

```

1  // Created: 22 Feb 25 - Last Edit: 16 Apr 25
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 = 0x38; // Software Version = 3.8
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
26  byte  in_cmnd = 0; // Serial command in
27  byte  ser_prm1 = 0; // Serial data in / out
28  byte  ser_prm2 = 0; // Serial data in / out
29  String ser_prmstr = ""; // Serial data in - as string of digits
30  String sub_prmstr = ""; // Serial data in - without ending LF (0x0A) character
31
32  byte  flash = 0;
33  byte  prog_actv = 0;
34  byte  prog_done = 0;
35  int   set_adr = 0;
36
37  bool  debug = false; // To allow all debug messages to be displayed by the Serial Monitor, set debug = true
38  bool  mon_cur = false; // Set = true to display live values of current measured by Nano A-D converters
39
40  // Array with output values from Nano ADCs corresponding to load currents of 0.25A up to 5.0A
41  int  trip_slct[] = {60, 103, 147, 190, 233, 276, 320, 363, 406, 449, 492, 536, 579, 622, 665, 709, 752, 795, 838, 881};
42
43  int  sen1trip; // Power District 1 - Set break current level - approx 1.5A as default
44  int  sen2trip; // Power District 2 - Set break current level - approx 1.5A as default
45  int  sen1read; // Power District 1 - Current value from Sensor 1 - Nano ADC Pin A1 (15)

```

```

46 int  sen2read;      // Power District 2 - Current value from Sensor 2 - Nano ADC Pin A3 (17)
47 unsigned long  tmstart1;  // Time when Over-Current detected in Power District 1
48 unsigned long  timenow1;  // Time read after Over-Current detected in Power District 1
49 unsigned long  elapsed1;  // Duration of Over-Current in Power District 1
50 unsigned long  ovrtime1;  // Duration (ms) of Over-Current before breaking in Power District 1
51 unsigned long  trycnct1;  // Multiples of 250ms before attempting reconnection in Power District 1
52 unsigned long  tmstart2;  // Time when Over-Current detected in Power District 2
53 unsigned long  timenow2;  // Time read after Over-Current detected in Power District 2
54 unsigned long  elapsed2;  // Duration of Over-Current in Power District 2
55 unsigned long  ovrtime2;  // Duration (ms) of Over-Current before breaking in Power District 1
56 unsigned long  trycnct2;  // Multiples of 250ms before attempting reconnection in Power District 1
57 unsigned long  tmautosw;  // Time when AutoReverser switches - ready to check for overcurrent in active District
58 unsigned long  chkautsw;  // Duration allowed for over-current detection
59                        // .. after AutoReverser switches = 4 x ovrtime1 or 4 x ovrtime2
60 unsigned long  nxtadcop;  // Next time to display ADC values in Serial Monitor if mon_cur = true (at 200msec intervals)
61 bool  enblout1 = false;  // Output Enabled - Power District 1
62 bool  enblout2 = false;  // Output Enabled - Power District 2
63 bool  ovrcurr1 = false;  // Over-Current Detected - Power District 1
64 bool  ovrcurr2 = false;  // Over-Current Detected - Power District 2
65 bool  pwbreak1 = false;  // Power Breaker Active - Power District 1
66 bool  pwbreak2 = false;  // Power Breaker Active - Power District 2
67 bool  autorct1 = true;   // Auto Reconnect Active - Power District 1
68 bool  autorct2 = true;   // Auto Reconnect Active - Power District 2
69 bool  killpwr1 = false;  // Break via soft command - Power District 1 (Can only resume via soft command or by Manual Reset)
70 bool  killpwr2 = false;  // Break via soft command - Power District 2
71 bool  autorvsr = false;  // AutoReverser Mode not enabled
72
73 int  dpb_addr = 0;
74 int  t;        // temp
75 int  v;        // temp
76 int  i;
77
78 byte  busy_dcc = 0;  // Set = 1 when any DCC command(s) being executed
79 byte  arev_act = 0;  // Set = 0 when AutoReverser not active, otherwise = 1 or 2 for active Power District
80
81 typedef struct
82 {
83     int  cv_adr;
84     byte cv_val;
85 } cv_pair;
86
87 byte  cv_value;
88 int  SET_CV_Address = 31;  // This Address is for setting CV'S like a Loco using Ops Mode - for this
89                        // .. application it is made the same as the Power Breaker Address in CV 41 & CV42
90 int  Accessory_Address = 1;  // This Address is the default Accessory Address - not used for Power Breaker

```

```

91 byte CV_DECODER_MASTER_RESET = 120;    // This is the CV Address for Full Reset - load a value of 120
92                                         // .. to this location and then press the Nano Reset button
93                                         // .. Return to default CV values can also be achieved by loading any value
94                                         // .. other than 0xAD (173) to CV50 and then pressing the Nano Reset button
95 byte CV_To_Store_SET_CV_Address = 121; // The address used to change CVs using Ops Mode (set as 31 above) is
96                                         // .. stored in this location, and in the following CV if greater than 256
97 byte CV_Accessory_Address = CV_ACCESSORY_DECODER_ADDRESS_LSB; // CV01 - the Board Address
98 byte accadrlo = lowByte(Accessory_Address);
99 byte accadrhi = highByte(Accessory_Address) & 0x07;
100 byte scvadrlo = lowByte(SET_CV_Address);
101 byte scvadrhi = highByte(SET_CV_Address) & 0x3F;
102
103 cv_pair FactoryDefaultCVs [] =
104 {
105     // These two CVs define the Long Accessory Address
106     //{CV_ACCESSORY_DECODER_ADDRESS_LSB, Accessory_Address&0xFF},
107     //{CV_ACCESSORY_DECODER_ADDRESS_MSB, (Accessory_Address>>8)&0x07},
108     {CV_ACCESSORY_DECODER_ADDRESS_LSB, accadrlo},
109     {CV_ACCESSORY_DECODER_ADDRESS_MSB, accadrhi},
110
111     {CV_MULTIFUNCTION_EXTENDED_ADDRESS_MSB, 0},
112     {CV_MULTIFUNCTION_EXTENDED_ADDRESS_LSB, 0},
113     // Accessory Decoder Short Address
114     // {CV_29_CONFIG,CV29_ACCESSORY_DECODER|CV29_OUTPUT_ADDRESS_MODE|CV29_F0_LOCATION},
115     // Accessory Decoder Long Address
116     {CV_29_CONFIG, CV29_ACCESSORY_DECODER|CV29_OUTPUT_ADDRESS_MODE|CV29_EXT_ADDRESSING | CV29_F0_LOCATION},
117
118     {CV_DECODER_MASTER_RESET, 0},
119     {CV_To_Store_SET_CV_Address, scvadrlo},    // LSB Set CV Address
120     {CV_To_Store_SET_CV_Address+1,scvadrlo},  // MSB Set CV Address
121     {30, 0},    // Not Used
122     {31, 0},    // Not Used
123     {32, 0},    // Not Used
124     {33, 0},    // Not Used
125     {34, 0},    // Not Used
126     {35, 0},    // Not Used
127     {36, 0},    // Not Used
128     {37, 0},    // Not Used
129     {38, 0},    // Not Used
130     {39, 0},    // Not Used
131     {40, 0},    // Not Used
132     {41, 31},   // Power Breaker Address LSB (31)
133     {42, 0},    // Power Breaker Address MSB
134     {43, 6},    // Current Sensor 1 Limit - Index (+1) into trip_slct[] array - set at 1.5A
135     {44, 6},    // Current Sensor 2 Limit - Index (+1) into trip_slct[] array - set at 1.5A

```

```

136 {45, 25}, // Allowed duration of overcurrent before breaking Power District 1 (msec) - set at 25ms
137 {46, 25}, // Allowed duration of overcurrent before breaking Power District 2 (msec) - set at 25ms
138 {47, 12}, // Time before attempting reconnection after break Power District 1 (250msec steps) - set at 3sec
139 {48, 12}, // Time before attempting reconnection after break Power District 2 (250msec steps) - set at 3sec
140 {49, 0}, // Use as AutoReverser if = 0x5A
141 {50, 0}, // Reset to default CVs if CV50 not equal to 173 (0xAD = "All Default")
142 {51, 0}, // Not Used
143 {52, 10}, // Manual-Auto Reset 1 - 0x0A (10) = Auto (default) - any other value for Manual
144 {53, 10}, // Manual-Auto Reset 2 - 0x0A (10) = Auto (default) - any other value for Manual
145 {54, 0}, // Output current values to serial monitor if not zero
146 {55, 0}, // Set debug = true if not zero
147 {56, 0}, // Not Used
148 {57, 0}, // Not Used
149 {58, 0}, // Not Used
150 {59, 0}, // Not Used
151 {60, 0}, // Not Used
152 {109, 68}, // "D" - Extended Decoder Version
153 {110, 80}, // "P"
154 {111, 66}, // "B"
155 {112, dpb_vern}, // Software Version
156 };
157 uint8_t FactoryDefaultCVIndex = sizeof(FactoryDefaultCVs)/sizeof(cv_pair);
158 //void notifyCVResetFactoryDefault()
159 //{
160 // Make FactoryDefaultCVIndex non-zero and equal to number of CV's to be reset
161 // to flag to the loop() function that a reset to Factory Defaults needs to be done
162 // FactoryDefaultCVIndex = sizeof(FactoryDefaultCVs)/sizeof(cv_pair);
163 //};
164 void(* resetFunc) (void) = 0; // Declare reset function at address 0
165
166 //*****
167 void setup() {
168   Serial.begin(38400); // Used either to display debug messages via the serial Monitor, or to link to PC COM port
169
170   // Setup which External Interrupt to use for the DCC input, and the associated Pin (2) - only used for programming CVs
171   DCC.pin(0, 2, 0);
172   // Call the main DCC Init function to enable the DCC Receiver
173   DCC.init( MAN_ID_DIY, dpb_vern, FLAGS_OUTPUT_ADDRESS_MODE | FLAGS_DCC_ACCESSORY_DECODER, CV_To_Store_SET_CV_Address);
174   // delay(100);
175
176   pinMode(adr_prog, INPUT_PULLUP); // Set as Address Programming enable - when connected to GND effectively
177   // .. enables NmraDCC and allows the Address (Accessory and Program on Main)
178   // .. of the DualPowerBreaker to be set by issuing a DCC Accessory command
179   // .. to that address
180   digitalWrite(adr_prog, HIGH); // Probably unnecessary - belt and braces !

```

```

181 // Initialize the control and external indicator digital pins as outputs set LOW
182 pinMode(en_pout1, OUTPUT);
183 digitalWrite(en_pout1, LOW); // Enable Power District 1 output - set Off initially
184 pinMode(en_pout2, OUTPUT);
185 digitalWrite(en_pout2, LOW); // Enable Power District 2 output - set Off initially
186 pinMode(man_rst1, INPUT_PULLUP); // Manual reset District 1 via pushbutton (normally-open)
187 pinMode(alm_out1, OUTPUT);
188 digitalWrite(alm_out1, LOW); // Alarm output (District 1 over-current) to piezo sounder
189 pinMode(alm_led1, OUTPUT);
190 digitalWrite(alm_led1, LOW); // Alarm output (District 1 over-current) to external Red LED
191 pinMode(man_rst2, INPUT_PULLUP); // Manual reset District 2 via pushbutton (normally-open)
192 pinMode(alm_out2, OUTPUT);
193 digitalWrite(alm_out2, LOW); // Alarm output (District 2 over-current) to piezo sounder
194 pinMode(alm_led2, OUTPUT);
195 digitalWrite(alm_led2, LOW); // Alarm output (District 2 over-current) to external Red LED
196
197 if ((DCC.getCV(CV_DECODER_MASTER_RESET)== CV_DECODER_MASTER_RESET) || (DCC.getCV(50) != 0xAD))
198 {
199     // Reset all defined CVs to default values if value of CV120 = 120 or CV50 is not equal to 173 (0xAD = "All Default")
200     for (int j=0; j < sizeof(FactoryDefaultCVs)/sizeof(cv_pair); j++)
201         DCC.setCV( FactoryDefaultCVs[j].cv_adr, FactoryDefaultCVs[j].cv_val);
202     digitalWrite(alm_out1, HIGH);
203     delay (1000);
204     digitalWrite(alm_out1, LOW); // Flash Alarm Out 1 Red LED when CVs loaded
205     DCC.setCV(50, 0xAD);
206     if (debug) {
207         Serial.print("Reset to Default CVs, CV50 = ");
208         Serial.println(DCC.getCV(50), DEC) ;
209     }
210 }
211
212 // Check that current Software Version is stored in CV 112 in EEPROM
213 if (dpb_vern != byte (DCC.getCV(112))) {
214     DCC.setCV(112, dpb_vern); // Only save if Software Version has been updated
215     delay(5);
216 }
217
218 // Set all operating parameters in accordance with values stored in EEPROM
219 cv_value = byte (DCC.getCV(43));
220 sen1trip = trip_slct[cv_value - 1];
221 cv_value = byte (DCC.getCV(44));
222 sen2trip = trip_slct[cv_value - 1];
223 ovrtime1 = byte (DCC.getCV(45));
224 ovrtime2 = byte (DCC.getCV(46));
225 trycnc1 = 250 * DCC.getCV(47);

```

```

226 trycnct2 = 250 * DCC.getCV(48);
227 cv_value = byte (DCC.getCV(49));
228 if (cv_value == 0x5A) {
229     autorvsr = true;    // AutoReverser Mode enabled
230 } else {
231     autorvsr = false;  // AutoReverser Mode not enabled
232 }
233 dpb_addr = DCC.getCV(41) + (256 * DCC.getCV(42));
234 t = DCC.getCV(121) + (256 * DCC.getCV(122));
235 if (t != dpb_addr) {
236     cv_value = byte (DCC.getCV(41));
237     DCC.setCV(121, cv_value); // Update Program Main Address if Accessory Address has been updated
238     delay(5);
239     cv_value = byte (DCC.getCV(42));
240     DCC.setCV(122, cv_value);
241     delay(5);
242 }
243 cv_value = byte (DCC.getCV(52));
244 if ((cv_value == 10) || autorvsr) {
245     autorct1 = true;
246 } else {
247     autorct1 = false;
248 }
249 cv_value = byte (DCC.getCV(53));
250 if ((cv_value == 10) || autorvsr) {
251     autorct2 = true;
252 } else {
253     autorct2 = false;
254 }
255 cv_value = byte (DCC.getCV(55));
256 if (cv_value != 0) debug = true;
257 cv_value = byte (DCC.getCV(54));
258 if (cv_value != 0) {
259     mon_cur = true;
260     nxtadcop = millis(); // Initialise timer for output of ADC current values
261 }
262
263 tmautosw = 0; // Initialise timer for Auto Reverser switching
264
265 if (debug) {
266     cv_value = dpb_vern;
267     cv_value = cv_value >> 4;
268     Serial.print("DualPowerBreaker Version = ");
269     Serial.print(cv_value); // Major = MS Nibble
270     Serial.print(".");

```

```

271 cv_value = dpb_vern;
272 cv_value = cv_value & 0x0F;
273 Serial.println(cv_value); // Minor = LS Nibble
274 Serial.print("DualPowerBreaker Address = ");
275 Serial.println(dpb_addr, DEC) ;
276 t = DCC.getCV(121) + (256 * DCC.getCV(122));
277 Serial.print("Program-On-Main Address = ");
278 Serial.println(t, DEC) ;
279 Serial.print("Trip 1 Level = ");
280 Serial.println(sen1trip, DEC) ;
281 Serial.print("Trip 2 Level = ");
282 Serial.println(sen2trip, DEC) ;
283 Serial.print("Trip 1 Time = ");
284 Serial.println(ovrtime1, DEC) ;
285 Serial.print("Trip 2 Time = ");
286 Serial.println(ovrtime2, DEC) ;
287 Serial.print("Reconnect 1 Delay = ");
288 Serial.println(trycnct1, DEC) ;
289 Serial.print("Reconnect 2 Delay = ");
290 Serial.println(trycnct2, DEC) ;
291 if (!autorvsr) {
292     if (autorct1) {
293         Serial.println(F("Manual-Auto Reset 1 = Auto")); // Note use of F macro to move text out of dynamic memory space
294     } else {
295         Serial.println(F("Manual-Auto Reset 1 = Manual"));
296     }
297     if (autorct2) {
298         Serial.println(F("Manual-Auto Reset 2 = Auto"));
299     } else {
300         Serial.println(F("Manual-Auto Reset 2 = Manual"));
301     }
302     Serial.println(F("Normal Breaker Mode Enabled"));
303 } else {
304     Serial.println(F("AutoReverser Mode Enabled"));
305 }
306 }
307 if ((debug) && (mon_cur)) {
308     Serial.println(F("Outputting current values to Serial Monitor"));
309 }
310
311 if (!autorvsr) {
312     digitalWrite(en_pout1, HIGH); // Switch on Power District 1
313     enblout1 = true;
314     digitalWrite(alm_led1, LOW);
315     digitalWrite(en_pout2, HIGH); // Switch on Power District 2

```



```

316     enblout2 = true;
317     digitalWrite(alm_led2, LOW);
318     arev_act = 0; // Not in AutoReverser mode
319 } else {
320     digitalWrite(en_pout2, LOW); // Switch off Power District 2
321     enblout2 = false;
322     digitalWrite(alm_led2, HIGH);
323     delay(2); // Power Districts 1 & 2 outputs are linked together in anti-phase
324     digitalWrite(en_pout1, HIGH); // Switch on Power District 1
325     enblout1 = true;
326     digitalWrite(alm_led1, LOW);
327     arev_act = 1; // Power District 1 active in AutoReverser mode
328 }
329 }
330
331 //*****
332 void loop()
333 {
334     //The NmraDcc.process() method MUST be called frequently from this loop()
335     //function for correct library operation and to process all DCC packets
336     DCC.process();
337
338     delay(2); // Sets normal execution time of loop() if no configuration operations in progress
339
340     // ===== Check Serial (USB) port for received configuration and status commands =====
341     if (Serial.available() > 0) {
342         // Read the incoming byte:
343         in_cmnd = Serial.read();
344         if (in_cmnd == 0x4E || in_cmnd == 0x6E) {
345             // "N" - NOP command
346             debug = false; // Ensure neither debug nor mon_cur flags are set to avoid USB conflicts with terminal protocol
347             mon_cur = false;
348             DCC.setCV(55, 0);
349             delay(5);
350             DCC.setCV(54, 0);
351             delay(5);
352             Serial.println(F("N >> OK")); // .. send ACK response
353         }
354         else if (in_cmnd == 0x53 || in_cmnd == 0x73) {
355             // "S" - Return values of all 16 CVs holding Power Breaker parameters - CV41 to CV56
356             Serial.print(F("S >> CVs - ")); // Output copy of command
357             for(i = 41; i < 57; i++){
358                 cv_value = DCC.getCV(i); // Fetch CV
359                 if (i != 56) {
360                     Serial.print(cv_value, DEC);

```



```

361     Serial.print(F(", ")); // Output as decimal values, comma-separated
362 } else {
363     Serial.println(cv_value, DEC);
364 }
365 }
366 }
367 else if (in_cmnd == 0x41 || in_cmnd == 0x61) {
368     // "A" - Return PowerBreaker Address command
369     ser_prm1 = DCC.getCV(42); // Address MSB
370     ser_prm2 = DCC.getCV(41); // Address LSB
371     t = (ser_prm1 * 256) + ser_prm2;
372     Serial.print(F("A >> Address = "));
373     Serial.println(t, DEC); // Output as decimal value
374 }
375 else if (in_cmnd == 0x56 || in_cmnd == 0x76) {
376     // "V" - Return Version command
377     Serial.print(F("V >> "));
378     cv_value = dpb_vern;
379     cv_value = cv_value >> 4;
380     Serial.print(cv_value, DEC); // Major = MS Nibble
381     Serial.print("."); // Major = MS Nibble
382     cv_value = dpb_vern;
383     cv_value = cv_value & 0x0F;
384     Serial.println(cv_value, DEC); // Minor = LS Nibble
385 }
386 if (in_cmnd == 0x47 || in_cmnd == 0x67) {
387     // "G" - Enable Debug Messages command
388     debug = true;
389     DCC.setCV(55, 0x0F); // Set debug CV to non-zero value
390     delay(5);
391     Serial.println(F("G >> OK")); // .. send ACK response
392 }
393 if (in_cmnd == 0x4B || in_cmnd == 0x6B) {
394     // "K" - Enable ADC Display command
395     mon_cur = true;
396     DCC.setCV(54, 0x0F); // Set current monitor CV to non-zero value
397     delay(5);
398     nxtadcop = millis(); // Initialise timer for output of ADC current values
399     Serial.println(F("K >> OK")); // .. send ACK response
400 }
401 else if (in_cmnd == 0x54 || in_cmnd == 0x74) {
402     // "T" - Set Power Breaker Address command - add new address - up to 4 numeric characters
403     ser_prmstr = Serial.readString(); // Get rest of input characters
404     t = ser_prmstr.length();
405     sub_prmstr = ser_prmstr.substring(0, (t - 1));

```

```

406 Serial.print(F("T"));
407 Serial.print(sub_prmstr); // Output copy of command
408 Serial.print(F(" >> "));
409 v = ser_prmstr.toInt();
410 if (v == 0 || v > 2043) {
411     Serial.println(F("Invalid Address - not in range 1 - 2043"));
412 } else {
413     t = DCC.getCV(41) + (256 * DCC.getCV(42));
414     if (t != v) {
415         ser_prm1 = highByte(v);
416         ser_prm2 = lowByte(v);
417         DCC.setCV(41, ser_prm2); // Update Accessory Address
418         delay(5); // .. if not equal to current stored value
419         DCC.setCV(42, ser_prm1);
420         delay(5);
421         DCC.setCV(121, ser_prm2); // Update Program Main Address to match
422         delay(5);
423         DCC.setCV(122, ser_prm1);
424         delay(5);
425     }
426     Serial.print(F("Address = "));
427     Serial.println(v, DEC); // .. send ACK response
428 }
429 }
430 else if (in_cmnd == 0x43 || in_cmnd == 0x63) {
431     // "C" - Set Trip Current command - add index (1-20) - up to 2 numeric characters
432     delayMicroseconds(300);
433     if (Serial.available() > 0) {
434         ser_prm1 = Serial.read(); // Get District Select parameter
435         delayMicroseconds(300);
436         ser_prmstr = Serial.readString(); // Get rest of input characters
437         t = ser_prmstr.length();
438         sub_prmstr = ser_prmstr.substring(0, (t - 1)); // Remove Newline character
439         if (ser_prm1 == 0x0A) ser_prm1 = 0x2D; // Replace Newline with '-'
440         Serial.print(F("C"));
441         Serial.print((char)ser_prm1);
442         Serial.print(sub_prmstr); // Output copy of command
443         Serial.print(F(" >> "));
444         v = ser_prmstr.toInt();
445         if (v == 0 || v > 20) {
446             Serial.println(F("Invalid Index - not in range 1 - 20"));
447         } else {
448             cv_value = lowByte(v); // Extract index value to fit byte
449             if (ser_prm1 == 0x31) {
450                 // Check CV43 value - District 1 Trip Index

```

```

451     if (cv_value != DCC.getCV(43)) {
452         sen1trip = trip_slct[cv_value]; // Update Trip Current value only if value has changed
453         DCC.setCV(43, cv_value);        // .. and load to CV memory
454         delay(5);
455     }
456     Serial.print(F("Dist 1 Trip Current = "));
457     Serial.print(cv_value*0.25, 2);
458     Serial.println(F("A"));              // .. send ACK response = Trip Current
459 } else if (ser_prm1 == 0x32) {
460     // Check CV44 value - District 2 Trip Index
461     if (cv_value != DCC.getCV(44)) {
462         sen2trip = trip_slct[cv_value]; // Update Trip Current value only if value
463         DCC.setCV(44, cv_value);        // .. has changed, and load to CV memory
464         delay(5);
465     }
466     Serial.print(F("Dist 2 Trip Current = "));
467     Serial.print(cv_value*0.25, 2);
468     Serial.println(F("A"));              // .. send ACK response = Trip Current
469 } else {
470     Serial.println(F("Invalid District - must be 1 or 2"));
471 }
472 }
473 }
474 }
475 else if (in_cmnd == 0x44 || in_cmnd == 0x64) {
476     // "D" - Set Trip Delay command - add value 5 - 255msecs
477     delayMicroseconds(300);
478     if (Serial.available() > 0) {
479         ser_prm1 = Serial.read(); // Get District Select parameter
480         delayMicroseconds(300);
481         ser_prmstr = Serial.readString(); // Get rest of input characters
482         t = ser_prmstr.length();
483         sub_prmstr = ser_prmstr.substring(0, (t - 1)); // Remove Newline character
484         if (ser_prm1 == 0x0A) ser_prm1 = 0x2D; // Replace Newline with '-'
485         Serial.print(F("D"));
486         Serial.print((char)ser_prm1);
487         Serial.print(sub_prmstr); // Output copy of command
488         Serial.print(F(" >> "));
489         v = ser_prmstr.toInt();
490         if (v < 5 || v > 255) {
491             Serial.println(F("Invalid Delay - not in range 5 - 255"));
492         } else {
493             cv_value = lowByte(v); // Adjust value to fit byte
494             if (ser_prm1 == 0x31) {
495                 // Check CV45 value - District 1 Trip Delay

```

```

496     if (cv_value != DCC.getCV(45)) {
497         ovrtime1 = cv_value;    // Update Trip Delay value only if value
498         DCC.setCV(45, cv_value); // .. has changed and load to CV memory
499         delay(5);
500     }
501     Serial.print(F("Dist 1 Trip Delay = "));
502     Serial.print(cv_value, DEC);
503     Serial.println(F("msec"));           // .. send ACK response = Trip Delay
504 } else if (ser_prm1 == 0x32) {
505     // Check CV46 value - District 2 Trip Delay
506     if (cv_value != DCC.getCV(46)) {
507         ovrtime1 = cv_value;    // Update Trip Delay value only if value
508         DCC.setCV(46, cv_value); // .. has changed, and load to CV memory
509         delay(5);
510     }
511     Serial.print(F("Dist 2 Trip Delay = "));
512     Serial.print(cv_value, DEC);
513     Serial.println(F("msec"));           // .. send ACK response = Trip Delay
514 } else {
515     Serial.println(F("Invalid District - must be 1 or 2"));
516 }
517 }
518 }
519 }
520 else if (in_cmnd == 0x52 || in_cmnd == 0x72) {
521     // "R" - Set Reconnect Delay command - add index (1-240) - units of 0.25sec
522     delayMicroseconds(300);
523     if (Serial.available() > 0) {
524         ser_prm1 = Serial.read();    // Get District Select parameter
525         delayMicroseconds(300);
526         ser_prmstr = Serial.readString(); // Get rest of input characters
527         t = ser_prmstr.length();
528         sub_prmstr = ser_prmstr.substring(0, (t - 1)); // Remove Newline character
529         if (ser_prm1 == 0x0A) ser_prm1 = 0x2D; // Replace Newline with '-'
530         Serial.print(F("R"));
531         Serial.print((char)ser_prm1);
532         Serial.print(sub_prmstr); // Output copy of command
533         Serial.print(F(" >> "));
534         v = ser_prmstr.toInt();
535         if (v == 0 || v > 240) {
536             Serial.println(F("Invalid Index - not in range 1 - 240"));
537         } else {
538             cv_value = lowByte(v); // Adjust value to fit byte
539             if (ser_prm1 == 0x31) {
540                 // Check CV47 value - District 1 Reconnect Delay Index

```

```

541     if (cv_value != DCC.getCV(47)) {
542         trycnct1 = 250 * cv_value; // Update Reconnect Delay index only if value
543         DCC.setCV(47, cv_value); // .. has changed and load to CV memory
544         delay(5);
545     }
546     Serial.print(F("Dist 1 Reconnect Delay = "));
547     Serial.print(cv_value*0.25, 2);
548     Serial.println(F("sec")); // .. send ACK response = Reconnect Delay
549 } else if (ser_prm1 == 0x32) {
550     // Check CV48 value - District 2 Reconnect Delay Index
551     if (cv_value != DCC.getCV(48)) {
552         trycnct2 = 250 * cv_value; // Update Reconnect Delay value only if value
553         DCC.setCV(48, cv_value); // .. has changed, and load to CV memory
554         delay(5);
555     }
556     Serial.print(F("Dist 2 Reconnect Delay = "));
557     Serial.print(cv_value*0.25, 2);
558     Serial.println(F("sec")); // .. send ACK response = Reconnect Delay
559 } else {
560     Serial.println(F("Invalid District - must be 1 or 2"));
561 }
562 }
563 }
564 }
565 else if (in_cmnd == 0x4D || in_cmnd == 0x6D) {
566     // "M" - Enable/Disable Auto Reconnect command - add 1 or 2 to select District
567     // .. then A to enable Auto, M to enable Manual
568     delayMicroseconds(300);
569     if (Serial.available() > 0) {
570         ser_prm1 = Serial.read(); // Get District Select parameter
571         delayMicroseconds(300);
572         ser_prmstr = Serial.readString(); // Get rest of input characters
573         if (ser_prm1 == 0x0A) ser_prm1 = 0x2D; // Replace Newline with '-'
574         t = ser_prmstr.length();
575         if (t > 1) {
576             ser_prm2 = ser_prmstr.charAt(0);
577         } else {
578             ser_prm2 = 0x2D;
579         }
580         Serial.print(F("M"));
581         Serial.print((char)ser_prm1);
582         Serial.print((char)ser_prm2); // Output copy of command
583         Serial.print(F(" >> "));
584         cv_value = 0x00;
585         if (ser_prm1 != 0x31 && ser_prm1 != 0x32) {

```

```

586 Serial.println(F("Invalid District - must be 1 or 2"));
587 cv_value = 0xFF; // Prepare to exit command
588 } else {
589   if (ser_prm2 == 0x4D || ser_prm2 == 0x6D) {
590     cv_value = 0x00; // Prepare to set Manual Reconnect
591   } else if (ser_prm2 == 0x41 || ser_prm2 == 0x61) {
592     cv_value = 0x0A; // Prepare to set Auto Reconnect
593   } else {
594     Serial.println(F("Invalid Parameter - must be A or M"));
595     cv_value = 0xFF; // Prepare to exit command
596   }
597 }
598 if (cv_value != 0xFF) {
599   if (ser_prm1 == 0x31) {
600     if (cv_value != DCC.getCV(52)) {
601       // Check CV52 value - Manual-Auto Reconnect District 1
602       if (cv_value == 0x0A) {
603         autorct1 = true; // Auto
604       } else {
605         autorct1 = false; // Manual
606       }
607       DCC.setCV(52, cv_value); // Load to CV memory only if value has changed
608       delay(5);
609     }
610     Serial.print(F("Dist 1 Reconnect = "));
611     if (autorct1) {
612       Serial.println(F("Automatic"));
613     } else {
614       Serial.println(F("Manual"));
615     }
616   } else if (ser_prm1 == 0x32) {
617     if (cv_value != DCC.getCV(53)) {
618       // Check CV53 value - Manual-Auto Reconnect District 2
619       if (cv_value == 0x0A) {
620         autorct2 = true; // Auto
621       } else {
622         autorct2 = false; // Manual
623       }
624       DCC.setCV(53, cv_value); // Load to CV memory only if value has changed
625       delay(5);
626     }
627     Serial.print(F("Dist 2 Reconnect = "));
628     if (autorct2) {
629       Serial.println(F("Automatic"));
630     } else {

```

```

631         Serial.println(F("Manual"));
632     }
633 }
634 }
635 }
636 }
637 else if (in_cmnd == 0x46 || in_cmnd == 0x66) {
638     // "F" - Reset PowerBreaker to Default CV Values command
639     DCC.setCV(50, 0); // Set CV50 = 0
640     delay(5);
641     Serial.println("F >> CVs Reset"); // .. send ACK response
642     delay(5);
643     resetFunc(); // .. then call system reset to load default CVs on restart
644 }
645 else if (in_cmnd == 0x42 || in_cmnd == 0x62) {
646     // "B" - Break Power command - add 1 = Dist 1, or 2 = Dist 2, or B = Both Districts
647     delayMicroseconds(300);
648     if (Serial.available() > 0) {
649         ser_prm1 = Serial.read(); // Get District selection
650         if (ser_prm1 == 0x0A) ser_prm1 = 0x2D; // Replace Newline with '-'
651         Serial.print(F("B"));
652         Serial.print((char)ser_prm1); // Output copy of command
653         Serial.print(F(" >> "));
654         if (autorvsr) {
655             Serial.println(F("Remove AutoReverser connections and Disable AutoReverser (UD)"));
656         } else if (ser_prm1 == 0x31) {
657             pwbreak1 = true;
658             ovrcurr1 = true;
659             killpwr1 = true;
660             digitalWrite(alm_led1, HIGH); // Switch on Alarm LED 1
661             digitalWrite(en_pout1, LOW); // Switch off Power District 1
662             enblout1 = false;
663             Serial.println(F("Dist 1 Switched Off")); // ACK response
664         } else if (ser_prm1 == 0x32) {
665             pwbreak2 = true;
666             ovrcurr2 = true;
667             killpwr2 = true;
668             digitalWrite(alm_led2, HIGH); // Switch on Alarm LED 2
669             digitalWrite(en_pout2, LOW); // Switch off Power District 2
670             enblout2 = false;
671             Serial.println(F("Dist 2 Switched Off")); // ACK response
672         } else if (ser_prm1 == 0x42 || ser_prm1 == 0x62) {
673             pwbreak1 = true;
674             pwbreak2 = true;
675             ovrcurr1 = true;

```



```

676     ovrcurr2 = true;
677     killpwr1 = true;
678     killpwr2 = true;
679     digitalWrite(alm_led1, HIGH); // Switch on Alarm LED 1
680     digitalWrite(alm_led2, HIGH); // Switch on Alarm LED 2
681     digitalWrite(en_pout1, LOW); // Switch off Power District 1
682     digitalWrite(en_pout2, LOW); // Switch off Power District 2
683     enblout1 = false;
684     enblout2 = false;
685     Serial.println(F("Dists 1 & 2 Switched Off")); // ACK response
686 } else {
687     Serial.println(F("Invalid District - must be 1, 2 or B"));
688 }
689 }
690 }
691 else if (in_cmnd == 0x50 || in_cmnd == 0x70) {
692     // "P" - Resume Power command - add 1 = Dist 1, or 2 = Dist 2, or B = Both Districts
693     delayMicroseconds(300);
694     if (Serial.available() > 0) {
695         ser_prm1 = Serial.read(); // Get District selection
696         if (ser_prm1 == 0x0A) ser_prm1 = 0x2D; // Replace Newline with '-'
697         Serial.print(F("P"));
698         Serial.print((char)ser_prm1); // Output copy of command
699         Serial.print(F(" >> "));
700         if (autorvsr) {
701             Serial.println(F("Remove AutoReverser connections and Disable AutoReverser (UD)"));
702         } else if (ser_prm1 == 0x31) {
703             pwbreak1 = false;
704             ovrcurr1 = false;
705             killpwr1 = false;
706             digitalWrite(alm_led1, LOW); // Switch off Alarm LED 1
707             digitalWrite(en_pout1, HIGH); // Switch on Power District 1
708             enblout1 = true;
709             Serial.println(F("Dist 1 Switched On")); // ACK response
710         } else if (ser_prm1 == 0x32) {
711             pwbreak2 = false;
712             ovrcurr2 = false;
713             killpwr2 = false;
714             digitalWrite(alm_led2, LOW); // Switch off Alarm LED 2
715             digitalWrite(en_pout2, HIGH); // Switch on Power District 2
716             enblout2 = true;
717             Serial.println(F("Dist 2 Switched On")); // ACK response
718         } else if (ser_prm1 == 0x42 || ser_prm1 == 0x62) {
719             pwbreak1 = false;
720             pwbreak2 = false;

```

```

721     ovrcurr1 = false;
722     ovrcurr2 = false;
723     killpwr1 = false;
724     killpwr2 = false;
725     digitalWrite(alm_led1, LOW); // Switch off Alarm LED 1
726     digitalWrite(alm_led2, LOW); // Switch off Alarm LED 2
727     digitalWrite(en_pout1, HIGH); // Switch on Power District 1
728     digitalWrite(en_pout2, HIGH); // Switch on Power District 2
729     enblout1 = true;
730     enblout2 = true;
731     Serial.println(F("Dists 1 & 2 Switched On")); // ACK response
732   } else {
733     Serial.println(F("Invalid District - must be 1, 2 or B"));
734   }
735 }
736 }
737 else if (in_cmnd == 0x55 || in_cmnd == 0x75) {
738   // "U" - Enable/Disable Auto Reverser command - add E to enable, D to disable
739   delayMicroseconds(300);
740   if (Serial.available() > 0) {
741     ser_prm1 = Serial.read(); // Get Enable/Disable parameter
742     if (ser_prm1 == 0x0A) ser_prm1 = 0x2D; // Replace Newline with '-'
743     Serial.print(F("U"));
744     Serial.print((char)ser_prm1); // Output copy of command
745     Serial.print(F(" >> "));
746     if (ser_prm1 == 0x45 || ser_prm1 == 0x65) {
747       DCC.setCV(49, 0x5A); // Set CV49 = 90
748       delay(5);
749       digitalWrite(en_pout2, LOW); // Switch off Power District 2
750       enblout2 = false;
751       digitalWrite(alm_led2, HIGH); // Switch on Alarm LED 2
752       delay(2); // Power Districts 1 & 2 outputs are linked together in anti-phase
753       digitalWrite(en_pout1, HIGH); // Switch on Power District 1
754       enblout1 = true;
755       digitalWrite(alm_led1, LOW); // Switch off Alarm LED 1
756       autorvsr = true; // AutoReverser Mode enabled
757       arev_act = 1; // Power District 1 active in AutoReverser mode
758       Serial.println(F("AutoReverser Mode Enabled")); // ACK response
759     } else if (ser_prm1 == 0x44 || ser_prm1 == 0x64) {
760       DCC.setCV(49, 0x00); // Set CV49 = 0
761       delay(5);
762       digitalWrite(en_pout1, LOW); // Switch off Power District 1
763       enblout1 = false;
764       digitalWrite(alm_led1, HIGH); // Switch on Alarm LED 1
765       digitalWrite(en_pout2, LOW); // Switch off Power District 2

```

```

766     enblout2 = false;
767     digitalWrite(alm_led2, HIGH);    // Switch on Alarm LED 2
768     autorvsr = false;
769     arev_act = 0;    // AutoReverser Mode disabled
770     Serial.println(F("AutoReverser Mode Disabled - Both Districts Switched Off")); // ACK response
771 } else {
772     Serial.println(F("Invalid Parameter - must be E to Enable or D to Disable"));
773 }
774 }
775 }
776 }
777
778 // ===== Check for Address Programming Link fitted =====
779 if ((digitalRead(adr_prog) == LOW) && (busy_dcc != 1)) {
780     if (prog_done == 0) {
781         delay(100);
782         if (digitalRead(adr_prog) == LOW) {
783             // Address Programming is active
784             digitalWrite(en_pout1, LOW);    // Switch off Power District 1
785             enblout1 = false;
786             digitalWrite(en_pout2, LOW);    // Switch off Power District 2
787             enblout2 = false;
788             digitalWrite(alm_led1, HIGH);    // Switch Alarm 1 Red LED On
789             prog_actv = 1;
790             busy_dcc = 1;
791             Serial.println(F("Address Programming Active"));
792         }
793     }
794 }
795 else if ((digitalRead(adr_prog) == LOW) && (busy_dcc == 1)) {
796     if (prog_done == 1) {
797         flash = flash + 1;
798         //Flash Alarm 1 Red LED as reminder to remove Programming Link
799         if (flash == 100){
800             digitalWrite(alm_led1, LOW);    // Switch off Alarm 1 Red LED
801         }
802         else{
803             if (flash == 200){
804                 digitalWrite(alm_led1, HIGH); // Switch on Alarm 1 Red LED
805                 flash = 0;
806             }
807         }
808     }
809 } else if ((digitalRead(adr_prog) == HIGH) && (prog_actv == 1)) {
810     delay(20);

```

```

811 // Programming Link removed - Address Programming will be abandoned
812 set_adr = 0; // Clear any entered address data
813 prog_actv = 0; // Disable Address Programming
814 prog_done = 0;
815 busy_dcc = 0; // Allow normal switch actions
816 Serial.println(F("Address Programming Inactive"));
817 digitalWrite(alm_led1, LOW); // Switch off Alarm 1 Red LED
818 delay(200);
819 resetFunc(); // Call system reset - execution stops here then returns to start
820 }
821 // ===== Start Current Monitoring =====
822
823 if (mon_cur) {
824     if (nxtadcop <= (millis() - 200)) {
825         // Print District Current Values read from Nano ADCs
826         t = analogRead(sens_in1); // Read District 1 Current
827         Serial.print(F("Dist 1 ADC = "));
828         Serial.print(t, DEC);
829         t = analogRead(sens_in2); // Read District 2 Current
830         Serial.print(F(" - Dist 2 ADC = "));
831         Serial.println(t, DEC);
832         nxtadcop = millis(); // Reset timer for next output of ADC current values
833     }
834 }
835
836 if (!autorvsr) {
837     // ==== Normal Power Breaker Operation =====
838     if ((digitalRead(adr_prog) == HIGH) && (busy_dcc != 1) && (prog_actv != 1)) {
839         // Check Power District 1
840         if (pwbreak1 && autorct1 && !killpwr1) {
841             elapsed1 = millis() - tmstart1; // Check if time to try reconnect after break
842             if (elapsed1 > trycnc1) {
843                 // Attempt reconnect if Auto Reconnect active and Not Soft Break
844                 if (debug) {
845                     Serial.print(F("OverCurrent Period Elapsed"));
846                     Serial.println(F(" - Reconnect Power District 1"));
847                 }
848                 digitalWrite(en_pout1, HIGH); // Switch on Power District 1
849                 enblout1 = true;
850                 digitalWrite(alm_led1, LOW); // Switch off Alarm LED 1
851                 digitalWrite(alm_out1, LOW); // Switch off Alarm Out 1 (Sounder)
852                 pwbreak1 = false;
853                 ovrcurr1=false;
854             }
855         }
856     }
857 }

```

```

856 else {
857     senlread = analogRead(sens_in1); // Read District 1 Current
858     if (senlread > senltrip) {
859         if (ovrcurr1) {
860             elapsed1 = millis() - tmstart1; // Over-current already detected - check if still true for set time
861             if (elapsed1 > ovrtime1) {
862                 // Initiate break - set Alarm 1
863                 digitalWrite(en_pout1, LOW); // Switch off Power District 1
864                 enblout1 = false;
865                 digitalWrite(alm_led1, HIGH); // Switch on Alarm LED 1
866                 digitalWrite(alm_out1, HIGH); // Switch on Alarm Out 1 (Sunder)
867                 if (debug) {
868                     Serial.println(F("Break Power District 1"));
869                 }
870                 pwbreak1 = true;
871                 tmstart1 = millis();
872             }
873         }
874         else {
875             ovrcurr1 = true; // Detected load > set trip current - set flag and start timer
876             tmstart1 = millis();
877         }
878     }
879     else {
880         ovrcurr1 = false; // Current has fallen below trip level - start monitoring again
881         elapsed1 = 0;
882         senlread = 0; // Clear last read District 1 current
883     }
884 }
885 // Check Power District 2
886 if (pwbreak2 && autorct2 && !killpwr2) {
887     elapsed2 = millis() - tmstart2; // Check if time to try reconnect after break
888     if (elapsed2 > trycnct2) {
889         // Attempt reconnect if Auto Reconnect active and Not Soft Break
890         if (debug) {
891             Serial.print(F("OverCurrent Period Elapsed"));
892             Serial.println(F(" - Reconnect Power District 2"));
893         }
894         digitalWrite(en_pout2, HIGH); // Switch on Power District 2
895         enblout2 = true;
896         digitalWrite(alm_led2, LOW); // Switch off Alarm LED 2
897         digitalWrite(alm_out2, LOW); // Switch off Alarm Out 2 (Sunder)
898         pwbreak2 = false;
899         ovrcurr2=false;
900     }

```

```

901 }
902 else {
903     sen2read = analogRead(sens_in2); // Read District 2 Current
904     if (sen2read > sen2trip) {
905         if (ovrcurr2) {
906             elapsed2 = millis() - tmstart2; // Over-current already detected - check if still true for set time
907             if (elapsed2 > ovrtime2) {
908                 // Initiate break - set Alarm 2
909                 digitalWrite(en_pout2, LOW); // Switch off Power District 2
910                 enblout2 = false;
911                 digitalWrite(alm_led2, HIGH); // Switch on Alarm LED 2
912                 digitalWrite(alm_out2, HIGH); // Switch on Alarm Out 2 (Sunder)
913                 if (debug) {
914                     Serial.println(F("Break Power District 2"));
915                 }
916                 pwbreak2 = true;
917                 tmstart2 = millis();
918             }
919         }
920         else {
921             ovrcurr2 = true; // Detected load > set trip current - set flag and start timer
922             tmstart2 = millis();
923         }
924     }
925     else {
926         ovrcurr2 = false; // Current has fallen below trip level - start monitoring again
927         elapsed2 = 0;
928         sen2read = 0; // Clear last read District 2 current
929     }
930 }
931 }
932 } else {
933     // ==== Operation as Auto Reverser =====
934     if ((digitalRead(adr_prog) == HIGH) && (busy_dcc != 1) && (prog_actv != 1) && (arev_act == 1)) {
935         // Check Power District 1
936         if (pwbreak1) {
937             elapsed1 = millis() - tmstart1; // Check if time to try reconnect after break
938             if (elapsed1 > trycnct1) {
939                 // Attempt reconnect if Auto Reconnect active and Not Soft Break
940                 if (debug) {
941                     Serial.print(F("OverCurrent Period Elapsed"));
942                     Serial.println(F(" - Reconnect Power District 1"));
943                 }
944                 digitalWrite(en_pout1, HIGH); // Switch on Power District 1
945                 enblout1 = true;

```

```

946     digitalWrite(alm_led1, LOW);    // Switch off Alarm LED 1
947     digitalWrite(alm_out1, LOW);    // Switch off Alarm Out 1 (Sunder)
948     pwbreak1 = false;
949     ovrcurr1=false;
950 }
951 } else {
952     sen1read = analogRead(sens_in1); // Read District 1 Current
953     if (sen1read > sen1trip) {
954         if (ovrcurr1) {
955             elapsed1 = millis() - tmstart1;    // Over-current already detected - check if still true for set time
956             if (elapsed1 > ovrtime1) {
957                 // Initiate AutoReverser switch to District 2 - disable District 1 output and set Alarm 1
958                 digitalWrite(en_pout1, LOW);    // Switch off Power District 1
959                 enblout1 = false;
960                 digitalWrite(alm_led1, HIGH);    // Switch on Alarm LED 1 (probably not used when in AutoReverser mode)
961                 digitalWrite(alm_out1, HIGH);    // Switch on Alarm Out 1 (Sunder, " ")
962                 if (tmautosw == 0) {tmautosw = millis() - (6 * ovrtime1);} // Set tmautosw for first time through main loop
963                 chkautsw = millis() - tmautosw; // Check if District switched again within (4 * ovrtime1) after previous switch
964                 if (chkautsw < (4 * ovrtime1)) {
965                     pwbreak1 = true; // Prepare to retry switching Power District 1 on after delay
966                     tmstart1 = millis();
967                     if (debug) {
968                         Serial.print(F("Break Power Districts 1 & 2")); // Power District 2 is already off
969                         Serial.println(F(" - In Power District 1 - Both to reconnect"));
970                     }
971                 } else {
972                     delay(2);
973                     digitalWrite(en_pout2, HIGH); // Switch on Power District 2
974                     enblout2 = true;
975                     digitalWrite(alm_led2, LOW);    // Switch off Alarm LED 2 (probably not used when in AutoReverser mode)
976                     digitalWrite(alm_out2, LOW);    // Switch off Alarm Out 2 (Sunder, " ")
977                     tmautosw = millis(); // Start check period for continuing overload on newly active Power District
978                     arev_act = 2; // Set power District 2 as active
979                     if (debug) {
980                         Serial.println(F("Break Power District 1 - District 2 Active"));
981                     }
982                 }
983             }
984         } else {
985             ovrcurr1 = true; // Detected load > set trip current - set flag and start timer
986             tmstart1 = millis();
987             if (debug) {
988                 Serial.println(F("Overcurrent Detected District 1"));
989             }
990         }
991     }

```



```

991     } else {
992         ovrcurr1 = false; // Current has fallen below trip level - start monitoring again
993         elapsed1 = 0;
994         sen1read = 0;      // Clear last read District 1 current
995     }
996 }
997 }
998 if ((digitalRead(adr_prog) == HIGH) && (busy_dcc != 1) && (prog_actv != 1) && (arev_act == 2)) {
999     // Check Power District 2
1000     if (pwbreak2) {
1001         elapsed2 = millis() - tmstart2; // Check if time to try reconnect after break
1002         if (elapsed2 > trycnct2) {
1003             // Attempt reconnect if Auto Reconnect active and Not Soft Break
1004             if (debug) {
1005                 Serial.print(F("OverCurrent Period Elapsed"));
1006                 Serial.println(F(" - Reconnect Power District 2"));
1007             }
1008             digitalWrite(en_pout2, HIGH); // Switch on Power District 2
1009             enblout2 = true;
1010             digitalWrite(alm_led2, LOW); // Switch off Alarm LED 2
1011             digitalWrite(alm_out2, LOW); // Switch off Alarm Out 2 (Sounder)
1012             pwbreak2 = false;
1013             ovrcurr2=false;
1014         }
1015     }
1016     else {
1017         sen2read = analogRead(sens_in2); // Read District 2 Current
1018         if (sen2read > sen2trip) {
1019             if (ovrcurr2) {
1020                 elapsed2 = millis() - tmstart2; // Over-current already detected - check if still true for set time
1021                 if (elapsed2 > ovrtime2) {
1022                     // Initiate AutoReverser switch to District 2 - disable District 1 output and set Alarm 1
1023                     digitalWrite(en_pout2, LOW); // Switch off Power District 2
1024                     enblout2 = false;
1025                     digitalWrite(alm_led2, HIGH); // Switch on Alarm LED 2 (probably not used when in AutoReverser mode)
1026                     digitalWrite(alm_out2, HIGH); // Switch on Alarm Out 2 (Sounder, " ")
1027                     if (tmautosw == 0) {tmautosw = millis() - (6 * ovrtime2);} // Set tmautosw for first time through main loop
1028                     chkautsw = millis() - tmautosw; // Check if District switched again within (4 * ovrtime1) after previous switch
1029                     if (chkautsw < (4 * ovrtime2)) {
1030                         pwbreak2 = true; // Prepare to retry switching Power District 2 on after delay
1031                         tmstart2 = millis();
1032                         if (debug) {
1033                             Serial.print(F("Break Power Districts 1 & 2")); // Power District 1 is already off
1034                             Serial.println(F(" - In Power District 2 - Both to reconnect"));
1035                         }

```

```

1036     } else {
1037         delay(2);
1038         digitalWrite(en_pout1, HIGH); // Switch on Power District 1
1039         enblout1 = true;
1040         digitalWrite(alm_led1, LOW); // Switch off Alarm LED 1 (probably not used when in AutoReverser mode)
1041         digitalWrite(alm_out1, LOW); // Switch off Alarm Out 1 (Sounder, " ")
1042         tmautosw = millis(); // Start check period for continuing overload on newly active Power District
1043         arev_act = 1; // Set power District 1 as active
1044         if (debug) {
1045             Serial.println(F("Break Power District 2 - District 1 Active"));
1046         }
1047     }
1048 }
1049 }
1050 else {
1051     ovrcurr2 = true; // Detected load > set trip current - set flag and start timer
1052     tmstart2 = millis();
1053     if (debug) {
1054         Serial.println(F("Overcurrent Detected District 1"));
1055     }
1056 }
1057 }
1058 else {
1059     ovrcurr2 = false; // Current has fallen below trip level - start monitoring again
1060     elapsed2 = 0;
1061     sen2read = 0; // Clear last read District 2 current
1062 }
1063 }
1064 }
1065 }
1066
1067 // ===== Check for Manual Reset inputs active to reconnect Power Districts =====
1068 if ((digitalRead(man_rst1) == LOW) && (pwbreak1 || killpwr1)) {
1069     delay(30); // Only acted upon if District 1 power break is currently active
1070     if (digitalRead(man_rst1) == LOW) {
1071         while (digitalRead(man_rst1) == LOW){
1072             // Reset switch 1 pressed
1073         } // Wait for Reset switch 1 to be released
1074         // Reset Power District 1 - irrespective of whether Auto Reset is active or not
1075         pwbreak1 = false;
1076         ovrcurr1 = false;
1077         killpwr1 = false;
1078         digitalWrite(alm_led1, LOW); // Switch off Alarm LED 1
1079         digitalWrite(alm_out1, LOW); // Switch off Alarm Out 1 (Sounder)
1080         digitalWrite(en_pout1, HIGH); // Switch on Power District 1

```

```

1081     enblout1 = true;
1082 }
1083 }
1084 if ((digitalRead(man_rst2) == LOW) && (pwbreak2 || killpwr2)) {
1085     delay(30); // Only acted upon if District 2 power break is currently active
1086     if (digitalRead(man_rst2) == LOW) {
1087         while (digitalRead(man_rst2) == LOW){
1088             // Reset switch 2 pressed
1089         } // Wait for Reset switch 2 to be released
1090         // Reset Power District 2
1091         pwbreak2 = false;
1092         ovrcurr2 = false;
1093         killpwr2 = false;
1094         digitalWrite(alm_led2, LOW); // Switch off Alarm LED 2
1095         digitalWrite(alm_out2, LOW); // Switch off Alarm Out 2 (Sounder)
1096         digitalWrite(en_pout2, HIGH); // Switch on Power District 2
1097         enblout2 = true;
1098     }
1099 }
1100 } // End main loop()
1101
1102 //*****
1103
1104 // This function is called whenever a normal DCC Turnout Packet is received in Output Addressing Mode
1105 extern void notifyDccAccTurnoutOutput( uint16_t Addr, uint8_t Direction, uint8_t OutputPower ) {
1106     byte adr_lsb;
1107     byte adr_msb;
1108     if (prog_actv == 1 && prog_done == 0) {
1109         // Address Programming is active - acceptable addresses, 0001 to 2043
1110         // Received Address is accepted as Output Address for the Dual Power Breaker
1111         set_adr = Addr;
1112         if ((set_adr > 0) && (set_adr < 2044)) {
1113             Serial.print("Address = ");
1114             Serial.print(Addr, DEC);
1115             Serial.println(" Entered");
1116         } else {
1117             set_adr = 0;
1118             Serial.println("Entered Address must be between 1 and 2043");
1119         }
1120         // Transfer entered Output Address (if any) to CV41, 42 and CV121, 122
1121         if (set_adr != 0) {
1122             adr_lsb = lowByte(set_adr);
1123             adr_msb = highByte(set_adr);
1124             DCC.setCV(41, adr_lsb);
1125             delay(5);

```

```
1126     DCC.setCV(42, adr_msb);
1127     delay(5);
1128     Serial.print("Accessory Address Stored = ");
1129     Serial.println(set_adr, DEC);
1130     DCC.setCV(121, adr_lsb);
1131     delay(5);
1132     DCC.setCV(122, adr_msb);
1133     delay(5);
1134     Serial.print("Program-on-Main Address Stored = ");
1135     Serial.println(set_adr, DEC);
1136 }
1137 set_adr = 0;    // Clear any entered address data
1138 prog_done = 1; // Prevent re-entry or setup operations until programming link removed
1139 flash = 0;     // Prepare to blink Red LED
1140     Serial.println("Address Programming Complete - Remove Programming Link");
1141 }
1142 }
1143
```