```perl
# ============================================================================
# FILE: DnB_GradeCrossing.pm                                      9/12/2020
#
# SERVICES:  DnB GRADE CROSSING FUNCTIONS
#
# DESCRIPTION:
#    This perl module provides grade crossing related functions used by the
#    DnB model railroad control program.
#
# PERL VERSION: 5.24.1
#
# ============================================================================
use strict;
# ----------------------------------------------------------------------------
# Package Declaration
# ----------------------------------------------------------------------------
package DnB_GradeCrossing;
require Exporter;
our @ISA = qw(Exporter);

our @EXPORT = qw(
   ProcessGradeCrossing
   GcChildProcess
   TestGradeCrossing
);

use DnB_Sensor;
use DnB_Signal;
use DnB_Turnout;
use DnB_Message;
use Forks::Super;
use Time::HiRes qw(sleep);

# ============================================================================
# FUNCTION:  ProcessGradeCrossing
#
# DESCRIPTION:
#    This routine is used to process the specified grade crossing. It is called
#    once an iteration by the main program loop. State data that is used for
#    grade crossing control is persisted in the %GradeCrossingData hash. Each
#    grade crossing is in one of the following states; 'idle', 'gateLower',
#    'approach', 'road', 'gateRaise' or 'depart'. %GradeCrossingData values,
#    sensor bits, and code within this routine, transition the signal through
#    these states. Operation is as follows.
#
#    1. Configuration and initializations set in %GradeCrossingData hash.
#
#    2. In 'idle' state, a train approaching the grade crossing is detected by
#    sensors 'AprEast', 'AprWest', or 'Road'. This causes the signals to begin
#    flashing. 'SigRun' is set to 'on'. 'GateDelay' is set and the state
#    transitions to 'gateLower'.
#
#    3. In 'gateLower' state, GateDelay is performed and the 'AprTimer' is set.
#    If gates are available, they are lowered. Then the state transitions to
#    'approach'. The GateDelay value is used to better simulate proto-typical
#    signal operation.
#
#    4. In 'approach' state, if 'road' state is not achieved before 'AprTimer'
#    expires, the code transitions to the 'gateRaise' state. This could occur if
#    the train stops or backs away before reaching the 'Road' sensor. An active
```

```
#    'Road' sensor causes transition to the 'road' state.
#
#    5. In 'road' state, a short timeout is set into 'RoadTimer'. Additional
#    'Road' sensor activity reloads this timer. This maintains 'road' state
#    while the train occupies the grade crossing. When no further 'Road' sensor
#    activity is reported, 'RoadTimer' will expire. The state transitions to
#    'gateRaise'.
#
#    6. In 'gateRaise' state, if grade crossing does not have gates, 'DepTimer'
#    is set and the state transitions to 'depart'. Otherwise, the gates are
#    raised. Once completed (servo pid == 0), 'DepTimer' is set and the state
#    transitions to 'depart'.
#
#    7. In the 'depart' state, the signal lamp flashing is stopped and 'SigRun'
#    is set to 'off'. Outbound train 'AprEast' or 'AprWest' sensor activity
#    restarts the 'DepTimer' maintaining the 'depart' state. Once the last car
#    of the outbound train is past the 'AprEast' or 'AprWest' sensor, the
#    'DepTimer' expires and the state transitions to 'idle'.
#
#    If the train backs up, 'Road' sensor activity will transition the state to
#    'idle'. From 'idle', the active 'Road' sensor will start a new signaling
#    cycle.
#
# CALLING SYNTAX:
#    $result = &ProcessGradeCrossing($gc, \%GradeCrossingData, \%SensorBit,
#              \%TurnoutData, \%MCP23017, \%SensorState, $WebDataDir);
#
# ARGUMENTS:
#    $Gc                 Index to data in %GradeCrossingData.
#    $GradeCrossingData  Pointer to %GradeCrossingData hash.
#    $SensorBit          Pointer to %SensorBit hash.
#    $TurnoutData        Pointer to %TurnoutData hash. (needed for gates and sound)
#    $MCP23017           Pointer to %MCP23017 hash. (GPIO definitions)
#    $SensorState        Pointer to %SensorState hash.
#    $WebDataDir         Directory for dynamic web data content.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error
#
# ACCESSED GLOBAL VARIABLES:
#    $main::$Opt{w}
# ============================================================================
sub ProcessGradeCrossing {
   my($Gc, $GradeCrossingData, $SensorBit, $TurnoutData, $MCP23017, $SensorState,
      $WebDataDir) = @_;
   my(@gates);

# Isolate the current grade crossing sensor bit values and get the current time.
   my($aprEastSensor) = &GetSensorBit($$GradeCrossingData{$Gc}{'AprEast'},
                                      $SensorBit, $SensorState);
   my($roadSensor) = &GetSensorBit($$GradeCrossingData{$Gc}{'Road'}, $SensorBit,
                                   $SensorState);
   my($aprWestSensor) = &GetSensorBit($$GradeCrossingData{$Gc}{'AprWest'},
                                      $SensorBit, $SensorState);
   my($cTime) = time;

   &DisplayDebug(2, "ProcessGradeCrossing $Gc, State: " .
             "$$GradeCrossingData{$Gc}{'State'}   aprEastSensor: $aprEastSensor" .
             "   roadSensor: $roadSensor   aprWestSensor: $aprWestSensor   " .
             "cTime: $cTime");
```

```
121
122  # Idle state code. ------------------------------------------------------------
123     if ($$GradeCrossingData{$Gc}{'State'} eq 'idle') {
124        if ($roadSensor == 1 or $aprEastSensor == 1 or $aprWestSensor == 1) {
125           if ($$GradeCrossingData{$Gc}{'SigRun'} ne 'on') {
126              &DisplayMessage("ProcessGradeCrossing $Gc, '" .
127                        $$GradeCrossingData{$Gc}{'State'} .
128                        "' start signals");
129
130              # Start lamps and approach sound effect.
131              Forks::Super::write_stdin($$GradeCrossingData{$Gc}{'Pid'},
132                 'start:apr');
133           }
134
135           $$GradeCrossingData{$Gc}{'SigRun'} = 'on';
136           $$GradeCrossingData{$Gc}{'GateDelay'} = $cTime + .5;
137           $$GradeCrossingData{$Gc}{'State'} = 'gateLower';
138           &DisplayMessage("ProcessGradeCrossing $Gc, 'idle' --> " .
139                           "'$$GradeCrossingData{$Gc}{'State'}'.");
140        }
141     }
142
143  # GateLower state code. --------------------------------------------------------
144     if ($$GradeCrossingData{$Gc}{'State'} eq 'gateLower') {
145
146        # Wait GateDelay. If gates are available, lower them. Then transition
147        # to approach state.
148        if ($$GradeCrossingData{$Gc}{'GateDelay'} < $cTime) { # Delay time done?
149           if ($$GradeCrossingData{$Gc}{'Gate'} ne '') {
150              @gates = split(",", $$GradeCrossingData{$Gc}{'Gate'});
151              foreach my $gate (@gates) {
152                 &DisplayMessage("ProcessGradeCrossing $Gc, '" .
153                           $$GradeCrossingData{$Gc}{'State'} . " state' close " .
154                           "gate: $gate");
155                 &MoveTurnout('Close', $gate, $TurnoutData);
156              }
157           }
158           $$GradeCrossingData{$Gc}{'AprTimer'} = $cTime + 10;
159           $$GradeCrossingData{$Gc}{'State'} = 'approach';
160           &DisplayMessage("ProcessGradeCrossing $Gc, 'gateLower' --> " .
161                           "'$$GradeCrossingData{$Gc}{'State'}'.");
162        }
163     }
164
165  # Approach state code. ---------------------------------------------------------
166     if ($$GradeCrossingData{$Gc}{'State'} eq 'approach') {
167        if ($roadSensor == 1) {
168           $$GradeCrossingData{$Gc}{'RoadTimer'} = $cTime + 1;    # Set RoadTimer
169           $$GradeCrossingData{$Gc}{'State'} = 'road';
170           &DisplayMessage("ProcessGradeCrossing $Gc, 'approach' --> " .
171                           "'$$GradeCrossingData{$Gc}{'State'}'.");
172
173           # Change to roadside sound effect. Commented out, need better sound
174           # module.
175  #        Forks::Super::write_stdin($$GradeCrossingData{$Gc}{'Pid'}, 'start:road');
176        }
177        elsif ($$GradeCrossingData{$Gc}{'AprTimer'} < $cTime) { # AprTimer timeout?
178           $$GradeCrossingData{$Gc}{'State'} = 'gateRaise';
179           &DisplayMessage("ProcessGradeCrossing $Gc, 'approach' " .
180                           "==> '$$GradeCrossingData{$Gc}{'State'}'.");
```

```
181         }
182      }
183
184   # Road state code. ------------------------------------------------------
185      if ($$GradeCrossingData{$Gc}{'State'} eq 'road') {
186         if ($roadSensor == 1) {
187            $$GradeCrossingData{$Gc}{'RoadTimer'} = $cTime + 1;  # Update RoadTimer
188         }
189         else {
190            if ($$GradeCrossingData{$Gc}{'RoadTimer'} < $cTime) { # timeout?
191               $$GradeCrossingData{$Gc}{'State'} = 'gateRaise';
192               &DisplayMessage("ProcessGradeCrossing $Gc, 'road' --> " .
193                               "'$$GradeCrossingData{$Gc}{'State'}'.");
194
195               # Set back to approach sound effect. Commented out, road sound not
196               # currently used.
197               # Forks::Super::write_stdin($$GradeCrossingData{$Gc}{'Pid'},
198               # 'start:apr');
199            }
200         }
201      }
202
203   # GateRaise state code. -------------------------------------------------
204      if ($$GradeCrossingData{$Gc}{'State'} eq 'gateRaise') {
205
206         # If no gates, transition to depart state.
207         if ($$GradeCrossingData{$Gc}{'Gate'} eq '') {
208            $$GradeCrossingData{$Gc}{'DepTimer'} = $cTime + 1;    # Set DepTimer
209            $$GradeCrossingData{$Gc}{'State'} = 'depart';
210            &DisplayMessage("ProcessGradeCrossing $Gc, 'gateRaise' --> " .
211                            "'$$GradeCrossingData{$Gc}{'State'}'.");
212         }
213         else {
214            if ($$GradeCrossingData{$Gc}{'GateServo'} == 0) {
215               @gates = split(",", $$GradeCrossingData{$Gc}{'Gate'});
216               foreach my $gate (@gates) {
217                  &DisplayMessage("ProcessGradeCrossing $Gc, '" .
218                                  $$GradeCrossingData{$Gc}{'State'} .
219                                  " state' open gate: $gate");
220                  &MoveTurnout('Open', $gate, $TurnoutData);
221               }
222               $$GradeCrossingData{$Gc}{'GateServo'} = $gates[0];
223               &DisplayMessage("ProcessGradeCrossing $Gc, '" .
224                               $$GradeCrossingData{$Gc}{'State'} .
225                               " state' waiting for gate " .
226                               $$GradeCrossingData{$Gc}{'GateServo'} . " to open.");
227            }
228            elsif ($$TurnoutData{$$GradeCrossingData{$Gc}{'GateServo'}}{Pid} == 0) {
229               $$GradeCrossingData{$Gc}{'GateServo'} = 0;
230               $$GradeCrossingData{$Gc}{'DepTimer'} = $cTime + 1;  # Set DepTimer
231               $$GradeCrossingData{$Gc}{'State'} = 'depart';
232               &DisplayMessage("ProcessGradeCrossing $Gc, 'gateRaise' " .
233                               "--> '$$GradeCrossingData{$Gc}{'State'}'.");
234            }
235         }
236      }
237
238   # Depart state code. ----------------------------------------------------
239      if ($$GradeCrossingData{$Gc}{'State'} eq 'depart') {
240         if ($$GradeCrossingData{$Gc}{'SigRun'} ne 'off') {
```

```
241            &DisplayMessage("ProcessGradeCrossing $Gc, '" .
242                            $$GradeCrossingData{$Gc}{'State'} . "' stop signals");
243            Forks::Super::write_stdin($$GradeCrossingData{$Gc}{'Pid'}, 'stop');
244            $$GradeCrossingData{$Gc}{'SigRun'} = 'off';
245         }
246
247         # If roadSensor sets, the train backed up. Transition to idle state to
248         # start a new grade crossing cycle.
249         if ($roadSensor == 1) {
250            $$GradeCrossingData{$Gc}{'State'} = 'idle';
251            &DisplayMessage("ProcessGradeCrossing $Gc, 'depart' ==> " .
252                            "'$$GradeCrossingData{$Gc}{'State'}'.");
253         }
254
255         # Stay in depart state until approach sensors are inactive. This prevents
256         # the start of a new grade crossing cycle by departing train. We also
257         # get here if an approach sensor is blocked by a stopped train.
258         elsif ($aprEastSensor == 1 or $aprWestSensor == 1) {
259            $$GradeCrossingData{$Gc}{'DepTimer'} = $cTime + 1;   # Set DepTimer
260         }
261
262         # Transition to idle state after DepTimer expires.
263         elsif ($$GradeCrossingData{$Gc}{'DepTimer'} < $cTime) {
264            $$GradeCrossingData{$Gc}{'State'} = 'idle';
265            &DisplayMessage("ProcessGradeCrossing $Gc, 'depart' --> " .
266                            "'$$GradeCrossingData{$Gc}{'State'}'.");
267         }
268      }
269
270   # Update webserver data. ----------------------------------------------------
271   #    GC01: <state>:<lamps>:<gates>:<aprW>:<road>:<aprE>
272
273      if (defined($main::Opt{w})) {
274         if ($$GradeCrossingData{'00'}{'WebUpdate'} <= 0) {
275            my($state) = $$GradeCrossingData{$Gc}{'State'};
276            my($lamps) = $$GradeCrossingData{$Gc}{'SigRun'};
277            my($gatePos) = 'none';
278
279            if ($$GradeCrossingData{$Gc}{'Gate'} ne '') {
280               if ($state eq 'idle' or $state eq 'gateRaise' or $state eq 'depart') {
281                  $gatePos = 'Open';
282               }
283               else {
284                  $gatePos = 'Closed';
285               }
286            }
287            my($data) = join(': ', "GC$Gc", join(':', $state, $lamps, $gatePos,
288                                    $aprWestSensor, $roadSensor, $aprEastSensor));
289            my(@array);
290            my($gcFile) = join('/', $WebDataDir, "GC$Gc-overlay.dat");
291            if ($state =~ m/idle/i) {
292               @array = ('GC-Off.png');
293            }
294            else {
295               @array = ('GC-On.gif');     # Flash rXr symbol for this GC.
296            }
297            &WriteFile($gcFile, \@array, '');
298
299            if ($Gc eq '01') {
300               $$GradeCrossingData{'00'}{"GC$Gc"} = $data;  # Save until last GC.
```

```
301             }
302             elsif ($Gc eq '02') {
303                @array = ($$GradeCrossingData{'00'}{"GC01"}, $data);
304                &WriteFile("$WebDataDir/grade.dat", \@array, '');
305                $$GradeCrossingData{'00'}{'WebUpdate'} = 10;
306             }
307          }
308          elsif ($Gc eq '02') {
309             $$GradeCrossingData{'00'}{'WebUpdate'}--;
310          }
311       }
312       return 0;
313    }
314
315    # ============================================================================
316    # FUNCTION:  GcChildProcess
317    #
318    # DESCRIPTION:
319    #    This routine is launched as a child process during main program startup
320    #    and is used to start and stop grade crossing signal lamp flash operation.
321    #    Since Forks::Super does not allow a child to fork to another child, any
322    #    servo driven gate timing and positioning for the signal must be done by
323    #    the caller.
324    #
325    #    A dedicated GcChildProcess is started for each grade crossing. The returned
326    #    child Pid value is stored in the %GradeCrossingData hash. This Pid value
327    #    is used in the Forks::Super::write_stdin message to send commands to the
328    #    proper GcChildProcess instance.
329    #
330    # CALLING SYNTAX:
331    #    $pid = fork { os_priority => 1, sub => \&GcChildProcess,
332    #                  child_fh => "in socket",
333    #                  args => [ $Gc, $SignalChildPid, \%SignalData,
334    #                            \%GradeCrossingData, \%SensorChip, \%MCP23017 ] };
335    #
336    #       $GradeCrossing       The signal to be processed.
337    #       $SignalChildPid      PID of child signal refresh process.
338    #       $SignalData          Pointer to %SignalData hash.
339    #       $GradeCrossingData   Pointer to the %GradeCrossingData hash.
340    #       $SensorChip          Pointer to the %SensorChip hash.
341    #       $MCP23017            Pointer to the %MCP23017 hash.
342    #
343    #    The SuperForks 'child_fh' functionality is used for communication between
344    #    the parent and child processes. The parent sends a start/stop signal message
345    #    to the child's stdin. The message must be formatted as follows.
346    #
347    #       start:apr  - Start flashing lamps with bell sound 1.
348    #       start:road - Start flashing lamps with bell sound 2.
349    #       stop       - Stop lamp flash and bell sound.
350    #       exit       - Terminate GcChildProcess.
351    #
352    # SEND DATA TO CHILD:
353    #    Forks::Super::write_stdin($GcChildPid, 'start:apr'));
354    #    Forks::Super::write_stdin($GcChildPid, 'start:road'));
355    #    Forks::Super::write_stdin($GcChildPid, 'stop'));
356    #    Forks::Super::write_stdin($GcChildPid, 'exit'));
357    #
358    # RETURNED VALUES:
359    #    PID of child process = Success, 0 = Error
360    #
```

```
# ACCESSED GLOBAL VARIABLES:
#    $main::ChildName
# ============================================================================
sub GcChildProcess {
   my($GradeCrossing, $SignalChildPid, $SignalData, $GradeCrossingData,
      $SensorChip, $MCP23017) = @_;
   my($x, @buffer, $lampColor, %sndCtrl, $sndSet, $sndClr, $data);
   my($cmd) = '';  my($lampFlash) = 0;

   $main::ChildName = "GcChildProcess$GradeCrossing";
   &DisplayMessage("GcChildProcess${GradeCrossing} started.");

# Setup grade crossing specific working variables.
   my($signalNmbr) = $$GradeCrossingData{$GradeCrossing}{Signal};
   if ($$GradeCrossingData{$GradeCrossing}{'SoundApr'} =~
         m/^(\d),(GPIO)(.)(\d)$/) {
      $sndCtrl{'apr'}{'chip'} = $1;
      $sndCtrl{'apr'}{'port'} = join("", $2, $3);
      $sndCtrl{'apr'}{'gpio'} = join("", $2, $3, $4);
      $sndCtrl{'apr'}{'olat'} = join("", "OLAT", $3);
      $sndCtrl{'apr'}{'bitSet'} = 1 << $4;
      $sndCtrl{'apr'}{'bitClr'} = ~$sndCtrl{'apr'}{'bitSet'};
   }
   if ($$GradeCrossingData{$GradeCrossing}{'SoundRoad'} =~
         m/^(\d),(GPIO)(.)(\d)$/) {
      $sndCtrl{'road'}{'chip'} = $1;
      $sndCtrl{'road'}{'port'} = join("", $2, $3);
      $sndCtrl{'road'}{'gpio'} = join("", $2, $3, $4);
      $sndCtrl{'road'}{'olat'} = join("", "OLAT", $3);
      $sndCtrl{'road'}{'bitSet'} = 1 << $4;
      $sndCtrl{'road'}{'bitClr'} = ~$sndCtrl{'road'}{'bitSet'};
   }
   &DisplayDebug(1, "GcChildProcess${GradeCrossing}, using " .
                    "signalNmbr: $signalNmbr " .
                    "sndApr: '" . $sndCtrl{'apr'}{'gpio'} . "' " .
                    "sndRoad: '" . $sndCtrl{'road'}{'gpio'} . "'");

# Run the main processing loop.
   while (1) {
      push(@buffer, <STDIN>);
#     if ($#buffer >= 0) {
#        for ($x = 0; $x <= $#buffer; $x++) {
#           print "x: $x - '$buffer[$x]' \n";
#        }
#     }

      # ----------
      # Check for a new complete message and process if found.
      if ($buffer[0] =~ m/(start):(apr)/i or $buffer[0] =~ m/(start):(road)/i or
          $buffer[0] =~ m/(stop)/i or $buffer[0] =~ m/(exit)/i) {
         $cmd = lc $1;
         $sndSet = lc $2;

         if ($sndSet eq 'apr') {
            $sndClr = 'road';
         }
         elsif ($sndSet eq 'road') {
            $sndClr = 'apr';
         }
         else {
```

```
421             $sndClr = '';
422          }
423          splice(@buffer, 0, 1);         # Remove processed record.
424  #        &DisplayDebug(3, "GcChildProcess${GradeCrossing}, cmd: " .
425  #                         "'$cmd'   sndSet: '$sndSet'");
426       }
427
428       # ----------
429       # Process new command, if any.
430       if ($cmd ne "") {
431          if ($cmd eq "start") {
432             if ($lampFlash == 0) {
433                $lampColor = 'Red';
434                $lampFlash = 1;
435             }
436
437             # Clear opposite sound activation control bit
438             if ($sndClr ne '') {
439                &ClearControlBit($sndClr, \%sndCtrl, $SensorChip, $MCP23017);
440             }
441
442             # Set new sound activation control bit.
443             if ($sndSet ne '' and exists($sndCtrl{$sndSet}{'chip'})) {
444                $data = $$SensorChip{ $sndCtrl{$sndSet}{'chip'} }{'Obj'}
445                        ->read_byte($$MCP23017{ $sndCtrl{$sndSet}{'port'} });
446                $data = $data | $sndCtrl{$sndSet}{'bitSet'};
447                $$SensorChip{ $sndCtrl{$sndSet}{'chip'} }{'Obj'}
448                ->write_byte($data, $$MCP23017{ $sndCtrl{$sndSet}{'olat'} });
449             }
450          }
451          elsif ($cmd eq "stop" and $lampFlash == 1) {
452             $lampColor = 'Off';
453          }
454          elsif ($cmd eq "exit") {
455             &DisplayMessage("GcChildProcess${GradeCrossing} " .
456                                      "commanded to exit.");
457
458             # Turn off signal lamps.
459             &SetSignalColor($signalNmbr, 'Off', $SignalChildPid,
460                             $SignalData, '');
461
462             # Clear sound activation control bits
463             &ClearControlBit('apr', \%sndCtrl, $SensorChip, $MCP23017);
464             &ClearControlBit('road', \%sndCtrl, $SensorChip, $MCP23017);
465             last;       # Break out of while loop and exit.
466          }
467          $cmd = "";                     # Remove processed command.
468       }
469
470       # ----------
471       # Change lamp state.
472       if ($lampFlash == 1) {
473          if ($lampColor eq 'Off') {
474             $lampFlash = 0;
475
476             # Clear sound activation control bits
477             &ClearControlBit('apr', \%sndCtrl, $SensorChip, $MCP23017);
478             &ClearControlBit('road', \%sndCtrl, $SensorChip, $MCP23017);
479          }
480          elsif ($lampColor eq 'Red') {
```

```
481             $lampColor = 'Grn';
482          }
483          else {
484             $lampColor = 'Red';
485          }
486
487          if (&SetSignalColor($signalNmbr, $lampColor, $SignalChildPid,
488                              $SignalData, '')) {
489             &DisplayError("GcChildProcess${GradeCrossing}, " .
490                           "SetSignalColor returned error.");
491          }
492       }
493       sleep 0.8;            # Sets signal flash rate.
494    }
495    &DisplayMessage("GcChildProcess${GradeCrossing} terminated.");
496    exit(0);
497 }
498
499 # ============================================================================
500 # FUNCTION:  ClearControlBit
501 #
502 # DESCRIPTION:
503 #    This routine is used by GcChildProcess for clearing the specified sound
504 #    activation control bit.
505 #
506 # CALLING SYNTAX:
507 #    $result = &ClearControlBit($Snd, $sndCtrlHash, $SensorChip);
508 #
509 # ARGUMENTS:
510 #    $Snd               Hash index, 'apr' or 'road'.
511 #    $sndCtrlHash       Pointer to GcChildProcess sndCtrl hash.
512 #    $SensorChip        Pointer to the %SensorChip hash.
513 #    $MCP23017          Pointer to $MCP23017 hash.
514 #
515 # RETURNED VALUES:
516 #    0 = Success,  1 = Error.
517 #
518 # ACCESSED GLOBAL VARIABLES:
519 #    None.
520 # ============================================================================
521 sub ClearControlBit {
522    my($Snd, $sndCtrlHash, $SensorChip, $MCP23017) = @_;
523    my($data);
524
525    if (exists($$sndCtrlHash{$Snd}{'chip'})) {
526       $data = $$SensorChip{ $$sndCtrlHash{$Snd}{'chip'} }{'Obj'}
527               ->read_byte($$MCP23017{ $$sndCtrlHash{$Snd}{'port'} });
528       $data = $data & $$sndCtrlHash{$Snd}{'bitClr'};
529       $$SensorChip{ $$sndCtrlHash{$Snd}{'chip'} }{'Obj'}
530          ->write_byte($data, $$MCP23017{ $$sndCtrlHash{$Snd}{'olat'} });
531    }
532    return 0;
533 }
534
535 # ============================================================================
536 # FUNCTION:  TestGradeCrossing
537 #
538 # DESCRIPTION:
539 #    This routine cycles the specified grade crossing signal ranges.
540 #
```

```
541  # CALLING SYNTAX:
542  #    $result = &TestGradeCrossing($Range, \%GradeCrossingData, \%TurnoutData);
543  #
544  # ARGUMENTS:
545  #    $Range               Signal number or range to use.
546  #    $GradeCrossingData   Pointer to GradeCrossingData hash.
547  #    $TurnoutData         Pointer to %TurnoutData hash.
548  #
549  # RETURNED VALUES:
550  #    0 = Success,  1 = Error.
551  #
552  # ACCESSED GLOBAL VARIABLES:
553  #    $main::MainRun
554  # ============================================================================
555  sub TestGradeCrossing {
556
557     my($Range, $GradeCrossingData, $TurnoutData) = @_;
558     my($result, @gates, $gate);
559     my(@gcList) = split(",", $Range);
560
561     &DisplayDebug(2, "TestGradeCrossing, Entry ...   Range: '$Range'" .
562                      "   gcList: @gcList");
563
564     while ($main::MainRun) {
565
566     # Start approach signal.
567        foreach my $gc (@gcList) {
568           $gc = "0${gc}" if (length($gc) == 1);
569           if (exists $$GradeCrossingData{$gc}) {
570              &DisplayMessage("TestGradeCrossing, start:apr grade " .
571                              "crossing $gc");
572              Forks::Super::write_stdin($$GradeCrossingData{$gc}{'Pid'},
573                                        'start:apr');
574              sleep 1;              # Time for realistic lamp start.
575           }
576           else {
577              &DisplayError("TestGradeCrossing, invalid grade " .
578                            "crossing: $gc");
579              return 1;
580           }
581
582     # Lower gates if grade crossing is so equipt.
583           @gates = split(",", $$GradeCrossingData{$gc}{'Gate'});
584           foreach my $gate (@gates) {
585              &DisplayDebug(1, "TestGradeCrossing, Close gate: $gate");
586              $result = &MoveTurnout('Close', $gate, $TurnoutData);
587              if ($result == 1) {
588                 &DisplayDebug(1, "TestGradeCrossing, gate: $gate " .
589                                  "returned error.");
590              }
591              elsif ($result == 2) {
592                 &DisplayDebug(1, "TestGradeCrossing, gate: $gate " .
593                                  "returned already in position.");
594              }
595           }
596           Forks::Super::pause 2;
597        }
598        Forks::Super::pause 4;
599
600  # Change to 'road' grade crossing sound. Commented out, need better sound module.
```

```
601         foreach my $gc (@gcList) {
602            $gc = "0${gc}" if (length($gc) == 1);
603   #        &DisplayMessage("TestGradeCrossing, start:road grade crossing $gc");
604   #        Forks::Super::write_stdin($$GradeCrossingData{$gc}{'Pid'}, 'start:road');
605         }
606          Forks::Super::pause 4;
607
608   # Stop signal.
609         foreach my $gc (@gcList) {
610            $gc = "0${gc}" if (length($gc) == 1);
611            &DisplayMessage("TestGradeCrossing, stop grade crossing $gc");
612            @gates = split(",", $$GradeCrossingData{$gc}{'Gate'});
613            foreach my $gate (@gates) {
614               $result = &MoveTurnout('Open', $gate, $TurnoutData);
615               if ($result == 1) {
616                  &DisplayDebug(1, "TestGradeCrossing, gate: $gate " .
617                                   "returned error.");
618               }
619               elsif ($result == 2) {
620                  &DisplayDebug(1, "TestGradeCrossing, gate: $gate " .
621                                   "returned already in position.");
622               }
623            }
624
625            # If gates for this crossing, wait for gate open to complete before
626            # stopping lamp flash.
627            if ($#gates >= 0) {
628               &DisplayDebug(1, "TestGradeCrossing, waiting for gate " .
629                                "$gates[0] move to complete.");
630               while ($$TurnoutData{$gates[0]}{Pid} > 0) {
631                  sleep 0.5;
632               }
633            }
634            Forks::Super::write_stdin($$GradeCrossingData{$gc}{'Pid'}, 'stop');
635            Forks::Super::pause 2;
636         }
637         Forks::Super::pause 4;
638      }
639      return 0;
640   }
641
642   return 1;
643
```