```
--------------------------------------------------------------------------------
-- Company:                     GUSTECH
-- Engineer:                    Thomas W. Gustin
--
-- Create Date:    18:46:00 01/22/07
-- Design Name:    SPI 3-byte Write Module
-- Module Name:    SPI3byteWr_Ver2 - Behavioral
-- Project Name:   any that uses SPI serial communications
-- Target Device:  initially xc3s200-4pq208
-- Tool versions:  Xilinx 7.1i
-- Description:         This is a second version of attempting to build a
-- generic VHDL module that will eventually be
-- instantiated into a single state machine that does
-- all of the serial communications tasks required to
-- write and read data to nonvolatile registers, including
-- high-voltage store status checking.

-- The initial target device is an Intersil X9271 256-tap
-- digital potentiometers with a total of 16 8-bit registers.

-- This particular test module is only generating 3-Byte writes
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
--           Revision 1.00 - Port changes and significant build after the 54 state
--              gray code is running. (16 Jan 07)
-- Revision 2.00 - This version created from "SPI3ByteWr_Ver1.vhd" that used
--  a 54 state gray code engine.  This version uses a 66 state
--  gray code engine that includes a mandatory 2uS minimum
--  negation state for the chip select signal at the end of
--  a valid write sequence (but, prior to the 10mS high voltage
--  write operation).

-- Additional Comments:        developed on GUSTECH's dime
--                     contains custom 66 count gray code engine
--                     for near-silent and lowest power running
--                           based upon "Gray66Cnt.xls".
--
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity SPI3byteWr_Ver2 is
    Port ( SPIwrCLK : in std_logic; -- clock for SPI Writes;
                                                -- high end is 5MHz, no lower frequency limit

           n_RST : in std_logic;     -- active low asynchronous reset

           n_STRT : in std_logic;  -- active low module start run enable
                                                -- asserted low to start, held low until BUSY asserted
                                                -- tested only at gray code count = 0.
                                                -- Once running, will continue full sequence.
                                                -- If a fault is detected on an input value on the
```

```
                                        -- assertion of n_STRT, the engine will NOT run and
                                        -- the fault condition will be posted (see EngFAULT below)

    DeviceID : in std_logic_vector(3 downto 0);
                                        -- SPI type code: 0101b passed in for this X9271 device
                                        -- Note: the device code may be different for other devices
                                        -- Internally registered on n_STRT assertion so that it can be
                                        -- externally changed for the next operation while this
                                        -- "write" is being executed

    BusADD : in std_logic_vector(1 downto 0);
                                        -- SPI bus address passed in to match the hardware
                                        -- address lines for a multi-device configuration
                                        -- Internally registered on n_STRT assertion so that it can be
                                        -- externally changed for the next operation while this
                                        -- "write" is being executed

    InsOpCode : in std_logic_vector(3 downto 0);
                                        -- 3 byte Instruction Operation Codes:
                                        -- These codes are device and function specific,
                                        -- and they are passed in (five of eight codes):
                                        -- 1001 = read wiper counter register (not this VHDL)
                                        -- 1010 = write wiper counter register
                                        --                 this operation does NOT require High Voltage Write
                                        -- 1011 = read data register ** (not this VHDL)
                                        -- 1100 = write data register **
                                        --                 this operation requires subsequent High Voltage Write
                                        -- 0101 = read status (WIP bit)  (not this VHDL)
                                        -- ** register depends on RegADD and BankADD below
                                        -- The other three operations are NOT 3 bytes in length
                                        -- Therefore, only codes 1010 and 1100 are handled by
                                        -- this particular state machine.
                                        -- Internally registered on n_STRT assertion so that it can be
                                        -- externally changed for the next operation while this
                                        -- "write" is being executed

    RegADD : in std_logic_vector(1 downto 0);
                                        -- Data Register Address select (1 of 4 per bank):
                                        -- 00 = register 0
                                        -- 01 = register 1
                                        -- 10 = register 2
                                        -- 11 = register 3
                                        -- Internally registered on n_STRT assertion so that it can be
                                        -- externally changed for the next operation while this
                                        -- "write" is being executed

    BankADD : in std_logic_vector(1 downto 0);
                                        -- Register Bank Address select (1 of 4):
                                        -- 00 = bank 0 (data and wiper counter registration)
                                        -- 01 = bank 1 (data read and write only; spare storage)
                                        -- 10 = bank 2 (data read and write only; spare storage)
                                        -- 11 = bank 3 (data read and write only; spare storage)
                                        -- Internally registered on n_STRT assertion so that it can be
                                        -- externally changed for the next operation while this
                                        -- "write" is being executed

    NVdataOut : in std_logic_vector(7 downto 0);
                                        -- Byte of data passed in to be written to the wiper counter
                                        -- or a data register, depending upon the OpCode; and the
                                        -- data register is dependent upon the RegADD and BankADD.
                                        -- Internally registered on n_STRT assertion so that it can be
                                        -- externally changed for the next operation while this
```

```
                                    -- "write" is being executed

                            SPI3BwrBsy : out std_logic;
                                    -- asserted high while a write operation is being executed
                                    -- goes low after the write to the SPI device has been
                                    -- completed.  However, this module does NOT check to make
                                    -- sure the high voltage write operation is completed
                                    -- which takes another 5-10mS more.  This  needs to be done
                                    -- by another module that reads the status (WIP bit) after
                                    -- this module is no longer busy.

                            EngFAULT : out std_logic;
                                    -- This failure flag will be asserted if there is a problem
                                    -- found with any of the "passed-in" data when this module's
                                    -- n_STRT signal is asserted low (incorrect OpCode, incorrect
                                    -- bank address for a write wiper counter register operation,
                                    -- and incorrect register address for a write wiper counter
                                    -- register operation).

        n_CS : out std_logic;   -- asserted low chip select to SPI device

        SCL : out std_logic;    -- 2.5MHz(max) Serial clock to SPI device

        SI : out std_logic);    -- Serial Input data line to SPI device

end SPI3byteWr_Ver2;  -- end of port listing

-- ***************************************************************************

architecture Behavioral of SPI3byteWr_Ver2 is

-- ***************************************************************************

-- Internal signal declarations

        signal GrayCnt66 : std_logic_vector(6 downto 0);
                -- 7-bit    66 count gray code counter
                -- true, only-one-bit-change-at-a-time sequence of unique states

        signal nextGray : std_logic_vector (6 downto 0);
                -- 7-bit 66 count gray code "NEXT COUNT" lookup

        signal n_ENA : std_logic;
                -- asserted low by the assertion of n_STRT and there is no
                -- fault condition discovered on any of the input data
                -- It is negated (high) by the last count state of the engine

        signal BD0 : std_logic;         -- individual "bad data" instances that
        signal BD1 : std_logic;         -- are tested as a group for the final
        signal BD2 : std_logic;         -- combined signal BadData (below)
        signal BD3 : std_logic;         -- Added these because compiler failed when
                                        -- all were combined into a single statement

        signal BadData : std_logic;
                -- asserted high asynchronously whenever there is any kind of
                -- faulty data on any data combinations for this VHDL module.
                -- There is no effect or action until the assertion of n_STRT.
                -- If BadData is asserted on n_STRT, then EngFAULT will be
                -- asserted and the engine will NOT be enabled for a run.
                -- If BadData is negated on n_STRT assertion, then n_ENA will be
                -- asserted (low) to enable the running of the engine, and
                -- (simultaneously) all of the data inputs will be registered for
```

```
                -- eventual SI output assignments.

        signal RegInputs : std_logic_vector (23 downto 0);
                -- This is the 24 bit input data register that captures the inputs
                -- as the engine is being enabled because there is no "BadData"
                -- on the assertion of the n_STRT (start engine) signal.


-- ********************************************************************

begin


-- ********************************************************************

--      "BadData" generator:            {{verifies valid input data to this VHDL module}}

-- These concurrent statements look at the raw data inputs
-- and determines if the data set is good or bad for this particular VHDL.
-- The state of BadData is considered valid on the assertion of n_STRT.
-- If the data is bad, then the engine will NOT be enabled and the
-- output EngFAULT will be asserted.  If BadData = 0, then the main
-- Gray Code engine will be started and the inputs registered.

        BD0 <= '1' when
                                (DeviceID /= "0101") = TRUE -- only valid Device ID for the X9271
                                else '0';
                                        --  Special Note: this test may need expansion if other devices
                                        --  using this engine also have different Device ID's.

        BD1 <= '1' when
                                (InsOpCode /= "1010"
                                 and InsOpCode /= "1100") = TRUE
                                                --      "1010" for write wiper counter register, or
                                                --      "1100" for write data register
                                else '0';

        BD2 <= '1' when
                                (InsOpCode = "1010" AND RegADD /= "00")   = TRUE
                                                -- If doing a write wiper counter register operation
                                                -- the Register addresses must be zeroes
                                else '0';

        BD3 <= '1' when
                                (InsOpCode = "1010" AND BankADD /= "00") = TRUE
                                                -- If doing a write wiper counter register operation
                                                -- the Bank addresses must be zeroes
                                else '0';

        BadData <= '1' when -- BadData if any data test failed
                (BD0 = '1' or BD1 = '1' or BD2 = '1' or BD3 = '1')
                                        else '0';

-- ********************************************************************

--      "EngFAULT" generator: {{post a fault on start attempt if bad data}}

engflt : process (SPIwrCLK,n_RST)
begin
  if n_RST = '0' then -- turn Off fault indicator on reset, otherwise:
    EngFAULT <= '0';
```

```
  elsif (SPIwrCLK'event and SPIwrCLK = '1') then -- on the rising clock:
                        if (n_STRT = '0'    AND          -- a request to start is received
                                BadData = '1') THEN              -- and the input data is NOT OK
                                        EngFAULT <= '1';  -- then Post the error, otherwise:
                        elsif
                                (n_STRT = '0'    AND          -- a request to start is received
                                BadData = '0') THEN              -- and the input data is is OK
                                        EngFAULT <= '0';  -- then remove the error indication
                        end if;
  end if;
end process engflt;

--  Application Note: If bad data is applied to this VHDL module and an
--  attempt to start the engine is detected, then EngFAULT will be asserted
--  (high) and the engine will NOT run.
--  To clear the fault condition (without a full reset, that is) the user
--  simply needs to correct the input data and try to start the engine
--  again.  If the data is OK, the previous fault condition will be
--  cleared and the engine will start normally.  If another fault exists
--  with the new data, EngFAULT will remain asserted until the proper
--  data exists on the inputs at the assertion of n_STRT.

-- ********************************************************************

--          "RegInputs" generator: {{Latch the input data on engine startup}}

RegisterInputs : process (SPIwrCLK,n_RST)
begin
  if n_RST = '0' then -- clear input registers on reset, otherwise:
    RegInputs <= "000000000000000000000000";
  elsif (SPIwrCLK'event and SPIwrCLK = '1') then -- on the rising clock:
                        if (n_ENA = '0' AND GrayCnt66 = "0000001") then
                                        -- the counter is now at count=1 (it's running); register inputs
                        RegInputs(23 downto 0) <= DeviceID(3 downto 0) &  -- 23->20
                                                  "00" &                  -- 19->18
                                                  BusADD(1 downto 0) &          -- 17->16
                                                  InsOpCode(3 downto 0) & -- 15->12
                                                  RegADD(1 downto 0) &   -- 11->10
                                                  BankADD(1 downto 0) &  -- 9->8
                                                  NVdataOut(7 downto 0); -- 7->0

                        end if;
  end if;
end process RegisterInputs;

-- Application Note: Once the engine is running (SPI3BwrBsy = 1), the user
-- should wait at least one SPIwrCLK cycle before making any chnages to the
-- data applied to this module.  Since the data was internally registered
-- by this module, the user can then prepare the data inputs for the next
-- write operation while the current one is being executed, giving the user
-- about 13uS for this preparation work before the current write is completed.
-- If the engine is writing to a data register, then another delay is provided
-- for executing the (very long) high voltage non-volatile write operation.
-- If the engine is writing to a wiper register, then the next operation can
-- occur immediately following the current write operation (when SPI3BwrBsy
-- is negated low).  The 13uS minimum time goes up as a slower clock is used.

-- ********************************************************************

--          "n_ENA" generator: {{Controls the running of the main engine}}

enaEngine : process (SPIwrCLK,n_RST)
```

```
begin
  if n_RST = '0' then -- disable counter on reset, otherwise:
    n_ENA <= '1';
  elsif (SPIwrCLK'event and SPIwrCLK = '1') then -- on the rising clock:
                        if (n_STRT = '0' AND          -- a request to start is received
                                BadData = '0' AND                   -- and the input data is OK and
                                GrayCnt66 = "0000000") then -- the counter is now at zero
                                        n_ENA <= '0';  -- then enable the counter to run, otherwise:
                        elsif (n_ENA = '0' and GrayCnt66 = "1000000") then
                                        n_ENA <= '1'; -- disable the counter on the last count = 40
                        end if;
  end if;
end process enaEngine;

-- ***********************************************************************

        SPI3BwrBsy <= NOT n_ENA;  -- drive the BUSY port, inverted Enable

-- ***********************************************************************

-- The main sequence engine, a 66 count gray code engine is:

graystate : Process (n_ENA, n_RST, GrayCnt66)
begin
If (n_ENA = '1' or n_RST = '0') then nextGray <= "0000000";
                                -- counter = zero during reset or when Not enabled

        else        -- if enabled and Not in Reset then find next count:

case GrayCnt66 is -- This unique code by Tom Gustin 22->25 Jan 07
-- Count #1: 0->1hex
    when "0000000" =>
      nextGray <= "0000001";
-- Count #2: 1->3hex
    when "0000001" =>
      nextGray <= "0000011";
-- Count #3: 3->2hex
    when "0000011" =>
      nextGray <= "0000010";
-- Count #4: 2->6hex
    when "0000010" =>
      nextGray <= "0000110";
-- Count #5: 6->7hex
    when "0000110" =>
      nextGray <= "0000111";
-- Count #6: 7->5hex
    when "0000111" =>
      nextGray <= "0000101";
-- Count #7: 5->4hex
    when "0000101" =>
      nextGray <= "0000100";
-- Count #8: 4->Chex
    when "0000100" =>
      nextGray <= "0001100";
-- Count #9: C->Dhex
    when "0001100" =>
      nextGray <= "0001101";
-- Count #10: D->Fhex
    when "0001101" =>
      nextGray <= "0001111";
-- Count #11: F->Ehex
    when "0001111" =>
```

```
        nextGray <= "0001110";
-- Count #12: E->Ahex
      when "0001110" =>
        nextGray <= "0001010";
-- Count #13: A->Bhex
      when "0001010" =>
        nextGray <= "0001011";
-- Count #14: B->9hex
      when "0001011" =>
        nextGray <= "0001001";
-- Count #15: 9->8hex
      when "0001001" =>
        nextGray <= "0001000";
-- Count #16: 8->18hex
      when "0001000" =>
        nextGray <= "0011000";
-- Count #17: 18->19hex
      when "0011000" =>
        nextGray <= "0011001";
-- Count #18: 19->1Bhex
      when "0011001" =>
        nextGray <= "0011011";
-- Count #19: 1B->1Ahex
      when "0011011" =>
        nextGray <= "0011010";
-- Count #20: 1A->1Ehex
      when "0011010" =>
        nextGray <= "0011110";
-- Count #21: 1E->1Fhex
      when "0011110" =>
        nextGray <= "0011111";
-- Count #22: 1F->1Dhex
      when "0011111" =>
        nextGray <= "0011101";
-- Count #23: 1D->1Chex
      when "0011101" =>
        nextGray <= "0011100";
-- Count #24: 1C->14hex
      when "0011100" =>
        nextGray <= "0010100";
-- Count #25: 14->15hex
      when "0010100" =>
        nextGray <= "0010101";
-- Count #26: 15->17hex
      when "0010101" =>
        nextGray <= "0010111";
-- Count #27: 17->16hex
      when "0010111" =>
        nextGray <= "0010110";
-- Count #28: 16->12hex
      when "0010110" =>
        nextGray <= "0010010";
-- Count #29: 12->13hex
      when "0010010" =>
        nextGray <= "0010011";
-- Count #30: 13->11hex
      when "0010011" =>
        nextGray <= "0010001";
-- Count #31: 11->10hex
      when "0010001" =>
        nextGray <= "0010000";
-- Count #32: 10-30hex
```

```
      when "0010000" =>
        nextGray <= "0110000";
-- Count #33: 30->31hex
      when "0110000" =>
        nextGray <= "0110001";
-- Count #34: 31-33hex
      when "0110001" =>
        nextGray <= "0110011";
-- Count #35: 33->32hex
      when "0110011" =>
        nextGray <= "0110010";
-- Count #36: 32->36hex
      when "0110010" =>
        nextGray <= "0110110";
-- Count #37: 36->37hex
      when "0110110" =>
        nextGray <= "0110111";
-- Count #38: 37->35hex
      when "0110111" =>
        nextGray <= "0110101";
-- Count #39: 35->34hex
      when "0110101" =>
        nextGray <= "0110100";
-- Count #40: 34->3Chex
      when "0110100" =>
        nextGray <= "0111100";
-- Count #41: 3C->3Dhex
      when "0111100" =>
        nextGray <= "0111101";
-- Count #42: 3D->3Fhex
      when "0111101" =>
        nextGray <= "0111111";
-- Count #43: 3F->3Ehex
      when "0111111" =>
        nextGray <= "0111110";
-- Count #44: 3E->3Ahex
      when "0111110" =>
        nextGray <= "0111010";
-- Count #45: 3A->3Bhex
      when "0111010" =>
        nextGray <= "0111011";
-- Count #46: 3B->39hex
      when "0111011" =>
        nextGray <= "0111001";
-- Count #47: 39->38hex
      when "0111001" =>
        nextGray <= "0111000";
-- Count #48: 38->28hex
      when "0111000" =>
        nextGray <= "0101000";
-- Count #49: 28->29hex
     when "0101000" =>
        nextGray <= "0101001";
-- Count #50: 29->2Bhex
     when "0101001" =>
        nextGray <= "0101011";
-- Count #51: 2B->2Ahex
     when "0101011" =>
        nextGray <= "0101010";
-- Count #52: 2A->2Ehex
     when "0101010" =>
        nextGray <= "0101110";
```

```
-- Count #53: 2E->2Fhex
    when "0101110" =>
       nextGray <= "0101111";
-- Count #54: 2F->2Dhex
    when "0101111" =>
       nextGray <= "0101101";
-- Count #55: 2D->2Chex
    when "0101101" =>
       nextGray <= "0101100";
-- Count #56: 2C->24hex
    when "0101100" =>
       nextGray <= "0100100";
-- Count #57: 24->25hex
    when "0100100" =>
       nextGray <= "0100101";
-- Count #58: 25->27hex
    when "0100101" =>
       nextGray <= "0100111";
-- Count #59: 27->26hex
    when "0100111" =>
       nextGray <= "0100110";
-- Count #60: 26->22hex
     when "0100110" =>
       nextGray <= "0100010";
-- Count #61: 22->23hex
     when "0100010" =>
       nextGray <= "0100011";
-- Count #62: 23->21hex
     when "0100011" =>
       nextGray <= "0100001";
-- Count #63: 21->20hex
     when "0100001" =>
       nextGray <= "0100000";
-- Count #64: 20->60hex
     when "0100000" =>
       nextGray <= "1100000";
-- Count #65: 60->40hex
    when "1100000" =>
       nextGray <= "1000000";
-- Count #66: 40->00hex
     when "1000000" =>
       nextGray <= "0000000";

     when others => -- unlikely miscounts return to zero
       nextGray <= "0000000";
  end case;
          end if;
End Process graystate;

-- ***********************************************************************

--                      MAIN Gray Code Register set (7 D-Flipflops)
--                      On every rising SPI Write Clock, the "nextGray" state value
--                      is clocked into the flipflops; Q-outputs = "GrayCnt66"
--                      (see "GrayState" Process above) based upon "Gray66Cnt.xls".

RunEng : Process (SPIwrCLK)
begin
          If (SPIwrCLK'Event AND SPIwrCLK = '1') THEN
                    GrayCnt66 <= nextGray;
          End if;
End Process RunEng;
```

```
-- ********************************************************************

-- "n_CS" generator:  {{SPI device Chip Select signal generator}}
--  Per "Gray66Cnt.xls" n_CS is enabled for assertion at the
--  third count state of #03 (actually "low" on the fourth count
--  state of #02), and is prepared for negation at the 65rd count
--  state of #60 (acutally "high on the 66th count state of #40).

CSdrv : process (SPIwrCLK,n_RST)
begin
  if n_RST = '0' then -- negate Chip Select on reset, otherwise:
    n_CS <= '1';
  elsif (SPIwrCLK'event and SPIwrCLK = '1') then -- on the rising clock:
                        if GrayCnt66 = "0000011" then -- if the counter is now at count=03
                                        n_CS <= '0';  -- then enable the Chip Select signal.
                        elsif GrayCnt66 = "1100000" then -- if the counter is at count=60
                                        n_CS <= '1'; -- then disable the Chip Select signal.
                        end if;
  end if;
end process CSdrv;

-- ********************************************************************

-- "SCL" generator:  {{SPI device Serial Clock signal generator}}
-- Per "Gray66Cnt.xls" SCL is enabled for assertion 24 times every odd
-- count starting with the 5th and ending with the 51st, and is enabled
-- for negation 24 times every even count starting with the 6th and
-- ending with the 52nd.  Recall that the output always lags the
-- enabling count by one count.  Therefore, the first assertion will
-- actually occur on the 6th count, Gray Code = 07, while the last
-- negation will actually occur on the 53rd count, Gray Code = 2E.

SCLdrv : process (SPIwrCLK,n_rst)
begin
            if n_RST = '0' then -- negate Serial Clock on reset, otherwise:
    SCL <= '0';
  elsif (SPIwrCLK'event and SPIwrCLK = '1') then -- on the rising clock:
                        if (GrayCnt66 = "0000110" OR -- assert SCL on GrayCount = 06       OR
                                        GrayCnt66 = "0000101" OR -- assert SCL on GrayCount = 05        OR
                                        GrayCnt66 = "0001100" OR -- assert SCL on GrayCount = 0C        OR
                                        GrayCnt66 = "0001111" OR -- assert SCL on GrayCount = 0F        OR
                                        GrayCnt66 = "0001010" OR -- assert SCL on GrayCount = 0A        OR
                                        GrayCnt66 = "0001001" OR -- assert SCL on GrayCount = 09        OR
                                        GrayCnt66 = "0011000" OR -- assert SCL on GrayCount = 18        OR
                                        GrayCnt66 = "0011011" OR -- assert SCL on GrayCount = 1B        OR
                                        GrayCnt66 = "0011110" OR -- assert SCL on GrayCount = 1E        OR
                                        GrayCnt66 = "0011101" OR -- assert SCL on GrayCount = 1D        OR
                                        GrayCnt66 = "0010100" OR -- assert SCL on GrayCount = 14        OR
                                        GrayCnt66 = "0010111" OR -- assert SCL on GrayCount = 17        OR
                                        GrayCnt66 = "0010010" OR -- assert SCL on GrayCount = 12        OR
                                        GrayCnt66 = "0010001" OR -- assert SCL on GrayCount = 11        OR
                                        GrayCnt66 = "0110000" OR -- assert SCL on GrayCount = 30        OR
                                        GrayCnt66 = "0110011" OR -- assert SCL on GrayCount = 33        OR
                                        GrayCnt66 = "0110110" OR -- assert SCL on GrayCount = 36        OR
                                        GrayCnt66 = "0110101" OR -- assert SCL on GrayCount = 35        OR
                                        GrayCnt66 = "0111100" OR -- assert SCL on GrayCount = 3C        OR
                                        GrayCnt66 = "0111111" OR -- assert SCL on GrayCount = 3F        OR
                                        GrayCnt66 = "0111010" OR -- assert SCL on GrayCount = 3A        OR
                                        GrayCnt66 = "0111001" OR -- assert SCL on GrayCount = 39        OR
                                        GrayCnt66 = "0101000" OR -- assert SCL on GrayCount = 28        OR
                                        GrayCnt66 = "0101011") then -- assert SCL on GrayCount = 2B then
```

```
                                        SCL <= '1';
                    else -- otherwise negate SCL
                                        SCL <= '0';
                    end if;
            end if;
end process SCLdrv;

-- ********************************************************************

-- "SI" Generator:  {{Serial Input data multiplexer for SPI device}}
-- This asynchronous conditional statement drives the appropriate bit
-- onto the SI signal line based upon the state of the Gray Code Engine.
-- Each of the 24 data items is driven for two engine states each.

SIgen : process (GrayCnt66,RegInputs(23 downto 0))
begin
  case GrayCnt66 is
    when "0000110" => SI <= RegInputs(23); -- Gray Count = 06
    when "0000111" => SI <= RegInputs(23); -- Gray Count = 07
    when "0000101" => SI <= RegInputs(22); -- Gray Count = 05
    when "0000100" => SI <= RegInputs(22); -- Gray Count = 04
                      when "0001100" => SI <= RegInputs(21); -- Gray Count = 0C
    when "0001101" => SI <= RegInputs(21); -- Gray Count = 0D
                      when "0001111" => SI <= RegInputs(20); -- Gray Count = 0F
    when "0001110" => SI <= RegInputs(20); -- Gray Count = 0E
                      when "0001010" => SI <= RegInputs(19); -- Gray Count = 0A
    when "0001011" => SI <= RegInputs(19); -- Gray Count = 0B
                      when "0001001" => SI <= RegInputs(18); -- Gray Count = 09
    when "0001000" => SI <= RegInputs(18); -- Gray Count = 08
                      when "0011000" => SI <= RegInputs(17); -- Gray Count = 18
    when "0011001" => SI <= RegInputs(17); -- Gray Count = 19
                      when "0011011" => SI <= RegInputs(16); -- Gray Count = 1B
    when "0011010" => SI <= RegInputs(16); -- Gray Count = 1A
                      when "0011110" => SI <= RegInputs(15); -- Gray Count = 1E
    when "0011111" => SI <= RegInputs(15); -- Gray Count = 1F
                      when "0011101" => SI <= RegInputs(14); -- Gray Count = 1D
    when "0011100" => SI <= RegInputs(14); -- Gray Count = 1C
                      when "0010100" => SI <= RegInputs(13); -- Gray Count = 14
    when "0010101" => SI <= RegInputs(13); -- Gray Count = 15
                      when "0010111" => SI <= RegInputs(12); -- Gray Count = 17
    when "0010110" => SI <= RegInputs(12); -- Gray Count = 16
                      when "0010010" => SI <= RegInputs(11); -- Gray Count = 12
    when "0010011" => SI <= RegInputs(11); -- Gray Count = 13
                      when "0010001" => SI <= RegInputs(10); -- Gray Count = 11
    when "0010000" => SI <= RegInputs(10); -- Gray Count = 10
                      when "0110000" => SI <= RegInputs(9); -- Gray Count = 30
    when "0110001" => SI <= RegInputs(9); -- Gray Count = 31
    when "0110011" => SI <= RegInputs(8); -- Gray Count = 33
    when "0110010" => SI <= RegInputs(8); -- Gray Count = 32
                      when "0110110" => SI <= RegInputs(7); -- Gray Count = 36
    when "0110111" => SI <= RegInputs(7); -- Gray Count = 37
                      when "0110101" => SI <= RegInputs(6); -- Gray Count = 35
    when "0110100" => SI <= RegInputs(6); -- Gray Count = 34
                      when "0111100" => SI <= RegInputs(5); -- Gray Count = 3C
    when "0111101" => SI <= RegInputs(5); -- Gray Count = 3D
                      when "0111111" => SI <= RegInputs(4); -- Gray Count = 3F
    when "0111110" => SI <= RegInputs(4); -- Gray Count = 3E
                      when "0111010" => SI <= RegInputs(3); -- Gray Count = 3A
    when "0111011" => SI <= RegInputs(3); -- Gray Count = 3B
                      when "0111001" => SI <= RegInputs(2); -- Gray Count = 39
    when "0111000" => SI <= RegInputs(2); -- Gray Count = 38
                      when "0101000" => SI <= RegInputs(1); -- Gray Count = 28
```

```
    when "0101001" => SI <= RegInputs(1); -- Gray Count = 29
                      when "0101011" => SI <= RegInputs(0); -- Gray Count = 2B
    when "0101010" => SI <= RegInputs(0); -- Gray Count = 2A
    when others => SI <= '0'; -- all other states = zero
  end case;
end process SIgen;

-- ***********************************************************************

end Behavioral; -- finished & compiled 25 Jan 07
```