Grade Crossing Controller

Contents

Revision History:

- 6 May 2024 Updated for version 3.2 restructuring and bug fixes
- 10 July, 2022 Updated for version 3.0 restructuring, initialization, and bug fixes
- 30 April 2022 Updated for version 2.0 with VarSpeedServo and change to TimeOutSecs
- 28 March 2022 Original version 1.3

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:

- Multiple tracks per crossing
- Handle different track counts left, right, and at the road crossing
- Three sensor operation:
    - "Distant" sensors per track to detect the approach and exit of a train
    - A crossing sensor at the road (per track) to confirm the train
- Handle a train hitting a distant sensor but stopping short of the crossing
- Allow a train to exit and hit the distant exit sensor without triggering the flashers
- Multiple separate crossings on a single Arduino depending on ports
- Auto-calibrating analog sensors using photoresistors or phototransistors
- Sensor sensitivity control
- Optional sound control
- Optional gate control either servo or digital
- Version 2.0 & later: Optionally use the VarSpeedServo library for gate control

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 *TimeoutSecs* seconds or until the sensor at the crossing is hit. If the timeout occurs the alert will stop. If the crossing sensor is hit the alert will start or continue until the crossing sensor is cleared. Hitting the crossing sensor will also set the opposing exit distant sensors to *ignore until cleared*. It will not trigger an alert on the exit of a train.

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 lites and sound are stopped after the crossing sensor clears.

The timer *MaxAlertSecs* controls the maximum time the lights and sound will run after being started.

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

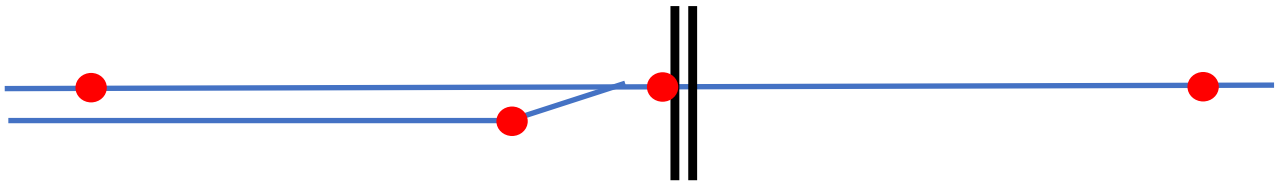
Digital sensor reading for track/train detection is straightforward. Analog sensors are integrated over *LoopMS* interval (milliseconds) to average variance in LED or other variable lighting. Analog sensors are also recalibrated every *RecalibrateMts* minutes to adjust for changes in room lighting.

```
// ************************************************************************
// Timing definitions
// |-WaitABitSecs-|B4 Start signal
//                |GateDlyMs|Delay for gate drop
//                |--TimeoutSecs---|Clear signal if crossing not hit
//                                  |GateDlyMs|Delay for gate raise
//                                            |LiteDlyMs|Delay sig stop
//                                  |---TimeoutSecs---|Cleanup
// ES.............................CS.............................WS
// East sensors                Crossing sensors              West sensors
//
// WaitABitSecs - delay from distant sensor hit until signal starts
// GateDlyMS    - delay between lites start and gates drop, gates raise and lites stop
// TimeOutSecs  - timeout if distant sensor hit but train stops short of crossing
//              - also on exit timeout for exit sensor ignored
// ************************************************************************
```

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
//
// dont change:
#define NumSnsrs NumTrks * 3

// Sensor types for in-track sensors: digital or analog
#define AlgType   1
#define DigType   2
// Crossing gate type:
#define NoGate    0
// DigType for on/off control
// SrvoType for servo-controlled gates
#define SrvoType  3

// Define crossing (XingDef) array elements
```

```
typedef struct {
  // input sensors analog or digital - DigType - AlgType
  int SnsrType;
  // sensors (EastSnsr[NumTrks], WestSnsr[NumTrks], XingSnsr[NumTrks])
  // or empty 0
  uint8_t SnsrLst[NumSnsrs];
  // 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 addresses digital or servo or empty 0
  uint8_t GateCntl[NumGates];
  // gate potentiometers for high/low adjust - 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 two gates and two gate pots.

```
// 1 trk south, 2 trks at crossing, 1 trk north
// main+siding south, 1 trk at crossing, 1 trk north
// A crossing is defined by the following array
XingDef Crossing [NumXings] =
//  SnsrType, EastSnsr[NumTrks], WestSnsr[NumTrks], XingSnsr[NumTrks],
  {  AlgType, A0, A1,            A2,    0,          A3, 0,
//     FirstLite, SoundCntl, GateCntl,
       2,         4,         9, 10,
//     GatePots,
//     Up,Dwn,Up,Dwn     GateType
       A6,A7, A6,A7,     SrvoType};
// and the WaitABitSecs array
// East wait, West wait
int WaitABitSecs[NumXings][NumTrks*2] =
  {0,0,0,0};
//
// Sensitivity - sensor must drop at least this % of trigger
#define Sentivity 30
//
// TimeoutSecs - turn off after distant hit but center not hit
#define TimeoutSecs 20

// MaxAlertSecs - turn off alert if no activity
#define MaxAlertSecs 90

// 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 2500

// LiteDlyMs - number of ms after sensing clears that lites and bell stop
#define LiteDlyMs 3000
```

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 3000
```

Other Defines

```
// if digital track sensors use clear for occupied, true
#define DigInvert true
//
// FlashIntvl - ms per flash for crossing lites
#define FlashIntvl 500
//
// ************************************************************************
// 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; 1 - events; 2 - flow and data
//   3 - even more
#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 lites 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 lites 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 `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 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.

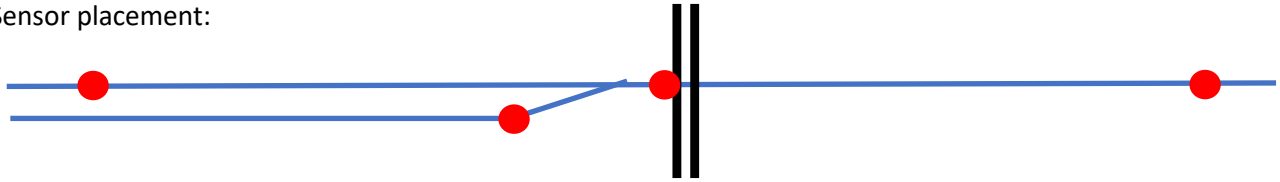
A quick way to trigger a short test of the grade crossing is to cover the crossing sensor until the sequence is started. It will cause the crossing to go through its full sequence and then reset.

Specifics for the Central Vermont

South Coventry

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

Sensor placement:



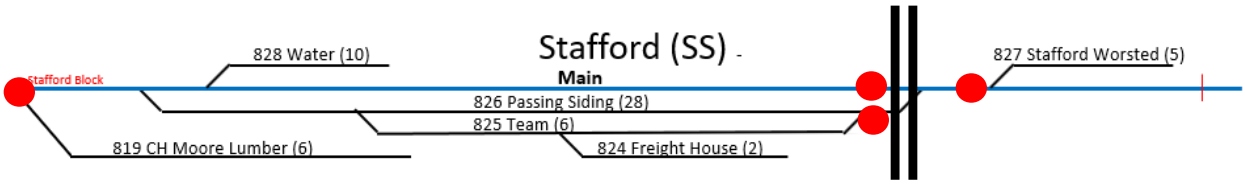
Software configuration:

```
// 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, EastSnsr[NumTrks], WestSnsr[NumTrks], XingSnsr[NumTrks],
  {  AlgType, A0, A1,            A3,    0,          A2, 0,
//     FirstLite, SoundCntl, GateCntl, GateType
       2,         4,         9, 10,    NoGate};
// and the WaitABitSecs array
// East wait, West wait
int WaitABitSecs[NumXings][NumTrks*2] =
  {0,0,0,0};
//
// Sensitivity - sensor must be within % of lowest to trigger
#define Sentivity 16
// if digital track sensors use clear for occupied, true
#define DigInvert true
//
// FlashIntvl - ms per flash for crossing lites
#define FlashIntvl 700
// TimeoutSecs - turn off after distant hit but center not hit
#define TimeoutSecs 20
// 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 4500
// LiteDlyMs - number of ms after sensing clears that lites 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:



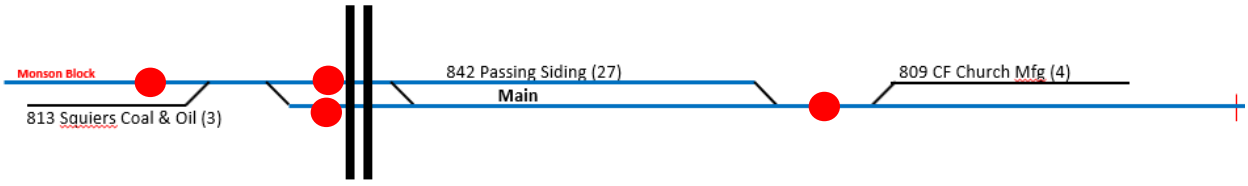
Software configuration:

```
// 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, EastSnsr[NumTrks], WestSnsr[NumTrks], XingSnsr[NumTrks],
  {  AlgType, A0, 0,             A3,    0,          A1,   A2,
//     FirstLite, SoundCntl, GateCntl, GateType
       2,         4,         0, 0,     NoGate};
// and the WaitABitSecs array
// East wait, West wait
int WaitABitSecs[NumXings][NumTrks*2] =
  {10,  0,  0,  0};
//
// Sensitivity - sensor must be within % of lowest to trigger
#define Sentivity 2
// if digital track sensors use clear for occupied, true
#define DigInvert true
//
// FlashIntvl - ms per flash for crossing lites
#define FlashIntvl 700
// TimeoutSecs - turn off after distant hit but center not hit
#define TimeoutSecs 20
// 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 4500
// LiteDlyMs - number of ms after sensing clears that lites 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, EastSnsr[NumTrks], WestSnsr[NumTrks], XingSnsr[NumTrks],
  {  AlgType,    A0, 0,            A1, 0,            A2,  A3,
//     FirstLite, SoundCntl, GateCntl, GateType
```

```
       2,         4,         0, 0,     NoGate};
// and the WaitABitSecs array
// East wait, West wait
int WaitABitSecs[NumXings][NumTrks*2] =
  {0,  0,  0,  0};
// Sensitivity - sensor must be within % of lowest to trigger
#define Sentivity 3
// if digital track sensors use clear for occupied, true
#define DigInvert true
//
// FlashIntvl - ms per flash for crossing lites
#define FlashIntvl 700
// TimeoutSecs - turn off after distant hit but center not hit
#define TimeoutSecs 20
// 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 4500
// LiteDlyMs - number of ms after sensing clears that lites and bell stop
#define LiteDlyMs 3000
// END DEFINITIONS for Central Vermont Monson
```