

```

1  // Created: 22 Feb 25 - Last Edit: 20 Jul 26
2  // Author: Terry Chamberlain (terry@a-train-systems.co.uk)
3  // Nano is powered at +9VDC from rectified DCC input (nominally 14VAC) via 7809 voltage regulator
4  // Regulated +5VDC from the Nano is used to power the 6N137 optoisolator which then feeds an
5  // .. isolated 5VAC version of the DCC input signal to Nano pin D2
6  // Nano Analog pins A1 and A3 are used to measure the District currents - all other Analog pins
7  // .. are connected to 0volts (GND) to minimise spurious crosstalk between ADCs
8
9  #include <NmraDcc.h>
10
11 NmraDcc DCC ;
12
13 const byte dpb_vern = 0x39; // Software Version = 3.9
14 const byte adr_prog = 11; // Set Accessory Address (pin D11)
15 const byte en_pout1 = 3; // Power District 1 Enable (pin D3)
16 const byte en_pout2 = 4; // Power District 2 Enable (pin D4)
17 const byte man_rst1 = 5; // Manual Reset Pushbutton 1 (pin D5)
18 const byte alm_out1 = 6; // Alarm Out 1 (Sounder) (pin D6)
19 const byte alm_led1 = 7; // Alarm LEDs 1 - Red (pin D7)
20 const byte man_rst2 = 8; // Manual Reset Pushbutton 2 (pin D8)
21 const byte alm_out2 = 9; // Alarm Out 2 (Sounder) (pin D9)
22 const byte alm_led2 = 10; // Alarm LEDs 2 - Red (pin D10)
23 const byte sens_in1 = 15; // Current Sensor 1 Input (pin A1)
24 const byte sens_in2 = 17; // Current Sensor 2 Input (pin A3)
25 const byte loop_mon = 12; // Loop period monitor (pin D12)
26
27 byte in_cmnd = 0; // Serial command in
28 byte ser_prm1 = 0; // Serial data in / out
29 byte ser_prm2 = 0; // Serial data in / out
30 String ser_prmstr = ""; // Serial data in - as string of digits
31 String sub_prmstr = ""; // Serial data in - without ending LF (0x0A) character
32
33 byte flash = 0;
34 byte prog_actv = 0;
35 byte prog_done = 0;
36 int set_adr = 0;
37
38 bool debug = false; // To allow all debug messages to be displayed by the Serial Monitor, set debug = true
39 bool mon_cur = false; // Set = true to display live values of current measured by Nano A-D converters
40
41 // Array with output values from Nano ADCs corresponding to load currents of 0.25A up to 5.0A
42 int trip_slct[] = {60, 103, 147, 190, 233, 276, 320, 363, 406, 449, 492, 536, 579, 622, 665, 709, 752, 795, 838, 881};
43
44 int sen1trip; // Power District 1 - Set break current level - approx 1.5A as default

```

```

45 int sen2trip;      // Power District 2 - Set break current level - approx 1.5A as default
46 int sen1read;      // Power District 1 - Current value from Sensor 1 - Nano ADC Pin A1 (15)
47 int sen2read;      // Power District 2 - Current value from Sensor 2 - Nano ADC Pin A3 (17)
48 unsigned long tmstart1; // Time when Over-Current detected in Power District 1
49 unsigned long timenow1; // Time read after Over-Current detected in Power District 1
50 unsigned long elapsed1; // Duration of Over-Current in Power District 1
51 unsigned long ovrtime1; // Duration (ms) of Over-Current before breaking in Power District 1
52 unsigned long trycnct1; // Multiples of 250ms before attempting reconnection in Power District 1
53 unsigned long tmstart2; // Time when Over-Current detected in Power District 2
54 unsigned long timenow2; // Time read after Over-Current detected in Power District 2
55 unsigned long elapsed2; // Duration of Over-Current in Power District 2
56 unsigned long ovrtime2; // Duration (ms) of Over-Current before breaking in Power District 1
57 unsigned long trycnct2; // Multiples of 250ms before attempting reconnection in Power District 1
58 unsigned long tmautosw; // Time when AutoReverser switches - ready to check for overcurrent in active District
59 unsigned long chkautsw; // Duration allowed for over-current detection
60                          // .. after AutoReverser switches = 4 x ovrtime1 or 4 x ovrtime2
61 unsigned long nxtadcop; // Next time to display ADC values in Serial Monitor if mon_cur = true (at 200msec intervals)
62 bool enblout1 = false; // Output Enabled - Power District 1
63 bool enblout2 = false; // Output Enabled - Power District 2
64 bool ovrcurr1 = false; // Over-Current Detected - Power District 1
65 bool ovrcurr2 = false; // Over-Current Detected - Power District 2
66 bool pwbreak1 = false; // Power Breaker Active - Power District 1
67 bool pwbreak2 = false; // Power Breaker Active - Power District 2
68 bool autorct1 = true;  // Auto Reconnect Active - Power District 1
69 bool autorct2 = true;  // Auto Reconnect Active - Power District 2
70 bool killpwr1 = false; // Break via soft command - Power District 1 (Can only resume via soft command or by Manual Reset)
71 bool killpwr2 = false; // Break via soft command - Power District 2
72 bool autorvsr = false; // AutoReverser Mode not enabled
73
74 int dpb_addr = 0;
75 int t;        // temp
76 int v;        // temp
77 int i;
78
79 byte busy_dcc = 0; // Set = 1 when any DCC command(s) being executed
80 byte arev_act = 0; // Set = 0 when AutoReverser not active, otherwise = 1 or 2 for active Power District
81
82 typedef struct
83 {
84     int cv_adr;
85     byte cv_val;
86 } cv_pair;
87
88 byte cv_value;

```

```

89  int  SET_CV_Address = 31;           // This Address is for setting CV'S like a Loco using Ops Mode - for this
90                                     // .. application it is made the same as the Power Breaker Address in CV 41 & CV42
91  int  Accessory_Address = 1;         // This Address is the default Accessory Address - not used for Power Breaker
92  byte CV_DECODER_MASTER_RESET = 120; // This is the CV Address for Full Reset - load a value of 120
93                                     // .. to this location and then press the Nano Reset button
94                                     // .. Return to default CV values can also be achieved by loading any value
95                                     // .. other than 0xAD (173) to CV50 and then pressing the Nano Reset button
96  byte CV_To_Store_SET_CV_Address = 121; // The address used to change CVs using Ops Mode (set as 31 above) is
97                                     // .. stored in this location, and in the following CV if greater than 256
98  byte CV_Accessory_Address = CV_ACCESSORY_DECODER_ADDRESS_LSB; // CV01 - the Board Address
99  byte accadrlo = lowByte(Accessory_Address);
100 byte accadrhi = highByte(Accessory_Address) & 0x07;
101 byte scvadrlo = lowByte(SET_CV_Address);
102 byte scvadrhi = highByte(SET_CV_Address) & 0x3F;
103
104 cv_pair FactoryDefaultCVs [] =
105 {
106     // These two CVs define the Long Accessory Address
107     //{CV_ACCESSORY_DECODER_ADDRESS_LSB, Accessory_Address&0xFF},
108     //{CV_ACCESSORY_DECODER_ADDRESS_MSB, (Accessory_Address>>8)&0x07},
109     {CV_ACCESSORY_DECODER_ADDRESS_LSB, accadrlo},
110     {CV_ACCESSORY_DECODER_ADDRESS_MSB, accadrhi},
111
112     {CV_MULTIFUNCTION_EXTENDED_ADDRESS_MSB, 0},
113     {CV_MULTIFUNCTION_EXTENDED_ADDRESS_LSB, 0},
114     // Accessory Decoder Short Address
115     //{CV_29_CONFIG,CV29_ACCESSORY_DECODER|CV29_OUTPUT_ADDRESS_MODE|CV29_F0_LOCATION},
116     // Accessory Decoder Long Address
117     {CV_29_CONFIG, CV29_ACCESSORY_DECODER|CV29_OUTPUT_ADDRESS_MODE|CV29_EXT_ADDRESSING | CV29_F0_LOCATION},
118
119     {CV_DECODER_MASTER_RESET, 0},
120     {CV_To_Store_SET_CV_Address, scvadrlo}, // LSB Set CV Address
121     {CV_To_Store_SET_CV_Address+1,scvadrlo}, // MSB Set CV Address
122     {30, 0}, // Not Used
123     {31, 0}, // Not Used
124     {32, 0}, // Not Used
125     {33, 0}, // Not Used
126     {34, 0}, // Not Used
127     {35, 0}, // Not Used
128     {36, 0}, // Not Used
129     {37, 0}, // Not Used
130     {38, 0}, // Not Used
131     {39, 0}, // Not Used
132     {40, 0}, // Not Used

```

```

133 {41, 31}, // Power Breaker Address LSB (31)
134 {42, 0}, // Power Breaker Address MSB
135 {43, 6}, // Current Sensor 1 Limit - Index (+1) into trip_slct[] array - set at 1.5A
136 {44, 6}, // Current Sensor 2 Limit - Index (+1) into trip_slct[] array - set at 1.5A
137 {45, 15}, // Allowed duration of overcurrent before breaking Power District 1 (msec) - set at 15ms
138 {46, 15}, // Allowed duration of overcurrent before breaking Power District 2 (msec) - set at 15ms
139 {47, 12}, // Time before attempting reconnection after break Power District 1 (250msec steps) - set at 3sec
140 {48, 12}, // Time before attempting reconnection after break Power District 2 (250msec steps) - set at 3sec
141 {49, 0}, // Use as AutoReverser if = 0x5A
142 {50, 0}, // Reset to default CVs if CV50 not equal to 173 (0xAD = "All Default")
143 {51, 0}, // Not Used
144 {52, 10}, // Manual-Auto Reset 1 - 0x0A (10) = Auto (default) - any other value for Manual
145 {53, 10}, // Manual-Auto Reset 2 - 0x0A (10) = Auto (default) - any other value for Manual
146 {54, 0}, // Output current values to serial monitor if not zero
147 {55, 0}, // Set debug = true if not zero
148 {56, 0}, // Not Used
149 {57, 0}, // Not Used
150 {58, 0}, // Not Used
151 {59, 0}, // Not Used
152 {60, 0}, // Not Used
153 {109, 68}, // "D" - Extended Decoder Version
154 {110, 80}, // "P"
155 {111, 66}, // "B"
156 {112, dpb_vern}, // Software Version
157 };
158 uint8_t FactoryDefaultCVIndex = sizeof(FactoryDefaultCVs)/sizeof(cv_pair);
159 //void notifyCVResetFactoryDefault()
160 //{
161 // Make FactoryDefaultCVIndex non-zero and equal to number of CV's to be reset
162 // to flag to the loop() function that a reset to Factory Defaults needs to be done
163 // FactoryDefaultCVIndex = sizeof(FactoryDefaultCVs)/sizeof(cv_pair);
164 //};
165 void(* resetFunc) (void) = 0; // Declare reset function at address 0
166
167 //*****
168 void setup() {
169     Serial.begin(38400); // Used either to display debug messages via the serial Monitor, or to link to PC COM port
170
171     // Setup which External Interrupt to use for the DCC input, and the associated Pin (2) - only used for programming CVs
172     DCC.pin(0, 2, 0);
173     // Call the main DCC Init function to enable the DCC Receiver
174     DCC.init( MAN_ID_DIY, dpb_vern, FLAGS_OUTPUT_ADDRESS_MODE | FLAGS_DCC_ACCESSORY_DECODER, CV_To_Store_SET_CV_Address);
175     // delay(100);
176

```

```

177 pinMode(adr_prog, INPUT_PULLUP); // Set as Address Programming enable - when connected to GND effectively
178                                     // .. enables NmraDCC and allows the Address (Accessory and Program on Main)
179                                     // .. of the DualPowerBreaker to be set by issuing a DCC Accessory command
180                                     // .. to that address
181 digitalWrite(adr_prog, HIGH); // Probably unnecessary - belt and braces !
182 // Initialize the control and external indicator digital pins as outputs set LOW
183 pinMode(en_pout1, OUTPUT);
184 digitalWrite(en_pout1, LOW); // Enable Power District 1 output - set Off initially
185 pinMode(en_pout2, OUTPUT);
186 digitalWrite(en_pout2, LOW); // Enable Power District 2 output - set Off initially
187 pinMode(man_rst1, INPUT_PULLUP); // Manual reset District 1 via pushbutton (normally-open)
188 pinMode(alm_out1, OUTPUT);
189 digitalWrite(alm_out1, LOW); // Alarm output (District 1 over-current) to piezo sounder
190 pinMode(alm_led1, OUTPUT);
191 digitalWrite(alm_led1, LOW); // Alarm output (District 1 over-current) to external Red LED
192 pinMode(man_rst2, INPUT_PULLUP); // Manual reset District 2 via pushbutton (normally-open)
193 pinMode(alm_out2, OUTPUT);
194 digitalWrite(alm_out2, LOW); // Alarm output (District 2 over-current) to piezo sounder
195 pinMode(alm_led2, OUTPUT);
196 digitalWrite(alm_led2, LOW); // Alarm output (District 2 over-current) to external Red LED
197 pinMode(loop_mon, OUTPUT);
198 digitalWrite(loop_mon, LOW); // Loop monitor
199
200 if ((DCC.getCV(CV_DECODER_MASTER_RESET)== CV_DECODER_MASTER_RESET) || (DCC.getCV(50) != 0xAD))
201 {
202     // Reset all defined CVs to default values if value of CV120 = 120 or CV50 is not equal to 173 (0xAD = "All Default")
203     for (int j=0; j < sizeof(FactoryDefaultCVs)/sizeof(cv_pair); j++ )
204         DCC.setCV( FactoryDefaultCVs[j].cv_adr, FactoryDefaultCVs[j].cv_val);
205     digitalWrite(alm_out1, HIGH);
206     delay (1000);
207     digitalWrite(alm_out1, LOW); // Flash Alarm Out 1 Red LED when CVs loaded
208     DCC.setCV(50, 0xAD);
209     if (debug) {
210         Serial.print("Reset to Default CVs, CV50 = ");
211         Serial.println(DCC.getCV(50), DEC) ;
212     }
213 }
214
215 // Check that current Software Version is stored in CV 112 in EEPROM
216 if (dpb_vern != byte (DCC.getCV(112))) {
217     DCC.setCV(112, dpb_vern); // Only save if Software Version has been updated
218     delay(5);
219 }
220

```

```

221 // Set all operating parameters in accordance with values stored in EEPROM
222 cv_value = byte (DCC.getCV(43));
223 sen1trip = trip_slct[cv_value - 1];
224 cv_value = byte (DCC.getCV(44));
225 sen2trip = trip_slct[cv_value - 1];
226 ovrtime1 = byte (DCC.getCV(45));
227 ovrtime2 = byte (DCC.getCV(46));
228 trycnct1 = 250 * DCC.getCV(47);
229 trycnct2 = 250 * DCC.getCV(48);
230 cv_value = byte (DCC.getCV(49));
231 if (cv_value == 0x5A) {
232     autorvsr = true; // AutoReverser Mode enabled
233 } else {
234     autorvsr = false; // AutoReverser Mode not enabled
235 }
236 dpb_addr = DCC.getCV(41) + (256 * DCC.getCV(42));
237 t = DCC.getCV(121) + (256 * DCC.getCV(122));
238 if (t != dpb_addr) {
239     cv_value = byte (DCC.getCV(41));
240     DCC.setCV(121, cv_value); // Update Program Main Address if Accessory Address has been updated
241     delay(5);
242     cv_value = byte (DCC.getCV(42));
243     DCC.setCV(122, cv_value);
244     delay(5);
245 }
246 cv_value = byte (DCC.getCV(52));
247 if ((cv_value == 10) || autorvsr) {
248     autorct1 = true;
249 } else {
250     autorct1 = false;
251 }
252 cv_value = byte (DCC.getCV(53));
253 if ((cv_value == 10) || autorvsr) {
254     autorct2 = true;
255 } else {
256     autorct2 = false;
257 }
258 cv_value = byte (DCC.getCV(55));
259 if (cv_value != 0) debug = true;
260 cv_value = byte (DCC.getCV(54));
261 if (cv_value != 0) {
262     mon_cur = true;
263     nxtadcop = millis(); // Initialise timer for output of ADC current values
264 }

```

```

265
266 tmautosw = 0; // Initialise timer for Auto Reverser switching
267
268 if (debug) {
269     cv_value = dpb_vern;
270     cv_value = cv_value >> 4;
271     Serial.print("DualPowerBreaker Version = ");
272     Serial.print(cv_value); // Major = MS Nibble
273     Serial.print(".");
274     cv_value = dpb_vern;
275     cv_value = cv_value & 0x0F;
276     Serial.println(cv_value); // Minor = LS Nibble
277     Serial.print("DualPowerBreaker Address = ");
278     Serial.println(dpb_addr, DEC) ;
279     t = DCC.getCV(121) + (256 * DCC.getCV(122));
280     Serial.print("Program-On-Main Address = ");
281     Serial.println(t, DEC) ;
282     Serial.print("Trip 1 Level = ");
283     Serial.println(sen1trip, DEC) ;
284     Serial.print("Trip 2 Level = ");
285     Serial.println(sen2trip, DEC) ;
286     Serial.print("Trip 1 Time = ");
287     Serial.println(ovrtime1, DEC) ;
288     Serial.print("Trip 2 Time = ");
289     Serial.println(ovrtime2, DEC) ;
290     Serial.print("Reconnect 1 Delay = ");
291     Serial.println(trycnct1, DEC) ;
292     Serial.print("Reconnect 2 Delay = ");
293     Serial.println(trycnct2, DEC) ;
294     if (!autorvsr) {
295         if (autorct1) {
296             Serial.println(F("Manual-Auto Reset 1 = Auto")); // Note use of F macro to move text out of dynamic memory space
297         } else {
298             Serial.println(F("Manual-Auto Reset 1 = Manual"));
299         }
300         if (autorct2) {
301             Serial.println(F("Manual-Auto Reset 2 = Auto"));
302         } else {
303             Serial.println(F("Manual-Auto Reset 2 = Manual"));
304         }
305         Serial.println(F("Normal Breaker Mode Enabled"));
306     } else {
307         Serial.println(F("AutoReverser Mode Enabled"));
308     }

```

```

309 }
310 if ((debug) && (mon_cur)) {
311   Serial.println(F("Outputting current values to Serial Monitor"));
312 }
313
314 if (!autorvsr) {
315   digitalWrite(en_pout1, HIGH); // Switch on Power District 1
316   enblout1 = true;
317   digitalWrite(alm_led1, LOW);
318   digitalWrite(en_pout2, HIGH); // Switch on Power District 2
319   enblout2 = true;
320   digitalWrite(alm_led2, LOW);
321   arev_act = 0; // Not in AutoReverser mode
322 } else {
323   digitalWrite(en_pout2, LOW); // Switch off Power District 2
324   enblout2 = false;
325   digitalWrite(alm_led2, HIGH);
326   delay(2); // Power Districts 1 & 2 outputs are linked together in anti-phase
327   digitalWrite(en_pout1, HIGH); // Switch on Power District 1
328   enblout1 = true;
329   digitalWrite(alm_led1, LOW);
330   arev_act = 1; // Power District 1 active in AutoReverser mode
331 }
332 }
333
334 //*****
335 void loop()
336 {
337   //The NmraDcc.process() method MUST be called frequently from this loop()
338   //function for correct library operation and to process all DCC packets
339   DCC.process();
340
341   digitalWrite(loop_mon, HIGH); // Loop monitor - code inactive
342   delayMicroseconds(500); // Sets execution time of loop() +500usec - nominally 730-800usec total
343   digitalWrite(loop_mon, LOW); // Loop monitor - code active
344
345   // ===== Check Serial (USB) port for received configuration and status commands =====
346   if (Serial.available() > 0) {
347     // Read the incoming byte:
348     in_cmnd = Serial.read();
349     if (in_cmnd == 0x4E || in_cmnd == 0x6E) {
350       // "N" - NOP command
351       debug = false; // Ensure neither debug nor mon_cur flags are set to avoid USB conflicts with terminal protocol
352       mon_cur = false;

```

```

353     DCC.setCV(55, 0);
354     delay(5);
355     DCC.setCV(54, 0);
356     delay(5);
357     Serial.println(F("N >> OK")); // .. send ACK response
358 }
359 else if (in_cmnd == 0x53 || in_cmnd == 0x73) {
360     // "S" - Return values of all 16 CVs holding Power Breaker parameters - CV41 to CV56
361     Serial.print(F("S >> CVs - ")); // Output copy of command
362     for(i = 41; i < 57; i++){
363         cv_value = DCC.getCV(i); // Fetch CV
364         if (i != 56) {
365             Serial.print(cv_value, DEC);
366             Serial.print(F(", ")); // Output as decimal values, comma-separated
367         } else {
368             Serial.println(cv_value, DEC);
369         }
370     }
371 }
372 else if (in_cmnd == 0x41 || in_cmnd == 0x61) {
373     // "A" - Return PowerBreaker Address command
374     ser_prm1 = DCC.getCV(42); // Address MSB
375     ser_prm2 = DCC.getCV(41); // Address LSB
376     t = (ser_prm1 * 256) + ser_prm2;
377     Serial.print(F("A >> Address = "));
378     Serial.println(t, DEC); // Output as decimal value
379 }
380 else if (in_cmnd == 0x56 || in_cmnd == 0x76) {
381     // "V" - Return Version command
382     Serial.print(F("V >> "));
383     cv_value = dpb_vern;
384     cv_value = cv_value >> 4;
385     Serial.print(cv_value, DEC); // Major Version = MS Nibble
386     Serial.print("."); // Major Version = MS Nibble
387     cv_value = dpb_vern;
388     cv_value = cv_value & 0x0F;
389     Serial.println(cv_value, DEC); // Minor Version = LS Nibble
390 }
391 else if (in_cmnd == 0x48 || in_cmnd == 0x68) {
392     // "H" - Return list of DPB Serial Monitor commands
393     Serial.println(F("H >> Available Commands - Use upper or lower case"));
394     Serial.println(F(" H - Help - Show this List of Commands")); //
395     Serial.println(F(" N - NOP - Set Normal Operation")); //
396     Serial.println(F(" S - Return Values of CV41-CV56")); //

```

```

397 Serial.println(F(" A - Return Set Accessory Address")); //
398 Serial.println(F(" V - Return Code Version Number")); //
399 Serial.println(F(" G - Enable Debug Messages")); //
400 Serial.println(F(" K - Enable Display of ADC Values")); //
401 Serial.println(F(" T - Set Accsy Address - Tnnnn [1 - 2043]")); //
402 Serial.println(F(" C - Set Trip Current - C1nn, C2nn [1 - 20]")); //
403 Serial.println(F(" D - Set Trip Delay - D1nnn, D2nnn [3 - 255]")); //
404 Serial.println(F(" R - Set Reconnect Delay - R1nnn, R2nnn [1 - 240]")); //
405 Serial.println(F(" M - Set Auto/Manual Reconnect - M1A, M1M, M2A, M2M")); //
406 Serial.println(F(" F - Set CV Values to Factory Default")); //
407 Serial.println(F(" B - Break Power District - B1, B2, BB")); //
408 Serial.println(F(" P - Resume Power District - P1, P2, PB")); //
409 Serial.println(F(" UE - AutoReverser Mode Enable")); //
410 Serial.println(F(" UD - AutoReverser Mode Disable")); //
411 }
412 if (in_cmnd == 0x47 || in_cmnd == 0x67) {
413     // "G" - Enable Debug Messages command
414     debug = true;
415     DCC.setCV(55, 0x0F); // Set debug CV to non-zero value
416     delay(5);
417     Serial.println(F("G >> OK")); // .. send ACK response
418 }
419 if (in_cmnd == 0x4B || in_cmnd == 0x6B) {
420     // "K" - Enable ADC Display command
421     mon_cur = true;
422     DCC.setCV(54, 0x0F); // Set current monitor CV to non-zero value
423     delay(5);
424     nxtadcop = millis(); // Initialise timer for output of ADC current values
425     Serial.println(F("K >> OK")); // .. send ACK response
426 }
427 else if (in_cmnd == 0x54 || in_cmnd == 0x74) {
428     // "T" - Set Power Breaker Address command - add new address - up to 4 numeric characters
429     ser_prmstr = Serial.readString(); // Get rest of input characters
430     t = ser_prmstr.length();
431     sub_prmstr = ser_prmstr.substring(0, (t - 1));
432     Serial.print(F("T"));
433     Serial.print(sub_prmstr); // Output copy of command
434     Serial.print(F(" >> "));
435     v = ser_prmstr.toInt();
436     if (v == 0 || v > 2043) {
437         Serial.println(F("Invalid Address - not in range 1 - 2043"));
438     } else {
439         t = DCC.getCV(41) + (256 * DCC.getCV(42));
440         if (t != v) {

```

```

441     ser_prm1 = highByte(v);
442     ser_prm2 = lowByte(v);
443     DCC.setCV(41, ser_prm2); // Update Accessory Address
444     delay(5); // .. if not equal to current stored value
445     DCC.setCV(42, ser_prm1);
446     delay(5);
447     DCC.setCV(121, ser_prm2); // Update Program Main Address to match
448     delay(5);
449     DCC.setCV(122, ser_prm1);
450     delay(5);
451 }
452 Serial.print(F("Address = "));
453 Serial.println(v, DEC); // .. send ACK response
454 }
455 }
456 else if (in_cmnd == 0x43 || in_cmnd == 0x63) {
457     // "C" - Set Trip Current command - add index (1-20) - up to 2 numeric characters
458     delayMicroseconds(300);
459     if (Serial.available() > 0) {
460         ser_prm1 = Serial.read(); // Get District Select parameter
461         delayMicroseconds(300);
462         ser_prmstr = Serial.readString(); // Get rest of input characters
463         t = ser_prmstr.length();
464         sub_prmstr = ser_prmstr.substring(0, (t - 1)); // Remove Newline character
465         if (ser_prm1 == 0x0A) ser_prm1 = 0x2D; // Replace Newline with '-'
466         Serial.print(F("C"));
467         Serial.print((char)ser_prm1);
468         Serial.print(sub_prmstr); // Output copy of command
469         Serial.print(F(" >> "));
470         v = ser_prmstr.toInt();
471         if (v == 0 || v > 20) {
472             Serial.println(F("Invalid Index - not in range 1 - 20"));
473         } else {
474             cv_value = lowByte(v); // Extract index value to fit byte
475             if (ser_prm1 == 0x31) {
476                 // Check CV43 value - District 1 Trip Index
477                 if (cv_value != DCC.getCV(43)) {
478                     sen1trip = trip_slct[cv_value]; // Update Trip Current value only if value has changed
479                     DCC.setCV(43, cv_value); // .. and load to CV memory
480                     delay(5);
481                 }
482                 Serial.print(F("Dist 1 Trip Current = "));
483                 Serial.print(cv_value*0.25, 2);
484                 Serial.println(F("A")); // .. send ACK response = Trip Current

```

```

485     } else if (ser_prm1 == 0x32) {
486         // Check CV44 value - District 2 Trip Index
487         if (cv_value != DCC.getCV(44)) {
488             sen2trip = trip_slct[cv_value]; // Update Trip Current value only if value
489             DCC.setCV(44, cv_value);        // .. has changed, and load to CV memory
490             delay(5);
491         }
492         Serial.print(F("Dist 2 Trip Current = "));
493         Serial.print(cv_value*0.25, 2);
494         Serial.println(F("A"));           // .. send ACK response = Trip Current
495     } else {
496         Serial.println(F("Invalid District - must be 1 or 2"));
497     }
498 }
499 }
500 }
501 else if (in_cmnd == 0x44 || in_cmnd == 0x64) {
502     // "D" - Set Trip Delay command - add value 3 - 255msecs
503     delayMicroseconds(300);
504     if (Serial.available() > 0) {
505         ser_prm1 = Serial.read(); // Get District Select parameter
506         delayMicroseconds(300);
507         ser_prmstr = Serial.readString(); // Get rest of input characters
508         t = ser_prmstr.length();
509         sub_prmstr = ser_prmstr.substring(0, (t - 1)); // Remove Newline character
510         if (ser_prm1 == 0x0A) ser_prm1 = 0x2D; // Replace Newline with '-'
511         Serial.print(F("D"));
512         Serial.print((char)ser_prm1);
513         Serial.print(sub_prmstr); // Output copy of command
514         Serial.print(F(" >> "));
515         v = ser_prmstr.toInt();
516         if (v < 3 || v > 255) {
517             Serial.println(F("Invalid Delay - not in range 3 - 255"));
518         } else {
519             cv_value = lowByte(v); // Adjust value to fit byte
520             if (ser_prm1 == 0x31) {
521                 // Check CV45 value - District 1 Trip Delay
522                 if (cv_value != DCC.getCV(45)) {
523                     ovrtime1 = cv_value; // Update Trip Delay value only if value
524                     DCC.setCV(45, cv_value); // .. has changed and load to CV memory
525                     delay(5);
526                 }
527                 Serial.print(F("Dist 1 Trip Delay = "));
528                 Serial.print(cv_value, DEC);

```

```

529     Serial.println(F("msec"));          // .. send ACK response = Trip Delay
530 } else if (ser_prm1 == 0x32) {
531     // Check CV46 value - District 2 Trip Delay
532     if (cv_value != DCC.getCV(46)) {
533         ovrtime1 = cv_value;          // Update Trip Delay value only if value
534         DCC.setCV(46, cv_value);      // .. has changed, and load to CV memory
535         delay(5);
536     }
537     Serial.print(F("Dist 2 Trip Delay = "));
538     Serial.print(cv_value, DEC);
539     Serial.println(F("msec"));          // .. send ACK response = Trip Delay
540 } else {
541     Serial.println(F("Invalid District - must be 1 or 2"));
542 }
543 }
544 }
545 }
546 else if (in_cmnd == 0x52 || in_cmnd == 0x72) {
547     // "R" - Set Reconnect Delay command - add index (1-240) - units of 0.25sec
548     delayMicroseconds(300);
549     if (Serial.available() > 0) {
550         ser_prm1 = Serial.read();      // Get District Select parameter
551         delayMicroseconds(300);
552         ser_prmstr = Serial.readString(); // Get rest of input characters
553         t = ser_prmstr.length();
554         sub_prmstr = ser_prmstr.substring(0, (t - 1)); // Remove Newline character
555         if (ser_prm1 == 0x0A) ser_prm1 = 0x2D;          // Replace Newline with '-'
556         Serial.print(F("R"));
557         Serial.print((char)ser_prm1);
558         Serial.print(sub_prmstr); // Output copy of command
559         Serial.print(F(" >> "));
560         v = ser_prmstr.toInt();
561         if (v == 0 || v > 240) {
562             Serial.println(F("Invalid Index - not in range 1 - 240"));
563         } else {
564             cv_value = lowByte(v); // Adjust value to fit byte
565             if (ser_prm1 == 0x31) {
566                 // Check CV47 value - District 1 Reconnect Delay Index
567                 if (cv_value != DCC.getCV(47)) {
568                     trycnct1 = 250 * cv_value; // Update Reconnect Delay index only if value
569                     DCC.setCV(47, cv_value);    // .. has changed and load to CV memory
570                     delay(5);
571                 }
572                 Serial.print(F("Dist 1 Reconnect Delay = "));

```

```

573     Serial.print(cv_value*0.25, 2);
574     Serial.println(F("sec"));           // .. send ACK response = Reconnect Delay
575 } else if (ser_prm1 == 0x32) {
576     // Check CV48 value - District 2 Reconnect Delay Index
577     if (cv_value != DCC.getCV(48)) {
578         trycnct2 = 250 * cv_value; // Update Reconnect Delay value only if value
579         DCC.setCV(48, cv_value);   // .. has changed, and load to CV memory
580         delay(5);
581     }
582     Serial.print(F("Dist 2 Reconnect Delay = "));
583     Serial.print(cv_value*0.25, 2);
584     Serial.println(F("sec"));           // .. send ACK response = Reconnect Delay
585 } else {
586     Serial.println(F("Invalid District - must be 1 or 2"));
587 }
588 }
589 }
590 }
591 else if (in_cmnd == 0x4D || in_cmnd == 0x6D) {
592     // "M" - Enable/Disable Auto Reconnect command - add 1 or 2 to select District
593     // .. then A to enable Auto, M to enable Manual
594     delayMicroseconds(300);
595     if (Serial.available() > 0) {
596         ser_prm1 = Serial.read(); // Get District Select parameter
597         delayMicroseconds(300);
598         ser_prmstr = Serial.readString(); // Get rest of input characters
599         if (ser_prm1 == 0x0A) ser_prm1 = 0x2D; // Replace Newline with '-'
600         t = ser_prmstr.length();
601         if (t > 1) {
602             ser_prm2 = ser_prmstr.charAt(0);
603         } else {
604             ser_prm2 = 0x2D;
605         }
606         Serial.print(F("M"));
607         Serial.print((char)ser_prm1);
608         Serial.print((char)ser_prm2); // Output copy of command
609         Serial.print(F(" >> "));
610         cv_value = 0x00;
611         if (ser_prm1 != 0x31 && ser_prm1 != 0x32) {
612             Serial.println(F("Invalid District - must be 1 or 2"));
613             cv_value = 0xFF; // Prepare to exit command
614         } else {
615             if (ser_prm2 == 0x4D || ser_prm2 == 0x6D) {
616                 cv_value = 0x00; // Prepare to set Manual Reconnect

```

```

617     } else if (ser_prm2 == 0x41 || ser_prm2 == 0x61) {
618         cv_value = 0x0A; // Prepare to set Auto Reconnect
619     } else {
620         Serial.println(F("Invalid Parameter - must be A or M"));
621         cv_value = 0xFF; // Prepare to exit command
622     }
623 }
624 if (cv_value != 0xFF) {
625     if (ser_prm1 == 0x31) {
626         if (cv_value != DCC.getCV(52)) {
627             // Check CV52 value - Manual-Auto Reconnect District 1
628             if (cv_value == 0x0A) {
629                 autorct1 = true; // Auto
630             } else {
631                 autorct1 = false; // Manual
632             }
633             DCC.setCV(52, cv_value); // Load to CV memory only if value has changed
634             delay(5);
635         }
636         Serial.print(F("Dist 1 Reconnect = "));
637         if (autorct1) {
638             Serial.println(F("Automatic"));
639         } else {
640             Serial.println(F("Manual"));
641         }
642     } else if (ser_prm1 == 0x32) {
643         if (cv_value != DCC.getCV(53)) {
644             // Check CV53 value - Manual-Auto Reconnect District 2
645             if (cv_value == 0x0A) {
646                 autorct2 = true; // Auto
647             } else {
648                 autorct2 = false; // Manual
649             }
650             DCC.setCV(53, cv_value); // Load to CV memory only if value has changed
651             delay(5);
652         }
653         Serial.print(F("Dist 2 Reconnect = "));
654         if (autorct2) {
655             Serial.println(F("Automatic"));
656         } else {
657             Serial.println(F("Manual"));
658         }
659     }
660 }

```

```

661     }
662 }
663 else if (in_cmnd == 0x46 || in_cmnd == 0x66) {
664     // "F" - Reset PowerBreaker to Default CV Values command
665     DCC.setCV(50, 0); // Set CV50 = 0
666     delay(5);
667     Serial.println("F >> CVs Reset"); // .. send ACK response
668     delay(5);
669     resetFunc(); // .. then call system reset to load default CVs on restart
670 }
671 else if (in_cmnd == 0x42 || in_cmnd == 0x62) {
672     // "B" - Break Power command - add 1 = Dist 1, or 2 = Dist 2, or B = Both Districts
673     delayMicroseconds(300);
674     if (Serial.available() > 0) {
675         ser_prm1 = Serial.read(); // Get District selection
676         if (ser_prm1 == 0x0A) ser_prm1 = 0x2D; // Replace Newline with '-'
677         Serial.print(F("B"));
678         Serial.print((char)ser_prm1); // Output copy of command
679         Serial.print(F(" >> "));
680         if (autorvsr) {
681             Serial.println(F("Remove AutoReverser connections and Disable AutoReverser (UD)"));
682         } else if (ser_prm1 == 0x31) {
683             pwbreak1 = true;
684             ovrcurr1 = true;
685             killpwr1 = true;
686             digitalWrite(alm_led1, HIGH); // Switch on Alarm LED 1
687             digitalWrite(en_pout1, LOW); // Switch off Power District 1
688             enblout1 = false;
689             Serial.println(F("Dist 1 Switched Off")); // ACK response
690         } else if (ser_prm1 == 0x32) {
691             pwbreak2 = true;
692             ovrcurr2 = true;
693             killpwr2 = true;
694             digitalWrite(alm_led2, HIGH); // Switch on Alarm LED 2
695             digitalWrite(en_pout2, LOW); // Switch off Power District 2
696             enblout2 = false;
697             Serial.println(F("Dist 2 Switched Off")); // ACK response
698         } else if (ser_prm1 == 0x42 || ser_prm1 == 0x62) {
699             pwbreak1 = true;
700             pwbreak2 = true;
701             ovrcurr1 = true;
702             ovrcurr2 = true;
703             killpwr1 = true;
704             killpwr2 = true;

```

```

705     digitalWrite(alm_led1, HIGH); // Switch on Alarm LED 1
706     digitalWrite(alm_led2, HIGH); // Switch on Alarm LED 2
707     digitalWrite(en_pout1, LOW); // Switch off Power District 1
708     digitalWrite(en_pout2, LOW); // Switch off Power District 2
709     enblout1 = false;
710     enblout2 = false;
711     Serial.println(F("Dists 1 & 2 Switched Off")); // ACK response
712 } else {
713     Serial.println(F("Invalid District - must be 1, 2 or B"));
714 }
715 }
716 }
717 else if (in_cmnd == 0x50 || in_cmnd == 0x70) {
718     // "P" - Resume Power command - add 1 = Dist 1, or 2 = Dist 2, or B = Both Districts
719     delayMicroseconds(300);
720     if (Serial.available() > 0) {
721         ser_prm1 = Serial.read(); // Get District selection
722         if (ser_prm1 == 0x0A) ser_prm1 = 0x2D; // Replace Newline with '-'
723         Serial.print(F("P"));
724         Serial.print((char)ser_prm1); // Output copy of command
725         Serial.print(F(" >> "));
726         if (autorvsr) {
727             Serial.println(F("Remove AutoReverser connections and Disable AutoReverser (UD)"));
728         } else if (ser_prm1 == 0x31) {
729             pwbreak1 = false;
730             ovrcurr1 = false;
731             killpwr1 = false;
732             digitalWrite(alm_led1, LOW); // Switch off Alarm LED 1
733             digitalWrite(en_pout1, HIGH); // Switch on Power District 1
734             enblout1 = true;
735             Serial.println(F("Dist 1 Switched On")); // ACK response
736         } else if (ser_prm1 == 0x32) {
737             pwbreak2 = false;
738             ovrcurr2 = false;
739             killpwr2 = false;
740             digitalWrite(alm_led2, LOW); // Switch off Alarm LED 2
741             digitalWrite(en_pout2, HIGH); // Switch on Power District 2
742             enblout2 = true;
743             Serial.println(F("Dist 2 Switched On")); // ACK response
744         } else if (ser_prm1 == 0x42 || ser_prm1 == 0x62) {
745             pwbreak1 = false;
746             pwbreak2 = false;
747             ovrcurr1 = false;
748             ovrcurr2 = false;

```

```

749     killpwr1 = false;
750     killpwr2 = false;
751     digitalWrite(alm_led1, LOW); // Switch off Alarm LED 1
752     digitalWrite(alm_led2, LOW); // Switch off Alarm LED 2
753     digitalWrite(en_pout1, HIGH); // Switch on Power District 1
754     digitalWrite(en_pout2, HIGH); // Switch on Power District 2
755     enblout1 = true;
756     enblout2 = true;
757     Serial.println(F("Dists 1 & 2 Switched On")); // ACK response
758 } else {
759     Serial.println(F("Invalid District - must be 1, 2 or B"));
760 }
761 }
762 }
763 else if (in_cmnd == 0x55 || in_cmnd == 0x75) {
764     // "U" - Enable/Disable Auto Reverser command - add E to enable, D to disable
765     delayMicroseconds(300);
766     if (Serial.available() > 0) {
767         ser_prm1 = Serial.read(); // Get Enable/Disable parameter
768         if (ser_prm1 == 0x0A) ser_prm1 = 0x2D; // Replace Newline with '-'
769         Serial.print(F("U"));
770         Serial.print((char)ser_prm1); // Output copy of command
771         Serial.print(F(" >> "));
772         if (ser_prm1 == 0x45 || ser_prm1 == 0x65) {
773             DCC.setCV(49, 0x5A); // Set CV49 = 90
774             delay(5);
775             digitalWrite(en_pout2, LOW); // Switch off Power District 2
776             enblout2 = false;
777             digitalWrite(alm_led2, HIGH); // Switch on Alarm LED 2
778             delay(2); // Power Districts 1 & 2 outputs are linked together in anti-phase
779             digitalWrite(en_pout1, HIGH); // Switch on Power District 1
780             enblout1 = true;
781             digitalWrite(alm_led1, LOW); // Switch off Alarm LED 1
782             autorvsr = true; // AutoReverser Mode enabled
783             arev_act = 1; // Power District 1 active in AutoReverser mode
784             Serial.println(F("AutoReverser Mode Enabled")); // ACK response
785         } else if (ser_prm1 == 0x44 || ser_prm1 == 0x64) {
786             DCC.setCV(49, 0x00); // Set CV49 = 0
787             delay(5);
788             digitalWrite(en_pout1, LOW); // Switch off Power District 1
789             enblout1 = false;
790             digitalWrite(alm_led1, HIGH); // Switch on Alarm LED 1
791             digitalWrite(en_pout2, LOW); // Switch off Power District 2
792             enblout2 = false;

```

```

793     digitalWrite(alm_led2, HIGH);    // Switch on Alarm LED 2
794     autorvsvr = false;
795     arev_act = 0;    // AutoReverser Mode disabled
796     Serial.println(F("AutoReverser Mode Disabled - Both Districts Switched Off")); // ACK response
797 } else {
798     Serial.println(F("Invalid Parameter - must be E to Enable or D to Disable"));
799 }
800 }
801 }
802 }
803
804 // ===== Check for Address Programming Link fitted =====
805 if ((digitalRead(adr_prog) == LOW) && (busy_dcc != 1)) {
806     if (prog_done == 0) {
807         delay(100);
808         if (digitalRead(adr_prog) == LOW) {
809             // Address Programming is active
810             digitalWrite(en_pout1, LOW);    // Switch off Power District 1
811             enblout1 = false;
812             digitalWrite(en_pout2, LOW);    // Switch off Power District 2
813             enblout2 = false;
814             digitalWrite(alm_led1, HIGH);    // Switch Alarm 1 Red LED On
815             prog_actv = 1;
816             busy_dcc = 1;
817             Serial.println(F("Address Programming Active"));
818         }
819     }
820 }
821 else if ((digitalRead(adr_prog) == LOW) && (busy_dcc == 1)) {
822     if (prog_done == 1) {
823         flash = flash + 1;
824         //Flash Alarm 1 Red LED as reminder to remove Programming Link
825         if (flash == 100){
826             digitalWrite(alm_led1, LOW);    // Switch off Alarm 1 Red LED
827         }
828         else{
829             if (flash == 200){
830                 digitalWrite(alm_led1, HIGH); // Switch on Alarm 1 Red LED
831                 flash = 0;
832             }
833         }
834     }
835 } else if ((digitalRead(adr_prog) == HIGH) && (prog_actv == 1)) {
836     delay(20);

```

```

837 // Programming Link removed - Address Programming will be abandoned
838 set_adr = 0; // Clear any entered address data
839 prog_actv = 0; // Disable Address Programming
840 prog_done = 0;
841 busy_dcc = 0; // Allow normal switch actions
842 Serial.println(F("Address Programming Inactive"));
843 digitalWrite(alm_led1, LOW); // Switch off Alarm 1 Red LED
844 delay(200);
845 resetFunc(); // Call system reset - execution stops here then returns to start
846 }
847 // ===== Start Current Monitoring =====
848
849 if (mon_cur) {
850   if (nxtadcop <= (millis() - 200)) {
851     // Print District Current Values read from Nano ADCs
852     t = analogRead(sens_in1); // Read District 1 Current
853     Serial.print(F("Dist 1 ADC = "));
854     Serial.print(t, DEC);
855     t = analogRead(sens_in2); // Read District 2 Current
856     Serial.print(F(" - Dist 2 ADC = "));
857     Serial.println(t, DEC);
858     nxtadcop = millis(); // Reset timer for next output of ADC current values
859   }
860 }
861
862 if (!autorvsr) {
863   // ==== Normal Power Breaker Operation =====
864   if ((digitalRead(adr_prog) == HIGH) && (busy_dcc != 1) && (prog_actv != 1)) {
865     // Check Power District 1
866     if (pwbreak1 && autorct1 && !killpwr1) {
867       elapsed1 = millis() - tmstart1; // Check if time to try reconnect after break
868       if (elapsed1 > trycnc1) {
869         // Attempt reconnect if Auto Reconnect active and Not Soft Break
870         if (debug) {
871           Serial.print(F("OverCurrent Period Elapsed"));
872           Serial.println(F(" - Reconnect Power District 1"));
873         }
874         digitalWrite(en_pout1, HIGH); // Switch on Power District 1
875         enblout1 = true;
876         digitalWrite(alm_led1, LOW); // Switch off Alarm LED 1
877         digitalWrite(alm_out1, LOW); // Switch off Alarm Out 1 (Sounder)
878         pwbreak1 = false;
879         ovrcurr1=false;
880       }

```

```

881 }
882 else {
883     sen1read = analogRead(sens_in1); // Read District 1 Current
884     if (sen1read > sen1trip) {
885         if (ovrcurr1) {
886             elapsed1 = millis() - tmstart1; // Over-current already detected - check if still true for set time
887             if (elapsed1 > ovrtime1) {
888                 // Initiate break - set Alarm 1
889                 digitalWrite(en_pout1, LOW); // Switch off Power District 1
890                 enblout1 = false;
891                 digitalWrite(alm_led1, HIGH); // Switch on Alarm LED 1
892                 digitalWrite(alm_out1, HIGH); // Switch on Alarm Out 1 (Sunder)
893                 if (debug) {
894                     Serial.println(F("Break Power District 1"));
895                 }
896                 pwbreak1 = true;
897                 tmstart1 = millis();
898             }
899         }
900         else {
901             ovrcurr1 = true; // Detected load > set trip current - set flag and start timer
902             tmstart1 = millis();
903         }
904     }
905     else {
906         ovrcurr1 = false; // Current has fallen below trip level - start monitoring again
907         elapsed1 = 0;
908         sen1read = 0; // Clear last read District 1 current
909     }
910 }
911 // Check Power District 2
912 if (pwbreak2 && autorct2 && !killpwr2) {
913     elapsed2 = millis() - tmstart2; // Check if time to try reconnect after break
914     if (elapsed2 > trycnct2) {
915         // Attempt reconnect if Auto Reconnect active and Not Soft Break
916         if (debug) {
917             Serial.print(F("OverCurrent Period Elapsed"));
918             Serial.println(F(" - Reconnect Power District 2"));
919         }
920         digitalWrite(en_pout2, HIGH); // Switch on Power District 2
921         enblout2 = true;
922         digitalWrite(alm_led2, LOW); // Switch off Alarm LED 2
923         digitalWrite(alm_out2, LOW); // Switch off Alarm Out 2 (Sunder)
924         pwbreak2 = false;

```

```

925     ovrcurr2=false;
926 }
927 }
928 else {
929     sen2read = analogRead(sens_in2); // Read District 2 Current
930     if (sen2read > sen2trip) {
931         if (ovrcurr2) {
932             elapsed2 = millis() - tmstart2; // Over-current already detected - check if still true for set time
933             if (elapsed2 > ovrtime2) {
934                 // Initiate break - set Alarm 2
935                 digitalWrite(en_pout2, LOW); // Switch off Power District 2
936                 enblout2 = false;
937                 digitalWrite(alm_led2, HIGH); // Switch on Alarm LED 2
938                 digitalWrite(alm_out2, HIGH); // Switch on Alarm Out 2 (Sunder)
939                 if (debug) {
940                     Serial.println(F("Break Power District 2"));
941                 }
942                 pwbreak2 = true;
943                 tmstart2 = millis();
944             }
945         }
946         else {
947             ovrcurr2 = true; // Detected load > set trip current - set flag and start timer
948             tmstart2 = millis();
949         }
950     }
951     else {
952         ovrcurr2 = false; // Current has fallen below trip level - start monitoring again
953         elapsed2 = 0;
954         sen2read = 0; // Clear last read District 2 current
955     }
956 }
957 }
958 } else {
959     // ==== Operation as Auto Reverser =====
960     if ((digitalRead(adr_prog) == HIGH) && (busy_dcc != 1) && (prog_actv != 1) && (arev_act == 1)) {
961         // Check Power District 1
962         if (pwbreak1) {
963             elapsed1 = millis() - tmstart1; // Check if time to try reconnect after break
964             if (elapsed1 > trycnct1) {
965                 // Attempt reconnect if Auto Reconnect active and Not Soft Break
966                 if (debug) {
967                     Serial.print(F("OverCurrent Period Elapsed"));
968                     Serial.println(F(" - Reconnect Power District 1"));

```

```

969     }
970     digitalWrite(en_pout1, HIGH); // Switch on Power District 1
971     enblout1 = true;
972     digitalWrite(alm_led1, LOW); // Switch off Alarm LED 1
973     digitalWrite(alm_out1, LOW); // Switch off Alarm Out 1 (Sounder)
974     pwbreak1 = false;
975     ovrcurr1=false;
976 }
977 } else {
978     senlread = analogRead(sens_in1); // Read District 1 Current
979     if (senlread > senltrip) {
980         if (ovrcurr1) {
981             elapsed1 = millis() - tmstart1; // Over-current already detected - check if still true for set time
982             if (elapsed1 > ovrtime1) {
983                 // Initiate AutoReverser switch to District 2 - disable District 1 output and set Alarm 1
984                 digitalWrite(en_pout1, LOW); // Switch off Power District 1
985                 enblout1 = false;
986                 digitalWrite(alm_led1, HIGH); // Switch on Alarm LED 1 (probably not used when in AutoReverser mode)
987                 digitalWrite(alm_out1, HIGH); // Switch on Alarm Out 1 (Sounder, " ")
988                 if (tmautosw == 0) {tmautosw = millis() - (6 * ovrtime1);} // Set tmautosw for first time through main loop
989                 chkautsw = millis() - tmautosw; // Check if District switched again within (4 * ovrtime1) after previous switch
990                 if (chkautsw < (4 * ovrtime1)) {
991                     pwbreak1 = true; // Prepare to retry switching Power District 1 on after delay
992                     tmstart1 = millis();
993                     if (debug) {
994                         Serial.print(F("Break Power Districts 1 & 2")); // Power District 2 is already off
995                         Serial.println(F(" - In Power District 1 - Both to reconnect"));
996                     }
997                 } else {
998                     delay(2);
999                     digitalWrite(en_pout2, HIGH); // Switch on Power District 2
1000                     enblout2 = true;
1001                     digitalWrite(alm_led2, LOW); // Switch off Alarm LED 2 (probably not used when in AutoReverser mode)
1002                     digitalWrite(alm_out2, LOW); // Switch off Alarm Out 2 (Sounder, " ")
1003                     tmautosw = millis(); // Start check period for continuing overload on newly active Power District
1004                     arev_act = 2; // Set power District 2 as active
1005                     if (debug) {
1006                         Serial.println(F("Break Power District 1 - District 2 Active"));
1007                     }
1008                 }
1009             }
1010         } else {
1011             ovrcurr1 = true; // Detected load > set trip current - set flag and start timer
1012             tmstart1 = millis();

```

```

1013         if (debug) {
1014             Serial.println(F("Overcurrent Detected District 1"));
1015         }
1016     }
1017 } else {
1018     ovrcurr1 = false; // Current has fallen below trip level - start monitoring again
1019     elapsed1 = 0;
1020     sen1read = 0;      // Clear last read District 1 current
1021 }
1022 }
1023 }
1024 if ((digitalRead(adr_prog) == HIGH) && (busy_dcc != 1) && (prog_actv != 1) && (arev_act == 2)) {
1025     // Check Power District 2
1026     if (pwbreak2) {
1027         elapsed2 = millis() - tmstart2; // Check if time to try reconnect after break
1028         if (elapsed2 > trycnct2) {
1029             // Attempt reconnect if Auto Reconnect active and Not Soft Break
1030             if (debug) {
1031                 Serial.print(F("OverCurrent Period Elapsed"));
1032                 Serial.println(F(" - Reconnect Power District 2"));
1033             }
1034             digitalWrite(en_pout2, HIGH); // Switch on Power District 2
1035             enblout2 = true;
1036             digitalWrite(alm_led2, LOW); // Switch off Alarm LED 2
1037             digitalWrite(alm_out2, LOW); // Switch off Alarm Out 2 (Sounder)
1038             pwbreak2 = false;
1039             ovrcurr2=false;
1040         }
1041     }
1042     else {
1043         sen2read = analogRead(sens_in2); // Read District 2 Current
1044         if (sen2read > sen2trip) {
1045             if (ovrcurr2) {
1046                 elapsed2 = millis() - tmstart2; // Over-current already detected - check if still true for set time
1047                 if (elapsed2 > ovrtime2) {
1048                     // Initiate AutoReverser switch to District 2 - disable District 1 output and set Alarm 1
1049                     digitalWrite(en_pout2, LOW); // Switch off Power District 2
1050                     enblout2 = false;
1051                     digitalWrite(alm_led2, HIGH); // Switch on Alarm LED 2 (probably not used when in AutoReverser mode)
1052                     digitalWrite(alm_out2, HIGH); // Switch on Alarm Out 2 (Sounder, " ")
1053                     if (tmautosw == 0) {tmautosw = millis() - (6 * ovrtime2);} // Set tmautosw for first time through main loop
1054                     chkautsw = millis() - tmautosw; // Check if District switched again within (4 * ovrtime1) after previous switch
1055                     if (chkautsw < (4 * ovrtime2)) {
1056                         pwbreak2 = true; // Prepare to retry switching Power District 2 on after delay

```

```

1057         tmstart2 = millis();
1058         if (debug) {
1059             Serial.print(F("Break Power Districts 1 & 2")); // Power District 1 is already off
1060             Serial.println(F(" - In Power District 2 - Both to reconnect"));
1061         }
1062     } else {
1063         delay(2);
1064         digitalWrite(en_pout1, HIGH); // Switch on Power District 1
1065         enblout1 = true;
1066         digitalWrite(alm_led1, LOW); // Switch off Alarm LED 1 (probably not used when in AutoReverser mode)
1067         digitalWrite(alm_out1, LOW); // Switch off Alarm Out 1 (Sunder, " ")
1068         tmautosw = millis(); // Start check period for continuing overload on newly active Power District
1069         arev_act = 1; // Set power District 1 as active
1070         if (debug) {
1071             Serial.println(F("Break Power District 2 - District 1 Active"));
1072         }
1073     }
1074 }
1075 }
1076 else {
1077     ovrcurr2 = true; // Detected load > set trip current - set flag and start timer
1078     tmstart2 = millis();
1079     if (debug) {
1080         Serial.println(F("Overcurrent Detected District 1"));
1081     }
1082 }
1083 }
1084 else {
1085     ovrcurr2 = false; // Current has fallen below trip level - start monitoring again
1086     elapsed2 = 0;
1087     sen2read = 0; // Clear last read District 2 current
1088 }
1089 }
1090 }
1091 }
1092
1093 // ===== Check for Manual Reset inputs active to reconnect Power Districts =====
1094 if ((digitalRead(man_rst1) == LOW) && (pwbreak1 || killpwr1)) {
1095     delay(30); // Only acted upon if District 1 power break is currently active
1096     if (digitalRead(man_rst1) == LOW) {
1097         while (digitalRead(man_rst1) == LOW){
1098             // Reset switch 1 pressed
1099         } // Wait for Reset switch 1 to be released
1100         // Reset Power District 1 - irrespective of whether Auto Reset is active or not

```

```

1101     pwbreak1 = false;
1102     ovrcurr1 = false;
1103     killpwr1 = false;
1104     digitalWrite(alm_led1, LOW); // Switch off Alarm LED 1
1105     digitalWrite(alm_out1, LOW); // Switch off Alarm Out 1 (Sounder)
1106     digitalWrite(en_pout1, HIGH); // Switch on Power District 1
1107     enblout1 = true;
1108 }
1109 }
1110 if ((digitalRead(man_rst2) == LOW) && (pwbreak2 || killpwr2)) {
1111     delay(30); // Only acted upon if District 2 power break is currently active
1112     if (digitalRead(man_rst2) == LOW) {
1113         while (digitalRead(man_rst2) == LOW){
1114             // Reset switch 2 pressed
1115         } // Wait for Reset switch 2 to be released
1116         // Reset Power District 2
1117         pwbreak2 = false;
1118         ovrcurr2 = false;
1119         killpwr2 = false;
1120         digitalWrite(alm_led2, LOW); // Switch off Alarm LED 2
1121         digitalWrite(alm_out2, LOW); // Switch off Alarm Out 2 (Sounder)
1122         digitalWrite(en_pout2, HIGH); // Switch on Power District 2
1123         enblout2 = true;
1124     }
1125 }
1126 } // End main loop()
1127
1128 //*****
1129
1130 // This function is called whenever a normal DCC Turnout Packet is received in Output Addressing Mode
1131 extern void notifyDccAccTurnoutOutput( uint16_t Addr, uint8_t Direction, uint8_t OutputPower ) {
1132     byte adr_lsb;
1133     byte adr_msb;
1134     if (prog_actv == 1 && prog_done == 0) {
1135         // Address Programming is active - acceptable addresses, 0001 to 2043
1136         // Received Address is accepted as Output Address for the Dual Power Breaker
1137         set_adr = Addr;
1138         if ((set_adr > 0) && (set_adr < 2044)) {
1139             Serial.print("Address = ");
1140             Serial.print(Addr, DEC);
1141             Serial.println(" Entered");
1142         } else {
1143             set_adr = 0;
1144             Serial.println("Entered Address must be between 1 and 2043");

```

```
1145     }
1146     // Transfer entered Output Address (if any) to CV41, 42 and CV121, 122
1147     if (set_adr != 0) {
1148         adr_lsb = lowByte(set_adr);
1149         adr_msb = highByte(set_adr);
1150         DCC.setCV(41, adr_lsb);
1151         delay(5);
1152         DCC.setCV(42, adr_msb);
1153         delay(5);
1154         Serial.print("Accessory Address Stored = ");
1155         Serial.println(set_adr, DEC);
1156         DCC.setCV(121, adr_lsb);
1157         delay(5);
1158         DCC.setCV(122, adr_msb);
1159         delay(5);
1160         Serial.print("Program-on-Main Address Stored = ");
1161         Serial.println(set_adr, DEC);
1162     }
1163     set_adr = 0;    // Clear any entered address data
1164     prog_done = 1;  // Prevent re-entry or setup operations until programming link removed
1165     flash = 0;      // Prepare to blink Red LED
1166     Serial.println("Address Programming Complete - Remove Programming Link");
1167 }
1168 }
```