```perl
# ============================================================================
# FILE: DnB_Signal.pm                                               8/03/2020
#
# SERVICES:  DnB SIGNAL FUNCTIONS
#
# DESCRIPTION:
#    This perl module provides signal related functions used by the DnB model
#    railroad control program.
#
# PERL VERSION: 5.24.1
#
# =============================================================================
use strict;
# -----------------------------------------------------------------------------
# Package Declaration
# -----------------------------------------------------------------------------
package DnB_Signal;
require Exporter;
our @ISA = qw(Exporter);

our @EXPORT = qw(
   Init_SignalDriver
   SetSignalColor
   SetSemaphoreSignal
   SignalChildProcess
   TestSignals
);

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

# =============================================================================
# FUNCTION:  Init_SignalDriver
#
# DESCRIPTION:
#    This routine initializes the GPIO pins associated with the LED driver on
#    the DnB model railroad. A shift register utilizing multiple 74HC595 chips
#    is used. Data is shifted in serially using GPIO pins connected to the data
#    and clock inputs of the shift register.
#
#    A second group of GPIOs is used to control the track power polarity relays.
#
# CALLING SYNTAX:
#    $result = &Init_SignalDriver(\%GpioData, $RegisterLength);
#
# ARGUMENTS:
#    $GpioData          Pointer to GPIO data.
#    $RegisterLength    Shift register bit length.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# =============================================================================
sub Init_SignalDriver {
   my($GpioData, $RegisterLength) = @_;
   my($x, $pin);

   &DisplayDebug(2, "Init_SignalDriver, RegisterLength: $RegisterLength");

# Create a Raspberry Pi object for each GPIO and set to defaults.

   foreach my $gpio (sort keys %$GpioData) {
      if ($$GpioData{$gpio}{'Obj'} == 0) {
         if ($gpio =~ m/^GPIO(\d*)_/) {
            $pin = $1;
            $$GpioData{$gpio}{'Obj'} = RPi::Pin->new($pin);
            if ($$GpioData{$gpio}{'Obj'} != 0) {
               &DisplayDebug(1, "Init_SignalDriver, $gpio object " .
                                "successfully created.");
               $$GpioData{$gpio}{'Obj'}->mode($$GpioData{$gpio}{'Mode'});
               if ($$GpioData{$gpio}{'Mode'} == 0) {
                  # 0=None, 1=Pulldown, 2=Pullup
                  $$GpioData{$gpio}{'Obj'}->pull(2);   # Enable pullup on pin.
               }
               elsif ($$GpioData{$gpio}{'Mode'} == 1) {
                  $$GpioData{$gpio}{'Obj'}->write(0);          # Set GPIO low.
               }
            }
            else {
               &DisplayError("Init_SignalDriver, failed to create " .
                             "$gpio object. $!");
               return 1;
            }
         }
         else {
            &DisplayError("Init_SignalDriver, failed to parse pin " .
                          "number from '$gpio'.");
            return 1;
         }
      }
      else {
         &DisplayWarning("Init_SignalDriver, $gpio object already active.");
      }
  }

# Test toggle.
#   while (1) {
#      foreach my $gpio (sort keys %$GpioData) {
#         $$GpioData{$gpio}{'Obj'}->write(1);
#      }
#      &DisplayDebug(1, "Init_SignalDriver, All GPIOs HIGH.");
#      sleep 2;
#      foreach my $gpio (sort keys %$GpioData) {
#         $$GpioData{$gpio}{'Obj'}->write(0);
#      }
#      &DisplayDebug(1, "Init_SignalDriver, All GPIOs LOW.");
#      sleep 2;
#   }
#   exit(0);

# Set all signals to 'Off'. GPIO27_SCLK, GPIO22_DATA, and GPIO17_XLAT are
# set to 0 from above GPIO instantiation.

   $$GpioData{'GPIO23_OUTE'}{'Obj'}->write(1);       # Blank outputs.
   for ($x = 0; $x < $RegisterLength; $x++) {
      $$GpioData{'GPIO27_SCLK'}{'Obj'}->write(1);    # Set SCLK high (store bit).
      $$GpioData{'GPIO27_SCLK'}{'Obj'}->write(0);   # Set SCLK low.
   }
   $$GpioData{'GPIO17_XLAT'}{'Obj'}->write(1);      # Set XLAT high (latch data).
   $$GpioData{'GPIO17_XLAT'}{'Obj'}->write(0);      # Set XLAT low.
   $$GpioData{'GPIO23_OUTE'}{'Obj'}->write(0);      # Enable outputs.

   &DisplayMessage("Init_SignalDriver, All signals and relays set to 'Off'.");
   return 0;
}

# =============================================================================
# FUNCTION:  SetSignalColor
#
# DESCRIPTION:
#    This routine sets the specified signal to the specified color. Each signal
#    LED is a two lead red/green device wired to the two consecutive register
#    bits. Red is illuminated with one current flow direction and green is
#    illuminated with the opposite current flow direction. Current direction is
#    controlled by which of the two register bits is set high/low. The local
#    signalColor hash holds the values for each color.
#
#    This routine is called by SetSemaphoreSignal to control lamp on/off. The
#    SemaphoreFlag argument is used to prevent this routine from setting the new
#    color value into $$SignalData{$Signal}{'Current'}. The SetSemaphoreSignal
#    routine will set the value once the associated servo move has completed.
#
#    The necessary mask values are created and sent to SignalChildProcess stdin
#    to set the specified signal (01-16) to the specified color.
#
# CALLING SYNTAX:
#    $result = &SetSignalColor($Signal, $Color, $SignalChildPid,
#                              \%SignalData, $SemaphoreFlag);
#
# ARGUMENTS:
#    $Signal            Signal number to set.
#    $Color             Signal color, 'Red', 'Grn', 'Yel', or 'Off'
#    $SignalChildPid    PID of child signal refresh process.
#    $SignalData        Pointer to SignalData hash.
#    $SemaphoreFlag     Suppresses setting of current color when set.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# =============================================================================
sub SetSignalColor {
   my($Signal, $Color, $SignalChildPid, $SignalData, $SemaphoreFlag) = @_;
   my($data1, $data2, $mask);

   my(%signalColor1) = ('Off' => 0b00, 'Red' => 0b01, 'Grn' => 0b10,
                        'Yel' => 0b01);
   my(%signalColor2) = ('Off' => 0b00, 'Red' => 0b01, 'Grn' => 0b10,
                        'Yel' => 0b10);

   &DisplayDebug(2, "SetSignalColor, Signal: $Signal   Color: " .
                    "$Color   SemaphoreFlag: '$SemaphoreFlag'");

   if ($Signal ne "") {

```

```perl
      # Create mask values for the specified signal.
      if ($Color eq 'Red' or $Color eq 'Grn' or $Color eq 'Off' or
          $Color eq 'Yel') {
         $mask = 0xFFFFFFFF & (~(0b11 << (($Signal - 1) * 2)));
         $data1 = $signalColor1{$Color} << (($Signal - 1) * 2);
         $data2 = $signalColor2{$Color} << (($Signal - 1) * 2);

         &DisplayDebug(2, "SetSignalColor, -----  16151413121110 9 " .
                          "8 7 6 5 4 3 2 1");
         &DisplayDebug(2, "SetSignalColor, mask:  " .
                          sprintf("%0.32b", $mask));
         &DisplayDebug(2, "SetSignalColor, data1: " .
                          sprintf("%0.32b", $data1));
         &DisplayDebug(2, "SetSignalColor, data2: " .
                          sprintf("%0.32b", $data2));

         Forks::Super::write_stdin($SignalChildPid, join(",", $mask, $data1,
                                                         $data2, "-\n"));
         $$SignalData{$Signal}{'Current'} = $Color unless ($SemaphoreFlag);
      }
      else {
         &DisplayError("SetSignalColor, invalid signal color: $Color");
         return 1;
      }
   }
   else {
      &DisplayError("SetSignalColor, invalid signal number: $Signal");
      return 1;
   }
   return 0;
}

# =============================================================================
# FUNCTION:  SetSemaphoreSignal
#
# DESCRIPTION:
#    This routine sets the specified Semaphore signal to the specified color.
#    SetSignalColor is called to set the lamp on (red) or off as required.
#    MoveTurnout is called to position the servo attached to the semaphore flag
#    board.
#
#   This routine is call for each iteration of the main loop until the 'Position'
#   value for the semaphore in SemaphoreData is set to the necessary color.
#
# CALLING SYNTAX:
#    $result = &SetSemaphoreSignal($Signal, $Color, $SignalChildPid, \%SignalData,
#                                  \%SemaphoreData, \%TurnoutData);
#
# ARGUMENTS:
#    $Signal           Signal number to set.
#    $Color            Signal color, 'Red', 'Grn', 'Yel', or 'Off'
#    $SignalChildPid   PID of child signal refresh process.
#    $SignalData       Pointer to %SignalData hash.
#    $SemaphoreData    Pointer to %SemaphoreData hash.
#    $TurnoutData      Pointer to %TurnoutData hash.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
```

```
#    None.
# =============================================================================
sub SetSemaphoreSignal {
   my($Signal, $Color, $SignalChildPid, $SignalData, $SemaphoreData,
      $TurnoutData) = @_;
   my($moveResult, $servo);
   my(%flagPosition) = ('Grn' => 'Open', 'Yel' => 'Middle', 'Red' => 'Close',
                        'Off' => 'Open');

   &DisplayDebug(1, "SetSemaphoreSignal, Signal: $Signal   Color: $Color");

   if ($Signal ne "" and exists($$SemaphoreData{$Signal})) {
      $servo = $$SemaphoreData{$Signal}{'Servo'};
      if ($$SemaphoreData{$Signal}{'InMotion'} == 1) {
         if ($$TurnoutData{$servo}{Pid} == 0) {
            $$SemaphoreData{$Signal}{'InMotion'} = 0;
            &DisplayDebug(2, "SetSemaphoreSignal, semaphore $Signal " .
                             "move completed.");

            # Turn on lamp unless color is off.
            if ($Color ne 'Off') {
               if (&SetSignalColor($Signal, 'Grn', $SignalChildPid, $SignalData,
                                   'semaphore')) {
                  &DisplayError("SetSemaphoreSignal, SetSignalColor " .
                                "$Signal 'Grn' returned error.");
                  return 1;
               }
               $$SemaphoreData{$Signal}{'Lamp'} = 'On';
            }
            $$SignalData{$Signal}{'Current'} = $Color;
            &DisplayMessage("SetSemaphoreSignal, semaphore $Signal " .
                            "set to $Color.");
         }
      }
      else {
         if ($$SignalData{$Signal}{'Current'} ne $Color) {
            &DisplayDebug(1, "SetSemaphoreSignal, moving semaphore " .
                             "$Signal to position $Color");

            # Turn off lamp.
            if (&SetSignalColor($Signal, 'Off', $SignalChildPid, $SignalData,
                                'semaphore')) {
               &DisplayError("SetSemaphoreSignal, SetSignalColor " .
                             " $Signal 'Off' returned error.");
               return 1;
            }
            $$SemaphoreData{$Signal}{'Lamp'} = 'Off';

            # Move semaphore flag board to requested position.
            $moveResult = &MoveTurnout($flagPosition{$Color}, $servo, $TurnoutData);
            if ($moveResult == 0) {
               $$SemaphoreData{$Signal}{'InMotion'} = 1;
               &DisplayDebug(2, "SetSemaphoreSignal, semaphore " .
                                "$Signal move inprogress.");
            }
            elsif ($moveResult == 1) {
               &DisplayError("SetSemaphoreSignal, MoveTurnout $servo '" .
                             $flagPosition{$Color} . "' returned error.");
               return 1;
            }

            # If MoveTurnout uses return 2, the servo is already in the
            # requested position. Complete the related processing.
            elsif ($moveResult == 2) {

               # Turn on lamp unless color is off.
               if ($Color ne 'Off') {
                  if (&SetSignalColor($Signal, 'Grn', $SignalChildPid,
                                      $SignalData, 'semaphore')) {
                     &DisplayError("SetSemaphoreSignal, SetSignalColor " .
                                   "$Signal 'Grn' returned error.");
                     return 1;
                  }
                  $$SemaphoreData{$Signal}{'Lamp'} = 'On';
               }
               $$SignalData{$Signal}{'Current'} = $Color;
            }
         }
      }
   }
   else {
      &DisplayError("SetSemaphoreSignal, invalid signal number: $Signal");
      return 1;
   }
   return 0;
}

# =============================================================================
# FUNCTION:  SignalChildProcess
#
# DESCRIPTION:
#    This routine is launched as a child process during main program startup
#    and is used to communicate with the 74HC595 shift registers. This frees
#    the main code from the constant need to toggle the yellow signals between
#    red and green. The LEDs used in the signals should all be of similar
#    electrical specifications and color characterists.
#
#    Two time delays (select statements) are used to balance the red/green 'on'
#    time. This provides for coarse level adjustment of the yellow color for all
#    signals. These values should be set with the variable resistors on the shift
#    register board set to mid position. Then, the variable resistors are used
#    for fine adjustment of each signals yellow color.
#
#    The time delays further control the repetition rate of the while loop.
#    This rate should be just high enough to eliminate flicker when the yellow
#    color is displayed; about 25-30 cycles per second. The lowest possible
#    cycle rate is desired to minimize CPU loading by the while loop.
#
#    The while loop further optimizes itself by checking for any yellow signal
#    indications. Yellow signals, with opposite red/green registerBit variable
#    settings, will produce a non-zero result when XOR'd. When no yellow signals
#    are being displayed, the while loop repetition rate is reduced to 4 cycles
#    per second.
#
# CALLING SYNTAX:
#    $pid = fork { sub => \&SignalChildProcess, child_fh => "in socket",
#                  args => [ \%GpioData ] };
#
#       $GpioData          Pointer to the %GpioData hash.
#
```

```
#    The SuperForks 'child_fh' functionality is used for communication between
#    the parent and child processes. The parent sends new signal settings to the
#    child's stdin. The new data is stored in the child variables and used until
#    subsequently updated.
#
#    To minimize input processing within this subroutine, the data message must
#    be formatted as follows.
#
#    <sigMask>,<sigColor1>,<sigColor2>,<terminator>
#
#       <sigMask>    - 32 bit mask, all 1's, signal position two bits set to 0.
#       <sigColor1>  - 32 bit mask, all 1's, signal position set to color value.
#       <sigColor2>  - 32 bit mask, all 1's, signal position set to color value.
#       <terminator> - "-\n".
#
# SEND DATA TO CHILD:
#    Forks::Super::write_stdin($SignalChildPid, join(",", $sigMask, $sigColor1,
#                                                    $sigColor2, "-\n"));
#
# RETURNED VALUES:
#    PID of child process = Success, 0 = Error
#
# ACCESSED GLOBAL VARIABLES:
#    $main::ChildName
# =============================================================================
sub SignalChildProcess {
   my($GpioData) = @_;
   my($x, @buffer, $yellowSig);

# Default shift register bits.
   my($registerBits1) = 0x00000000;
   my($registerBits2) = 0x00000000;

   $main::ChildName = 'SignalChildProcess';
   &DisplayMessage("SignalChildProcess started.");

   while (1) {
      push(@buffer, <STDIN>);

      # Check for a new complete message and process if found.
      if ($buffer[0] =~ m/(.+?),(.+?),(.+?),-/) {
#         for ($x = 0; $x <= $#buffer; $x++) {
#            print "x: $x - $buffer[$x]";
#         }
         $registerBits1 = (($registerBits1 & $1) | $2);
         $registerBits2 = (($registerBits2 & $1) | $3);
         $yellowSig = $registerBits1 ^ $registerBits2;

#         &DisplayDebug(3, "SignalChildProcess, 1: " .
#                          sprintf("%0.32b", $1));
#         &DisplayDebug(3, "SignalChildProcess, 2: " .
#                          sprintf("%0.32b", $2));
#         &DisplayDebug(3, "SignalChildProcess, 3: " .
#                          sprintf("%0.32b", $3));
#         &DisplayDebug(1, "SignalChildProcess, registerBits1: " .
#                          sprintf("%0.32b", $registerBits1));
#         &DisplayDebug(1, "SignalChildProcess, registerBits2: " .
#                          sprintf("%0.32b", $registerBits2));
         splice(@buffer, 0, 1);    # Remove processed record.
      }

      # Send data to 74HC595s - GPIO17_XLAT, GPIO23_OUTE, GPIO27_SCLK, GPIO22_DATA
      for my $pos (reverse(0..31)) {
         $$GpioData{'GPIO27_SCLK'}{'Obj'}->write(0);       # Set SCLK low.
         $$GpioData{'GPIO22_DATA'}{'Obj'}->write(($registerBits1 >> $pos) & 0x01);
         $$GpioData{'GPIO27_SCLK'}{'Obj'}->write(1);       # Set SCLK high
      }
      $$GpioData{'GPIO27_SCLK'}{'Obj'}->write(0);          # Set SCLK low.
      $$GpioData{'GPIO17_XLAT'}{'Obj'}->write(1);          # Set XLAT high
      $$GpioData{'GPIO17_XLAT'}{'Obj'}->write(0);          # Set XLAT low.

      sleep 0.25 unless ($yellowSig);
      sleep 0.006;                         # Adjust for coarse yellow color.

      for my $pos (reverse(0..31)) {
         $$GpioData{'GPIO27_SCLK'}{'Obj'}->write(0);       # Set SCLK low.
         $$GpioData{'GPIO22_DATA'}{'Obj'}->write(($registerBits2 >> $pos) & 0x01);
         $$GpioData{'GPIO27_SCLK'}{'Obj'}->write(1);       # Set SCLK high
      }
      $$GpioData{'GPIO27_SCLK'}{'Obj'}->write(0);          # Set SCLK low.
      $$GpioData{'GPIO17_XLAT'}{'Obj'}->write(1);          # Set XLAT high
      $$GpioData{'GPIO17_XLAT'}{'Obj'}->write(0);          # Set XLAT low.

      sleep 0.25 unless ($yellowSig);
      sleep 0.019;                         # Adjust for coarse yellow color.
   }

   &DisplayMessage("SignalChildProcess terminated.");
   exit(0);
}

# =============================================================================
# FUNCTION:  TestSignals
#
# DESCRIPTION:
#    This routine cycles the specified signal range between the available colors.
#
# CALLING SYNTAX:
#    $result = &TestSignals($Range, $SignalChildPid, \%SignalData,
#                           \%GradeCrossingData, \%SemaphoreData, \%TurnoutData);
#
# ARGUMENTS:
#    $Range               Signal number or range to use.
#    $SignalChildPid      PID of child signal refresh process.
#    $SignalData          Pointer to %SignalData hash.
#    $GradeCrossingData   Pointer to %GradeCrossingData hash.
#    $SemaphoreData       Pointer to %SemaphoreData hash.
#    $TurnoutData         Pointer to %TurnoutData hash. (semaphore flag board)
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
#    $main::MainRun
# =============================================================================
sub TestSignals {

   my($Range, $SignalChildPid, $SignalData, $GradeCrossingData, $SemaphoreData,
      $TurnoutData) = @_;
   my($result, $signal, $start, $end, $nmbr, $color, @signalNumbers);
```

```perl
   my($cntSignal) = scalar keys %$SignalData;
   my(@signalList) = ();  my(@colorList) = ();
   my($random, $gradecrossing) = (0,0);
   my(%colorHash) = (1 => 'Red', 2 => 'Grn', 3 => 'Yel', 4 => 'Off');

   &DisplayDebug(1, "TestSignals, Entry ... SignalChildPid: " .
                    "$SignalChildPid   Range: '$Range'");

   # ==============================
   # Set specified color and exit.

   if ($Range =~ m/^(Red):(\d+)/i or $Range =~ m/^(Grn):(\d+)/i or
       $Range =~ m/^(Yel):(\d+)/i or $Range =~ m/^(Off):(\d+)/i) {
      $color = ucfirst(lc $1);
      $signal = $2;
      if ($signal > $cntSignal or $signal <= 0) {
         &DisplayError("TestSignals, invalid signal number: $signal");
         return 1;
      }
      $signal = "0${signal}" if (length($signal) == 1);
      if (exists ($$SemaphoreData{$signal})) {
         &DisplayDebug(1, "TestSignals, Semaphore signal: $signal");
         while ($$SignalData{$signal}{Current} ne $color) {
            return 1 if (&SetSemaphoreSignal($signal, $color, $SignalChildPid,
                         $SignalData, $SemaphoreData, $TurnoutData));
            sleep 0.5;                        # Wait for servo move.
         }
      }
      else {
         return 1 if (&SetSignalColor($signal, $color, $SignalChildPid,
                                      $SignalData, ''));
      }
      &DisplayMessage("Signal $signal set to '$color'.");
      exit(0);
   }
   elsif ($Range =~ m/^(Red).*/i or $Range =~ m/^(Grn).*/i or
          $Range =~ m/^(Yel).*/i or $Range =~ m/^(Off).*/i) {
      $color = ucfirst(lc $1);
      foreach my $signal (1..12) {
         $signal = "0${signal}" if (length($signal) == 1);
         if (exists ($$SemaphoreData{$signal})) {
            &DisplayDebug(1, "TestSignals, Semaphore signal: $signal");
            while ($$SignalData{$signal}{Current} ne $color) {
               return 1 if (&SetSemaphoreSignal($signal, $color, $SignalChildPid,
                            $SignalData, $SemaphoreData, $TurnoutData));
               sleep 0.5;                     # Wait for servo move.
            }
         }
         else {
            return 1 if (&SetSignalColor($signal, $color, $SignalChildPid,
                                         $SignalData, ''));
         }
         &DisplayDebug(1, "TestSignals, Signal $signal is set to " .
                          "$$SignalData{$signal}{Current}");
      }
      &DisplayMessage("All signals set to '$color'.");
      exit(0);
   }

   # ==============================
   # Process special modifiers and then setup for looped testing.

   if ($Range =~ m/r.*\d/i) {
      $random = 1;
      $Range =~ s/r//i;
   }
   if ($Range =~ m/g.*\d/i) {
      $gradecrossing = 1;
      $Range =~ s/g//i;
      sleep 1;                # Give GcChildProcess time to start.
   }

   if ($Range =~ m/(\d+):(\d+)/) {   # Range specified.
      $start = $1;
      $end = $2;
      if ($start > $end or $start <= 0 or $start > $cntSignal or $end <= 0 or
          $end > $cntSignal) {
         &DisplayError("TestSignals, invalid signal range: '$Range'" .
                       "   cntSignal: $cntSignal");
         return 1;
      }
      for ($signal = $start; $signal <= $end; $signal++) {
         push (@signalList, $signal);
      }
   }
   else {
      @signalList = split(",", $Range);
      foreach my $signal (@signalList) {
         if ($signal !~ /^\d+$/ or $signal > $cntSignal or $signal <= 0) {
            &DisplayError("TestSignals, invalid signal number: $signal");
            return 1;
         }
      }
   }

   &DisplayDebug(1, "TestSignals, signalList: '@signalList'");

   # ==============================
   # Begin looped testing.

   while ($main::MainRun) {
      # For random testing, we randomize the signalNumbers list and also the
      # signal color. For non-random, we set each color.

      if ($random == 1) {
         &ShuffleArray(\@signalList);
         foreach my $signal (@signalList) {
            last unless ($main::MainRun);
            $signal = "0${signal}" if (length($signal) == 1);
            $color = $colorHash{(int(rand(4))+1)};
            if ($gradecrossing == 1) {
               if ($color eq 'Grn') {
                  Forks::Super::write_stdin($$GradeCrossingData{'01'}{'Pid'},
                                            'start:apr');
               }
               elsif ($color eq 'Yel') {
                  Forks::Super::write_stdin($$GradeCrossingData{'02'}{'Pid'},
                                            'start:road');
               }
               elsif ($color eq 'Off') {
                  Forks::Super::write_stdin($$GradeCrossingData{'01'}{'Pid'},
                                            'stop');
               }
               elsif ($color eq 'Red') {
                  Forks::Super::write_stdin($$GradeCrossingData{'02'}{'Pid'},
                                            'stop');
               }
            }
            &DisplayMessage("TestSignals, Signal: $signal   Color: $color");
            if (exists ($$SemaphoreData{$signal})) {
               &DisplayDebug(1, "TestSignals, Semaphore signal: $signal");
               while ($$SignalData{$signal}{Current} ne $color) {
                  return 1 if (&SetSemaphoreSignal($signal, $color,
                               $SignalChildPid, $SignalData, $SemaphoreData,
                               $TurnoutData));
                  sleep 0.5;                     # Wait for servo move.
               }
            }
            else {
               return 1 if (&SetSignalColor($signal, $color, $SignalChildPid,
                                            $SignalData, ''));
            }
            &DisplayDebug(1, "TestSignals, Signal $signal is set to " .
                             "$$SignalData{$signal}{Current}");
            sleep 0.5;
         }
      }
      else {
         # Create colorList test sequence.
         if ($#colorList < 0) {
            foreach my $nmbr (sort keys %colorHash) {
               push (@colorList, $colorHash{$nmbr});
            }
         }
         foreach my $color (@colorList) {
            if ($gradecrossing == 1) {
               if ($color eq 'Grn') {
                  Forks::Super::write_stdin($$GradeCrossingData{'01'}{'Pid'},
                                            'start:apr');
               }
               elsif ($color eq 'Yel') {
                  Forks::Super::write_stdin($$GradeCrossingData{'02'}{'Pid'},
                                            'start:road');
               }
               elsif ($color eq 'Off') {
                  Forks::Super::write_stdin($$GradeCrossingData{'01'}{'Pid'},
                                            'stop');
               }
               elsif ($color eq 'Red') {
                  Forks::Super::write_stdin($$GradeCrossingData{'02'}{'Pid'},
                                            'stop');
               }
            }
            foreach my $signal (@signalList) {
               last unless ($main::MainRun);
               $signal = "0${signal}" if (length($signal) == 1);
               &DisplayMessage("TestSignals, Signal: $signal   Color: $color");
               if (exists ($$SemaphoreData{$signal})) {
                  &DisplayDebug(1, "TestSignals, Semaphore signal: $signal");
                  while ($$SignalData{$signal}{Current} ne $color) {
                     return 1 if (&SetSemaphoreSignal($signal, $color,
                                  $SignalChildPid, $SignalData, $SemaphoreData,
                                  $TurnoutData));
                     sleep 0.5;                # Wait for servo move.
                  }
               }
               else {
                  return 1 if (&SetSignalColor($signal, $color, $SignalChildPid,
                                               $SignalData, ''));
               }
               &DisplayDebug(1, "TestSignals, Signal $signal is set" .
                                " to $$SignalData{$signal}{Current}");
               sleep 0.75;
            }
         }
      }
      sleep 2;    # Show set signal color(s) for this time delay.
   }

   # Set signal related servos to their open position.
   foreach my $nmbr (keys(%$TurnoutData)) {
      if ($$TurnoutData{$nmbr}{'Id'} =~ m/semaphore/i or
          $$TurnoutData{$nmbr}{'Id'} =~ m/gate/i) {
         $result = &MoveTurnout('Open', $nmbr, $TurnoutData);
         while ($$TurnoutData{$nmbr}{'Pid'}) {
            sleep 0.25;
         }
      }
   }
   return 0;
}

return 1;
```