Yosemite in HO

FROM HALFDOME TO CAMP CURRY

Don Evans | 6x9 layout | 07/11/2018

What is it?

- This document describes a 6-foot by 9-foot HO scale model railroad layout, created in the single car garage walled off from the double garage to preserve temperature and humidity during the cold winters and hot summers of Eagle, Idaho. The scale is HO, 1:87.1 ratio.

- This layout is intended to incorporate some of the features of my favorite place on earth, Yosemite Valley, California, USA. I have included such features as Vernal Falls, Merced River, Half Dome, Camp Curry, campgrounds and a portion of the Awhanhee Hotel.

What is included?

- I will attempt to give a comprehensive presentation of the entire layout, including images of each portion of it, as well as close-ups of particularily interesting items.

- The operation of the layout is largely handled by electronic devices. The motion and sound of the steam engines is controlled by standard off-the-counter DCC hand held remote connected to a five ampere DCC amplifier.

- The operation of the turnouts can be handled two ways. Each turnout has a solenoid actuator integrally installed, so that it can either be remotely switched, or can be manually switched at the turnout itself. The remote operation of the turnouts is handled through a micro-processor called Arduino Uno. This device collects desired turnout positions from a panel that contains a toggle switch for each turnout, of which there are sixteen. The status of each toggle switch is collected into an integrated circuit 16 bit shift-register in the switch panel, and then shifted out serially on a pair of conductors, to be delivered into the micro-processor. This occurs continuously at a very rapid pace so that when a switch is thrown on the panel the change of status of that switch is instantly recognized by the micro-processor.

- When the processor receives the status information from the toggle switch panel, it relays that information, again by serial communications, to a waiting shift-register in another panel that contains Yellow/Green LED lights indicating the status of each of the sixteen turnouts on the layout.

- Simultaneously, the micro-processor compares the new status of the toggle switches with a previously memorized status. If there is a change of one or more toggle switch positions, the new information is also output to a set of turnout current drivers, which will then output the appropriate polarity pulse of current to the solenoid of the targeted turnout.

- To fully illuminate the electronics that I have just described, I will be including in this document, full electronic schematics together with listing of the C-language code that I have written for the micro-processor to execute.

- All documentation will be available on my website: www.lazuli.com, by using the button named 'Links' in the left side of the home page. This documentation will include this document, all schematic diagrams for the electronics and the code written for the Arduino Uno and Nano micro-processors.

Now, let's get started with an image showing the full extend of the layout. Remember that this layout is housed inside a garage/workshop and the surroundings are not pretty!

You can see that it is bounded on the south long side by a sky image and some far off snow covered mountains just above the granite wall that extends from Half Dome to your left to another granite monolith.

THE NEXT SERIES OF IMAGES WILL SHOW THE VARIOUS FEATURES AND I WILL DESCRIBE WHAT IS SEEN IN EACH OF THEM.

This image looks down upon the corner of the table that emphasizes Camp Curry, with several camp sites, a couple of log cabins and three tent cabins to emulate those found in profusion in Camp Curry.

This looks down upon a few of the camp sites, including one that has Susan and Steve's Airstream, all shiny and bright!

A closer look at a meadow with some critters gamboling around within it.

A shot of Vernal Falls and a view of the Merced River, crossed by a wooden trestle.

Half Dome looking down upon Camp Curry and environs. Poetic licence allow the tracks to run underneath traveling through hand made wooden portals.

A different and much longer wooden trestle crossing the river and extending over several tracks. This trestle is two-way so that the trains can pass each other as they cross the river.

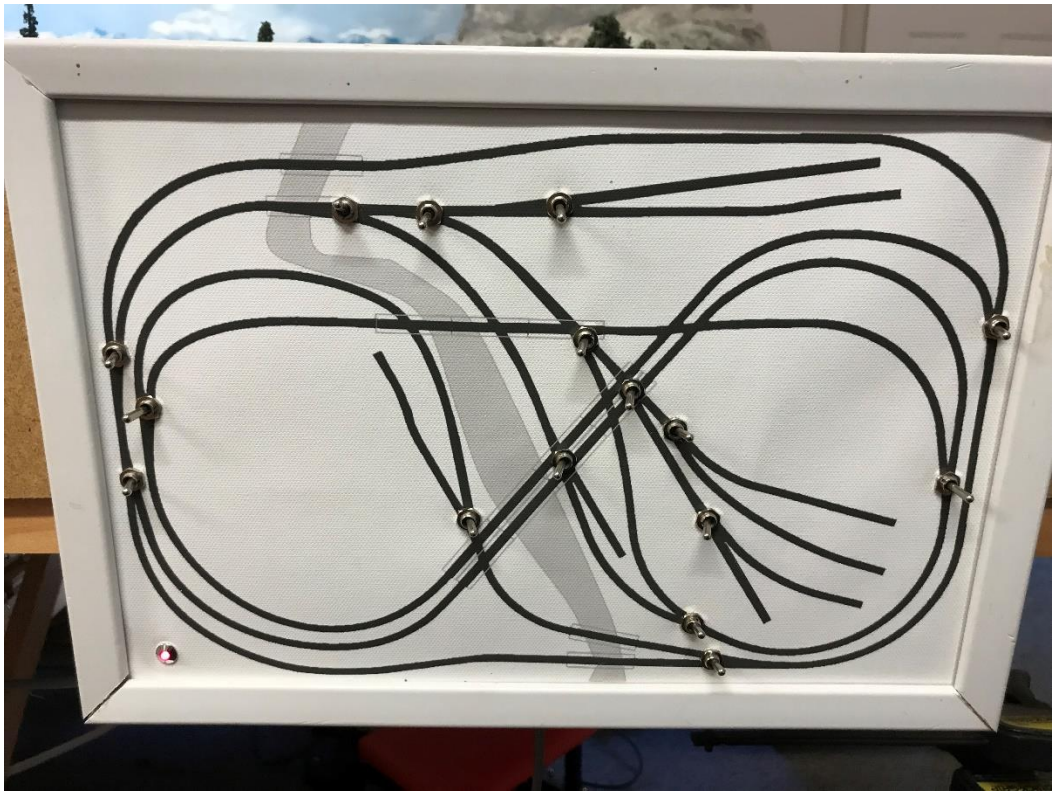
The road from Camp Curry that ultimately disappears into the area below Half Dome crosses a main track and has active, working, crossing gates. These gates are operated by an Arduino Nano, a very small micro-processor located under the layout. Optical sensors are located between ties upstream and downstream of the crossing. These sensors detect the arrival of an engine from either direction. Upon detection, the blinking red warning lights start flashing and an audible crossing bell begins to ring. Shortly thereafter the Nano activates two servo-motors that lower the crossing arms. When the last car in the train passes over the crossing, the servo-motors are caused to raise the arms and shortly after the lights cease flashing and the bell stops ring. Note the roof of a depot in the picture.

Here we have a structure that emulates at least a major portion of the Awhanhee Hotel in Yosemit Valley. The windows all have interior scenes glued in place behind the glass. A few patrons of the bar are sitting outside enjoying the wonderful scenery.

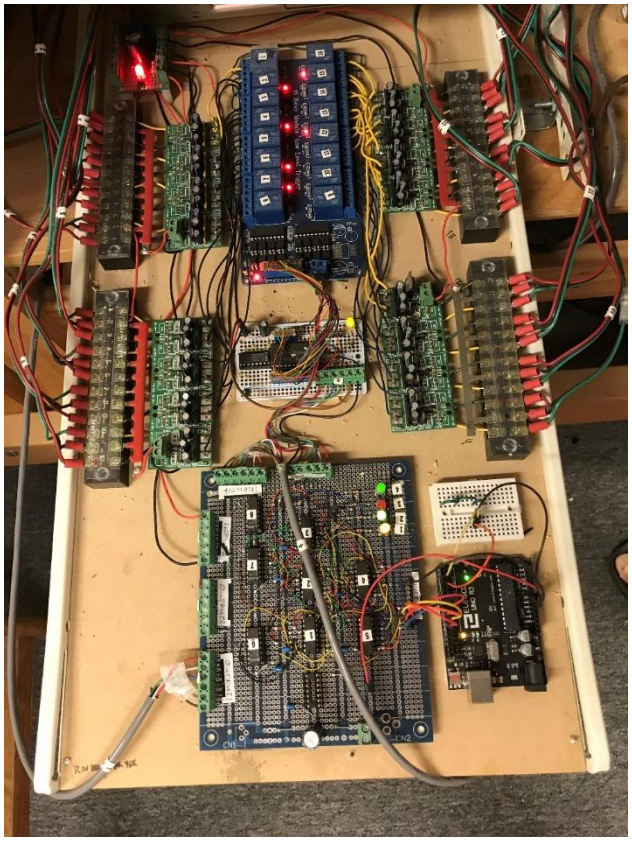
Some hardy souls making the climb up the precarious passage up the side of Half Dome. I note that some of the boards have been kicked askew?

Now, we get to the electronics associated with this HO layout. All control features are handled through the use of Arduino Uno's, with the exception of the Crossing Gates that utilize a smaller Arduino Nano.

As mention in the opening statement, a toggle switch panel has been prepared with a toggle switch representing each of the sixteen turnouts. The switches are located on the panel according to their turnout location on the image of the layout. The direction of a turnout, either straight through or turnout, is determined by the direction of it's associated toggle switch.



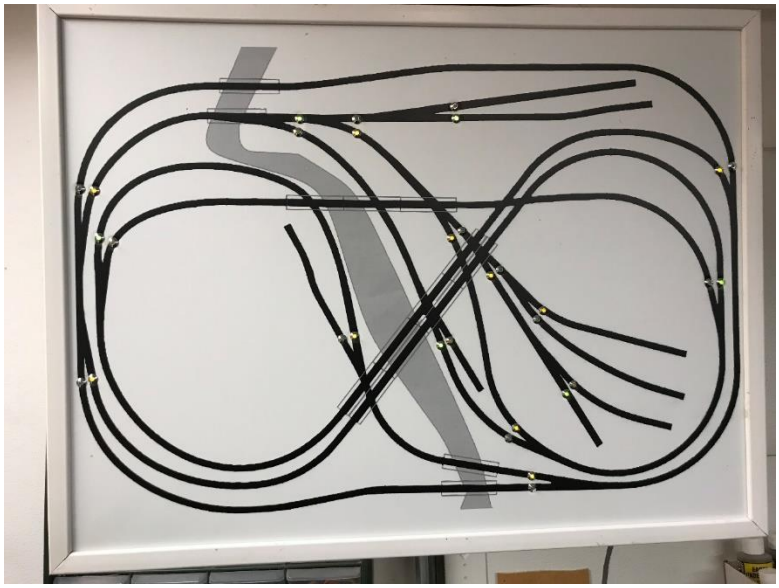
The layout is reasonably faithfully portrayed in the black lines denoting tracks. Now, having the toggle switches allows the Arduino Uno responsible for activating each turnout, to obey the wishes of the Operator. In order to accomplish this, the toggle switch positions are gathered into a digital chip called a shift register. The sixteen bits of data would look something like: 1001110101110101, with a 1 denoting a straight through turnout position and a 0 indicating a turnout position. Sixteen bits of data would normally require at least an eighteen wire cable to get the information from the panel to the micro-processor. I have simplified things by entering the data into the digital shift register in 'parallel' meaning each toggle switch position is separately entered simultaneously into the register. The data is then shifted out bit by bit and sent down a single pair of wires to land in a similar shift register at the Uno location. The difference in the two registers is that one enters data in parallel and shifts out serially, while the receiver registers enters data serially and extracts the data in parallel. Considered a matching pair.



This image shows the electronics associated with the turnout operations. The small board in the lower right corner is the Arduino Uno. The larger printed circuit board next to the Uno is a circuit containing four channels of serial data input/output. One of the channels receives the serial data from the turnout toggle switch panel. Another channel outputs serial data to a register that converts that serial information into parallel data that can then be inserted into the relay bank shown by the blue rows of relays in the upper part of the image.

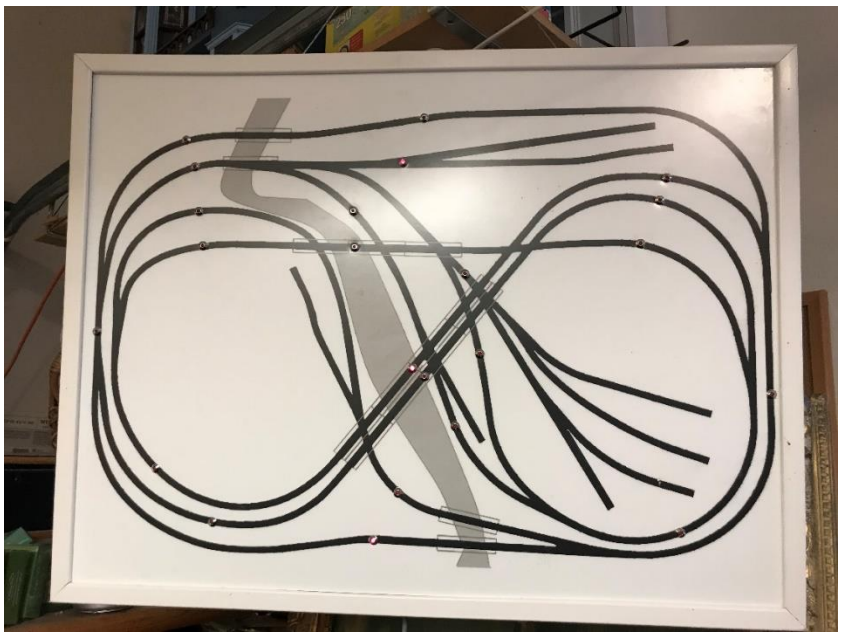
It is not desirable to apply a continuous source of energy to each turnout mechanism, as it is wasteful and heat producing. So, to activate a turnout mechanism, a short, powerful 'pulse' of energy is sent to the mechanism and this quickly causes a solenoid attached to the turnout to activate. Each mechanism consists of two coils of wire forming an electro-magnet. Pulse one of the coils and the turnout moves to straight through, pulse the other coil and the movement is to turnout. The circuits that provide these pulses are the small green boards along each side of the drawer. A pulse is sent out only if and when a toggle switch changes position. So long as the toggle has not changed, no action is required and no pulses are output, thereby saving power and reducing heat to a minimum.

The position of each toggle switch must also be displayed to the operator and this is accomplished by yet another serial output channel on the large printed circuit board. The switch position data is displayed on the following panel:



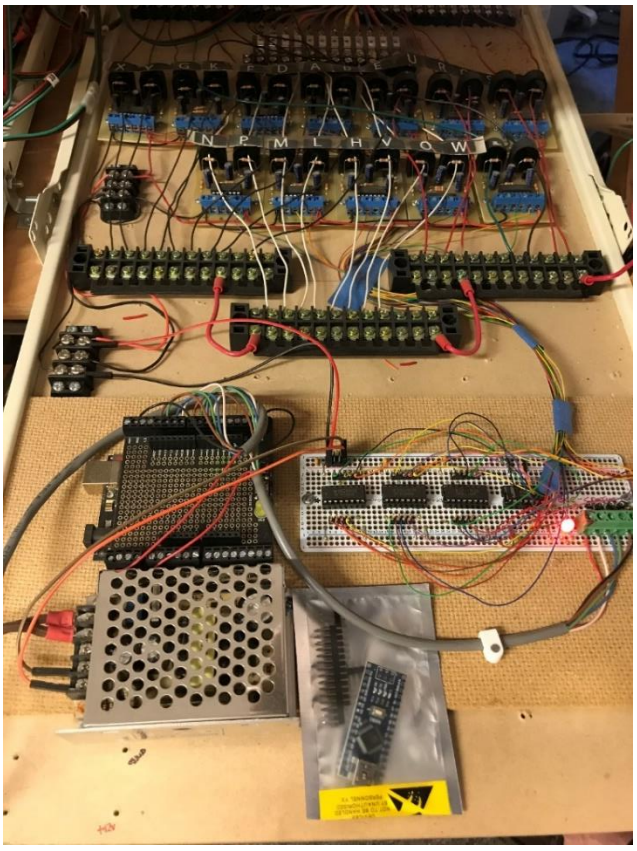
The LED lights don't show well in the image, but some can be seen. Green indicates a straight through condition of the turnout and Yellow is used for the sideways turnout condition. Each turnout has a pair of LEDs, Green and Yello, with only one of the colors displayed at any given time. This sixteen bits of data is serially transmitted from the electronics in the foregoing image, and extracted in parallel so as to provide the necessary information for each turnout LED.

On this HO layout, the tracks are energized by a DCC digital encoded electrical waveform. This allows the operator of the system to control up to six engines simultaneously without any interference between engines. DCC allows speed changes, direction of motion changes, even whistles to be blown, bells rung. All completely independent of each other. I decided to partition the entire track arrangement into individual blocks. That is, I have cut the track at specific points so as to allow detection of an engine when it enters that block. The DCC signal is the same for each block, but each block is provided that DCC signal through a separate path allow me to detect the presence of an engine as it enters each block.

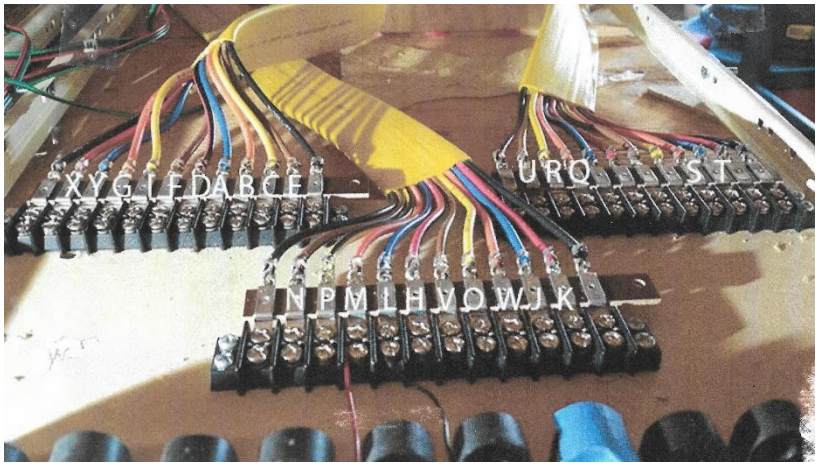


This panel receives serial data from an additional Arduino Uno, causing a Red LED to illuminate when an engine is detected in the block. There are twenty-two blocks and this image shows engines in two of the blocks. Here again, twenty-two bits of data would require a very large, 24 wire wide, cable. However, by serializing the data, a pair of wires is all that is required.

The electronics required to accomplish block occupancy detection accomplishes the task by detecting the DCC current feeding the engines motor. No engine in a block, no current detected. Engine enters block, consumes energy to drive its motor, thereby causing current to flow and that current is detected by passing the feed wire for the block through a magnetic toroid or coil of wire that feeds a sensitive circuit detecting occupancy when current flows through the feed wire. This method does not detract from the power running the engine and so is free of heating.

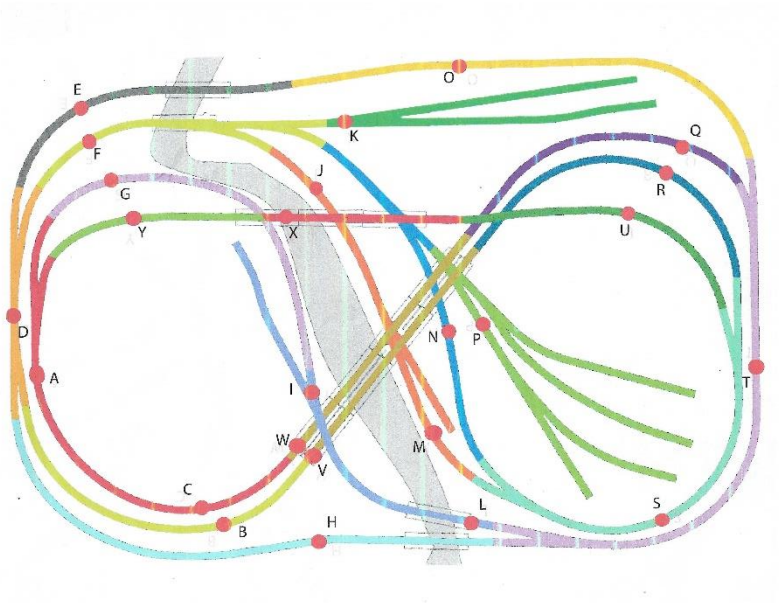


The electronics required to detect block occupancy is shown in this image. The Uno that does the heavy work is located in the lower left of the image. The current sensor elements for each of the twenty-two blocks are those blue objects in the upper part of the image. Incidentally, the little board inside the plastic envelope in the image is an Arduino Nano, showing how much smaller it is than is the Uno.



The Block Occupancy circuits all have to be brought from the individual blocks to a common point so as to pass the DCC signal through a toroidal current sensor. This image shows the 24 leads coming into terminal blocks and then leads were extended through the current sensors shown at the bottom of the image.

This image shows the block extents and includes the letter identifying each block as also shown in the image above on the terminal strips.

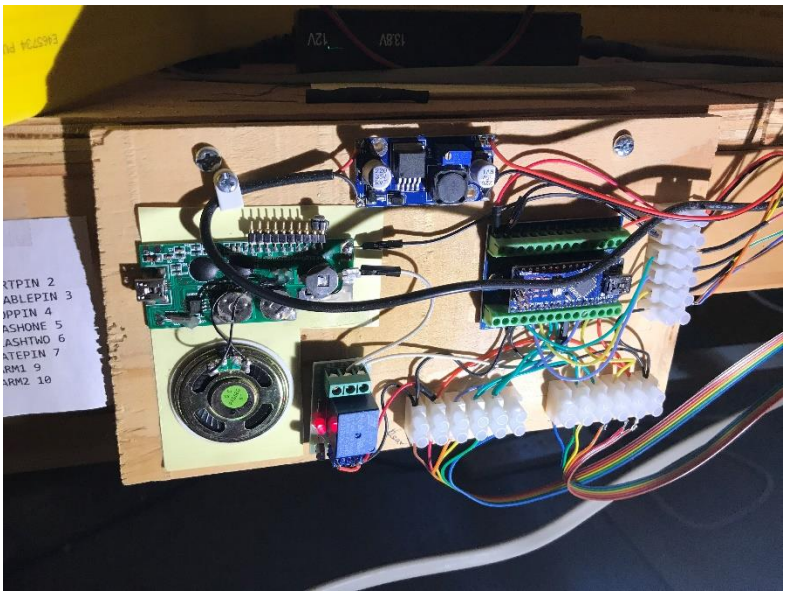


The lower image describes the Block by Letter, and the associated pin number in a 26 conductor cable running to the electronics shown in earlier images.

26 conductor cable connector layout

n.c.	V	T	R	P	O	M	K	H	X	E	B	Gnd
26	24	22	20	18	16	14	12	10	8	6	4	2
25	23	21	19	17	15	13	11	9	7	5	3	1
W	U	S	Q	Y	N	L	J	G	F	D	A	Vcc

These images show the ‘man-hole’ available after removing parts of the scenery. Because of the rectangular nature of the layout, some significant areas cannot be reached from the outer periphery.



A view of the Arduino Nano board that is responsible for the crossing gate electronics.

This is located under the layout and accessible only by crawling, sliding, otherwise getting down on the floor!

This is the HP Laptop that goes with the layout.
It is used to download programs into the Arduinos, and also allows monitoring any feedback from the downloaded software.

This completes the description of the various elements of my HO Yosemite! We now have the task of presenting the wiring diagrams for all those electronic circuits. The diagrams have the official name of 'schematics'. I will also give images of parts of the C language code inserted into the Arduinos. All the schematics and all the code is available on my website, www.lazuli.com.

This image shows the menu buttons on my website home page. Just click on the "Links" button to switch to the web page with all the various downloadable information related to my HO Yosemite!

Thanks for your interest and know that I will ultimately want to find an appreciative home for this:

HO Yosemite layout!

Appendix: C Language code for use in the Arduinos utilized in my HO Yosemite: These are merely screen grabs of the code while in the Arduino APP, available at https://www.arduino.cc/en/Main/Software, necessary for communicating from the desktop computer to/from the Arduino Uno or Nano. The actual code will be found for download from my website: www.lazuli.com.

CrossingGate-rev.1.11

```
// Crossing Gate Controller, version 1.10 Thu July 14, 2017 8:12AM
#include <PinChangeInt.h>
#include <Servo.h>  // servo library

// Note that this library disables PWM on
// pins 9 and 10!

Servo myservo1;  // servo control object
Servo myservo2;  // servo control object

/* This version is designed for the NANO.
   This code utilizes three PinChange Interrupts.

   1. On pin 2, the Start signal indicates the probable beginning of a train arrival.
   This ihterrupt routine first checks to see if the Disable pin is covered. If covered,
   this interrupt is ignored, because that indicates the train is leaving a gated block.
   If not covered, it sets the START flag, for the loop() to use to set the GATE state,
   and to set the delta t to then set the ARMS state.

   2. On pin 3, the Disable signal resets the START flag.

   3. On pin 4, the Stop signal indicates that both stop sensors are uncovered and
   sets the STOP flag, for the loop() to use to stop the ARMS state and start the delta t
   to then stop the GATE state.

   All three of these interrupts are de-bounced
   by setting/checking flags to indicate already serviced.

   4. Flasshers are on pins 5 and 6.  The Servos are on pins 9 and 10.

   5.    The photo transistors are short-circuit with light, open-circuit when covered.
    START trigger occurs when phototransistor becomes covered, therefore a RISING interrupt.
    Disable trigger is the same. The STOP trigger occurs when both stop detectors are uncovered,
    therefore a FALLING interrupt.
*/
#define STARTPIN 2
#define DISABLEPIN 3
#define STOPPIN 4
#define FLASHONE 5
#define FLASHTWO 6
#define GATEPIN 7
#define ARM1 9
#define ARM2 10
boolean volatile STARTFlag = false;
boolean volatile STOPDelayFlag = false;
boolean STOPFlag = false;
boolean GATEState = false;
boolean ArmUpState = false;
boolean ArmDownState = false;
```

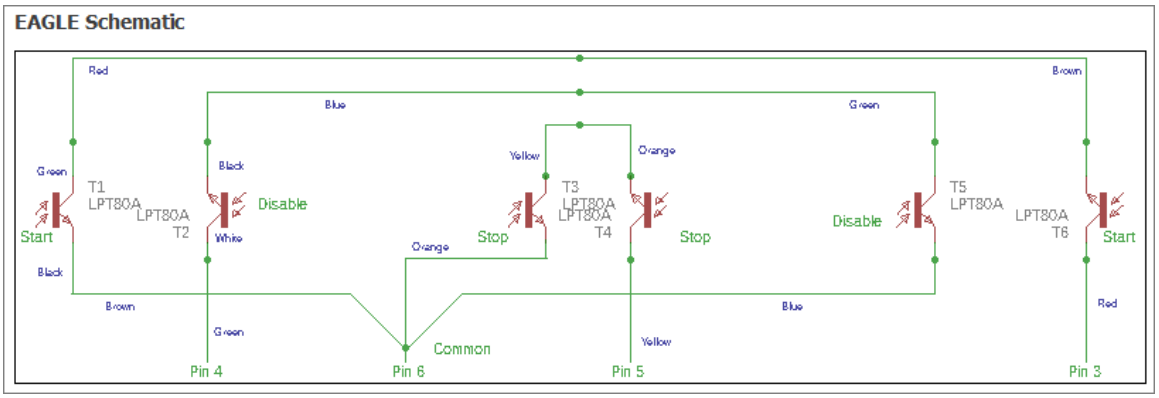
TogglesToLEDs-11-14-16

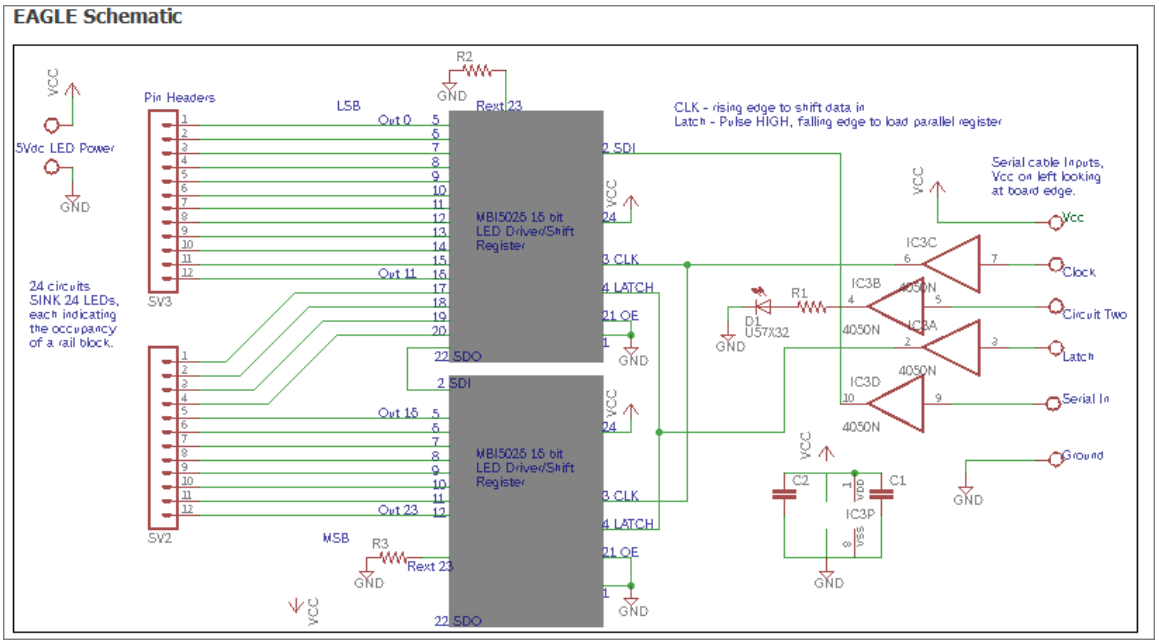
```
/*This file is intended to provide serial data transmission between the Toggle Panel
and the Turnout LED Panel and Turnout Actuators. Both channels of code herein utilize the same pin
configuration, the only exception when the serial pin is changed from Output to Input.
Otherwise all pin definiations and useages are identical.

Three pins are allocated to code control functions: Pin 2 to switch on/off the startup turnout exercise.
Pin 13 will illuminate the internal LED indicating the status of Pin 2.  Pin 3 determines the state of
the scanning process.
Both Pins 2 and 3 are set up so that when pulled LOW they enable their respective processes.
*/
boolean setUp = false;  // make true if you want to run through the turnout exercise?
boolean scanFlag = false;
byte setUpPin = 2;
byte scanPin = 3;
byte setupLED = 13;
byte channel0 = 7;
byte channel1 = 8;
byte clockPin = 9;
byte latchPin = 10;
byte serialPin = 11;
byte channel_A = 0;
byte channel_B = 1;
byte channel_C = 2;
byte channel_D = 3;
unsigned int dataReadIn = 0;
unsigned int previousData = 0;
unsigned int dataTemp;
int z = 0;
// Code run during 'setup()' will run only once at the beginning and never again unless 'reset'.
void setup() {
  //configure pin2 as an input and enable the internal pull-up resistor to test for 'setUp'.
  pinMode(setUpPin, INPUT_PULLUP);
  pinMode(scanPin, INPUT_PULLUP);
  pinMode(setupLED, OUTPUT);
  int sensorVal = digitalRead(setUpPin);
  if (sensorVal == HIGH) {
    digitalWrite(setupLED, LOW); setUp = false;
  } else {
    digitalWrite(setupLED, HIGH); setUp = true;
  }
  sensorVal = digitalRead(scanPin);
  if (sensorVal == HIGH) {
    scanFlag = false;
  } else {
    scanFlag = true;
  }
  Serial.begin(9600);
  Serial.println(".........TogglesToLEDs-11-14-16.........................");
// Set all used pins to OUTPUT (serialPin will be set to INPUT to read in toggles.
```

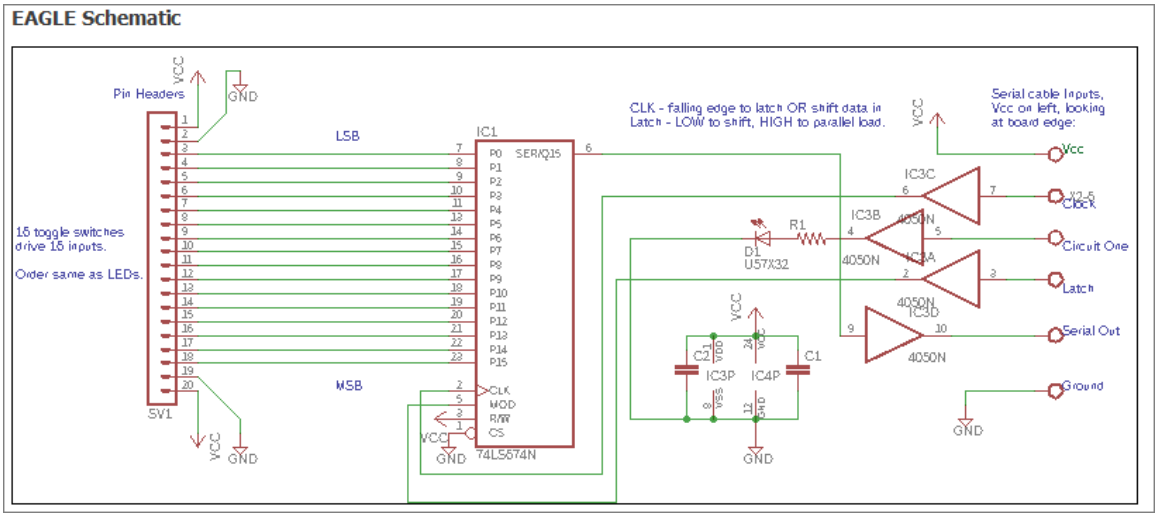
Occupancy_LEDs_01

```
/* This code sends out three consecutive 8 bit bytes to the
 *  Occupancy LED panel.  The bytes are labeled Low, Medium and High.
 *  The order: first bit out is the LSB of the Low byte, and so on,
 *  to the MSB of the High byte being the last bit out. Ergo: LSB first.
 *  The bytes received from the Occupancy detection hardware should foll
 *  the same pattern.
 */
  int i = 0;
  byte dataWordLow = 171;
  byte dataWordMedium = 226;
  byte dataWordHigh = 157;
  int A = 7;
  int B = 8;
  int Clk = 9;
  int Latch = 10;
  int SerialOut = 11;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial.println(".......Occupancy_LEDs_01.........................");
  for (int i=A; i<=SerialOut; i++){
  pinMode(i, OUTPUT);
  }
  digitalWrite(A,HIGH);
  digitalWrite(B,HIGH);
  digitalWrite(Clk,LOW);
  digitalWrite(Latch,LOW);
  digitalWrite(SerialOut,LOW);
}
void loop() {
  // put your main code here, to run repeatedly:
  delayMicroseconds(10);
  digitalWrite(A,LOW);
  digitalWrite(B,LOW);
  delayMicroseconds(10);
  sendOut( dataWordLow );
  sendOut( dataWordMedium );
  sendOut( dataWordHigh );
  latchIt();
  digitalWrite(A,HIGH);
  digitalWrite(B,HIGH);
  delay(10);
}
// Routine to clock the data in each bit...
void clockIt(){
    delayMicroseconds(5);
    digitalWrite(Clk,HIGH);
    delayMicroseconds(5);
    digitalWrite(Clk,LOW);
```

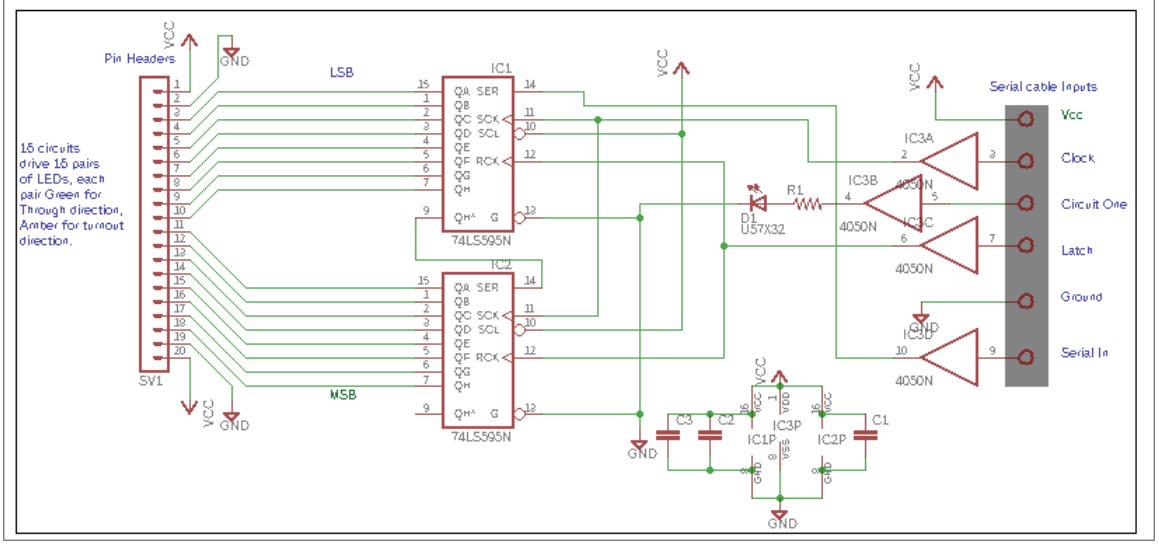
Now for some examples of the Electrical Schematic Diagrams that I created in a Drawing package called EAGLE, available for free download at www.eagle.com.



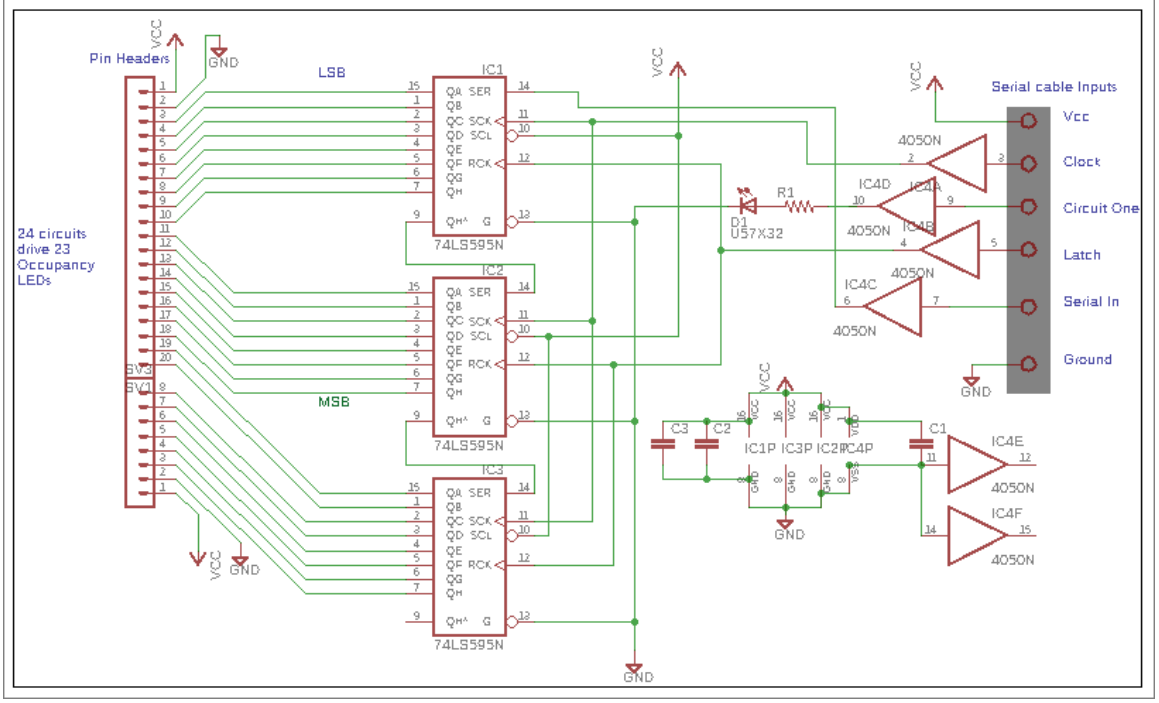


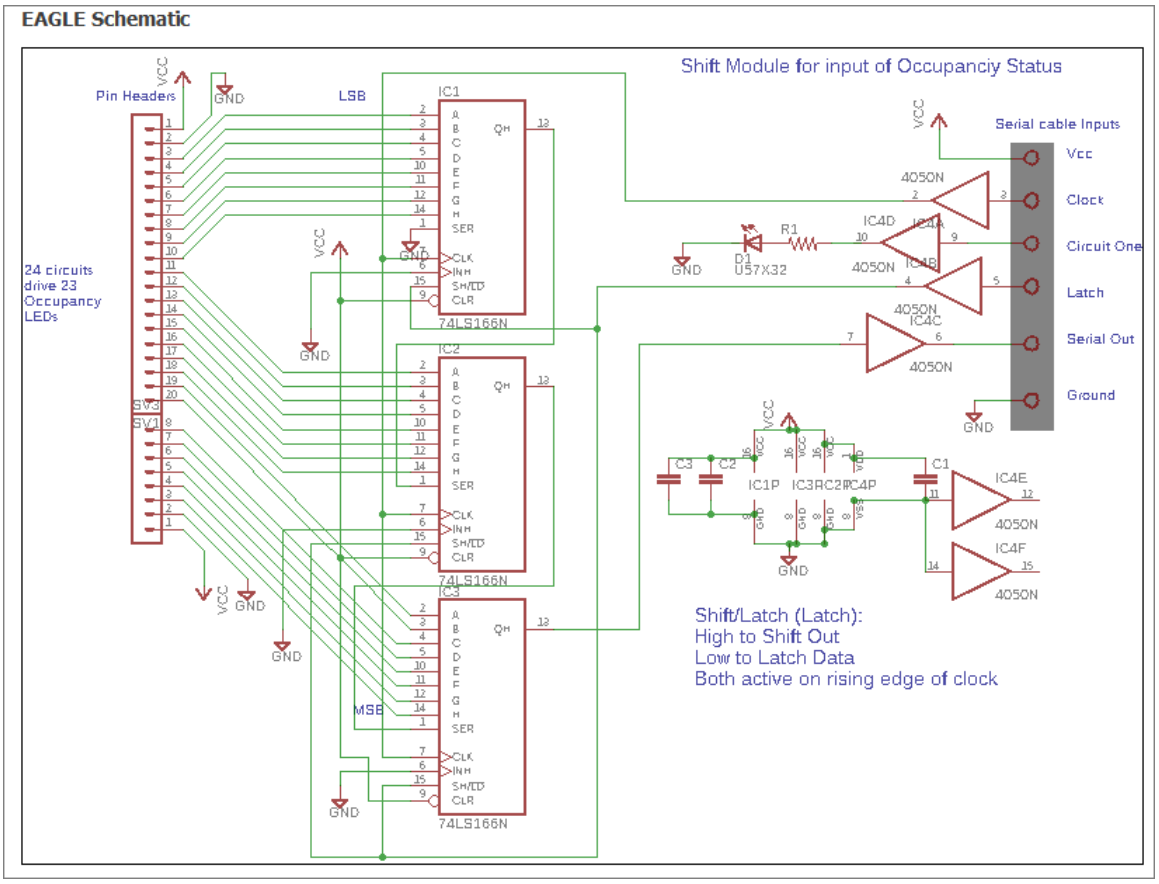


EAGLE Schematic



EAGLE Schematic





...th..th..that's all folks!