RR Signal Bridge 506P

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
//                                          __[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
// 506P more naming tweaks FlashToggle

#include <Wire.h>

int ThisBridge= 24; // This bridge's address, All the neighbors have a different signal bridge address, assume sequential integers
bool RSideIsWB = false; // Upstream lights (right side of tower) are westbound or wrong side=false
char Pversion[] = "i2CSigBridge506P";

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

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

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

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

```
      } else {
         if(TIBs[Bridges-ThisBridge+TBridgIndex])Serial.print("_TIB");else
Serial.print("####");
         if (Bridges != ThisBridge) {Serial.print("|");} else {Serial.print("}");}
         Serial.print(Bridges);
         if (Bridges != ThisBridge) {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
  // Record reported States
    TIBs[BridgeNum-ThisBridge+TBridgIndex] = rTIB;
    Sensor[BridgeNum-ThisBridge+TBridgIndex] = State;
    Updated[BridgeNum-ThisBridge+TBridgIndex] = true;
 //   printTIBs(); As receive event is like an interrupt, this was causing strange behavior in
serial output.
//    if(Sensor[TBridgIndex+1]) highlatch=true;
//    if(!Sensor[TBridgIndex+1] & highlatch & !TIB)
{highlatch=false;TIB=false;TIBs[TBridgIndex]=TIB;broadcastStateChg(ThisBridge);}
} // --- End of REceive Event ------------------------------------------------------------
---

// function that pushes the State change out to the neighbors -------------------------------
----
// The packet sent is |Reporting Bridge|TIB|sensor|
bool broadcastStateChg(int bridge)
{
  bool result=true;
  for (bridge=ThisBridge-Dwnnum; bridge<ThisBridge+Upnum+1; bridge++){
    //Serial.print(bridge);Serial.print(" : "); Serial.print(ThisBridge);Serial.print(TIB);
       if(bridge!=ThisBridge){
        Wire.beginTransmission(bridge); // transmit to device #bridge
        Wire.write(ThisBridge);        // sends bridge#
        Wire.write(TIB);
        Wire.write(ThBridgState);              // sends one byte state
        Wire.endTransmission();    // stop transmitting
        Serial.print("B:");Serial.print(bridge); Serial.print(" ");
       }
       delay(100);
  }
  Serial.println("");
  return result;
} // end of broadcastStateChg ---------------------------------------------------------
```