# TRAFFIC LIGHT 502A

*CBI Arduino Class*

```
// Signal Bridge controller for trains using i2C bus for communication between bridges
// Based on Wire Peripheral Receiver by Nicholas Zambetti <http://www.zambetti.com>

// Created 04 March 2022
// 3.05 bring SB online
// 4.01 switch NB/SB to EB/WB, Right side and wrong side of tracks flag and logic, moved light pins off of pin 13
//           NOTE: Increasing bridge numbers is westbound, decreasing bridge numbers is eastbound
//
//                    __[n]___________
//                    |_RSisWB=True__|
//                     G|G                    G|G
// WB--> |=|=|=|=|=|=|  Y|Y =|=|=|=|=|=|=|=   Y|Y =|=|=|=|=|=
//                     R|R                    R|R
//                                          __[n+1]_______
//                                         | RSisWB=False |   <-- Arduino in relation to lights and track
//                                         ----------------
//
// 4.02 cleaning up serial outputs
// 4.02 Added a report out at startup so when reset your neighbors get a fresh report.
// 4.03 reviewing TIB logic and latching
// 4.03 Added Update monitoring system
// Added RED/YELLOW for not updated on startup
// 403E Parameterizing for Ups & Downs in prep for master look. Have to restructure data definition and setting up.
// 403G new wiring scheme and i2C cleanup. Added Ping on 13 for healthcheck
// 403H added downstream RED TIB logic, separated startup and updated
// 501A serial statue output changes to numbering flow direction

#include <Wire.h>

int n = 25; // This bridge's address, All the neighbors have a different signal bridge address
bool RSideIsWB = true; // Upstream lights (right side of tower) are westbound or wrong side=false
char Pversion[] = "i2CSigBridge502A";

const int Dwnnum = 3; //3;
const int Upnum = 3; //3; // Number of upstream neighbors WestBound = increasing bridge numbers

const int ni = 3; // Local's offset in the TIBs array
const bool Onstate = true; // Parameterize Output States for futre reversing logic
const bool Offstate = false;
const int SenseN = 2 ; // Pin 2 for input
bool ok = false; //dummy for subroutines

// Wiring layout RPins are on the analog side of Arduino
//      Gl | Gr
//      Yl | Yr          Orientation of lights w/r to tower
//      Rl | Rr
//         |
//     ---------
//     |Arduino|
//     ---------

```

```
52  const int RedRPin = 12;
53  const int YellowRPin = 11;
54  const int GreenRPin = 10;
55  const int GreenLPin = 9; // 403G enabling the SIgBridge ribbon to lay flat, no swaps
56  const int YellowLPin = 8;
57  const int RedLPin = 7;
58
59  const int PingPin = 13;
60
61  int GreenEBPin;
62  int YellowEBPin;
63  int RedEBPin;
64  int GreenWBPin;
65  int YellowWBPin;
66  int RedWBPin;
67
68  const unsigned long flashrate = 1000; // how long to stay on
69  const unsigned long resendrate = 25000; // time between resends
70  const unsigned long PingRate = 800; // time between pings LED flash and potential Serial
71
72  int Bridges = 0; // loop counterS
73  int BridgeNum =0; // Variable to read transmissions
74
75
76  // boolean flags that drive the lights
77  bool GREENEB = false; // Booleans used for light control
78  bool YELLOWEB = false;
79  bool REDEB = false;
80  bool YelloFEB = false; // flag for flashing yellow
81  bool GREENWB = false; // Booleans used for light control
82  bool YELLOWWB = false;
83  bool REDWB = false;
84  bool YelloFWB = false; // flag for flashing yellow
85
86  bool n0 = false; // this bridges's state
87  bool n00 = false; // this bridge's state last cycle
88  bool highlatch=false; // Keeping track of the train in the upstream block
89  bool highlatchW=false; // Keeping track of the train in the downstream block
90  bool toggle = false; // flag for when to flash
91  bool btoggle = true; // flag for when to resend
92  bool Ptoggle = false; // flag for when to Ping
93  bool Startup = true; // flag for when in startup mode
94
95  int StartUpcyc = 10000; // How many cycles in startup
96  unsigned long timestamp = 0; // cycletime recorder
97  unsigned long timestampr = 0; // resend states rate
98  unsigned long PingStamp = 0; // timer for Healthcheck Ping
99  unsigned long elapsed = 0; // How long has it been?
100 unsigned long Cycle=0; // How many cyles since startup?
101
102 bool TIB = false; // Train In Block triggers by leading edge local sensor, reset by trailing
    edge TIBn1 or TIBn-1
103 bool TIBm1 = false; // Last time we checked (TIBminus1)
104
105 // Storage Array for the neighborhood, 3 west, 3 east
106 //
107 //                            | n-3 | n-2 | n-1 |  n  | n+1 | n+2 | n+3 |
108 bool TIBs[Dwnnum+1+Upnum]   = {false,false,false,false,false,false,false}; // keep track of
    who has Train In Block (TIB)
109 bool Sensor[Dwnnum+1+Upnum] = {false,false,false,false,false,false,false}; // room for sensor
    states although n-1 and n+1 are only the critical ones.
110 bool Updated[Dwnnum+1+Upnum] = {false,false,false,true,false,false,false}; // Keeping track
    if we heard from everyone yet.
111
112 bool allUpdated = false;
```

```
113  int updIndex = 0; // Update Index for looping through the update matrix
114
115  void setup()
116  {
117    //FIRST! Swap signal sides if necessary, tower is either on the right or left side of
     travel. RPins on right side of tower facing track, LPins on left side of tower
118  if (RSideIsWB) {GreenEBPin = GreenLPin; GreenWBPin = GreenRPin;} else { GreenEBPin =
     GreenRPin; GreenWBPin = GreenLPin;}
119  if (RSideIsWB) { RedEBPin = RedLPin;  RedWBPin = RedRPin;} else { RedEBPin = RedRPin;
     RedWBPin = RedLPin;}
120  if (RSideIsWB) { YellowEBPin = YellowLPin;  YellowWBPin = YellowRPin;} else { YellowEBPin =
     YellowRPin;  YellowWBPin = YellowLPin;}
121
122
123    pinMode(RedEBPin, OUTPUT);            // -------------- SETUP PINS --------
124    pinMode(GreenEBPin, OUTPUT);
125    pinMode(YellowEBPin, OUTPUT);
126    pinMode(RedWBPin, OUTPUT);
127    pinMode(GreenWBPin, OUTPUT);
128    pinMode(YellowWBPin, OUTPUT);
129    pinMode(PingPin, OUTPUT);
130    pinMode(SenseN, INPUT_PULLUP);
131
132    Wire.begin(n);                // join i2c bus with address #n  -----  i2C  setup  ----
     establish local address
133    Wire.onReceive(receiveEvent); // register event
134
135    Serial.begin(9600);           // start serial for output to local monitor
136    Serial.print(n);Serial.print(" = ");Serial.println(Pversion); // Identify on reset what
     code this is
137
138   timestamp=millis()+2000; // punch the clock
139   timestampr=millis()+2000;
140   PingStamp= millis()+2000;
141
142
143   printTIBs(); // print the starting landscape, only for looks, stats don't come until states
     change.
144
145
146  }
147
148  void loop() ///---------------------------------- MAIN LOOP ---------------------------------
149  {
150       // Check Hdwr Sensor
151       n0 = !digitalRead(SenseN);  // IS THERE ANYTHING IN FRONT OF THE SENSOR?
152       Sensor[ni]= n0;
153  //        Serial.print("n:n0 "); Serial.print(n); Serial.print(": "); Serial.println(n0);
154      // Check Sensors and set TIB, Local or n+1 means TIB upstream
155           if (n0 || Sensor[ni+1]){ TIB=true; TIBs[ni]=TIB; //Establish Train In Block and
     update array.
156                       }
157       // Report if changed state, broadcast changes and echo to monitor
158       // State change checking
159      if (TIB != TIBm1 || n0 != n00 || btoggle ){
160                       ok = broadcastStateChg( n ); printTIBs();
161                       Serial.print("Local: "); Serial.print(n); if(TIB) {Serial.print(" TIB
     ");}else {Serial.print(" !TIB "); }//Serial.println(TIB);
162                       }
163      if(btoggle) btoggle =!btoggle; // Oneshot the btoggle
164      // WB Light output
165      GREENWB = !TIB & !TIBs[ni+1] & !TIBs[ni-1] & !TIBs[ni+2] & !TIBs[ni+3] & !Startup ;
166      if (GREENWB) digitalWrite ( GreenWBPin, Onstate); else digitalWrite ( GreenWBPin,
     Offstate);
167      YELLOWWB = TIBs[ni+2] &  !TIBs[ni+1] & !TIB || Startup;
```

```
168      YelloFWB = TIBs[ni+3] & !TIBs[ni+2] & !TIBs[ni+1] & !TIB & toggle;
169      if (YELLOWWB||YelloFWB) digitalWrite ( YellowWBPin, Onstate); else digitalWrite (
     YellowWBPin, Offstate);
170      //if (YelloFWB) digitalWrite ( YellowWBPin, Onstate); else digitalWrite ( YellowWBPin,
     Offstate);
171      REDWB = TIBs[ni-1] || TIBs[ni+1] || TIB || Startup;
172      if (REDWB) digitalWrite ( RedWBPin, Onstate); else digitalWrite ( RedWBPin, Offstate);
173
174      //EB Light output
175      GREENEB = !TIB & !TIBs[ni+1] & !TIBs[ni-1] & !TIBs[ni-2] & !TIBs[ni-3] & !Startup ;
176      if (GREENEB) digitalWrite ( GreenEBPin, Onstate); else digitalWrite ( GreenEBPin,
     Offstate);
177      YELLOWEB = TIBs[ni-2] &  !TIBs[ni-1] & !TIB || Startup;
178      YelloFEB = TIBs[ni-3] & !TIBs[ni-2] & !TIBs[ni-1] & !TIB & toggle;
179      if (YELLOWEB||YelloFEB) digitalWrite ( YellowEBPin, Onstate); else digitalWrite (
     YellowEBPin, Offstate);
180      REDEB = TIBs[ni+1] || TIBs[ni-1] || TIB || Startup;
181      if (REDEB) digitalWrite ( RedEBPin, Onstate); else digitalWrite ( RedEBPin, Offstate);
182
183      delay(1);
184
185      // Flag maintenance
186      n00 =n0; // remember where we left off
187      TIBm1 = TIB; // remember where we left off
188      // Latches for TIB
189      //WB
190          if(Sensor[ni+1]) highlatchW=true;
191          if(!Sensor[ni+1] & highlatchW & !n0){highlatchW=false;TIB=false;TIBs[ni]=TIB;
     broadcastStateChg(n);}
192      //EB
193          if(Sensor[ni-1]) highlatch=true;
194          if(!Sensor[ni-1] & highlatch & !n0){highlatch=false;TIB=false;TIBs[ni]=TIB;
     broadcastStateChg(n);}
195
196     // TIMER MAINTENANCE
197      // flasher system
198        elapsed = millis() - timestamp;
199        if(elapsed > flashrate ) { toggle=!toggle; timestamp=millis(); }  // Square wave of
     flashrate
200      // resendrate
201         elapsed = millis() - timestampr;
202         if(elapsed > resendrate ) { btoggle=!btoggle; timestampr=millis(); } // Square wave of
     flashrate, give up after 5 cycles
203      // Pingrate
204         elapsed = millis() - PingStamp;
205         if(elapsed > PingRate ) { Ptoggle=!Ptoggle; PingStamp=millis(); } // Square wave Ping
206
207      // Reset record of comms
208       if (allUpdated) {   for (Bridges = n-ni; Bridges< n+Upnum+1; Bridges++) {
     Updated[Bridges-n+ni]=false;}
209                          Updated[ni]=true;}
210
211     // Check if all reported in
212
213          allUpdated = true; // Hope for the best, last man wins
214          for (updIndex=ni-Dwnnum; updIndex<ni+Upnum+1; updIndex++) {
215           allUpdated = Updated[updIndex] & allUpdated;
216           }
217     // InitialCycle Phase
218     Cycle = Cycle + 1;
219     if (Cycle >StartUpcyc)Startup=false;  // End of Startup!
220
221     // Ping!
222      if (Ptoggle) digitalWrite ( PingPin, Onstate); else digitalWrite ( PingPin, Offstate);
223
```

```
}// ------ END OF MAIN LOOP ----------


// ------ function printTBs that reports the current TIB states of the neighborhood ----------
-
//
void printTIBs()
{
  // Print out TIB States
  int BridgeSrt= n-ni;
  int BridgeEnd= n+Upnum+1;
  int BridgeIcr= 1;
  if(RSideIsWB) {BridgeSrt= n+Upnum+1-1;BridgeEnd= n-ni-1;BridgeIcr= -1;}
    //Update Line
   Serial.println();
   if(RSideIsWB){Serial.print("    ");}
   for (Bridges = BridgeSrt; Bridges!= BridgeEnd; Bridges=Bridges+BridgeIcr) {
    Serial.print("|");
    if (Updated[Bridges-n+ni] ){Serial.print("^");} else {Serial.print("_");}
    Serial.print(Sensor[Bridges-n+ni]);
    Serial.print("|");Serial.print("    ");
    }
    if(!RSideIsWB) {Serial.println("|");} else{Serial.println();}
//    // label line
//   for (Bridges = n-ni; Bridges< n+Upnum+1; Bridges++) {
//    Serial.print("|");Serial.print(Bridges);Serial.print("|");
//    Serial.print(Bridges);Serial.print(Bridges+1);
//    }
//   Serial.println("|");

   // STATE Value Line
   for (Bridges = BridgeSrt; Bridges!= BridgeEnd; Bridges=Bridges+BridgeIcr) {
     if(!RSideIsWB){  // If right round print upstream status first
        if (Bridges != n) {Serial.print("|");} else {Serial.print("}");}
        Serial.print(Bridges);
        if (Bridges != n) {Serial.print("|");} else {Serial.print("{");}
        if(TIBs[Bridges-n+ni])Serial.print("_TIB");else Serial.print("####");
     } else {
        if(TIBs[Bridges-n+ni])Serial.print("_TIB");else Serial.print("####");
        if (Bridges != n) {Serial.print("|");} else {Serial.print("}");}
        Serial.print(Bridges);
        if (Bridges != n) {Serial.print("|");} else {Serial.print("{");}
     }
  }
    if(!RSideIsWB) {Serial.println("|");} else{Serial.println();}
} // End of printTIB ------------------------------------------------------------------


// function that executes whenever data is received from other bridges ----------------------
// this function is registered as an event, see setup()
void receiveEvent(int howMany)
{
  while(2 < Wire.available()) // loop through all but the last
  {
    BridgeNum = Wire.read(); // receive byte as a character
    Serial.print("Incoming B#:");
     Serial.print(BridgeNum);         // print the character
  }
  bool rTIB = Wire.read();    // receive byte as a bool
  if (rTIB) {Serial.print(" TIB");} else {Serial.print(" !TIB.");}// Serial.print(rTIB); //
print the TIB flag
  Serial.print(" S:");
  bool State = Wire.read();    // receive byte as a bool
  if (State) {Serial.println(" ON");} else {Serial.println("OFF");} //Serial.println(State);
// print the Sensor state
```

```
286     // Record reported States
287       TIBs[BridgeNum-n+ni] = rTIB;
288       Sensor[BridgeNum-n+ni] = State;
289       Updated[BridgeNum-n+ni] = true;
290       printTIBs();
291   //    if(Sensor[ni+1]) highlatch=true;
292   //    if(!Sensor[ni+1] & highlatch & !TIB)
      {highlatch=false;TIB=false;TIBs[ni]=TIB;broadcastStateChg(n);}
293   } // --- End of REceive Event ----------------------------------------------------------
      ---
294
295   // function that pushes the State change out to the neighbors --------------------------
      ----
296   // The packet sent is |Reporting Bridge|TIB|sensor|
297   bool broadcastStateChg(int bridge)
298   {
299     bool result=true;
300     for (bridge=n-Dwnnum; bridge<n+Upnum+1; bridge++){
301       //Serial.print(bridge);Serial.print(" : "); Serial.print(n);Serial.print(TIB);
302          if(bridge!=n){
303           Wire.beginTransmission(bridge); // transmit to device #bridge
304           Wire.write(n);        // sends bridge#
305           Wire.write(TIB);
306           Wire.write(n0);              // sends one byte state
307           Wire.endTransmission();    // stop transmitting
308           Serial.print("B:");Serial.print(bridge); Serial.print(" ");
309          }
310          delay(100);
311     }
312     Serial.println("");
313     return result;
314   } // end of broadcastStateChg ----------------------------------------------------------
```