Merley Arrivals Documentation:

Usage and Technical Information

By Chris Field

## Overview

The arrivals area signalling system provides train detection through axle counters, completes the absolute block protection for the passenger running area of the railway and implements several driver-operated controls to provide signalled shunt moves.

It it consists of:

* gantry signals 45, 46 & 47
* platform home signal 48
* shunt signals 42, 43 & 44
* points 100 and 300
* axle counters
* plungers and buttons
* power supply (in cabinet under bridge)
* central control unit (in cabinet under bridge)
* cables and connections

The purpose of the system is to control movements of trains from the main line and yard into platforms 2, 3 & 4, as well as permitting shunt moves to set back from each platform as required, either to another platform or to return to the yard.

No signals are set to proceed by default, the presence of a train operating an axle counter, or operation of a button or plunger causes a route to be set, setting points and signals as required.

## Usage and Operation

Normal platform arrival

In normal running use, the system will detect a train just after passing the level crossing, and set a route into a platform by setting 47 and 48 signals as required. In its initial form, the route set will follow the previous train. The signals will be reset to danger as the train operates the axle counters and passes through the sections.

The driver can select a different platform on approach to 48 signal by operating the relevant plunger (identically to existing system).

Shunt from platform

Should a train need to shunt back from a platform, the driver must press the relevant button, located adjacent to platform 4. If the route is immediately available, the points will be set and the relevant shunt signal will clear to permit the train to set back past 48 signal. If the route is not available, due to an approaching train, the request is stored and no further train will be routed past 47 signal. The button must be pressed again, once the approaching train has passed 100 and 300 points and the route is clear. The points and signals will be set to permit the shunt move.

Once past 48 signal, the driver can continue to the yard if required, or stop and select an alternative platform to proceed to, using the selection buttons between the footbridge and 48 signal, which will set the points and signal as required. Once clear of 48 signal, any train waiting at 47 signal will be signalled into the platform to follow.

Exit from yard

To exit the yard, the driver should press the button on the relevant signal and await a proceed aspect. It will be necessary to press a plunger on the approach to 48 signal to select a platform.

Special functions

The buttons positioned between the footbridge and 48 signal are the “master” controls. These will override any other route set.

There is a button on 47 signal. This resets the axle counters for following section. Should a driver be waiting at 47 signal, while no shunt movement is taking place, this buttons should be operated to clear the section and provide a proceed aspect (in the case of a “mis-count” in the axle counter system).

## Technical Introduction

The system has a single interlocking processor, connected to a number of slave devices via a network connection. See system architecture diagram.

The interlocking processor is located in the cabinet under the footbridge, alongside a 24V power supply which supplies power to the slave devices via the network cable and through additional power supply cables.

## Detailed Technical Information

### Core System Processor

The core system software is written in python and is publicly accessible at https://github.com/chrismfield/signalling\_project

There are lots of ways to run python software; it could be on a PC or on a raspberry pi. For our application, it is running on low power "thin client" PC - this was chosen because it was cheaply available (£30 from ebay) and provides a suitable enclosure and power supply, and has no moving parts (fanless) so should be maintenance free. The system runs linux and auto-starts the python signalling software.

The PC is configured to turn on as soon as power is applied, and to run the interlocking software, MQTT server and Webserver on boot.

### Network

The network is an RS485 serial network implementing Modbus as its protocol. It uses standard CAT5 network cable, but IS NOT an ethernet network (don't try to connect it!). There is lots of information on the internet on RS485 and Modbus which I will not replicate here. Worth noting that it is a daisy-chained network, so a break in the network cable will affect all components down-stream of the break. The benefit of this is that there is only one cable (plus supplementary power where required) running the length of the layout.

The interface between the RS485 and the PC is made through a simple and cheap USB adaptor.

The RS485 network just needs two wires plus earth, The remaining wires in the CAT5 cable are used for power. Each core can support 1A of current and there are 3 cores parralleled to supply power. We can get just about all the current we need through the network cable, but there is some volt drop and are close to the current limit when point motors are operating at the same time as high-brightness LED signals, so additional power cables supplement this as required.

The wires are used as follows:

|  |  |
| --- | --- |
| Data A | Green&White |
| Data B | Green |
| Power Positive (+24V) | Orange&White (Primary)  Brown&White  Blue&White |
| Power Negative (0V) | Orange (Primary)  Brown  Blue |

### Slave Modules:

#### Axle counters

The axle counters work on by detecting the flanges as they pass three hall effect (magnetic) sensors associated with three small neodymium magnets fitted into the body of the axle counter. The axle counter counts an axle every time the middle sensor detects a magnetic change over a set threshold level, and then an end sensor detects and then returns to a normal level (to detect direction).

The normal level is determined at power-up; if the magnets have accumulated ferrous debris, clean them up and turn the whole system off and on.

The axle counters have a microchip PIC processor controlling them, programmed in PIC basic [ADD PIC CODE TO GITHUB]. The RS485 network is interfaced through a MAX485 network driver chip. Power is taken from the orange power wires and regulated down to 5V. Connections are provided from the Brown and Blue wire pairs but they are not connected to anything on the board.

#### Plungers

The buttons and plungers are connected to a slave board which is controlled and interfaced in a similar way as the axle counters. Each input has a "pull-up" resistor, so the when the plunger is operated, it pulls connects the input to ground to pull it to 0V.

#### Signals and Points

The signal and points output boards are controlled with a raspberry pi pico board, soldered onto the board with castellated edge connections. The pico is programmed in python and the code is also on github.

Each board has 8 outputs, which can be used independently (for signals) or in pairs (for reversing points). The boards need to be configured to their specific use to enable the correct functionality. This is done by adapting the config.json file - the "board index" needs to be configured such that each point or signal uses a separate board index (this allows the on-board logic to work correctly). There is also a set of dip switches on the board which configure whether the outputs are used independently or in pairs. Note that for signals, it is a common anode system, meaning the each input is switching the negative connection of the signal.

The boards also have a some digital and analogue inputs which are intended to be used for plungers, point detection or any other local sensor input required.

If being used for points, the board reports the detection status to the system. Detection can be set to be based on inputs (e.g. Microswitches), current (only for the first pair of outputs; detection based on 0 current detection following movement, for point actuators with built-in limit switches) or as a simple timer providing detection following command to change the point direction.

When being used to control signals, the board must receive an aspect command for each signal at least every 8 seconds, otherwise comms are assumed to have been lost and the signal will revert to danger.

Power is taken from the network cable, however for higher current applications (anything that can go over 1A), power should be supplied through the auxiliary power supply connection at 24V to prevent overloading the network cable.

### Software Logic

The python software is written to be generic, with all infrastructure track/layout elements being configured in a seperate file (default.json, as set in config.json). The layout configuration can be edited using the infrastructure editor tool, or by editing the json file directly (this requires a knowledge of the structure and rules of json files!). The elements of the infrastructure are :

• Sections

• Points

• Signals

• Axle counters

• Track circuits (not yet implemented)

• Plungers

• Signal levers (not yet implemented)

These elements are linked together to provide the train protection required.

Routes are configured to be set between sections, with signals and points set as required.

Triggers cause routes to be set (if not already in use).

There is a priority assigned to each trigger, which determines the order in which the triggers are processed.

### Configuration

Beyond the standard configuration, it is useful to understand that the modbus protocol requires each "slave" or "server" to have an address between 1 and 128. Each slave then has numbered "coils" (outputs) and inputs which can be set or read as applicable. This is the mechanism by which the signals and points are set, and the plungers, axle counters and detection are queried. Standard assignments of coils and inputs are used for each type of slave, but it should be noted that the "board index" allocates different registers to allow multiple signals and points to share one slave address.

### Web interface

In order to see the status of the various elements, and be able to manually set them when required, an interface is required. This is done primarily by implemented communications with the interlocking processor using an MQTT server. The server is also hosted on the same PC as is running the interlocking software. It is possible to use an MQTT client (like MQTT explorer) to login to the MQTT server on port 1885 and see all the reporting being sent out. Manual setting can be undertaken by setting using the following message formats:

• set/point/[point\_ref] : [normal/reverse]

• set/signal/[signal\_ref] : [aspect]

• set/route/[route\_ref] : [True/False]

• set/trigger/[trigger\_ref] : True

• Set/section/[section\_ref]/occstatus : [axle\_count]

In order to simplify this interface, a webpage has been created, which is served by a webserver also on the interlocking PC, which connects to the MQTT server, displays relevant status information and has buttons which send the relevant MQTT commands to the server, which in turn sends them onto the interlocking processor.

The PC has a wifi connection to a hub/access point which is labelled with connection details.

### Diagnostics

If the system is not working correctly, the following fault finding may help:

* Open cabinet and check PC is on
* Look at the USB/485 interface board and check the TX and RX leds are indicating comms continuously.
* Check the 24V power supply is providing power.

If several slaves are not responding, a wiring defect may be the cause; it should be possible to identify the location of the fault from the schematic. Reviewing the web page and MQTT messages will also provide an indication of where comms have been lost.

You can also give me a call on 07790 156386.