**Master Test Plan (MTP) for Hardware Track Controller**

**Overview**

The whole sub-system of the Hardware Track Controller can be separated into two components: the hardware component and the software component. The hardware component is implemented using Arduino, so combining all the method functions and significant member variables defined in the Arduino code, we can define “Arduino” as one class. To make Arduino communicate with the rest of the sub-system, a linking class called “SerialPort” is necessary to establish such serial communication. One end of the SerialPort class is connected to the Arduino and the other end of the SerialPort class is connected to the software component, which includes classes of “Wayside”, “HWTrackController”, and “PLC”. To ensure a working sub-system, each class will be tested separately as described below.

**Approach**

Each class will be tested separately in a designed order, and the approach to test the hardware component is different from the approach to test the software component by their nature.

The first class that will be tested is Arduino. Because the inputs of this class are entered by controlling physical buttons and joystick, I will test it by controlling those physical parts. Then I will look at the result from the serial monitor, the LEDs, and the LCD display to check if the class works properly. The specific test cases are included in the next section.

The second class that will be tested is SerialPort. Because SerialPort is handling the communication between the Arduino and the rest, and the Arduino class has been tested, I will also test it by controlling the physical buttons and joystick. I will make an instance of the SerialPort class in the test\_main file and print out the result to the terminal or the LCD display, and then check if the result looks correct.

The approach to test each of the rest of the classes will be the same. Instances of them will be created in the test\_main file. I will hardcode input to the instances and print out the output to the terminal, and then check if the result looks correct. Based on the relationships between classes, the best order to test them will be Wayside, PLC, and HWTrackController.

**Test Cases for each Class**

**Arduino:**

**Test case 1: Switch Track**

Input: Programmer toggles the track switch and presses the update button

Expected Output: the corresponding bit printed to the serial monitor is correct

**Test case 2: Activate Railway Crossing**

Input: Programmer toggles the crossing switch and presses the update button

Expected Output: the corresponding bit printed to the serial monitor is correct

**Test case 3: Select Block**

Input: Programmer controls the joystick to select block

Expected Output: the correct block ID shown to the LCD display

**SerialPort:**

**Test case 4: Connect Serial Port**

Input: Arduino connect to one of the ports on computer

Expected Output: Boolean value of true printed to the terminal

**Test case 5: Read Serial Data**

Input: Programmer presses the update button on physical UI

Expected Output: the correct bits printed to the terminal

**Test case 6: Write Serial Data**

Input: string of input data to Arduino

Expected Output: data shown on the LCD display

**PLC:**

**Test case 7: Read PLC File**

Input: plc file

Expected Output: valid file -> true; invalid file -> false

**Test case 8: Run PLC Program**

Input: plc file

Expected Output: correct action done to the system

**Wayside:**

**Test case 9: Initialize Wayside**

Input: vector of block pointers

Expected Output: member variables such as suggested speed and the vector of switch positions should be initialized, and the values should print to the terminal

**Test case 10: Encode**

Input: suggested speed, authority, block ID, and more as needed

Expected Output: a string of input ready to be sent to Arduino

**Test case 11: Decode**

Input: a string of output from Arduino

Expected Output: block ID, track position, railway crossing state, and more

**Test case 12: Set speed & Authority**

Input: suggested speed and authority

Expected Output: update to the corresponding member variables and print to the terminal

**Test case 13: Calculate Commanded Speed**

Input: suggested speed

Expected Output: command speed print to the terminal

**Test case 14: Get speed & Authority**

Input: none

Expected Output: commanded speed and authority printed to the terminal

**Test case 15: Update Wayside**

Input: this method will call the decode method and then call methods to set speed, authority, track positions, and more

Expected Output: all member variables updated and printed to the terminal

**Test case 16: Set/Get Occupancy and Maintenance Mode**

Input: occupancy and maintenance mode

Expected Output: update to the corresponding member variables and print to the terminal

**HWTrackController:**

**Test case 17: Create Wayside**

Input: vector of block pointers

Expected Output: a wayside instance is created and initialized

**Test case 18: Get Wayside Pointer**

Input: none

Expected Output: wayside pointer pointing to the only wayside

**Test case 19: Set Suggested Speed & Authority**

Input: suggested speed and authority

Expected Output: suggested speed and authority have been updated in the wayside instance

**Test case 20: Get Commanded Speed & Authority**

Input: none

Expected Output: commanded speed and authority in the wayside instance printed to the terminal

**Test case 21: Import PLC File**

Input: plc file

Expected Output: plc file has been imported

**Template for Tests**

Templates for each test will contain the outcome of the test (Pass or Fail), the specific outcome expected versus what happened when the test was performed and which user requested it. It will look like the following:

| **Test Case** | **Inputs** | **Expected Output** | **Pass/Fail** | **Failure Description** | **Tester** | **Data Tested** |
| --- | --- | --- | --- | --- | --- | --- |
| **Arduino** | | | | | | |
| Switch track | Toggle the switch and press the update button physically | LED indicator turns on/off  Track position changed in vector<bool> blockSwitchPosition | Pass | None | Justin | 12/12 |
| Activate crossing | Toggle the switch and press the update button physically | LED indicator turns on/off  Track position changed in vector<bool> blockCrossingState | Pass | None | Justin | 12/13 |
| Send id to arduino | Int id of block | Id shown on LCD display | pass | None | Justin | 12/12 |
| **UI Functionality** | | | | | | |
| Select block with id | Corresponding block clicked in the list | Int id set to the block id | pass | None | Justin | 12/12 |
| Hardware connection | Connect arduino | Hardware connection box checked | pass | None | Justin | 12/12 |
| Toggle Mode | Toggle mode button clicked | Check box state toggle everytime button clicked | pass | None | Justin | 12/12 |
| Train present | Block occuupancy info | Check box state changed accordingly | pass | None | Jutsin | 12/13 |
| Crossing state | Crossing state info | Check box state changed accordingly | pass | None | Justin | 12/13 |
| Switch state | Switch state info | Check box state changed accordingly | pass | None | Justin | 12/13 |
| **Vital check with PLC** | | | | | | |
| Read PLC file | Plc file | Return bool = true | pass | None | Jutsin | 12/13 |
| Check if track switch is allowed | Train present set to 1/0  Switch track on physical UI | If train present == 1  Switch failed  If train present ==0  Switch succeeded | pass | None | Justin | 12/13 |
| Switch track | Block info & speed & authority | Track switched accordingly | pass | None | Justin | 12/14 |
| Crossng state | Block info & speed & authority | Crossing state changed accordingly | pass | None | Justin | 12/14 |