

Grade Crossing Controller

Contents

Grade Crossing Controller	1
The Project	2
Arduino Sketch: GradeCrossing	2
Features	2
The Design.....	2
Placing Sensors.....	3
Software: Definitions	3
Other Defines	5
Initial Setup/Reset Process	6
Board Schematic	6
Tips, Techniques, References.....	6
Specifics for the Central Vermont.....	6
South Coventry	6
Stafford	7
Monson	8

Revision History:

- 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:

- Three sensor (sets) operation:
 - Left and right “distant” sensors to detect the approach and exit of a train
 - A crossing sensor 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
- Timeout for train hitting a distant sensor but stopping short of the crossing
- Timeout 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
- Optional sound control
- Optional gate control either servo or digital
- Multiple separate crossings on a single Arduino depending on port availability

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 may either be a servo or 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 *Sensitivity* control.

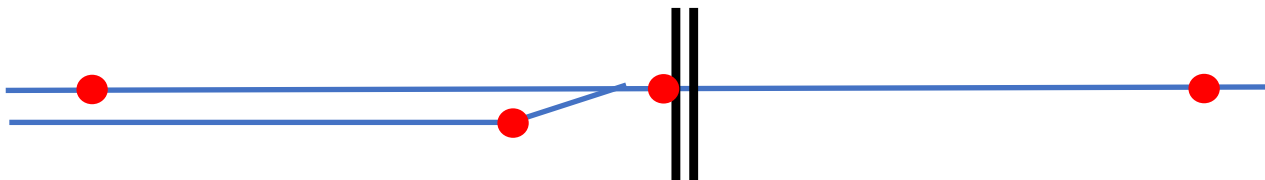
Other configurable setting: Flash interval; sound, lamp, and gate signal levels; delays for the crossing light, sound, and gate movement sequences;

```
// *****
// 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 //
// *****
```

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 too distant. The *crossing* sensor should be as close to the road as possible.

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

A crossing is defined with these defines and data structures:

```
// Define the active crossings - NumXings crossings
// each crossing - up to NumTrks tracks, up to 3 sensors per track
// number of gate controls per crossing
#define NumXings 1
#define NumTrks 2
#define NumGates 2

// Define crossing (XingDef) array elements
// DON'T CHANGE
#define NumSnsrs NumTrks * 3
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 address - digital mode or empty 0
uint8_t GateCntl[NumGates];
// gate potentiometers - servo mode or empty 0
uint8_t GatePots[NumGates*2];
// GateType SrvoType or DigType
int GateType;
} XingDef;

```

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 sensor is A3. 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 the crossing by the following array
XingDef Crossing [] =
// SnsrType, LeftSnsr[NumTrks], RightSnsr[NumTrks], CntrSnsr[NumTrks],
{ AlgType, A0, A1, A2, 0, A3, 0,
// WaitABitSecs array Left wait, Right wait
3,0,3,0,
// FirstLite, SoundCntl, GateCntl,
2, 4, 0, 0,
// GatePots,
// Up,Dwn,Up,Dwn GateType
0,0,0,0, NoGate};
//
// Sensitivity - sensor must be within % of lowest to trigger
#define Sensity 70
// if digital track sensors use clear for occupied, true
#define DigInvert true
// FlashIntvl - ms per flash for crossing lights
#define FlashIntvl 600
// ShortTripSecs - turn off after distant hit but center not hit
#define ShortTripSecs 30
// ExitSecs - ignore exit timer
#define ExitSecs 120
// GateDlyMs - number of ms after sensing and lights/bell start before gates drop
// - also number of ms after sensing clears gates go up
#define GateDlyMs 0 // 1500
// LiteDlyMs - number of ms after sensing clears that lights and bell stop
#define LiteDlyMs 2000.

```

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.

```

// if SnsrActiveLED is non-zero, light the LED indicated if any sensor is active
#define SnsrActiveLED 13
// #define SnsrActiveLED 0

```

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.

```
// hold occupied status for this time to minimize bounce (false clear)
// - for couplers, skeleton cars - in milliseconds
#define HoldOccMS 1000
```

Other Defines

```
// if digital track sensors use clear for occupied, true
#define DigInvert true
//
// *****
// END USER TUNEABLE Defines
// *****
// Change anything below here at your own risk
// *****
// analog sampling controls - needed to average light flicker
// length of sample integration in millisecs
// 34 ms for 60 cycle lighting mains
// 40 ms for 50 cycle lighting mains
#define LoopMS 34

// Servo defines for gate
#define VSServo true
#if VSServo
    #include <VarSpeedServo.h>
    #define ServoSpeed 10
#else
    #include <Servo.h>
#endif
#define FullArc 160
// Defines for EEPROM for Servo processing
// addresses in EEPROM
#define E_Flag 0
#define E_GateDown0 E_Flag + 2 + 1
#define E_GateDown1 E_GateDown0 + 2 + 1
#define E_GateUp0 E_GateDown1 + 2 + 1
#define E_GateUp1 E_GateUp0 + 2 + 1
#define E_Okay 2
#define E_Chngd 3
#define MinPotMove 10

// high/low defines for digital gate
#define GateDown HIGH
#define GateUp LOW

// high/low defines for sound
#define SndOff HIGH
#define SndOn LOW

// define lampon/off
#define LampOn LOW
#define LampOff HIGH

// ***** Trace & Debug
// 0 - no trace
#define Trace 0
// delay in ms for main loop sampling
// -- if tracing/debugging
#define MainLoopDlyMS 5000
```

Initial Setup/Reset Process

The setup process allows adjusting the gate servo positions if they are configured. At the end of the setup the sketch runs a cycle of the lamps/sound/gates and then goes into normal mode waiting for a sensor to be hit.

If servos are configured:

1. Setup will read EEPROM for previous up/down servo values
2. Setup begins a process to configure the first gate servo.
 - a. It sets 10 second timeout looking to see if the pots configured have changed. Both crossing lights will be on during this timeout.
 - b. If a pot has changed the gate will move to the new position, and the timeout is refreshed. The user can take as much time as they need to adjust the servo position.
3. After 10 seconds of no pot movement, the lights will flash for approximately 5 seconds.
4. Setup repeats the process for the second gate servo
5. Setup will rewrite EEPROM servo values if they were changed

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 *Sensitivity* 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 *Sensitivity* 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 realistic value.

If you use servos for the gate motion, we recommend that you use the VarSpeedServo library for realistic slow gate motion. This library can be found at <https://github.com/netlabtoolkit/VarSpeedServo>.

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.

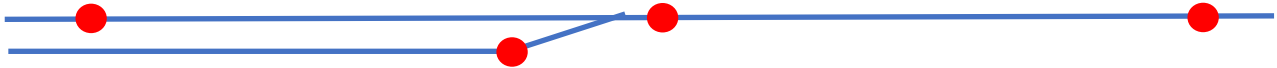
Specifics for the Central Vermont

South Coventry

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

Sensor placement:





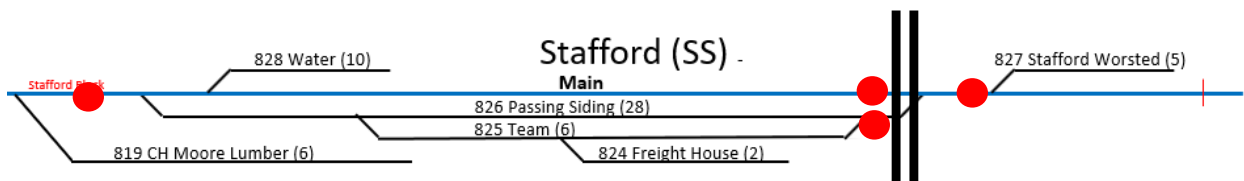
Software configuration:

```
// DEFINITIONS for Central Vermont South Coventry
// main+siding south, 1 trk at crossing, 1 trk north
// A crossing is defined by the following array
XingDef Crossing [NumXings] =
//  SnsrType, RightSnsr[NumTrks], LeftSnsr[NumTrks], CntrSnsr[NumTrks],
//  { AlgType, A0, A1, A3, 0, A2, 0,
//    FirstLite, SoundCntl, GateCntl, GateType
//    2, 4, 9, 10, NoGate};
// and the WaitABitSecs array
// Right wait, Left wait
int WaitABitSecs[NumXings][NumTrks*2] =
{0,0,0,0};
//
// Sensitivity - sensor must have dropped % to trigger
#define Sensity 70
// if digital track sensors use clear for occupied, true
#define DigInvert true
//
// FlashIntvl - ms per flash for crossing lights
#define FlashIntvl 700
// ShortTripSecs - turn off after distant hit but center not hit
#define ShortTripSecs 20
// GateDlyMs - number of ms after sensing and lights/bell start before gates
drop
// - also number of ms after sensing clears gates go up
#define GateDlyMs 4500
// LiteDlyMs - number of ms after sensing clears that lights and bell stop
#define LiteDlyMs 3500
// 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
// delay approach from south 10 secs
// A crossing is defined by the following array
XingDef Crossing [NumXings] =
```

```

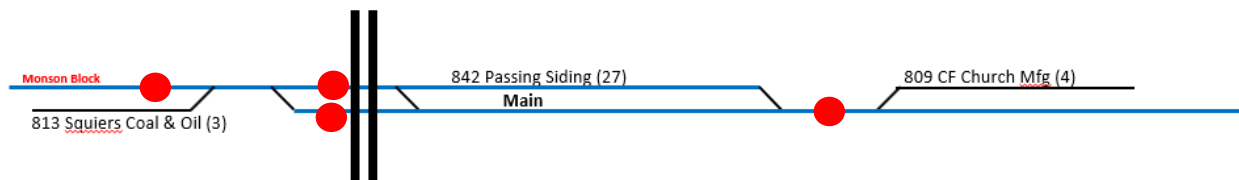
//  SnsrType, RightSnsr[NumTrks], LeftSnsr[NumTrks], CntrSnsr[NumTrks],
//  { AlgType, A0, 0, A3, 0, A1, A2,
//    FirstLite, SoundCntl, GateCntl, GateType
//      2, 4, 0, 0, NoGate};
// and the WaitABitSecs array
// Right wait, Left wait
int WaitABitSecs[NumXings][NumTrks*2] =
  {10, 0, 0, 0};
//
// Sensitivity - sensor must have dropped % to trigger
#define Sntivity 70
// if digital track sensors use clear for occupied, true
#define DigInvert true
//
// FlashIntvl - ms per flash for crossing lights
#define FlashIntvl 700
// ShortTripSecs - turn off after distant hit but center not hit
#define ShortTripSecs 20
// GateDlyMs - number of ms after sensing and lights/bell start before gates
drop
// - also number of ms after sensing clears gates go up
#define GateDlyMs 4500
// LiteDlyMs - number of ms after sensing clears that lights and bell stop
#define LiteDlyMs 3000
// END DEFINITIONS for Central Vermont Stafford

```

Monson

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

Sensor placement:



Software configuration:

```

// DEFINITIONS for Central Vermont Monson
// 1 trk south, 2 trks at crossing, 1 trk north
// 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 [NumXings] =
//  SnsrType, RightSnsr[NumTrks], LeftSnsr[NumTrks], CntrSnsr[NumTrks],
//  { AlgType, A0, 0, A1, 0, A2, A3,
//    FirstLite, SoundCntl, GateCntl, GateType
//      2, 4, 0, 0, NoGate};
// and the WaitABitSecs array
// Right wait, Left wait
int WaitABitSecs[NumXings][NumTrks*2] =
  {0, 0, 0, 0};
// Sensitivity - sensor must have dropped % to trigger
#define Sntivity 70

```



```
// if digital track sensors use clear for occupied, true
#define DigInvert true
//
// FlashIntvl - ms per flash for crossing lights
#define FlashIntvl 700
// ShortTripSecs - turn off after distant hit but center not hit
#define ShortTripSecs 20
// GateDlyMs - number of ms after sensing and lights/bell start before gates
drop
//          - also number of ms after sensing clears gates go up
#define GateDlyMs 4500
// LiteDlyMs - number of ms after sensing clears that lights and bell stop
#define LiteDlyMs 3000
// END DEFINITIONS for Central Vermont Monson
```