Grade Crossing Controller

Contents

Grade Crossing Controller 1

- The Project 2
- Arduino Sketch: GradeCrossing 2
  - Features 2
  - Quick Configuration 2
  - The Design 3
  - Placing Sensors 5
  - Software: Definitions 5
  - Initial Setup/Reset Process 6
  - Board Schematic 6
  - Tips, Techniques, References 6
- Specifics for the Central Vermont 7
  - South Coventry 7
  - Stafford 8
  - Monson 8

Revision History:

- 8 December 2025 Updated for version 4.0
  - Remove need for EEPROM library
  - Allow multiple center sensors
  - Simplify customization
- 7 April 2025 Updated for version 3.4 restructuring and bug fixes

Jon E. Schmidt
jontenor@gmail.com

The Project

On the Central Vermont in NorCal (www.cvrailroad.com) we have several grade crossings that need to be animated. We use the Walthers crossing flashers and the IOWA Scale Engineering sound module (https://www.iascaled.com/store/SND-XBELL) for our crossing. Since there are several trackage configurations we have decided to use an Arduino Nano as the crossing controller. This allows us flexibility for controlling approach, exit, and overall timing of the crossing signals.

This and associated documents may be found at https://github.com/joneschmidt/GradeCrossing.

Arduino Sketch: GradeCrossing

Features

The software provides:

- Asymmetrical track configurations
    - Each crossing may have one or more approach, exit, and crossing tracks
- Three sensor operation:
    - Left and right "distant" sensors to detect the approach and exit of a train
    - One or two crossing sensors at the road (per track) to confirm the train
    - Left, right, and center sensors may be "sets", 1 or more tracks and sensors for each set
- Distant sensors may be as distant or close as needed
    - Timers for distant sensor activating the crossing signal
    - Timer for train hitting a distant sensor but stopping short of the crossing
    - Timer for train exiting a distant exit sensor without triggering the flashers
- Digital or analog sensors
    - Auto-calibrating analog sensors using photoresistors or phototransistors with sensitivity control[1]
- Optional sound control
- Optional gate control
- Multiple separate crossings on a single Arduino depending on port availability

Quick Configuration

Make the top-level changes here for your crossing configuration:

```
// DEFINE THE ACTIVE CROSSINGS - DEFAULT
#define NumXings 1 // Number of grade crossings
#define NumTrks  1 // each crossing - max tracks in/out of crossing
#define NumCntr  1 // number of center sensors per track
#define NumGates 0 // 0 or 1
```

- NumXings – Normally one for the typical Nano.
- NumTrks – The maximum number of tracks in or out of a crossing. Normally one track through the crossing. Other configurations are supported, such as double track to the left of the

[1] A photo-sensitive device placed between the rails may be used for sensing. The sketch examines current versus average resistance of the device to detect an engine or train.

crossing, and single track through the crossing, and continuing single track to the right. In this case the NumTrks should be set to 2.

- NumCntr – The number of sensors on each track crossing the road. Normally 1 for a sensor on one side of the road. May be set to 2 if desired.
- NumGates – 0 or 1

```
// Define crossing (XingDef) array elements
// DON'T CHANGE
#define NumSnsrs NumTrks * (2 + NumCntr)
typedef struct {
  // input sensors analog or digital - DigType - AlgType
  int SnsrType;
  // sensors (LeftSnsr[NumTrks], RightSnsr[NumTrks], CntrSnsr[NumTrks])
  uint8_t SnsrLst[NumSnsrs];
  // WaitABit - time to delay lights after distant sensor hit
  int WaitABitSecs[NumTrks*2];
  // first crossing lamp address of two in sequence - left/right lamp flash control
  uint8_t FirstLite;
  // sound address - digital mode or empty 0
  uint8_t SoundCntl;
  // gate port - or empty 0
  uint8_t GatePort;
} XingDef;
```

- SnsrType – Sensors may be digital (DigType) or analog (AlgType) for photoresistors.
- SnsrLst – the ports to the sensors - left, right, and center, in that order.
- WaitABitSecs – the delay in seconds between the track sensor triggering and the start of the signal operating sequence. It must correspond to the SnsrLst above.
- FirstLite – the port of the first of two ports for the lamps in the signal. The second lamp port must be the FirstLite port plus 1.
- SoundCntl – the port for controlling the sound.
- GatePort – the port for controlling the crossing gates.

Make the detailed changes here for your crossing configuration:

```
// NumXings 1, NumTrks 1, NumCntr 1, NumGates 0 above to match
// Define the crossing by the following array
XingDef Crossing [] =
//  Sensor type,
{    AlgType,
//  PORTS: LeftSnsr[NumTrks], RightSnsr[NumTrks], CntrSnsr[NumCntr * NumTrks],
           A4,                 A3,                  A2,
// Delay seconds: WaitABitSecs array Left wait, Right wait
       0, 0,
//  PORTS: FirstLite, SoundCntl,
           10,        4,
// gate port - or empty 0
           0
};
```

The Design

When a train hits a distant sensor, the lights and sound alerts are activated after an optional *WaitABitSecs* delay. They will continue for *ShortTripSecs* seconds if a sensor at the crossing is not hit. If a crossing sensor is hit the alert will start or continue until the crossing sensor is cleared. Hitting the crossing sensor will also set the exit distant sensors to ignore the exit of a train.

The timer *ExitSecs* controls the maximum time the distant sensors will be ignored if the train doesn't exit.

The timer *GateDlyMs* controls when the gate is dropped after the lights are started, and when the gate is raised after the crossing sensor clears.

The timer *LiteDlyMs* controls when the lights and sound are stopped after the crossing sensor clears.

Sensors in the track may be either analog or digital. Sound control is a digital output. Gate control is a digital output. There are *define* statements which specify what is up/down or on/off.

Digital sensor reading for track/train detection is supported. Analog (photo-sensitive) sensors are integrated over *LoopMS* interval (milliseconds) to average variance in LED or other variable lighting. The sensitivity of the photo-based sensors is configured by the *Sentivity* control.

```
// *************************************************************************
// Timing definitions
// |-WaitABitSecs-|Sound and lights start
//                |GateDlyMs|Delay for gate drop
//                |--ShortTripSecs---|Clear signal if crossing not hit
//                                      |GateDlyMs|Delay for gate raise
//                                                |LiteDlyMs|Delay sig stop
//                                      |-----------ExitSecs-------|Cleanup
// LS.................................CS.................................RS
// Left sensors                 Crossing sensors               Right sensors
//
// WaitABitSecs - delay from distant sensor hit to signal starts
// GateDlyMS - delay between lights start and gates drop
// LiteDlyMS - delay between gates raise and lights stop
// ShortTripSecs - timeout if distant sensor hit but train stops short of crossing
// ExitSecs  - ignore exit sensor //
*************************************************************************
```

Other configurable setting: Flash interval; sound, lamp, and gate signal levels; delays for the crossing light, sound, and gate movement sequences;

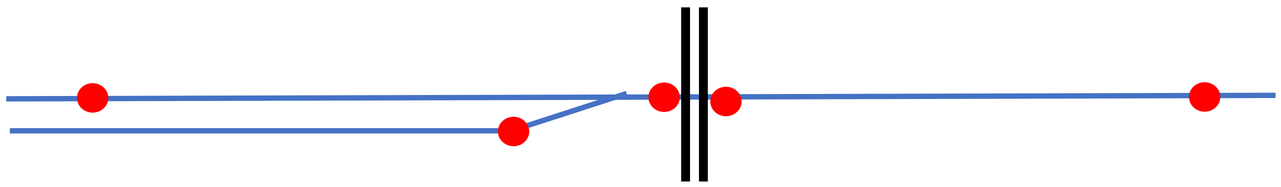
```
// high/low defines for sound - USER-CHANGEABLE
#define SndOff   HIGH
#define SndOn    LOW
// define lampon/off - USER-CHANGEABLE
#define LampOn   LOW
#define LampOff  HIGH

// Sensitivity - sensor must have dropped at least this percent to trigger
#define Sentivity 50

// if digital track sensors use clear for occupied, true
#define DigInvert true
// FlashIntvl - ms per flash for crossing lites
#define FlashIntvl 600
// ShortTripSecs - turn off after distant hit but center not hit
#define ShortTripSecs 30
// ExitSecs - ignore exit timer
#define ExitSecs 60
```

```
// GateDlyMs - number of ms after sensing and lites/bell start before gates drop
//           - also number of ms after sensing clears gates go up
#define GateDlyMs 1000. // 2500
// LiteDlyMs - number of ms after sensing clears that lites and bell stop
#define LiteDlyMs 2000.
// time to hold occupied status minimize false clears [2]
#define HoldOccMS 2000
// if SnsrActiveLED is non-zero, light the LED [3]
//     indicated if any sensor is active
#define SnsrActiveLED 13
//no-op #define SnsrActiveLED 0
```

Placing Sensors

Two *distant* sensors and a *crossing* sensor are required as a minimum. The distant sensors should be placed where one desires the crossing signal to start. An optional delay can be invoked if a distant sensor is very distant. The *crossing* sensor should be as close to the road as possible. A second crossing sensor may be placed at the road if desired.

In other trackage configurations sensors may be placed to reflect operational considerations. For example, a main line with a siding could have the main line sensors reasonably distant, but it would make sense for the siding sensor to be near the fouling point of the turnout to enter the main, allowing a train in the siding to approach the crossing and wait for a meet without triggering the signals.



Software: Definitions

The definition below represents the track diagram shown above: A main track with a siding. The sensors are analog and on the main track are A0 & A2. The siding sensor is A1. The crossing sensors are A3 and A4. The crossing lights are digital outputs 2 & 3. The sound is digital output 4. There are no gates.

```
// 2 tracks main+siding left, 1 trk at crossing, 1 trk right
#define NumXings 1 // Number of grade crossings
#define NumTrks  2 // each crossing - NumTrks tracks
#define NumCntr  2 // number of center sensors per track
#define NumGates 0 // 0 or 1
```

[2] When I tested on the layout, I used a train with logging cars and a very short logging caboose. This caused the sensors to alternate between clear and occupied while the train was passing. I decided to place a timer to hold a sensor *occupied* (in software) to control swings caused by skeleton cars or couplings.

[3] Sensor setup: When installing or debugging an installation, I found a need to confirm that the Arduino was detecting the sensor properly. I decided to use the on-board LED 13 for this purpose. You can choose some other digital output for this purpose. A 0 (zero) will disable this feature.

```
// Define the crossing by the following array
XingDef Crossing [] =
//  SnsrType,
  {  AlgType,
//  LeftSnsr[NumTrks], RightSnsr[NumTrks], CntrSnsr[NumCntr*NumTrks],
  {   A0,  A1,            A2, 0,             A3,    A4,   0, 0,
// WaitABitSecs array
//   Left wait,        Right wait
      3,   0,             3,  0,
//     FirstLite, SoundCntl
       2,         4,
// gate port - or empty 0
       0
};
```

Initial Setup/Reset Process

The setup runs a cycle of the lamps, sound, and gates and then goes into normal mode waiting for a sensor to be hit.

Board Schematic

There is a separate document for the board wiring: GateCrossingBoard*.pdf. A commercial version of the board is available from Model Railroad Control Systems (https://www.modelrailroadcontrolsystems.com/arduino-nano-grade-crossing-break-out/).

Tips, Techniques, References

The document repository for the grade crossing controller including the latest copies of the sketch may be found at https://github.com/joneschmidt/GradeCrossing.

Installation of the photo-optical sensors can be challenging if only one set of eyes are available. There is also a reference above to the *`Sentivity`* definition. We recommend that the user refer to the *ResistTest* sketch found at (https://github.com/joneschmidt/ResistTest). This sketch will help identify whether an optical sensor is working and connected properly. I use it to select the *`Sentivity`* setting such that the board doesn't give false positives. Running a car across the sensors and observing the drop in reading will help determine a value.

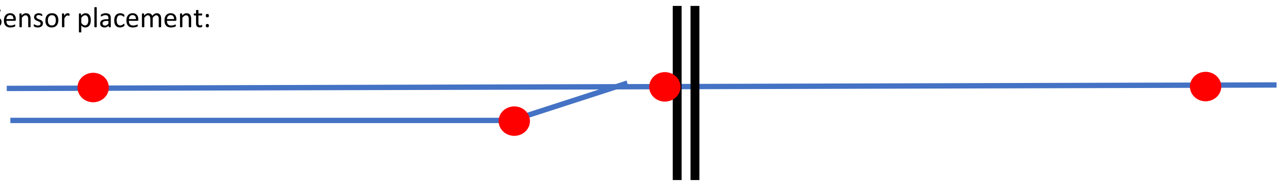
A quick way to trigger a short test of the grade crossing is to cover the crossing sensor until the sequence is started. It will cause the crossing to go through its full sequence and then reset.

Specifics for the Central Vermont

South Coventry

Note re crossing lights: The **red** leads are the +5 volt common.

Sensor placement:



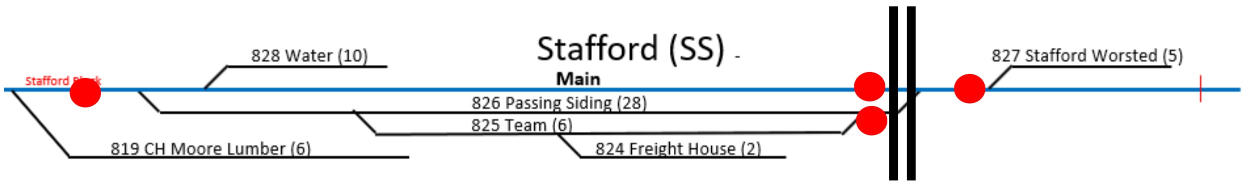
Software configuration:

```
// DEFINITIONS for Central Vermont South Coventry
// 2 tracks main+siding south, 1 trk at crossing, 1 trk north, no gates
#define NumXings 1 // Number of grade crossings
#define NumTrks  2 // each crossing - NumTrks tracks
#define NumCntr  1 // number of center sensors per track
#define NumGates 0 // 0 or 1
// Define the crossing by the following array
XingDef Crossing [] =
//   SnsrType, LeftSnsr[NumTrks], RightSnsr[NumTrks], CntrSnsr[NumCntr*NumTrks],
  {  AlgType, A0,  A1,            A2, 0,             A3,    0,
// WaitABitSecs array Left wait, Right wait
       3,0,3,0,
//     FirstLite, SoundCntl
       2,          4,
// gate port - or empty 0
          0
};//
// END DEFINITIONS for Central Vermont South Coventry
```

Stafford

Note re crossing lights: The **black** leads are the +5 volt common.

Sensor placement:

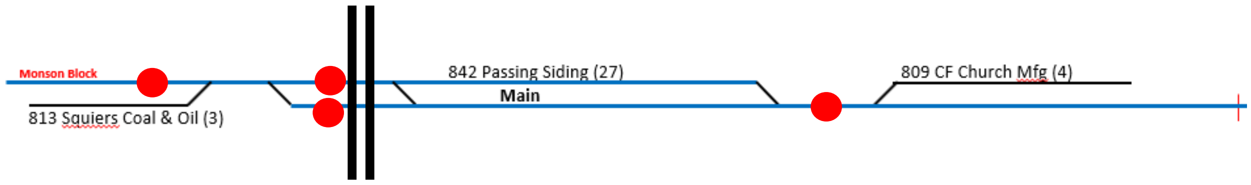


Software configuration:

```
// DEFINITIONS for Central Vermont Stafford
// 1 trk south, 2 trks at crossing, 1 trk north, no gates
#define NumXings 1 // Number of grade crossings
#define NumTrks  1 // each crossing - max tracks in/out of crossing
#define NumCntr  2 // number of center sensors per track
#define NumGates 0 // 0 or 1

// A crossing is defined by the following array
XingDef Crossing [] =
//  SnsrType, LeftSnsr[NumTrks], RightSnsr[NumTrks], CntrSnsr[NumCntr*NumTrks],
  {  AlgType, A0,                A1,                      A3,    A2,
// WaitABitSecs array Left wait, Right wait
       3,   3,
//     FirstLite, SoundCntl,
       2,          4,
// gate port - or empty 0
           0
};//
// END DEFINITIONS for Central Vermont Stafford
```

Monson

Sensor placement:



Software configuration:

```
// DEFINITIONS for Central Vermont Monson
// 1 trk south, 2 trks at crossing, 1 trk north, no gates
#define NumXings 1 // Number of grade crossings
#define NumTrks  2 // each crossing - max tracks in/out of crossing
#define NumCntr  1 // number of center sensors per track
#define NumGates 0 // 0 or 1
// delay approach from south 0 secs
// approach from south probably exits on trk 2 main north
// A crossing is defined by the following array
XingDef Crossing [] =
//  SnsrType, LeftSnsr[NumTrks], RightSnsr[NumTrks], CntrSnsr[NumTrks],
  {  AlgType,    A0, 0,            A1, 0,            A2,  A3,
// WaitABitSecs array Left wait, Right wait
       3,0,6,0,
//     FirstLite, SoundCntl,
       2,         4,
// gate port - or empty 0
          0
};// END DEFINITIONS for Central Vermont Monson
```