

```

1  # =====
2  # FILE: DnB_Sensor.pm                                     7/02/2020
3  #
4  # SERVICES:  DnB SENSOR FUNCTIONS
5  #
6  # DESCRIPTION:
7  #   This perl module provides sensor related functions used by the DnB model
8  #   railroad control program.
9  #
10 # PERL VERSION: 5.24.1
11 #
12 # =====
13 use strict;
14 # -----
15 # Package Declaration
16 # -----
17 package DnB_Sensor;
18 require Exporter;
19 our @ISA = qw(Exporter);
20
21 our @EXPORT = qw(
22     I2C_InitSensorDriver
23     KeypadChildProcess
24     ButtonChildProcess
25     PositionChildProcess
26     GetSensorBit
27     ReadKeypad
28     GetButton
29     TestSensorBits
30     TestSensorTones
31     TestKeypad
32 );
33
34 use DnB_Message;
35 use Time::HiRes qw(gettimeofday sleep);
36
37 # =====
38 # FUNCTION:  I2C_InitSensorDriver
39 #
40 # DESCRIPTION:
41 #   This routine initializes the sensor I2C driver board on the DnB model
42 #   railroad. It sets parameters that are common to all sensor ports. The
43 #   I/O PI Plus board utilizes two MCP23017 chips. Each chip has two 8 bit
44 #   ports of configurable GPIO pins. Each chip is configured based on the
45 #   values in the %SensorChip hash.
46 #
47 #   Chip 3 is initialized for a 'Storm K Range' 4x4 keypad. MCP23017 GPIO
48 #   pins are direct connected as follows. Row (letter) GPIOs are set as
49 #   input + pullup. Columns set as outputs.
50 #
51 #   The %SensorChip{chip}{'Obj'} hash key is written with the driver object
52 #   pointer for use in sensor data reading.
53 #
54 #       Keypad pin:      1 2 3 4 5 6 7 8
55 #       Keypad col/row:  A B 1 2 3 4 D C
56 #       GPIOA pin:      3 4 5 6 7 8 9 10
57 #       GPIOA bit:      0 1 2 3 4 5 6 7
58 #       GPIODIRA:       1 1 0 0 0 0 1 1      1 = Input, 0 = Output
59 #
60 # CALLING SYNTAX:

```

```

61 # $result = &I2C_InitSensorDriver($ChipNmbr, \%MCP23017, \%SensorChip);
62 #
63 # ARGUMENTS:
64 # $ChipNmbr          Chip number being initialized.
65 # $MCP23017          Pointer to %MCP23017 internal register definitions
66 # $SensorChip        Pointer to %SensorChip hash.
67 #
68 # RETURNED VALUES:
69 # 0 = Success, 1 = Error.
70 #
71 # ACCESSED GLOBAL VARIABLES:
72 # None.
73 # =====
74 sub I2C_InitSensorDriver {
75
76     my($ChipNmbr, $MCP23017, $SensorChip) = @_;
77     my($driver);
78     my(@temp) = ();
79
80     &DisplayDebug(2, "I2C_InitSensorDriver, ChipNmbr: $ChipNmbr I2C_Address: " .
81                   sprintf("0x%.2X", $$SensorChip{$ChipNmbr}{'Addr'}));
82
83     $driver = RPi::I2C->new($$SensorChip{$ChipNmbr}{'Addr'});
84     unless ($driver->check_device($$SensorChip{$ChipNmbr}{'Addr'})) {
85         &DisplayError("I2C_InitSensorDriver, Failed to initialize I2C address: " .
86                     sprintf("0x%.2X", $$SensorChip{$ChipNmbr}{'Addr'}) .
87                     " - $!");
88         return 1;
89     }
90     $$SensorChip{$ChipNmbr}{'Obj'} = $driver;
91
92     # Set MCP23017 BANK (bit7) = 0 (sets MCP23017 register addresses)
93     $driver->write_byte(0x00, $$MCP23017{'IOCON'});
94
95     # Set port direction bits.
96     $driver->write_byte($$SensorChip{$ChipNmbr}{'DirA'}, $$MCP23017{'IODIRA'});
97     $driver->write_byte($$SensorChip{$ChipNmbr}{'DirB'}, $$MCP23017{'IODIRB'});
98
99     # Set port polarity bits.
100    $driver->write_byte($$SensorChip{$ChipNmbr}{'PolA'}, $$MCP23017{'IOPOLA'});
101    $driver->write_byte($$SensorChip{$ChipNmbr}{'PolB'}, $$MCP23017{'IOPOLB'});
102
103    # Set port pullup enable bits.
104    $driver->write_byte($$SensorChip{$ChipNmbr}{'PupA'}, $$MCP23017{'GPPUA'});
105    $driver->write_byte($$SensorChip{$ChipNmbr}{'PupB'}, $$MCP23017{'GPPUB'});
106
107    # Build temporary array for debug output.
108    push(@temp, $driver->read_byte($$MCP23017{'IOCON'})); # Get current IOCON.
109    push(@temp, $driver->read_byte($$MCP23017{'IODIRA'})); # Get current IODIRA.
110    push(@temp, $driver->read_byte($$MCP23017{'IODIRB'})); # Get current IODIRB.
111    push(@temp, $driver->read_byte($$MCP23017{'IOPOLA'})); # Get current IOPOLA.
112    push(@temp, $driver->read_byte($$MCP23017{'IOPOLB'})); # Get current IOPOLB.
113
114    &DisplayDebug(2, "I2C_InitSensorDriver - Initialized IODIRA " .
115                  "IODIRB IOPOLA IOPOLB IOCON: " .
116                  sprintf("0x%.2X 0x%.2X 0x%.2X 0x%.2X 0x%.2X", @temp));
117
118    # Debug code.
119    while (1) {
120        $message = "I2C Address: " . sprintf("0x%.2X", $$SensorChip{$ChipNmbr}

```

```

121 #                                     {'Addr'}));
122 #     $message = $message . "    GPIOB: " . sprintf("%0.8b",
123 #                                     $driver->read_byte($MCP23017{'GPIOB'}));
124 #     $message = $message . "    GPIOA: " .
125 #                                     sprintf("%0.8b", $driver->read_byte($MCP23017{'GPIOA'}));
126 #     &DisplayMessage("I2C_InitSensorDriver - $message");
127 #     sleep 1;
128 # }
129 # exit(0);
130
131 return 0;
132 }
133
134 # =====
135 # FUNCTION: KeypadChildProcess
136 #
137 # DESCRIPTION:
138 #     This routine is launched as a child process during main program startup
139 #     and is used to return user input from the 'Storm K Range' 4x4 button
140 #     keypad. This keypad is connected to a MCP23017 port as follows.
141 #
142 #         row/col   1   2   3   4
143 #         |         |   |   |   |
144 #     A  -----0---1---2---3---
145 #         |         |   |   |   |
146 #     B  -----4---5---6---7---
147 #         |         |   |   |   |
148 #     C  -----8---9---A---B---
149 #         |         |   |   |   |
150 #     D  -----C---D---E---F---
151 #         |         |   |   |   |
152 #
153 #     Keypad pin:      1 2 3 4 5 6 7 8
154 #     Keypad col/row:  A B 1 2 3 4 D C
155 #     GPIOA pin:      3 4 5 6 7 8 9 10
156 #     GPIOA bit:      0 1 2 3 4 5 6 7
157 #     GPIODIRA:       1 1 0 0 0 0 1 1      1 = Input, 0 = Output
158 #
159 #     A dedicated child process is used to improve the reliability of keypad
160 #     entries. Forks::Super is used between the parent and child to read data
161 #     from the child's STDERR filehandle. Do no other output to STDERR within
162 #     this routine. DisplayMessage and DisplayDebug are permitted since they
163 #     use STDOUT for messaging.
164 #
165 #     The %KeypadData hash provides keypad specific data and state information.
166 #     Data is accessed using a hash index specified in the $Keypad variable.
167 #
168 #     The %cols hash holds 4 values, each has one of the keypad column driver
169 #     bits low. These bits are configured as outputs by I2C_InitSensorDriver.
170 #     The %col hash keys map to the %matrix hash primary keys.
171 #
172 #     The %matrix hash contains the resulting button value for each of the 16
173 #     combinations of col/row. I2C_InitSensorDriver configures the input pins
174 #     with pullup enabled which results in a value of 0xC3 when no button is
175 #     pressed. The hash secondary key corresponds to 0xC3 with one of the input
176 #     bits low. Input is ignored if multiple buttons are pressed.
177 #
178 #     Note that the physical rotational orientation of the keypad, that is the
179 #     keys that are the top row, will necessitate changes to the %matrix hash
180 #     values. Current matrix values are for the orientation with the keypad

```

```

181 # connector at the 6 o'clock position.
182 #
183 # CALLING SYNTAX:
184 # $KeypadChildPid = fork {os_priority => 4, sub => \&KeypadChildProcess,
185 # child_fh => "err socket",
186 # args => [ $Keypad, \%KeypadData, \%MCP23017,
187 # \%SensorChip ] };
188 #
189 # $read_key = Forks::Super::read_stderr($KeypadChildPid);
190 #
191 # ARGUMENTS:
192 # $Keypad KeypadData entry to use.
193 # $KeypadData Pointer to %KeypadData hash.
194 # $MCP23017 Pointer to MCP23017 internal register definitions
195 # $SensorChip Pointer to %SensorChip hash.
196 #
197 # RETURNED VALUES:
198 # 0-F = Pressed button via read_stderr.
199 #
200 # ACCESSED GLOBAL VARIABLES:
201 # $main::ChildName
202 # =====
203 sub KeypadChildProcess {
204     my($Keypad, $KeypadData, $MCP23017, $SensorChip) = @_;
205     my($row, $button);
206     my($chip) = $$KeypadData{$Keypad}{'Chip'};
207     my(%cols) = (1 => 0xFB, 2 => 0xF7, 3 => 0xEF, 4 => 0xDF);
208     my(%matrix) = (1 => { 0xC2 => '0', 0xC1 => '4', 0x43 => '8', 0x83 => 'C'},
209                    2 => { 0xC2 => '1', 0xC1 => '5', 0x43 => '9', 0x83 => 'D'},
210                    3 => { 0xC2 => '2', 0xC1 => '6', 0x43 => 'A', 0x83 => 'E'},
211                    4 => { 0xC2 => '3', 0xC1 => '7', 0x43 => 'B', 0x83 => 'F'});
212
213     $main::ChildName = 'KeypadChildProcess';
214     &DisplayMessage("KeypadChildProcess started.");
215     &DisplayDebug(2, "Keypad: $Keypad chip: $chip");
216
217     if ($$SensorChip{$chip}{'Obj'} == 0) {
218         &DisplayMessage("**** error: KeypadChildProcess, No SensorChip object " .
219                        "for chip $chip. Call I2C_InitSensorDriver routine first.");
220         &DisplayMessage("KeypadChildProcess terminated.");
221         sleep 2;
222         exit(0);
223     }
224
225     while(1) {
226         $button = -1;
227         foreach my $col (1,2,3,4) {
228             $$SensorChip{$chip}{'Obj'}->write_byte($cols{$col},
229                                                    $$MCP23017{ $$KeypadData{$Keypad}{'Col'} });
230             sleep 0.02; # Delay for button debounce.
231             $row = $$SensorChip{$chip}{'Obj'}->read_byte($$MCP23017{
232                                                         $$KeypadData{$Keypad}{'Row'} }) & 0xC3;
233             &DisplayDebug(3, "ReadKeypad, Keypad: $Keypad col: " .
234                            sprintf("%0.8b", $cols{$col}) . " row: " .
235                            sprintf("%0.8b", $row));
236
237             # Process if valid single button keypress. Ignore held down button.
238             if ($row == 0xC2 or $row == 0xC1 or $row == 0x43 or $row == 0x83) {
239                 $button = $matrix{$col}{$row}; # Get keypress result value.
240                 if ($button != $$KeypadData{$Keypad}{'Last'}) {

```

```

241         $$KeypadData{$Keypad}{'Last'} = $button;
242         print STDERR "$button";           # Send key press.
243         &DisplayDebug(3, "ReadKeypad, button '$button' pressed.");
244     }
245     last;
246 }
247 }
248
249 # Clear 'Last' if no button is pressed.
250 if ($button == -1 and $$KeypadData{$Keypad}{'Last'} != -1) {
251     $$KeypadData{$Keypad}{'Last'} = -1;
252     sleep 0.02;           # Delay for button debounce.
253 }
254 sleep 0.1;               # Loop delay.
255 }
256
257 &DisplayMessage("KeypadChildProcess terminated.");
258 sleep 2;
259 exit(0);
260 }
261
262 # =====
263 # FUNCTION:  ButtonChildProcess
264 #
265 # DESCRIPTION:
266 #   This routine is launched as a child process during main program startup
267 #   and is used to return user input from the 'Storm K Range' 1x4 button
268 #   keypad. This keypad is connected to a MCP23017 port as follows.
269 #
270 #       button      D   C   B   A
271 #               |   |   |   |
272 #       common  --0---0---0---0
273 #
274 #       ButtonPad 1:      c  D C B A      ButtonPad 2:      c  D C B A
275 #       Button pin:       1  2 3 4 5      Button pin:       1  2 3 4 5
276 #       GPIOA pin:       2  6 5 4 3      GPIOA pin:       2 10 9 8 7
277 #       GPIOA bit:       0  1 2 3      GPIOA bit:       4  5 6 7
278 #
279 #   A dedicated child process is used to improve the reliability of a double
280 #   button press. Forks::Super is used between the parent and child to read
281 #   data from the child's STDERR filehandle. Do no other output to STDERR
282 #   within this routine. DisplayMessage and DisplayDebug are permitted since
283 #   they use STDOUT for messaging.
284 #
285 #   Two button data messages are generated, single press (s<num>) and double
286 #   press (d<num>). <num> is the button index in the %ButtonData hash. The
287 #   parent must read the child's data at a rate greater than the expected
288 #   user input rate.
289 #
290 #   Multiple button press events may be present in a message, e.g. 's01d01'.
291 #   Check first for d01 input and discard the s01 input if present.
292 #
293 #   The %ButtonData{<num>}{'Obj'} references must be set prior to launching
294 #   this child process.
295 #
296 # CALLING SYNTAX:
297 #   $ButtonChildPid = fork {os_priority => 4, sub => \&ButtonChildProcess,
298 #                           child_fh => "err socket",
299 #                           args => [ \%ButtonData, \%MCP23017, \%SensorChip ] };
300 #

```

```

301 # $read_button = Forks::Super::read_stderr($ButtonChildPid);
302 #
303 # ARGUMENTS:
304 # $ButtonData          Pointer to %ButtonData hash.
305 # $MCP23017            Pointer to MCP23017 internal register definitions
306 # $SensorChip          Pointer to %SensorChip hash.
307 #
308 # RETURNED VALUES:
309 # s<num> - Button <num> has been single pressed.
310 # d<num> - Button <num> has been double pressed.
311 #
312 # ACCESSED GLOBAL VARIABLES:
313 # $main::ChildName
314 # =====
315 sub ButtonChildProcess {
316     my($ButtonData, $MCP23017, $SensorChip) = @_;
317     my($port, $mask, $chip, $check);
318
319     $main::ChildName = 'ButtonChildProcess';
320     &DisplayMessage("ButtonChildProcess started.");
321
322     while(1) {
323         foreach my $button (keys %$ButtonData) {
324             next if ($button == 0xFF); # Ignore shutdown button entry
325             $chip = $$ButtonData{$button}{'Chip'};
326             if ($$ButtonData{$button}{'Bit'} =~ m/^(GPIO.)(\d)/) {
327                 $port = $1;
328                 $mask = 1 << $2;
329
330                 # Read the port and isolate the bit value.
331                 $check = $$SensorChip{$chip}{'Obj'}->read_byte($$MCP23017{$port});
332                 $check = $check & $mask;
333
334                 # 'Last' is used to handle a held down button. Only use the
335                 # transition from 0 to 1 as a button press.
336                 if ($check != 0) {
337                     if ($$ButtonData{$button}{'Last'} == 1) {
338                         $$ButtonData{$button}{'PressTime'} = gettimeofday;
339                         next;
340                     }
341                     if ((gettimeofday - $$ButtonData{$button}{'PressTime'}) < 1) {
342                         print STDERR "d${button}"; # Send double press.
343                         $$ButtonData{$button}{'PressTime'} = 0; # New press cycle.
344                         &DisplayDebug(3, "ButtonChildProcess, button: d${button}");
345                     }
346                     else {
347                         print STDERR "s${button}"; # Send single press
348                         $$ButtonData{$button}{'PressTime'} = gettimeofday;
349                         &DisplayDebug(3, "ButtonChildProcess, button: s${button}");
350                     }
351                     $$ButtonData{$button}{'Last'} = 1;
352                 }
353                 else {
354                     $$ButtonData{$button}{'Last'} = 0; # Button released.
355                 }
356             }
357         }
358         sleep 0.05; # Loop delay.
359     }
360 }

```

```

361     &DisplayMessage("ButtonChildProcess terminated.");
362     sleep 2;
363     exit(0);
364 }
365
366 # =====
367 # FUNCTION: PositionChildProcess
368 #
369 # DESCRIPTION:
370 #     This routine is launched as a child process during main program startup.
371 #     It periodically reads the train hold position sensors associated with the
372 #     holdover tracks and sets the appropriate panel LEDs to provide a visual
373 #     indication of train position in these hidden tracks. Warning point (yellow)
374 #     and stop point (red) LEDs are used.
375 #
376 # CALLING SYNTAX:
377 #     $result = &PositionChildProcess(\%SensorBit, \%PositionLed, \%SensorChip,
378 #                                     \%MCP23017);
379 #
380 # ARGUMENTS:
381 #     $SensorBit      Pointer to %SensorBit hash.
382 #     $PositionLed    Pointer to %PositionLed hash.
383 #     $SensorChip      Pointer to %SensorChip hash.
384 #     $MCP23017       Pointer to MCP23017 internal register definitions.
385 #
386 # RETURNED VALUES:
387 #     0 = Success, 1 = Error.
388 #
389 # ACCESSED GLOBAL VARIABLES:
390 #     $main::ChildName
391 # =====
392 sub PositionChildProcess {
393     my($SensorBit, $PositionLed, $SensorChip, $MCP23017) = @_;
394     my($chip, $port, $pos, $senBit, $ledBits);
395
396     $main::ChildName = 'PositionChildProcess';
397     &DisplayMessage("PositionChildProcess started.");
398
399     while(1) {
400         foreach my $led (sort keys(%$PositionLed)) {
401             $chip = $$SensorBit{$led}{'Chip'};
402             if ($$SensorBit{$led}{'Bit'} =~ m/^(GPIO.)(\d)/) {
403                 $port = $1;
404                 $pos = $2;
405
406                 # Read sensor port and isolate the bit value.
407                 $senBit = $$SensorChip{$chip}{'Obj'}->read_byte($$MCP23017{$port});
408                 $senBit = ($senBit >> $pos) & 1; # Position and isolate.
409
410                 $chip = $$PositionLed{$led}{'Chip'};
411                 if ($$PositionLed{$led}{'Bit'} =~ m/^(GPIO.)(\d)/) {
412                     $port = $1;
413                     $pos = $2;
414
415                     # Update associated LED bit value.
416                     $ledBits = $$SensorChip{$chip}{'Obj'}->read_byte(
417                         $$MCP23017{$port});
418                     $ledBits = $ledBits & ~(1 << $pos); # Clear bit position.
419                     $ledBits = $ledBits | ($senBit << $pos); # Set bit position.
420                     $$SensorChip{$chip}{'Obj'}->write_byte($ledBits,

```

```

421         $$MCP23017{ $$PositionLed{$led}{'Olat'}}});
422     }
423 }
424 }
425     sleep 0.5;           # Loop delay.
426 }
427
428 &DisplayMessage("PositionChildProcess terminated.");
429 sleep 1;
430 exit(0);
431 }
432
433 # =====
434 # FUNCTION:  GetSensorBit
435 #
436 # DESCRIPTION:
437 #     This routine returns the current value of the specified sensor bit. The
438 #     proper SensorState hash index is determined based on the requested bit
439 #     number. The bit number must include leading zero (0) if less than 10 for
440 #     proper index key in %SensorBit hash.
441 #
442 # CALLING SYNTAX:
443 #     $result = &GetSensorBit($BitNumber, \%SensorBit, \%SensorState);
444 #
445 # ARGUMENTS:
446 #     $BitNumber           Bit position to check (index in %SensorBit)
447 #     $SensorBit           Pointer to %SensorBit hash.
448 #     $SensorState        Pointer to %SensorState hash.
449 #
450 # RETURNED VALUES:
451 #     Bit value: 0 or 1
452 #
453 # ACCESSED GLOBAL VARIABLES:
454 #     None.
455 # =====
456 sub GetSensorBit {
457     my($BitNumber, $SensorBit, $SensorState) = @_;
458     my($bitMask) = 1 << ($BitNumber % 16);
459
460     return 1 if ($$SensorState{ $$SensorBit{$BitNumber}{'Chip'}} & $bitMask);
461
462     return 0;
463 }
464
465 # =====
466 # FUNCTION:  TestSensorBits
467 #
468 # DESCRIPTION:
469 #     This routine displays the sensor state bits on the console. The user can
470 #     manually activate each sensor and observe the expected result. This test
471 #     loops indefinitely until the user enters ctrl+c.
472 #
473 # CALLING SYNTAX:
474 #     $result = &TestSensorBits($Range, \%MCP23017, \%SensorChip, \%SensorState);
475 #
476 # ARGUMENTS:
477 #     $Range               Chip number or range to use.
478 #     $MCP23017            Pointer to MCP23017 internal register definitions
479 #     $SensorChip          Pointer to %SensorChip hash.
480 #     $SensorState        Pointer to %SensorState hash.

```



```

481 #
482 # RETURNED VALUES:
483 # 0 = Success, 1 = Error.
484 #
485 # ACCESSED GLOBAL VARIABLES:
486 # $main::MainRun
487 # =====
488 sub TestSensorBits {
489     my($Range, $MCP23017, $SensorChip, $SensorState) = @_;
490     my($chip, $start, $end, $msg, @chipList);
491     my($cntr) = 0;
492
493     &DisplayDebug(2, "TestSensorBits, Entry ... Range: '$Range'");
494
495     if ($Range =~ m/(\d+):(\d+)/) { # Range specified.
496         $start = $1;
497         $end = $2;
498         if ($start > $end or $start < 1 or $start > 4 or $end < 1 or $end > 4) {
499             &DisplayError("TestSensorBits, invalid sensor chip range: '$Range'");
500             return 1;
501         }
502         for ($chip = $start; $chip <= $end; $chip++) {
503             push (@chipList, $chip);
504         }
505     }
506     else {
507         @chipList = split(",", $Range);
508     }
509     &DisplayDebug(1, "TestSensorBits, chipList: '@chipList'");
510
511     &DisplayMessage("TestSensorBits - enter ctrl+c to exit.\n");
512     &DisplayMessage("TestSensorBits ----- " .
513         "bit 76543210      bit 76543210");
514     while($main::MainRun) {
515         foreach my $chip (@chipList) {
516             $msg = sprintf("%0.6d", $cntr); # Show program activity on console.
517             if (exists($$SensorChip{$chip})) {
518                 $$SensorState{$chip} =
519                     ($$SensorChip{$chip}{'Obj'}->read_byte($$MCP23017{'GPIOB'}) << 8) |
520                     ($$SensorChip{$chip}{'Obj'}->read_byte($$MCP23017{'GPIOA'}));
521
522                 $msg = $msg . " I2C Address: " .
523                     sprintf("0x%.2X", $$SensorChip{$chip}{'Addr'});
524                 $msg = $msg . " GPIOB: " .
525                     sprintf("%0.8b", ($$SensorState{$chip} >> 8));
526                 $msg = $msg . " GPIOA: " .
527                     sprintf("%0.8b", ($$SensorState{$chip} & 0xFF));
528                 &DisplayMessage("TestSensorBits - $msg");
529             }
530             else {
531                 &DisplayError("TestSensorBits, invalid sensor range: '$Range'");
532                 return 1;
533             }
534         }
535         $cntr++;
536         sleep 1;
537     }
538     return 0;
539 }
540

```

```

541 # =====
542 # FUNCTION: TestSensorTones
543 #
544 # DESCRIPTION:
545 #   This routine tests all sensors. A sensor ID number of tones are sounded
546 #   when a sensor becomes active and a double tone is sounded when a sensor
547 #   becomes inactive. This facilitates operability testing of the layout
548 #   remote sensors; e.g. by manually blocking the IR light path. This test
549 #   loops indefinitely until the user enters ctrl+c.
550 #
551 # CALLING SYNTAX:
552 #   $result = &TestSensorTones(\%MCP23017, \%SensorChip, \%SensorState,
553 #                               \%SensorBit);
554 #
555 # ARGUMENTS:
556 #   $MCP23017          Pointer to MCP23017 internal register definitions
557 #   $SensorChip         Pointer to %SensorChip hash.
558 #   $SensorState        Pointer to %SensorState hash.
559 #   $SensorBit          Pointer to %SensorBit hash.
560 #
561 # RETURNED VALUES:
562 #   0 = Success, 1 = Error.
563 #
564 # ACCESSED GLOBAL VARIABLES:
565 #   $main::MainRun
566 # =====
567 sub TestSensorTones {
568     my($MCP23017, $SensorChip, $SensorState, $SensorBit) = @_;
569     my($bitNbr, $cur, $prev, $x, $y);
570     my(%localSensorState) = ('1' => 0, '2' => 0);
571
572     &DisplayDebug(2, "TestSensorTones, Entry ...");
573
574     &DisplayMessage("TestSensorTones, =====");
575     &DisplayMessage("TestSensorTones, Waiting for change in sensor " .
576                    "bits 0-31 ...");
577     while($main::MainRun) {
578
579         # Get all sensor states.
580         $localSensorState{'1'} =
581             ($SensorChip{'1'}{'Obj'}->read_byte($MCP23017{'GPIOB'}) << 8) |
582             $SensorChip{'1'}{'Obj'}->read_byte($MCP23017{'GPIOA'});
583         $localSensorState{'2'} =
584             ($SensorChip{'2'}{'Obj'}->read_byte($MCP23017{'GPIOB'}) << 8) |
585             $SensorChip{'2'}{'Obj'}->read_byte($MCP23017{'GPIOA'});
586
587         # If a bit has changed, report change.
588         for ($x = 0; $x < 32; $x++) {
589             $bitNbr = sprintf("%0.2d", $x);
590             next if ($SensorBit{$bitNbr}{'Desc'} =~ m/spare/ or
591                    $SensorBit{$bitNbr}{'Desc'} =~ m/Unused/ );
592             $cur = &GetSensorBit($bitNbr, $SensorBit, \%localSensorState);
593             $prev = &GetSensorBit($bitNbr, $SensorBit, $SensorState);
594
595             if (($cur - $prev) == 1) { # Bit now set.
596                 &DisplayMessage("TestSensorTones, Sensor bit $bitNbr" .
597                                " has set (1). [" .
598                                $SensorBit{$bitNbr}{'Desc'} . "]);
599                 for ($y = 0; $y < $x; $y++) {
600                     &PlaySound("Lock.wav");

```

```

601         sleep 0.5;
602     }
603     last; # Skip remaining bits
604 }
605 elseif (($prev - $cur) == 1) { # Bit now reset.
606     &DisplayMessage("TestSensorTones, Sensor bit $bitNmbr" .
607         " has reset (0). [" .
608             $$SensorBit{$bitNmbr}{'Desc'} . "]");
609     &PlaySound("Unlock.wav");
610     last; # Skip remaining bits
611 }
612 }
613
614 # Update %SensorState hash with just read sensor states.
615 $$SensorState{'1'} = $localSensorState{'1'};
616 $$SensorState{'2'} = $localSensorState{'2'};
617 sleep 0.5;
618 }
619 return 0;
620 }
621
622 # =====
623 # FUNCTION: TestKeypad
624 #
625 # DESCRIPTION:
626 # This routine displays the pressed button on the keypad. It also sets and
627 # resets the 1st entry LED with each key press. The individual turnout
628 # buttons will also be displayed when pressed. This test loops until the
629 # user enters ctrl+c.
630 #
631 # CALLING SYNTAX:
632 # $result = &TestKeypad($Keypad, \%KeypadData, \%ButtonData, \%GpioData,
633 #     \%MCP23017, \%SensorChip, \%ButtonChild);
634 #
635 # ARGUMENTS:
636 # $KeypadId KeypadData entry to test.
637 # $KeypadData Pointer to %KeypadData hash.
638 # $ButtonData Pointer to %ButtonData hash.
639 # $GpioData Pointer to %GpioData hash.
640 # $MCP23017 Pointer to MCP23017 internal register definitions
641 # $SensorChip Pointer to %SensorChip hash.
642 # $KeypadChildPid $KeypadChildPid value.
643 # $ButtonChildPid $ButtonChildPid value.
644 #
645 # RETURNED VALUES:
646 # 0 = Success, 1 = Error.
647 #
648 # ACCESSED GLOBAL VARIABLES:
649 # $main::MainRun
650 # =====
651 sub TestKeypad {
652     my($KeypadId, $KeypadData, $ButtonData, $GpioData, $MCP23017, $SensorChip,
653         $KeypadChildPid, $ButtonChildPid) = @_;
654     my($button, $value, $lockLed, $key, @input);
655
656     &DisplayDebug(2, "TestKeypad, Entry ... KeypadId: $KeypadId");
657
658     $KeypadId = sprintf("%0.2d", $KeypadId); # Add leading 0 for proper key.
659     if (exists $$KeypadData{$KeypadId}) {
660         while($main::MainRun) {

```

```

661
662 # Keypad buttons.
663 $button = Forks::Super::read_stderr($KeypadChildPid);
664 if ($button eq '') {
665     &DisplayMessage("TestKeypad, KeypadId: $KeypadId - No " .
666                     "button pressed.");
667 }
668 else {
669     $button = substr($button, 0, 1); # 1st character only if multiple.
670     &DisplayMessage("TestKeypad, KeypadId: $KeypadId - " .
671                     "keypad button pressed: '$button'");
672
673     # Read keypad 1st entry LED.
674     $value = $$GpioData{ $$KeypadData{$KeypadId}{ 'Gpio' } }{ 'Obj' }->read;
675
676     # Compliment the value.
677     $value = (~$value) & 1;
678
679     # Write keypad 1st entry LED.
680     $$GpioData{ $$KeypadData{$KeypadId}{ 'Gpio' } }{ 'Obj' }->write($value);
681     &DisplayMessage("TestKeypad, KeypadId: $KeypadId - 1st " .
682                     "entry LED set to $value");
683 }
684
685 # Single buttons.
686 $button = Forks::Super::read_stderr($ButtonChildPid);
687 if ($button ne '') {
688     if ($button =~ m/d(\d+)/) {
689         $value = $1;
690         &DisplayMessage("TestKeypad, button double press: " .
691                         "$$ButtonData{$1}{ 'Desc' }");
692     }
693     elsif ($button =~ m/s(\d+)/) {
694         $value = $1;
695         &DisplayMessage("TestKeypad, button single press: " .
696                         "$$ButtonData{$1}{ 'Desc' }");
697     }
698     else {
699         &DisplayMessage("TestKeypad, invalid button response: " .
700                         "'$button'");
701     }
702     if ($value ge '04' and $value le '07') {
703
704         # Read Holdover route lock LED.
705         $lockLed = $$GpioData{ 'GPIO26_HLCK' }{ 'Obj' }->read;
706
707         # Compliment the value.
708         $lockLed = (~$lockLed) & 1;
709
710         # Write keypad 1st entry LED.
711         $$GpioData{ 'GPIO26_HLCK' }{ 'Obj' }->write($lockLed);
712         &DisplayMessage("TestKeypad, Lock led set to $lockLed");
713     }
714 }
715 else {
716     &DisplayMessage("TestKeypad, No single button input.");
717 }
718
719 # Read and display shutdown button state.
720 $button = $$GpioData{ 'GPIO21_SHDN' }{ 'Obj' }->read;

```

```
721         &DisplayMessage("TestKeypad, Shutdown button: $button");
722         sleep 2;
723     }
724 }
725 else {
726     &DisplayError("TestKeypad, Keypad $KeypadId is not supported.");
727     return 1;
728 }
729 return 0;
730 }
731
732 return 1;
733
```