RR SignalBridge Controller 505P

TPM

```
// 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
//                                        __[ThisBridge+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
// 505P revised nomenclature for Paul

#include <Wire.h>

int ThisBridge= 25; // This bridge's address, All the neighbors have a different signal bridge address, assume sequential integers
bool RSideIsWB = true; // Upstream lights (right side of tower) are westbound or wrong side=false
char Pversion[] = "i2CSigBridge505P";

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

const int TBridgIndex = 3; // ThisBrdgeIndex = Local's offset in the TIBs array
const bool Onstate = true; // Parameterize Output States for future reversing logic
const bool Offstate = false;
const int SensorPin = 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
53  const int RedRPin = 12;
54  const int YellowRPin = 11;
55  const int GreenRPin = 10;
56  const int GreenLPin = 9; // 403G enabling the SIgBridge ribbon to lay flat, no swaps
57  const int YellowLPin = 8;
58  const int RedLPin = 7;
59
60  const int PingPin = 13;
61
62  int GreenEBPin;
63  int YellowEBPin;
64  int RedEBPin;
65  int GreenWBPin;
66  int YellowWBPin;
67  int RedWBPin;
68
69  const unsigned long flashrate = 1000; // how long to stay on
70  const unsigned long resendrate = 55000; // time between resends  25000 for real hardware 55000
    for simulator
71  const unsigned long PingRate = 800; // time between pings LED flash and potential Serial
72  int StartUpcyc = 10; // How many cycles in startup 10000 for real hardware and 10 for
    simulator
73
74  int Bridges = 0; // loop counterS
75  int BridgeNum =0; // Variable to read transmissions
76  int UBridges = 0; // loop counter for updates
77
78  // boolean flags that drive the lights
79  bool GREENEB = false; // Booleans used for light control
80  bool YELLOWEB = false;
81  bool REDEB = false;
82  bool YelloFEB = false; // flag for flashing yellow
83  bool GREENWB = false; // Booleans used for light control
84  bool YELLOWWB = false;
85  bool REDWB = false;
86  bool YelloFWB = false; // flag for flashing yellow
87
88  bool ThBridgState = false; // this bridges's state
89  bool ThBridgStateLastCycle = false; // this bridge's state last cycle
90  bool highlatch=false; // Keeping track of the train in the upstream block
91  bool highlatchW=false; // Keeping track of the train in the downstream block
92  bool toggle = false; // flag for when to flash
93  bool btoggle = true; // flag for when to resend
94  bool Ptoggle = false; // flag for when to Ping
95  bool Startup = true; // flag for when in startup mode
96
97  unsigned long timestamp = 0; // cycletime recorder
98  unsigned long timestampr = 0; // resend states rate
99  unsigned long PingStamp = 0; // timer for Healthcheck Ping
100 unsigned long elapsed = 0; // How long has it been?
101 unsigned long Cycle=0; // How many cyles since startup?
102
103 bool TIB = false; // Train In Block triggers by leading edge local sensor, reset by trailing
    edge TIBn1 or TIBThisBridge-1
104 bool TIB_lastCycle = false; // Last time we checked (TIB last cycle)
105
106 // Storage Array for the neighborhood, 3 west, 3 east
107 //
108 //                           | ThisBridge-3 | ThisBridge-2 | ThisBridge-1 |  ThisBridge |
    ThisBridge+1 | ThisBridge+2 | ThisBridge+3 |
109 bool TIBs[Dwnnum+1+Upnum]   = {false,          false,          false,          false,
    false,          false,         false}; // keep track of who has Train In Block (TIB)
110 bool Sensor[Dwnnum+1+Upnum] = {false,          false,          false,          false,
    false,          false,         false}; // room for sensor states although ThisBridge-1 and
```

```
ThisBridge+1 are only the critical ones.
111 bool Updated[Dwnnum+1+Upnum] ={false,          false,          false,          false,
    false,          false,          false}; // Keeping track if we heard from everyone yet.
112
113 bool allUpdated = false; // Logical flag for if we heard from the whole hood or not
114 int updIndex = 0; // Update Index for looping through the update matrix
115
116 // Variables for display looping, need to print ascending or descending
117   int BridgeSrt= ThisBridge-TBridgIndex;
118   int BridgeEnd= ThisBridge+Upnum+1;
119   int BridgeIcr= 1;
120
121 void setup()
122 {
123   //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
124 if (RSideIsWB) {GreenEBPin = GreenLPin; GreenWBPin = GreenRPin;} else { GreenEBPin =
    GreenRPin; GreenWBPin = GreenLPin;}
125 if (RSideIsWB) { RedEBPin = RedLPin;  RedWBPin = RedRPin;} else { RedEBPin = RedRPin;
    RedWBPin = RedLPin;}
126 if (RSideIsWB) { YellowEBPin = YellowLPin;  YellowWBPin = YellowRPin;} else { YellowEBPin =
    YellowRPin;  YellowWBPin = YellowLPin;}
127
128 if(RSideIsWB) {BridgeSrt= ThisBridge+Upnum+1-1;BridgeEnd= ThisBridge-TBridgIndex-1;BridgeIcr=
    -1;} // Setup descending display if necessary
129
130   pinMode(RedEBPin, OUTPUT);            // -------------- SETUP PINS --------
131   pinMode(GreenEBPin, OUTPUT);
132   pinMode(YellowEBPin, OUTPUT);
133   pinMode(RedWBPin, OUTPUT);
134   pinMode(GreenWBPin, OUTPUT);
135   pinMode(YellowWBPin, OUTPUT);
136   pinMode(PingPin, OUTPUT);
137   pinMode(SensorPin, INPUT_PULLUP);  // Infrared sensor needs the pullup
138
139   Wire.begin(ThisBridge);                // join i2c bus with address #ThisBridge -----  i2C
    setup  ---- establish local address
140   Wire.onReceive(receiveEvent); // register event
141
142   Serial.begin(9600);           // start serial for output to local monitor
143   Serial.println(); Serial.print(ThisBridge);Serial.print(" = ");Serial.println(Pversion); //
    Identify on reset what code this is
144
145  timestamp=millis()+2000; // punch the clock, drives toggle square wave for Yellow flashing
146  timestampr=millis()+2000; // drives the btoggle square wave for when to broadcast the status
147  PingStamp= millis()+2000; // watchdog ptoggle square wave for heartbeat (output to onboard
    LED on Pin 13)
148
149  // Setup for Indexes used in the Print Routine, Normally biased to printing neighbors
    ascending, need to descend when RSideIsWB
150
151
152  printTIBs(); // print the starting landscape, only for looks, stats don't come until states
    change.
153
154
155 }
156
157 void loop() ///--------------------------------- MAIN LOOP ---------------------------------
158 {
159      // Check Hdwr Sensor
160      ThBridgState= !digitalRead(SensorPin);  // IS THERE ANYTHING IN FRONT OF THE SENSOR?
161      Sensor[TBridgIndex]= ThBridgState; // Put it into the array
162 //        Serial.print("n:ThBridgStateState "); Serial.print(ThisBridge); Serial.print(": ");
    Serial.println(ThBridgState);
```

```
163      // Check Sensors and set TIB, Local or ThisBridge+1 means TIB upstream
164           if (ThBridgState|| Sensor[TBridgIndex+1]){ TIB=true; TIBs[TBridgIndex]=TIB;
     //Establish Train In Block and update array.
165                       }
166       // Report if changed state, broadcast changes and echo to monitor
167       // State change checking
168      if (TIB != TIB_lastCycle || ThBridgState!= ThBridgStateLastCycle || btoggle ){
169                       ok = broadcastStateChg( ThisBridge); printTIBs();
170                       Serial.print("Local: "); Serial.print(ThisBridge); if(TIB)
     {Serial.print(" TIB ");}else {Serial.print(" !TIB "); }//Serial.println(TIB);
171                       }
172      if(btoggle) btoggle =!btoggle; // Oneshot the btoggle
173
174      // WB Light output   --- HERE'S the meat of what is happening -- determining which light
     should be on
175      GREENWB = !TIB & !TIBs[TBridgIndex+1] & !TIBs[TBridgIndex-1] & !TIBs[TBridgIndex+2] &
     !TIBs[TBridgIndex+3] & !Startup ;
176      if (GREENWB) digitalWrite ( GreenWBPin, Onstate); else digitalWrite ( GreenWBPin,
     Offstate);
177      YELLOWWB = TIBs[TBridgIndex+2] &  !TIBs[TBridgIndex+1] & !TIB || Startup;
178      YelloFWB = TIBs[TBridgIndex+3] & !TIBs[TBridgIndex+2] & !TIBs[TBridgIndex+1] & !TIB &
     toggle;
179      if (YELLOWWB||YelloFWB) digitalWrite ( YellowWBPin, Onstate); else digitalWrite (
     YellowWBPin, Offstate);
180      //if (YelloFWB) digitalWrite ( YellowWBPin, Onstate); else digitalWrite ( YellowWBPin,
     Offstate);
181      REDWB = TIBs[TBridgIndex-1] || TIBs[TBridgIndex+1] || TIB || Startup;
182      if (REDWB) digitalWrite ( RedWBPin, Onstate); else digitalWrite ( RedWBPin, Offstate);
183
184      //EB Light output
185      GREENEB = !TIB & !TIBs[TBridgIndex+1] & !TIBs[TBridgIndex-1] & !TIBs[TBridgIndex-2] &
     !TIBs[TBridgIndex-3] & !Startup ;
186      if (GREENEB) digitalWrite ( GreenEBPin, Onstate); else digitalWrite ( GreenEBPin,
     Offstate);
187      YELLOWEB = TIBs[TBridgIndex-2] &  !TIBs[TBridgIndex-1] & !TIB || Startup;
188      YelloFEB = TIBs[TBridgIndex-3] & !TIBs[TBridgIndex-2] & !TIBs[TBridgIndex-1] & !TIB &
     toggle;
189      if (YELLOWEB||YelloFEB) digitalWrite ( YellowEBPin, Onstate); else digitalWrite (
     YellowEBPin, Offstate);
190      REDEB = TIBs[TBridgIndex+1] || TIBs[TBridgIndex-1] || TIB || Startup;
191      if (REDEB) digitalWrite ( RedEBPin, Onstate); else digitalWrite ( RedEBPin, Offstate);
192
193      delay(1);
194
195      // Flag maintenance
196      ThBridgStateLastCycle =ThBridgState; // remember where we left off
197      TIB_lastCycle = TIB; // remember where we left off
198      // Latches for TIB
199      //WB
200          if(Sensor[TBridgIndex+1]) highlatchW=true;
201          if(!Sensor[TBridgIndex+1] & highlatchW & !ThBridgState)
     {highlatchW=false;TIB=false;TIBs[TBridgIndex]=TIB; broadcastStateChg(ThisBridge);}
202      //EB
203          if(Sensor[TBridgIndex-1]) highlatch=true;
204          if(!Sensor[TBridgIndex-1] & highlatch & !ThBridgState)
     {highlatch=false;TIB=false;TIBs[TBridgIndex]=TIB; broadcastStateChg(ThisBridge);}
205
206     // TIMER MAINTENANCE
207      // flasher system
208        elapsed = millis() - timestamp;
209        if(elapsed > flashrate ) { toggle=!toggle; timestamp=millis(); }  // Square wave of
     flashrate
210      // resendrate
211         elapsed = millis() - timestampr;
212         if(elapsed > resendrate ) { btoggle=!btoggle; timestampr=millis(); } // Square wave of
```

```
resendrate
213      // Pingrate
214         elapsed = millis() - PingStamp;
215         if(elapsed > PingRate ) { Ptoggle=!Ptoggle; PingStamp=millis(); } // Square wave Ping
216
217      // Reset record of comms
218       if (allUpdated) {   for (UBridges = ThisBridge-TBridgIndex; UBridges<
ThisBridge+Upnum+1; UBridges++) { Updated[UBridges-ThisBridge+TBridgIndex]=false;}
219                           Updated[TBridgIndex]=true;}
220
221     // Check if all reported in
222
223         allUpdated = true; // Hope for the best, last man wins
224         for (updIndex=TBridgIndex-Dwnnum; updIndex<TBridgIndex+Upnum+1; updIndex++) {
225          allUpdated = Updated[updIndex] & allUpdated; // Logical AND each report if all true
then allUPDATED!!
226          }
227     // InitialCycle Phase
228     Cycle = Cycle + 1;  // number of loop cycles to signify startup mode.
229     if (Cycle >StartUpcyc)Startup=false;  // End of Startup!
230
231     // Ping!
232      if (Ptoggle) digitalWrite ( PingPin, Onstate); else digitalWrite ( PingPin, Offstate);
233
234 }// ------ END OF MAIN LOOP ----------
235
236
237 // ------ function printTBs that reports the current TIB states of the neighborhood ----------
-
238 //
239 void printTIBs()
240 {
241   // Print out TIB States
242
243      //Update Line ----|^1|   or |^1|----| descend or ascend based on RSideIsWB ^ means
successful communitcation, 0,1 is the Sensor state
244      //    |_0|    |_0|    |_0|    |^0|    |_0|    |_0|    |_0|  <-- Update line
245      //####|25|####|24|####|23|####}22{####|21|####|20|####|19|  <-- STATE Value Line ####=
empty track, _TIB = Train In Block, |BridgeNumber|, }ThisBridge{
246      //
247      // |_0|    |_0|    |_0|    |^0|    |_0|    |_0|    |_0|    |  Different look if on the
other side of the track
248      // |17|####|18|####|19|####}20{####|21|####|22|####|23|####|   Note bias for bridge
reporting the upstream TIB
249
250     Serial.println(); // Start with a clean line
251     if(RSideIsWB){Serial.print("    ");}
252      for (Bridges = BridgeSrt; Bridges!= BridgeEnd; Bridges=Bridges+BridgeIcr) {  // For the
whole neighborhood
253       Serial.print("|");
254       if (Updated[Bridges-ThisBridge+TBridgIndex] ){Serial.print("^");} else
{Serial.print("_");} // ^ means that bridge passed an update
255       Serial.print(Sensor[Bridges-ThisBridge+TBridgIndex]); // The reported sensor state
256       Serial.print("|");Serial.print("    "); // space out to the next
257      }
258      if(!RSideIsWB) {Serial.println("|");} else{Serial.println();}
259
260
261     // STATE Value Line
262     for (Bridges = BridgeSrt; Bridges!= BridgeEnd; Bridges=Bridges+BridgeIcr) {
263       if(!RSideIsWB){  // If right round print upstream status first
264          if (Bridges != ThisBridge) {Serial.print("|");} else {Serial.print("}");}
265          Serial.print(Bridges);
266          if (Bridges != ThisBridge) {Serial.print("|");} else {Serial.print("{");}
267          if(TIBs[Bridges-ThisBridge+TBridgIndex])Serial.print("_TIB");else
```

```
      Serial.print("####");
268        } else {
269           if(TIBs[Bridges-ThisBridge+TBridgIndex])Serial.print("_TIB");else
      Serial.print("####");
270           if (Bridges != ThisBridge) {Serial.print("|");} else {Serial.print("}");}
271           Serial.print(Bridges);
272           if (Bridges != ThisBridge) {Serial.print("|");} else {Serial.print("{");}
273        }
274     }
275       if(!RSideIsWB) {Serial.println("|");} else{Serial.println();}
276   } // End of printTIB ------------------------------------------------------------------
277
278
279   // function that executes whenever data is received from other bridges ----------------------
280   // this function is registered as an event, see setup()
281   void receiveEvent(int howMany)
282   {
283     while(2 < Wire.available()) // loop through all but the last
284     {
285       BridgeNum = Wire.read(); // receive byte as a character
286       Serial.print("Incoming B#:");
287        Serial.print(BridgeNum);         // print the character
288     }
289     bool rTIB = Wire.read();    // receive byte as a bool
290     if (rTIB) {Serial.print(" TIB");} else {Serial.print(" !TIB.");}// Serial.print(rTIB); //
      print the TIB flag
291     Serial.print(" S:");
292     bool State = Wire.read();    // receive byte as a bool
293     if (State) {Serial.println(" ON");} else {Serial.println("OFF");} //Serial.println(State);
      // print the Sensor state
294     // Record reported States
295       TIBs[BridgeNum-ThisBridge+TBridgIndex] = rTIB;
296       Sensor[BridgeNum-ThisBridge+TBridgIndex] = State;
297       Updated[BridgeNum-ThisBridge+TBridgIndex] = true;
298    //   printTIBs(); As receive event is like an interrupt, this was causing strange behavior in
      serial output.
299   //    if(Sensor[TBridgIndex+1]) highlatch=true;
300   //    if(!Sensor[TBridgIndex+1] & highlatch & !TIB)
      {highlatch=false;TIB=false;TIBs[TBridgIndex]=TIB;broadcastStateChg(ThisBridge);}
301   } // --- End of REceive Event ------------------------------------------------------------
      ---
302
303   // function that pushes the State change out to the neighbors -------------------------------
      ----
304   // The packet sent is |Reporting Bridge|TIB|sensor|
305   bool broadcastStateChg(int bridge)
306   {
307     bool result=true;
308     for (bridge=ThisBridge-Dwnnum; bridge<ThisBridge+Upnum+1; bridge++){
309       //Serial.print(bridge);Serial.print(" : "); Serial.print(ThisBridge);Serial.print(TIB);
310          if(bridge!=ThisBridge){
311           Wire.beginTransmission(bridge); // transmit to device #bridge
312           Wire.write(ThisBridge);        // sends bridge#
313           Wire.write(TIB);
314           Wire.write(ThBridgState);              // sends one byte state
315           Wire.endTransmission();    // stop transmitting
316           Serial.print("B:");Serial.print(bridge); Serial.print(" ");
317          }
318          delay(100);
319     }
320     Serial.println("");
321     return result;
322   } // end of broadcastStateChg ---------------------------------------------------------
```