Model Railroad System

2.2.2

Generated by Doxygen 1.9.1

Chapter 1

Preface

This document outlines using an Arduino Uno microprocessor board to control upto eight signals with up to eight lamps (LEDs) per signal.

Chapter 2

Introduction

I will be building an interlocking plant module with 5 two-headed signals.

To drive all of these signals I will be using an Arduino and a Max72XX Led driver (see http://playground.arduino.cc/Main/MAX72XXHardware for more information about general uses for the Arduino and the Max72XX chips). This article describes the hardware involved, the firmware (software on the Arduino) and the host computer software (using the Dispatcher program from my Model Railroad System).

2.1 The layout module

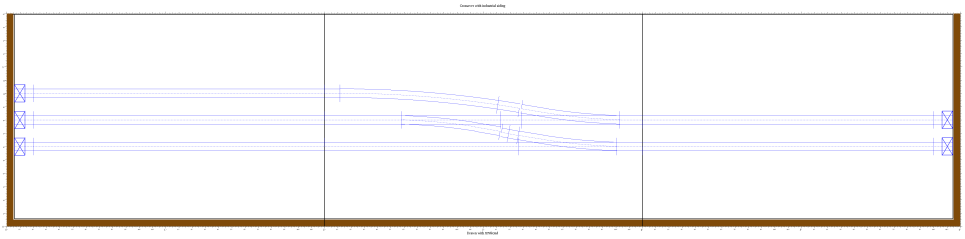


Figure 2.1 Crossover with siding

The layout module is a simple double track main line with a single crossover and an industrial siding, as shown here. There will be a two-track signal bridge (Oregon Rail Supply #151) at the east (right) end of the interlocking plant and a three-track signal bridge (Oregon Rail Supply #154, cut down to three tracks) at the west (left) end of the interlocking plant. At the east end will be a 3 over 2 on track 1 (upper/north main line) and a 3 over 3 on track 2 (lower/south main line). At the west end will be a 1 over 3 on the siding exit, a 3 over 3 on track 1 (upper/north main line) and a 3 over 1 on track 2 (lower/south main line).

2.2 Hardware being used

I will be using Oregon Rail Supply signal bridges, one 2-track (#151) and one 4-track (#154, cut down to 3-tracks). I will be using 2mm x 1.25mm chip LEDs (Mouser part numbers 720-LGR971-KN-1,

720-LYR976-PS-36, and 720-LSR976-NR-1) on small circuit boards to light these signals. There will be an Arduino Uno (Mouser part number 782-A000066) and a home built board (based on the Arduino Playground circuit) containing a Max7221 (Mouser Parts: Mouser Project). This manual describes how I built these signals and how I will control them from my Linux computer, using a CTC panel created with my Model Railroad System Dispatcher program. A ZIP archive containing the PCB Layout/assembly files is available in the file SignalDriverMax72xx.zip in the same folder as this PDF file.

Chapter 3

Signal Driver board

The Signal Driver board is assembled on a piece of "strip board", specifically a 3.5 inch by 2.5 inch piece cut from a BusBoard Prototype Systems BPS-MAR-ST6U-001 (included in the Mouser project).

After cutting this piece from the board some of the copper foil needs to be carefully removed. This is done with a sharp hobby knife and a soldering iron is used to heat the copper to make it easy to peel. The PCB Layout/assembly Zip file includes a PostScript file named SignalDriverMax72xx.back.ps which is an actual sized drawing of what the foil should look like. Here is a side-by-side view of an actual board and the SignalDriverMax72xx.back.ps drawing:

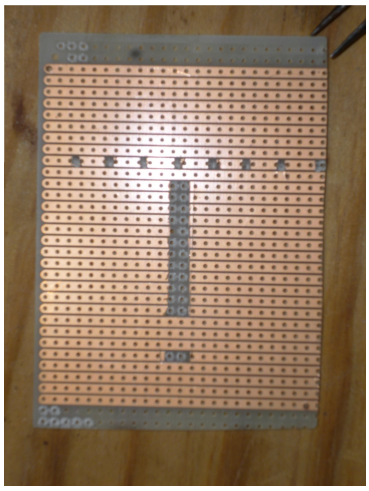


Figure 3.1 Photo of the Signal Driver circuit board (foil side)

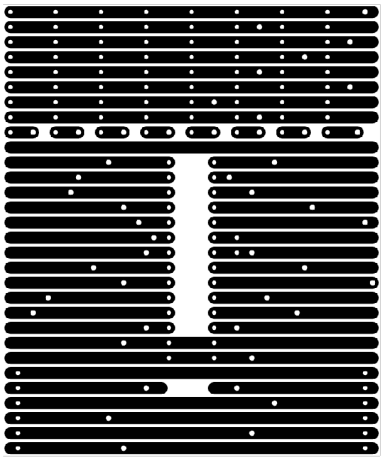


Figure 3.2 Signal Driver Foil side PCB layout

I cut the board to have two strip rows above and below the foil layout to provide a place to drill mounting holes that would not interfere with the circuit elements.

The next step is to run the vertical connections, using solid hookup wire. I used a different color for each "layer". Staring with layer group2 (ground) in black.

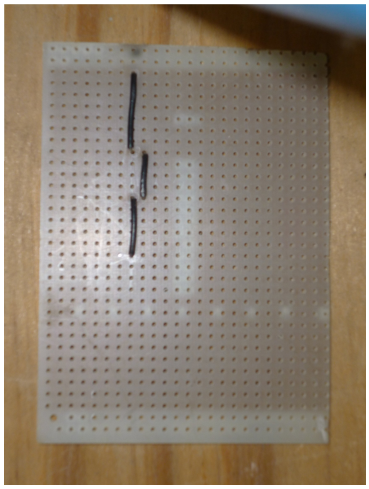


Figure 3.3 Photo of group2 (ground) wires in black

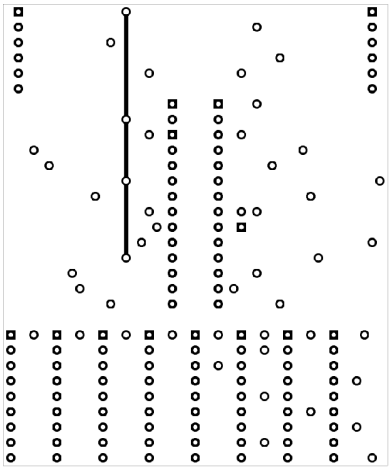


Figure 3.4 Group2 (ground) PCB Layout

Then layer group3 (power) in red.

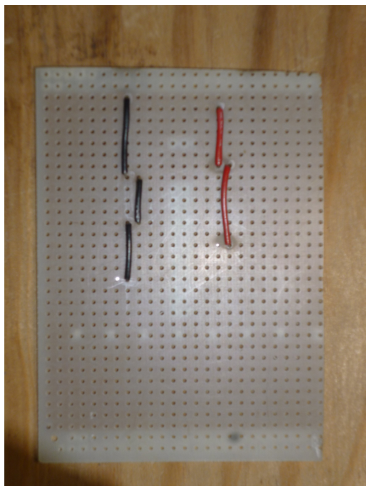


Figure 3.5 Group3 (power) with red wire

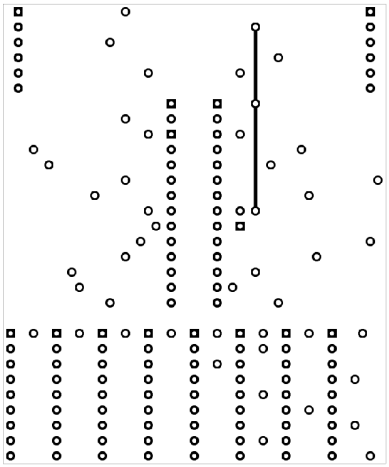


Figure 3.6 SignalDriverMax72xx_group3.png

Then layer group4 (signal1) in yellow.

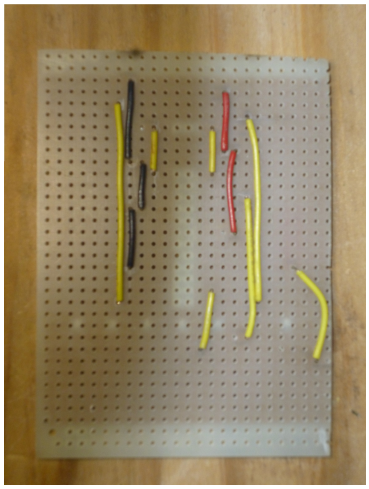


Figure 3.7 Group4 (signal1) in Yellow

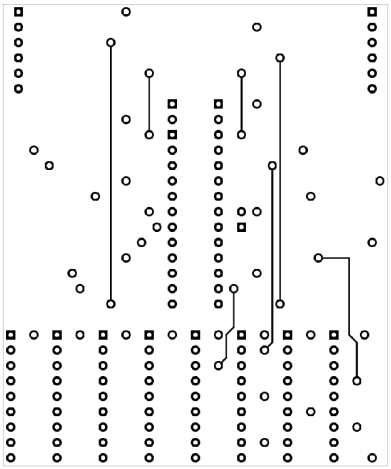


Figure 3.8 Group4 (signal1) PCB Layout

Then layer group5 (signal2) in green.

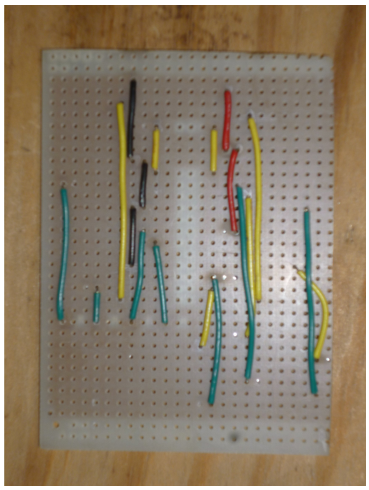


Figure 3.9 Group5 (signal 2) in green

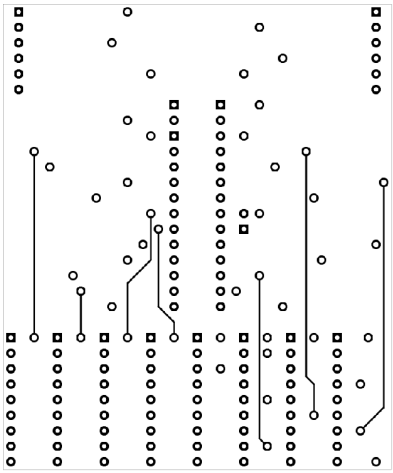


Figure 3.10 PCB layer group5 (signal2)

Then layer group6 (signal3) in blue.

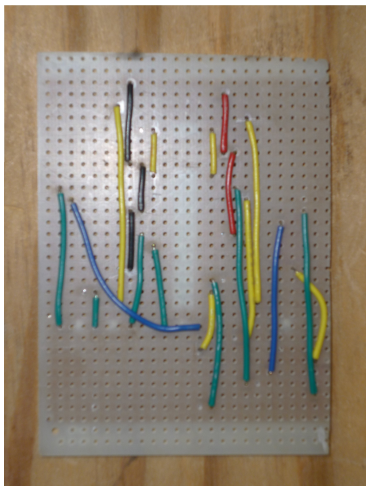


Figure 3.11 Photo of group6 (signal3) in blue.

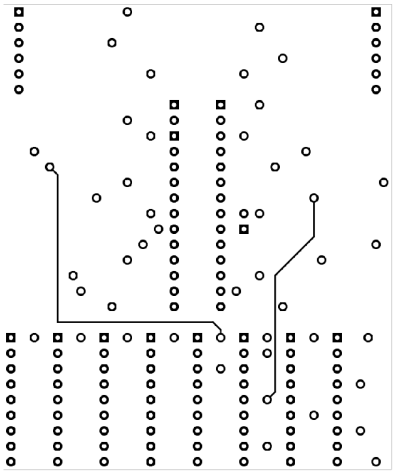


Figure 3.12 PCB layer group6 (signal3)

Then layer group7 (signal4) in white.

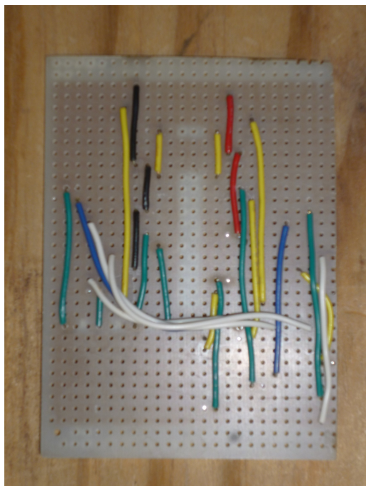


Figure 3.13 Photo of group 7 (signal 4) in white

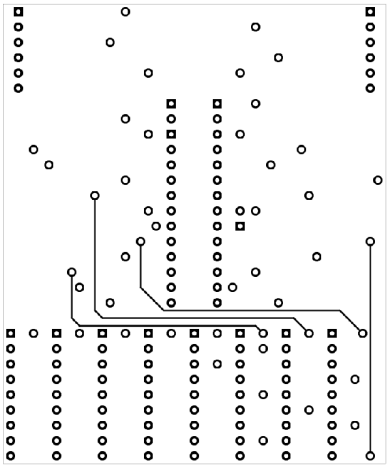


Figure 3.14 PCB layout of group7 (signal4)

Finally, the headers, IC socket, and the passive components are installed. There is a trick to installing the IC socket and the headers: solder only one pin, then while pushing the socket or header against the board, reheat the solder to make it re-flow. This should cause the socket or header to snap squarely to the board. You might have to push some of the wires to one side to install the IC socket and the 9-pin headers, but if you were careful about routing the wires, this should not be a problem. The resistor needs to have one of its leads bent 180 degrees to allow it to be mounted on end. The unbent pin should go next to the where the red wires are installed. C2 (the larger electrolytic capacitor) is polarized. The negative lead (the shorter one next to the stripe) goes towards the IC socket. The resistor and the capacitors should be mounted as tightly to the board as possible. You can solder one lead and the reheat the solder to carefully position them tight and square.

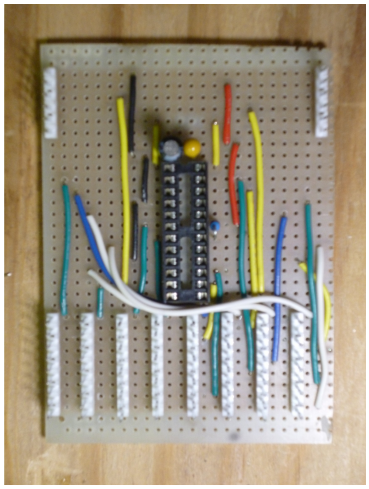


Figure 3.15 Photo of front assembly

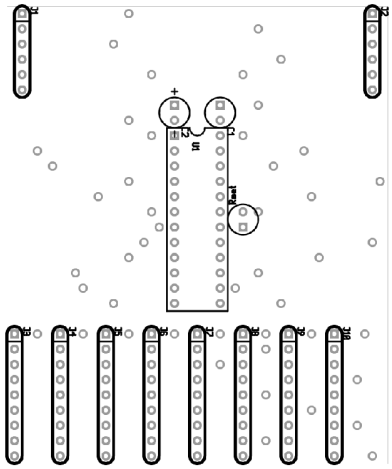


Figure 3.16 PCB layout of front assembly

Here is another view of the completed circuit board. This angle view gives a better view of the assembly. The next step is to carefully inspect the board, looking closely with a magnifier looking for solder bridges or bad solder joints.

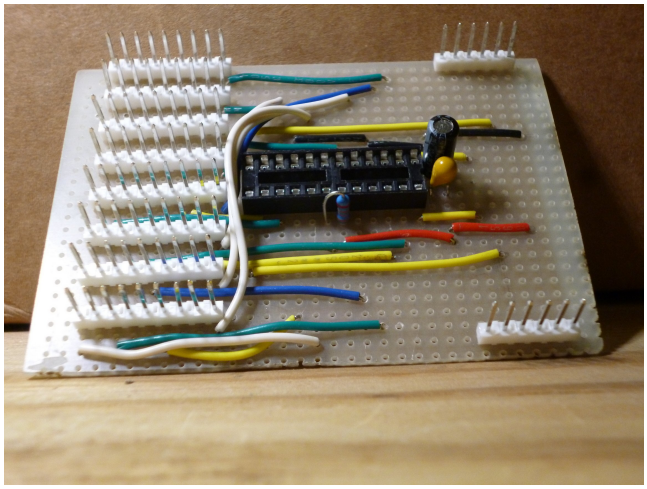


Figure 3.17 Photo of front assembly at an angle

Then you can use an Ohmmeter (or a multimeter in Ohmmeter mode) to check the circuit paths from each pin of the IC socket. The text file named `SignalDriverMax72xx.pcb.u1` in the PCB Layout/assembly zip-file contains a listing of the connections to each pin of the IC socket. Here is a version of the front assembly diagram with the pin numbers indicated.

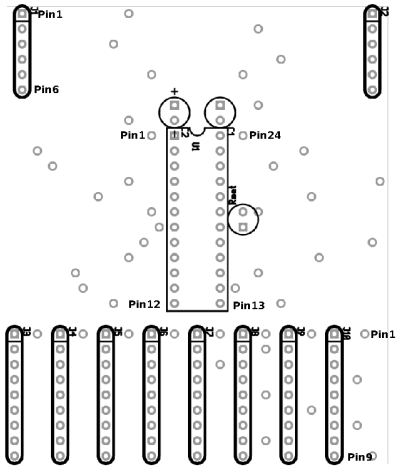


Figure 3.18 PCB layout of the front assembly with pin numbers

Chapter 4

Connecting the Signal Driver Board

The Signal Driver board is connected with a home made connector cable.

The cable is a six conductor ribbon cable (DigiKey part number MC06G-25-ND). One end of the cable is attached to a 6-pin .1 inch (2.54mm) IDC header plug and the other end connected to a "plug" made from a small piece of strip-board and a couple of pieces of .1 inch (2.54mm) pitch breakaway headers, 2 pins at the power and ground end and 3 pins at the digital I/O end. Some of foil should be removed (this prevents possible shorts). The cable is soldered to the foil side and the headers are mounted on the component side. The cable is secured with a wire tie and some hot glue. This connector fits on top of the Arduino Uno as shown. Make sure the pins are in the correct header position!

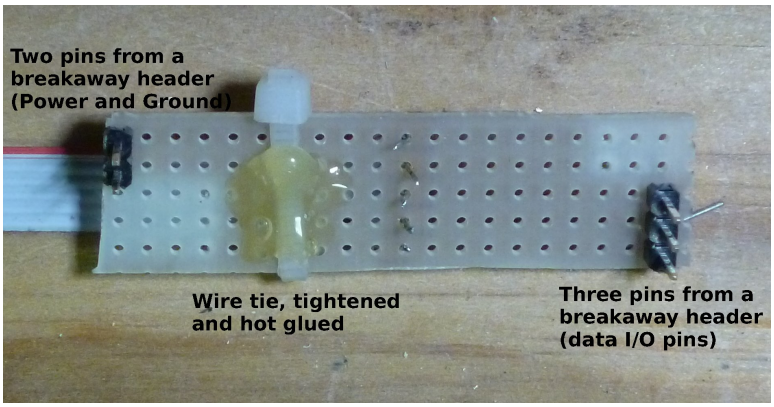


Figure 4.1 Component side of Uno connector

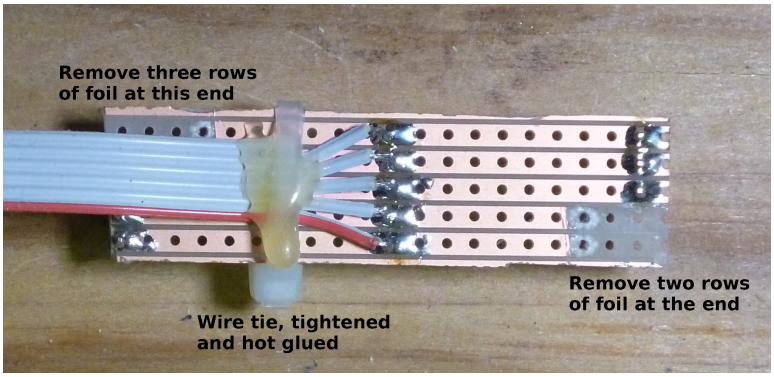


Figure 4.2 Uno Connector, Foil Side

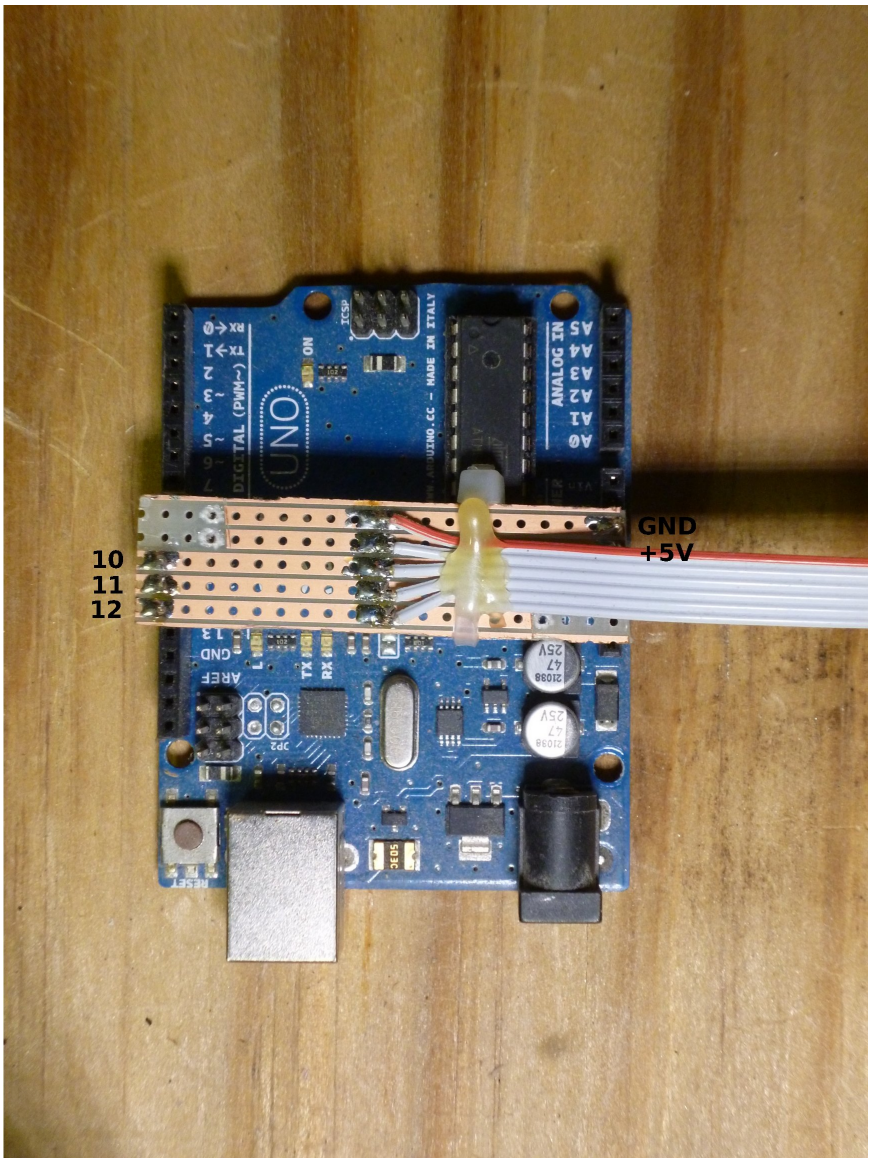


Figure 4.3 Connector on Uno

The IDC plug is attached to the other end and I used an Exacto Knife to press the wires into the IDC slots. Mouser sells a `$30 tool` to do this, if you prefer.

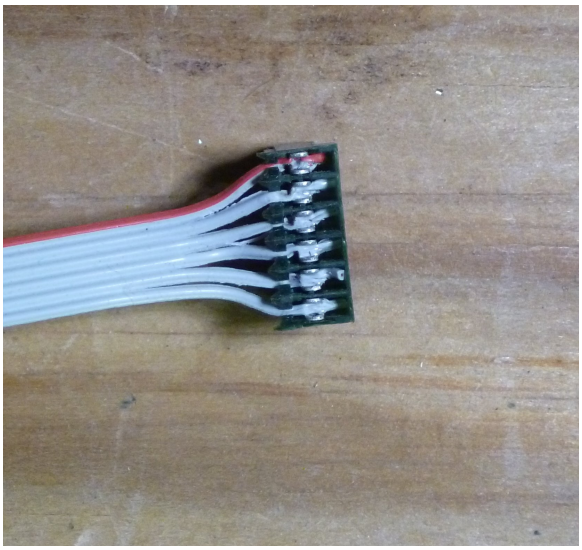


Figure 4.4 Connector Plug, Installed

Figure 4.5 Installing Connector Plugs

Chapter 5

Signal Driver board cables

Nine conductor ribbon cables (`DigiKey part number MC09G-25-ND`) are used to connect between the Signal Driver Board and the signals.

One end gets a 9-pin header plug and the other end gets a small circuit board with small screw terminals. The actual LEDs in the signals are connected to wire wrap wire, but wire wrap wire is too delicate to run long distances, but is needed to fit in the small brass tubes the signal targets are mounted on. Once under the layout bench-work, the wire wrap wire gets connected with screw terminals to the much more robust ribbon cable. The small circuit boards are again made from pieces of strip-board, with nine strips, eleven holes long. After cutting the boards, some of the copper is removed and four 1/8 inch (3.5mm) holes are drilled.

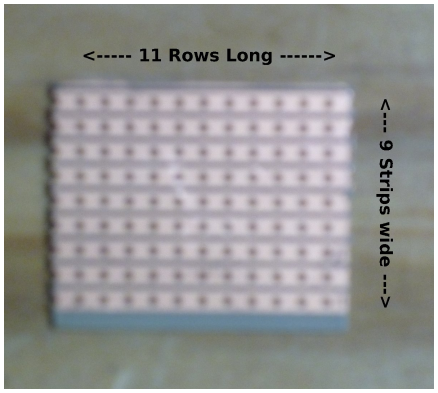


Figure 5.1 Signal Connector Board, bare

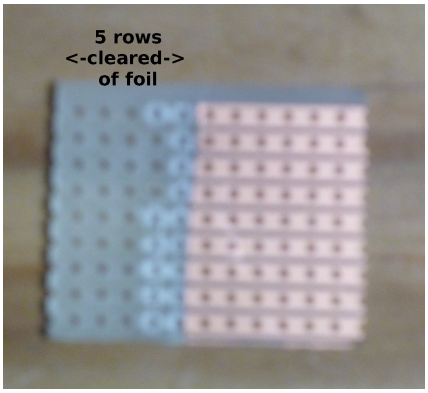


Figure 5.2 Signal Connector Board, copper removed

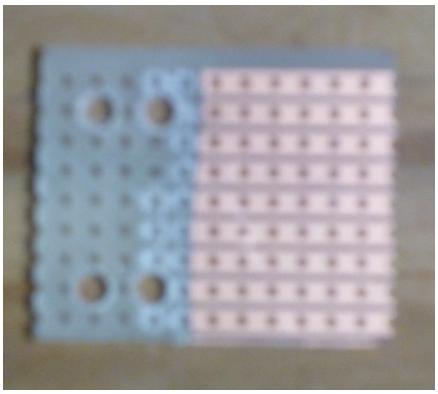


Figure 5.3 Signal Connector Board, holes drilled

Next the screw terminal blocks are soldered to the board (this is actually a 4 position terminal block with a 5 position terminal block next to it – Mouser does not stock the 9 position version of these terminal blocks).
Then the conductors at one end of the cable is zipped back about a 3/4 inch (18mm) and about 1/4 inch (6mm) of the ends are stripped and tinned. These tinned conductors are then fed into holes in the circuit board and soldered. Finally a wire tie is used to secure the cable and act as a strain relief.

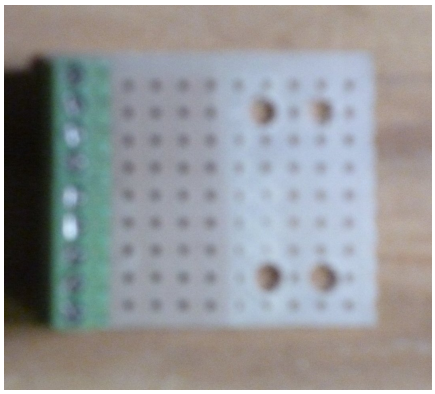


Figure 5.4 Signal Connector Board, terminal blocks installed

Figure 5.5 Signal Connector Cable, wires stripped and tinned

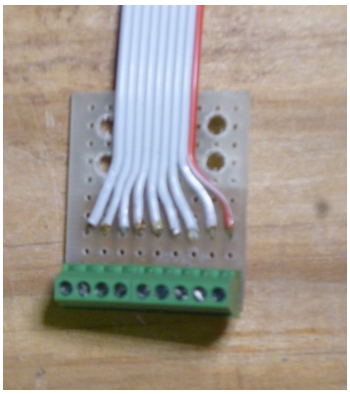


Figure 5.6 Signal Connector Board, cable soldered on

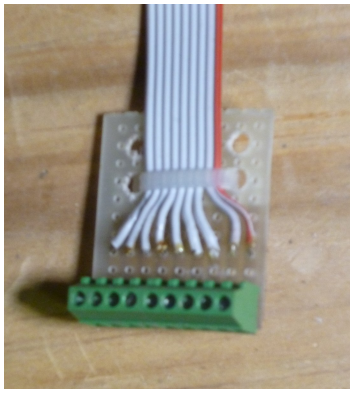


Figure 5.7 Signal Connector Board, cable secured with wire tie

Finally, a 9 position header plug is installed on the other end of the cable.

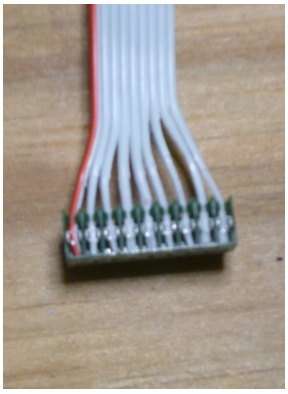


Figure 5.8 Signal Connector Cable, header plug installed

About cable lengths: each cable should be long enough to reach from where the signal wire bundles emerge under the layout to where the Signal Driver Board is mounted. It is always better to cut the ribbon cables longer than needed since excess cable can be managed in various ways, but a short cable is not useable. The length of the wire wrap wires should be as short as you can get away with, which means the terminal block ends should be as close as possible to the place where the signal wire bundles emerge under the layout.

Chapter 6

Assembling signal targets

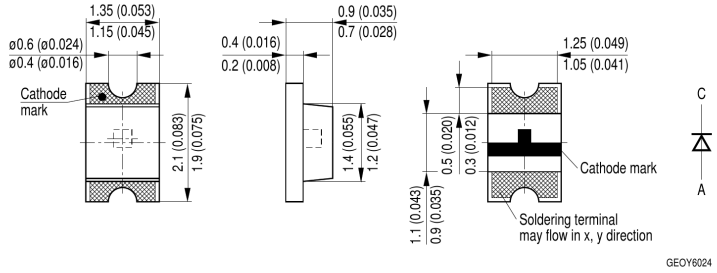
The next step is to assemble the signal targets.

I used 2mm x 1.25mm chip LEDs, made by Osram and sold by Mouser (part numbers Green: 720-LGR971-↩ KN-1, Yellow: 720-LYR976-PS-36, and Super Red: 720-LSR976-NR-1). These are $.10 each in single quantities and the price goes down to about five and a half cents each in quantities of 100. If you decide to use chip LEDs instead of regular 3mm LEDs with leads, be sure to get some extras, because you will likely loose one or two.

Figure 6.1 Photo of a typical chip LED

Version 1.0 **LG R971**

Package Outline 9) page 19
Maßzeichnung 9) Seite 19



Approximate Weight:	3 mg
Gewicht:	3 mg

2012-11-06 10 OSRAM Opto Semiconductors

Figure 6.2 Outline drawing of a typical chip LED

These devices come on a tape carrier. This is something normally meant to go in a robot feeding device that places the chips on circuit boards in robotic factory. To handle these devices by hand you will need to make a tool to hold them. I made a tool from a standard round toothpick. I used a razor saw to cut one of the end points off, sanded cut flat and applied a dab of Detail Tack (available from Micro-Mark for $7.95). This stuff dries clear and remains tacky (sticky). This lets you pick up the chips from their tape carrier and hold them in place as you re-flow the solder to secure them to the circuit board.

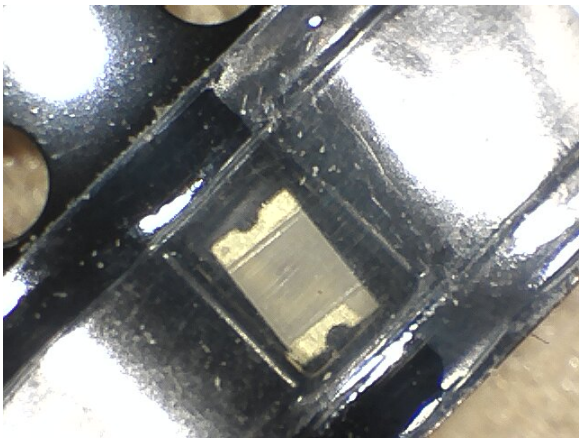
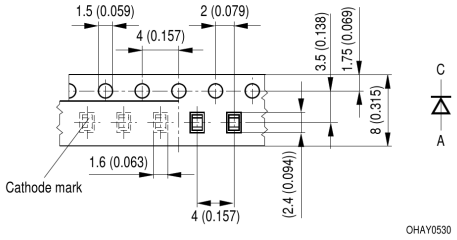


Figure 6.3 Signal Chip LED In Carrier

Method of Taping [9] *page 19*
Gurtung [9] *Seite 19*



OSRAM
Opto Semiconductors

Figure 6.4 Chip Tape specs (page 13 of the data sheet)

You will also need a supply of wire wrap wire in a number of colors (this is available from DigiKey), a supply of cut off resistor leads (or really any small solid bare wired cut into pieces about 1/2 to 3/4 inches long), and a piece of Strip-board two foil strips wide (.2 inches / 5mm) that is long enough to make little circuit boards for your targets – you will need 6 holes for 3 color targets, 4 holes for 2 color targets, and 2 holes for single color targets. I also used a block of foam as a work surface, since it let me push the resistor leads into it.

The first step is to remove some of the foil. One side is the common side (cathode end) and the other is for one-of connections (anode). The common / cathode is connected with a resistor lead that will be soldered to the brass tube the signal targets will be mounted to. The anode side will be connected with wire wrap wire, one per chip and color coded (I used green, yellow, and red for the upper head and blue, white, and black for the lower head). Once the foil bits have been removed, strip and feed the wire wrap wire on one side and push the resistor lead through a hole on the other side and then solder the wires. Be sure to spread a thin layer of solder down the length of the common side.

Figure 6.5 Signal Target And Strip Board

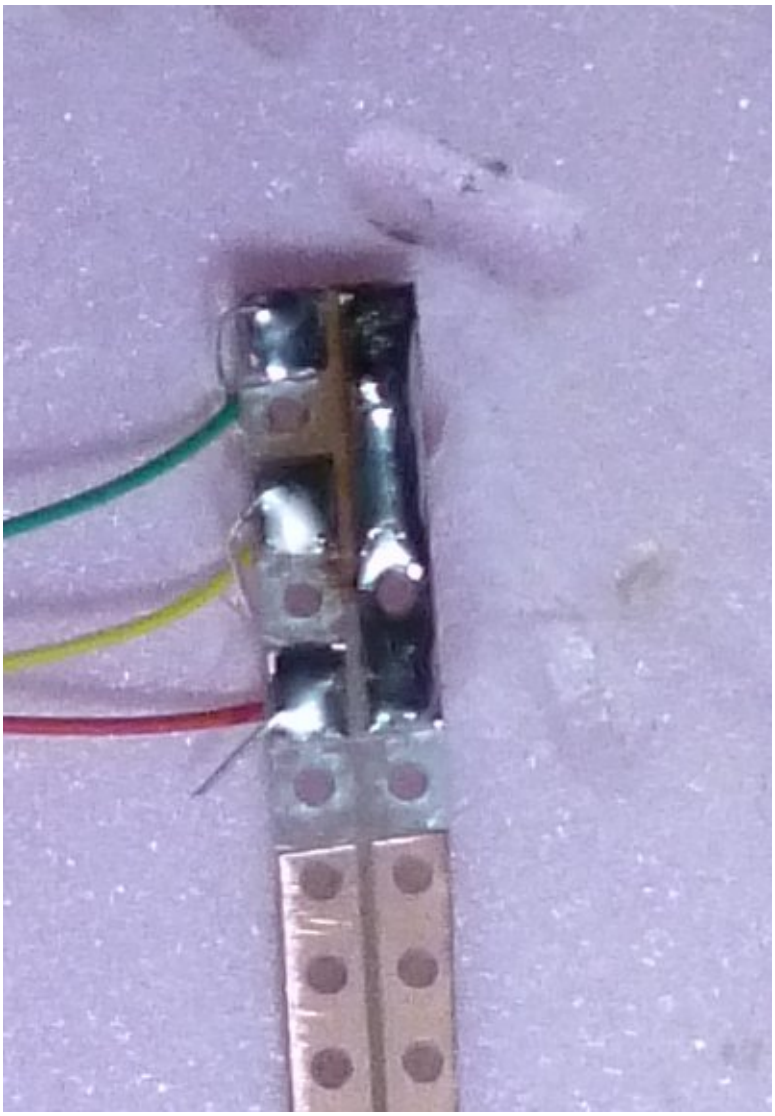


Figure 6.6 Signal Circuit Board, Wires Soldered

Now we can solder on the LED chips. The carrier tape has a clear cover strip over the top. *Carefully* peel this back (a hobby knife can help with this). Only peel back *one* chip at a time. Once the cover strip has been peeled back the chips can very easily bounce out and promptly vanish! Or at least become disorientated... It is important to remember that the side of the tape with the holes is the cathode end, so you should keep the hole side oriented to the same side as the common side of the circuit board. Once you have peeled the cover off one chip, use your pickup tool (toothpick with flattened end with detail tack on it) to pick up the chip and being careful not to twist the toothpick position the chip on the circuit board. Using a small electronics soldering iron briefly reheat the solder on each side to secure the chip. You will want to wait for the solder to cool on the first side before reheating the second side. Be sure to inspect your work and test the chip before continuing on to the next chip. You will probably want to do all of chips of each color before moving on to the next color, since there is no obvious way to tell which color a chip is.

Figure 6.7 Signal Chip LED Soldered

Once the chips have been soldered to the circuit board, each target's circuit board can be cut off the strip and that target's circuit board can be glued to the back of the target with a CA (superglue) adhesive. Finally, the circuit board can be covered with opaque black paint or LIQUID ELECTRICAL TAPE (available from Micro-Mark). The targets can now be assembled with their brackets to the signal masts (3/32 inch brass tubing). Route the wire wrap wires inside the brass tubing (use a 1mm drill to drill holes near where the wires come off the circuit boards). The common lead (the resistor lead wire) can be soldered to the brass tubing and an additional wire wrap wire soldered to the end of the tube.

Chapter 7

Programming the Arduino

The C++ source code for the Arduino is in the file SignalDriverMax72xx.ino. It uses the *LedControl* library, so the code starts by including the header file:

```
#include <LedControl.h>
```

Then since it is using scanf() and various string function, it includes stdio.h and string.h:

```
#include <stdio.h>
#include <string.h>
```

Then it allocates a **LedControl** object:

```
/** Create a new LedControl.
 * We use pins 12,11 and 10 for the SPI interface
 * With our hardware we have connected pin 12 to the DATA IN-pin (1) of the first MAX7221
 * pin 11 is connected to the CLK-pin(13) of the first MAX7221
 * pin 10 is connected to the LOAD-pin(12) of the first MAX7221
 * We will only have a single MAX7221 attached to the arduino
 */
LedControl lc1=LedControl(12,11,10,1);
```

Next the setup function initializes the MAX72xx chip and sends an announcement to the host computer over the serial port:

```
/** The setup function initializes the MAX72xx chip and sends an
  * announcement to the host computer over the serial port.
  */
void setup() {
  /* Set max intensity */
  lc1.setIntensity(0,15);
  /* Set all signals to 'dark' (no lights on). */
  lc1.clearDisplay(0);
  /* Wake up display. */
  lc1.shutdown(0,false);
  /* Announce ourself to the host */
  Serial.begin(115200);
  Serial.println("Signal Driver Max72XX 0.1");
  Serial.print("\n»");
  Serial.flush();
  test = false;
}
```

Next the signal aspects are defined. These values assuming that the signal heads are wired bottom to top, with the LEDs wired from bit 0 to 5 as: lower red, lower yellow, lower green, upper red, upper yellow, and upper green. (See Wiring the signals. Wiring the signals below.)

```
/* Signal Aspects */
/** Red over Red (Stop) */
#define R_R B00001001
/** Red over Yellow (Approach Limited) */
#define R_Y B00001010
/** Red over Green (Slow Clear) */
#define R_G B00001100
/** Yellow over Red (Approach) */
```

```
#define Y_R B00010001
/** Green over red (Clear) */
#define G_R B00100001
/** Dark (all lights off) */
#define DARK B00000000
```

Next we have a helper function to convert from an aspect name sent from the host computer to the Arduino.

```
/** Test for each signal aspect string and when a match
  * Occurs, return the corresponding bit pattern.
  * @param aspectname The aspect text sent from the host.
  * @returns The bit pattern to display the selected aspect.
  */
int GetAspectBits(const char *aspectname) {
  if (strcasecmp("R_R",aspectname) == 0) return R_R;
  else if (strcasecmp("R_Y",aspectname) == 0) return R_Y;
  else if (strcasecmp("R_G",aspectname) == 0) return R_G;
  else if (strcasecmp("Y_R",aspectname) == 0) return Y_R;
  else if (strcasecmp("G_R",aspectname) == 0) return G_R;
  else if (strcasecmp("DARK",aspectname) == 0) return DARK;
  else return -1;
}
```

Next comes the main loop function. Here we read a one line command from the host computer and decide what to do. There are only three commands defined:

- One to turn all of the LEDs off.
- One to set the aspect of one signal.
- And a final command to initiate a test sequence.

```
/** The main loop function.  Here we read a one line command from
  * the host computer and decide what to do.  There are only three commands
  * defined:
  *   - One to turn all of the LEDs off.
  *   - One to set the aspect of one signal.
  *   - And one to initiate a test sequence.
  */
void loop() {
  char buffer[256]; /* Command line buffer. */
  char p_buffer[32]; /* Test prompt buffer. */
  int  len;         /* Line length. */
  char unused;
  int n;

  /* check if in test mode. */
  if (test) {
    /* Display the current test pattern on the current signal. */
    lc1.setRow(0, i_digit, i_bits);
    /* Print the current signal number and the current test pattern. */
    sprintf(p_buffer,"\n%d:%02x»", i_digit, i_bits);
    Serial.print(p_buffer);
    Serial.flush();
    delay(1000);   /* One second sleep. */
    /* Compute the next pattern and/or signal number. */
    switch(i_bits) {
     case B00000000: /* Last pattern. Next signal number. */
        if (i_digit >= e_digit) { /* Last signal number, go to first signal number. */
         i_digit = s_digit;
        } else { /* Next signal number. */
         i_digit++;
        }
        i_bits  = B00000001; /* First pattern. */
        break;
     case B11111111: /* If at all on, go to all off. */
        i_bits = B00000000;
        break;
     case B10000000: /* If at top LED, go to all on. */
        i_bits = B11111111;
        break;
     default: /* Otherwise, shift left one bit. */
        i_bits = i_bits « 1;
        break;
    }
```

```
}
/* If there is serial data available... */
if (Serial.available() > 0) {
  /* If testing, stop the test and clear the signal. */
  if (test) {
    test = false;
    lc1.setRow(0, i_digit, B00000000);
  }
  /* Read a line from the serial port (USB connection
     from the host computer. */
  len = Serial.readBytesUntil('\r',buffer,sizeof(buffer)-1);
  if (len <= 1) {
    /* Reissue command prompt. */
    Serial.print("\n»");
    Serial.flush();
    return;
  }
  buffer[len] = '\0';
  switch (toupper(buffer[0])) {
    case 'D': /* Clear all signals to Dark. */
      lc1.clearDisplay(0);
      break;
    case 'S': /* Set one signal. */
      {

        char aspect[10];
        int  signalnum, aspectbits;
        if (sscanf(buffer,"%c %d %9s",&unused,&signalnum,aspect) != 3) {
          Serial.println("\nSyntax error (Set command)!");
        } else {
          /* Parse aspect string. */
          aspectbits = GetAspectBits(aspect);
          /* Check for legal aspect string. */
          if (aspectbits < 0) {
            Serial.println("\nSyntax error (Bad aspect)!");
          /* Check for legal signal number. */
          } else if (signalnum >= 0 && signalnum < 8) {
            lc1.setRow(0, signalnum, (byte) aspectbits);
          } else {
            Serial.println("\nSyntax error (Bad signal number)!");
          }
        }
        break;
      }
    case 'T': /*
               * Test mode. Test one or more signals, lighting LEDs in a sequence of patterns:
               * First one LED, from bottom to top, then all on, than all off.  Repeat with the
               * next signal.  After the last signal in the test, start over with the first
               * signal in the test.  Repeat forever or until another command is sent.
               */
      /* Parse command, getting the number of conversions.
       * One conversion means no arguments -- test all eight signals.  Two conversions means one
       * argument -- test one signal. Three conversions means two arguments -- test a range of
       * signals. */
      n = sscanf(buffer,"%c %d %d",&unused,&s_digit,&e_digit);
      /* sprintf(p_buffer,"\n*** n = %d",n);
      Serial.println(p_buffer);*/
      /* Fan out on conversion count. */
      switch (n) {
        case 1: /* No arguments -- test all signals. */
          s_digit = 0;
          e_digit = 7;
          i_digit = s_digit;
          i_bits = B00000001;
          test = true;
          break;
        case 2: /* One argument -- test one signal. */
          e_digit = s_digit;
          if (s_digit < 0 || s_digit > 7) {
            Serial.println("\nSyntax error (Bad signal number)!");
            break;
          }
          i_digit = s_digit;
          i_bits = B00000001;
          test = true;
          break;
        case 3: /* Two arguments -- test a range of signals. */
          if (s_digit < 0 || s_digit > 7) {
            Serial.println("\nSyntax error (Bad signal number)!");
            break;
          }
```

```
            if (e_digit < 0 || e_digit > 7) {
              Serial.println("\nSyntax error (Bad signal number)!");
              break;
            }
            i_digit = s_digit;
            i_bits = B00000001;
            test = true;
            break;
          default: /* Something else -- spit out an error message. */
            Serial.println("\nUnknown command!");
            break;
        }
        break;
      default:
         Serial.println("\nUnknown command!");
         break;
    }
    /* Reissue command prompt. */
    Serial.print("\n»");
    Serial.flush();
  }
} /* End of Main loop */
```

7.1 Wiring the signals.

I used this color coding for the signal LEDs when I wired them:

Green The upper target head's green LED (uppermost LED of the upper target).

Yellow The upper target head's yellow LED (middle LED of the upper target).

Red The upper target head's red LED (bottom LED of the upper target).

Blue The lower target head's green LED (uppermost LED of the lower target).

White The lower target head's yellow LED (middle LED of the lower target).

Black The lower target head's red LED (bottom LED of the lower target).

Thus the connections to the terminal blocks at the ends of the signal cables are made as shown here. If a target has fewer than three LEDs, then the wires for the missing LEDs are also missing.

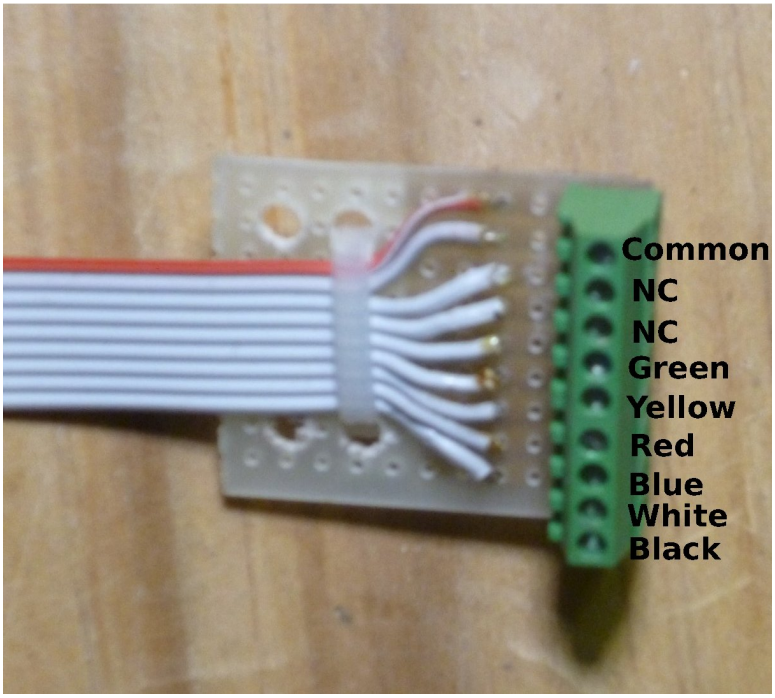


Figure 7.1 Signal Connector Board, Wiring Color Codes

Once you have entered the code and verified that it compiles and uploaded it to the Arduino, you can test the code with the Serial Monitor tool on the Arduino IDE. Be sure to set the baud rate to 115200. You can then type commands into the Serial Monitor tool's send bar, as shown here.

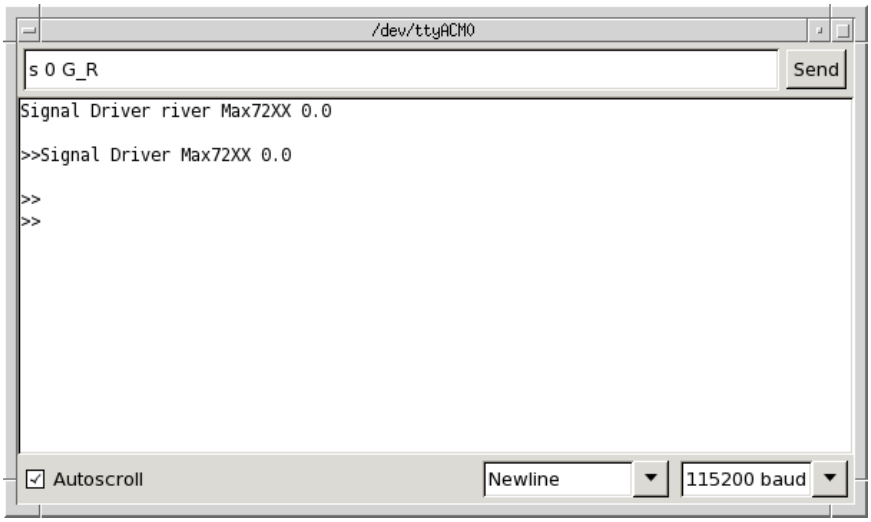


Figure 7.2 Serial Monitor, Test Sketch

Chapter 8

Programming the Host Computer

The host interface to the Ardunio SignalDriverMax72xx is via a virtual serial port over the USB interface.

The host computer sends text commands down the serial port and the Ardunio in turn sends data down its SPI interface to the MAX72XX, which in turn lights up the signal LEDs.

I wrote a simple Tcl SNIT type (OO class) that implements this interface. The Tcl code is in the file SignalDriverMax72xx_Host.tcl. The constructor connects to the Ardunio by opening the virtual serial port. Then signals can then be lit with selected aspects with the instance method `set`, which takes two arguments, a signal number (0 to 7 inclusive) and an aspect string, which is one of:

- g_r (Green over Red – Clear)
- y_r (Yellow over Red – Approach)
- r_r (Red over Red – [Absolute] Stop)
- r_g (Red over Green – Slow Clear)
- r_y (Red over Yellow – Approach Limited)
- dark (all lights off)

There is also an instance method, `dark`, which turns all of the signal LEDs off.

Typical usage:

```
# Load the code
package require SignalDriverMax72xx_Host
# Connect to the Ardunio on /dev/ttyACM0
SignalDriverMax72xx controlpoint1 -portname /dev/ttyACM0
# Define symbolic names for the signals
# East end (Westbound) of Control Point 1 on track 2
set CP1w2 0
# East end (Westbound) of Control Point 1 on track 1
set CP1w1 1
# West end (Eastbound) of Control Point 1 on track 2
set CP1e2 2
# West end (Eastbound) of Control Point 1 on track 1
set CP1e1 3
# West end (Eastbound) of Control Point 1 on siding
set CP1eS 4
# Set all signals to Red over Red
controlpoint1 set $CP1w2 r_r
controlpoint1 set $CP1w1 r_r
controlpoint1 set $CP1e2 r_r
controlpoint1 set $CP1e1 r_r
controlpoint1 set $CP1eS r_r
# Set Track 1 for clear (Green over Red) Eastbound
controlpoint1 set $CP1e1 g_r
# Set Track 2 for clear (Green over Red) Westbound
controlpoint1 set $CP1w2 g_r
```

Chapter 9

Module Index

9.1 Modules

Here is a list of all modules:

Chapter 10

Class Index

10.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 11

Module Documentation

11.1 Ardunio Signal Driver using a MAX72XX

This is the software downloaded to the Ardunio to interface to the MAX72XX LED multiplexer driving the signals.

Macros

- #define R_R B00001001

  Red over Red (Stop)
- #define R_Y B00001010

  Red over Yellow (Approach Limited)
- #define R_G B00001100

  Red over Green (Slow Clear)
- #define Y_R B00010001

  Yellow over Red (Approach)
- #define G_R B00100001

  Green over red (Clear)
- #define DARK B00000000

  Dark (all lights off)

Functions

- void setup ()

  The setup function initializes the MAX72xx chip and sends an announcement to the host computer over the serial port.
- int GetAspectBits (const char *aspectname)

  Test for each signal aspect string and when a match Occurs, return the corresponding bit pattern.
- void loop ()

  The main loop function.

Variables

- LedControl lc1 =LedControl(12,11,10,1)

  Create a new LedControl.
- int s_digit

  Start digit.
- int e_digit

  End digit.
- int i_digit

  Current digit.
- int i_bits

  Current bits.
- boolean test = false

  Flag indicating if we are in test mode.

11.1.1 Detailed Description

This is the software downloaded to the Ardunio to interface to the MAX72XX LED multiplexer driving the signals.

11.1.2 Macro Definition Documentation

11.1.2.1 DARK

```
#define DARK B00000000
```

Dark (all lights off)

Definition at line 68 of file SignalDriverMax72xx.ino.

11.1.2.2 G_R

```
#define G_R B00100001
```

Green over red (Clear)

Definition at line 66 of file SignalDriverMax72xx.ino.

11.1.2.3 R_G

```
#define R_G B00001100
```

Red over Green (Slow Clear)

Definition at line 62 of file SignalDriverMax72xx.ino.

11.1.2.4 R_R

```
#define R_R B00001001
```

Red over Red (Stop)

Definition at line 58 of file SignalDriverMax72xx.ino.

11.1.2.5 R_Y

```
#define R_Y B00001010
```

Red over Yellow (Approach Limited)

Definition at line 60 of file SignalDriverMax72xx.ino.

11.1.2.6 Y_R

```
#define Y_R B00010001
```

Yellow over Red (Approach)

Definition at line 64 of file SignalDriverMax72xx.ino.

11.1.3 Function Documentation

11.1.3.1 GetAspectBits()

```
int GetAspectBits (
            const char * aspectname )
```

Test for each signal aspect string and when a match Occurs, return the corresponding bit pattern.

Parameters

aspectname	The aspect text sent from the host.

Returns

The bit pattern to display the selected aspect.

Definition at line 75 of file SignalDriverMax72xx.ino.

11.1.3.2 loop()

```
void loop ( )
```

The main loop function.

Here we read a one line command from the host computer and decide what to do. There are only three commands defined:

- One to turn all of the LEDs off.
- One to set the aspect of one signal.
- And one to initiate a test sequence.

Definition at line 92 of file SignalDriverMax72xx.ino.

11.1.3.3 setup()

```
void setup ( )
```

The setup function initializes the MAX72xx chip and sends an announcement to the host computer over the serial port.

Definition at line 41 of file SignalDriverMax72xx.ino.

11.1.4 Variable Documentation

11.1.4.1 e_digit

```
int e_digit
```

End digit.

Definition at line 29 of file SignalDriverMax72xx.ino.

11.1.4.2 i_bits

```
int i_bits
```

Current bits.

Definition at line 33 of file SignalDriverMax72xx.ino.

11.1.4.3 i_digit

```
int i_digit
```

Current digit.

Definition at line 31 of file SignalDriverMax72xx.ino.

11.1.4.4 lc1

```
LedControl lc1 =LedControl(12,11,10,1)
```

Create a new LedControl.

We use pins 12,11 and 10 for the SPI interface With our hardware we have connected pin 12 to the DATA IN-pin (1) of the first MAX7221 pin 11 is connected to the CLK-pin(13) of the first MAX7221 pin 10 is connected to the LOAD-pin(12) of the first MAX7221
We will only have a single MAX7221 attached to the arduino

Definition at line 20 of file SignalDriverMax72xx.ino.

11.1.4.5 s_digit

```
int s_digit
```

Start digit.

Definition at line 27 of file SignalDriverMax72xx.ino.

11.1.4.6 test

```
boolean test = false
```

Flag indicating if we are in test mode.

Definition at line 35 of file SignalDriverMax72xx.ino.

Chapter 12

Class Documentation

12.1 SignalDriverMax72xx Class Reference

SignalDriverMax72xx is a Snit type (OO class) that implements the host interface to the SignalDriverMax72xx program running on an Arduino.

Public Member Functions

- SignalDriverMax72xx (name,...)

  Constructor: open the port, configure it, set a readable file event, and prime the port.
- dark ()

  Method to turn off all LEDs.
- set (signo, aspect)

  Method to set the aspect for one signal.
- ~SignalDriverMax72xx ()

  Destructor: close the port.

Static Public Member Functions

- static validate (object)

  Type validation typemethod.

Private Member Functions

- _ReadPort ()

  Method to gobble from the Arduino.

Private Attributes

- portfd

  Variable to hold the port fd.
- _ready

  Variable to hold ready (for a command) state.

Static Private Attributes

- static validateaspects

  Validation type for aspects.
- static validatesignalnums

  Validation type for signal numbers.

12.1.1 Detailed Description

SignalDriverMax72xx is a Snit type (OO class) that implements the host interface to the SignalDriverMax72xx program running on an Arduino.

It provides an abstraction of the serial port interface that controls signals multiplexed by the MAX72xx chip. This version assumes a that there is only one SignalDriverMax72xx driver boards (1 to 8 signals, numbered 0 through 7) connected to the Arduino. This version assumes that only these aspects are valid (case folded):

g_r (Green over Red – Clear) y_r (Yellow over Red – Approach) r_r (Red over Red – [Absolute] Stop) r_g (Red over Green – Slow Clear) r_y (Red over Yellow – Approach Limited) dark (all lights off)

Definition at line 19 of file SignalDriverMax72xx_Host.tcl.

12.1.2 Constructor & Destructor Documentation

12.1.2.1 SignalDriverMax72xx()

```
SignalDriverMax72xx::SignalDriverMax72xx (
            name ,
             ...  )
```

Constructor: open the port, configure it, set a readable file event, and prime the port.

Parameters

name	The name of the object to be created.
...	Options: • -portname The name of the USB Serial port connecting to the Uno.

12.1.2.2 ~SignalDriverMax72xx()

```
SignalDriverMax72xx::~SignalDriverMax72xx ( )
```

Destructor: close the port.

12.1.3 Member Function Documentation

12.1.3.1 _ReadPort()

```
SignalDriverMax72xx::_ReadPort ( )  [private]
```

Method to gobble from the Arduino.

12.1.3.2 dark()

```
SignalDriverMax72xx::dark ( )
```

Method to turn off all LEDs.

12.1.3.3 set()

```
SignalDriverMax72xx::set (
            signo ,
            aspect  )
```

Method to set the aspect for one signal.

Parameters

signo	Signal number
aspect	The desired aspect

12.1.3.4 validate()

```
static SignalDriverMax72xx::validate (
            object  )  [static]
```

Type validation typemethod.

Parameters

object	An object to be validated.

12.1.4 Member Data Documentation

12.1.4.1 _ready

```
SignalDriverMax72xx::_ready  [private]
```

Variable to hold ready (for a command) state.

Definition at line 51 of file SignalDriverMax72xx_Host.tcl.

12.1.4.2 portfd

```
SignalDriverMax72xx::portfd  [private]
```

Variable to hold the port fd.

Definition at line 38 of file SignalDriverMax72xx_Host.tcl.

12.1.4.3 validateaspects

```
SignalDriverMax72xx::validateaspects  [static], [private]
```

Validation type for aspects.

Definition at line 24 of file SignalDriverMax72xx_Host.tcl.

12.1.4.4 validatesignalnums

```
SignalDriverMax72xx::validatesignalnums  [static], [private]
```

Validation type for signal numbers.

Definition at line 28 of file SignalDriverMax72xx_Host.tcl.

Index