<td>Associate the given rail with a particular station and check to see that the getter method returns the assigned station our given rail.</td>
<td>P</td>
</tr>
<tr>
<td>Assign a specific number to a rail and check to see that the getter method returns the assigned rail number.</td>
<td>P</td>
</tr>
<tr>
<td>Assign a specific row to a rail and check to see that the getter method returns the assigned row number.</td>
<td>P</td>
</tr>
<tr>
<td><strong>Switch Class</strong></td>
<td></td>
</tr>
<tr>
<td>Test the constructor by creating a new Rail.</td>
<td>P</td>
</tr>
<tr>
<td>Assign a specific polarity to the switch and check the getter method to see the expected value.</td>
<td>P</td>
</tr>
<tr>
<td>Connect a new rail to the left of the given switch and set a different rail to the right of the given switch and check the getter method to see the values being assigned correctly.</td>
<td>P</td>
</tr>
<tr>
<td>Assign a specific position MIDDLE, LEFT or RIGHT to the rails connected to the left and right of the given switch and check the respective getter methods to see the values being assigned correctly.</td>
<td>P</td>
</tr>
<tr>
<td>Assign a specific station to a given switch and check the getter method to see the station being assigned correctly.</td>
<td>P</td>
</tr>
<tr>
<td>Assign a specific number to a given switch and check the getter method to see the switch number being assigned correctly.</td>
<td>P</td>
</tr>
</tbody>
</table>
Assign the specific direction LEFT or RIGHT that the train is moving when it encounters the particular switch and check the getter method to see that it is being assigned correctly.

### Sensor Class
- Test the constructor by creating a new Sensor.
- Assign the given sensor to a given rail which is in the MIDDLE, LEFT or RIGHT and check to see the getter method that it is being assigned correctly.
- Assign the given sensor to a particular rail and check to see the getter method that it is being assigned correctly.
- Assign a particular sensor number to the given rail and check to see the getter method that it is being assigned correctly.

### ControllerMain Class
- Test whether the maximum number of rails being read from the XML file is equal to the expected value of 8.
- Test whether the maximum number of switches being read from the XML file is equal to the expected value of 4.
- Test for sensor events by assigning Boolean into a Boolean array and checking for the assigned sensor values for particular sensors.
- Test for the number of stations being read in from the XML file with the expected value of 5.
- Test for the first station being read from the XML file, and check for the station name being “Station 1 - 3rd Street Terminal”, the number of rails equal to 7, the number of switches equal to 2, the number of sensors equal to 6 and the station number equal to 1.
- Test for the first station being read from the XML file, and check for the station name being “Station 2 – Sullivan Village Terminal”, the number of rails equal to 6, the number of switches equal to 0, the number of sensors equal to 9 and the station number equal to 2.
- Test for the first station being read from the XML file, and check for the station name being “Station 4 - Farinon Terminal”, the number of rails equal to 6, the number of switches equal to 3, the number of sensors equal to 8 and the station number equal to 4.
- Test for the first station being read from the XML file, and check for the station name being “Station 5 - Metzgar Terminal”, the number of rails equal to 7, the number of switches equal to 2, the number of sensors equal to 6 and the station number equal to 5.

### BuilderDecoder Class
- Test the constructor by creating a new BuilderDecoder
- Test that the packet sent from the PC is correctly transmitted via all the RS-232 Octopus ports by checking the five RS-232 outputs in the oscilloscope. The value printed on the PC must match the value observed in the oscilloscope.
- Test for the retransmission of the correct packet after the BuilderDecoder receives an error byte
- Test that the BuilderDecoder can decode a packet coming in from the one of
the RS-232 Octopus ports and correctly calculate the CRC byte for the incoming packet.

<table>
<thead>
<tr>
<th>MaintenanceInterface Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test for the correct display of the track layout on the screen including changing the rail and switch values.</td>
</tr>
<tr>
<td>Test for the correct packets being built by the BuilderDecoder class when the various settings in the Maintenance GUI are changed.</td>
</tr>
</tbody>
</table>
Quality Assurance Audit Report for Board #: 1

Date: April 11, 08

Tested by: Serdar, Tom, Dan, Jeff

<table>
<thead>
<tr>
<th>Test</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Circuits</td>
<td></td>
</tr>
<tr>
<td>Tested the output of LM 317 to receive 5V.</td>
<td>F</td>
</tr>
<tr>
<td>Tested the output of LM 317 to receive 22V.</td>
<td>F</td>
</tr>
<tr>
<td>Tested the output of L200 to receive 18 V.</td>
<td>F</td>
</tr>
<tr>
<td>Tested 18V source with current limiting by hooking up the H-Bridge to the regulator along with the outputs of the H-Bridge hooked up with the rails. First tested normal operation, then shorted the rails. After shorting the rails we again tested normal operation with the H-Bridge to make sure the train moved in both directions.</td>
<td>F</td>
</tr>
</tbody>
</table>

| Sensor Circuits                   |           |
| Hooked up the sensor output connector pins of the board to the outputs of the sensor on the tracks. Plugged in 24V AC/DC adaptor to input port on board. Did not put any train next to sensor and measured output voltage at each input to flip-flop that will record if a train passed a sensor. Each voltage read 0, which was what was expected. | F         |
| Same set up as the first sensor circuit test except that the train was put over sensor and measured outputs on each port again. Measured 5V which was as expected. | F         |

| Switch Circuits                   |           |
| Hooked up the three outputs of each switch to the three outputs on the board respectively. Hooked up the 24V AC/DC converter to the board. Hooked up a 5 volt supply with pin to touch the gates of the transistors to model the pulse from the PIC to turn the switch. To test put switch in one direction and touched the gate of the appropriate transistor to flip the switch. Also tested in other direction with other transistor. Did this four times to test that all outputs operated properly. | F         |
| Used the same set up as above but hooked up communication with hyperterminal. In this set up we did not need the 5 volt power supply because we had the pulse from the PIC. Tested same operations | F         |

| Rail Circuits                     |           |
| Hooked up H-Bridge with all appropriate connections, and hooked up output pins to rails. Wanted normal operation of trains on rails in both directions. | F         |
| Same set up as test 1 except hooked up digital voltage input of H-Bridge to 5V regulator instead of 18V regulator and wanted normal operation of trains in both direction. | F         |
| Same set up as 2 with varying input PWM signals. Expected normal operation with 40% duty cycle to 100% duty cycle. | F         |
| Short circuit scenario tested previously. | F         |
### Quality Assurance Audit Report for Board #2

**Date:** April 11, 08  
**Tested by:** Serdar, Tom, Dan, Jeff

<table>
<thead>
<tr>
<th>Test</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Power Circuits</strong></td>
<td></td>
</tr>
<tr>
<td>Tested the output of LM 317 to receive 5V.</td>
<td></td>
</tr>
<tr>
<td>Tested the output of LM 317 to receive 22V.</td>
<td></td>
</tr>
<tr>
<td>Tested the output of L200 to receive 18 V.</td>
<td></td>
</tr>
<tr>
<td>Tested 18V source with current limiting by hooking up the H-Bridge to the regulator along with the outputs of the H-Bridge hooked up with the rails. First tested normal operation, then shorted the rails. After shorting the rails we again tested normal operation with the H-Bridge to make sure the train moved in both directions.</td>
<td></td>
</tr>
<tr>
<td><strong>Sensor Circuits</strong></td>
<td></td>
</tr>
<tr>
<td>Hooked up the sensor output connector pins of the board to the outputs of the sensor on the tracks. Plugged in 24V AC/DC adaptor to input port on board. Did not put any train next to sensor and measured output voltage at each input to flip-flop that will record if a train passed a sensor. Each voltage read 0, which was what was expected.</td>
<td></td>
</tr>
<tr>
<td>Same set up as the first sensor circuit test except that the train was put over sensor and measured outputs on each port again. Measured 5V which what was expected.</td>
<td></td>
</tr>
<tr>
<td><strong>Switch Circuits</strong></td>
<td></td>
</tr>
<tr>
<td>Hooked up the three outputs of each switch to the three outputs on the board respectively. Hooked up the 24V AC/DC converter to the board. Hooked up a 5 volt supply with pin to touch the gates of the transistors to model the pulse from the PIC to turn the switch. To test put switch in one direction and touched the gate of the appropriate transistor to flip the switch. Also tested in other direction with other transistor. Did this four times to test that all outputs operated properly.</td>
<td></td>
</tr>
<tr>
<td>Used the same set up as above but hooked up communication with hyper-terminal. In this set up we did not need the 5 volt power supply because we had the pulse from the PIC. Tested same operations</td>
<td></td>
</tr>
<tr>
<td><strong>Rail Circuits</strong></td>
<td></td>
</tr>
<tr>
<td>Hooked up H-Bridge with all appropriate connections, and hooked up output pins to rails. Wanted normal operation of trains on rails in both directions.</td>
<td></td>
</tr>
<tr>
<td>Same set up as test 1 except hooked up digital voltage input of H-Bridge to 5V regulator instead of 18V regulator and wanted normal operation of trains in both direction.</td>
<td></td>
</tr>
<tr>
<td>Same set up as 2 with varying input PWM signals. Expected normal operation with 40% duty cycle to 100% duty cycle.</td>
<td></td>
</tr>
<tr>
<td>Short circuit scenario tested previously.</td>
<td></td>
</tr>
</tbody>
</table>
Quality Assurance Audit Report for Board #: 3

Date: April 11, 08
Tested by: serdar, Tom, Dan, Jeff

<table>
<thead>
<tr>
<th>Test</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Circuits</td>
<td></td>
</tr>
<tr>
<td>Tested the output of LM 317 to receive 5V.</td>
<td>P</td>
</tr>
<tr>
<td>Tested the output of LM 317 to receive 22V.</td>
<td>P</td>
</tr>
<tr>
<td>Tested the output of L200 to receive 18 V.</td>
<td>P</td>
</tr>
<tr>
<td>Tested 18V source with current limiting by hooking up the H-Bridge to the regulator along with the outputs of the H-Bridge hooked up with the rails. First tested normal operation, then shorted the rails. After shorting the rails we again tested normal operation with the H-Bridge to make sure the train moved in both directions.</td>
<td>P</td>
</tr>
<tr>
<td>Sensor Circuits</td>
<td></td>
</tr>
<tr>
<td>Hooked up the sensor output connector pins of the board to the outputs of the sensor on the tracks. Plugged in 24V AC/DC adaptor to input port on board. Did not put any train next to sensor and measured output voltage at each input to flip-flop that will record if a train passed a sensor. Each voltage read 0, which was what was expected.</td>
<td>P</td>
</tr>
<tr>
<td>Same set up as the first sensor circuit test except that the train was put over sensor and measured outputs on each port again. Measured 5V which what was expected.</td>
<td>P</td>
</tr>
<tr>
<td>Switch Circuits</td>
<td></td>
</tr>
<tr>
<td>Hooked up the three outputs of each switch to the three outputs on the board respectively. Hooked up the 24V AC/DC converter to the board. Hooked up a 5 volt supply with pin to touch the gates of the transistors to model the pulse from the PIC to turn the switch. To test put switch in one direction and touched the gate of the appropriate transistor to flip the switch. Also tested in other direction with other transistor. Did this four times to test that all outputs operated properly.</td>
<td>P</td>
</tr>
<tr>
<td>Used the same set up as above but hooked up communication with hyper-terminal. In this set up we did not need the 5 volt power supply because we had the pulse from the PIC. Tested same operations</td>
<td>P</td>
</tr>
<tr>
<td>Rail Circuits</td>
<td></td>
</tr>
<tr>
<td>Hooked up H-Bridge with all appropriate connections, and hooked up output pins to rails. Wanted normal operation of trains on rails in both directions.</td>
<td>F</td>
</tr>
<tr>
<td>Same set up as test 1 except hooked up digital voltage input of H-Bridge to 5V regulator instead of 18V regulator and wanted normal operation of trains in both direction.</td>
<td>P</td>
</tr>
<tr>
<td>Same set up as 2 with varying input PWM signals. Expected normal operation with 40% duty cycle to 100% duty cycle.</td>
<td>P</td>
</tr>
<tr>
<td>Short circuit scenario tested previously.</td>
<td>P</td>
</tr>
</tbody>
</table>
Microcontroller Quality Assurance Audit Report for Board #:

Date: 4/11/08

Tested by: [Signature]

<table>
<thead>
<tr>
<th>Test</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sensor Polling Subsystem</strong></td>
<td></td>
</tr>
<tr>
<td>Connect the board to the PC via RS-232 and to the reed sensor wires on the train tracks. Open a HyperTerminal window on the PC.</td>
<td>N/A</td>
</tr>
<tr>
<td>Run trains over the reed switches manually and verify there is data output on the screen.</td>
<td>P</td>
</tr>
<tr>
<td>Verify the correct byte is being transmitted for each switch being tripped.</td>
<td>P</td>
</tr>
<tr>
<td>Run multiple trains over sensors simultaneously and verify that both trains are detected.</td>
<td>P</td>
</tr>
<tr>
<td>Attach a function generator to a rail pair and apply 22 volts. Run a train down the track and verify that it is seen.</td>
<td>P</td>
</tr>
<tr>
<td><strong>Rail Control</strong></td>
<td></td>
</tr>
<tr>
<td>Attach a board to a PC via RS-232 and an oscilloscope to a rail pair. Connect the appropriate outputs of the slave PICs to rail pair wires.</td>
<td>N/A</td>
</tr>
<tr>
<td>Send a packet to the PIC to have it set the rail to a specific setting. Check the oscilloscope to verify that the correct PWM is created.</td>
<td>P</td>
</tr>
<tr>
<td>Apply the above technique to multiple rail pairs and verify that rails can be set to different PWM’s independently.</td>
<td>P</td>
</tr>
<tr>
<td>Verify that the PWMs change gradually.</td>
<td>P</td>
</tr>
<tr>
<td><strong>Switch Control</strong></td>
<td></td>
</tr>
<tr>
<td>Attach board to the switch wires on the train tracks and the board to the PC via RS-232.</td>
<td>N/A</td>
</tr>
<tr>
<td>Send a switching packet and verify the switch locks to the other position. Roll a train across to verify it is adequately switched.</td>
<td>P</td>
</tr>
<tr>
<td>Send a packet to switch it back and roll the train across again.</td>
<td>P</td>
</tr>
<tr>
<td><strong>RS-232 Communication Subsystem</strong></td>
<td></td>
</tr>
<tr>
<td>Open HyperTerminal on the PC and connect a RS-232 cable</td>
<td>N/A</td>
</tr>
<tr>
<td>Connect all appropriate power cables and an RS-232 cable to the system board and verify that the board is powered.</td>
<td>P</td>
</tr>
<tr>
<td>Send a packet via HyperTerminal and verify there is a response</td>
<td>P</td>
</tr>
<tr>
<td>Connect a second board to a different RS-232 port on the PC and open a second HyperTerminal for this port.</td>
<td>N/A</td>
</tr>
<tr>
<td>Send a packet to the second board and verify that the ports can be accessed individually.</td>
<td>P</td>
</tr>
</tbody>
</table>
## Microcontroller Quality Assurance Audit Report for Board #2

**Date:** 4/11/08  
**Tested by:** serdar + Enzi

<table>
<thead>
<tr>
<th>Test</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sensor Polling Subsystem</strong></td>
<td></td>
</tr>
<tr>
<td>Connect the board to the PC via RS-232 and to the reed sensor wires on the train tracks. Open a HyperTerminal window on the PC.</td>
<td>N/A</td>
</tr>
<tr>
<td>Run trains over the reed switches manually and verify there is data output on the screen.</td>
<td>P</td>
</tr>
<tr>
<td>Verify the correct byte is being transmitted for each switch being tripped.</td>
<td>P</td>
</tr>
<tr>
<td>Run multiple trains over sensors simultaneously and verify that both trains are detected.</td>
<td>P</td>
</tr>
<tr>
<td>Attach a function generator to a rail pair and apply 22 volts. Run a train down the track and verify that it is seen.</td>
<td>P</td>
</tr>
<tr>
<td><strong>Rail Control</strong></td>
<td></td>
</tr>
<tr>
<td>Attach a board to a PC via RS-232 and an oscilloscope to a rail pair. Connect the appropriate outputs of the slave PICs to rail pair wires.</td>
<td>N/A</td>
</tr>
<tr>
<td>Send a packet to the PIC to have it set the rail to a specific setting. Check the oscilloscope to verify that the correct PWM is created.</td>
<td>P</td>
</tr>
<tr>
<td>Apply the above technique to multiple rail pairs and verify that rails can be set to different PWM’s independently.</td>
<td>P</td>
</tr>
<tr>
<td>Verify that the PWMs change gradually.</td>
<td>P</td>
</tr>
<tr>
<td><strong>Switch Control</strong></td>
<td></td>
</tr>
<tr>
<td>Attach board to the switch wires on the train tracks and the board to the PC via RS-232.</td>
<td>N/A</td>
</tr>
<tr>
<td>Send a switching packet and verify the switch locks to the other position. Roll a train across to verify it is adequately switched.</td>
<td>P</td>
</tr>
<tr>
<td>Send a packet to switch it back and roll the train across again.</td>
<td>P</td>
</tr>
<tr>
<td><strong>RS-232 Communication Subsystem</strong></td>
<td></td>
</tr>
<tr>
<td>Open HyperTerminal on the PC and connect a RS-232 cable</td>
<td>N/A</td>
</tr>
<tr>
<td>Connect all appropriate power cables and an RS-232 cable to the system board and verify that the board is powered.</td>
<td>P</td>
</tr>
<tr>
<td>Send a packet via HyperTerminal and verify there is a response</td>
<td>P</td>
</tr>
<tr>
<td>Connect a second board to a different RS-232 port on the PC and open a second HyperTerminal for this port.</td>
<td>N/A</td>
</tr>
<tr>
<td>Send a packet to the second board and verify that the ports can be accessed individually.</td>
<td>P</td>
</tr>
</tbody>
</table>
# Microcontroller Quality Assurance Audit Report for Board # 3

**Date:** 4/11/08

**Tested by:**

<table>
<thead>
<tr>
<th>Test</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor Polling Subsystem</td>
<td>N/A</td>
</tr>
<tr>
<td>Connect the board to the PC via RS-232 and to the reed sensor wires on the train tracks. Open a HyperTerminal window on the PC.</td>
<td>N/A</td>
</tr>
<tr>
<td>Run trains over the reed switches manually and verify there is data output on the screen.</td>
<td>Passed</td>
</tr>
<tr>
<td>Verify the correct byte is being transmitted for each switch being tripped.</td>
<td>P</td>
</tr>
<tr>
<td>Run multiple trains over sensors simultaneously and verify that that both trains are detected.</td>
<td>P</td>
</tr>
<tr>
<td>Attach a function generator to a rail pair and apply 22 volts. Run a train down the track and verify that it is seen.</td>
<td>P</td>
</tr>
<tr>
<td>Rail Control</td>
<td>P</td>
</tr>
<tr>
<td>Attach a board to a PC via RS-232 and an oscilloscope to a rail pair. Connect the appropriate outputs of the slave PICs to rail pair wires.</td>
<td>N/A</td>
</tr>
<tr>
<td>Send a packet to the PIC to have it set the rail to a specific setting. Check the oscilloscope to verify that the correct PWM is created.</td>
<td>P</td>
</tr>
<tr>
<td>Apply the above technique to multiple rail pairs and verify that rails can be set to different PWM’s independently.</td>
<td>P</td>
</tr>
<tr>
<td>Verify that the PWMs change gradually.</td>
<td>P</td>
</tr>
<tr>
<td>Switch Control</td>
<td>P</td>
</tr>
<tr>
<td>Attach board to the switch wires on the train tracks and the board to the PC via RS-232.</td>
<td>N/A</td>
</tr>
<tr>
<td>Send a switching packet and verify the switch locks to the other position. Roll a train across to verify it is adequately switched.</td>
<td>P</td>
</tr>
<tr>
<td>Send a packet to switch it back and roll the train across again.</td>
<td>P</td>
</tr>
<tr>
<td>RS-232 Communication Subsystem</td>
<td>P</td>
</tr>
<tr>
<td>Open HyperTerminal on the PC and connect a RS-232 cable</td>
<td>N/A</td>
</tr>
<tr>
<td>Connect all appropriate power cables and an RS-232 cable to the system board and verify that the board is powered.</td>
<td>P</td>
</tr>
<tr>
<td>Send a packet via HyperTerminal and verify there is a response</td>
<td>P</td>
</tr>
<tr>
<td>Connect a second board to a different RS-232 port on the PC and open a second HyperTerminal for this port.</td>
<td>N/A</td>
</tr>
<tr>
<td>Send a packet to the second board and verify that the ports can be accessed individually.</td>
<td>P</td>
</tr>
</tbody>
</table>
LART CS08 Software Test Plan
Austin Robison

1. **ControllerMain Test**
The purpose of this test is to evaluate the ControllerMain class in the API. The main function of the ControllerMain is to parse the XML layout file and populate the data structures – this test makes sure that this functionality is correct.

**Steps:**
- b. Start Eclipse – make sure the code is within the workspace directory somewhere.
- c. Open the file ControllerMainTest.java.
- d. Click on the “run” button.

**Criteria to Pass:**
The ControllerMainTest runs three test methods – if all three pass then the test is successful. Once the test is finished, it will display the number of failures and errors. If both of these are zero, the test passes.

Tested by:  

Date:  4/6/08  

Software Version:  4/6/08 backup  

Pass / Fail:  Pass

2. **Rail Test**
The purpose of this test is to evaluate the Rail class in the API. The main function of the Rail class is to hold all the information about a rail pair in the system – this test makes sure that this functionality is correct.

**Steps:**
- b. Start Eclipse – make sure the code is within the workspace directory somewhere.
- c. Open the file RailTest.java.
- d. Click on the “run” button.

**Criteria to Pass:**
The RailTest runs six test methods – if all six pass then the test is successful. Once the test is finished, it will display the number of failures and errors. If both of these are zero, the test passes.

Tested by:  

Date:  

Software Version:  

Pass / Fail:  

3. **Sensor Test**

The purpose of this test is to evaluate the Sensor class in the API. The main function of the Sensor class is to hold all the information about a sensor in the system – this test makes sure that this functionality is correct.

**Steps:**

a. Boot Computer.
b. Start Eclipse – make sure the code is within the workspace directory somewhere.
c. Open the file SensorTest.java.
d. Click on the “run” button.

**Criteria to Pass:**

The SensorTest runs four test methods – if all four pass then the test is successful. Once the test is finished, it will display the number of failures and errors. If both of these are zero, the test passes.

Tested by: **Austin Robinson**

Date: **4/6/08**

Software Version: **4/6/08 backup**

Pass / Fail: **Pass**

4. **Switch Test**

The purpose of this test is to evaluate the Switch class in the API. The main function of the Switch class is to hold all the information about a switch in the system – this test makes sure that this functionality is correct.

**Steps:**

a. Boot Computer.
b. Start Eclipse – make sure the code is within the workspace directory somewhere.
c. Open the file SwitchTest.java.
d. Click on the “run” button.

**Criteria to Pass:**
The SwitchTest runs nine test methods – if all nine pass then the test is successful. Once the test is finished, it will display the number of failures and errors. If both of these are zero, the test passes.

Tested by: Austin Hobson

Date: 4/16/08

Software Version: 4/6/08 backup

Pass / Fail: pass

5. Station Test
The purpose of this test is to evaluate the Station class in the API. The main function of the Station class is to hold all the information about a station in the system, including all the associated rails, sensors, and switches – this test makes sure that this functionality is correct.

Steps:
 a. Boot Computer.
 b. Start Eclipse – make sure the code is within the workspace directory somewhere.
 c. Open the files StationTest.java and Station.java.
 d. In Station.java, comment out the last line of the setRailValue method that says “build(rv, sv);” so that no packets will be sent out.
 e. In Station.java, comment out the last line of the setSwitchValue method that says “build(rv, sv);” so that no packets will be sent out.
 f. Save all files.
 g. Click on the “run” button.
 h. Once the test is done, un-comment the 2 lines that were commented out for this test and save again.

Criteria to Pass:
The StationTest runs six test methods – if all six pass then the test is successful. Once the test is finished, it will display the number of failures and errors. If both of these are zero, the test passes.

Tested by: Austin Hobson

Date: 4/16/08

Software Version: 4/6/08 backup

Pass / Fail: pass
6. Packet Building and Sending Test
The purpose of this test is to evaluate the constructor and the build method of the
BuilderDecoder class in the API. The main function of the BuilderDecoder class is to
deal with serial port communication – this test makes sure that the Station class can set a
rail or switch value and correctly tell the BuilderDecoder to build and transmit a packet to
different serial ports.

Steps:
a. Boot Computer.
b. Start Eclipse – make sure the code is within the workspace directory somewhere.
c. Open the file Test1.java.
d. In Test1.java, comment out the last two lines of the main method so that the only line that
will be run says “new Test1().testPackets(new ControllerMain());” to make sure the right
test method is called.
e. Save all files.
f. Turn on oscilloscope.
g. Attach oscilloscope probe to the transmit and ground pins of the first serial port of the
octopus (pins 3 and 5, respectively).
h. Set up oscilloscope to trigger on a rising edge of input one with the Y axis set up at 10
v/div and the X axis set up at 5 ms/div.
i. Set up oscilloscope for single run mode.
j. Click on the “run” button on the PC.
k. The PC will print out that it sent one packet to each of the first 5 serial ports of the
octopus. Ensure that what is printed on the screen matches the expected value (see
criteria to pass) and that this is what is displayed on the oscilloscope for that port. (ttyS4
is port 1, ttyS5 is port 2, etc).
l. Attach the oscilloscope to the next port and repeat steps g – l until all 5 serial ports have
been checked.

Criteria to Pass:
For this test to pass, all five ports must print out the expected packet on the screen. Also,
for each port the display on the oscilloscope must show an RS232 encoding of the
expected packet value. The expected printout is as follows (all values are in
hexadecimal, and all each byte is separated by a period):

2f.1c.0.0.0.0.0.0.0.0.0.0.9. sent to /dev/ttyS4
2f.0.4.0.0.0.0.0.0.0.0.6f. sent to /dev/ttyS5
2f.0.0.24.0.0.0.0.0.0.0.0.0bf. sent to /dev/ttyS6
2f.0.0.0.34.0.0.0.0.0.0.0.0.9b. sent to /dev/ttyS7
2f.0.0.0.0.0.0.0.0.0.0.1.21. sent to /dev/ttyS8

Tested by: ____________________________ __________________________

Date: 4/6

Software Version: 4/6 ________________
7. **Maintenance Interface Test**

The purpose of this test is to evaluate the maintenance interface. This will test the main functions of the interface, which are displaying the track layout on the screen, as well as changing rail and switch values.

**Steps:**

a. Boot Computer.
b. Start Eclipse – make sure the code is within the workspace directory somewhere.
c. Open the file MaintenanceInterface.java.
d. Click on the “run” button.
e. Make sure the first station is loaded correctly – 2 rail segments on top, 3 in the middle, 2 on the bottom, one switch at the top pointing right, and one switch at the bottom pointing left.
f. Change the top left combo box to 5 and click the check box below it that says “Top”.
g. Click on the top switch to change it to turning.
h. In the top combo box, select station 2, and make sure the second station is loaded correctly – 2 rail segments on top, 2 in the middle, 2 on the bottom, and no switches.
i. Change the bottom right combo box to 8 and click the check box below it that says “Bottom”.

j. In the top combo box, select station 3, and make sure the third station is loaded correctly – 2 rail segments on top, 3 in the middle, 3 on the bottom, one switch on top pointing right, one switch in the middle pointing left, and 2 switches on the bottom, with one in each direction.

k. In the top combo box, select station 4, and make sure the fourth station is loaded correctly – 2 rail segments on top, 3 in the middle, 2 on the bottom, one switch on top pointing left, and two switches in the middle both pointing right.

l. Change the top right combo box to 3, click the check box below it for top, then click the check box below it for bottom as well.

m. In the top combo box, select station 5, and make sure the fifth station is loaded correctly – 2 rail segments on top, 3 in the middle, 2 on the bottom, one switch on top pointing left, and one switch on the bottom pointing right.

n. Use the top combo box to switch back to station 4 and ensure that the top right rail is still set to 3 with both check boxes selected.

o. Use the top combo box to switch back to station 2 and ensure that the bottom right rail is still set to 8 with the “Bottom” check box still selected.

p. Use the top combo box to switch back to station 1 and ensure that the top left rail is still set to 5 with the “Top” check box still selected, and that the top switch is still set to turn.

**Criteria to Pass:**

To pass the test, all the conditions above must be satisfied, and the program must print out the following data to show that it sent packets correctly.
8. **Packet Decoding and CRC Test**

The purpose of this test is to evaluate the ability of the BuilderDecoder class to decode incoming packets and process them correctly. This test will also ensure that the BuilderDecoder can correctly evaluate the CRC byte for incoming packets. This test will first send bytes to the PC to map a port to a station. Next it will test the ability of the BuilderDecoder to decode correct sensor data. Next it will test the ability of the BuilderDecoder to discover and process a byte with an incorrect CRC. Finally, it will test the ability of the maintenance interface to process sensor data correctly on the screen.

**Steps:**

a. Boot the computer running the java code, as well as another one with an available serial port.

b. On the second computer, open the serial input/output monitor program.

c. Connect the transmit pin (pin 3) of the second computer to the receive pin (pin 2) of the first port of the octopus. Connect the two ground pins (pin 5) together.

d. Start Eclipse – make sure the code is within the workspace directory somewhere.


e. Open the file MaintenanceInterface.java.

f. Click on the “run” button.


g. Using the serial input/output monitor program, send a packet “0242”.

h. Change the second computer to be connected to the second port of the octopus.

i. Using the serial input/output monitor program, send a packet “0363”.

j. Change the second computer to be connected to the third port of the octopus.

k. Using the serial input/output monitor program, send a packet “0484”.

l. Change the second computer to be connected to the fourth port of the octopus.

m. Using the serial input/output monitor program, send a packet “05A5”.

n. Change the second computer to be connected to the fifth port of the octopus.
o. Using the serial input/output monitor program, send a packet “0121”. This finishes the station mapping.

p. Send a packet “000121” to the java computer. The top left sensor on the screen should now be highlighted as active.

q. Connect the computer to the fourth port of the octopus.

r. Send a packet “01FF06” to the java computer.

s. Connect the computer to the third port of the octopus.

t. Send a packet “000121” to the java computer.

u. Connect the computer to the second port of the octopus.

v. Send a packet “01FF06” to the java computer.

w. Connect the computer to the first port of the octopus.

x. Send a packet “000121” to the java computer.

y. On the maintenance interface screen, select station 5 from the combo box. All sensors for this station should be shown as active.

z. On the maintenance interface screen, select station 4 from the combo box. The top left sensor for this station should be shown as active.

aa. On the maintenance interface screen, select station 3 from the combo box. All sensors for this station should be shown as active.

bb. On the maintenance interface screen, select station 2 from the combo box. The top left sensor for this station should be shown as active.

cc. Send a packet “01FF05” to the java computer. This packet should not be accepted because it had a bad CRC value. The java computer should print out “wrote 7 to port /dev/ttyS4”.

Criteria to Pass:

For this test to pass, everything in the procedures that says it should happen must happen. In step p only the top left sensor may appear active. In step y, all sensors must appear active. In step z, only the top left sensor may appear active. In step aa, all sensors must appear active. In step bb, only the top left sensor may appear active. In step cc, all sensors must appear the same on the screen.

Tested by: [Signature]

Date: 4/6

Software Version: 4/6 backup

Pass / Fail: Pass

9. Packet Retransmission Test
The purpose of this test is to ensure that if the BuilderDecoder receives an error byte that it can retransmit the correct packet.

Steps:
a. Boot the computer running the java code, as well as another one with an available serial port.
b. On the second computer, open the serial input/output monitor program.
c. Connect the transmit pin (pin 3) of the second computer to the receive pin (pin 2) of the first port of the octopus. Connect the two ground pins (pin 5) together.
d. Start Eclipse – make sure the code is within the workspace directory somewhere.
e. Open the file MaintenanceInterface.java.
f. Click on the “run” button.
g. Set the value of the top left combo box to 1.
h. Set the value of the top left combo box to 2.
i. Set the value of the top left combo box to 3.
j. Set the value of the top left combo box to 4.
k. From the other computer, use the serial input/output monitor program to send the value “91”. The java computer should print out “available: 1” and then on the next line “2f.3.0.0.0.0.0.0.0.0.49. sent to /dev/ttyS4”
l. On the maintenance interface, use the top combo box to switch to station 2.
m. Set the value of the top left combo box to 5.
n. Connect the other computer to the second port of the octopus.
o. From the other computer, use the serial input/output monitor program to send the value “00”. The java computer should print out “available: 1” and then on the next line “2f.5.0.0.0.0.0.0.0.0.0.db. sent to /dev/ttyS5”
p. On the maintenance interface, use the top combo box to switch to station 3.
q. Set the value of the top left combo box to 6.
r. Set the value of the top left combo box to 7.
s. Set the value of the top left combo box to 8.
t. Connect the other computer to the third port of the octopus.
u. From the other computer, use the serial input/output monitor program to send the value “00”. The java computer should print out “available: 1” and then on the next line “2f.6.0.0.0.0.0.0.0.0.0.92. sent to /dev/ttyS6”
v. From the other computer, use the serial input/output monitor program to send the value “06”. The java computer should print out “available: 1” and then on the next line “2f.6.0.0.0.0.0.0.0.0.0.92. sent to /dev/ttyS6”

Criteria to Pass:
The test passes if and only if all conditions specified in the procedure happen as expected.

Tested by: Austin Robinson

Date: 4/6

Software Version: 4/6 backup

Pass / Fail: Pass
#include <18F458.h>

#device adc=8
#device HIGH_INTS=true

//Serial most important..then polls..then msg from other pics.
///#PRIORITY rda,timer0,timer3,ccp1,ext

#FUSES NOWDT  //No Watch Dog Timer
#FUSES WDT128 //Watch Dog Timer uses 1:128 Postscale
#FUSES HS    //High speed Osc (> 4mhz)

#FUSES PROTECT

#FUSES BROWNOUT  //Reset when brownout detected
#FUSES BORV20   //Brownout reset at 2.0V

#use delay(clock=40000000)

#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8,stream=PC)
#use rs232(baud=9600,parity=N,xmit=PIN_D4,rcv=PIN_D5,bits=8,stream=DBG)

#use rs232(baud=9600, float_high, bits=9, xmit=PIN_B0, rcv=PIN_B0,stream=PIC)

/*
 * PIC-to-PIC Communication Definitions
 *
*/
#define OUR_ID 4
#define MAX_MESSAGES 4
#define MAX_LENGTH 5

int8 i;
int8 j;
byte pbus_buffer[MAX_MESSAGES][MAX_LENGTH+1];
byte next_in=0;
byte next_out=0;
enum pbus_states {PBUS_IDLE=0x80,PBUS_NEED_LEN=0x81,PBUS_NEED_TO=0x82,
PBUS_NEED_FROM=0x83,PBUS_NEED_CS=0x84};
byte pbus_state=PBUS_IDLE;
byte checksum;
byte last_byte;

#define ninth_bit = RS232_ERRORS.7
#define collision = RS232_ERRORS.6
#define intf = 11.1

int1 s0;
int1 s1;
int1 s2;
int1 s3;
int1 s4;
int1 s5;
int1 s6;
int1 s7;
int1 s8;

int8 station_packet[2] = {0,0};
int1 station_registered = 0;

int1 sensors[9];
int1 temp[9];

int1 train_passed;
char sensor_packet[3] = {0,0,0};

char switches;
char slave0[3];
char slave1[3];
char slave2[3];
int8 tries = 0;
#DEFINE MAX_INPUT 20
char input[MAX_INPUT];

//Size is 10+CRCbyte
#DEFINE PSIZE 11
char packet[PSIZE]={0,0,0,0,0,0,0,0,0,0,0};
int1 packet_found = 0;
int1 CRC_OK = 0;
int8 CRC;
char error_byte = 0xFF;
char last_ok_crc = 0x00;

#define QSIZE 10
char queue[QSIZE]={0,0,0,0,0,0,0,0,0,0,0};
char task0 = 0;
char task3 = 0;
int1 t1_active = 0;
int1 t3_active = 0;
int1 s0l=0;
int1 s0r=0;
int1 s1l=0;
int1 s1r=0;
int1 s2l=0;
int1 s2r=0;
int1 s3l=0;
int1 s3r=0;
int1 s0l3=0;
int1 s0r3=0;
int1 s1l3=0;
int1 s1r3=0;
int1 s2l3=0;
int1 s2r3=0;
int1 s3l3=0;
int1 s3r3=0;

// Each tick is 1/10Mhz = 0.1usec
// To get 5.6msec, need 56000 ticks
int16 CCP_VAL = 56000;
// Each cycle is 1/10MHz = 0.1us
// for 1msec, need 10000 counts
// 65536 - 10000 = 55536
int16 POLL_VAL = 55536;
#include "C:\Program Files\PICC\Projects\NetworkSystem\MasterPIC\MasterPIC.h"
#include <input.c>
#include <crc.c>

// If both tasks in queue is 0, queue is empty
int1 queue_isEmpty()
{
    int1 empty = 1;
    // int8 i = 0;

    for(i = 0; i<QSIZE; i++)
    {
        if(queue[i] != 0x00) {
            empty = 0;
            break;
        }
    }
    return empty;
}

// If both tasks in queue is non-zero, queue is full
int1 queue_isFull()
{
    int1 full = 1;
    // int8 i = 0;

    for(i = 0; i<QSIZE; i++)
    {
        if(queue[i] == 0x00) {
            full = 0;
            break;
        }
    }
    return full;
}

void queue_addTask(char task){
    // int8 i = 0;
    for(i = 0; i<QSIZE; i++)
    {
        if(queue[i] == 0x00) {
            queue[i] = task;
            break;
        }
    }
}

char queue_remTask(){
    // int8 i = 0;
    char task;
    task = queue[0];
    // Shift the array
    for(i = 0; i<QSIZE; i++)
    {
        queue[i] = queue[i+1];
        if(queue[i+1] == 0x00) break;
    }
return task;
}

//PIC-to-PIC Interrupt System
#include

void pbus_isr() {
  byte data;

  if(kbhit(PIC)) {
    data=getc(PIC);
    if(ninth_bit) {
      switch(pbus_state) {
        case PBUS_IDLE : if(data==0xf1)
          pbus_state=PBUS_NEED_TO;
        break;
        case PBUS_NEED_TO :
          if(data==OUR_ID){
            pbus_state=PBUS_NEED_FROM;}
          else
            pbus_state=PBUS_IDLE;
          checksum=data;
        break;
        case PBUS_NEED_FROM :
          pbus_buffer[next_in][0] = data;
          pbus_state=PBUS_NEED_LEN;
          checksum^=data;
        break;
        case PBUS_NEED_LEN :
          last_byte = data+1;
          pbus_buffer[next_in][1] = data;
          pbus_state=2;
          checksum^=data;
        break;
      }
    } else
      if(pbus_state==PBUS_NEED_CS) {
        if(checksum==data){
          next_in = (next_in+1) % MAX_MESSAGES;
        }
        pbus_state=PBUS_IDLE;
      } else if(pbus_state<0x80) {
        pbus_buffer[next_in][pbus_state] = data;
        checksum^=data;
        if(++pbus_state>last_byte)
          pbus_state=PBUS_NEED_CS;
      }
    }
  }
}
void pbus_send( byte * message, byte to, byte len) {
    byte checksum,i;
    
    retry:
        checksum=len^OUR_ID^to;
        
        //Check this...might be unnecessary
        disable_interrupts(GLOBAL);
        collision=false;
        ninth_bit=1;
        
        fputs(0xf1,PIC);   if(collision) goto error;
        fputs(to,PIC);     if(collision) goto error;
        fputs(OUR_ID,PIC); if(collision) goto error;
        fputs(len,PIC);    if(collision) goto error;
        ninth_bit=0;
        for(i=1;i<=len;++i) {
            checksum ^= *message;
            fputs(*(message++),PIC); if(collision) goto error;
        }
        
        fputs(checksum,PIC);   if(collision) goto error;
        intf=false;
        
        enable_interrupts(GLOBAL);
        return;
        
    error:
        delay_ms(6);
        enable_interrupts(GLOBAL);
        goto retry;
}

#define RDA_HIGH

void serial_interrupt_isr() {

    //fgets(input,DBG);
    fgets(input,PC);
    
    for(i = 0; i<MAX_INPUT; i++){
        if(input[i] == 0x2F){
            station_registered = 1;
            packet_found = 1;
            for(j = 0; j<PSIZE; j++){
                packet[j] = input[i+j+1];
            }
            break;
        }
    }

    if(packet_found){
        CRC = generate_8bit_crc(packet, PSIZE-1, CRC_CCITT);
        fprintf(DBG, "\r\nCRC = %2X",CRC);
        fprintf(DBG, "\r\npacket10 = %2X",packet[10]);
    }
if(CRC == packet[PSIZE-1]) {
    CRC_OK = 1;
    last_ok_crc = CRC;
}

if(!CRC_OK) {
    fputc(last_ok_crc, PC);
}

if(CRC_OK) {
    fprintf(DBG, "\n\n\np0 = %2X", packet[0]);
    fprintf(DBG, "p1 = %2X", packet[1]);
    fprintf(DBG, "p2 = %2X", packet[2]);
    fprintf(DBG, "p3 = %2X", packet[3]);
    fprintf(DBG, "p4 = %2X", packet[4]);
    fprintf(DBG, "p5 = %2X", packet[5]);
    fprintf(DBG, "p6 = %2X", packet[6]);
    fprintf(DBG, "p7 = %2X", packet[7]);
    fprintf(DBG, "p8 = %2X", packet[8]);
    fprintf(DBG, "p9 = %2X", packet[9]);

    slave0[0] = packet[0];
    slave0[1] = packet[1];
    slave0[2] = packet[2];

    slave1[0] = packet[3];
    slave1[1] = packet[4];
    slave1[2] = packet[5];

    slave2[0] = packet[6];
    slave2[1] = packet[7];
    slave2[2] = packet[8];

    switches = packet[9];

    pbus_send(slave0, 0, 3);
    pbus_send(slave1, 1, 3);
    pbus_send(slave2, 2, 3);

    /*
    while(!input(PIN_B1)||!input(PIN_B2)||!input(PIN_B3)){
        if(!input(PIN_B1)){
            pbus_send(slave0, 0, 3);
        }
        if(!input(PIN_B2)){
            pbus_send(slave1, 0, 3);
        }
        if(!input(PIN_B3)){
            pbus_send(slave2, 0, 3);
        }
    }
*/
tries++; if(tries>=5){
    fprintf(DBG,"Error in connecting SlavePic: ");
    if(!input(PIN_B1)) fprintf(DBG,"0,");
    if(!input(PIN_B2)) fprintf(DBG,"1,");
    if(!input(PIN_B3)) fprintf(DBG,"2,");
    break;
}
}*/

if(switches != 0x00) {  //If the switch byte is non-zero..
    if(!queue_isFull()) {  //And if the queue is not full
        // queue_addTask(0x01);
        // queue_addTask(0x02);
        queue_addTask(switches);  //Add the task
        fprintf(DBG,"queue is: %2X%2X%2X%2X%2X%2X%2X%2X",
                queue[0],queue[1],queue[2],queue[3],queue[4],
                queue[5],queue[6],queue[7],queue[8],queue[9]);
    
        if(!t1_active) {  //if queue1 is not active,
            set_timer1(CCP_VAL-1);  //set timer1 to interrupt immediately
        }
    }
    else{
        fprintf(DBG,"Queue is full maaan");
    }
}

packet[0] = 0;
packet[1] = 0;
packet[2] = 0;
packet[3] = 0;
packet[4] = 0;
packet[5] = 0;
packet[6] = 0;
packet[7] = 0;
packet[8] = 0;
packet[9] = 0;
for(i = 0; i<20; i++){
    input[i] = 0;
}

packet_found = 0;
CRC_OK = 0;

}/*
* Sensor Interrupt System
* Sensor1-A0
* Sensor2-A1
* Sensor3-A2
* Sensor4-A3
* Sensor5-A4
* Sensor6-A5
* Sensor7-E0
* Sensor8-E1
*/

#definetimer

voidsensor_isr(){

OUTPUT_HIGH(PIN_C0);
delay_us(10);

sensor_packet[0]=0;
sensor_packet[1]=0;

train_passed = 0;

s0 = !input(PIN_A0);
s1 = !input(PIN_A1);
s2 = !input(PIN_A2);
s3 = !input(PIN_A3);
s4 = !input(PIN_A4);
s5 = !input(PIN_A5);
s6 = !input(PIN_E0);
s7 = !input(PIN_E1);
s8 = !input(PIN_E2);

if(s0 && (s0 != sensors[0]) ||
    s1 && (s1 != sensors[1]) ||
    s2 && (s2 != sensors[2]) ||
    s3 && (s3 != sensors[3]) ||
    s4 && (s4 != sensors[4]) ||
    s5 && (s5 != sensors[5]) ||
    s6 && (s6 != sensors[6]) ||
    s7 && (s7 != sensors[7]) ||
    s8 && (s8 != sensors[8])){

    train_passed = 1;
}

sensors[0] = s0;
sensors[1] = s1;
sensors[2] = s2;
sensors[3] = s3;
sensors[4] = s4;
sensors[5] = s5;
sensors[6] = s6;
sensors[7] = s7;
sensors[8] = s8;

if(train_passed) {
  s0 | s1<<1 | s2<<2 | s3<<3 | s4<<4 | s5<<5 | s6<<6 | s7<<7;
  sensor_packet[0] = sensor_packet[0] | s8;
  sensor_packet[2] = generate_8bit_crc(sensor_packet,2,CRC_CCITT);
  fputc(sensor_packet[0],PC);
  fputc(sensor_packet[1],PC);
  fputc(sensor_packet[2],PC);
  fprintf(DBG,"
Sensor[0]:%2X
Sensor[1]:%2X
SensorCRC:%2X",sensor_packet[0],sensor_packet[1],sensor_packet[2]);
  //fputs(sensor_packet,PC);
}
set_timer0(POLL_VAL);  //So timer counts 10000 clicks..1msec
OUTPUT_LOW(PIN_C0);

#define CCP1
void queue1_isr() {
  //Debug
  OUTPUT_HIGH(PIN_B6);
  delay_us(100);
  OUTPUT_LOW(PIN_B6);

  //We must stop outputting high on the pins since 5.6msec passed
  //If we're activated after a non-active state, then outputting low can't hurt
  if (s0l==1) OUTPUT_LOW(PIN_C2);  //Switch 0/L
  if (s0r==1) OUTPUT_LOW(PIN_C3);  //Switch 0/R
  if (s1l==1) OUTPUT_LOW(PIN_D0);  //Switch 1/L
  if (s1r==1) OUTPUT_LOW(PIN_D1);  //Switch 1/R
  if (s2l==1) OUTPUT_LOW(PIN_C5);  //Switch 2/L
  if (s2r==1) OUTPUT_LOW(PIN_C4);  //Switch 2/R
if (s3l==1) OUTPUT_LOW(PIN_D3);  //Switch 3/L
if (s3r==1) OUTPUT_LOW(PIN_D2);  //Switch 3/R

if(queue_isEmpty()) {
    set_timer1(0);
    t1_active = 0;
}
else {
    //Set the flag
    t1_active = 1;

    //Now we can get new task and accomplish it
    task0 = queue_remTask();

    s0l = (task0 & 0x01);
    s0r = (task0 & 0x02)>>1;
    s1l = (task0 & 0x04)>>2;
    s1r = (task0 & 0x08)>>3;
    s2l = (task0 & 0x10)>>4;
    s2r = (task0 & 0x20)>>5;
    s3l = (task0 & 0x40)>>6;
    s3r = (task0 & 0x80)>>7;

    OUTPUT_BIT(PIN_C2,s0l);  //Switch 0/L
    OUTPUT_BIT(PIN_C3,s0r);  //Switch 0/R
    OUTPUT_BIT(PIN_D0,s1l);  //Switch 1/L
    OUTPUT_BIT(PIN_D1,s1r);  //Switch 1/R
    OUTPUT_BIT(PIN_C5,s2l);  //Switch 2/L
    OUTPUT_BIT(PIN_C4,s2r);  //Switch 2/R
    OUTPUT_BIT(PIN_D3,s3l);  //Switch 3/L
    OUTPUT_BIT(PIN_D2,s3r);  //Switch 3/R

    set_timer1(0);                     //Set timer1 to interrupt again in 5.6msec

    /*if(!queue_isEmpty()){
        //If there are more tasks
        if(!t3_active){
            //AND if timer3 is not aware..make him work
            set_timer3(CCP_VAL-1);
        }
    } */
}

#if CCP2

void queue2_isr(){

}
#endif
//Debug
OUTPUT_HIGH(PIN_B7);
delay_us(100);
OUTPUT_LOW(PIN_B7);

//We must stop outputting high on the pins since 5.6msec passed
//If we're activated after a non-active state, then outputting low can't hurt
if (s0l3==1) OUTPUT_LOW(PIN_C2); //Switch 0/L
if (s0r3==1) OUTPUT_LOW(PIN_C3); //Switch 0/R
if (s1l3==1) OUTPUT_LOW(PIN_D0); //Switch 1/L
if (s1r3==1) OUTPUT_LOW(PIN_D1); //Switch 1/R
if (s2l3==1) OUTPUT_LOW(PIN_C5); //Switch 2/L
if (s2r3==1) OUTPUT_LOW(PIN_C4); //Switch 2/R
if (s3l3==1) OUTPUT_LOW(PIN_D3); //Switch 3/L
if (s3r3==1) OUTPUT_LOW(PIN_D2); //Switch 3/R*

if(queue_isEmpty()){  
  set_timer3(0);
  t3_active = 0;
}  

else{  
  //Set the flag
  t3_active = 1;
  OUTPUT_HIGH(PIN_D2);
  //Now we can get new task and accomplish it
  task3 = queue_remTask();

  s0l3 = (task3 & 0x01);
  s0r3 = (task3 & 0x02)>>1;
  s1l3 = (task3 & 0x04)>>2;
  s1r3 = (task3 & 0x08)>>3;
  s2l3 = (task3 & 0x10)>>4;
  s2r3 = (task3 & 0x20)>>5;
  s3l3 = (task3 & 0x40)>>6;
  s3r3 = (task3 & 0x80)>>7;

  OUTPUT_BIT(PIN_C2,s0l3); //Switch 0/L
  OUTPUT_BIT(PIN_C3,s0r3); //Switch 0/R
  OUTPUT_BIT(PIN_D0,s1l3); //Switch 1/L
  OUTPUT_BIT(PIN_D1,s1r3); //Switch 1/R
  OUTPUT_BIT(PIN_C5,s2l3); //Switch 2/L
  OUTPUT_BIT(PIN_C4,s2r3); //Switch 2/R
  OUTPUT_BIT(PIN_D3,s3l3); //Switch 3/L
  OUTPUT_BIT(PIN_D2,s3r3); //Switch 3/R

  set_timer3(0); //Set timer0 to interrupt again in 5.6msec
void main()
{

delay_ms(1000);
OUTPUT_LOW(PIN_C0);

setup_adc_ports(NO_ANALOGS);
setup_adc(ADC_OFF);
setup_psp(PSP.Disabled);
setup_spi(SPI_SS.Disabled);
setup_wdt(WDT_OFF);

    //For Switch Queues
setup_ccp1(CCP_COMPARE_INT);
setup_timer_1(T1_INTERNAL);
CCP_1 = CCP_VAL;
set_timer1(0);
enable_interruptions(INT_CCP1);

    //delay_us(5600);            //Give delay so ccps dont clash

setup_ccp2(CCP_COMPARE_INT|CCP_USE_TIMER3);
setup_timer_3(T3_INTERNAL);
CCP_2 = CCP_VAL;
set_timer3(0);
//enable_interruptions(INT_CCP2);

setup_comparator(NC_NC_NC_NC);
setup_vref(FALSE);

    //For Sensor Polling
setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
set_timer0(60536);
enable_interruptions(INT_TIMER0);

    //For PC2PIC Comm.
enable_interruptions(INT_RDA);

    //For PIC2PIC Comm.
enable_interrupts(INT_EXT);
ext_int_edge(0,H_TO_L);

enable_interrupts(GLOBAL);

//fprintf(DBG,"\r\nOur unit ID is \%d\r\n",OUR_ID);

if(input_state(PIN_B7)) station_packet[0] += 4;
if(input_state(PIN_B6)) station_packet[0] += 2;
if(input_state(PIN_B5)) station_packet[0] += 1;

station_packet[1] = generate_8bit_crc(station_packet,1,CRC_CCITT);
for(i = 0; i<20; i++){
   input[i] = 0;
}

fprintf(DBG,"\r\nStationID[0]:%2X\r\nStationCRC[1]:%2X",station_packet[0],station_packet[1]);

while(!station_registered){
   //fputs(station_packet,PC);
fputc(station_packet[0],PC);
fputc(station_packet[1],PC);
delay_ms(100);
}

delay_ms(100);
fprintf(DBG,"got out of loop");

do{
   
   if(next_in!=next_out) {
      fprintf(DBG,"\r\nMessage from #%d: ",pbus_buffer[next_out][0]);
      for(i=2;i<=pbus_buffer[next_out][1]+1;++i){
         //fprintf(DBG," %2X",pbus_buffer[next_out][i]);
         if(pbus_buffer[next_out][i] == 0xFF){
            fprintf(DBG,"There was erROR!\n\r\n");
         }
      }

      next_out=(next_out+1) % MAX_MESSAGES;
   }
*/

if(kbhit(PC))
   if(toupper(fgetc(PC))=='S') {
      fprintf(PC,"\r\nSend to: ");
   } /*

if(kbhit(PC))
   if(toupper(fgetc(PC))=='S') {
      fprintf(PC,"\r\nSend to: ");
   } /*

file:///Users/home/Desktop/MasterPIC.txt (11 of 12) [4/14/08 11:23:45 PM]
to=gethex();
fprintf(PC,"\nLength: ");
len=gethex();
for(i=0;i<len;++i) {
    fprintf(PC,"\nByte %d: ",i+1);
    msg[i]=gethex();
}
pbus_send(msg,to,len);
fprintf(PC,"\nSent.");
}/*
}
while(true);
```c
#include <18F2431.h>

#define MAX_MESSAGES 4
#define MAX_LENGTH 5 //Total length of message to be sent

char error_byte = 0xFF;
int1 error = 0;

byte pbus_buffer[MAX_MESSAGES][MAX_LENGTH+1];
byte next_in = 0;
byte next_out = 0;
enum pbus_states {PBUS_IDLE=0x80,PBUS_NEED_LEN=0x81,PBUS_NEED_TO=0x82,
PBUS_NEED_FROM=0x83,PBUS_NEED_CS=0x84};
byte pbus_state = PBUS_IDLE;
byte checksum;
byte last_byte;

#define ninth_bit RS232_ERRORS.7
#define collision RS232_ERRORS.6
#define intf 11.1

int8 packet[3];

int1 p0a;
int1 p0b;
int1 p1a;
int1 p1b;
int1 p2a;
int1 p2b;
```
int8 s0;
int8 s1;
int8 s2;

int8 OUR_ID;
int32 duty, period, num;

int32 desired0 = 0;
int32 desired1 = 0;
int32 desired2 = 0;

int32 current0 = 0;
int32 current1 = 0;
int32 current2 = 0;
#include "C:\Program Files\PIC\Projects\NetworkSystem\SlavePIC\SlavePIC.h"
#include <input.c>

#define EXT

void pbus_isr() {
    byte data;

    if(kbhit(PIC)) {
        data=getc(PIC);

        if(ninth_bit) {
            switch(pbus_state) {
            case PBUS_IDLE : if(data==0xf1)
                pbus_state=PBUS_NEED_TO;
                break;
            case PBUS_NEED_TO :
                if(data==OUR_ID)
                    pbus_state=PBUS_NEED_FROM;
                else
                    pbus_state=PBUS_IDLE;
                checksum=data;
                break;
            case PBUS_NEED_FROM :
                pbus_buffer[next_in][0] = data;
                pbus_state=PBUS_NEED_LEN;
                checksum^=data;
                break;
            case PBUS_NEED_LEN :
                last_byte = data+1;
                pbus_buffer[next_in][1] = data;
                pbus_state=2;
                checksum^=data;
                break;
            }
        } else
            if(pbus_state==PBUS_NEED_CS) {
                if(checksum==data){
                    next_in = (next_in+1) % MAX_MESSAGES;
                    OUTPUT_HIGH(PIN_C2);
                }
            else{
                OUTPUT_LOW(PIN_C2);
                fprintf(PC,"error");
            }
        pbus_state=PBUS_IDLE;
    } else if(pbus_state<0x80) {
}
pbus_buffer[next_in][pbus_state] = data;
    checksum^=data;
    if(++pbus_state>last_byte)
        pbus_state=PBUS_NEED_CS;
    }
}
}

void pbus_send( byte * message, byte to, byte len) {
    byte checksum,i;

    retry:
        checksum=len^OUR_ID^to;
        disable_interrupts(GLOBAL);
        collision=false;
        ninth_bit=1;

        fputc(0xf1,PIC);      if(collision) goto errrr;
        fputc(to,PIC);        if(collision) goto errrr;
        fputc(OUR_ID,PIC);    if(collision) goto errrr;
        fputc(len,PIC);       if(collision) goto errrr;
        ninth_bit=0;
        for(i=1;i<=len;++i) {
            checksum^=*message;
            fputc(*(message++),PIC); if(collision) goto errrr;
        }
        fputc(checksum,PIC);  if(collision) goto errrr;
        intf=false;
        enable_interrupts(GLOBAL);
        return;

errrr:
    delay_ms(6);
    enable_interrupts(GLOBAL);
    goto retry;
}

/*
Polarities:
0 - Negative
1 - Positive

Duty Cycle速s:
*/

void set_rails() {

    s0 = packet[0] & 0x0F;

p0a = (packet[0] & 0x10)>>4;
p0b = (packet[0] & 0x20)>>5;

s1 = packet[1] & 0x0F;
p1a = (packet[1] & 0x10)>>4;
p1b = (packet[1] & 0x20)>>5;

s2 = packet[2] & 0x0F;
p2a = (packet[2] & 0x10)>>4;
p2b = (packet[2] & 0x20)>>5;

OUTPUT_BIT(PIN_A0,p0a);
OUTPUT_BIT(PIN_A1,p0b);
OUTPUT_BIT(PIN_A2,p1a);
OUTPUT_BIT(PIN_A3,p1b);
OUTPUT_BIT(PIN_A4,p2a);
OUTPUT_BIT(PIN_A7,p2b);

//R0-pin26
duty = 9 + (6*s0);
desired0 = (period * duty)/25;

/*num = period * duty;
num = num / 25;
set_power_pwm4_duty(num);*/

//R1-pin23
duty = 9 + (6*s1);
desired1 = (period * duty)/25;
/*num = period * duty;
num = num / 25;
set_power_pwm2_duty(num);*/

//R2-pin21
duty = 9 + (6*s2);
desired2 = (period * duty)/25;
/*num = period * duty;
num = num / 25;
set_power_pwm0_duty(num);*/
enable_interrupts(INT_CCP1);

void rail0_isr() {

}
if(current0 == desired0 &&
  current1 == desired1 &&
  current2 == desired2 ){

    disable_interrupts(INT_CCP1);
}

else{

  if(current0 != desired0){
    if(current0 < desired0) current0 = current0+1;
    if(current0 > desired0) current0 = current0-1;

    set_power_pwm4_duty(current0);
  }

  if(current1 != desired1){
    if(current1 < desired1) current1 = current1+1;
    if(current1 > desired1) current1 = current1-1;

    set_power_pwm2_duty(current1);
  }

  if(current2 != desired2){
    if(current2 < desired2) current2 = current2+1;
    if(current2 > desired2) current2 = current2-1;

    set_power_pwm0_duty(current2);
  }

  set_timer1(0);
}


void main() {
  byte to,len,msg[MAX_LENGTH];
  byte i;
  setup_oscillator(OSC_8MHZ);

  if      (!input(PIN_B7) && !input(PIN_B6)) OUR_ID = 0;
  else if (!input(PIN_B7) &&  input(PIN_B6)) OUR_ID = 1;
  else if ( input(PIN_B7) && !input(PIN_B6)) OUR_ID = 2;
  else if ( input(PIN_B7) &&  input(PIN_B6)) OUR_ID = 3;

  if ( OUR_ID == 0 )

  else if ( OUR_ID == 1 )

  else if ( OUR_ID == 2 )

  else if ( OUR_ID == 3 )

}
period = 3875;
setup_power_pwm_pins(PWM_BOTH_ON,PWM_BOTH_ON,PWM_BOTH_ON,PWM_OFF);
    //PPWM channels 0 and 1 are both on and always opposite values
setup_power_pwm(PWM_CLOCK_DIV_64|PWM_FREE_RUN|PWM_DEAD_CLOCK_DIV_2,16,0,period,0,1,0);
    //add dead time for reactive loads

setup_ccp1(CCP_COMPARE_INT);
setup_timer_1(T1_INTERNAL); //Count 1ms
CCP_1 = 100;
set_timer1(0);
    // enable_interrupts(INT_CCP1);

delay_ms(1000);
fprintf(PC,"\r\nOur unit ID is %d\r\nPress S to send.\r\n",OUR_ID);

ext_int_edge( h_to_l );
enable_interrupts(int_ext);
OUTPUT_HIGH(PIN_C2);

enable_interrupts(global);

do {
    /* if(error){
       //error_byte = 0xFF;
       //pbus_send(error_byte,4,1);
       //error = 0;
    }*/

    if(next_in!=next_out) {
        fprintf(PC,"\r\nMessage from #%d: ",pbus_buffer[next_out][0]);
        for(i=2;i<=pbus_buffer[next_out][1]+1;++i){
            fprintf(PC,"\r\nRail %d Speed: %2X",i-1, pbus_buffer[next_out][i]);
            packet[i-2] = pbus_buffer[next_out][i];
        }
        next_out=(next_out+1) % MAX_MESSAGES;
        set_rails();
    }

    //Needed for debugging
    if(kbhit(PC))
if(toupper(fgetc(PC))=='S') {
    fprintf(PC,"\r\nSend to: ");
    to=gethex();
    fprintf(PC,"\r\nLength: ");
    len=gethex();
    for(i=0;i<len;++i) {
        fprintf(PC,"\r\nByte %d: ",i+1);
        msg[i]=gethex();
    }
    pbus_send(msg,to,len);
    fprintf(PC,"\r\nSent.");
}

} while (TRUE);
import junit.framework.*;

public class ControllerMainTest extends TestCase {

    ControllerMain c;
    BuilderDecoder b;

    protected void setUp() throws Exception {
        super.setUp();
        c = new ControllerMain();
        b = new BuilderDecoder();
    }

    public void testGetStationList() {
        assertEquals(5, c.getStations().size());
        assertEquals(5, c.getNumStations());
        assertEquals(8, c.getMaxRails());
        assertEquals(4, c.getMaxSwitches());

        Station s0 = c.getStations().get(0);
        assertEquals("Station 1 - 3rd Street Terminal",
                s0.getName());
        assertEquals(7, s0.getNumRails());
        assertEquals(2, s0.getSwitches().size());
        assertEquals(6, s0.getSensors().size());
        assertEquals(1, s0.getNum());

        Station s1 = c.getStations().get(1);
        assertEquals("Station 2 - Sullivan Village Terminal",
                s1.getName());
        assertEquals(6, s1.getNumRails());
        assertEquals(0, s1.getSwitches().size());
        assertEquals(9, s1.getSensors().size());
        assertEquals(2, s1.getNum());

        Station s2 = c.getStations().get(2);
        assertEquals("Station 3 - Skillman Terminal",
                s2.getName());
        assertEquals(8, s2.getNumRails());
        assertEquals(4, s2.getSwitches().size());
        assertEquals(9, s2.getSensors().size());
    }
}
assertEquals(3, s2.getNum());

Station s3 = c.getStations().get(3);
assertEquals("Station 4 - Farinon Terminal",
            s3.getName());
assertEquals(6, s3.getNumRails());
assertEquals(3, s3.getSwitches().size());
assertEquals(8, s3.getSensors().size());
assertEquals(4, s3.getNum());

Station s4 = c.getStations().get(4);
assertEquals("Station 5 - Metzgar Terminal",
            s4.getName());
assertEquals(7, s4.getNumRails());
assertEquals(2, s4.getSwitches().size());
assertEquals(6, s4.getSensors().size());
assertEquals(5, s4.getNum());
}

public void testMax(){
    assertEquals(8, c.getMaxRails());
    assertEquals(4, c.getMaxSwitches());
}

public void testSensorEvent(){
    boolean[] sensorValues = {true, false, true, false,
                              true, false, false, false,
                              false, false, false, false,
                              false, false, false, false};
    c.sensorEvent(0, sensorValues);
    Station s = c.getStations().get(0);
    assertEquals(true, s.getSensorValue(s.getSensors().get(0)));
    assertEquals(false, s.getSensorValue(s.getSensors().get(1)));
    assertEquals(true, s.getSensorValue(s.getSensors().get(2)));
    assertEquals(false, s.getSensorValue(s.getSensors().get(3)));
    assertEquals(true, s.getSensorValue(s.getSensors().get(4)));
    assertEquals(false, s.getSensorValue(s.getSensors().get(5)));
}
}
import junit.framework.TestCase;

public class RailTest extends TestCase {

    Rail r;
    Station s;
    BuilderDecoder b;

    protected void setUp() throws Exception {
        r = new Rail();
        s = new Station();
        b = new BuilderDecoder();
        super.setUp();
    }

    public void testConstructor() {
        Station s1 = new Station("Skillman", b, 15, 5, 0, 0);
        Station s2 = new Station("3rd Street", b, 15, 5, 0, 0);
        Rail r1 = new Rail(8, s1);
        Rail r2 = new Rail(1, s2);
        Rail r3 = new Rail(2, 1, s2, r2, r1, 350);
        assertEquals(r2, r3.getLeft());
        assertEquals(r1, r3.getRight());
        assertEquals(s2, r3.getStation());
        assertEquals(2, r3.getNum());
        assertEquals(1, r3.getRow());
        assertEquals(350, r3.getLength());
    }

    public void testSetLeft() {
        Station s3 = new Station("Farinon", b, 15, 5, 0, 0);
        Rail r4 = new Rail(16, s3);
        r.setLeft(r4);
        assertEquals(r4, r.getLeft());
    }

    public void testSetRight() {
        Station s4 = new Station("Metzgar", b, 15, 5, 0, 0);
        Rail r5 = new Rail(24, s4);
        r.setRight(r5);
    }
}
assertEquals(r5, r.getRight());

public void testSetStation() {
    Station s5 = new Station("Farinon", b, 15, 5, 0, 0);
    r.setStation(s5);
    assertEquals(s5, r.getStation());
}

class TestRail {
    public void testSetNum() {
        r.setNum(2);
        assertEquals(2, r.getNum());
    }

    public void testSetRow() {
        r.setRow(3);
        assertEquals(3, r.getRow());
    }

    private Station s5 = new Station("Farinon", b, 15, 5, 0, 0);
    private Rail r = new Rail(b, 15, 5, 0, 0);
}

private class Rail {
    private class Station {
        public Station(String name, int b, int x, int y, int z) {
            // Constructor implementation
        }
    }
    public Station getStation() {
        return s5;
    }
    public int getNum() {
        return 2;
    }
    public int getRow() {
        return 3;
    }
}

import junit.framework.*;

general SensorTest extends TestCase {

    Sensor s;
    Rail r;
    Station st;
    Rail.railPosition rp;
    BuilderDecoder b;

    protected void setUp() throws Exception {
        b = new BuilderDecoder();
        s = new Sensor();
        r = new Rail();
        st = new Station();
        super.setUp();
    }

    public void testConstructor(){
        rp = Rail.railPosition.RAIL_LEFT;
        s = new Sensor(r,2,rp);
        assertEquals(r,s.getRail());
        assertEquals(2,s.getNum());
        assertEquals(rp,s.getPosition());
    }

    public void testSetPosition() {
        Sensor s1 = new Sensor();
        Rail.railPosition rp1 = Rail.railPosition.RAIL_RIGHT;
        s1.setPosition(rp1);
        assertEquals(rp1,s1.getPosition());
    }

    public void testSetRail() {
        Rail r1 = new Rail();
        s.setRail(r1);
        assertEquals(r1,s.getRail());
    }

    public void testSetNum() {
        Sensor s1 = new Sensor();
    }
}
Sensor s2 = new Sensor();
s1.setNum(10);
assertEquals(10, s1.getNum());
s2.setNum(2);
assertEquals(2, s2.getNum());
import junit.framework.*;

class StationTest extends TestCase {

    Station s;
    BuilderDecoder b;

    protected void setUp() throws Exception {
        super.setUp();
        s = new Station();
        b = new BuilderDecoder();
    }

    public void testConstructorAndName() {
        s = new Station("Station1", b, 15, 5, 3, 2);
        assertEquals("Station1", s.getName());
        s.setName("abcde");
        assertEquals("abcde", s.getName());
        assertEquals(3, s.getNumRails());
        assertEquals(2, s.getNum());
    }

    public void testRails() {
        Rail r1 = new Rail();
        Rail r2 = new Rail();
        Rail r3 = new Rail();
        s.addRail(r1);
        assertEquals(r1, s.getRails().get(0));
        s.addRail(r2);
        assertEquals(2, s.getRails().size());
        s.addRail(r3);
        assertEquals(3, s.getRails().size());
    }

    public void testSwitches() {
        Switch s1 = new Switch();
        Switch s2 = new Switch();
        Switch s3 = new Switch();
        s.addSwitch(s1);
        assertEquals(s1, s.getSwitches().get(0));
        s.addSwitch(s2);
    }
}
public void testSensors(){
    Sensor s1 = new Sensor();
    Sensor s2 = new Sensor();
    Sensor s3 = new Sensor();
    s.addSensor(s1);
    assertEquals(s1, s.getSensors().get(0));
    s.addSensor(s2);
    assertEquals(2, s.getSensors().size());
    s.addSensor(s3);
    assertEquals(3, s.getSensors().size());
}

public void testRailValue(){
    Rail r = new Rail();
    r.setNum(0);
    s.addRail(r);
    assertEquals(0, s.getRailValue(r));
    s.setRailValue(r, 5, true, false);
    assertEquals(5, s.getRailValue(r));
    s.setRailValue(r, 5, false, true);
    assertEquals(5, s.getRailValue(r));
    s.setRailValue(r, 5, true, true);
    assertEquals(5, s.getRailValue(r));
    s.setRailValue(r, 5, false, false);
    assertEquals(5, s.getRailValue(r));
}

public void testSwitchValue(){
    Switch sw = new Switch();
    s.addSwitch(sw);
    assertEquals(Station.STRAIGHT, s.getSwitchValue(sw));
    s.setSwitchValue(sw, Station.TURN);
    assertEquals(Station.TURN, s.getSwitchValue(sw));
}
import junit.framework.TestCase;

public class SwitchTest extends TestCase {

    Switch s;
    Station st;
    Rail r;
    BuilderDecoder b;
    Switch.direction dr;

    protected void setUp() throws Exception {
        super.setUp();
        s = new Switch();
        st = new Station();
        r = new Rail();
        b = new BuilderDecoder();
    }

    public void testConstructor() {
        Station st1 = new Station("3rd Street", b, 15, 5, 0, 0);
        Rail r1 = new Rail();
        Rail r2 = new Rail();
        Rail.railPosition rp1 = Rail.railPosition.RAIL_LEFT;
        Rail.railPosition rp2 = Rail.railPosition.RAIL_RIGHT;
        dr = Switch.direction.RIGHT;
        s = new Switch(r1, r2, rp1, rp2, st1, 1, dr, false);
        assertEquals(r1, s.getLeftConnect());
        assertEquals(r2, s.getRightConnect());
        assertEquals(rp1, s.getLeftPosition());
        assertEquals(rp2, s.getRightPosition());
        assertEquals(st1, s.getStation());
        assertEquals(1, s.getNum());
        assertEquals(dr, s.getDir());
        assertEquals(false, s.getPolarity());
    }

    public void testSetPolarity() {
        s.setPolarity(true);
        assertEquals(true, s.getPolarity());
    }
}
public void testSetLeftConnect() {
    Rail r1 = new Rail();
    s.setLeftConnect(r1);
    assertEquals(r1,s.getLeftConnect());
}

public void testSetRightConnect() {
    Rail r2 = new Rail();
    s.setRightConnect(r2);
    assertEquals(r2,s.getRightConnect());
}

public void testSetLeftPosition() {
    Rail.railPosition rp3 = Rail.railPosition.RAIL_MIDDLE;
    s.setLeftPosition(rp3);
    assertEquals(rp3,s.getLeftPosition());
}

public void testSetRightPosition() {
    Rail.railPosition rp4 = Rail.railPosition.RAIL_RIGHT;
    s.setRightPosition(rp4);
    assertEquals(rp4,s.getRightPosition());
}

public void testSetStation() {
    Switch s1 = new Switch();
    Switch s2 = new Switch();
    Station st1 = new Station("3rd Street",b,15,5,0,0);
    Station st2 = new Station();
    s1.setStation(st1);
    s2.setStation(st2);
    assertEquals(st1,s1.getStation());
    assertEquals(st2,s2.getStation());
    assertNotSame(s1,s2);
}

public void testSetNum() {
    s.setNum(3);
    assertEquals(3,s.getNum());
}
public void testSetDir() {
    dr = Switch.direction.LEFT;
    s.setDir(dr);
    assertEquals(dr, s.getDir());
}

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;

public class Test1 {

    public Test1() {
    }

    public void testPackets(ControllerMain cm) {
        List<Station> stations = cm.getStations();
        stations.get(0).setRailValue(stations.get(0).getRails().get(0),
                12, true, false);
        stations.get(1).setRailValue(stations.get(1).getRails().get(1),
                4, false, false);
        stations.get(2).setRailValue(stations.get(2).getRails().get(3),
                4, false, true);
        stations.get(3).setRailValue(stations.get(3).getRails().get(4),
                4, true, true);
        stations.get(4).setSwitchValue(stations.get(4).getSwitches().get(0),
                Station.TURN);
    }

    public void testCRC() {
        try {
            BuilderDecoder bd = new BuilderDecoder();
            BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
            while (true) {
                String str = reader.readLine();
                char[] c = str.toCharArray();
                byte[] b = new byte[c.length / 2];

                if (c.length == 22) {
                    for (int i = 0; i < c.length; i += 2) {
                        int j = Integer.parseInt(str.substring(i, i+2), 16);
                        if (j > 128) {
                            j -= 256;
                        }
                    }
                }
        }
    }
}
b[i/2] = Byte.parseByte(Integer.toString(j));
}
byte crc = bd.generateCRC(b, 10, true);
if (crc < 0) {
    System.out.println("CRC: " + Integer.toHexString(crc + 256));
} else {
    System.out.println("CRC: " + Integer.toHexString(crc));
}
else if (c.length == 4) {
    for (int i = 0; i < c.length; i+= 2) {
        int j = Integer.parseInt(str.substring(i, i+2), 16);
        if (j > 128) {
            j -= 256;
        }
        b[i/2] = Byte.parseByte(Integer.toString(j));
    }
    byte crc = bd.generateCRC(b, 2, false);
    if (crc < 0) {
        System.out.println("CRC: " + Integer.toHexString(crc + 256));
    } else {
        System.out.println("CRC: " + Integer.toHexString(crc));
    }
}
else if (c.length == 2) {
    for (int i = 0; i < c.length; i+= 2) {
        int j = Integer.parseInt(str.substring(i, i+2), 16);
        if (j > 128) {
            j -= 256;
        }
        b[i/2] = Byte.parseByte(Integer.toString(j));
    }
    byte crc = bd.generateCRC(b, 1, false);
    if (crc < 0) {
        System.out.println("CRC: " + Integer.toHexString(crc + 256));
    } else {
        System.out.println("CRC: " + Integer.toHexString(crc));
    }
}
else {
    System.out.println("invalid length");
}
}
}
}
catch (IOException e){
    System.out.println(e);
}
}

public void testSensors(ControllerMain cm){
    MaintenanceInterface mi = new MaintenanceInterface(cm);
    List<Station> stations = cm.getStations();
    boolean[] b = {false, false, false, false,
                   false, false, false, false,
                   false, false, true, false,
                   false, true, true, false};
    cm.sensorEvent(0, b);
    mi.sensorEvent(0, b);
    boolean[] b1 = {false, false, false, false,
                    false, false, false, false,
                    false, false, false, true,
                    true, false, false, true};
    long time = System.currentTimeMillis();
    while(System.currentTimeMillis() < time + 5000){
        cm.sensorEvent(0, b1);
        mi.sensorEvent(0, b1);
        cm.sensorEvent(1, b1);
        mi.sensorEvent(1, b1);
    }
}

public static void main(String[] args) {
    new Test1().testPackets(new ControllerMain());
    // new Test1().testCRC();
    // new Test1().testSensors(new ControllerMain());
}
Caps on this page are bypass for the corresponding chips

LAFAYETTE Lafayette College Easton, PA ECE Department

TITLE RAIL_PICS

SIZE

AUTHOR

JEFF LETOSKI

COURSE ECE492

SHEET 1 / 1

3-9-2008 21:17
This part requires a 24 pin socket.
Red, Black, and Green refer to the switch connection colors.