Arduino-based block occupancy and train direction code structure

Jerry Grochow

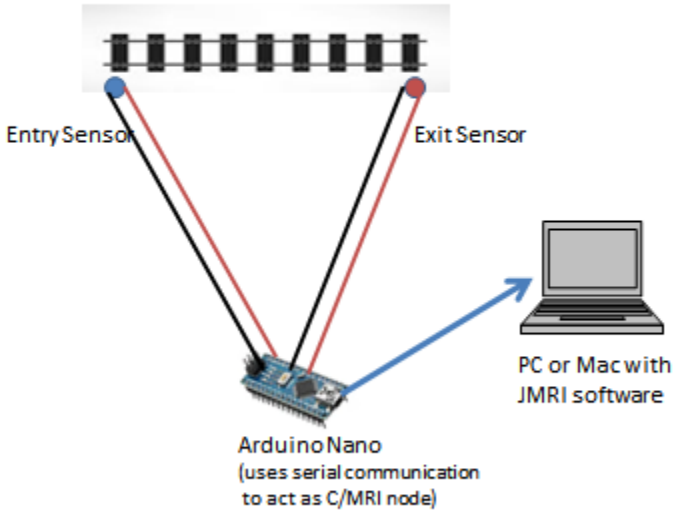
Designing a block monitoring system to accurately report block occupancy and train direction in all circumstances isn't as easy as it might seem. There are several implementation approaches to try, each with advantages and disadvantages. And a number of questions related to anticipated train movement have to be resolved to get it working right.

Defining the problem

In my case, I wanted to monitor several sections of track, not necessarily contiguous, to determine whether a train is occupying each monitored block and, if so, the direction it is moving. The monitoring system had to be inexpensive to assemble and able to be installed on my existing layout one section at a time (no rebuilding from scratch or taking the whole layout out of commission for months!).

Using "point" sensors (infrared (IR) detectors, photoresistors, Hall Effect sensors, RFID readers, and the like – any device and circuit that will trigger based on a train passing by), I put a sensor at the entry and exit of each section of track I wanted to monitor (see Figure 1). I wrote an Arduino sketch (program) to determine train direction and occupancy based on the state of the two sensors on each block.

FIGURE 1: BLOCK MONITORING SYSTEM SCHEMATIC



Of necessity, my sensors were going to be placed on blocks of varying lengths so that sometimes trains would be occupying a block but not triggering either one. Furthermore, there was the "powerup" situation where a train could be triggering none, one or both sensors.

There are also some oddball (i.e. not "prototypical") circumstances that might be interesting to deal with someday, but again, I didn't want to complicate my environment for situations that rarely occur. Some examples:

- What if a long train reverses direction within a block before triggering the second sensor? Or reverses direction when it is still activating both sensors?
- What if a short train stops within a block (when neither sensor is activated) and changes direction?
- What if a second train enters the block before the first train exits (traveling in the same direction)?
- And, of course, what if two trains enter the block going in opposite directions (as frequently seems to happen when my grandchildren are helping me operate the railroad!)?

What I eventually found is that there are a few dozen situations of train movement that complicate the use of just an entry/exit sensor pair to monitor a block. The rest of this article details how I worked through (or around) most of them to get the system I wanted for the price I wanted.

Train detection sensors

My monitoring system uses IR sensors connected to an Arduino as in Figure 1, although any type of sensor can be used. I use the JMRI (Java Model Railroad Interface) application suite to display this information on a control panel and a layout panel (see Figures 2 and 3) and can also use it to direct automatic train movement.

FIGURE 2: JMRI CONTROL PANEL

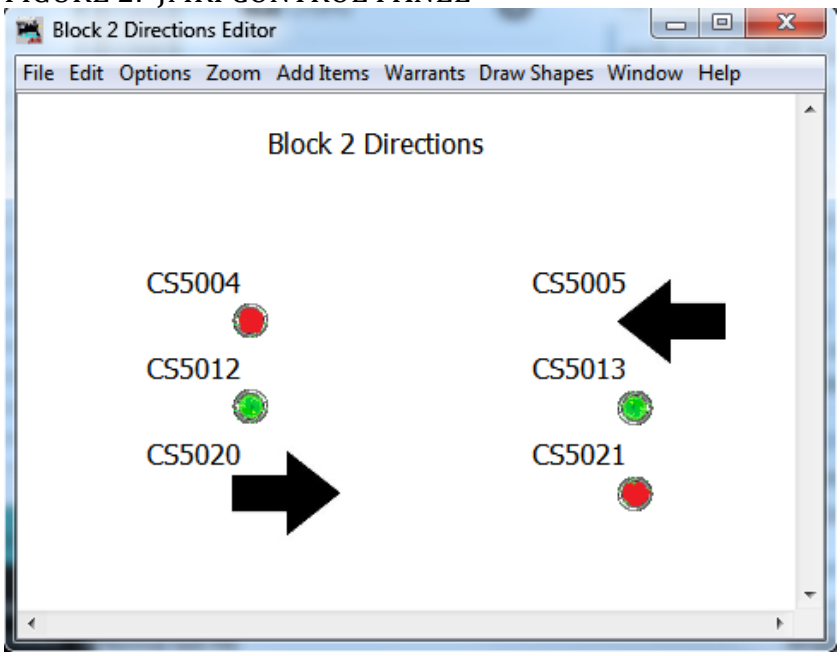
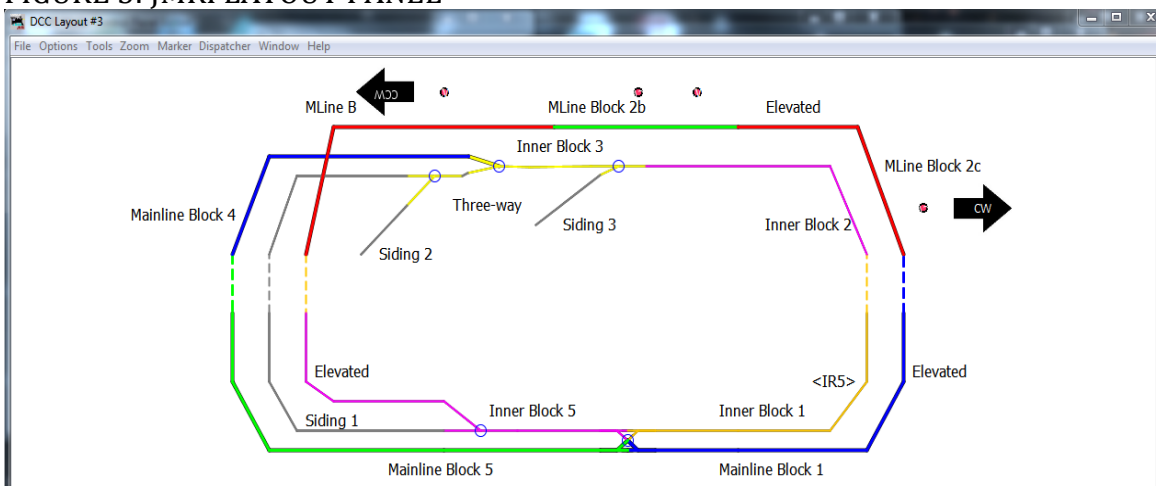


FIGURE 3: JMRI LAYOUT PANEL



Processing data from different types of sensors

I purchased several different types of sensors to experiment with and implemented Arduino code to handle them all. The key difference is whether the connected Arduino pin goes HIGH (5v) or LOW (GND) when the sensor is activated. I created a variable and added some code to handle this difference. For an IR "reflector" module with a digital output installed (active when the IR beam is reflected off a train back to the IR receiver), the sensor output pin goes LOW when active. This detector can also be modified and installed in "break-beam" mode (IR transmitter and receiver positioned across from each other so the IR beam will be interrupted by a passing train). In this case, the output pin is LOW until the beam is broken when it goes HIGH. If an analog IR detector is being used (where the receiver is attached to one of the Arduino's analog pins and the voltage indicates the strength of the beam), the signals are reversed and LOW means active in break-beam mode and HIGH means active in reflector mode. For typical photo-resistor modules, LOW means active.

Designing the Arduino sketch

The key to creating a robust train movement monitoring system is the Arduino program (or "sketch" as it is called in the Arduino community). I started with a few simple cases and then added more as I expanded the program. This iterative approach is typical of the way large computer programs are developed and makes both the design and programming (and debugging!) easier. The program code is available for you to try out or modify via Github (a free online service for sharing programs and documentation) at https://github.com/jerryg2003/MRR-arduino . Other than powerup situations, the code can figure out what is going on in all prototypical situations.

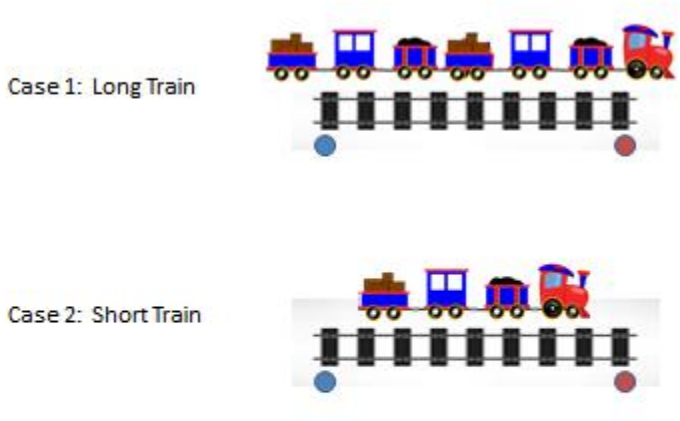
The two cases I started with are diagrammed in Figure 4:

1. All trains are at least as long as the monitored block (they have to trigger at least one sensor to be occupying the block).

2. Some trains may be shorter than the monitored block (they can occupy the block without triggering either sensor).

My analysis showed that looking only at whether the one or both sensors are currently active would not be sufficient to decide which way the train is moving. I needed more data to figure this out. [Geoff Bunza's system leaves it up to a program running on your PC to determine train movement. My goal was to figure that out directly on the Arduino.]

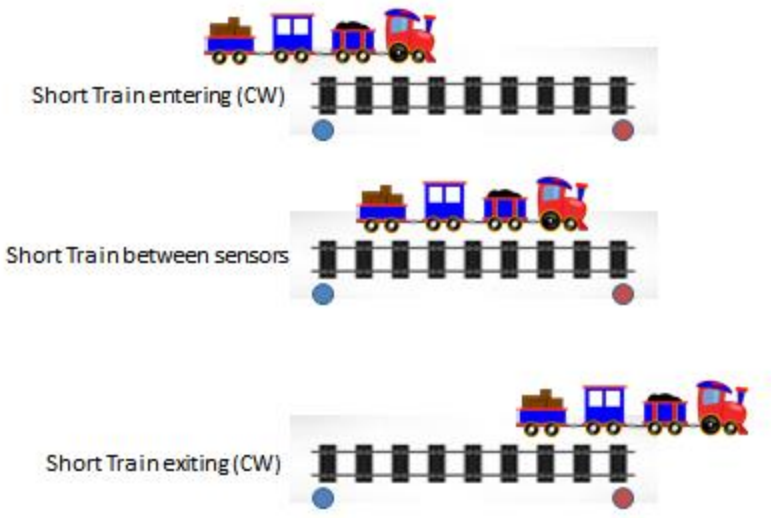
FIGURE 4: TRAIN MOVEMENT PAST SENSORS



For example, if only the left sensor in the diagram (which I also call the entry sensor) is active, the train could have come in from the left (going clockwise) or it could be a short train that is traversing the block from the right (counter clockwise) and has already passed by the right sensor (also called the exit sensor) leaving it inactive. Adding information about which sensor was triggered first, however, makes it possible to decide direction of travel in many of the possible cases. If the left sensor is triggered first (on an empty block), then the train is moving clockwise. Similarly, if the right sensor is triggered first, the train is moving counterclockwise. If both sensors are activated by a long train, knowing which was triggered first still allows you to tell the direction of the train. So I clearly needed to keep track not only of which sensors were active or inactive, but also the order is which they became active or inactive, that is, what they were doing in the past.

The short-train case can be even more complicated (Figure 5) because it is possible for the train to be moving between the two sensors (with neither activated) but the block still occupied. Again, you need more information, such as what the status of the sensors were in the past, or what direction the train was traveling in the past, or whether the block was occupied in the past. The question for me was what information was really needed to handle all of the expected situations.

FIGURE 5: Short Train Movement



[At this point, some people will want to skip right to copying the finished program and start running some trains. For others, however, I thought it might be interesting to discuss the way in which this type of analysis is accomplished. Read on at your own risk!]

To understand the Arduino sketch, it helps to understand some basic terminology. The Arduino sketch takes in information ("inputs") and computes other information ("outputs"). The current and previous status of each of the two sensors are the four inputs that I will use to determine current block occupancy and current direction of travel as outputs. The inputs tell the status or "state" of a block at a particular point in time. These inputs are called "state variables," and if they have been correctly identified, contain all of the information needed to determine the outputs in all situations. Using only two state variables (the current status of the entry and exit sensors) is not sufficient to determine the outputs in all situations. I have now added two more, the immediately prior status of the sensors, to see if using four state variables will be sufficient to complete the picture.

Each state variable has two possible values: each sensor can either be active or inactive (for simplification, ignoring the case where we don't know the state of the sensor). Four variables, each with two possible values means there are sixteen possible combinations or "system states" (two raised to the fourth power equals 16 – you can list them for yourself, or you can refer to the list at the end of this article). I refer to each system state by abbreviations for the status of each sensor, present and prior. "AIAI" means that the entry sensor is currently active, the exit sensor is inactive, the entry sensor was active in the previous time we checked (in my case, a second ago), and the exit sensor was previously inactive at that time. Of the sixteen

possible system states, one does not make much sense on my model railroad: IIAA (both sensors currently inactive, but both active a second ago). Unless you are running Acelas at scale speed approaching 150 MPH, your train wouldn't be able to occupy the block triggering both sensors simultaneously in one second and pass out in the next second. Determining block occupancy and direction of travel for the other 15 system states will be a matter of working out logically what the sensor states are telling us.

FIGURE 6: SHORT TRAIN POSITION DIAGRAM

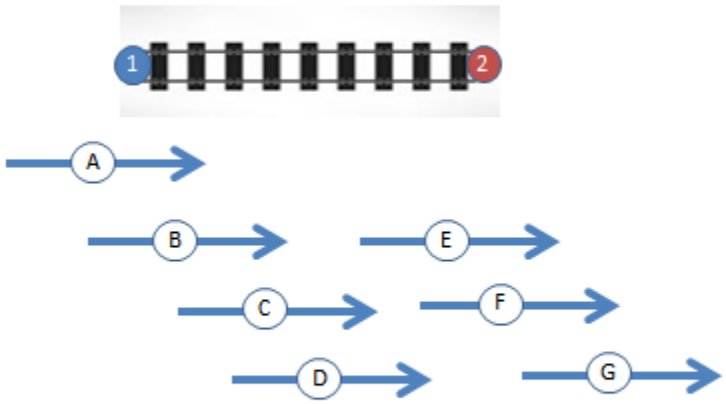
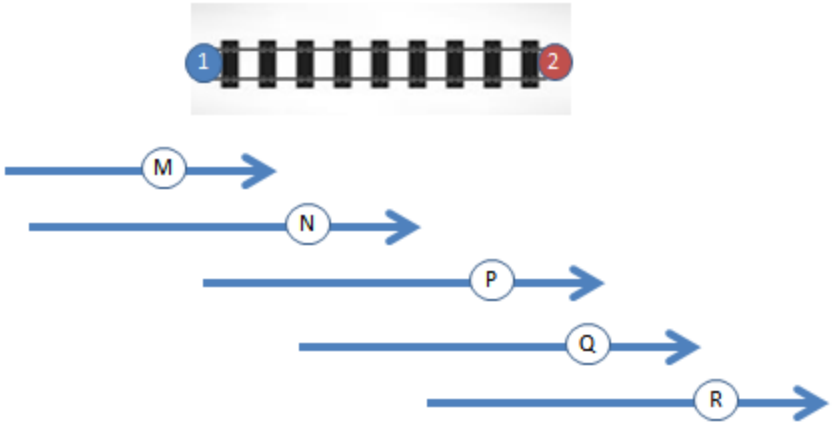


FIGURE 7: LONG TRAIN POSITION DIAGRAM



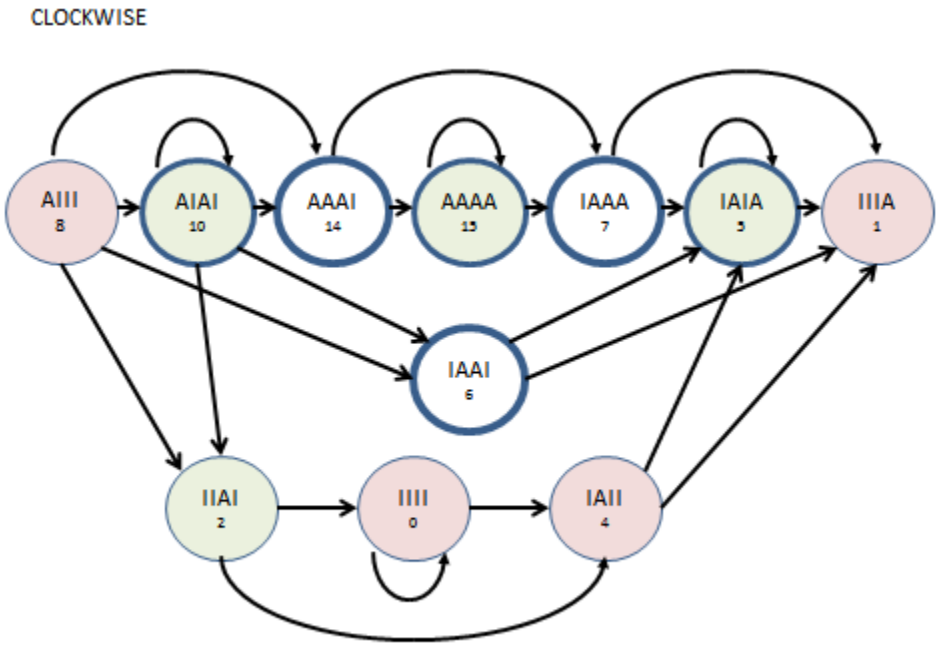
I mapped out all the states the sensors could be reporting by referring to a diagram showing possible train positions with respect to the block. In Figure 6, each arrow represents where a short train moving clockwise is at a particular point in time when the status of the sensors is captured. Figure 7 shows a similar diagram for a long train, where the exit sensor is activated before the entry sensor goes inactive. [Similar diagrams were also prepared for trains moving counterclockwise.] For each position, here is what the sensors will show. Note that P and Q could represent two different situations depending on how long a particular train is and how fast it is moving:

State ID	Current Sensor 1	Current Sensor 2	Prev Sensor 1	Prev Sensor 2	State Abbrev	Binary Equiv
A	Active	Inactive	Inactive	Inactive	AIII	8
B	Active	Inactive	Active	Inactive	AIAI	10
C	Inactive	Inactive	Active	Inactive	IIAI	2
D	Inactive	Inactive	Inactive	Inactive	IIII	0
E	Inactive	Active	Inactive	Inactive	IAII	4
F	Inactive	Active	Inactive	Active	IAIA	5
G	Inactive	Inactive	Inactive	Active	IIIA	1

State ID	Current Sensor 1	Current Sensor 2	Prev Sensor 1	Prev Sensor 2	State Abbrev	Binary Equiv
M	Active	Inactive	Inactive	Active	AIII	8
N	Active	Inactive	Active	Inactive	AAII	10
P	Active Active	Active Active	Active Active	Inactive Active	AAAI AAAA	14 15
Q	Inactive Inactive	Active Active	Active Active	Inactive Active	IAAI IAAA	6 7
R	Inactive	Active	Inactive	Active	IAIA	5

Figure 9 is another way of capturing the information in Figures 7 and 8 and the table. It is called a "state transition diagram" and it shows all the possible states that the sensors will report with a train moving in a clockwise direction. It also shows via the arrows how the train might progress from one state to another. In several cases, states might be skipped (for a short or fast moving train, for example), or the same state might be reported over and over (for a long or slow moving train).

FIGURE 9: CLOCKWISE TRAIN STATE TRANSITION DIAGRAM



I found that a train going clockwise can end up reporting 11 of the 15 different sensor states! Similarly, a train going counterclockwise can be in 11 of the 15 states, although some different ones (this is left as an exercise for the reader; the diagram can be found in Github). In fact, there are only three states in each direction that are

unique to that direction (in Figure 9, these circles are unshaded and have a bold border). There are three more states (AIAI, AAAA, and IAIA, colored in green, bold border) that are possible with the train moving in either direction. In these cases, determining train direction requires more information. If a train is in state AAAA (as when a long train covers both sensors for at least a second), it must be moving clockwise if it was previously moving clockwise, and vice versa for counterclockwise. In addition, if the sensors are in state IIIA on this diagram (and correspondingly, state IIAI on the counterclockwise diagram), the train direction is the previously reported direction. This means there is a fifth state variable, previously reported train direction, that is required to determine train direction if the sensors are in one of these four sensor states.

That still leaves the states shown in red – four more states where direction cannot be uniquely determined by referring even to five state variables. For one of these, IIII, knowing whether the block was previously occupied (a sixth state variable) will allow us to determine direction. Similarly, for sensors showing states IIAI or IIIA (nothing currently active, but either the entry or exit sensor previously active), then if the block was previously occupied, direction is as previously reported (note that going clockwise, state IIIA means the train has left the block so direction is no longer of interest; going counterclockwise, however, this state could appear right after state IAII).

There is a seventh state variable that will allow us to determine a few more situations and that is whether this is the first sensor reading after powerup or not. In the case of first reading after powerup, unfortunately, we have several ambiguous situations and there is not really anything we can do to resolve them. If power has been on a while and this block has already been traversed, then we know that hitting state AIII (entry sensor active) means the train is moving clockwise. However, if state AIII is found before the block has been traversed and cleared, we don't know if that is due to a train entering the block clockwise or a short train (that had been sitting between the two sensors) leaving the block counterclockwise. [This is the one situation where having a block occupancy detector (BOD) would allow you to determine direction of travel. The BOD would be active when power was turned on and the sensor triggered first will tell you which way the train is moving. If a long train is triggering both sensors, then having the BOD doesn't add any information as you already know the block is occupied. Having a BOD would also help in distinguishing certain change-of-direction scenarios.]

Luckily, we don't have to deal with all 128 possible combinations of seven variables as some are used only in a few situations. But even with seven state variables, you still can't determine direction of travel in every single situation. I could add that BOD, but it seems like it is time to get back to running trains. And about all those of situations of trains changing direction, or two trains entering the same block, well, there are several solutions that come to mind, including isolating those blocks and adding BODs everywhere. I'm pretty sure I can make this work, but I don't expect to get around to it for quite some time. Another solution is to add more sensors so that

you can use the results of multiple sensors to determine direction of travel. That too involves a fair amount of work, and, depending on the length of your trains, will still result in some cases where you end up with the original problem.

Which leads me to another "solution": waiting it out. Eventually, the block will be empty. So set a timer, perhaps 10 seconds, and if nothing changes during that time, just clear everything to the inactive, unoccupied state. Trains will start moving again and tripping sensors and your program will start to determine what direction they are moving all other again! This may not be very satisfying from a "correctness" viewpoint, but it sure works if your goal is to get back to running trains!

What this discussion should show you is that writing computer programs to account for every possible situation encountered on a model railroad is difficult work, comparable to detailing a locomotive or scratch-building a train depot for prototype realism. But you can get a lot of enjoyment taking something out of the box, maybe doing a little weathering, and adding it to your fleet. You can also use a program that has some known deficiencies, but still can add a lot to your layout, and improved whenever you get around to it.

By the way, about those two trains entering a block from opposite directions: using my monitoring system connected through JMRI, I can automatically issue a command to cut power!

Jerry Grochow retired after multiple positions to IT in consulting, finance, and education. He now mentors entrepreneurs, writes on topics that interest him, and helps introduce his grandkids to what has become a renewed passion for model railroading in the digital age.

Appendix: COMBINATIONS OF FOUR SENSOR STATE VARIABLES

Binary code equivalent	S1 - S2 – Prev S1 – Prev S2	Computed Direction	Extra Conditionals
0	IIII	=PrevDir	if PrevOcc
1	IIIA	CCW	if PrevDir = CCW
2	IIAI	CW	if PrevDir = CW
3	IIAA	-	not allowed
4	IAII	CCW/CW	if notPrevOcc / if PrevOcc*
5	IAIA	=PrevDir	
6	IAAI	CW	
7	IAAA	CW	

Binary code equivalent	S1 - S2 – Prev S1 – Prev S2	Computed Direction	Extra Conditionals
8	AIII	CW/CCW	if notPrevOcc / if PrevOcc*
9	AIIA	CCW	
10	AIAI	=PrevDir	
11	AIAA	CCW	
12	AAII	Indet.	also at powerup*
13	AAIA	CCW	
14	AAAI	CW	
15	AAAA	=PrevDir	

* These states can also occur at powerup, in which case it is not possible to say whether the train is going to move CW or CCW until another sensor is activated.