```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.trb_net_std.all;
use work.nxyter_components.all;

entity adc_ad9228 is
  port (
    CLK_IN               : in  std_logic;
    RESET_IN             : in  std_logic;
    CLK_ADCDAT_IN        : in  std_logic;

    ADC0_SCLK_IN         : in  std_logic;  -- Sampling Clock ADC0
    ADC0_SCLK_OUT        : out std_logic;
    ADC0_DATA_A_IN       : in  std_logic;
    ADC0_DATA_B_IN       : in  std_logic;
    ADC0_DATA_C_IN       : in  std_logic;
    ADC0_DATA_D_IN       : in  std_logic;
    ADC0_DCLK_IN         : in  std_logic;  -- Data Clock from ADC0
    ADC0_FCLK_IN         : in  std_logic;  -- Frame Clock from ADC0

    ADC1_SCLK_IN         : in  std_logic;  -- Sampling Clock ADC1
    ADC1_SCLK_OUT        : out std_logic;
    ADC1_DATA_A_IN       : in  std_logic;
    ADC1_DATA_B_IN       : in  std_logic;
    ADC1_DATA_C_IN       : in  std_logic;
    ADC1_DATA_D_IN       : in  std_logic;
    ADC1_DCLK_IN         : in  std_logic;  -- Data Clock from ADC1
    ADC1_FCLK_IN         : in  std_logic;  -- Frame Clock from ADC1

    ADC0_DATA_A_OUT      : out std_logic_vector(11 downto 0);
    ADC0_DATA_B_OUT      : out std_logic_vector(11 downto 0);
    ADC0_DATA_C_OUT      : out std_logic_vector(11 downto 0);
    ADC0_DATA_D_OUT      : out std_logic_vector(11 downto 0);
    ADC0_DATA_VALID_OUT  : out std_logic;

    ADC1_DATA_A_OUT      : out std_logic_vector(11 downto 0);
    ADC1_DATA_B_OUT      : out std_logic_vector(11 downto 0);
    ADC1_DATA_C_OUT      : out std_logic_vector(11 downto 0);
    ADC1_DATA_D_OUT      : out std_logic_vector(11 downto 0);
    ADC1_DATA_VALID_OUT  : out std_logic;
    ADC0_NOTLOCK_COUNTER : out unsigned(7 downto 0);
    ADC1_NOTLOCK_COUNTER : out unsigned(7 downto 0);

    ERROR_ADC0_OUT       : out std_logic;
    ERROR_ADC1_OUT       : out std_logic;
    DEBUG_IN             : in  std_logic_vector(3 downto 0);
    DEBUG_OUT            : out std_logic_vector(15 downto 0)
    );
end adc_ad9228;

architecture Behavioral of  adc_ad9228 is

  -- DDR Generic Handler
  signal DDR_DATA_CLK           : std_logic;
  signal q_0                    : std_logic_vector(19 downto 0);
  signal q_1                    : std_logic_vector(19 downto 0);

  -- NotLock Counters
  signal adc0_frame_notlocked   : std_logic;
```



```
  signal adc0_frame_notlocked_p : std_logic;
  signal adc0_notlock_ctr       : unsigned(7 downto 0);
  signal adc0_bit_shift         : unsigned(1 downto 0);
  signal adc0_bit_shift_last    : unsigned(1 downto 0);
  signal adc0_bit_shift_change  : std_logic;

  signal adc1_frame_notlocked   : std_logic;
  signal adc1_frame_notlocked_p : std_logic;
  signal adc1_notlock_ctr       : unsigned(7 downto 0);
  signal adc1_bit_shift         : unsigned(1 downto 0);
  signal adc1_bit_shift_last    : unsigned(1 downto 0);
  signal adc1_bit_shift_change  : std_logic;

  -- Merge Data
  type q_map_t          is array(0 to 4) of std_logic_vector(3 downto 0);
  type adc_data_buf_t   is array(0 to 4) of std_logic_vector(15 downto 0);
  type adc_data_t       is array(0 to 3) of std_logic_vector(11 downto 0);

  signal adc0_data_buf          : adc_data_buf_t;
  signal adc0_frame_ctr         : unsigned(2 downto 0);
  signal adc0_frame_locked      : std_logic;

  signal adc0_new_data_t        : std_logic;
  signal adc0_data_t            : adc_data_t;

  signal adc1_data_buf          : adc_data_buf_t;
  signal adc1_frame_ctr         : unsigned(2 downto 0);
  signal adc1_frame_locked      : std_logic;

  signal adc1_new_data_t        : std_logic;
  signal adc1_data_t            : adc_data_t;

  -- Clock Transfer
  signal adc0_fifo_empty        :  std_logic;
  signal adc0_fifo_full         :  std_logic;
  signal adc0_write_enable      :  std_logic;
  signal adc0_read_enable       :  std_logic;
  signal adc0_read_enable_t     :  std_logic;
  signal adc0_read_enable_tt    :  std_logic;
  signal adc0_fifo_reset        :  std_logic;

  signal adc1_fifo_empty        :  std_logic;
  signal adc1_fifo_full         :  std_logic;
  signal adc1_write_enable      :  std_logic;
  signal adc1_read_enable       :  std_logic;
  signal adc1_read_enable_t     :  std_logic;
  signal adc1_read_enable_tt    :  std_logic;
  signal adc1_fifo_reset        :  std_logic;

  -- Error
  signal error_adc0_o           : std_logic;
  signal error_adc1_o           : std_logic;

  -- Output
  signal adc0_data_valid_o      : std_logic;
  signal adc0_data_f            : adc_data_t;
  signal adc0_data_o            : adc_data_t;

  signal adc1_data_valid_o      : std_logic;
  signal adc1_data_f            : adc_data_t;
  signal adc1_data_o            : adc_data_t;
```

```
  -- Resets
  signal RESET_CLK_ADCDAT_IN    : std_logic;
  signal RESET_DDR_DATA_CLK     : std_logic;

begin

  PROC_DEBUG: process (DEBUG_IN)
  begin
    case DEBUG_IN is
      when x"0" =>
        -- DEBUG
        DEBUG_OUT(0)            <= CLK_IN;
        DEBUG_OUT(1)            <= DDR_DATA_CLK;
        DEBUG_OUT(2)            <= adc0_bit_shift_change;
        DEBUG_OUT(3)            <= adc0_write_enable;
        DEBUG_OUT(4)            <= adc0_fifo_full;
        DEBUG_OUT(5)            <= adc0_fifo_empty;
        DEBUG_OUT(6)            <= adc0_frame_locked;
        DEBUG_OUT(7)            <= adc0_new_data_t;
        DEBUG_OUT(8)            <= adc0_read_enable;
        DEBUG_OUT(9)            <= adc0_read_enable_t;
        DEBUG_OUT(10)           <= adc0_read_enable_tt;
        DEBUG_OUT(11)           <= adc0_data_valid_o;
        DEBUG_OUT(15 downto 12) <= (others => '0');

      when x"1" =>
        DEBUG_OUT               <= adc0_data_buf(0);

      when x"2" =>
        DEBUG_OUT               <= adc0_data_buf(1);

      when x"3" =>
        DEBUG_OUT               <= adc0_data_buf(2);

      when x"4" =>
        DEBUG_OUT               <= adc0_data_buf(3);

      when x"5" =>
        DEBUG_OUT               <= adc0_data_buf(4);

      --when x"e" =>
      --  DEBUG_OUT               <= q_0(15 downto 0);

      --when x"f" =>
      --  DEBUG_OUT               <= q_1(15 downto 0);

      when others =>
        DEBUG_OUT               <= (others => '0');
    end case;
  end process PROC_DEBUG;

  ------------------------------------------------------------------------------
  -- Reset Domain Transfer
  ------------------------------------------------------------------------------
  signal_async_trans_RESET_IN: signal_async_trans
    port map (
      CLK_IN      => CLK_ADCDAT_IN,
      SIGNAL_A_IN => RESET_IN,
      SIGNAL_OUT  => RESET_CLK_ADCDAT_IN
    );
```



```
  signal_async_trans_RESET_IN_2: signal_async_trans
    port map (
      CLK_IN      => DDR_DATA_CLK,
      SIGNAL_A_IN => RESET_IN,
      SIGNAL_OUT  => RESET_DDR_DATA_CLK
    );

  ------------------------------------------------------------------------------

  adc_ddr_generic_1: adc_ddr_generic
    port map (
      clk_0          => ADC0_DCLK_IN,
      clk_1          => ADC1_DCLK_IN,
      clkdiv_reset   => RESET_CLK_ADCDAT_IN,
      eclk           => CLK_ADCDAT_IN,
      reset_0        => RESET_DDR_DATA_CLK,
      reset_1        => RESET_DDR_DATA_CLK,
      sclk           => DDR_DATA_CLK,

      datain_0(0)    => ADC0_DATA_A_IN,
      datain_0(1)    => ADC0_DATA_B_IN,
      datain_0(2)    => ADC0_DATA_C_IN,
      datain_0(3)    => ADC0_DATA_D_IN,
      datain_0(4)    => ADC0_FCLK_IN,

      datain_1(0)    => ADC1_DATA_A_IN,
      datain_1(1)    => ADC1_DATA_B_IN,
      datain_1(2)    => ADC1_DATA_C_IN,
      datain_1(3)    => ADC1_DATA_D_IN,
      datain_1(4)    => ADC1_FCLK_IN,

      q_0            => q_0,
      q_1            => q_1
      );

  ------------------------------------------------------------------------------

  PROC_MERGE_DATA0: process(DDR_DATA_CLK)
    variable q_0_map  : q_map_t;
  begin
    if (rising_edge(DDR_DATA_CLK)) then
      -- Remap DDR Output q_value
      for I in 0 to 4 loop
        q_0_map(I) := q_0(I + 0) & q_0(I + 5) & q_0(I + 10) & q_0(I + 15);
      end loop;

      for I in 0 to 4 loop
        adc0_data_buf(I)(3 downto 0)  <= q_0_map(I);
        adc0_data_buf(I)(15 downto 4) <= adc0_data_buf(I)(11 downto 0);
      end loop;

      if (RESET_DDR_DATA_CLK = '1') then
        adc0_new_data_t        <= '0';
        adc0_frame_ctr         <= (others => '0');
        adc0_frame_locked      <= '0';
        adc0_bit_shift         <= "00";
        adc0_bit_shift_last    <= "00";
        adc0_bit_shift_change  <= '0';
      else
        -- Test Frame Clock Pattern
        adc0_new_data_t                <= '0';
        case adc0_data_buf(4) is        -- adc0_data_buf(4) is frame clock
```

```
        when "0000111111000000" =>
          for I in 0 to 3 loop
            adc0_data_t(I)            <= adc0_data_buf(I)(11 downto 0);
          end loop;
          adc0_new_data_t             <= '1';
          adc0_bit_shift              <= "00";

        when "0001111110000001" =>
          for I in 0 to 3 loop
            adc0_data_t(I)            <= adc0_data_buf(I)(12 downto 1);
          end loop;
          adc0_new_data_t             <= '1';
          adc0_bit_shift              <= "01";

        when "0011111100000011" =>
          for I in 0 to 3 loop
            adc0_data_t(I)            <= adc0_data_buf(I)(13 downto 2);
          end loop;
          adc0_new_data_t             <= '1';
          adc0_bit_shift              <= "10";

        when "0111111000000111" =>
          for I in 0 to 3 loop
            adc0_data_t(I)            <= adc0_data_buf(I)(14 downto 3);
          end loop;
          adc0_new_data_t             <= '1';
          adc0_bit_shift              <= "11";

        when others => null;

      end case;

      -- ADC Lock Status
      if (adc0_new_data_t = '1') then
        adc0_frame_ctr                <= (others => '0');
        adc0_frame_locked             <= '1';
      elsif (adc0_frame_ctr < x"4") then
        adc0_frame_ctr                <= adc0_frame_ctr + 1;
      else
        adc0_frame_locked             <= '0';
      end if;

      adc0_bit_shift_last             <= adc0_bit_shift;
      if (adc0_bit_shift /= adc0_bit_shift_last) then
        adc0_bit_shift_change         <= '1';
      else
        adc0_bit_shift_change         <= '0';
      end if;

    end if;
  end if;
end process PROC_MERGE_DATA0;

-------------------------------------------------------------------------------

PROC_MERGE_DATA1: process(DDR_DATA_CLK)
  variable q_1_map  : q_map_t;
begin
  if (rising_edge(DDR_DATA_CLK)) then
    -- Remap DDR Output q_value
    for I in 0 to 4 loop
      q_1_map(I) := q_1(I + 0) & q_1(I + 5) & q_1(I + 10) & q_1(I + 15);
```



```
    end loop;

    for I in 0 to 4 loop
      adc1_data_buf(I)(3 downto 0)  <= q_1_map(I);
      adc1_data_buf(I)(15 downto 4) <= adc1_data_buf(I)(11 downto 0);
    end loop;

    if (RESET_DDR_DATA_CLK = '1') then
      adc1_new_data_t         <= '0';
      adc1_frame_ctr          <= (others => '0');
      adc1_frame_locked       <= '0';
      adc1_bit_shift          <= "00";
      adc1_bit_shift_last     <= "00";
      adc1_bit_shift_change   <= '0';
    else
      -- Test Frame Clock Pattern
      adc1_new_data_t                 <= '0';
      case adc1_data_buf(4) is          -- adc1_data_buf(4) is frame clock
        when "0000111111000000" =>
          for I in 0 to 3 loop
            adc1_data_t(I)            <= adc1_data_buf(I)(11 downto 0);
          end loop;
          adc1_new_data_t             <= '1';
          adc1_bit_shift              <= "00";

        when "0001111110000001" =>
          for I in 0 to 3 loop
            adc1_data_t(I)            <= adc1_data_buf(I)(12 downto 1);
          end loop;
          adc1_new_data_t             <= '1';
          adc1_bit_shift              <= "01";

        when "0011111100000011" =>
          for I in 0 to 3 loop
            adc1_data_t(I)            <= adc1_data_buf(I)(13 downto 2);
          end loop;
          adc1_new_data_t             <= '1';
          adc1_bit_shift              <= "10";

        when "0111111000000111" =>
          for I in 0 to 3 loop
            adc1_data_t(I)            <= adc1_data_buf(I)(14 downto 3);
          end loop;
          adc1_new_data_t             <= '1';
          adc1_bit_shift              <= "11";

        when others => null;

      end case;

      -- ADC Lock Status
      if (adc1_new_data_t = '1') then
        adc1_frame_ctr                <= (others => '0');
        adc1_frame_locked             <= '1';
      elsif (adc1_frame_ctr < x"4") then
        adc1_frame_ctr                <= adc1_frame_ctr + 1;
      else
        adc1_frame_locked             <= '0';
      end if;

      adc1_bit_shift_last             <= adc1_bit_shift;
      if (adc1_bit_shift /= adc1_bit_shift_last) then
```

```
        adc1_bit_shift_change      <= '1';
      else
        adc1_bit_shift_change      <= '0';
      end if;

    end if;
  end if;
end process PROC_MERGE_DATA1;

-------------------------------------------------------------------------------
-- Tansfer to CLK_IN
-------------------------------------------------------------------------------

fifo_adc_48to48_dc_1: fifo_adc_48to48_dc
  port map (
    Data(11 downto 0)  => adc0_data_t(0),
    Data(23 downto 12) => adc0_data_t(1),
    Data(35 downto 24) => adc0_data_t(2),
    Data(47 downto 36) => adc0_data_t(3),
    WrClock            => DDR_DATA_CLK,
    RdClock            => CLK_IN,
    WrEn               => adc0_write_enable,
    RdEn               => adc0_read_enable,
    Reset              => RESET_IN,
    RPReset            => adc0_fifo_reset,
    Q(11 downto 0)     => adc0_data_f(0),
    Q(23 downto 12)    => adc0_data_f(1),
    Q(35 downto 24)    => adc0_data_f(2),
    Q(47 downto 36)    => adc0_data_f(3),
    Empty              => adc0_fifo_empty,
    Full               => adc0_fifo_full
    );

-- Readout Handler
adc0_fifo_reset      <= RESET_IN;
adc0_write_enable    <= adc0_new_data_t and not adc0_fifo_full;
adc0_read_enable     <= not adc0_fifo_empty;

PROC_ADC0_FIFO_READ: process(CLK_IN)
begin
  if (rising_edge(CLK_IN)) then
    adc0_read_enable_t   <= adc0_read_enable;
    if (RESET_IN = '1') then
      adc0_read_enable_tt  <= '0';
      for I in 0 to 3 loop
        adc0_data_o(I)     <= (others => '0');
      end loop;
      adc0_data_valid_o    <= '0';
    else
      -- Read enable
      adc0_read_enable_tt  <= adc0_read_enable_t;

      if (adc0_read_enable_tt = '1') then
        for I in 0 to 3 loop
          adc0_data_o(I)   <= adc0_data_f(I);
        end loop;
        adc0_data_valid_o  <= '1';
      else
        adc0_data_valid_o  <= '0';
      end if;
    end if;
  end if;
```



```
end process PROC_ADC0_FIFO_READ;

-------------------------------------------------------------------------------

fifo_adc_48to48_dc_2: fifo_adc_48to48_dc
  port map (
    Data(11 downto 0)  => adc1_data_t(0),
    Data(23 downto 12) => adc1_data_t(1),
    Data(35 downto 24) => adc1_data_t(2),
    Data(47 downto 36) => adc1_data_t(3),
    WrClock            => DDR_DATA_CLK,
    RdClock            => CLK_IN,
    WrEn               => adc1_new_data_t,
    RdEn               => adc1_read_enable,
    Reset              => RESET_IN,
    RPReset            => adc1_fifo_reset,
    Q(11 downto 0)     => adc1_data_f(0),
    Q(23 downto 12)    => adc1_data_f(1),
    Q(35 downto 24)    => adc1_data_f(2),
    Q(47 downto 36)    => adc1_data_f(3),
    Empty              => adc1_fifo_empty,
    Full               => adc1_fifo_full
    );

-- Readout Handler
adc1_fifo_reset      <= RESET_IN;
adc1_write_enable    <= adc1_new_data_t and not adc1_fifo_full;
adc1_read_enable     <= not adc1_fifo_empty;

PROC_ADC1_FIFO_READ: process(CLK_IN)
begin
  if (rising_edge(CLK_IN)) then
    if (RESET_IN = '1') then
      adc1_read_enable_t   <= '0';
      adc1_read_enable_tt  <= '0';
      for I in 0 to 3 loop
        adc1_data_o(I)     <= (others => '0');
      end loop;
      adc1_data_valid_o    <= '0';
    else
      -- Read enable
      adc1_read_enable_t   <= adc1_read_enable;
      adc1_read_enable_tt  <= adc1_read_enable_t;

      if (adc1_read_enable_tt = '1') then
        for I in 0 to 3 loop
          adc1_data_o(I)   <= adc1_data_f(I);
        end loop;
        adc1_data_valid_o  <= '1';
      else
        adc1_data_valid_o  <= '0';
      end if;
    end if;
  end if;
end process PROC_ADC1_FIFO_READ;

-------------------------------------------------------------------------------
-- Lock Monitor
-------------------------------------------------------------------------------

level_to_pulse_1: level_to_pulse
  port map (
```

```
    CLK_IN    => DDR_DATA_CLK,
    RESET_IN  => RESET_DDR_DATA_CLK,
    LEVEL_IN  => not adc0_frame_locked,
    PULSE_OUT => adc0_frame_notlocked_p
    );

level_to_pulse_2: level_to_pulse
  port map (
    CLK_IN    => DDR_DATA_CLK,
    RESET_IN  => RESET_DDR_DATA_CLK,
    LEVEL_IN  => not adc1_frame_locked,
    PULSE_OUT => adc1_frame_notlocked_p
    );

pulse_dtrans_1: pulse_dtrans
  generic map (
    CLK_RATIO => 2
    )
  port map (
    CLK_A_IN    => DDR_DATA_CLK,
    RESET_A_IN  => RESET_DDR_DATA_CLK,
    PULSE_A_IN  => adc0_frame_notlocked_p,
    CLK_B_IN    => CLK_IN,
    RESET_B_IN  => RESET_IN,
    PULSE_B_OUT => adc0_frame_notlocked
    );

pulse_dtrans_2: pulse_dtrans
  generic map (
    CLK_RATIO => 2
    )
  port map (
    CLK_A_IN    => DDR_DATA_CLK,
    RESET_A_IN  => RESET_DDR_DATA_CLK,
    PULSE_A_IN  => adc1_frame_notlocked_p,
    CLK_B_IN    => CLK_IN,
    RESET_B_IN  => RESET_IN,
    PULSE_B_OUT => adc1_frame_notlocked
    );

PROC_NOTLOCK_COUNTER: process(CLK_IN)
begin
  if (rising_edge(CLK_IN)) then
    if (RESET_IN = '1') then
      adc0_notlock_ctr     <= (others => '0');
      adc1_notlock_ctr     <= (others => '0');
    else
      if (adc0_frame_notlocked = '1') then
        adc0_notlock_ctr   <= adc0_notlock_ctr + 1;
      end if;

      if (adc1_frame_notlocked = '1') then
        adc1_notlock_ctr   <= adc1_notlock_ctr + 1;
      end if;
    end if;
  end if;
end process PROC_NOTLOCK_COUNTER;

PROC_ERROR: process(CLK_IN)
begin
  if (rising_edge(CLK_IN)) then
    if (RESET_IN = '1') then
```



```
      error_adc0_o     <= '0';
      error_adc1_o     <= '0';
    else
      error_adc0_o     <= '0';
      error_adc1_o     <= '0';

      if (adc0_frame_notlocked = '1' or
          adc0_bit_shift_change = '1') then
        error_adc0_o   <= '1';
      end if;

      if (adc1_frame_notlocked = '1' or
          adc1_bit_shift_change = '1') then
        error_adc1_o   <= '1';
      end if;
    end if;
  end if;
end process PROC_ERROR;

-- Output

ADC0_SCLK_OUT        <= ADC0_SCLK_IN;
ADC1_SCLK_OUT        <= ADC1_SCLK_IN;

ADC0_DATA_A_OUT      <= adc0_data_o(0);
ADC0_DATA_B_OUT      <= adc0_data_o(1);
ADC0_DATA_C_OUT      <= adc0_data_o(2);
ADC0_DATA_D_OUT      <= adc0_data_o(3);
ADC0_DATA_VALID_OUT  <= adc0_data_valid_o;

ADC1_DATA_A_OUT      <= adc1_data_o(0);
ADC1_DATA_B_OUT      <= adc1_data_o(1);
ADC1_DATA_C_OUT      <= adc1_data_o(2);
ADC1_DATA_D_OUT      <= adc1_data_o(3);
ADC1_DATA_VALID_OUT  <= adc1_data_valid_o;

ADC0_NOTLOCK_COUNTER <= adc0_notlock_ctr;
ADC1_NOTLOCK_COUNTER <= adc1_notlock_ctr;

ERROR_ADC0_OUT       <= error_adc0_o;
ERROR_ADC1_OUT       <= error_adc1_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.nxyter_components.all;

entity adc_spi_master is
  generic (
    SPI_SPEED : unsigned(7 downto 0) := x"32"
    );
  port(
    CLK_IN                : in    std_logic;
    RESET_IN              : in    std_logic;

    -- SPI connections
    SCLK_OUT              : out   std_logic;
    SDIO_INOUT            : inout std_logic;
    CSB_OUT               : out   std_logic;
```

```
    -- Internal Interface
    INTERNAL_COMMAND_IN    : in    std_logic_vector(31 downto 0);
    COMMAND_ACK_OUT        : out   std_logic;
    SPI_DATA               : out   std_logic_vector(31 downto 0);
    SPI_LOCK_IN            : in    std_logic;

    -- Slave bus
    SLV_READ_IN            : in    std_logic;
    SLV_WRITE_IN           : in    std_logic;
    SLV_DATA_OUT           : out   std_logic_vector(31 downto 0);
    SLV_DATA_IN            : in    std_logic_vector(31 downto 0);
    SLV_ACK_OUT            : out   std_logic;
    SLV_NO_MORE_DATA_OUT   : out   std_logic;
    SLV_UNKNOWN_ADDR_OUT   : out   std_logic;

    -- Debug Line
    DEBUG_OUT              : out   std_logic_vector(15 downto 0)
    );
end entity;

architecture Behavioral of adc_spi_master is

  signal sdio_i        : std_logic;
  signal sdio_x        : std_logic;
  signal sdio          : std_logic;

  signal sclk_o        : std_logic;
  signal command_ack_o : std_logic;

  -- SPI Master
  signal csb_o                : std_logic;
  signal spi_start            : std_logic;

  signal spi_busy             : std_logic;
  signal takeover_sdio        : std_logic;
  signal wait_timer_start     : std_logic;
  signal sendbyte_seq_start   : std_logic;
  signal readbyte_seq_start   : std_logic;
  signal sendbyte_byte        : std_logic_vector(7 downto 0);
  signal read_seq_ctr         : std_logic;
  signal reg_data             : std_logic_vector(31 downto 0);

  signal spi_busy_x           : std_logic;
  signal wait_timer_start_x   : std_logic;
  signal sendbyte_seq_start_x : std_logic;
  signal sendbyte_byte_x      : std_logic_vector(7 downto 0);
  signal readbyte_seq_start_x : std_logic;
  signal read_seq_ctr_x       : std_logic;
  signal reg_data_x           : std_logic_vector(31 downto 0);

  signal sdio_sendbyte        : std_logic;
  signal sclk_sendbyte        : std_logic;
  signal sendbyte_done        : std_logic;

  signal sclk_readbyte        : std_logic;
  signal readbyte_byte        : std_logic_vector(7 downto 0);
  signal readbyte_done        : std_logic;

  type STATES is (S_RESET,
                  S_IDLE,
                  S_START,
```



```
                  S_START_WAIT,

                  S_SEND_CMD_A,
                  S_SEND_CMD_A_WAIT,
                  S_SEND_CMD_B,
                  S_SEND_CMD_B_WAIT,

                  S_SEND_DATA,
                  S_SEND_DATA_WAIT,
                  S_GET_DATA,
                  S_GET_DATA_WAIT,

                  S_STOP,
                  S_STOP_WAIT
                  );
  signal STATE, NEXT_STATE : STATES;

  -- SPI Timer
  signal wait_timer_done         : std_logic;

  -- TRBNet Slave Bus
  signal slv_data_out_o          : std_logic_vector(31 downto 0);
  signal slv_no_more_data_o      : std_logic;
  signal slv_unknown_addr_o      : std_logic;
  signal slv_ack_o               : std_logic;
  signal spi_chipid              : std_logic_vector(6 downto 0);
  signal spi_rw_bit              : std_logic;
  signal spi_registerid          : std_logic_vector(12 downto 0);
  signal spi_register_data       : std_logic_vector(7 downto 0);
  signal spi_register_value_read : std_logic_vector(7 downto 0);

begin

  -- Timer
  timer_static_1: timer_static
    generic map (
      CTR_WIDTH => 8,
      CTR_END   => to_integer(SPI_SPEED srl 2)
      )
    port map (
      CLK_IN        => CLK_IN,
      RESET_IN      => RESET_IN,
      TIMER_START_IN => wait_timer_start,
      TIMER_DONE_OUT => wait_timer_done
      );

  adc_spi_sendbyte_1: adc_spi_sendbyte
    generic map (
      SPI_SPEED => SPI_SPEED
      )
    port map (
      CLK_IN             => CLK_IN,
      RESET_IN           => RESET_IN,
      START_IN           => sendbyte_seq_start,
      BYTE_IN            => sendbyte_byte,
      SEQUENCE_DONE_OUT  => sendbyte_done,
      SDIO_OUT           => sdio_sendbyte,
      SCLK_OUT           => sclk_sendbyte
      );

  adc_spi_readbyte_1: adc_spi_readbyte
    generic map (
```

```
    SPI_SPEED => SPI_SPEED
    )
  port map (
    CLK_IN            => CLK_IN,
    RESET_IN          => RESET_IN,
    START_IN          => readbyte_seq_start,
    BYTE_OUT          => readbyte_byte,
    SEQUENCE_DONE_OUT => readbyte_done,
    SDIO_IN           => sdio,
    SCLK_OUT          => sclk_readbyte
    );

-- Debug Line

DEBUG_OUT(0)           <= CLK_IN;
DEBUG_OUT(1)           <= sclk_o;
DEBUG_OUT(2)           <= SDIO_INOUT;
DEBUG_OUT(3)           <= csb_o;
DEBUG_OUT(4)           <= spi_busy;
DEBUG_OUT(5)           <= wait_timer_done;
DEBUG_OUT(6)           <= sendbyte_seq_start;
DEBUG_OUT(7)           <= sendbyte_done;
DEBUG_OUT(8)           <= sclk_sendbyte;
DEBUG_OUT(9)           <= sdio_sendbyte;
DEBUG_OUT(10)          <= sclk_readbyte;
DEBUG_OUT(11)          <= takeover_sdio;

-- Sync SPI SDIO Line
sdio_i <= SDIO_INOUT;

PROC_I2C_LINES_SYNC: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      sdio_x <= '1';
      sdio   <= '1';
    else
      sdio_x <= sdio_i;
      sdio   <= sdio_x;
    end if;
  end if;
end process PROC_I2C_LINES_SYNC;

PROC_I2C_MASTER_TRANSFER: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      spi_busy             <= '1';
      sendbyte_seq_start   <= '0';
      readbyte_seq_start   <= '0';
      sendbyte_byte        <= (others => '0');
      wait_timer_start     <= '0';
      reg_data             <= (others => '0');
      read_seq_ctr         <= '0';
      STATE                <= S_RESET;
    else
      spi_busy             <= spi_busy_x;
      sendbyte_seq_start   <= sendbyte_seq_start_x;
      readbyte_seq_start   <= readbyte_seq_start_x;
      sendbyte_byte        <= sendbyte_byte_x;
      wait_timer_start     <= wait_timer_start_x;
      reg_data             <= reg_data_x;
```



```
      read_seq_ctr         <= read_seq_ctr_x;
      STATE                <= NEXT_STATE;
    end if;
  end if;
end process PROC_I2C_MASTER_TRANSFER;


PROC_I2C_MASTER: process(STATE,
                         spi_start,
                         wait_timer_done,
                         sendbyte_done,
                         readbyte_done
                         )

begin
  -- Defaults
  takeover_sdio          <= '0';
  sclk_o                 <= '0';
  csb_o                  <= '0';
  spi_busy_x             <= '1';
  sendbyte_seq_start_x   <= '0';
  sendbyte_byte_x        <= (others => '0');
  readbyte_seq_start_x   <= '0';
  wait_timer_start_x     <= '0';
  reg_data_x             <= reg_data;
  read_seq_ctr_x         <= read_seq_ctr;

  case STATE is

    when S_RESET =>
      reg_data_x <= (others => '0');
      NEXT_STATE <= S_IDLE;

    when S_IDLE =>
      csb_o        <= '1';
      if (spi_start = '1') then
        reg_data_x <= x"8000_0000";  -- Set Running , clear all other bits
        NEXT_STATE <= S_START;
      else
        spi_busy_x     <= '0';
        reg_data_x     <= reg_data and x"7fff_ffff";  -- clear running bit;
        read_seq_ctr_x <= '0';
        NEXT_STATE     <= S_IDLE;
      end if;

      -- SPI START Sequence
    when S_START =>
      wait_timer_start_x <= '1';
      NEXT_STATE         <= S_START_WAIT;

    when S_START_WAIT =>
      if (wait_timer_done = '0') then
        NEXT_STATE <= S_START_WAIT;
      else
        takeover_sdio <= '1';
        NEXT_STATE    <= S_SEND_CMD_A;
      end if;

      -- I2C SEND CMD Part1
    when S_SEND_CMD_A =>
      takeover_sdio              <= '1';
      sendbyte_byte_x(7)         <= spi_rw_bit;
```

```
      sendbyte_byte_x(6 downto 5) <= "00";
      sendbyte_byte_x(4 downto 0) <= spi_registerid(12 downto 8);
      sendbyte_seq_start_x        <= '1';
      NEXT_STATE                  <= S_SEND_CMD_A_WAIT;

    when S_SEND_CMD_A_WAIT =>
      takeover_sdio <= '1';
      if (sendbyte_done = '0') then
        NEXT_STATE <= S_SEND_CMD_A_WAIT;
      else
        NEXT_STATE <= S_SEND_CMD_B;
      end if;

      -- I2C SEND CMD Part1
    when S_SEND_CMD_B =>
      takeover_sdio               <= '1';
      sendbyte_byte_x(7 downto 0) <= spi_registerid(7 downto 0);
      sendbyte_seq_start_x        <= '1';
      NEXT_STATE                  <= S_SEND_CMD_B_WAIT;

    when S_SEND_CMD_B_WAIT =>
      takeover_sdio <= '1';
      if (sendbyte_done = '0') then
        NEXT_STATE <= S_SEND_CMD_B_WAIT;
      else
        if (spi_rw_bit = '1') then
          NEXT_STATE        <= S_GET_DATA;
        else
          NEXT_STATE        <= S_SEND_DATA;
        end if;
      end if;

      -- I2C SEND DataWord
    when S_SEND_DATA =>
      takeover_sdio          <= '1';
      sendbyte_byte_x        <= spi_register_data;
      sendbyte_seq_start_x   <= '1';
      NEXT_STATE             <= S_SEND_DATA_WAIT;

    when S_SEND_DATA_WAIT =>
      takeover_sdio <= '1';
      if (sendbyte_done = '0') then
        NEXT_STATE <= S_SEND_DATA_WAIT;
      else
        NEXT_STATE <= S_STOP;
      end if;

      -- I2C GET DataWord
    when S_GET_DATA =>
      readbyte_seq_start_x   <= '1';
      NEXT_STATE             <= S_GET_DATA_WAIT;

    when S_GET_DATA_WAIT =>
      if (readbyte_done = '0') then
        NEXT_STATE <= S_GET_DATA_WAIT;
      else
        reg_data_x(7 downto 0) <= readbyte_byte;
        NEXT_STATE             <= S_STOP;
      end if;

      -- SPI STOP Sequence
    when S_STOP =>
```



```
      wait_timer_start_x    <= '1';
      NEXT_STATE            <= S_STOP_WAIT;

    when S_STOP_WAIT =>
      if (wait_timer_done = '0') then
        NEXT_STATE <= S_STOP_WAIT;
      else
        reg_data_x <= reg_data or x"4000_0000"; -- Set DONE Bit
        NEXT_STATE <= S_IDLE;
      end if;

  end case;
end process PROC_I2C_MASTER;

------------------------------------------------------------------------------
-- TRBNet Slave Bus
------------------------------------------------------------------------------
--
--   Write bit definition
--   ====================
--
--   D[31]    SPI_GO          0 => don't do anything on SPI,
--                            1 => start SPI access
--   D[30]    SPI_ACTION      0 => write byte, 1 => read byte
--   D[20:8]  SPI_CMD         SPI Register Id
--   D[7:0]   SPI_DATA        data to be written
--
--   Read bit definition
--   ===================
--
--   D[31]    RUNNING         whatever
--   D[30]    SPI_DONE        whatever
--   D[29:21] reserved        reserved
--   D[20:16] debug           subject to change, don't use
--   D[15:8]  reserved        reserved
--   D[7:0]   SPI_DATA        result of SPI read operation
--

PROC_SLAVE_BUS: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      slv_data_out_o     <= (others => '0');
      slv_no_more_data_o <= '0';
      slv_unknown_addr_o <= '0';
      slv_ack_o          <= '0';
      spi_start          <= '0';
      command_ack_o      <= '0';

      spi_chipid              <= (others => '0');
      spi_rw_bit              <= '0';
      spi_registerid          <= (others => '0');
      spi_register_data       <= (others => '0');
      spi_register_value_read <= (others => '0');

    else
      slv_data_out_o     <= (others => '0');
      slv_unknown_addr_o <= '0';
      slv_no_more_data_o <= '0';

      spi_start          <= '0';
      command_ack_o      <= '0';
```

```
       --if (spi_busy = '0' and INTERNAL_COMMAND_IN(31) = '1') then
       --  spi_rw_bit         <= INTERNAL_COMMAND_IN(30);
       --  spi_registerid     <= INTERNAL_COMMAND_IN(20 downto 8);
       --  spi_register_data  <= INTERNAL_COMMAND_IN(7 downto 0);
       --  spi_start          <= '1';
       --  command_ack_o      <= '1';
       --  slv_ack_o          <= '1';
       --
       --elsif (SLV_WRITE_IN  = '1') then
       if (SLV_WRITE_IN  = '1') then
         if (spi_busy = '0' and SLV_DATA_IN(31) = '1') then
           spi_rw_bit        <= SLV_DATA_IN(30);
           spi_registerid    <= SLV_DATA_IN(20 downto 8);
           spi_register_data <= SLV_DATA_IN(7 downto 0);
           spi_start         <= '1';
           slv_ack_o         <= '1';
         else
           slv_ack_o         <= '1';
         end if;

       elsif (SLV_READ_IN = '1') then
         if (spi_busy = '1') then
           slv_no_more_data_o <= '1';
           slv_ack_o          <= '0';
         else
           slv_data_out_o     <= reg_data;
           slv_ack_o          <= '1';
         end if;

       else
         slv_ack_o            <= '0';
       end if;

     end if;
   end if;
  end process PROC_SLAVE_BUS;


  ------------------------------------------------------------------------------
  -- Output Signals
  ------------------------------------------------------------------------------

  -- SPI Outputs
  SDIO_INOUT      <= sdio_sendbyte when (takeover_sdio = '1')
                     else 'Z';

  SCLK_OUT        <= sclk_o or
                     sclk_sendbyte or
                     sclk_readbyte;

  CSB_OUT         <= csb_o;
  COMMAND_ACK_OUT <= command_ack_o;

  -- Slave Bus
  SLV_DATA_OUT         <= slv_data_out_o;
  SLV_NO_MORE_DATA_OUT <= slv_no_more_data_o;
  SLV_UNKNOWN_ADDR_OUT <= slv_unknown_addr_o;
  SLV_ACK_OUT          <= slv_ack_o;

end Behavioral;
library ieee;
```



```
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.nxyter_components.all;


entity adc_spi_readbyte is
  generic (
    SPI_SPEED : unsigned(7 downto 0) := x"32"
    );
  port(
    CLK_IN               : in  std_logic;
    RESET_IN             : in  std_logic;

    START_IN             : in  std_logic;
    BYTE_OUT             : out std_logic_vector(7 downto 0);
    SEQUENCE_DONE_OUT    : out std_logic;

    -- SPI connections
    SDIO_IN              : in   std_logic;
    SCLK_OUT             : out std_logic
    );
end entity;

architecture Behavioral of adc_spi_readbyte is

  -- Send Byte
  signal sclk_o             : std_logic;
  signal spi_start          : std_logic;

  signal sequence_done_o    : std_logic;
  signal spi_byte           : unsigned(7 downto 0);
  signal bit_ctr            : unsigned(3 downto 0);
  signal spi_ack_o          : std_logic;
  signal wait_timer_start   : std_logic;

  signal sequence_done_o_x  : std_logic;
  signal spi_byte_x         : unsigned(7 downto 0);
  signal bit_ctr_x          : unsigned(3 downto 0);
  signal spi_ack_o_x        : std_logic;
  signal wait_timer_start_x : std_logic;

  type STATES is (S_IDLE,
                  S_UNSET_SCKL,
                  S_UNSET_SCKL_HOLD,
                  S_GET_BIT,
                  S_SET_SCKL,
                  S_NEXT_BIT,
                  S_DONE
                  );
  signal STATE, NEXT_STATE : STATES;

  -- Wait Timer
  signal wait_timer_done    : std_logic;

begin

  -- Timer
  timer_static_1: timer_static
    generic map(
      CTR_WIDTH => 8,
```

```
    CTR_END   => to_integer(SPI_SPEED srl 1)
    )
  port map (
    CLK_IN         => CLK_IN,
    RESET_IN       => RESET_IN,
    TIMER_START_IN => wait_timer_start,
    TIMER_DONE_OUT => wait_timer_done
    );

PROC_READ_BYTE_TRANSFER: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      sequence_done_o  <= '0';
      bit_ctr          <= (others => '0');
      spi_ack_o        <= '0';
      wait_timer_start <= '0';
      STATE            <= S_IDLE;
    else
      sequence_done_o  <= sequence_done_o_x;
      spi_byte         <= spi_byte_x;
      bit_ctr          <= bit_ctr_x;
      spi_ack_o        <= spi_ack_o_x;
      wait_timer_start <= wait_timer_start_x;
      STATE            <= NEXT_STATE;
    end if;
  end if;
end process PROC_READ_BYTE_TRANSFER;

PROC_READ_BYTE: process(STATE,
                        START_IN,
                        wait_timer_done,
                        bit_ctr
                        )
begin
  sclk_o             <= '0';
  sequence_done_o_x  <= '0';
  spi_byte_x         <= spi_byte;
  bit_ctr_x          <= bit_ctr;
  spi_ack_o_x        <= spi_ack_o;
  wait_timer_start_x <= '0';

  case STATE is
    when S_IDLE =>
      if (START_IN = '1') then
        spi_byte_x         <= (others => '0');
        bit_ctr_x          <= x"7";
        wait_timer_start_x <= '1';
        NEXT_STATE         <= S_UNSET_SCKL;
      else
        NEXT_STATE         <= S_IDLE;
      end if;

      -- SPI Read byte
    when S_UNSET_SCKL =>
      wait_timer_start_x   <= '1';
      NEXT_STATE           <= S_UNSET_SCKL_HOLD;

    when S_UNSET_SCKL_HOLD =>
      if (wait_timer_done = '0') then
        NEXT_STATE <= S_UNSET_SCKL_HOLD;
      else
```



```
        NEXT_STATE <= S_GET_BIT;
      end if;

    when S_GET_BIT =>
      spi_byte_x(0)        <= SDIO_IN;
      wait_timer_start_x   <= '1';
      NEXT_STATE           <= S_SET_SCKL;

    when S_SET_SCKL =>
      sclk_o  <= '1';
      if (wait_timer_done = '0') then
        NEXT_STATE <= S_SET_SCKL;
      else
        wait_timer_start_x <= '1';
        NEXT_STATE         <= S_NEXT_BIT;
      end if;

    when S_NEXT_BIT =>
      sclk_o  <= '1';
      if (bit_ctr > 0) then
        bit_ctr_x          <= bit_ctr - 1;
        spi_byte_x         <= spi_byte sll 1;
        wait_timer_start_x <= '1';
        NEXT_STATE         <= S_UNSET_SCKL;
      else
        NEXT_STATE         <= S_DONE;
      end if;

    when S_DONE =>
      sclk_o  <= '1';
      sequence_done_o_x <= '1';
      NEXT_STATE        <= S_IDLE;

  end case;
end process PROC_READ_BYTE;

-------------------------------------------------------------------------------
-- Output Signals
-------------------------------------------------------------------------------

SEQUENCE_DONE_OUT <= sequence_done_o;
BYTE_OUT          <= spi_byte;

-- I2c Outputs
SCLK_OUT <= sclk_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.nxyter_components.all;


entity adc_spi_sendbyte is
  generic (
    SPI_SPEED : unsigned(7 downto 0) := x"32"
    );
  port(
    CLK_IN               : in  std_logic;
    RESET_IN             : in  std_logic;
```

```
    START_IN             : in  std_logic;
    BYTE_IN              : in  std_logic_vector(7 downto 0);
    SEQUENCE_DONE_OUT    : out std_logic;

    -- SPI connections
    SCLK_OUT             : out std_logic;
    SDIO_OUT             : out std_logic
    );
end entity;

architecture Behavioral of adc_spi_sendbyte is

  -- Send Byte
  signal sclk_o             : std_logic;
  signal sdio_o             : std_logic;
  signal spi_start          : std_logic;

  signal sequence_done_o    : std_logic;
  signal spi_byte           : unsigned(7 downto 0);
  signal bit_ctr            : unsigned(3 downto 0);
  signal wait_timer_start   : std_logic;

  signal sequence_done_o_x  : std_logic;
  signal spi_byte_x         : unsigned(7 downto 0);
  signal bit_ctr_x          : unsigned(3 downto 0);
  signal wait_timer_start_x : std_logic;

  type STATES is (S_IDLE,
                  S_SET_SDIO,
                  S_SET_SCLK,
                  S_NEXT_BIT,
                  S_DONE
                  );
  signal STATE, NEXT_STATE : STATES;

  -- Wait Timer
  signal wait_timer_done    : std_logic;

begin

  -- Timer
  timer_static_1: timer_static
    generic map (
      CTR_WIDTH => 8,
      CTR_END   => to_integer(SPI_SPEED srl 1)
      )
    port map (
      CLK_IN         => CLK_IN,
      RESET_IN       => RESET_IN,
      TIMER_START_IN => wait_timer_start,
      TIMER_DONE_OUT => wait_timer_done
      );

  PROC_SEND_BYTE_TRANSFER: process(CLK_IN)
  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1' ) then
        sequence_done_o  <= '0';
        bit_ctr          <= (others => '0');
        wait_timer_start <= '0';
        STATE            <= S_IDLE;
```



```
      else
        sequence_done_o  <= sequence_done_o_x;
        spi_byte         <= spi_byte_x;
        bit_ctr          <= bit_ctr_x;
        wait_timer_start <= wait_timer_start_x;
        STATE            <= NEXT_STATE;
      end if;
    end if;
  end process PROC_SEND_BYTE_TRANSFER;

  PROC_SEND_BYTE: process(STATE,
                          START_IN,
                          wait_timer_done,
                          bit_ctr
                          )
  begin
    sdio_o             <= '0';
    sclk_o             <= '0';
    sequence_done_o_x  <= '0';
    spi_byte_x         <= spi_byte;
    bit_ctr_x          <= bit_ctr;
    wait_timer_start_x <= '0';

    case STATE is
      when S_IDLE =>
        if (START_IN = '1') then
          spi_byte_x         <= BYTE_IN;
          bit_ctr_x          <= x"7";
          wait_timer_start_x <= '1';
          NEXT_STATE         <= S_SET_SDIO;
        else
          NEXT_STATE <= S_IDLE;
        end if;

      when S_SET_SDIO =>
        sdio_o <= spi_byte(7);
        if (wait_timer_done = '0') then
          NEXT_STATE         <= S_SET_SDIO;
        else
          wait_timer_start_x <= '1';
          NEXT_STATE         <= S_SET_SCLK;
        end if;

      when S_SET_SCLK =>
        sdio_o <= spi_byte(7);
        sclk_o <= '1';
        if (wait_timer_done = '0') then
          NEXT_STATE         <= S_SET_SCLK;
        else
          NEXT_STATE         <= S_NEXT_BIT;
        end if;

      when S_NEXT_BIT =>
        sdio_o <= spi_byte(7);
        sclk_o <= '1';
        if (bit_ctr > 0) then
          bit_ctr_x          <= bit_ctr - 1;
          spi_byte_x         <= spi_byte sll 1;
          wait_timer_start_x <= '1';
          NEXT_STATE         <= S_SET_SDIO;
        else
          NEXT_STATE         <= S_DONE;
```

```
        end if;

      when S_DONE =>
        sdio_o <= spi_byte(7);
        sclk_o <= '1';
        sequence_done_o_x    <= '1';
        NEXT_STATE           <= S_IDLE;

    end case;
  end process PROC_SEND_BYTE;

  ------------------------------------------------------------------------------
  -- Output Signals
  ------------------------------------------------------------------------------

  SEQUENCE_DONE_OUT <= sequence_done_o;

  -- SPI Outputs
  SDIO_OUT <= sdio_o;
  SCLK_OUT <= sclk_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity bus_async_trans is
  generic (
    BUS_WIDTH : integer range 2 to 32   := 8;
    NUM_FF    : integer range 2 to 4    := 2
    );
  port (
    CLK_IN       : in  std_logic;
    RESET_IN     : in  std_logic;
    SIGNAL_A_IN  : in  std_logic_vector(BUS_WIDTH - 1 downto 0);
    SIGNAL_OUT   : out std_logic_vector(BUS_WIDTH - 1 downto 0)
    );

end entity;

architecture Behavioral of bus_async_trans is
  type buffer_t is array(0 to NUM_FF - 1) of
    std_logic_vector(BUS_WIDTH - 1 downto 0);
  signal signal_ff      : buffer_t;

begin

  ------------------------------------------------------------------------------
  -- Clock CLK_IN Domain
  ------------------------------------------------------------------------------

  PROC_SYNC_SIGNAL: process(CLK_IN)
  begin
    if( rising_edge(CLK_IN) ) then
      signal_ff(NUM_FF - 1)      <= SIGNAL_A_IN;
      if( RESET_IN = '1' ) then
        for i in NUM_FF - 2 downto 0 loop
          signal_ff(i)           <= (others => '0');
        end loop;
      else
        for i in NUM_FF - 2 downto 0 loop
          signal_ff(i)           <= signal_ff(i + 1);
```



```
        end loop;
      end if;
    end if;
  end process PROC_SYNC_SIGNAL;

  -- Output Signals
  SIGNAL_OUT      <= signal_ff(0);

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.nxyter_components.all;

entity debug_multiplexer is
  generic (
    NUM_PORTS : integer range 1 to 32 := 1
    );
  port(
    CLK_IN               : in  std_logic;
    RESET_IN             : in  std_logic;

    DEBUG_LINE_IN        : in  debug_array_t(0 to NUM_PORTS-1);
    DEBUG_LINE_OUT       : out std_logic_vector(15 downto 0);

    -- Slave bus
    SLV_READ_IN          : in  std_logic;
    SLV_WRITE_IN         : in  std_logic;
    SLV_DATA_OUT         : out std_logic_vector(31 downto 0);
    SLV_DATA_IN          : in  std_logic_vector(31 downto 0);
    SLV_ADDR_IN          : in  std_logic_vector(15 downto 0);
    SLV_ACK_OUT          : out std_logic;
    SLV_NO_MORE_DATA_OUT : out std_logic;
    SLV_UNKNOWN_ADDR_OUT : out std_logic

    );
end entity;

architecture Behavioral of debug_multiplexer is

  signal port_select        : std_logic_vector(7 downto 0);
  signal debug_line_o       : std_logic_vector(15 downto 0);

  signal slv_data_out_o     : std_logic_vector(31 downto 0);
  signal slv_no_more_data_o : std_logic;
  signal slv_unknown_addr_o : std_logic;
  signal slv_ack_o          : std_logic;

begin

  PROC_MULTIPLEXER: process(port_select,
                            DEBUG_LINE_IN)
  begin
    if (unsigned(port_select) < NUM_PORTS) then
      debug_line_o               <=
        DEBUG_LINE_IN(to_integer(unsigned(port_select)));
    elsif (unsigned(port_select) = NUM_PORTS) then
      -- Checkerboard
      for I in 0 to 7 loop
        debug_line_o(I * 2)      <= CLK_IN;
```

```
      debug_line_o(I * 2 + 1)  <= not CLK_IN;
    end loop;
  else
    debug_line_o               <= (others => '1');
  end if;
end process PROC_MULTIPLEXER;

PROC_SLAVE_BUS: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      slv_data_out_o      <= (others => '0');
      slv_no_more_data_o  <= '0';
      slv_unknown_addr_o  <= '0';
      slv_ack_o           <= '0';
      port_select         <= (others => '0');
    else
      slv_ack_o           <= '1';
      slv_unknown_addr_o  <= '0';
      slv_no_more_data_o  <= '0';
      slv_data_out_o      <= (others => '0');

      if (SLV_WRITE_IN  = '1') then
        case SLV_ADDR_IN is
          when x"0000" =>
            if (unsigned(SLV_DATA_IN(7 downto 0)) < NUM_PORTS + 1) then
              port_select               <= SLV_DATA_IN(7 downto 0);
            end if;
            slv_ack_o                   <= '1';

          when others =>
            slv_unknown_addr_o          <= '1';
            slv_ack_o                   <= '0';
        end case;

      elsif (SLV_READ_IN = '1') then
        case SLV_ADDR_IN is
          when x"0000" =>
            slv_data_out_o(7 downto 0)  <= port_select;
            slv_data_out_o(31 downto 8) <= (others => '0');

          when others =>
            slv_unknown_addr_o          <= '1';
            slv_ack_o                   <= '0';
        end case;

      else
        slv_ack_o                       <= '0';
      end if;
    end if;
  end if;
end process PROC_SLAVE_BUS;

-------------------------------------------------------------------------------
-- Output Signals
-------------------------------------------------------------------------------

SLV_DATA_OUT         <= slv_data_out_o;
SLV_NO_MORE_DATA_OUT <= slv_no_more_data_o;
SLV_UNKNOWN_ADDR_OUT <= slv_unknown_addr_o;
SLV_ACK_OUT          <= slv_ack_o;
```



```
  DEBUG_LINE_OUT       <= debug_line_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.nxyter_components.all;

entity fifo_44_data_delay_my is
    port (
        Data          : in   std_logic_vector(43 downto 0);
        Clock         : in   std_logic;
        WrEn          : in   std_logic;
        RdEn          : in   std_logic;
        Reset         : in   std_logic;
        AmEmptyThresh : in   std_logic_vector(7 downto 0);
        Q             : out  std_logic_vector(43 downto 0);
        Empty         : out  std_logic;
        Full          : out  std_logic;
        AlmostEmpty   : out  std_logic;
        DEBUG_OUT     : out std_logic_vector(15 downto 0)
        );
end entity;

architecture Behavioral of fifo_44_data_delay_my is
  constant BUS_WIDTH      : integer := 8;
  constant DATA_WIDTH     : integer := 44;
  constant FULL_LEVEL     : unsigned(BUS_WIDTH - 1 downto 0) := (others => '1');

  signal write_address    : std_logic_vector(BUS_WIDTH - 1 downto 0);
  signal write_data       : std_logic_vector(DATA_WIDTH - 1 downto 0);
  signal write_enable     : std_logic;
  signal write_ctr        : unsigned(BUS_WIDTH - 1 downto 0);
  signal write_ctr_x      : unsigned(BUS_WIDTH - 1 downto 0);
  signal full_o           : std_logic;

  signal read_address     : std_logic_vector(BUS_WIDTH - 1 downto 0);
  signal read_enable      : std_logic;
  signal read_enable_last : std_logic;
  signal read_ctr         : unsigned(BUS_WIDTH - 1 downto 0);
  signal read_ctr_x       : unsigned(BUS_WIDTH - 1 downto 0);
  signal read_data        : std_logic_vector(DATA_WIDTH - 1 downto 0);
  signal empty_o          : std_logic;
  signal empty_o_x        : std_logic;
  signal almost_empty_o   : std_logic;
  signal almost_empty_o_x : std_logic;

  signal Q_o              : std_logic_vector(DATA_WIDTH - 1 downto 0);
  signal Q_o_x            : std_logic_vector(DATA_WIDTH - 1 downto 0);

begin

  -------------------------------------------------------------------------------

  DEBUG_OUT(0)              <= Clock;
  DEBUG_OUT(1)              <= WrEn;
  DEBUG_OUT(2)              <= write_enable;
  DEBUG_OUT(3)              <= RdEn;
  DEBUG_OUT(4)              <= read_enable;
  DEBUG_OUT(5)              <= read_enable_last;
```

```
  DEBUG_OUT(6)              <= full_o;
  DEBUG_OUT(7)              <= empty_o;
  DEBUG_OUT(15 downto 8)    <= std_logic_vector(write_ctr - read_ctr);

  ------------------------------------------------------------------------------

  ram_fifo_delay_256x44_1: ram_fifo_delay_256x44
    port map (
      WrAddress => write_address,
      RdAddress => read_address,
      Data      => write_data,
      WE        => not Reset,
      RdClock   => Clock,
      RdClockEn => read_enable,
      Reset     => Reset,
      WrClock   => Clock,
      WrClockEn => write_enable,
      Q         => read_data
      );

  ------------------------------------------------------------------------------
  -- RAM Handler
  ------------------------------------------------------------------------------
  PROC_MEM_WRITE_TRANSFER: process(Clock)
  begin
    if( rising_edge(Clock) ) then
      if( Reset = '1' ) then
        write_ctr            <= (others => '0');
        read_ctr             <= (others => '0');
        read_enable_last     <= '0';
        Q_o                  <= (others => '0');
      else
        write_ctr            <= write_ctr_x;
        read_ctr             <= read_ctr_x;
        read_enable_last     <= read_enable;
        Q_o                  <= Q_o_x;
      end if;
    end if;
  end process PROC_MEM_WRITE_TRANSFER;

  PROC_MEM_WRITE: process(WrEn,
                          RdEn,
                          Data,
                          write_ctr,
                          read_ctr,
                          read_data,
                          read_enable_last,
                          full_o,
                          empty_o,
                          AmEmptyThresh
                          )
    variable delta_ctr      : unsigned(BUS_WIDTH - 1 downto 0);
    variable full           : std_logic;
    variable empty          : std_logic;
    variable almost_empty   : std_logic;

  begin

    -- Fill Level
    delta_ctr               := write_ctr - read_ctr;

    -- Empty
```



```
    if (delta_ctr = 0) then
      empty             := '1';
    else
      empty             := '0';
    end if;

    -- Almost Empty
    if (delta_ctr < unsigned(AmEmptyThresh)) then
      almost_empty      := '1';
    else
      almost_empty      := '0';
    end if;

    -- Full
    if (delta_ctr = FULL_LEVEL) then
      full              := '1';
    else
      full              := '0';
    end if;

    full_o         <= full;
    empty_o        <= empty;
    almost_empty_o <= almost_empty;

    -- FIFO Writes
    if (WrEn = '1' and full = '0') then
      write_address          <= write_ctr;
      write_data             <= Data;
      write_enable           <= '1';
      write_ctr_x            <= write_ctr + 1;
    else
      write_address          <= (others => '0');
      write_data             <= (others => '0');
      write_enable           <= '0';
      write_ctr_x            <= write_ctr;
    end if;

    -- FIFO Reads
    if (RdEn = '1' and empty = '0') then
      read_address           <= read_ctr;
      read_enable            <= '1';
      read_ctr_x             <= read_ctr + 1;
    else
      read_address           <= (others => '0');
      read_enable            <= '0';
      read_ctr_x             <= read_ctr;
    end if;

    if (read_enable_last = '1') then
      Q_o_x                  <= read_data;
    else
      Q_o_x                  <= (others => '0');
    end if;

  end process PROC_MEM_WRITE;

  ------------------------------------------------------------------------------
  -- Output Signals
  ------------------------------------------------------------------------------

  Q              <= Q_o;
  Empty          <= empty_o;
```

```
  Full          <= full_o;
  AlmostEmpty   <= almost_empty_o;

end Behavioral;
--------------------------------------------------------------------------------
--
-- Gray Decoder
--
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity gray_decoder is

  generic (
    WIDTH : integer range 2 to 32 := 12    -- Register Width
    );

  port (
    CLK_IN     : in std_logic;
    RESET_IN   : in std_logic;

    -- Input
    GRAY_IN    : in  std_logic_vector(WIDTH - 1 downto 0);

    -- OUTPUT
    BINARY_OUT : out std_logic_vector(WIDTH - 1 downto 0)
    );

end entity;


architecture Behavioral of gray_decoder is

  signal binary_o : std_logic_vector(WIDTH - 1 downto 0);

begin  -- Gray_Decoder

  PROC_DECODER: process (CLK_IN)
    variable b : std_logic_vector(WIDTH -1 downto 0) := (others => '0');
  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1' ) then
        b := (others => '0');
      else
        b(WIDTH - 1) := GRAY_IN(WIDTH - 1);

        for I in (WIDTH - 2) downto 0 loop
          b(I) := b(I + 1) xor GRAY_IN(I);
        end loop;
      end if;
    end if;
    binary_o <= b;
  end process PROC_DECODER;

-- Output
  BINARY_OUT <= binary_o;

end Behavioral;
--------------------------------------------------------------------------------
--
```



```
-- Gray EnCcoder
--
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity gray_encoder is
  generic (
    WIDTH : integer range 2 to 32 := 12   -- Register Width
    );

  port (
    CLK_IN      : in std_logic;
    RESET_IN    : in std_logic;

    -- Input
    BINARY_IN   : in  std_logic_vector(WIDTH - 1 downto 0);

    -- OUTPUT
    GRAY_OUT    : out std_logic_vector(WIDTH - 1 downto 0)
    );

end entity;

architecture Behavioral of  gray_encoder is

  signal gray_o : std_logic_vector(WIDTH - 1 downto 0);

begin

  PROC_ENCODER: process (CLK_IN)
  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1' ) then
        gray_o <= (others => '0');
      else
        gray_o(WIDTH - 1) <= BINARY_IN(WIDTH -1);
        for I in (WIDTH - 2) downto 0 loop
          gray_o(I) <= BINARY_IN(I + 1) xor BINARY_IN(I);
        end loop;
      end if;
    end if;

  end process PROC_ENCODER;

  -- Output
  GRAY_OUT <= gray_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity level_to_pulse is

  port (
    CLK_IN        : in std_logic;
    RESET_IN      : in std_logic;

    LEVEL_IN      : in std_logic;
    PULSE_OUT     : out std_logic
```

```
    );

end entity;

architecture Behavioral of level_to_pulse is

  type STATES is (IDLE,
                  WAIT_LOW
                );
  signal STATE, NEXT_STATE : STATES;

  signal pulse_o          : std_logic;

begin

  PROC_CONVERT_TRANSFER:process(CLK_IN)
  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1' ) then
        STATE     <= IDLE;
      else
        STATE     <= NEXT_STATE;
      end if;
    end if;
  end process PROC_CONVERT_TRANSFER;


  PROC_CONVERT: process(STATE,
                        LEVEL_IN
                        )
  begin

    case STATE is
      when IDLE =>
        if (LEVEL_IN = '1') then
          pulse_o     <= '1';
          NEXT_STATE  <= WAIT_LOW;
        else
          pulse_o     <= '0';
          NEXT_STATE  <= IDLE;
        end if;

      when WAIT_LOW =>
        pulse_o       <= '0';
        if (LEVEL_IN = '0') then
          NEXT_STATE  <= IDLE;
        else
          NEXT_STATE  <= WAIT_LOW;
        end if;

    end case;

  end process PROC_CONVERT;

  -- Output Signals
  PULSE_OUT    <= pulse_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```



```
library work;
use work.nxyter_components.all;

entity nx_control is
  port(
    CLK_IN                  : in  std_logic;
    RESET_IN                : in  std_logic;

    -- Monitor PLL Locks
    PLL_NX_CLK_LOCK_IN      : in std_logic;
    PLL_ADC_DCLK_LOCK_IN    : in std_logic;
    PLL_ADC_SCLK_LOCK_IN    : in std_logic;

    -- Signals
    I2C_SM_RESET_OUT        : out std_logic;
    I2C_REG_RESET_OUT       : out std_logic;
    NX_TS_RESET_OUT         : out std_logic;
    I2C_ONLINE_IN           : in  std_logic;
    OFFLINE_OUT             : out std_logic;

    -- Error
    ERROR_ALL_IN            : in  std_logic_vector(7 downto 0);

    -- Slave bus
    SLV_READ_IN             : in  std_logic;
    SLV_WRITE_IN            : in  std_logic;
    SLV_DATA_OUT            : out std_logic_vector(31 downto 0);
    SLV_DATA_IN             : in  std_logic_vector(31 downto 0);
    SLV_ADDR_IN             : in  std_logic_vector(15 downto 0);
    SLV_ACK_OUT             : out std_logic;
    SLV_NO_MORE_DATA_OUT    : out std_logic;
    SLV_UNKNOWN_ADDR_OUT    : out std_logic;

    DEBUG_OUT               : out std_logic_vector(15 downto 0)
    );
end entity;

architecture Behavioral of nx_control is

  -- Offline Handler
  signal offline_force_internal   : std_logic;
  signal offline_force            : std_logic;
  signal offline_o                : std_logic;
  signal offline_on               : std_logic;
  signal online_on                : std_logic;
  signal offline_last             : std_logic;

  -- I2C Reset
  signal i2c_sm_reset_start       : std_logic;
  signal i2c_reg_reset_start      : std_logic;
  signal nx_ts_reset_start        : std_logic;

  signal i2c_sm_reset_o           : std_logic;
  signal i2c_reg_reset_o          : std_logic;
  signal nx_ts_reset_o            : std_logic;


  type STATES is (S_IDLE,
                  S_I2C_SM_RESET,
                  S_I2C_SM_RESET_WAIT,
                  S_I2C_REG_RESET,
                  S_I2C_REG_RESET_WAIT,
```

```
                  S_NX_TS_RESET,
                  S_NX_TS_RESET_WAIT
                  );

  signal STATE : STATES;

  -- Wait Timer
  signal wait_timer_start         : std_logic;
  signal wait_timer_init          : unsigned(7 downto 0);
  signal wait_timer_done          : std_logic;

  -- PLL Locks
  signal pll_nx_clk_lock          : std_logic;
  signal pll_adc_dclk_lock        : std_logic;
  signal pll_adc_sclk_lock        : std_logic;

  signal pll_nx_clk_notlock       : std_logic;
  signal pll_adc_dclk_notlock     : std_logic;
  signal pll_adc_sclk_notlock     : std_logic;

  signal pll_nx_clk_notlock_ctr   : unsigned(15 downto 0);
  signal pll_adc_dclk_notlock_ctr : unsigned(15 downto 0);
  signal pll_adc_sclk_notlock_ctr : unsigned(15 downto 0);

  signal clear_notlock_counters   : std_logic;

  -- Nxyter Data Clock
  signal nx_data_clk_dphase_o     : std_logic_vector(3 downto 0);
  signal nx_data_clk_finedelb_o   : std_logic_vector(3 downto 0);

  -- Slave Bus
  signal slv_data_out_o           : std_logic_vector(31 downto 0);
  signal slv_no_more_data_o       : std_logic;
  signal slv_unknown_addr_o       : std_logic;
  signal slv_ack_o                : std_logic;

begin

  DEBUG_OUT(0)            <= CLK_IN;
  DEBUG_OUT(1)            <= i2c_sm_reset_o;
  DEBUG_OUT(2)            <= i2c_reg_reset_o;
  DEBUG_OUT(3)            <= nx_ts_reset_o;
  DEBUG_OUT(4)            <= PLL_NX_CLK_LOCK_IN;
  DEBUG_OUT(5)            <= pll_nx_clk_lock;
  DEBUG_OUT(6)            <= PLL_ADC_DCLK_LOCK_IN;
  DEBUG_OUT(7)            <= pll_adc_dclk_lock;
  DEBUG_OUT(8)            <= PLL_ADC_SCLK_LOCK_IN;
  DEBUG_OUT(9)            <= pll_adc_sclk_lock;


  DEBUG_OUT(10)           <= I2C_ONLINE_IN;
  DEBUG_OUT(11)           <= offline_force;
  DEBUG_OUT(12)           <= offline_force_internal;
  DEBUG_OUT(13)           <= offline_o;
  DEBUG_OUT(14)           <= online_on;
  DEBUG_OUT(15)           <= '0';

  timer_1: timer
    generic map (
      CTR_WIDTH => 8
      )
    port map (
```



```
    CLK_IN          => CLK_IN,
    RESET_IN        => RESET_IN,
    TIMER_START_IN  => wait_timer_start,
    TIMER_END_IN    => wait_timer_init,
    TIMER_DONE_OUT  => wait_timer_done
    );

------------------------------------------------------------------------------
-- Offline Handler
------------------------------------------------------------------------------

offline_force_internal <= '0';

PROC_NXYTER_OFFLINE: process(CLK_IN)
  variable offline_state : std_logic_vector(1 downto 0) := "00";
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      offline_on         <= '0';
      online_on          <= '0';
      offline_o          <= '1';
      offline_last       <= '0';
    else
      if (offline_force = '1' or offline_force_internal = '1') then
        offline_o        <= '1';
      else
        offline_o        <= not I2C_ONLINE_IN;
      end if;

      -- Offline State changes
      offline_on         <= '0';
      online_on          <= '0';
      offline_last       <= offline_o;
      offline_state      := offline_o & offline_last;

      case offline_state is
        when  "01" =>
          offline_on     <= '1';

        when  "10" =>
          online_on      <= '0';

        when others => null;
      end case;
    end if;
  end if;
end process PROC_NXYTER_OFFLINE;

------------------------------------------------------------------------------
-- I2C SM Reset
------------------------------------------------------------------------------

PROC_I2C_SM_RESET: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      wait_timer_start   <= '0';
      i2c_sm_reset_o     <= '0';
      i2c_reg_reset_o    <= '0';
      nx_ts_reset_o      <= '0';
      STATE              <= S_IDLE;
    else
```

```
      i2c_sm_reset_o     <= '0';
      i2c_reg_reset_o    <= '0';
      nx_ts_reset_o      <= '0';
      wait_timer_start   <= '0';

      case STATE is
        when S_IDLE =>
          if (i2c_sm_reset_start = '1') then
            STATE          <= S_I2C_SM_RESET;
          elsif (i2c_reg_reset_start = '1') then
            STATE          <= S_I2C_REG_RESET;
          elsif (nx_ts_reset_start = '1') then
            STATE          <= S_NX_TS_RESET;
          else
            STATE          <= S_IDLE;
          end if;

        when S_I2C_SM_RESET =>
          i2c_sm_reset_o   <= '1';
          wait_timer_init  <= x"8f";
          wait_timer_start <= '1';
          STATE            <= S_I2C_SM_RESET_WAIT;

        when S_I2C_SM_RESET_WAIT =>
          i2c_sm_reset_o   <= '1';
          if (wait_timer_done = '0') then
            STATE          <= S_I2C_SM_RESET_WAIT;
          else
            STATE          <= S_IDLE;
          end if;

        when S_I2C_REG_RESET =>
          i2c_reg_reset_o  <= '1';
          wait_timer_init  <= x"8f";
          wait_timer_start <= '1';
          STATE            <= S_I2C_REG_RESET_WAIT;

        when S_I2C_REG_RESET_WAIT =>
          i2c_reg_reset_o  <= '1';
          if (wait_timer_done = '0') then
            STATE          <= S_I2C_REG_RESET_WAIT;
          else
            STATE          <= S_IDLE;
          end if;

        when S_NX_TS_RESET =>
          nx_ts_reset_o    <= '1';
          wait_timer_init  <= x"01";
          wait_timer_start <= '1';
          STATE            <= S_NX_TS_RESET_WAIT;

        when S_NX_TS_RESET_WAIT =>
          nx_ts_reset_o    <= '1';
          if (wait_timer_done = '0') then
            STATE          <= S_NX_TS_RESET_WAIT;
          else
            STATE          <= S_IDLE;
          end if;

      end case;
    end if;
  end if;
```



```
end process PROC_I2C_SM_RESET;

-------------------------------------------------------------------------------
-- PLL Not Lock Counters
-------------------------------------------------------------------------------

signal_async_trans_1: signal_async_trans
  port map (
    CLK_IN      => CLK_IN,
    SIGNAL_A_IN => PLL_NX_CLK_LOCK_IN,
    SIGNAL_OUT  => pll_nx_clk_lock
    );

signal_async_trans_2: signal_async_trans
  port map (
    CLK_IN      => CLK_IN,
    SIGNAL_A_IN => PLL_ADC_DCLK_LOCK_IN,
    SIGNAL_OUT  => pll_adc_dclk_lock
    );

signal_async_trans_3: signal_async_trans
  port map (
    CLK_IN      => CLK_IN,
    SIGNAL_A_IN => PLL_ADC_SCLK_LOCK_IN,
    SIGNAL_OUT  => pll_adc_sclk_lock
    );

level_to_pulse_1: level_to_pulse
  port map (
    CLK_IN    => CLK_IN,
    RESET_IN  => RESET_IN,
    LEVEL_IN  => not pll_nx_clk_lock,
    PULSE_OUT => pll_nx_clk_notlock
    );

level_to_pulse_2: level_to_pulse
  port map (
    CLK_IN    => CLK_IN,
    RESET_IN  => RESET_IN,
    LEVEL_IN  => not pll_adc_dclk_lock,
    PULSE_OUT => pll_adc_dclk_notlock
    );

level_to_pulse_3: level_to_pulse
  port map (
    CLK_IN    => CLK_IN,
    RESET_IN  => RESET_IN,
    LEVEL_IN  => not pll_adc_sclk_lock,
    PULSE_OUT => pll_adc_sclk_notlock
    );

PROC_PLL_UNLOCK_COUNTERS: process (CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1') then
      pll_nx_clk_notlock_ctr     <= (others => '0');
      pll_adc_dclk_notlock_ctr   <= (others => '0');
      pll_adc_sclk_notlock_ctr   <= (others => '0');
    else
      if (clear_notlock_counters = '1') then
        pll_nx_clk_notlock_ctr      <= (others => '0');
```

```
        pll_adc_dclk_notlock_ctr    <= (others => '0');
        pll_adc_sclk_notlock_ctr    <= (others => '0');
      else
        if (pll_nx_clk_notlock = '1') then
          pll_nx_clk_notlock_ctr    <= pll_nx_clk_notlock_ctr + 1;
        end if;

        if (pll_adc_dclk_notlock = '1') then
          pll_adc_dclk_notlock_ctr  <= pll_adc_dclk_notlock_ctr + 1;
        end if;

        if (pll_adc_sclk_notlock = '1') then
          pll_adc_sclk_notlock_ctr  <= pll_adc_sclk_notlock_ctr + 1;
        end if;
      end if;
    end if;
  end if;
end process PROC_PLL_UNLOCK_COUNTERS;

--------------------------------------------------------------------------------
-- Slave Bus
--------------------------------------------------------------------------------

PROC_NX_REGISTERS: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      slv_data_out_o             <= (others => '0');
      slv_no_more_data_o         <= '0';
      slv_unknown_addr_o         <= '0';
      slv_ack_o                  <= '0';
      i2c_sm_reset_start         <= '0';
      i2c_reg_reset_start        <= '0';
      nx_ts_reset_start          <= '0';
      offline_force              <= '0';
      nx_data_clk_dphase_o       <= x"7";
      nx_data_clk_finedelb_o     <= x"0";
      clear_notlock_counters     <= '0';
    else
      slv_unknown_addr_o         <= '0';
      slv_no_more_data_o         <= '0';
      slv_data_out_o             <= (others => '0');
      i2c_sm_reset_start         <= '0';
      i2c_reg_reset_start        <= '0';
      nx_ts_reset_start          <= '0';
      clear_notlock_counters     <= '0';

      if (SLV_WRITE_IN  = '1') then
        case SLV_ADDR_IN is
          when x"0000" =>
            i2c_sm_reset_start          <= '1';
            slv_ack_o                   <= '1';

          when x"0001" =>
            i2c_reg_reset_start         <= '1';
            slv_ack_o                   <= '1';

          when x"0002" =>
            nx_ts_reset_start           <= '1';
            slv_ack_o                   <= '1';

          when x"0003" =>
```



```
            offline_force               <= SLV_DATA_IN(0);
            slv_ack_o                   <= '1';

          when x"000a" =>
            clear_notlock_counters      <= '1';
            slv_ack_o                   <= '1';

          when others =>
            slv_unknown_addr_o          <= '1';
            slv_ack_o                   <= '0';
        end case;

      elsif (SLV_READ_IN = '1') then
        case SLV_ADDR_IN is
          when x"0003" =>
            slv_data_out_o(0)           <= offline_force;
            slv_data_out_o(31 downto 1) <= (others => '0');
            slv_ack_o                   <= '1';

          when x"0004" =>
            slv_data_out_o(0)           <= I2C_ONLINE_IN;
            slv_data_out_o(31 downto 1) <= (others => '0');
            slv_ack_o                   <= '1';

          when x"0005" =>
            slv_data_out_o(0)           <= offline_o;
            slv_data_out_o(31 downto 1) <= (others => '0');
            slv_ack_o                   <= '1';

          when x"0006" =>
            slv_data_out_o(0)           <= pll_nx_clk_lock;
            slv_data_out_o(31 downto 1) <= (others => '0');
            slv_ack_o                   <= '1';

          when x"0007" =>
            slv_data_out_o(0)           <= pll_adc_dclk_lock;
            slv_data_out_o(31 downto 1) <= (others => '0');
            slv_ack_o                   <= '1';

          when x"0008" =>
            slv_data_out_o(0)           <= pll_adc_sclk_lock;
            slv_data_out_o(31 downto 1) <= (others => '0');
            slv_ack_o                   <= '1';

          when x"0009" =>
            slv_data_out_o(15 downto 0) <= pll_nx_clk_notlock_ctr;
            slv_data_out_o(31 downto 6) <= (others => '0');
            slv_ack_o                   <= '1';

          when x"000a" =>
            slv_data_out_o(15 downto 0) <= pll_adc_dclk_notlock_ctr;
            slv_data_out_o(31 downto 6) <= (others => '0');
            slv_ack_o                   <= '1';

          when x"000b" =>
            slv_data_out_o(15 downto 0) <= pll_adc_sclk_notlock_ctr;
            slv_data_out_o(31 downto 6) <= (others => '0');
            slv_ack_o                   <= '1';

          when x"000c" =>
            slv_data_out_o(7 downto 0)  <= ERROR_ALL_IN;
            slv_data_out_o(31 downto 8) <= (others => '0');
```

```
            slv_ack_o                   <= '1';

          when others =>
            slv_unknown_addr_o          <= '1';
            slv_ack_o                   <= '0';
        end case;

      else
        slv_ack_o                       <= '0';
      end if;
    end if;
  end if;
end process PROC_NX_REGISTERS;

-- Output Signals
SLV_DATA_OUT         <= slv_data_out_o;
SLV_NO_MORE_DATA_OUT <= slv_no_more_data_o;
SLV_UNKNOWN_ADDR_OUT <= slv_unknown_addr_o;
SLV_ACK_OUT          <= slv_ack_o;

I2C_SM_RESET_OUT     <= i2c_sm_reset_o;
I2C_REG_RESET_OUT    <= i2c_reg_reset_o;
NX_TS_RESET_OUT      <= nx_ts_reset_o;
OFFLINE_OUT          <= offline_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.trb_net_std.all;
use work.trb_net_components.all;
use work.trb3_components.all;
use work.nxyter_components.all;

entity nx_data_delay is
  port(
    CLK_IN                  : in  std_logic;
    RESET_IN                : in  std_logic;

    -- Signals
    NX_FRAME_IN             : in  std_logic_vector(31 downto 0);
    ADC_DATA_IN             : in  std_logic_vector(11 downto 0);
    NEW_DATA_IN             : in  std_logic;

    NX_FRAME_OUT            : out std_logic_vector(31 downto 0);
    ADC_DATA_OUT            : out std_logic_vector(11 downto 0);
    NEW_DATA_OUT            : out std_logic;

    FIFO_DELAY_IN           : in  std_logic_vector(7 downto 0);

    -- Slave bus
    SLV_READ_IN             : in  std_logic;
    SLV_WRITE_IN            : in  std_logic;
    SLV_DATA_OUT            : out std_logic_vector(31 downto 0);
    SLV_DATA_IN             : in  std_logic_vector(31 downto 0);
    SLV_ADDR_IN             : in  std_logic_vector(15 downto 0);
    SLV_ACK_OUT             : out std_logic;
    SLV_NO_MORE_DATA_OUT    : out std_logic;
    SLV_UNKNOWN_ADDR_OUT    : out std_logic;
```



```
    DEBUG_OUT               : out std_logic_vector(15 downto 0)
    );
end entity;

architecture Behavioral of nx_data_delay is

  -- FIFO Write Handler
  signal fifo_data_in           : std_logic_vector(43 downto 0);
  signal fifo_full              : std_logic;
  signal fifo_write_enable      : std_logic;
  signal fifo_reset             : std_logic;

  -- FIFO READ
  signal fifo_data_out          : std_logic_vector(43 downto 0);
  signal fifo_read_enable       : std_logic;
  signal fifo_empty             : std_logic;
  signal fifo_almost_empty      : std_logic;

  signal fifo_read_enable_t     : std_logic;
  signal fifo_read_enable_tt    : std_logic;
  signal nx_frame_o             : std_logic_vector(31 downto 0);
  signal adc_data_o             : std_logic_vector(11 downto 0);
  signal new_data_o             : std_logic;

  -- Fifo Delay
  signal fifo_delay             : std_logic_vector(7 downto 0);
  signal fifo_delay_reset       : std_logic;

  -- Slave Bus
  signal slv_data_o             : std_logic_vector(31 downto 0);
  signal slv_no_more_data_o     : std_logic;
  signal slv_unknown_addr_o     : std_logic;
  signal slv_ack_o              : std_logic;
  signal fifo_reset_r           : std_logic;
  signal debug_r                : std_logic;

  -- Misc
  signal fifo_select            : std_logic;
  signal debug_fifo             : std_logic_vector(15 downto 0);

begin

  -- Debug
  PROC_DEBUG_MULTIPLEXER: process(debug_r)
  begin
    if (debug_r = '0') then
    DEBUG_OUT(0)            <= CLK_IN;
    DEBUG_OUT(1)            <= fifo_reset;
    DEBUG_OUT(2)            <= fifo_full;
    DEBUG_OUT(3)            <= fifo_write_enable;
    DEBUG_OUT(4)            <= fifo_empty;
    DEBUG_OUT(5)            <= fifo_almost_empty;
    DEBUG_OUT(6)            <= fifo_read_enable;
    DEBUG_OUT(7)            <= fifo_read_enable_t;
    DEBUG_OUT(8)            <= fifo_read_enable_tt;
    DEBUG_OUT(9)            <= new_data_o;
    DEBUG_OUT(12 downto 10) <= NX_FRAME_IN(11 downto 9);
    DEBUG_OUT(15 downto 13) <= nx_frame_o(11 downto 9);
    --DEBUG_OUT(15 downto 13) <= fifo_data_out(11 downto 9);
    else
      DEBUG_OUT             <= debug_fifo;
    end if;
```

```
  end process PROC_DEBUG_MULTIPLEXER;

  ------------------------------------------------------------------------------
  -- FIFO Delay Handler
  ------------------------------------------------------------------------------

  fifo_44_data_delay_my_1: fifo_44_data_delay_my
    port map (
      Data          => fifo_data_in,
      Clock         => CLK_IN,
      WrEn          => fifo_write_enable,
      RdEn          => fifo_read_enable,  --LOOP??
      Reset         => fifo_reset,
      AmEmptyThresh => fifo_delay,
      Q             => fifo_data_out,      --fifo_data_out_0,
      Empty         => fifo_empty,         -- fifo_empty_0,
      Full          => fifo_full,          --fifo_full_0,
      AlmostEmpty   => fifo_almost_empty,  -- fifo_almost_empty_0,
      DEBUG_OUT     => debug_fifo
      );

  fifo_read_enable             <= not fifo_almost_empty;
  fifo_reset                   <= RESET_IN or fifo_reset_r or fifo_delay_reset;
  fifo_data_in(31 downto 0)    <= NX_FRAME_IN;
  fifo_data_in(43 downto 32)   <= ADC_DATA_IN;
  fifo_write_enable            <= NEW_DATA_IN and not fifo_full;

  -- FIFO Read Handler
  PROC_FIFO_READ: process(CLK_IN)
  begin
    if (rising_edge(CLK_IN)) then
      if (RESET_IN = '1' or fifo_delay_reset = '1') then
        fifo_read_enable_t   <= '0';
        fifo_read_enable_tt  <= '0';

        nx_frame_o           <= (others => '0');
        adc_data_o           <= (others => '0');
        new_data_o           <= '0';
      else
        -- Read enable
        fifo_read_enable_t   <= fifo_read_enable;
        fifo_read_enable_tt  <= fifo_read_enable_t;

        if (fifo_read_enable_tt = '1') then
          nx_frame_o         <= fifo_data_out(31 downto 0);
          adc_data_o         <= fifo_data_out(43 downto 32);
          new_data_o         <= '1';
        else
          nx_frame_o         <= x"ffff_ffff";
          adc_data_o         <= x"fff";
          new_data_o         <= '0';
        end if;
      end if;
    end if;
  end process PROC_FIFO_READ;

  PROC_FIFO_DELAY: process(CLK_IN)
  begin
    if( rising_edge(CLK_IN) ) then
      if (RESET_IN = '1') then
        fifo_delay             <= x"02";
        fifo_delay_reset       <= '0';
```



```
      else
        fifo_delay_reset       <= '0';
        if ((FIFO_DELAY_IN /= fifo_delay) and
            (unsigned(FIFO_DELAY_IN) >= 2)          and
            (unsigned(FIFO_DELAY_IN) <= 250)
            ) then
            fifo_delay         <= FIFO_DELAY_IN;
            fifo_delay_reset   <= '1';
        else
          fifo_delay_reset     <= '0';
        end if;
      end if;
    end if;
  end process PROC_FIFO_DELAY;

  ------------------------------------------------------------------------------
  -- TRBNet Slave Bus
  ------------------------------------------------------------------------------

  -- Give status info to the TRB Slow Control Channel
  PROC_FIFO_REGISTERS: process(CLK_IN)
  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1' ) then
        slv_data_o             <= (others => '0');
        slv_ack_o              <= '0';
        slv_unknown_addr_o     <= '0';
        slv_no_more_data_o     <= '0';
        fifo_reset_r           <= '0';
        debug_r                <= '0';
        fifo_select            <= '0';
      else
        slv_data_o             <= (others => '0');
        slv_unknown_addr_o     <= '0';
        slv_no_more_data_o     <= '0';
        fifo_reset_r           <= '0';

        if (SLV_READ_IN  = '1') then
          case SLV_ADDR_IN is
            when x"0000" =>
              slv_data_o( 7 downto 0)  <= fifo_delay;
              slv_data_o(31 downto 8)  <= (others => '0');
              slv_ack_o                <= '1';

            when x"0001" =>
              slv_data_o(0)            <= debug_r;
              slv_data_o(31 downto 1)  <= (others => '0');
              slv_ack_o                <= '1';

            when x"0002" =>
              slv_data_o(0)            <= fifo_select;
              slv_data_o(31 downto 1)  <= (others => '0');
              slv_ack_o                <= '1';

            when others =>
              slv_unknown_addr_o       <= '1';
              slv_ack_o                <= '0';
          end case;

        elsif (SLV_WRITE_IN  = '1') then
          case SLV_ADDR_IN is
            when x"0000" =>
```

```
              fifo_reset_r             <= '1';
              slv_ack_o                <= '1';

            when x"0001" =>
              debug_r                  <= SLV_DATA_IN(0);
              slv_ack_o                <= '1';

            when x"0002" =>
              fifo_select              <= SLV_DATA_IN(0);
              slv_ack_o                <= '1';

            when others =>
              slv_unknown_addr_o       <= '1';
              slv_ack_o                <= '0';

          end case;
        else
          slv_ack_o                    <= '0';
        end if;
      end if;
    end if;
  end process PROC_FIFO_REGISTERS;

  -- Output Signals
  NX_FRAME_OUT          <= nx_frame_o;
  ADC_DATA_OUT          <= adc_data_o;
  NEW_DATA_OUT          <= new_data_o;

  SLV_DATA_OUT          <= slv_data_o;
  SLV_NO_MORE_DATA_OUT  <= slv_no_more_data_o;
  SLV_UNKNOWN_ADDR_OUT  <= slv_unknown_addr_o;
  SLV_ACK_OUT           <= slv_ack_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.trb_net_std.all;
use work.trb_net_components.all;
use work.nxyter_components.all;

entity nx_data_receiver is
  port(
    CLK_IN               : in  std_logic;
    RESET_IN             : in  std_logic;
    NX_DATA_CLK_TEST_IN  : in std_logic;
    TRIGGER_IN           : in  std_logic;

    -- nXyter Ports
    NX_TIMESTAMP_CLK_IN  : in  std_logic;
    NX_TIMESTAMP_IN      : in  std_logic_vector (7 downto 0);

    -- ADC Ports
    ADC_CLK_DAT_IN       : in  std_logic;
    ADC_FCLK_IN          : in  std_logic_vector(1 downto 0);
    ADC_DCLK_IN          : in  std_logic_vector(1 downto 0);
    ADC_SAMPLE_CLK_OUT   : out std_logic;
    ADC_A_IN             : in  std_logic_vector(1 downto 0);
    ADC_B_IN             : in  std_logic_vector(1 downto 0);
    ADC_NX_IN            : in  std_logic_vector(1 downto 0);
```



```
    ADC_D_IN             : in  std_logic_vector(1 downto 0);
    ADC_SCLK_LOCK_OUT    : out std_logic;

    -- Outputs
    NX_TIMESTAMP_OUT     : out std_logic_vector(31 downto 0);
    ADC_DATA_OUT         : out std_logic_vector(11 downto 0);
    NEW_DATA_OUT         : out std_logic;

    -- Slave bus
    SLV_READ_IN          : in  std_logic;
    SLV_WRITE_IN         : in  std_logic;
    SLV_DATA_OUT         : out std_logic_vector(31 downto 0);
    SLV_DATA_IN          : in  std_logic_vector(31 downto 0);
    SLV_ADDR_IN          : in  std_logic_vector(15 downto 0);
    SLV_ACK_OUT          : out std_logic;
    SLV_NO_MORE_DATA_OUT : out std_logic;
    SLV_UNKNOWN_ADDR_OUT : out std_logic;

    ERROR_OUT            : out std_logic;
    DEBUG_OUT            : out std_logic_vector(15 downto 0)
    );
end entity;

architecture Behavioral of nx_data_receiver is

  -- Clock Check
  signal counter_nx_domain          : unsigned(7 downto 0);
  signal counter_nx_ref_domain      : unsigned(7 downto 0);
  signal counter_nx_diff            : unsigned(7 downto 0);

  ------------------------------------------------------------------------------
  -- NX_TIMESTAMP_CLK Domain
  ------------------------------------------------------------------------------

  -- FIFO DC Input Handler
  signal nx_fifo_write_enable       : std_logic;
  signal nx_timestamp_fff           : std_logic_vector(7 downto 0);
  signal nx_timestamp_ff            : std_logic_vector(7 downto 0);
  signal nx_fifo_full               : std_logic;
  signal nx_fifo_delay              : unsigned(3 downto 0);
  signal nx_fifo_reset              : std_logic;

  -- NX_TIMESTAMP_IN Process
  signal frame_byte_ctr             : unsigned(1 downto 0);
  signal nx_frame_word              : std_logic_vector(31 downto 0);
  signal nx_new_frame               : std_logic;

  -- Frame Sync Process
  signal frame_byte_pos             : unsigned(1 downto 0);

  -- RS Sync FlipFlop
  signal nx_frame_synced            : std_logic;
  signal rs_sync_set                : std_logic;
  signal rs_sync_reset              : std_logic;

  -- Parity Check
  signal parity_error               : std_logic;

  -- NX Clock Active
  signal nx_clk_active_ff_0         : std_logic;
  signal nx_clk_active_ff_1         : std_logic;
  signal nx_clk_active_ff_2         : std_logic;
```

```
-- ADC Ckl Generator
signal adc_clk_skip               : std_logic;
signal adc_sampling_clk           : std_logic;
signal johnson_ff_0               : std_logic;
signal johnson_ff_1               : std_logic;
signal johnson_counter_sync       : std_logic_vector(1 downto 0);
signal adc_clk_ok                 : std_logic;

signal pll_adc_sampling_clk_o     : std_logic;
signal pll_adc_sampling_clk_lock  : std_logic;
signal pll_adc_sampling_clk_reset : std_logic;

-- PLL ADC Monitor
signal pll_adc_not_lock           : std_logic;
signal pll_adc_not_lock_f         : std_logic_vector(1 downto 0);
signal pll_adc_not_lock_ctr       : unsigned(11 downto 0);
signal pll_adc_not_lock_ctr_clear : std_logic;

-- ADC RESET
signal adc_clk_ok_last            : std_logic;
signal adc_reset_sync_s           : std_logic;
signal adc_reset_sync             : std_logic;
signal adc_reset_ctr              : unsigned(11 downto 0);

-- Reset Handler
signal startup_reset              : std_logic;
signal rs_wait_timer_start        : std_logic;
signal rs_wait_timer_done         : std_logic;

signal rs_timeout_timer_start     : std_logic;
signal rs_timeout_timer_done      : std_logic;
signal rs_timeout_timer_reset     : std_logic;

type R_STATES is (R_IDLE,
                  R_SET_ALL_RESETS,
                  R_WAIT_1,
                  R_WAIT_NX_FRAME_RATE_OK,
                  R_PLL_WAIT_LOCK,
                  R_WAIT_ADC_OK,
                  R_WAIT_DATA_HANDLER_OK
                  );
signal R_STATE : R_STATES;

signal frame_rates_reset          : std_logic;
signal sampling_clk_reset         : std_logic;
signal adc_reset                  : std_logic;
signal adc_reset_p                : std_logic;
signal output_handler_reset       : std_logic;

signal reset_handler_counter      : unsigned(15 downto 0);
signal reset_handler_busy         : std_logic;
signal reset_timeout_flag         : std_logic;

------------------------------------------------------------------------------
-- CLK_IN Domain
------------------------------------------------------------------------------

-- NX FIFO READ ENABLE
signal nx_fifo_read_enable        : std_logic;
signal nx_fifo_empty              : std_logic;
signal nx_read_enable             : std_logic;
```



```
signal nx_fifo_data_valid_tt      : std_logic;
signal nx_fifo_data_valid_t       : std_logic;
signal nx_fifo_data_valid         : std_logic;

signal nx_fifo_data               : std_logic_vector(31 downto 0);

-- NX FIFO READ
type delay_array_t is array(0 to 15) of std_logic_vector(31 downto 0);
signal nx_timestamp_d             : delay_array_t;
signal nx_timestamp_t             : std_logic_vector(31 downto 0);
signal nx_new_timestamp           : std_logic;
signal nx_new_timestamp_ctr       : unsigned(3 downto 0);
signal nx_fifo_data_f             : std_logic_vector(31 downto 0);

-- Resync Counter Process
signal resync_counter             : unsigned(11 downto 0);
signal resync_ctr_inc             : std_logic;
signal nx_clk_active              : std_logic;

-- Parity Error Counter Process
signal parity_error_counter       : unsigned(11 downto 0);
signal parity_error_ctr_inc       : std_logic;

signal reg_nx_frame_synced        : std_logic;

------------------------------------------------------------------------------
-- ADC Data Handler
------------------------------------------------------------------------------

-- ADC Handler
signal ADC_RESET_AD9228           : std_logic;
signal adc_data                   : std_logic_vector(11 downto 0);
signal test_adc_data              : std_logic_vector(11 downto 0);
signal adc_data_valid             : std_logic;

signal adc_data_t                 : std_logic_vector(11 downto 0);
signal adc_new_data               : std_logic;
signal adc_new_data_ctr           : unsigned(3 downto 0);
signal adc_notlock_ctr            : unsigned(7 downto 0);
signal ADC_DEBUG                  : std_logic_vector(15 downto 0);
signal adc_debug_type             : std_logic_vector(3 downto 0);

-- Data Output Handler
type STATES is (IDLE,
                WAIT_ADC,
                WAIT_TIMESTAMP
                );
signal STATE : STATES;
signal STATE_d                    : std_logic_vector(1 downto 0);

signal nx_timestamp_o             : std_logic_vector(31 downto 0);
signal adc_data_o                 : std_logic_vector(11 downto 0);
signal new_data_o                 : std_logic;

-- Check Nxyter Data Clock via Johnson Counter
signal nx_data_clock_test_0       : std_logic;
signal nx_data_clock_test_1       : std_logic;
signal nx_data_clock              : std_logic;
signal nx_data_clock_state        : std_logic_vector(3 downto 0);
signal nx_data_clock_ok           : std_logic;

signal pll_adc_sample_clk_dphase  : std_logic_vector(3 downto 0);
```

```
  signal pll_adc_sample_clk_finedelb : std_logic_vector(3 downto 0);

  -- Rate Calculations
  signal nx_frame_rate_ctr           : unsigned(27 downto 0);
  signal nx_frame_rate               : unsigned(27 downto 0);
  signal adc_frame_rate_ctr          : unsigned(27 downto 0);
  signal adc_frame_rate              : unsigned(27 downto 0);
  signal frame_rate_ctr              : unsigned(27 downto 0);
  signal frame_rate                  : unsigned(27 downto 0);
  signal parity_rate_ctr             : unsigned(27 downto 0);
  signal parity_rate                 : unsigned(27 downto 0);
  signal rate_timer_ctr              : unsigned(27 downto 0);

  -- Error
  signal error_adc0                  : std_logic;
  signal error_adc1                  : std_logic;
  signal error_o                     : std_logic;
  signal error_status_bits           : std_logic_vector(11 downto 0);

  -- Rate Errors
  signal nx_frame_rate_offline_last  : std_logic;
  signal nx_frame_rate_offline       : std_logic;
  signal nx_frame_rate_error         : std_logic;
  signal adc_frame_rate_error        : std_logic;
  signal frame_rate_error            : std_logic;
  signal parity_rate_error           : std_logic;
  signal reset_for_offline           : std_logic;

  -- Slave Bus
  signal slv_data_out_o              : std_logic_vector(31 downto 0);
  signal slv_no_more_data_o          : std_logic;

  signal slv_unknown_addr_o          : std_logic;
  signal slv_ack_o                   : std_logic;

  signal reset_resync_ctr            : std_logic;
  signal reset_parity_error_ctr      : std_logic;
  signal debug_adc                   : std_logic_vector(1 downto 0);
  signal reset_handler_start_r       : std_logic;
  signal reset_handler_counter_clear : std_logic;
  signal adc_bit_shift               : unsigned(3 downto 0);
  signal johnson_counter_sync_r      : unsigned(1 downto 0);
  signal pll_adc_sample_clk_dphase_r : unsigned(3 downto 0);

  -- Reset Domain Transfers
  signal RESET_NX_TIMESTAMP_CLK_IN   : std_logic;
  signal RESET_NX_DATA_CLK_TEST_IN   : std_logic;

  signal debug_state                 : std_logic_vector(3 downto 0);

begin

  PROC_DEBUG_MULT: process(debug_adc,
                           adc_data,
                           adc_data_valid,
                           test_adc_data,
                           adc_clk_ok,
                           adc_clk_ok_last,
                           adc_clk_skip,
                           adc_reset_sync,
                           adc_reset_sync_s,
                           ADC_RESET_AD9228,
```



```
                           nx_new_frame,
                           adc_reset_ctr,
                           nx_fifo_full,
                           nx_fifo_write_enable,
                           nx_fifo_empty,
                           nx_fifo_read_enable,
                           nx_fifo_data_valid,
                           nx_new_timestamp,
                           adc_new_data,
                           STATE_d,
                           new_data_o,
                           nx_frame_synced,
                           rs_sync_reset
                           )
begin
  case debug_adc is
    when "00" =>
      -- Default
      DEBUG_OUT(0)            <= CLK_IN;
      DEBUG_OUT(1)            <= TRIGGER_IN;
      DEBUG_OUT(2)            <= nx_fifo_full;
      DEBUG_OUT(3)            <= nx_fifo_write_enable;
      DEBUG_OUT(4)            <= nx_fifo_empty;
      DEBUG_OUT(5)            <= nx_fifo_empty;
      DEBUG_OUT(6)            <= nx_fifo_read_enable;
      DEBUG_OUT(7)            <= nx_fifo_data_valid;
      DEBUG_OUT(8)            <= adc_data_valid;
      DEBUG_OUT(9)            <= nx_new_timestamp;
      DEBUG_OUT(10)           <= adc_new_data;
      DEBUG_OUT(11)           <= nx_fifo_reset;
      DEBUG_OUT(12)           <= '0';
      DEBUG_OUT(13)           <= nx_new_frame;
      DEBUG_OUT(14)           <= new_data_o;
      DEBUG_OUT(15)           <= nx_frame_synced;

    when "01" =>
      -- Reset Handler
      DEBUG_OUT(0)            <= CLK_IN;
      DEBUG_OUT(1)            <= nx_new_frame;
      DEBUG_OUT(2)            <= adc_clk_skip;
      DEBUG_OUT(3)            <= adc_clk_ok;
      DEBUG_OUT(4)            <= adc_reset_sync;
      DEBUG_OUT(5)            <= adc_reset;
      DEBUG_OUT(6)            <= ADC_RESET_AD9228;
      DEBUG_OUT(7)            <= pll_adc_not_lock;
      DEBUG_OUT(8)            <= reset_for_offline;
      DEBUG_OUT(9)            <= nx_fifo_reset;
      DEBUG_OUT(10)           <= reset_handler_busy;
      DEBUG_OUT(11)           <= sampling_clk_reset;
      DEBUG_OUT(15 downto 12) <= debug_state;

    when "10" =>
      -- AD9228 Handler Debug output
      DEBUG_OUT               <= ADC_DEBUG;

    when "11" =>
      -- Test Channel
      DEBUG_OUT(0)            <= CLK_IN;
      DEBUG_OUT(3 downto 1)   <= debug_state(2 downto 0);
      DEBUG_OUT(4)            <= reset_handler_busy;
      DEBUG_OUT(5)            <= nx_frame_rate_offline;
      DEBUG_OUT(6)            <= nx_frame_rate_error;
```

```
      DEBUG_OUT(7)            <= pll_adc_not_lock;
      DEBUG_OUT(8)            <= error_adc0;
      DEBUG_OUT(9)            <= adc_frame_rate_error;
      DEBUG_OUT(10)           <= nx_fifo_reset;
      DEBUG_OUT(11)           <= sampling_clk_reset;
      DEBUG_OUT(12)           <= adc_reset;
      DEBUG_OUT(13)           <= output_handler_reset;
      DEBUG_OUT(14)           <= frame_rate_error;
      DEBUG_OUT(15)           <= reset_timeout_flag;
  end case;

end process PROC_DEBUG_MULT;

-------------------------------------------------------------------------------
-- Reset Domain Transfer
-------------------------------------------------------------------------------
signal_async_trans_RESET_IN: signal_async_trans
  port map (
    CLK_IN      => NX_TIMESTAMP_CLK_IN,
    SIGNAL_A_IN => RESET_IN,
    SIGNAL_OUT  => RESET_NX_TIMESTAMP_CLK_IN
  );

signal_async_trans_RESET_IN_2: signal_async_trans
  port map (
    CLK_IN      => NX_DATA_CLK_TEST_IN,
    SIGNAL_A_IN => RESET_IN,
    SIGNAL_OUT  => RESET_NX_DATA_CLK_TEST_IN
  );

-------------------------------------------------------------------------------
-- Check NX Data Clk
-------------------------------------------------------------------------------
PROC_COUNTER_NX_CLOCK: process(NX_TIMESTAMP_CLK_IN)
begin
  if (rising_edge(NX_TIMESTAMP_CLK_IN) ) then
    if( RESET_NX_TIMESTAMP_CLK_IN = '1' ) then
      counter_nx_domain <= (others => '0');
    else
      counter_nx_domain <= counter_nx_domain + 1;
    end if;
  end if;
end process PROC_COUNTER_NX_CLOCK;

PROC_COUNTER_NX_REF_CLOCK: process(NX_DATA_CLK_TEST_IN)
begin
  if (rising_edge(NX_DATA_CLK_TEST_IN) ) then
    if (NX_DATA_CLK_TEST_IN = '1') then
      counter_nx_ref_domain <= (others => '0');
    else
      counter_nx_ref_domain <= counter_nx_ref_domain + 1;
    end if;
  end if;
end process PROC_COUNTER_NX_REF_CLOCK;

counter_nx_diff <= counter_nx_ref_domain - counter_nx_domain;

-------------------------------------------------------------------------------
-- ADC CLK DOMAIN
-------------------------------------------------------------------------------

pll_adc_sampling_clk_reset  <=  sampling_clk_reset;
```



```
-- Shift dphase show 0 as optimal value
pll_adc_sample_clk_dphase   <=
  std_logic_vector(pll_adc_sample_clk_dphase_r - 1);

pll_adc_sampling_clk_2: pll_adc_sampling_clk
  port map (
    CLK       => adc_sampling_clk,

    RESET     => pll_adc_sampling_clk_reset,
    FINEDELB0 => pll_adc_sample_clk_finedelb(0),
    FINEDELB1 => pll_adc_sample_clk_finedelb(1),
    FINEDELB2 => pll_adc_sample_clk_finedelb(2),
    FINEDELB3 => pll_adc_sample_clk_finedelb(3),
    DPHASE0   => pll_adc_sample_clk_dphase(0),
    DPHASE1   => pll_adc_sample_clk_dphase(1),
    DPHASE2   => pll_adc_sample_clk_dphase(2),
    DPHASE3   => pll_adc_sample_clk_dphase(3),
    CLKOP     => open,
    CLKOS     => pll_adc_sampling_clk_o,
    LOCK      => pll_adc_sampling_clk_lock
    );

signal_async_trans_2: signal_async_trans
  port map (
    CLK_IN      => CLK_IN,
    SIGNAL_A_IN => not pll_adc_sampling_clk_lock,
    SIGNAL_OUT  => pll_adc_not_lock
    );

PROC_PLL_LOCK_COUNTER: process(CLK_IN)
begin
  if (rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' or pll_adc_not_lock_ctr_clear = '1') then
      pll_adc_not_lock_f     <= (others => '0');
      pll_adc_not_lock_ctr   <= (others => '0');
    else
      pll_adc_not_lock_f(0)  <= pll_adc_not_lock;
      pll_adc_not_lock_f(1)  <= pll_adc_not_lock_f(0);

      if (pll_adc_not_lock_f = "01") then
        pll_adc_not_lock_ctr <= pll_adc_not_lock_ctr + 1;
      end if;
    end if;
  end if;
end process PROC_PLL_LOCK_COUNTER;

ADC_RESET_AD9228   <= RESET_IN or adc_reset;
adc_ad9228_1: adc_ad9228
  port map (
    CLK_IN              => CLK_IN,
    RESET_IN            => ADC_RESET_AD9228,
    CLK_ADCDAT_IN       => ADC_CLK_DAT_IN,

    ADC0_SCLK_IN        => pll_adc_sampling_clk_o,
    ADC0_SCLK_OUT       => ADC_SAMPLE_CLK_OUT,
    ADC0_DATA_A_IN      => ADC_NX_IN(0),
    ADC0_DATA_B_IN      => ADC_B_IN(0),
    ADC0_DATA_C_IN      => ADC_A_IN(0),
    ADC0_DATA_D_IN      => ADC_D_IN(0),
    ADC0_DCLK_IN        => ADC_DCLK_IN(0),
    ADC0_FCLK_IN        => ADC_FCLK_IN(0),
```

```
    ADC1_SCLK_IN         => pll_adc_sampling_clk_o,
    ADC1_SCLK_OUT        => open,
    ADC1_DATA_A_IN       => ADC_NX_IN(1),
    ADC1_DATA_B_IN       => ADC_A_IN(1),
    ADC1_DATA_C_IN       => ADC_B_IN(1),
    ADC1_DATA_D_IN       => ADC_D_IN(1),
    ADC1_DCLK_IN         => ADC_DCLK_IN(1),
    ADC1_FCLK_IN         => ADC_FCLK_IN(1),

    ADC0_DATA_A_OUT      => adc_data,
    ADC0_DATA_B_OUT      => test_adc_data,
    ADC0_DATA_C_OUT      => open,
    ADC0_DATA_D_OUT      => open,
    ADC0_DATA_VALID_OUT  => adc_data_valid,

    ADC1_DATA_A_OUT      => open,
    ADC1_DATA_B_OUT      => open,
    ADC1_DATA_C_OUT      => open,
    ADC1_DATA_D_OUT      => open,
    ADC1_DATA_VALID_OUT  => open,

    ADC0_NOTLOCK_COUNTER => adc_notlock_ctr,
    ADC1_NOTLOCK_COUNTER => open,

    ERROR_ADC0_OUT       => error_adc0,
    ERROR_ADC1_OUT       => error_adc1,
    DEBUG_IN             => adc_debug_type,
    DEBUG_OUT            => ADC_DEBUG
    );

timer_static_RESET_TIMER: timer_static
  generic map (
    CTR_WIDTH => 20,
    CTR_END   => 500000 -- 1ms
    )
  port map (
    CLK_IN         => CLK_IN,
    RESET_IN       => RESET_IN,
    TIMER_START_IN => rs_wait_timer_start,
    TIMER_DONE_OUT => rs_wait_timer_done
    );

timer_static_RESET_TIMEOUT: timer_static
  generic map (
    CTR_WIDTH => 26,
    CTR_END   => 10000000 -- 1s
    )
  port map (
    CLK_IN         => CLK_IN,
    RESET_IN       => rs_timeout_timer_reset,
    TIMER_START_IN => rs_timeout_timer_start,
    TIMER_DONE_OUT => rs_timeout_timer_done
    );

pulse_dtrans_1: pulse_dtrans
  generic map (
    CLK_RATIO => 4
    )
  port map (
    CLK_A_IN    => NX_TIMESTAMP_CLK_IN,
    RESET_A_IN  => RESET_NX_TIMESTAMP_CLK_IN,
```



```
    PULSE_A_IN  => adc_reset_sync_s,
    CLK_B_IN    => CLK_IN,
    RESET_B_IN  => RESET_IN,
    PULSE_B_OUT => adc_reset_sync
    );

PROC_RESET_HANDLER: process(CLK_IN)
begin
  if (rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      frame_rates_reset           <= '0';
      nx_fifo_reset               <= '0';
      sampling_clk_reset          <= '0';
      adc_reset_p                 <= '0';
      adc_reset                   <= '0';
      output_handler_reset        <= '0';

      rs_wait_timer_start         <= '0';
      rs_timeout_timer_start      <= '0';
      rs_timeout_timer_reset      <= '1';
      reset_handler_counter       <= (others => '0');
      reset_handler_busy          <= '0';
      reset_timeout_flag          <= '0';
      startup_reset               <= '1';
      R_STATE                     <= R_IDLE;
    else
      frame_rates_reset           <= '0';
      nx_fifo_reset               <= '0';
      sampling_clk_reset          <= '0';
      adc_reset_p                 <= '0';
      adc_reset                   <= '0';
      output_handler_reset        <= '0';

      rs_wait_timer_start         <= '0';
      rs_timeout_timer_start      <= '0';
      rs_timeout_timer_reset      <= '0';
      reset_handler_busy          <= '1';

      debug_state   <= x"0";

      if (reset_handler_counter_clear = '1') then
        reset_handler_counter       <= (others => '0');
      end if;

      if (rs_timeout_timer_done = '1') then
        -- Reset Timeout
        reset_timeout_flag          <= '1';
      end if;

      case R_STATE is
        when R_IDLE =>
          if (reset_for_offline     = '1' or
              --pll_adc_not_lock      = '1' or
              --adc_reset_sync        = '1' or
              reset_handler_start_r = '1' or
              startup_reset         = '1'
              ) then
            if (reset_handler_counter_clear = '0') then
              reset_handler_counter <= reset_handler_counter + 1;
            end if;
            R_STATE                 <= R_SET_ALL_RESETS;
          else
```

```
            reset_handler_busy      <= '0';
            R_STATE                 <= R_IDLE;
          end if;

        when R_SET_ALL_RESETS =>
          frame_rates_reset         <= '1';
          nx_fifo_reset             <= '1';
          sampling_clk_reset        <= '1';
          adc_reset_p               <= '1';
          adc_reset                 <= '1';
          output_handler_reset      <= '1';

          rs_wait_timer_start       <= '1';  -- wait 1mue to settle
          R_STATE                   <= R_WAIT_1;
          debug_state               <= x"1";

        when R_WAIT_1 =>
          if (rs_wait_timer_done = '0') then
            nx_fifo_reset           <= '1';
            sampling_clk_reset      <= '1';
            adc_reset               <= '1';
            output_handler_reset    <= '1';
            R_STATE                 <= R_WAIT_1;
          else
            -- Release NX Fifo Reset + Start Timeout HAndler
            sampling_clk_reset      <= '1';
            adc_reset               <= '1';
            output_handler_reset    <= '1';
            reset_timeout_flag      <= '0';
            rs_timeout_timer_start  <= '1';
            R_STATE                 <= R_WAIT_NX_FRAME_RATE_OK;
          end if;
          debug_state               <= x"2";

        when R_WAIT_NX_FRAME_RATE_OK =>
          if (nx_frame_rate_offline = '0' and
              nx_frame_rate_error   = '0') then
            -- Release PLL Reset
            adc_reset               <= '1';
            output_handler_reset    <= '1';
            R_STATE                 <= R_PLL_WAIT_LOCK;
          else
            sampling_clk_reset      <= '1';
            adc_reset               <= '1';
            output_handler_reset    <= '1';
            R_STATE                 <= R_WAIT_NX_FRAME_RATE_OK;
          end if;
          debug_state               <= x"3";

        when R_PLL_WAIT_LOCK =>
          if (pll_adc_not_lock = '1') then
            adc_reset               <= '1';
            output_handler_reset    <= '1';
            R_STATE                 <= R_PLL_WAIT_LOCK;
          else
            -- Release ADC Reset
            output_handler_reset    <= '1';
            R_STATE                 <= R_WAIT_ADC_OK;
          end if;
          debug_state               <= x"4";

        when R_WAIT_ADC_OK =>
```



```
          if (error_adc0 = '0' and
              adc_frame_rate_error = '0') then
            -- Release Output Handler Reset
            R_STATE                 <= R_WAIT_DATA_HANDLER_OK;
          else
            R_STATE                 <= R_WAIT_ADC_OK;
          end if;
          debug_state               <= x"5";

        when R_WAIT_DATA_HANDLER_OK =>
          if (frame_rate_error = '0') then
            startup_reset           <= '0';
            R_STATE                 <= R_IDLE;
          else
            R_STATE                 <= R_WAIT_DATA_HANDLER_OK;
          end if;
          debug_state               <= x"6";

      end case;
    end if;
  end if;
end process PROC_RESET_HANDLER;

-------------------------------------------------------------------------------
-- NX_TIMESTAMP_CLK_IN Domain
-------------------------------------------------------------------------------

-- Merge TS Data 8bit to 32Bit Timestamp Frame
PROC_8_TO_32_BIT: process(NX_TIMESTAMP_CLK_IN)
begin
  if (rising_edge(NX_TIMESTAMP_CLK_IN) ) then
    if( RESET_NX_TIMESTAMP_CLK_IN = '1' ) then
      frame_byte_ctr    <= (others => '0');
      nx_frame_word     <= (others => '0');
      nx_timestamp_ff   <= (others => '0');
      nx_new_frame      <= '0';
    else
      nx_timestamp_fff  <= NX_TIMESTAMP_IN;
      nx_timestamp_ff   <= nx_timestamp_fff;
      nx_new_frame      <= '0';

      case frame_byte_pos is
        when "11" => nx_frame_word(31 downto 24) <= nx_timestamp_ff;
                     frame_byte_ctr              <= frame_byte_ctr + 1;

        when "10" => nx_frame_word(23 downto 16) <= nx_timestamp_ff;
                     frame_byte_ctr              <= frame_byte_ctr + 1;

        when "01" => nx_frame_word(15 downto  8) <= nx_timestamp_ff;
                     frame_byte_ctr              <= frame_byte_ctr + 1;

        when "00" => nx_frame_word( 7 downto  0) <= nx_timestamp_ff;
                     if (frame_byte_ctr = "11") then
                       nx_new_frame              <= '1';
                     end if;
                     frame_byte_ctr              <= (others => '0');
      end case;
    end if;
  end if;
end process PROC_8_TO_32_BIT;

-- Frame Sync process
```

```
  PROC_SYNC_TO_NX_FRAME: process(NX_TIMESTAMP_CLK_IN)
  begin
    if (rising_edge(NX_TIMESTAMP_CLK_IN) ) then
      if( RESET_NX_TIMESTAMP_CLK_IN = '1' ) then
        frame_byte_pos    <= "11";
        rs_sync_set       <= '0';
        rs_sync_reset     <= '0';
      else
        rs_sync_set       <= '0';
        rs_sync_reset     <= '0';
        if (nx_new_frame = '1') then
          case nx_frame_word is
            when x"7f7f7f06" =>
              rs_sync_set         <= '1';
              frame_byte_pos      <= frame_byte_pos - 1;

            when x"7f7f067f" =>
              rs_sync_reset       <= '1';
              frame_byte_pos      <= frame_byte_pos - 2;

            when x"7f067f7f" =>
              rs_sync_reset       <= '1';
              frame_byte_pos      <= frame_byte_pos - 3;

            when x"067f7f7f" =>
              rs_sync_reset       <= '1';
              frame_byte_pos      <= frame_byte_pos - 4;

            when others =>
              frame_byte_pos      <= frame_byte_pos - 1;
          end case;
        else
          frame_byte_pos          <= frame_byte_pos - 1;
        end if;
      end if;
    end if;
  end process PROC_SYNC_TO_NX_FRAME;

  -- RS FlipFlop to hold Sync Status
  PROC_RS_FRAME_SYNCED: process(NX_TIMESTAMP_CLK_IN)
  begin
    if (rising_edge(NX_TIMESTAMP_CLK_IN) ) then
      if (RESET_NX_TIMESTAMP_CLK_IN = '1' or rs_sync_reset = '1') then
        nx_frame_synced <= '0';
      elsif (rs_sync_set = '1') then
        nx_frame_synced <= '1';
      end if;
    end if;
  end process PROC_RS_FRAME_SYNCED;

  -- Check Parity
  PROC_PARITY_CHECK: process(NX_TIMESTAMP_CLK_IN)
    variable parity_bits : std_logic_vector(22 downto 0);
    variable parity      : std_logic;
  begin
    if (rising_edge(NX_TIMESTAMP_CLK_IN) ) then
      if (RESET_NX_TIMESTAMP_CLK_IN = '1') then
        parity_error       <= '0';
      else
        parity_error       <= '0';
        if (nx_new_frame = '1' and nx_frame_synced = '1') then
          -- Timestamp Bit #6 is excluded (funny nxyter-bug)
```



```
          parity_bits         := nx_frame_word(31)           &
                                 nx_frame_word(30 downto 24) &
                                 nx_frame_word(21 downto 16) &
                                 nx_frame_word(14 downto  8) &
                                 nx_frame_word( 2 downto  1);
          parity              := xor_all(parity_bits);

          if (parity /= nx_frame_word(0)) then
            parity_error   <= '1';
          end if;
        end if;
      end if;
    end if;
  end process PROC_PARITY_CHECK;

  fifo_ts_32to32_dc_1: fifo_ts_32to32_dc
    port map (
      Data          => nx_frame_word,
      WrClock       => NX_TIMESTAMP_CLK_IN,
      RdClock       => CLK_IN,
      WrEn          => nx_fifo_write_enable,
      RdEn          => nx_fifo_read_enable,
      Reset         => nx_fifo_reset,
      RPReset       => nx_fifo_reset,
      Q             => nx_fifo_data_f,
      Empty         => nx_fifo_empty,
      Full          => nx_fifo_full
      );

  nx_fifo_write_enable   <= nx_new_frame and not nx_fifo_full;

  PROC_NX_CLK_ACT: process(NX_TIMESTAMP_CLK_IN)
  begin
    if (rising_edge(NX_TIMESTAMP_CLK_IN)) then
      if(RESET_NX_TIMESTAMP_CLK_IN = '1' ) then
        nx_clk_active_ff_0 <= '0';
        nx_clk_active_ff_1 <= '0';
        nx_clk_active_ff_2 <= '0';
      else
        nx_clk_active_ff_0 <= not nx_clk_active_ff_2;
        nx_clk_active_ff_1 <= nx_clk_active_ff_0;
        nx_clk_active_ff_2 <= nx_clk_active_ff_1;
      end if;
    end if;
  end process PROC_NX_CLK_ACT;

  -- ADC Sampling Clock Generator using a Johnson Counter
  PROC_ADC_SAMPLING_CLK_GENERATOR: process(NX_TIMESTAMP_CLK_IN)
  begin
    if (rising_edge(NX_TIMESTAMP_CLK_IN)) then
      if (RESET_NX_TIMESTAMP_CLK_IN = '1') then
        johnson_ff_0   <= '0';
        johnson_ff_1   <= '0';
      else
        if (adc_clk_skip = '0') then
          johnson_ff_0     <= not johnson_ff_1;
          johnson_ff_1     <= johnson_ff_0;
          adc_sampling_clk <= not johnson_ff_1;
        end if;
      end if;
    end if;
    adc_sampling_clk <= johnson_ff_0;
```

```
end process PROC_ADC_SAMPLING_CLK_GENERATOR;

-- Adjust johnson_counter_sync to show optimal value at 0
johnson_counter_sync <= std_logic_vector(johnson_counter_sync_r + 3);
PROC_ADC_SAMPLING_CLK_SYNC: process(NX_TIMESTAMP_CLK_IN)
  variable adc_clk_state : std_logic_vector(1 downto 0);
begin
  if (rising_edge(NX_TIMESTAMP_CLK_IN)) then
    if (RESET_NX_TIMESTAMP_CLK_IN = '1') then
      adc_clk_skip       <= '0';
      adc_clk_ok         <= '0';
    else
      adc_clk_state      := johnson_ff_1 & johnson_ff_0;
      adc_clk_skip       <= '0';
      if (nx_new_frame = '1') then
        if (adc_clk_state /= johnson_counter_sync) then
          adc_clk_skip   <= '1';
          adc_clk_ok     <= '0';
        else
          adc_clk_ok     <= '1';
        end if;
      end if;
    end if;
  end if;
end process PROC_ADC_SAMPLING_CLK_SYNC;

PROC_ADC_RESET: process(NX_TIMESTAMP_CLK_IN)
begin
  if (rising_edge(NX_TIMESTAMP_CLK_IN)) then
    if (RESET_NX_TIMESTAMP_CLK_IN = '1') then
      adc_clk_ok_last     <= '0';
      adc_reset_sync_s    <= '0';
    else
      adc_reset_sync_s    <= '0';
      adc_clk_ok_last     <= adc_clk_ok;
      if (adc_clk_ok_last = '0' and adc_clk_ok = '1') then
        adc_reset_sync_s  <= '1';
      end if;
    end if;
  end if;
end process PROC_ADC_RESET;

PROC_RESET_CTR: process(CLK_IN)
begin
  if (rising_edge(CLK_IN)) then
    if (RESET_IN = '1') then
      adc_reset_ctr        <= (others => '0');
    else
      if (adc_reset_p = '1') then
        adc_reset_ctr      <= adc_reset_ctr + 1;
      end if;
    end if;
  end if;
end process PROC_RESET_CTR;

--------------------------------------------------------------------------------
-- NX CLK_IN Domain
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
-- Gray Decode Timestamp Frame (Timestamp and Channel Id)
--------------------------------------------------------------------------------
```



```
gray_decoder_TIMESTAMP: gray_decoder          -- Decode nx_timestamp
  generic map (
    WIDTH => 14
    )
  port map (
    CLK_IN                 => CLK_IN,
    RESET_IN               => RESET_IN,
    GRAY_IN(13 downto 7)   => not nx_fifo_data_f(30 downto 24),
    GRAY_IN( 6 downto 0)   => not nx_fifo_data_f(22 downto 16),
    BINARY_OUT(13 downto 7) => nx_fifo_data(30 downto 24),
    BINARY_OUT(6 downto 0)  => nx_fifo_data(22 downto 16)
    );

gray_decoder_CHANNEL_ID: gray_decoder          -- Decode Channel_ID
  generic map (
    WIDTH => 7
    )
  port map (
    CLK_IN     => CLK_IN,
    RESET_IN   => RESET_IN,
    GRAY_IN    => nx_fifo_data_f(14 downto 8),
    BINARY_OUT => nx_fifo_data(14 downto 8)
    );

-- Leave other bits untouched
PROC_GRAY_DECODE: process(CLK_IN)
begin
  if (rising_edge(CLK_IN) ) then
    nx_fifo_data(31)          <= nx_fifo_data_f(31);
    nx_fifo_data(23)          <= nx_fifo_data_f(23);
    nx_fifo_data(15)          <= nx_fifo_data_f(15);
    nx_fifo_data(7 downto 0)  <= nx_fifo_data_f(7 downto 0);
  end if;
end process PROC_GRAY_DECODE;

--------------------------------------------------------------------------------
-- FIFO Read Handler
--------------------------------------------------------------------------------

nx_fifo_read_enable     <= not nx_fifo_empty;

PROC_NX_FIFO_READ_ENABLE: process(CLK_IN)
begin
  if (rising_edge(CLK_IN) ) then
    nx_fifo_data_valid_tt    <= nx_fifo_read_enable;
    if(RESET_IN = '1') then
      nx_fifo_data_valid_t   <= '0';
      nx_fifo_data_valid     <= '0';
    else
      -- Delay read signal by two Clock Cycles
      nx_fifo_data_valid_t   <= nx_fifo_data_valid_tt;
      nx_fifo_data_valid     <= nx_fifo_data_valid_t;
    end if;
  end if;
end process PROC_NX_FIFO_READ_ENABLE;

PROC_NX_FIFO_READ: process(CLK_IN)
begin
  if (rising_edge(CLK_IN) ) then
    if (RESET_IN = '1') then
      nx_timestamp_t         <= (others => '0');
```

```
      nx_new_timestamp       <= '0';
      nx_new_timestamp_ctr   <= (others => '0');
      for I in 1 to 15 loop
        nx_timestamp_d(I)    <= (others => '0');
      end loop;
    else
      if (nx_fifo_data_valid = '1') then
        -- Delay Data relative to ADC by 8 steps
        for I in 1 to 15 loop
          nx_timestamp_d(I)  <= nx_timestamp_d(I - 1);
        end loop;
        nx_timestamp_d(0)    <= nx_fifo_data;
        nx_timestamp_t       <= nx_timestamp_d(to_integer(nx_fifo_delay));
        nx_new_timestamp     <= '1';
        nx_new_timestamp_ctr <= nx_new_timestamp_ctr + 1;
      else
        nx_timestamp_t       <= x"deadbeef";
        nx_new_timestamp     <= '0';
      end if;
    end if;
  end if;
end process PROC_NX_FIFO_READ;

-------------------------------------------------------------------------------
-- Status Counters
-------------------------------------------------------------------------------

-- Domain Transfers
pulse_dtrans_2: pulse_dtrans
  generic map (
    CLK_RATIO => 3
    )
  port map (
    CLK_A_IN    => NX_TIMESTAMP_CLK_IN,
    RESET_A_IN  => RESET_NX_TIMESTAMP_CLK_IN,
    PULSE_A_IN  => rs_sync_reset,
    CLK_B_IN    => CLK_IN,
    RESET_B_IN  => RESET_IN,
    PULSE_B_OUT => resync_ctr_inc
    );

pulse_dtrans_3: pulse_dtrans
  generic map (
    CLK_RATIO => 3
    )
  port map (
    CLK_A_IN    => NX_TIMESTAMP_CLK_IN,
    RESET_A_IN  => RESET_NX_TIMESTAMP_CLK_IN,
    PULSE_A_IN  => parity_error,
    CLK_B_IN    => CLK_IN,
    RESET_B_IN  => RESET_IN,
    PULSE_B_OUT => parity_error_ctr_inc
    );

-- nx_frame_synced --> CLK_IN Domain
signal_async_trans_1: signal_async_trans
  port map (
    CLK_IN      => CLK_IN,
    SIGNAL_A_IN => nx_frame_synced,
    SIGNAL_OUT  => reg_nx_frame_synced
    );
```



```
-- Counters
PROC_RESYNC_COUNTER: process(CLK_IN)
begin
  if (rising_edge(CLK_IN) ) then
    if (RESET_IN = '1' or reset_resync_ctr = '1') then
      resync_counter   <= (others => '0');
    else
      if (resync_ctr_inc = '1') then
        resync_counter <= resync_counter + 1;
      end if;
    end if;
  end if;
end process PROC_RESYNC_COUNTER;

PROC_PARITY_ERROR_COUNTER: process(CLK_IN)
begin
  if (rising_edge(CLK_IN) ) then
    if (RESET_IN = '1' or reset_parity_error_ctr = '1') then
      parity_error_counter   <= (others => '0');
    else
      if (parity_error_ctr_inc = '1') then
        parity_error_counter <= parity_error_counter + 1;
      end if;
    end if;
  end if;
end process PROC_PARITY_ERROR_COUNTER;

-------------------------------------------------------------------------------
-- ADC Fifo Handler
-------------------------------------------------------------------------------
PROC_ADC_DATA_READ: process(CLK_IN)
  variable adcval : unsigned(11 downto 0) := (others => '0');
begin
  if (rising_edge(CLK_IN) ) then
    if (RESET_IN = '1') then
      adc_data_t         <= (others => '0');
      adc_new_data       <= '0';
      adc_new_data_ctr   <= (others => '0');
    else
      if (adc_bit_shift(3) = '1') then
        adcval             := unsigned(adc_data) rol
                              to_integer(adc_bit_shift(2 downto 0));
      else
        adcval             := unsigned(adc_data) ror
                              to_integer(adc_bit_shift(2 downto 0));
      end if;
      if (adc_data_valid = '1') then
        adc_data_t       <= std_logic_vector(adcval);
        adc_new_data     <= '1';
        adc_new_data_ctr <= adc_new_data_ctr + 1;
      else
        adc_data_t       <= x"aff";
        adc_new_data     <= '0';
      end if;
    end if;
  end if;
end process PROC_ADC_DATA_READ;

-------------------------------------------------------------------------------
-- Output handler
-------------------------------------------------------------------------------
PROC_OUTPUT_HANDLER: process(CLK_IN)
```

```
 begin
   if (rising_edge(CLK_IN) ) then
     if (RESET_IN = '1' or output_handler_reset = '1') then
       nx_timestamp_o <= (others => '0');
       adc_data_o     <= (others => '0');
       new_data_o     <= '0';
       STATE          <= IDLE;
     else
       case STATE is

         when IDLE =>
           STATE_d <= "00";
           if (nx_new_timestamp = '1' and adc_new_data = '1') then
             nx_timestamp_o  <= nx_timestamp_t;
             adc_data_o      <= adc_data_t;
             new_data_o      <= '1';
             STATE           <= IDLE;
           elsif (nx_new_timestamp = '1') then
             nx_timestamp_o  <= nx_timestamp_t;
             adc_data_o      <= (others => '0');
             new_data_o      <= '0';
             STATE           <= WAIT_ADC;
           elsif (adc_new_data = '1') then
             adc_data_o      <= adc_data_t;
             nx_timestamp_o  <= (others => '0');
             new_data_o      <= '0';
             STATE           <= WAIT_TIMESTAMP;
           else
             nx_timestamp_o  <= (others => '0');
             adc_data_o      <= (others => '0');
             new_data_o      <= '0';
             STATE           <= IDLE;
           end if;

         when WAIT_ADC =>
           STATE_d <= "01";
           if (adc_new_data = '1') then
             adc_data_o      <= adc_data_t;
             new_data_o      <= '1';
             STATE           <= IDLE;
           else
             new_data_o      <= '0';
             STATE           <= WAIT_ADC;
           end if;

          when WAIT_TIMESTAMP =>
           STATE_d <= "10";
           if (nx_new_timestamp = '1') then
             nx_timestamp_o  <= nx_timestamp_t;
             new_data_o      <= '1';
             STATE           <= IDLE;
           else
             new_data_o      <= '0';
             STATE           <= WAIT_TIMESTAMP;
           end if;

       end case;
     end if;
   end if;
 end process PROC_OUTPUT_HANDLER;

 ------------------------------------------------------------------------------
```



```
 -- Rate Counters + Rate Error Check
 ------------------------------------------------------------------------------
 PROC_RATE_COUNTER: process(CLK_IN)
 begin
   if (rising_edge(CLK_IN) ) then
     if (RESET_IN = '1' or frame_rates_reset = '1') then
       nx_frame_rate_ctr      <= (others => '0');
       nx_frame_rate          <= (others => '0');
       adc_frame_rate_ctr     <= (others => '0');
       adc_frame_rate         <= (others => '0');
       frame_rate_ctr         <= (others => '0');
       frame_rate             <= (others => '0');
       parity_rate_ctr        <= (others => '0');
       parity_rate            <= (others => '0');
       rate_timer_ctr         <= (others => '0');
     else
       if (rate_timer_ctr < x"5f5e100") then
         rate_timer_ctr                  <= rate_timer_ctr + 1;

         if (nx_fifo_data_valid = '1') then
           nx_frame_rate_ctr             <= nx_frame_rate_ctr + 1;
         end if;

         if (adc_data_valid = '1') then
           adc_frame_rate_ctr            <= adc_frame_rate_ctr + 1;
         end if;

         if (new_data_o = '1') then
           frame_rate_ctr                <= frame_rate_ctr + 1;
         end if;

         if (parity_error_ctr_inc = '1') then
           parity_rate_ctr               <= parity_rate_ctr + 1;
         end if;
       else
         rate_timer_ctr                  <= (others => '0');
         nx_frame_rate                   <= nx_frame_rate_ctr;
         adc_frame_rate                  <= adc_frame_rate_ctr;
         frame_rate                      <= frame_rate_ctr;
         parity_rate                     <= parity_rate_ctr;

         nx_frame_rate_ctr(27 downto 1)  <= (others => '0');
         nx_frame_rate_ctr(0)            <= nx_fifo_data_valid;

         adc_frame_rate_ctr(27 downto 1) <= (others => '0');
         adc_frame_rate_ctr(0)           <= adc_data_valid;

         frame_rate_ctr(27 downto 1)     <= (others => '0');
         frame_rate_ctr(0)               <= new_data_o;

         parity_rate_ctr(27 downto 1)    <= (others => '0');
         parity_rate_ctr(0)              <= parity_error_ctr_inc;
       end if;
     end if;
   end if;
 end process PROC_RATE_COUNTER;

 -- Check Rates for errors
 PROC_RATE_ERRORS: process(CLK_IN)
 begin
   if (rising_edge(CLK_IN)) then
     if (RESET_IN = '1') then
```

```
      nx_frame_rate_offline      <= '1';
      nx_frame_rate_offline_last <= '1';
      nx_frame_rate_error        <= '0';
      adc_frame_rate_error       <= '0';
      frame_rate_error           <= '0';
      parity_rate_error          <= '0';
      reset_for_offline          <= '0';
    else
      if (nx_frame_rate < 2) then
        -- Assuming nxyter not online or does not receive clock
        nx_frame_rate_offline    <= '1';
      else
        nx_frame_rate_offline    <= '0';
      end if;

      if ((nx_frame_rate < x"1dc_d642"  or
           nx_frame_rate > x"1dc_d652")) then
        nx_frame_rate_error      <= '1';
      else
        nx_frame_rate_error      <= '0';
      end if;

      if ((adc_frame_rate < x"1dc_d64e" or
           adc_frame_rate > x"1dc_d652")) then
        adc_frame_rate_error     <= '1';
      else
        adc_frame_rate_error     <= '0';
      end if;

      if ((frame_rate < x"1dc_d64e" or
           frame_rate > x"1dc_d652")) then
        frame_rate_error         <= '1';
      else
        frame_rate_error         <= '0';
      end if;

      if (parity_rate > 0) then
        parity_rate_error         <= '1';
      else
        parity_rate_error         <= '0';
      end if;

      -- Reset Request to Reset Handler
      nx_frame_rate_offline_last <= nx_frame_rate_offline;
      if (nx_frame_rate_offline_last = '1' and
          nx_frame_rate_offline      = '0') then
        reset_for_offline        <= '1';
      else
        reset_for_offline        <= '0';
      end if;
    end if;
  end if;
end process PROC_RATE_ERRORS;

------------------------------------------------------------------------------
-- TRBNet Slave Bus
------------------------------------------------------------------------------

-- Give status info to the TRB Slow Control Channel
PROC_FIFO_REGISTERS: process(CLK_IN)
begin
  if (rising_edge(CLK_IN) ) then
```



```
    if( RESET_IN = '1' ) then
      slv_data_out_o                <= (others => '0');
      slv_ack_o                     <= '0';
      slv_unknown_addr_o            <= '0';
      slv_no_more_data_o            <= '0';
      reset_resync_ctr              <= '0';
      reset_parity_error_ctr        <= '0';
      debug_adc                     <= (others => '0');
      johnson_counter_sync_r        <= "00";
      pll_adc_sample_clk_dphase_r   <= x"0";
      pll_adc_sample_clk_finedelb   <= (others => '0');
      pll_adc_not_lock_ctr_clear    <= '0';
      nx_fifo_delay                 <= x"8";
      reset_handler_start_r         <= '0';
      reset_handler_counter_clear   <= '0';
      adc_bit_shift                 <= x"0";
      adc_debug_type                <= (others => '0');
    else
      slv_data_out_o                <= (others => '0');
      slv_ack_o                     <= '0';
      slv_unknown_addr_o            <= '0';
      slv_no_more_data_o            <= '0';
      reset_resync_ctr              <= '0';
      reset_parity_error_ctr        <= '0';
      pll_adc_not_lock_ctr_clear    <= '0';
      reset_handler_start_r         <= '0';
      reset_handler_counter_clear   <= '0';

      if (SLV_READ_IN  = '1') then
        case SLV_ADDR_IN is
          when x"0000" =>
            slv_data_out_o                <= nx_timestamp_t;
            slv_ack_o                     <= '1';

          when x"0001" =>
            slv_data_out_o(0)             <= nx_fifo_full;
            slv_data_out_o(1)             <= nx_fifo_empty;
            slv_data_out_o(2)             <= '0';
            slv_data_out_o(3)             <= '0';
            slv_data_out_o(4)             <= nx_fifo_data_valid;
            slv_data_out_o(5)             <= adc_new_data;
            slv_data_out_o(29 downto 5)   <= (others => '0');
            slv_data_out_o(30)            <= '0';
            slv_data_out_o(31)            <= reg_nx_frame_synced;
            slv_ack_o                     <= '1';

          when x"0002" =>
            slv_data_out_o(11 downto  0)  <=
              std_logic_vector(resync_counter);
            slv_data_out_o(31 downto 12)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0003" =>
            slv_data_out_o(11 downto  0)  <=
              std_logic_vector(parity_error_counter);
            slv_data_out_o(31 downto 12)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0004" =>
            slv_data_out_o(11 downto  0)  <=
              std_logic_vector(pll_adc_not_lock_ctr);
            slv_data_out_o(31 downto 12)  <= (others => '0');
```

```
            slv_ack_o                     <= '1';

          when x"0005" =>
            slv_data_out_o(1 downto  0)   <= johnson_counter_sync_r;
            slv_data_out_o(31 downto 2)   <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0006" =>
            slv_data_out_o(3 downto 0)    <=
              std_logic_vector(pll_adc_sample_clk_dphase_r);
            slv_data_out_o(31 downto 4)   <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0007" =>
            slv_data_out_o(3 downto 0)    <= pll_adc_sample_clk_finedelb;
            slv_data_out_o(31 downto 4)   <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0008" =>
            slv_data_out_o(11 downto 0)   <= adc_data_t;
            slv_data_out_o(31 downto 12)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0009" =>
            slv_data_out_o(11 downto 0)   <= std_logic_vector(adc_reset_ctr);
            slv_data_out_o(31 downto 12)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"000a" =>
            slv_data_out_o(31 downto 0)   <= (others => '0');
            slv_ack_o                     <= '1';

          when x"000b" =>
            slv_data_out_o(0)             <= reset_handler_busy;
            slv_data_out_o(31 downto 1)   <= (others => '0');
            slv_ack_o                     <= '1';

          when x"000c" =>
            slv_data_out_o(15 downto 0)   <= reset_handler_counter;
            slv_data_out_o(31 downto 6)   <= (others => '0');
            slv_ack_o                     <= '1';

          when x"000d" =>
            slv_data_out_o(3 downto 0)    <= std_logic_vector(nx_fifo_delay);
            slv_data_out_o(31 downto 4)   <= (others => '0');
            slv_ack_o                     <= '1';

          when x"000e" =>
            slv_data_out_o(3 downto 0)    <= std_logic_vector(adc_bit_shift);
            slv_data_out_o(31 downto 4)   <= (others => '0');
            slv_ack_o                     <= '1';

          when x"000f" =>
            slv_data_out_o(7 downto 0)    <=
              std_logic_vector(adc_notlock_ctr);
            slv_data_out_o(31 downto 8)   <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0010" =>
            slv_data_out_o(27 downto 0)   <= std_logic_vector(nx_frame_rate);
            slv_data_out_o(31 downto 28)  <= (others => '0');
            slv_ack_o                     <= '1';
```



```
          when x"0011" =>
            slv_data_out_o(27 downto 0)   <= std_logic_vector(adc_frame_rate);
            slv_data_out_o(31 downto 28)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0012" =>
            slv_data_out_o(11 downto 0)   <= test_adc_data;
            slv_data_out_o(31 downto 12)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0013" =>
            slv_data_out_o(27 downto 0)   <= std_logic_vector(frame_rate);
            slv_data_out_o(31 downto 28)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0014" =>
            slv_data_out_o(11 downto 0)    <= error_status_bits;
            slv_data_out_o(31 downto 8)   <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0015" =>
            slv_data_out_o(27 downto 0)   <= parity_rate;
            slv_data_out_o(31 downto 28)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"001e" =>
            slv_data_out_o(1 downto 0)    <= debug_adc;
            slv_data_out_o(31 downto 2)   <= (others => '0');
            slv_ack_o                     <= '1';

          when x"001f" =>
            slv_data_out_o(3 downto 0)    <= adc_debug_type;
            slv_data_out_o(31 downto 4)   <= (others => '0');
            slv_ack_o                     <= '1';

          when others  =>
            slv_unknown_addr_o            <= '1';
        end case;

      elsif (SLV_WRITE_IN  = '1') then
        case SLV_ADDR_IN is
          when x"0002" =>
            reset_resync_ctr              <= '1';
            slv_ack_o                     <= '1';

          when x"0003" =>
            reset_parity_error_ctr        <= '1';
            slv_ack_o                     <= '1';

          when x"0004" =>
            pll_adc_not_lock_ctr_clear    <= '1';
            slv_ack_o                     <= '1';

          when x"0005" =>
            johnson_counter_sync_r        <= SLV_DATA_IN(1 downto 0);
            reset_handler_start_r         <= '1';
            slv_ack_o                     <= '1';

          when x"0006" =>
            pll_adc_sample_clk_dphase_r   <=
              unsigned(SLV_DATA_IN(3 downto 0));
```

```
            reset_handler_start_r         <= '1';
            slv_ack_o                     <= '1';

          when x"0007" =>
            pll_adc_sample_clk_finedelb   <= SLV_DATA_IN(3 downto 0);
            reset_handler_start_r         <= '1';
            slv_ack_o                     <= '1';

          when x"000b" =>
            reset_handler_start_r         <= '1';
            slv_ack_o                     <= '1';

          when x"000c" =>
            reset_handler_counter_clear   <= '1';
            slv_ack_o                     <= '1';

          when x"000d" =>
            nx_fifo_delay                 <=
              unsigned(SLV_DATA_IN(3 downto 0));
            slv_ack_o                     <= '1';

          when x"000e" =>
            adc_bit_shift                 <=
              unsigned(SLV_DATA_IN(3 downto 0));
            slv_ack_o                     <= '1';

          when x"001e" =>
            debug_adc                     <= SLV_DATA_IN(1 downto 0);
            slv_ack_o                     <= '1';

          when x"001f" =>
            adc_debug_type                <=
              unsigned(SLV_DATA_IN(3 downto 0));
            slv_ack_o                     <= '1';

          when others  =>
            slv_unknown_addr_o            <= '1';

        end case;
      end if;
    end if;
  end if;
end process PROC_FIFO_REGISTERS;

-- ErrorPROC_ERROR: process(CLK_IN)
PROC_ERROR: process(CLK_IN)
begin
  if (rising_edge(CLK_IN)) then
    if (RESET_IN = '1') then
      error_status_bits  <= (others => '0');
      error_o            <= '0';
    else
      if (error_adc0            = '1'  or
          pll_adc_not_lock      = '1'  or
          nx_frame_rate_offline = '1'  or
          nx_frame_rate_error   = '1'  or
          adc_clk_ok            = '0'  or
          parity_error_ctr_inc  = '1'  or
          reg_nx_frame_synced   = '0'  or
          adc_frame_rate_error  = '1'  or
          parity_rate_error     = '1'
          ) then
```



```
        error_o             <= '1';
      else
        error_o             <= '0';
      end if;

      error_status_bits(0)  <= nx_frame_rate_offline;
      error_status_bits(1)  <= frame_rate_error;
      error_status_bits(2)  <= nx_frame_rate_error;
      error_status_bits(3)  <= adc_frame_rate_error;
      error_status_bits(4)  <= parity_rate_error;
      error_status_bits(5)  <= not reg_nx_frame_synced;
      error_status_bits(6)  <= error_adc0;
      error_status_bits(7)  <= pll_adc_not_lock;
      error_status_bits(8)  <= not adc_clk_ok;
      error_status_bits(9)  <= '0';
      error_status_bits(10) <= '0';
      error_status_bits(11) <= '0';
    end if;
  end if;
end process PROC_ERROR;

-- Output Signals

NX_TIMESTAMP_OUT       <= nx_timestamp_o
                          when new_data_o = '1' else x"0000_0000";
ADC_DATA_OUT           <= adc_data_o when new_data_o = '1' else x"000";
NEW_DATA_OUT           <= new_data_o;
ADC_SCLK_LOCK_OUT      <= pll_adc_sampling_clk_lock;
ERROR_OUT              <= error_o;

SLV_DATA_OUT           <= slv_data_out_o;
SLV_NO_MORE_DATA_OUT   <= slv_no_more_data_o;
SLV_UNKNOWN_ADDR_OUT   <= slv_unknown_addr_o;
SLV_ACK_OUT            <= slv_ack_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.trb_net_std.all;
use work.nxyter_components.all;

entity nx_data_validate is
  port (
    CLK_IN               : in  std_logic;
    RESET_IN             : in  std_logic;

    -- Inputs
    NX_TIMESTAMP_IN      : in  std_logic_vector(31 downto 0);
    ADC_DATA_IN          : in  std_logic_vector(11 downto 0);
    NEW_DATA_IN          : in  std_logic;

    -- Outputs
    TIMESTAMP_OUT        : out std_logic_vector(13 downto 0);
    CHANNEL_OUT          : out std_logic_vector(6 downto 0);
    TIMESTAMP_STATUS_OUT : out std_logic_vector(2 downto 0);
    ADC_DATA_OUT         : out std_logic_vector(11 downto 0);
    DATA_VALID_OUT       : out std_logic;

    NX_TOKEN_RETURN_OUT  : out std_logic;
```

```
    NX_NOMORE_DATA_OUT   : out std_logic;

    -- Slave bus
    SLV_READ_IN          : in  std_logic;
    SLV_WRITE_IN         : in  std_logic;
    SLV_DATA_OUT         : out std_logic_vector(31 downto 0);
    SLV_DATA_IN          : in std_logic_vector(31 downto 0);
    SLV_ADDR_IN          : in std_logic_vector(15 downto 0);
    SLV_ACK_OUT          : out std_logic;
    SLV_NO_MORE_DATA_OUT : out std_logic;
    SLV_UNKNOWN_ADDR_OUT : out std_logic;

    DEBUG_OUT            : out std_logic_vector(15 downto 0)
    );

end entity;

architecture Behavioral of nx_data_validate is

  -- Gray Decoder
  signal nx_timestamp         : std_logic_vector(13 downto 0);
  signal nx_channel_id        : std_logic_vector( 6 downto 0);

  -- TIMESTAMP_BITS
  signal new_timestamp        : std_logic;
  signal valid_frame_bits     : std_logic_vector(3 downto 0);
  signal status_bits          : std_logic_vector(1 downto 0);
  signal parity_error         : std_logic;
  signal adc_data             : std_logic_vector(11 downto 0);

  -- Validate Timestamp
  signal timestamp_o          : std_logic_vector(13 downto 0);
  signal channel_o            : std_logic_vector(6 downto 0);
  signal timestamp_status_o   : std_logic_vector(2 downto 0);
  signal adc_data_o           : std_logic_vector(11 downto 0);
  signal data_valid_o         : std_logic;

  signal nx_token_return_o    : std_logic;
  signal nx_nomore_data_o     : std_logic;

  signal invalid_frame_ctr    : unsigned(15 downto 0);
  signal overflow_ctr         : unsigned(15 downto 0);
  signal pileup_ctr           : unsigned(15 downto 0);

  signal trigger_rate_inc     : std_logic;
  signal frame_rate_inc       : std_logic;
  signal pileup_rate_inc      : std_logic;
  signal overflow_rate_inc    : std_logic;

  -- Self Trigger
  signal self_trigger_o       : std_logic;

  -- Rate Calculation
  signal nx_trigger_ctr_t     : unsigned(27 downto 0);
  signal nx_frame_ctr_t       : unsigned(27 downto 0);
  signal nx_pileup_ctr_t      : unsigned(27 downto 0);
  signal nx_overflow_ctr_t    : unsigned(27 downto 0);
  signal nx_rate_timer        : unsigned(27 downto 0);

  -- ADC Averages
  signal adc_average_divisor  : unsigned(3 downto 0);
  signal adc_average_ctr      : unsigned(8 downto 0);
```



```
  signal adc_average_sum      : unsigned(24 downto 0);
  signal adc_average          : unsigned(11 downto 0);
  signal adc_data_last        : std_logic_vector(11 downto 0);
  signal adc_av               : std_logic;

  -- Config
  signal readout_type         : std_logic_vector(1 downto 0);

  -- Slave Bus
  signal slv_data_out_o       : std_logic_vector(31 downto 0);
  signal slv_no_more_data_o   : std_logic;
  signal slv_unknown_addr_o   : std_logic;
  signal slv_ack_o            : std_logic;
  signal clear_counters       : std_logic;
  signal nx_hit_rate          : unsigned(27 downto 0);
  signal nx_frame_rate        : unsigned(27 downto 0);
  signal nx_pileup_rate       : unsigned(27 downto 0);
  signal nx_overflow_rate     : unsigned(27 downto 0);
  signal invalid_adc : std_logic;

begin

  -- Debug Line
  DEBUG_OUT(0)                    <= CLK_IN;
  DEBUG_OUT(1)                    <= nx_token_return_o;
  DEBUG_OUT(2)                    <= nx_nomore_data_o;
  DEBUG_OUT(3)                    <= data_valid_o;
  DEBUG_OUT(4)                    <= new_timestamp;
  DEBUG_OUT(5)                    <= self_trigger_o;
  DEBUG_OUT(8 downto 6)           <= (others => '0');
  DEBUG_OUT(15 downto 9)          <= channel_o;
  --DEBUG_OUT(6 downto 4)           <= timestamp_status_o;
  --DEBUG_OUT(7)                    <= nx_token_return_o;
  --DEBUG_OUT(8)                    <= invalid_adc;--nx_nomore_data_o;

  --DEBUG_OUT(15 downto 9)          <= channel_o;

  ------------------------------------------------------------------------------
  -- Data Separation
  ------------------------------------------------------------------------------

  -- Separate Timestamp-, Status-, Parity- and Frame-bits
  PROC_TIMESTAMP_BITS: process (CLK_IN)
  begin
    if( rising_edge(CLK_IN) ) then
      if (RESET_IN = '1') then
        valid_frame_bits    <= (others => '0');
        nx_timestamp        <= (others => '0');
        status_bits         <= (others => '0');
        parity_error        <= '0';
        new_timestamp       <= '0';
        adc_data            <= (others => '0');
      else
        if (NEW_DATA_IN = '1') then
          valid_frame_bits(3)       <= NX_TIMESTAMP_IN(31);
          valid_frame_bits(2)       <= NX_TIMESTAMP_IN(23);
          valid_frame_bits(1)       <= NX_TIMESTAMP_IN(15);
          valid_frame_bits(0)       <= NX_TIMESTAMP_IN(7);
          nx_timestamp(13 downto 7) <= NX_TIMESTAMP_IN(30 downto 24);
          nx_timestamp(6 downto 0)  <= NX_TIMESTAMP_IN(22 downto 16);
          status_bits               <= NX_TIMESTAMP_IN(2 downto 1);
          parity_error              <= NX_TIMESTAMP_IN(0);
```

```
        adc_data                <= ADC_DATA_IN;
        new_timestamp           <= '1';
      else
        valid_frame_bits        <= (others => '0');
        nx_timestamp            <= (others => '0');
        status_bits             <= (others => '0');
        parity_error            <= '0';
        adc_data                <= (others => '0');
        new_timestamp           <= '0';
      end if;
    end if;
  end if;
end process PROC_TIMESTAMP_BITS;

-------------------------------------------------------------------------------
-- Filter only valid events
-------------------------------------------------------------------------------

PROC_VALIDATE_TIMESTAMP: process (CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if (RESET_IN = '1') then
      timestamp_o           <= (others => '0');
      channel_o             <= (others => '0');
      timestamp_status_o    <= (others => '0');
      adc_data_o            <= (others => '0');
      data_valid_o          <= '0';
      nx_token_return_o     <= '0';
      nx_nomore_data_o      <= '0';
      trigger_rate_inc      <= '0';
      frame_rate_inc        <= '0';
      pileup_rate_inc       <= '0';
      overflow_rate_inc     <= '0';

      invalid_frame_ctr     <= (others => '0');
      overflow_ctr          <= (others => '0');
      pileup_ctr            <= (others => '0');
    else
      timestamp_o           <= (others => '0');
      channel_o             <= (others => '0');
      timestamp_status_o    <= (others => '0');
      adc_data_o            <= (others => '0');
      data_valid_o          <= '0';
      trigger_rate_inc      <= '0';
      frame_rate_inc        <= '0';
      pileup_rate_inc       <= '0';
      overflow_rate_inc     <= '0';
      invalid_adc           <= '0';

      if (new_timestamp = '1') then
        case valid_frame_bits is

          -- Data Frame
          when "1000" =>
            ---- Check Overflow
            if ((status_bits(0) = '1') and (clear_counters = '0')) then
              overflow_ctr              <= overflow_ctr + 1;
              overflow_rate_inc         <= '1';
            end if;

            -- Check PileUp
            if ((status_bits(1) = '1') and (clear_counters = '0')) then
```



```
              pileup_ctr                <= pileup_ctr + 1;
              pileup_rate_inc           <= '1';
            end if;

            -- Take Timestamp
            timestamp_o                 <= nx_timestamp;
            channel_o                   <= nx_channel_id;
            timestamp_status_o(2)       <= parity_error;
            timestamp_status_o(1 downto 0) <= status_bits;
            adc_data_o                  <= adc_data;
            data_valid_o                <= '1';

            if (adc_data = x"aff") then
              invalid_adc               <= '1';
            end if;

            nx_token_return_o           <= '0';
            nx_nomore_data_o            <= '0';
            trigger_rate_inc            <= '1';

          -- Token return and nomore_data
          when "0000" =>
            nx_token_return_o           <= '1';
            nx_nomore_data_o            <= nx_token_return_o;

          when others =>
            -- Invalid frame, not empty, discard timestamp
            if (clear_counters = '0') then
              invalid_frame_ctr         <= invalid_frame_ctr + 1;
            end if;
            nx_token_return_o           <= '0';
            nx_nomore_data_o            <= '0';

        end case;

        frame_rate_inc                  <= '1';

      else
        nx_token_return_o               <= nx_token_return_o;
        nx_nomore_data_o                <= nx_nomore_data_o;
      end if;

      -- Reset Counters
      if (clear_counters = '1') then
        invalid_frame_ctr               <= (others => '0');
        overflow_ctr                    <= (others => '0');
        pileup_ctr                      <= (others => '0');
      end if;
    end if;
  end if;
end process PROC_VALIDATE_TIMESTAMP;

PROC_CAL_RATES: process (CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if (RESET_IN = '1') then
      nx_trigger_ctr_t      <= (others => '0');
      nx_frame_ctr_t        <= (others => '0');
      nx_rate_timer         <= (others => '0');
      nx_hit_rate           <= (others => '0');
      nx_frame_rate         <= (others => '0');
    else
```

```
      if (nx_rate_timer < x"5f5e100") then
        if (trigger_rate_inc = '1') then
          nx_trigger_ctr_t             <= nx_trigger_ctr_t + 1;
        end if;
        if (frame_rate_inc = '1') then
          nx_frame_ctr_t               <= nx_frame_ctr_t + 1;
        end if;
        if (pileup_rate_inc = '1') then
          nx_pileup_ctr_t              <= nx_pileup_ctr_t + 1;
        end if;
        if (overflow_rate_inc = '1') then
          nx_overflow_ctr_t            <= nx_overflow_ctr_t + 1;
        end if;
        nx_rate_timer                  <= nx_rate_timer + 1;
      else
        nx_hit_rate                    <= nx_trigger_ctr_t;
        nx_frame_rate                  <= nx_frame_ctr_t;
        nx_pileup_rate                 <= nx_pileup_ctr_t;
        nx_overflow_rate               <= nx_overflow_ctr_t;

        nx_trigger_ctr_t(27 downto 1)  <= (others => '0');
        nx_trigger_ctr_t(0)            <= trigger_rate_inc;

        nx_frame_ctr_t(27 downto 1)    <= (others => '0');
        nx_frame_ctr_t(0)              <= frame_rate_inc;

        nx_pileup_ctr_t(27 downto 1)   <= (others => '0');
        nx_pileup_ctr_t(0)             <= pileup_rate_inc;

        nx_overflow_ctr_t(27 downto 1) <= (others => '0');
        nx_overflow_ctr_t(0)           <= overflow_rate_inc;

        nx_rate_timer                  <= (others => '0');
      end if;
    end if;
  end if;
end process PROC_CAL_RATES;

PROC_ADC_AVERAGE: process(CLK_IN)
begin
  if (rising_edge(CLK_IN) ) then
    if (RESET_IN = '1') then
      adc_average_ctr    <= (others => '0');
      adc_average_sum    <= (others => '0');
      adc_average        <= (others => '0');
      adc_data_last      <= (others => '0');
      adc_av             <= '0';
    else
      adc_av             <= '0';
      if ((adc_average_ctr srl to_integer(adc_average_divisor)) > 0) then
        adc_average      <= (adc_average_sum srl
                             to_integer(adc_average_divisor))(11 downto 0);
        adc_average_sum  <= (others => '0');
        adc_average_ctr  <= (others => '0');
        adc_av           <= '1';
      elsif (data_valid_o = '1') then
        adc_average_sum  <= adc_average_sum + unsigned(adc_data_o);
        adc_average_ctr  <= adc_average_ctr + 1;
      end if;

      if (data_valid_o = '1') then
        adc_data_last    <= adc_data_o;
```



```
      end if;
    end if;
  end if;
end process PROC_ADC_AVERAGE;

------------------------------------------------------------------------------
-- TRBNet Slave Bus
------------------------------------------------------------------------------

-- Give status info to the TRB Slow Control Channel
PROC_FIFO_REGISTERS: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      slv_data_out_o         <= (others => '0');
      slv_ack_o              <= '0';
      slv_unknown_addr_o     <= '0';
      slv_no_more_data_o     <= '0';
      clear_counters         <= '0';
      adc_average_divisor    <= x"3";
    else
      slv_data_out_o         <= (others => '0');
      slv_unknown_addr_o     <= '0';
      slv_no_more_data_o     <= '0';
      clear_counters         <= '0';

      if (SLV_READ_IN  = '1') then
        case SLV_ADDR_IN is

          when x"0000" =>
            slv_data_out_o(15 downto 0)  <=
              std_logic_vector(invalid_frame_ctr);
            slv_data_out_o(31 downto 16) <= (others => '0');
            slv_ack_o                    <= '1';

          when x"0001" =>
            slv_data_out_o(15 downto 0)  <=
              std_logic_vector(overflow_ctr);
            slv_data_out_o(31 downto 16) <= (others => '0');
            slv_ack_o                    <= '1';

          when x"0002" =>
            slv_data_out_o(15 downto 0)  <=
              std_logic_vector(pileup_ctr);
            slv_data_out_o(31 downto 16) <= (others => '0');
            slv_ack_o                    <= '1';

          when x"0003" =>
            -- Reserved
            slv_data_out_o               <= (others => '0');
            slv_ack_o                    <= '1';

          when x"0004" =>
            slv_data_out_o(27 downto 0)  <=
              std_logic_vector(nx_hit_rate);
            slv_data_out_o(31 downto 28) <= (others => '0');
            slv_ack_o                    <= '1';

          when x"0005" =>
            slv_data_out_o(27 downto 0)  <=
              std_logic_vector(nx_frame_rate);
            slv_data_out_o(31 downto 28) <= (others => '0');
```

```
            slv_ack_o                    <= '1';

          when x"0006" =>
            slv_data_out_o(11 downto 0)   <= adc_data_last;
            slv_data_out_o(31 downto 12)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0007" =>
            slv_data_out_o(11 downto 0)   <= std_logic_vector(adc_average);
            slv_data_out_o(31 downto 12)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0008" =>
            slv_data_out_o(3 downto 0)    <=
              std_logic_vector(adc_average_divisor);
            slv_data_out_o(31 downto 4)   <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0009" =>
            slv_data_out_o(27 downto 0)    <=
              std_logic_vector(nx_pileup_rate);
            slv_data_out_o(31 downto 28)   <= (others => '0');
            slv_ack_o                     <= '1';

          when x"000a" =>
            slv_data_out_o(27 downto 0)    <=
              std_logic_vector(nx_overflow_rate);
            slv_data_out_o(31 downto 28)   <= (others => '0');
            slv_ack_o                     <= '1';

          when others  =>
            slv_unknown_addr_o           <= '1';
            slv_ack_o                    <= '0';
        end case;

      elsif (SLV_WRITE_IN  = '1') then
        case SLV_ADDR_IN is
          when x"0000" =>
            clear_counters               <= '1';
            slv_ack_o                    <= '1';

          when x"0008" =>
            adc_average_divisor           <= SLV_DATA_IN(3 downto 0);
            slv_ack_o                     <= '1';

          when others  =>
            slv_unknown_addr_o           <= '1';
            slv_ack_o                    <= '0';
        end case;
      else
        slv_ack_o                        <= '0';
      end if;
    end if;
  end if;
end process PROC_FIFO_REGISTERS;

-------------------------------------------------------------------------------
-- Output Signals
-------------------------------------------------------------------------------

TIMESTAMP_OUT         <= timestamp_o;
CHANNEL_OUT           <= channel_o;
```



```
  TIMESTAMP_STATUS_OUT  <= timestamp_status_o;
  ADC_DATA_OUT          <= adc_data_o;
  DATA_VALID_OUT        <= data_valid_o;
  NX_TOKEN_RETURN_OUT   <= nx_token_return_o;
  NX_NOMORE_DATA_OUT    <= nx_nomore_data_o;

  -- Slave
  SLV_DATA_OUT          <= slv_data_out_o;
  SLV_NO_MORE_DATA_OUT  <= slv_no_more_data_o;
  SLV_UNKNOWN_ADDR_OUT  <= slv_unknown_addr_o;
  SLV_ACK_OUT           <= slv_ack_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.nxyter_components.all;
use work.trb3_components.all;

entity nx_event_buffer is
  generic (
    BOARD_ID : std_logic_vector(1 downto 0) := "11"
    );
  port (
    CLK_IN                    : in  std_logic;
    RESET_IN                  : in  std_logic;
    RESET_DATA_BUFFER_IN      : in  std_logic;
    NXYTER_OFFLINE_IN         : in  std_logic;

    -- Data Buffer FIFO
    DATA_IN                   : in  std_logic_vector(31 downto 0);
    DATA_CLK_IN               : in  std_logic;
    EVT_NOMORE_DATA_IN        : in  std_logic;

    -- Trigger
    TRIGGER_IN                : in  std_logic;
    FAST_CLEAR_IN             : in  std_logic;
    TRIGGER_BUSY_OUT          : out std_logic;
    EVT_BUFFER_FULL_OUT       : out std_logic;

    --Response from FEE
    FEE_DATA_OUT              : out std_logic_vector(31 downto 0);
    FEE_DATA_WRITE_OUT        : out std_logic;
    FEE_DATA_ALMOST_FULL_IN   : in  std_logic;

    -- Slave bus
    SLV_READ_IN               : in  std_logic;
    SLV_WRITE_IN              : in  std_logic;
    SLV_DATA_OUT              : out std_logic_vector(31 downto 0);
    SLV_DATA_IN               : in  std_logic_vector(31 downto 0);
    SLV_ADDR_IN               : in  std_logic_vector(15 downto 0);
    SLV_ACK_OUT               : out std_logic;
    SLV_NO_MORE_DATA_OUT      : out std_logic;
    SLV_UNKNOWN_ADDR_OUT      : out std_logic;

    ERROR_OUT                 : out std_logic;

    DEBUG_OUT                 : out std_logic_vector(15 downto 0)
    );
```

```
end entity;

architecture Behavioral of nx_event_buffer is

  --Data channel
  signal fee_data_o           : std_logic_vector(31 downto 0);
  signal fee_data_write_o     : std_logic;
  signal trigger_busy_o       : std_logic;
  signal evt_data_flush       : std_logic;

  type STATES is (S_IDLE,
                  S_FLUSH_BUFFER_WAIT
                  );
  signal STATE : STATES;

  -- FIFO
  signal fifo_reset           : std_logic;
  signal fifo_read_enable     : std_logic;

  -- FIFO Input Handler
  signal fifo_next_word       : std_logic_vector(31 downto 0);
  signal fifo_full            : std_logic;
  signal fifo_write_enable    : std_logic;
  signal fifo_almost_full_thr : std_logic_vector(10 downto 0);

    -- NOMORE_DATA RS FlipFlop
  signal flush_end_enable_set : std_logic;
  signal flush_end_enable     : std_logic;

  -- FIFO Read Handler
  signal fifo_o               : std_logic_vector(31 downto 0);
  signal fifo_empty           : std_logic;
  signal fifo_read_start      : std_logic;
  signal fifo_almost_full     : std_logic;

  signal fifo_read_enable_s   : std_logic;
  signal fifo_read_busy       : std_logic;
  signal fifo_no_data         : std_logic;
  signal fifo_read_done       : std_logic;
  signal evt_buffer_full_o    : std_logic;
  signal fifo_data            : std_logic_vector(31 downto 0);

  type R_STATES is (R_IDLE,
                    R_NOP1,
                    R_NOP2,
                    R_READ_WORD
                    );

  signal R_STATE : R_STATES;

  -- Event Buffer Output Handler
  signal evt_data_clk              : std_logic;
  signal evt_data_flushed          : std_logic;

  signal fifo_read_enable_f        : std_logic;
  signal fifo_read_enable_f2       : std_logic;
  signal fifo_flush_ctr            : unsigned(10 downto 0);
  signal fifo_flush_ctr_last       : unsigned(10 downto 0);

  signal evt_data_flushed_x        : std_logic;
  signal fifo_flush_ctr_x          : unsigned(10 downto 0);
  signal flush_end_enable_reset_x  : std_logic;
```



```
  type F_STATES is (F_IDLE,
                    F_FLUSH,
                    F_END
                    );

  signal F_STATE, F_NEXT_STATE : F_STATES;

  -- Error Status
  signal fifo_almost_full_p    : std_logic;
  signal error_status_o        : std_logic;
  signal fifo_full_rate_ctr    : unsigned(19 downto 0);
  signal fifo_full_rate        : unsigned(19 downto 0);
  signal rate_timer_ctr        : unsigned(27 downto 0);

  -- Slave Bus
  signal slv_data_out_o        : std_logic_vector(31 downto 0);
  signal slv_no_more_data_o    : std_logic;
  signal slv_unknown_addr_o    : std_logic;
  signal slv_ack_o             : std_logic;

  signal register_fifo_status  : std_logic_vector(7 downto 0);

  signal data_wait             : std_logic;

begin

  DEBUG_OUT(0)           <= CLK_IN;
  DEBUG_OUT(1)           <= DATA_CLK_IN;
  DEBUG_OUT(2)           <= fifo_empty;
  DEBUG_OUT(3)           <= fifo_almost_full;
  DEBUG_OUT(4)           <= RESET_DATA_BUFFER_IN;
  DEBUG_OUT(5)           <= trigger_busy_o;
  DEBUG_OUT(6)           <= TRIGGER_IN;
  DEBUG_OUT(7)           <= evt_data_flush;
  DEBUG_OUT(8)           <= flush_end_enable;
  DEBUG_OUT(9)           <= evt_data_clk;
  DEBUG_OUT(10)          <= fee_data_write_o;
  DEBUG_OUT(11)          <= evt_data_flushed;
  DEBUG_OUT(12)          <= '0';
  DEBUG_OUT(13)          <= EVT_NOMORE_DATA_IN;
  DEBUG_OUT(14)          <= FAST_CLEAR_IN;
  DEBUG_OUT(15)          <= FEE_DATA_ALMOST_FULL_IN;

  ------------------------------------------------------------------------------
  --
  ------------------------------------------------------------------------------

  PROC_DATA_HANDLER: process(CLK_IN)
  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1' ) then
        evt_data_flush       <= '0';
        trigger_busy_o       <= '0';
        STATE                <= S_IDLE;
      else
        evt_data_flush       <= '0';
        trigger_busy_o       <= '1';

        if (FAST_CLEAR_IN = '1') then
          STATE                      <= S_IDLE;
        else
```

```
        case STATE is
          when S_IDLE =>
            if (NXYTER_OFFLINE_IN = '1') then
              trigger_busy_o             <= '0';
              STATE                      <= S_IDLE;
            elsif (TRIGGER_IN = '1') then
              evt_data_flush             <= '1';
              STATE                      <= S_FLUSH_BUFFER_WAIT;
            else
              trigger_busy_o             <= '0';
              STATE                      <= S_IDLE;
            end if;

          when S_FLUSH_BUFFER_WAIT =>
            if (evt_data_flushed = '0') then
              STATE                      <= S_FLUSH_BUFFER_WAIT;
            else
              STATE                      <= S_IDLE;
            end if;

        end case;
      end if;
    end if;
  end if;
end process PROC_DATA_HANDLER;

-------------------------------------------------------------------------------
-- FIFO Input Handler
-------------------------------------------------------------------------------

-- Send data to FIFO
fifo_32_data_1: fifo_32_data
  port map (
    Data         => fifo_next_word,
    Clock        => CLK_IN,
    WrEn         => fifo_write_enable,
    RdEn         => fifo_read_enable,
    Reset        => fifo_reset,
    AmFullThresh => fifo_almost_full_thr,
    Q            => fifo_o,
    Empty        => fifo_empty,
    Full         => fifo_full,
    AlmostFull   => fifo_almost_full
    );

fifo_reset       <= RESET_IN or RESET_DATA_BUFFER_IN;
fifo_read_enable <= fifo_read_enable_f or fifo_read_enable_s;

PROC_FIFO_WRITE_HANDLER: process(CLK_IN)
begin
  if(rising_edge(CLK_IN)) then
    if(RESET_IN = '1' or RESET_DATA_BUFFER_IN = '1') then
      fifo_write_enable   <= '0';
    else
      fifo_write_enable   <= '0';
      fifo_next_word      <= x"deadbeef";

      if (DATA_CLK_IN      = '1' and
          fifo_full        = '0' and
          fifo_almost_full = '0') then
        fifo_next_word    <= DATA_IN;
        fifo_write_enable <= '1';
```



```
      end if;

    end if;
  end if;
end process PROC_FIFO_WRITE_HANDLER;

PROC_FLUSH_END_RS_FF: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' or flush_end_enable_reset_x = '1') then
      flush_end_enable <= '0';
    else
      if (flush_end_enable_set = '1') then
        flush_end_enable <= '1';
      end if;
    end if;
  end if;
end process PROC_FLUSH_END_RS_FF;

flush_end_enable_set <= EVT_NOMORE_DATA_IN;

PROC_FLUSH_BUFFER_TRANSFER: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    fifo_read_enable_f2    <= fifo_read_enable_f;
    if( RESET_IN = '1' ) then
      evt_data_clk           <= '0';
      evt_data_flushed       <= '0';
      fifo_flush_ctr         <= (others => '0');
      fifo_flush_ctr_last    <= (others => '0');
      F_STATE                <= F_IDLE;
    else
      evt_data_flushed       <= evt_data_flushed_x;
      fifo_flush_ctr         <= fifo_flush_ctr_x;
      F_STATE                <= F_NEXT_STATE;
      evt_data_clk           <= fifo_read_enable_f2;

      if (F_STATE = F_END) then
        fifo_flush_ctr_last  <= fifo_flush_ctr_x;
      end if;
    end if;
  end if;
end process PROC_FLUSH_BUFFER_TRANSFER;

PROC_FLUSH_BUFFER: process(F_STATE,
                           evt_data_flush,
                           fifo_empty,
                           evt_data_clk,
                           flush_end_enable
                           )
begin
  -- Defaults
  fifo_read_enable_f        <= '0';
  fifo_flush_ctr_x          <= fifo_flush_ctr;
  evt_data_flushed_x        <= '0';
  flush_end_enable_reset_x  <= '0';

  -- Multiplexer fee_data_o
  if (evt_data_clk = '1') then
    fee_data_o                <= fifo_o;
    fee_data_write_o          <= '1';
  else
```

```
    fee_data_o                  <= (others => '1');
    fee_data_write_o            <= '0';
  end if;

  -- FIFO Read Handler
  case F_STATE is
    when F_IDLE =>
      if (evt_data_flush = '1') then
        fifo_flush_ctr_x        <= (others => '0');
        flush_end_enable_reset_x <= '1';
        F_NEXT_STATE            <= F_FLUSH;
      else
        F_NEXT_STATE            <= F_IDLE;
      end if;

    when F_FLUSH =>
      if (fifo_empty = '0') then
        fifo_read_enable_f      <= '1';
        fifo_flush_ctr_x        <= fifo_flush_ctr + 1;
        F_NEXT_STATE            <= F_FLUSH;
      else
        if (flush_end_enable = '0') then
          F_NEXT_STATE          <= F_FLUSH;
        else
          F_NEXT_STATE          <= F_END;
        end if;
      end if;

    when F_END =>
      evt_data_flushed_x        <= '1';
      F_NEXT_STATE              <= F_IDLE;

  end case;
end process PROC_FLUSH_BUFFER;

--------------------------------------------------------------------------------
-- FIFO Output Handler
--------------------------------------------------------------------------------

PROC_FIFO_READ_WORD: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      fifo_read_enable_s    <= '0';
      fifo_read_busy        <= '0';
      fifo_data             <= (others => '0');
      fifo_read_done        <= '0';
      fifo_no_data          <= '1';
      R_STATE               <= R_IDLE;
    else
      fifo_read_busy        <= '0';
      fifo_no_data          <= '0';
      fifo_read_done        <= '0';
      fifo_data             <= (others => '0');
      fifo_read_enable_s    <= '0';

      case R_STATE is
        when R_IDLE =>
          if (fifo_read_start = '1') then
            if (fifo_empty = '0') then
              fifo_read_enable_s <= '1';
              fifo_read_busy     <= '1';
```



```
              R_STATE           <= R_NOP1;
            else
              fifo_no_data      <= '1';
              fifo_read_done    <= '1';
              R_STATE           <= R_IDLE;
            end if;
          else
            R_STATE             <= R_IDLE;
          end if;

        when R_NOP1 =>
          fifo_read_busy        <= '1';
          R_STATE               <= R_NOP2;

        when R_NOP2 =>
          fifo_read_busy        <= '1';
          R_STATE               <= R_READ_WORD;

        when R_READ_WORD =>
          fifo_read_busy        <= '0';
          fifo_data             <= fifo_o;
          fifo_read_done        <= '1';
          R_STATE               <= R_IDLE;

      end case;
    end if;
  end if;

end process PROC_FIFO_READ_WORD;

--------------------------------------------------------------------------------
-- Rate Counters + Rate Error Check
--------------------------------------------------------------------------------
level_to_pulse_FIFO_FULL: level_to_pulse
  port map (
    CLK_IN    => CLK_IN,
    RESET_IN  => RESET_IN,
    LEVEL_IN  => fifo_almost_full,
    PULSE_OUT => fifo_almost_full_p
    );


PROC_RATE_COUNTER: process(CLK_IN)
begin
  if (rising_edge(CLK_IN) ) then
    if (RESET_IN = '1') then
      fifo_full_rate_ctr    <= (others => '0');
      fifo_full_rate        <= (others => '0');
      rate_timer_ctr        <= (others => '0');
      error_status_o        <= '0';
    else
      if (rate_timer_ctr < x"5f5e100") then
        rate_timer_ctr              <= rate_timer_ctr + 1;

        if (fifo_almost_full_p = '1') then
          fifo_full_rate_ctr        <= fifo_full_rate_ctr + 1;
        end if;
      else
        rate_timer_ctr              <= (others => '0');
        fifo_full_rate              <= fifo_full_rate_ctr;

        fifo_full_rate_ctr(19 downto 1) <= (others => '0');
```

```
        fifo_full_rate_ctr(0)          <= fifo_almost_full_p;

        if (fifo_full_rate > 0) then
          error_status_o               <= '1';
        else
          error_status_o               <= '0';
        end if;
      end if;
    end if;
  end if;
end process PROC_RATE_COUNTER;

-------------------------------------------------------------------------------
-- Slave Bus Slow Control
-------------------------------------------------------------------------------

register_fifo_status(0)            <= fifo_write_enable;
register_fifo_status(1)            <= fifo_full;
register_fifo_status(3 downto 2)   <= (others => '0');
register_fifo_status(4)            <= fifo_read_enable;
register_fifo_status(5)            <= fifo_empty;
register_fifo_status(7 downto 6)   <= (others => '0');

PROC_SLAVE_BUS: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      slv_data_out_o         <= (others => '0');
      slv_ack_o              <= '0';
      slv_unknown_addr_o     <= '0';
      slv_no_more_data_o     <= '0';

      fifo_read_start        <= '0';
      data_wait              <= '0';
      fifo_almost_full_thr   <= "00101011110"; -- default: 350 = 1.4k
    else
      slv_data_out_o         <= (others => '0');
      slv_ack_o              <= '0';
      slv_unknown_addr_o     <= '0';
      slv_no_more_data_o     <= '0';

      fifo_read_start        <= '0';
      data_wait              <= '0';

      if (data_wait = '1') then
        if (fifo_read_done = '0') then
          data_wait                    <= '1';
        else
          if (fifo_no_data = '0') then
            slv_data_out_o             <= fifo_data;
            slv_ack_o                  <= '1';
          else
            slv_no_more_data_o         <= '1';
            slv_ack_o                  <= '0';
          end if;
          data_wait                    <= '0';
        end if;

      elsif (SLV_READ_IN  = '1') then
        case SLV_ADDR_IN is
          when x"0000" =>
            fifo_read_start             <= '1';
```



```
            data_wait                  <= '1';

          when x"0001" =>
            slv_data_out_o(10 downto 0)   <= fifo_almost_full_thr;
            slv_data_out_o(31 downto 11)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0002" =>
            slv_data_out_o(10 downto 0)   <=
              std_logic_vector(fifo_flush_ctr_last);
            slv_data_out_o(31 downto 11)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0003" =>
             slv_data_out_o(19 downto 0)  <= fifo_full_rate;
             slv_data_out_o(31 downto 20) <= (others => '0');
             slv_ack_o                    <= '1';

          when x"0004" =>
             slv_data_out_o(0)            <= error_status_o;
             slv_data_out_o(31 downto 1)  <= (others => '0');
             slv_ack_o                    <= '1';

          when x"0005" =>
            slv_data_out_o(7 downto 0)    <= register_fifo_status;
            slv_data_out_o(31 downto 8)   <= (others => '0');
            slv_ack_o                     <= '1';

          when others  =>
            slv_unknown_addr_o            <= '1';
        end case;

      elsif (SLV_WRITE_IN  = '1') then
        case SLV_ADDR_IN is
          when x"0001" =>
            if (unsigned(slv_data_out_o(10 downto 0)) < 2040) then
              fifo_almost_full_thr        <= SLV_DATA_IN(10 downto 0);
            end if;
            slv_ack_o                     <= '1';

          when others  =>
            slv_unknown_addr_o            <= '1';
            slv_ack_o                     <= '0';
        end case;

      else
        slv_ack_o                        <= '0';
      end if;
    end if;
  end if;
end process PROC_SLAVE_BUS;


-- Output Signals

evt_buffer_full_o      <= fifo_almost_full;

TRIGGER_BUSY_OUT       <= trigger_busy_o;
EVT_BUFFER_FULL_OUT    <= evt_buffer_full_o;

FEE_DATA_OUT           <= fee_data_o;
FEE_DATA_WRITE_OUT     <= fee_data_write_o;
```

```
  SLV_DATA_OUT           <= slv_data_out_o;
  SLV_NO_MORE_DATA_OUT   <= slv_no_more_data_o;
  SLV_UNKNOWN_ADDR_OUT   <= slv_unknown_addr_o;
  SLV_ACK_OUT            <= slv_ack_o;

  ERROR_OUT              <= error_status_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.nxyter_components.all;

entity nx_fpga_timestamp is
  port (
    CLK_IN                  : in  std_logic;
    RESET_IN                : in  std_logic;
    NX_MAIN_CLK_IN          : in  std_logic;

    TIMESTAMP_SYNC_IN       : in  std_logic;
    TRIGGER_IN              : in  std_logic; -- must be in NX_MAIN_CLK_DOMAIN
    TIMESTAMP_CURRENT_OUT   : out unsigned(11 downto 0);
    TIMESTAMP_HOLD_OUT      : out unsigned(11 downto 0);
    TIMESTAMP_SYNCED_OUT    : out std_logic;
    TIMESTAMP_TRIGGER_OUT   : out std_logic;

    -- Slave bus
    SLV_READ_IN             : in  std_logic;
    SLV_WRITE_IN            : in  std_logic;
    SLV_DATA_OUT            : out std_logic_vector(31 downto 0);
    SLV_DATA_IN             : in  std_logic_vector(31 downto 0);
    SLV_ACK_OUT             : out std_logic;
    SLV_NO_MORE_DATA_OUT    : out std_logic;
    SLV_UNKNOWN_ADDR_OUT    : out std_logic;

    -- Debug Line
    DEBUG_OUT               : out std_logic_vector(15 downto 0)
    );
end entity;

architecture Behavioral of nx_fpga_timestamp is

  signal timestamp_ctr       : unsigned(11 downto 0);
  signal timestamp_current_o : unsigned(11 downto 0);
  signal timestamp_hold_o    : std_logic_vector(11 downto 0);
  signal timestamp_trigger_o : std_logic;
  signal timestamp_sync      : std_logic;

  signal timestamp_synced    : std_logic;
  signal timestamp_synced_o  : std_logic;

  signal fifo_full           : std_logic;
  signal fifo_write_enable   : std_logic;

  -- Reset
  signal RESET_NX_MAIN_CLK_IN : std_logic;

begin
```



```
  DEBUG_OUT(0)             <= CLK_IN;
  DEBUG_OUT(1)             <= TIMESTAMP_SYNC_IN;
  DEBUG_OUT(2)             <= timestamp_synced_o;
  DEBUG_OUT(3)             <= TRIGGER_IN;

  DEBUG_OUT(15 downto 4)   <= timestamp_hold_o(11 downto 0);

  ------------------------------------------------------------------------------
  -- Reset Domain Transfer
  ------------------------------------------------------------------------------
  signal_async_trans_RESET_IN: signal_async_trans
    port map (
      CLK_IN      => NX_MAIN_CLK_IN,
      SIGNAL_A_IN => RESET_IN,
      SIGNAL_OUT  => RESET_NX_MAIN_CLK_IN
    );

  ------------------------------------------------------------------------------
  -- NX Clock Domain
  ------------------------------------------------------------------------------

  signal_async_to_pulse_TIMESTAMP_SYNC_IN: signal_async_to_pulse
    generic map (
      NUM_FF => 3
      )
    port map (
      CLK_IN     => NX_MAIN_CLK_IN,
      RESET_IN   => RESET_NX_MAIN_CLK_IN,
      PULSE_A_IN => TIMESTAMP_SYNC_IN,
      PULSE_OUT  => timestamp_sync
      );

  -- Timestamp Process + Trigger
  PROC_TIMESTAMP_CTR: process (NX_MAIN_CLK_IN)
  begin
    if( rising_edge(NX_MAIN_CLK_IN) ) then
      if( RESET_NX_MAIN_CLK_IN = '1' ) then
        timestamp_ctr          <= (others => '0');
        timestamp_hold_o       <= (others => '0');
        timestamp_synced       <= '0';
      else
        timestamp_trigger_o    <= '1';
        timestamp_synced       <= '0';

        if (timestamp_sync = '1') then
          timestamp_ctr        <= (others => '0');
          timestamp_synced     <= '1';
        else
          if (TRIGGER_IN = '1') then
            timestamp_hold_o   <= std_logic_vector(timestamp_ctr);
            timestamp_trigger_o <= '1';
          end if;
          timestamp_ctr        <= timestamp_ctr + 1;
        end if;
      end if;
    end if;
  end process PROC_TIMESTAMP_CTR;

  timestamp_current_o        <= timestamp_ctr;

  ------------------------------------------------------------------------------
  -- Output Signals
```

```
  ------------------------------------------------------------------------------

  pulse_dtrans_1: pulse_dtrans
    generic map (
      CLK_RATIO => 4
      )
    port map (
      CLK_A_IN    => NX_MAIN_CLK_IN,
      RESET_A_IN  => RESET_NX_MAIN_CLK_IN,
      PULSE_A_IN  => timestamp_synced,
      CLK_B_IN    => CLK_IN,
      RESET_B_IN  => RESET_IN,
      PULSE_B_OUT => timestamp_synced_o
      );

  TIMESTAMP_CURRENT_OUT     <= timestamp_current_o;
  TIMESTAMP_HOLD_OUT        <= timestamp_hold_o;
  TIMESTAMP_SYNCED_OUT      <= timestamp_synced_o;
  TIMESTAMP_TRIGGER_OUT     <= timestamp_trigger_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.nxyter_components.all;

entity nx_histogram is
  generic (
    BUS_WIDTH    : integer   := 7;
    DATA_WIDTH   : integer   := 32
    );
  port (
    CLK_IN                 : in  std_logic;
    RESET_IN               : in  std_logic;


    NUM_AVERAGES_IN        : in  unsigned(2 downto 0);
    AVERAGE_ENABLE_IN      : in  std_logic;
    CHANNEL_ID_IN          : in  std_logic_vector(BUS_WIDTH - 1 downto 0);
    CHANNEL_DATA_IN        : in  std_logic_vector(DATA_WIDTH - 1 downto 0);
    CHANNEL_ADD_IN         : in  std_logic;
    CHANNEL_WRITE_IN       : in  std_logic;
    CHANNEL_WRITE_BUSY_OUT : out std_logic;

    CHANNEL_ID_READ_IN     : in  std_logic_vector(BUS_WIDTH - 1 downto 0);
    CHANNEL_READ_IN        : in  std_logic;
    CHANNEL_DATA_OUT       : out std_logic_vector(DATA_WIDTH - 1 downto 0);
    CHANNEL_DATA_VALID_OUT : out std_logic;
    CHANNEL_READ_BUSY_OUT  : out std_logic;

    DEBUG_OUT              : out std_logic_vector(15 downto 0)
    );

end entity;

architecture Behavioral of nx_histogram is

  -- Hist Fill/Ctr Handler
  type H_STATES is (H_IDLE,
                    H_WRITEADD_CHANNEL,
```



```
                    H_WRITE_CHANNEL
                    );
  signal H_STATE, H_NEXT_STATE : H_STATES;

  signal address_hist_m          : std_logic_vector(6 downto 0);
  signal address_hist_m_x        : std_logic_vector(6 downto 0);
  signal data_hist_m             : std_logic_vector(DATA_WIDTH - 1 downto 0);
  signal data_hist_m_x           : std_logic_vector(DATA_WIDTH - 1 downto 0);

  signal read_data_hist          : std_logic_vector(DATA_WIDTH - 1 downto 0);
  signal read_data_ctr_hist      : unsigned(7 downto 0);
  signal read_address_hist       : std_logic_vector(BUS_WIDTH - 1 downto 0);
  signal read_enable_hist        : std_logic;

  signal write_data_hist         : std_logic_vector(DATA_WIDTH - 1 downto 0);
  signal write_data_ctr_hist     : unsigned(7 downto 0);
  signal write_address_hist      : std_logic_vector(BUS_WIDTH - 1 downto 0);
  signal write_enable_hist       : std_logic;

  signal write_address           : std_logic_vector(BUS_WIDTH - 1 downto 0);
  signal write_data              : std_logic_vector(DATA_WIDTH - 1 downto 0);
  signal write_enable            : std_logic;

  signal channel_write_busy_o    : std_logic;

  -- Hist Read Handler
  signal read_address            : std_logic_vector(BUS_WIDTH - 1 downto 0);
  signal read_data               : std_logic_vector(DATA_WIDTH - 1 downto 0);
  signal read_enable_p           : std_logic;
  signal read_enable             : std_logic;
  signal channel_data_o          : std_logic_vector(DATA_WIDTH - 1 downto 0);
  signal channel_data_valid_o    : std_logic;
  signal channel_data_valid_o_f  : std_logic_vector(2 downto 0);
  signal channel_read_busy_o     : std_logic;

begin

  ------------------------------------------------------------------------------

  DEBUG_OUT(0)              <= CLK_IN;
  DEBUG_OUT(1)              <= channel_write_busy_o;
  DEBUG_OUT(2)              <= CHANNEL_ADD_IN;
  DEBUG_OUT(3)              <= write_enable_hist;
  DEBUG_OUT(4)              <= channel_read_busy_o;
  DEBUG_OUT(5)              <= CHANNEL_READ_IN;
  DEBUG_OUT(6)              <= read_enable;
  DEBUG_OUT(7)              <= channel_data_valid_o;
  DEBUG_OUT(15 downto 8)    <= channel_data_o(7 downto 0);

  ------------------------------------------------------------------------------

  ram_dp_128x40_hist: ram_dp_128x40
    port map (
      WrAddress          => write_address_hist,
      RdAddress          => read_address_hist,
      Data(31 downto 0)  => write_data_hist,
      Data(39 downto 32) => write_data_ctr_hist,
      WE                 => not RESET_IN,
      RdClock            => CLK_IN,
      RdClockEn          => read_enable_hist,
      Reset              => RESET_IN,
      WrClock            => CLK_IN,
```

```
    WrClockEn          => write_enable_hist,
    Q(31 downto 0)     => read_data_hist,
    Q(39 downto 32)    => read_data_ctr_hist
    );

ram_dp_128x32_result: ram_dp_128x32
  port map (
    WrAddress => write_address,
    RdAddress => read_address,
    Data      => write_data,
    WE        => not RESET_IN,
    RdClock   => CLK_IN,
    RdClockEn => read_enable,
    Reset     => RESET_IN,
    WrClock   => CLK_IN,
    WrClockEn => write_enable,
    Q         => read_data
    );

--------------------------------------------------------------------------------
-- Memory Handler
--------------------------------------------------------------------------------

pulse_to_level_1: pulse_to_level
  generic map (
    NUM_CYCLES => 2
    )
  port map (
    CLK_IN    => CLK_IN,
    RESET_IN  => RESET_IN,
    PULSE_IN  => read_enable_p,
    LEVEL_OUT => read_enable
    );

PROC_HIST_READ: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      read_enable_p               <= '0';
      read_address                <= (others => '0');
      channel_data_valid_o_f      <= (others => '0');
      channel_data_valid_o        <= '0';
      channel_data_o              <= (others => '0');
      channel_read_busy_o         <= '0';
    else
      channel_data_valid_o_f(2)   <= '0';
      channel_data_valid_o_f(1)   <= channel_data_valid_o_f(2);
      channel_data_valid_o_f(0)   <= channel_data_valid_o_f(1);

      read_enable_p               <= '0';
      read_address                <= (others => '0');
      channel_data_o              <= (others => '0');
      channel_data_valid_o        <= '0';
      channel_read_busy_o         <= '0';

      if (CHANNEL_READ_IN = '1') then
        read_enable_p             <= '1';
        read_address              <= CHANNEL_ID_READ_IN;
        channel_data_valid_o_f(2) <= '1';
      end if;

      if (channel_data_valid_o_f(0) = '1') then
```



```
        channel_data_o            <= read_data;
        channel_data_valid_o      <= '1';
      end if;
      if (channel_data_valid_o_f = "000" and CHANNEL_READ_IN = '0') then
        channel_read_busy_o       <= '0';
      else
        channel_read_busy_o       <= '1';
      end if;
    end if;
  end if;
end process PROC_HIST_READ;

PROC_HIST_HANDLER_TRANSFER: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      address_hist_m       <= (others => '0');
      data_hist_m          <= (others => '0');
      H_STATE              <= H_IDLE;
    else
      address_hist_m       <= address_hist_m_x;
      data_hist_m          <= data_hist_m_x;
      H_STATE              <= H_NEXT_STATE;
    end if;
  end if;
end process PROC_HIST_HANDLER_TRANSFER;

PROC_HIST_HANDLER: process(H_STATE,
                           CHANNEL_ID_IN,
                           CHANNEL_DATA_IN,
                           CHANNEL_ADD_IN,
                           CHANNEL_WRITE_IN
                           )
  variable new_data           : unsigned(31 downto 0);
begin
  address_hist_m_x            <= address_hist_m;
  data_hist_m_x               <= data_hist_m;

  case H_STATE is
    when H_IDLE =>
      write_address_hist      <= (others => '0');
      write_data_hist         <= (others => '0');
      write_data_ctr_hist     <= (others => '0');
      write_enable_hist       <= '0';
      write_address           <= (others => '0');
      write_data              <= (others => '0');
      write_enable            <= '0';
      channel_write_busy_o    <= '0';

      if (CHANNEL_ADD_IN = '1') then
        read_address_hist     <= CHANNEL_ID_IN;
        read_enable_hist      <= '1';
        address_hist_m_x      <= CHANNEL_ID_IN;
        data_hist_m_x         <= CHANNEL_DATA_IN;
        H_NEXT_STATE          <= H_WRITEADD_CHANNEL;
      elsif (CHANNEL_WRITE_IN = '1') then
        read_address_hist     <= (others => '0');
        read_enable_hist      <= '0';
        address_hist_m_x      <= CHANNEL_ID_IN;
        data_hist_m_x         <= CHANNEL_DATA_IN;
        H_NEXT_STATE          <= H_WRITE_CHANNEL;
      else
```

```
        read_address_hist     <= (others => '0');
        read_enable_hist      <= '0';
        address_hist_m_x      <= (others => '0');
        data_hist_m_x         <= (others => '0');
        H_NEXT_STATE          <= H_IDLE;
      end if;

    when H_WRITEADD_CHANNEL =>
      if (AVERAGE_ENABLE_IN = '0') then
        new_data              := std_logic_vector(unsigned(read_data_hist) +
                                                  unsigned(data_hist_m));
        write_data_ctr_hist   <= read_data_ctr_hist + 1;

        write_address         <= address_hist_m;
        write_data            <= new_data;
        write_enable          <= '1';
      elsif ((read_data_ctr_hist srl to_integer(NUM_AVERAGES_IN)) > 0)
      then
        new_data              := std_logic_vector(unsigned(data_hist_m));
        write_data_ctr_hist   <= x"01";

        write_address         <= address_hist_m;
        write_data            <= new_data;
        write_enable          <= '1';
      else
        new_data              := std_logic_vector(unsigned(read_data_hist) +
                                                  unsigned(data_hist_m));
        write_data_ctr_hist <= read_data_ctr_hist + 1;

        write_address         <= (others => '0');
        write_data            <= (others => '0');
        write_enable          <= '0';
      end if;

      read_address_hist       <= (others => '0');
      read_enable_hist        <= '0';
      write_address_hist      <= address_hist_m;
      write_data_hist         <= new_data;
      write_enable_hist       <= '1';
      channel_write_busy_o    <= '1';
      H_NEXT_STATE            <= H_IDLE;

    when H_WRITE_CHANNEL =>
      new_data                := unsigned(data_hist_m);
      read_address_hist       <= (others => '0');
      read_enable_hist        <= '0';
      write_address_hist      <= address_hist_m;
      write_data_hist         <= new_data;
      write_data_ctr_hist     <= (others => '0');
      write_enable_hist       <= '1';

      write_address           <= address_hist_m;
      write_data              <= new_data;
      write_enable            <= '1';
      channel_write_busy_o    <= '1';
      H_NEXT_STATE            <= H_IDLE;

  end case;

end process PROC_HIST_HANDLER;

-------------------------------------------------------------------------------
```



```
  -- Output Signals
  -------------------------------------------------------------------------------

  CHANNEL_WRITE_BUSY_OUT    <= channel_write_busy_o;
  CHANNEL_DATA_OUT          <= channel_data_o;
  CHANNEL_DATA_VALID_OUT    <= channel_data_valid_o;
  CHANNEL_READ_BUSY_OUT     <= channel_read_busy_o;

end Behavioral;

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.nxyter_components.all;

entity nx_histograms is
  port (
    CLK_IN               : in  std_logic;
    RESET_IN             : in  std_logic;

    RESET_HISTS_IN       : in  std_logic;

    CHANNEL_FILL_IN      : in  std_logic;
    CHANNEL_ID_IN        : in  std_logic_vector(6 downto 0);
    CHANNEL_ADC_IN       : in  std_logic_vector(11 downto 0);
    CHANNEL_PILEUP_IN    : in  std_logic;
    CHANNEL_OVERFLOW_IN  : in  std_logic;

    -- Slave bus
    SLV_READ_IN          : in  std_logic;
    SLV_WRITE_IN         : in  std_logic;
    SLV_DATA_OUT         : out std_logic_vector(31 downto 0);
    SLV_DATA_IN          : in  std_logic_vector(31 downto 0);
    SLV_ADDR_IN          : in  std_logic_vector(15 downto 0);
    SLV_ACK_OUT          : out std_logic;
    SLV_NO_MORE_DATA_OUT : out std_logic;
    SLV_UNKNOWN_ADDR_OUT : out std_logic;

    DEBUG_OUT            : out std_logic_vector(15 downto 0)
    );

end entity;

architecture Behavioral of nx_histograms is

  -- Hit Histogram
  signal hit_num_averages    : unsigned(2 downto 0);
  signal hit_average_enable  : std_logic;
  signal hit_write_busy      : std_logic;
  signal hit_read_busy       : std_logic;

  signal hit_write_id        : std_logic_vector(6 downto 0);
  signal hit_write_data      : std_logic_vector(31 downto 0);
  signal hit_write           : std_logic;
  signal hit_add             : std_logic;

  signal hit_read_id         : std_logic_vector(6 downto 0);
  signal hit_read            : std_logic;
  signal hit_read_data       : std_logic_vector(31 downto 0);
  signal hit_read_data_valid : std_logic;
```

```
  -- PileUp Histogram
  signal pileup_num_averages    : unsigned(2 downto 0);
  signal pileup_average_enable  : std_logic;
  signal pileup_write_busy      : std_logic;
  signal pileup_read_busy       : std_logic;

  signal pileup_write_id        : std_logic_vector(6 downto 0);
  signal pileup_write_data      : std_logic_vector(31 downto 0);
  signal pileup_write           : std_logic;
  signal pileup_add             : std_logic;

  signal pileup_read_id         : std_logic_vector(6 downto 0);
  signal pileup_read            : std_logic;
  signal pileup_read_data       : std_logic_vector(31 downto 0);
  signal pileup_read_data_valid : std_logic;

  -- OverFlow Histogram
  signal ovfl_num_averages    : unsigned(2 downto 0);
  signal ovfl_average_enable  : std_logic;
  signal ovfl_write_busy      : std_logic;
  signal ovfl_read_busy       : std_logic;

  signal ovfl_write_id        : std_logic_vector(6 downto 0);
  signal ovfl_write_data      : std_logic_vector(31 downto 0);
  signal ovfl_write           : std_logic;
  signal ovfl_add             : std_logic;

  signal ovfl_read_id         : std_logic_vector(6 downto 0);
  signal ovfl_read            : std_logic;
  signal ovfl_read_data       : std_logic_vector(31 downto 0);
  signal ovfl_read_data_valid : std_logic;

  -- ADC Value Histogram
  signal adc_num_averages    : unsigned(2 downto 0);
  signal adc_average_enable  : std_logic;
  signal adc_write_busy      : std_logic;
  signal adc_read_busy       : std_logic;

  signal adc_write_id        : std_logic_vector(6 downto 0);
  signal adc_write_data      : std_logic_vector(31 downto 0);
  signal adc_write           : std_logic;
  signal adc_add             : std_logic;

  signal adc_read_id         : std_logic_vector(6 downto 0);
  signal adc_read            : std_logic;
  signal adc_read_data       : std_logic_vector(31 downto 0);
  signal adc_read_data_valid : std_logic;

  -- Slave Bus
  signal slv_data_out_o       : std_logic_vector(31 downto 0);
  signal slv_no_more_data_o   : std_logic;
  signal slv_unknown_addr_o   : std_logic;
  signal slv_ack_o            : std_logic;

begin

  ------------------------------------------------------------------------------


 -- DEBUG_OUT(0)              <= CLK_IN;
 -- DEBUG_OUT(1)              <= CHANNEL_FILL_IN;
```



```
-- DEBUG_OUT(2)              <= hit_write_busy;
-- DEBUG_OUT(3)              <= pileup_write_busy;
-- DEBUG_OUT(4)              <= ovfl_write_busy;
-- DEBUG_OUT(5)              <= adc_write_busy;
--
-- DEBUG_OUT(6)              <= hit_read_busy;
-- DEBUG_OUT(7)              <= pileup_read_busy;
-- DEBUG_OUT(8)              <= ovfl_read_busy;
-- DEBUG_OUT(9)              <= adc_read_busy;
-- DEBUG_OUT(15 downto 10)   <= (others => '0');


-- DEBUG_OUT(15 downto 1)    <= SLV_ADDR_IN(14 downto 0);

 ------------------------------------------------------------------------------

 nx_histogram_hits: nx_histogram
   port map (
     CLK_IN                 => CLK_IN,
     RESET_IN               => RESET_IN,

     NUM_AVERAGES_IN        => hit_num_averages,
     AVERAGE_ENABLE_IN      => hit_average_enable,
     CHANNEL_ID_IN          => hit_write_id,
     CHANNEL_DATA_IN        => hit_write_data,
     CHANNEL_ADD_IN         => hit_add,
     CHANNEL_WRITE_IN       => hit_write,
     CHANNEL_WRITE_BUSY_OUT => hit_write_busy,

     CHANNEL_ID_READ_IN     => hit_read_id,
     CHANNEL_READ_IN        => hit_read,
     CHANNEL_DATA_OUT       => hit_read_data,
     CHANNEL_DATA_VALID_OUT => hit_read_data_valid,
     CHANNEL_READ_BUSY_OUT  => hit_read_busy,

     DEBUG_OUT              => DEBUG_OUT --open
     );

 nx_histogram_adc: nx_histogram
   port map (
     CLK_IN                 => CLK_IN,
     RESET_IN               => RESET_IN,

     NUM_AVERAGES_IN        => adc_num_averages,
     AVERAGE_ENABLE_IN      => adc_average_enable,
     CHANNEL_ID_IN          => adc_write_id,
     CHANNEL_DATA_IN        => adc_write_data,
     CHANNEL_ADD_IN         => adc_add,
     CHANNEL_WRITE_IN       => adc_write,
     CHANNEL_WRITE_BUSY_OUT => adc_write_busy,

     CHANNEL_ID_READ_IN     => adc_read_id,
     CHANNEL_READ_IN        => adc_read,
     CHANNEL_DATA_OUT       => adc_read_data,
     CHANNEL_DATA_VALID_OUT => adc_read_data_valid,
     CHANNEL_READ_BUSY_OUT  => adc_read_busy,

     DEBUG_OUT              => open
     );

 nx_histogram_pileup: nx_histogram
   port map (
```

```
    CLK_IN                => CLK_IN,
    RESET_IN              => RESET_IN,

    NUM_AVERAGES_IN       => pileup_num_averages,
    AVERAGE_ENABLE_IN     => pileup_average_enable,
    CHANNEL_ID_IN         => pileup_write_id,
    CHANNEL_DATA_IN       => pileup_write_data,
    CHANNEL_ADD_IN        => pileup_add,
    CHANNEL_WRITE_IN      => pileup_write,
    CHANNEL_WRITE_BUSY_OUT => pileup_write_busy,

    CHANNEL_ID_READ_IN    => pileup_read_id,
    CHANNEL_READ_IN       => pileup_read,
    CHANNEL_DATA_OUT      => pileup_read_data,
    CHANNEL_DATA_VALID_OUT => pileup_read_data_valid,
    CHANNEL_READ_BUSY_OUT  => pileup_read_busy,

    DEBUG_OUT             => open
    );

nx_histogram_ovfl: nx_histogram
  port map (
    CLK_IN                => CLK_IN,
    RESET_IN              => RESET_IN,

    NUM_AVERAGES_IN       => ovfl_num_averages,
    AVERAGE_ENABLE_IN     => ovfl_average_enable,
    CHANNEL_ID_IN         => ovfl_write_id,
    CHANNEL_DATA_IN       => ovfl_write_data,
    CHANNEL_ADD_IN        => ovfl_add,
    CHANNEL_WRITE_IN      => ovfl_write,
    CHANNEL_WRITE_BUSY_OUT => ovfl_write_busy,

    CHANNEL_ID_READ_IN    => ovfl_read_id,
    CHANNEL_READ_IN       => ovfl_read,
    CHANNEL_DATA_OUT      => ovfl_read_data,
    CHANNEL_DATA_VALID_OUT => ovfl_read_data_valid,
    CHANNEL_READ_BUSY_OUT  => ovfl_read_busy,

    DEBUG_OUT             => open
    );

-------------------------------------------------------------------------------
-- Fill Histograms
-------------------------------------------------------------------------------

PROC_FILL_HISTOGRAMS: process(CLK_IN)
begin
  if (rising_edge(CLK_IN)) then
    if (RESET_IN = '1') then
      hit_write_id                <= (others => '0');
      hit_write_data              <= (others => '0');
      hit_write                   <= '0';
      hit_add                     <= '0';

      adc_write_id                <= (others => '0');
      adc_write_data              <= (others => '0');
      adc_write                   <= '0';
      adc_add                     <= '0';

      pileup_write_id             <= (others => '0');
      pileup_write_data           <= (others => '0');
```



```
      pileup_write                <= '0';
      pileup_add                  <= '0';

      ovfl_write_id               <= (others => '0');
      ovfl_write_data             <= (others => '0');
      ovfl_write                  <= '0';
      ovfl_add                    <= '0';
    else
      hit_write_id                <= (others => '0');
      hit_write_data              <= (others => '0');
      hit_write                   <= '0';
      hit_add                     <= '0';

      adc_write_id                <= (others => '0');
      adc_write_data              <= (others => '0');
      adc_write                   <= '0';
      adc_add                     <= '0';

      pileup_write_id             <= (others => '0');
      pileup_write_data           <= (others => '0');
      pileup_write                <= '0';
      pileup_add                  <= '0';

      ovfl_write_id               <= (others => '0');
      ovfl_write_data             <= (others => '0');
      ovfl_write                  <= '0';
      ovfl_add                    <= '0';

      if (CHANNEL_FILL_IN = '1' and  hit_write_busy = '0') then
        hit_write_id                <= CHANNEL_ID_IN;
        hit_write_data              <= x"0000_0001";
        hit_add                     <= '1';

        adc_write_id                <= CHANNEL_ID_IN;
        adc_write_data(11 downto 0)   <= CHANNEL_ADC_IN;
        adc_write_data(31 downto 12)  <= (others => '0');
        adc_add                     <= '1';

        if (CHANNEL_PILEUP_IN = '1') then
          pileup_write_id             <= CHANNEL_ID_IN;
          pileup_write_data           <= x"0000_0001";
          pileup_add                  <= '1';
        end if;

        if (CHANNEL_OVERFLOW_IN = '1') then
          ovfl_write_id               <= CHANNEL_ID_IN;
          ovfl_write_data             <= x"0000_0001";
          ovfl_add                    <= '1';
        end if;

      end if;
    end if;
  end if;
end process PROC_FILL_HISTOGRAMS;

-------------------------------------------------------------------------------
-- Trbnet Slave Bus
-------------------------------------------------------------------------------

-- Give status info to the TRB Slow Control Channel
PROC_HISTOGRAMS_READ: process(CLK_IN)
begin
```

```
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1' ) then
        slv_data_out_o        <= (others => '0');
        slv_no_more_data_o    <= '0';
        slv_unknown_addr_o    <= '0';
        slv_ack_o             <= '0';

        hit_read_id           <= (others => '0');
        hit_read              <= '0';
        hit_num_averages      <= "000";
        hit_average_enable    <= '0';

        adc_read_id           <= (others => '0');
        adc_read              <= '0';
        adc_num_averages      <= "001";
        adc_average_enable    <= '1';

        pileup_read_id        <= (others => '0');
        pileup_read           <= '0';
        pileup_num_averages   <= "000";
        pileup_average_enable <= '0';

        ovfl_read_id          <= (others => '0');
        ovfl_read             <= '0';
        ovfl_num_averages     <= "000";
        ovfl_average_enable   <= '0';
      else
        slv_data_out_o        <= (others => '0');
        slv_unknown_addr_o    <= '0';
        slv_no_more_data_o    <= '0';

        hit_read_id           <= (others => '0');
        hit_read              <= '0';
        adc_read_id           <= (others => '0');
        adc_read              <= '0';
        pileup_read_id        <= (others => '0');
        pileup_read           <= '0';
        ovfl_read_id          <= (others => '0');
        ovfl_read             <= '0';

        if (hit_read_busy    = '1' or
            adc_read_busy    = '1' or
            pileup_read_busy = '1' or
            ovfl_read_busy   = '1' ) then
          if (hit_read_data_valid = '1') then
            slv_data_out_o                  <= hit_read_data;
            slv_ack_o                       <= '1';
          elsif (adc_read_data_valid = '1') then
            slv_data_out_o                  <= adc_read_data;
            slv_ack_o                       <= '1';
          elsif (pileup_read_data_valid = '1') then
            slv_data_out_o                  <= pileup_read_data;
            slv_ack_o                       <= '1';
          elsif (ovfl_read_data_valid = '1') then
            slv_data_out_o                  <= ovfl_read_data;
            slv_ack_o                       <= '1';
          else
            slv_ack_o                       <= '0';
          end if;

        elsif (SLV_READ_IN  = '1') then
          if (unsigned(SLV_ADDR_IN) >= x"0000" and
```



```
              unsigned(SLV_ADDR_IN) <= x"007f") then
            hit_read_id                     <= SLV_ADDR_IN(6 downto 0);
            hit_read                        <= '1';
            slv_ack_o                       <= '0';
          elsif (unsigned(SLV_ADDR_IN) >= x"0100" and
                 unsigned(SLV_ADDR_IN) <= x"017f") then
            pileup_read_id                  <= SLV_ADDR_IN(6 downto 0);
            pileup_read                     <= '1';
            slv_ack_o                       <= '0';
          elsif (unsigned(SLV_ADDR_IN) >= x"0200" and
                 unsigned(SLV_ADDR_IN) <= x"027f") then
            ovfl_read_id                    <= SLV_ADDR_IN(6 downto 0);
            ovfl_read                       <= '1';
            slv_ack_o                       <= '0';
          elsif (unsigned(SLV_ADDR_IN) >= x"0300" and
                 unsigned(SLV_ADDR_IN) <= x"037f") then
            adc_read_id                     <= SLV_ADDR_IN(6 downto 0);
            adc_read                        <= '1';
            slv_ack_o                       <= '0';
          else
            case SLV_ADDR_IN is
              when x"0080" =>
                slv_data_out_o(2 downto 0)   <=
                  std_logic_vector(hit_num_averages);
                slv_data_out_o(31 downto 3)  <= (others => '0');
                slv_ack_o                    <= '1';

              when x"0081" =>
                slv_data_out_o(0)            <= hit_average_enable;
                slv_data_out_o(31 downto 1)  <= (others => '0');
                slv_ack_o                    <= '1';

              when x"0180" =>
                slv_data_out_o(2 downto 0)   <=
                  std_logic_vector(pileup_num_averages);
                slv_data_out_o(31 downto 3)  <= (others => '0');
                slv_ack_o                    <= '1';

              when x"0181" =>
                slv_data_out_o(0)            <= pileup_average_enable;
                slv_data_out_o(31 downto 1)  <= (others => '0');
                slv_ack_o                    <= '1';

              when x"0280" =>
                slv_data_out_o(2 downto 0)   <=
                  std_logic_vector(ovfl_num_averages);
                slv_data_out_o(31 downto 3)  <= (others => '0');
                slv_ack_o                    <= '1';

              when x"0281" =>
                slv_data_out_o(0)            <= ovfl_average_enable;
                slv_data_out_o(31 downto 1)  <= (others => '0');
                slv_ack_o                    <= '1';

              when x"0380" =>
                slv_data_out_o(2 downto 0)   <=
                  std_logic_vector(adc_num_averages);
                slv_data_out_o(31 downto 3)  <= (others => '0');
                slv_ack_o                    <= '1';

              when x"0381" =>
                slv_data_out_o(0)            <= adc_average_enable;
```

```
              slv_data_out_o(31 downto 1)  <= (others => '0');
              slv_ack_o                    <= '1';

            when others =>
              slv_unknown_addr_o           <= '1';
              slv_ack_o                    <= '0';

          end case;
        end if;

      elsif (SLV_WRITE_IN  = '1') then

        case SLV_ADDR_IN is
          when x"0080" =>
            hit_num_averages               <= SLV_DATA_IN(2 downto 0);
            slv_ack_o                      <= '1';

          when x"0081" =>
            hit_average_enable             <= SLV_DATA_IN(0);
            slv_ack_o                      <= '1';

          when x"0180" =>
            pileup_num_averages            <= SLV_DATA_IN(2 downto 0);
            slv_ack_o                      <= '1';

          when x"0181" =>
            pileup_average_enable          <= SLV_DATA_IN(0);
            slv_ack_o                      <= '1';

          when x"0280" =>
            ovfl_num_averages              <= SLV_DATA_IN(2 downto 0);
            slv_ack_o                      <= '1';

          when x"0281" =>
            ovfl_average_enable             <= SLV_DATA_IN(0);
            slv_ack_o                      <= '1';

          when x"0380" =>
            adc_num_averages               <= SLV_DATA_IN(2 downto 0);
            slv_ack_o                      <= '1';

          when x"0381" =>
            adc_average_enable             <= SLV_DATA_IN(0);
            slv_ack_o                      <= '1';

          when others =>
            slv_unknown_addr_o             <= '1';
            slv_ack_o                      <= '0';

        end case;
      else
        slv_ack_o                          <= '0';
      end if;
    end if;
  end if;
end process PROC_HISTOGRAMS_READ;

-------------------------------------------------------------------------------
-- Output Signals
-------------------------------------------------------------------------------

-- Slave
```



```
  SLV_DATA_OUT          <= slv_data_out_o;
  SLV_NO_MORE_DATA_OUT  <= slv_no_more_data_o;
  SLV_UNKNOWN_ADDR_OUT  <= slv_unknown_addr_o;
  SLV_ACK_OUT           <= slv_ack_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.nxyter_components.all;

entity nx_i2c_master is
  generic (
    I2C_SPEED : unsigned(11 downto 0) := x"3e8"
    );
  port(
    CLK_IN               : in    std_logic;
    RESET_IN             : in    std_logic;

    -- I2C connections
    SDA_INOUT            : inout std_logic;
    SCL_INOUT            : inout std_logic;

    -- Internal Interface
    INTERNAL_COMMAND_IN  : in    std_logic_vector(31 downto 0);
    COMMAND_BUSY_OUT     : out   std_logic;
    I2C_DATA_OUT         : out   std_logic_vector(31 downto 0);
    I2C_DATA_BYTES_OUT   : out   std_logic_vector(31 downto 0);
    I2C_LOCK_IN          : in    std_logic;

    -- Slave bus
    SLV_READ_IN          : in    std_logic;
    SLV_WRITE_IN         : in    std_logic;
    SLV_DATA_OUT         : out   std_logic_vector(31 downto 0);
    SLV_DATA_IN          : in    std_logic_vector(31 downto 0);
    SLV_ADDR_IN          : in    std_logic_vector(15 downto 0);
    SLV_ACK_OUT          : out   std_logic;
    SLV_NO_MORE_DATA_OUT : out   std_logic;
    SLV_UNKNOWN_ADDR_OUT : out   std_logic;

    -- Debug Line
    DEBUG_OUT            : out std_logic_vector(15 downto 0)
    );
end entity;

architecture Behavioral of nx_i2c_master is

  signal sda_o                : std_logic;
  signal scl_o                : std_logic;

  signal sda_i                : std_logic;
  signal sda_x                : std_logic;
  signal sda                  : std_logic;

  signal scl_i                : std_logic;
  signal scl_x                : std_logic;
  signal scl                  : std_logic;
  signal command_busy_o       : std_logic;

  -- I2C Master
```

```
  signal sda_master            : std_logic;
  signal scl_master            : std_logic;
  signal i2c_start             : std_logic;
  signal i2c_busy              : std_logic;
  signal startstop_select      : std_logic;
  signal startstop_seq_start   : std_logic;
  signal sendbyte_seq_start    : std_logic;
  signal readbyte_seq_start    : std_logic;
  signal sendbyte_byte         : std_logic_vector(7 downto 0);
  signal read_seq_ctr          : std_logic;
  signal i2c_data              : std_logic_vector(31 downto 0);
  signal i2c_bytes             : std_logic_vector(31 downto 0);

  signal i2c_busy_x            : std_logic;
  signal startstop_select_x    : std_logic;
  signal startstop_seq_start_x : std_logic;
  signal sendbyte_seq_start_x  : std_logic;
  signal sendbyte_byte_x       : std_logic_vector(7 downto 0);
  signal readbyte_seq_start_x  : std_logic;
  signal read_seq_ctr_x        : std_logic;
  signal i2c_data_x            : std_logic_vector(31 downto 0);
  signal i2c_bytes_x           : std_logic_vector(31 downto 0);

  signal sda_startstop         : std_logic;
  signal scl_startstop         : std_logic;
  signal i2c_notready          : std_logic;
  signal startstop_done        : std_logic;

  signal sda_sendbyte          : std_logic;
  signal scl_sendbyte          : std_logic;
  signal sendbyte_ack          : std_logic;
  signal sendbyte_done         : std_logic;

  signal sda_readbyte          : std_logic;
  signal scl_readbyte          : std_logic;
  signal readbyte_byte         : std_logic_vector(31 downto 0);
  signal readbyte_done         : std_logic;

  type STATES is (S_RESET,
                  S_IDLE,
                  S_START,
                  S_START_WAIT,

                  S_SEND_CHIP_ID,
                  S_SEND_CHIP_ID_WAIT,
                  S_SEND_REGISTER,
                  S_SEND_REGISTER_WAIT,
                  S_SEND_DATA,
                  S_SEND_DATA_WAIT,
                  S_GET_DATA,
                  S_GET_DATA_WAIT,

                  S_STOP,
                  S_STOP_WAIT
                  );
  signal STATE, NEXT_STATE : STATES;

  -- TRBNet Slave Bus
  signal slv_data_out_o           : std_logic_vector(31 downto 0);
  signal slv_no_more_data_o       : std_logic;
  signal slv_unknown_addr_o       : std_logic;
  signal slv_ack_o                : std_logic;
```



```
  signal i2c_chipid               : std_logic_vector(6 downto 0);
  signal i2c_rw_bit               : std_logic;
  signal i2c_num_bytes            : unsigned(2 downto 0);
  signal i2c_registerid           : std_logic_vector(7 downto 0);
  signal i2c_register_data        : std_logic_vector(7 downto 0);
  signal i2c_register_value_read  : std_logic_vector(7 downto 0);

  signal disable_slave_bus        : std_logic;
  signal internal_command         : std_logic;
  signal internal_command_d       : std_logic;
  signal i2c_data_internal_o      : std_logic_vector(31 downto 0);
  signal i2c_data_internal_bytes_o : std_logic_vector(31 downto 0);
  signal i2c_data_slave           : std_logic_vector(31 downto 0);

begin

  -- Debug
  DEBUG_OUT(0)            <= CLK_IN;
  DEBUG_OUT(3 downto 1)   <= i2c_num_bytes; --i2c_data(7 downto 0);
  DEBUG_OUT(4)            <= startstop_seq_start;
  DEBUG_OUT(5)            <= readbyte_seq_start;
  DEBUG_OUT(6)            <= startstop_done;
  DEBUG_OUT(7)            <= sendbyte_done;
  DEBUG_OUT(8)            <= readbyte_done;

  --DEBUG_OUT(10 downto 9)  <= i2c_data(31 downto 30);
  DEBUG_OUT(9)            <= i2c_busy;
  DEBUG_OUT(10)           <= i2c_busy;
  DEBUG_OUT(11)           <= i2c_busy;
  DEBUG_OUT(12)           <= sda_o;
  DEBUG_OUT(13)           <= scl_o;
  DEBUG_OUT(14)           <= sda;
  DEBUG_OUT(15)           <= scl;
  --DEBUG_OUT(12 downto 9)  <= i2c_data(31 downto 28);

  -- Start / Stop Sequence
  nx_i2c_startstop_1: nx_i2c_startstop
    generic map (
      I2C_SPEED => I2C_SPEED
      )
    port map (
      CLK_IN            => CLK_IN,
      RESET_IN          => RESET_IN,
      START_IN          => startstop_seq_start,
      SELECT_IN         => startstop_select,
      SEQUENCE_DONE_OUT => startstop_done,
      SDA_OUT           => sda_startstop,
      SCL_OUT           => scl_startstop,
      NREADY_OUT        => i2c_notready
      );

  nx_i2c_sendbyte_1: nx_i2c_sendbyte
    generic map (
      I2C_SPEED => I2C_SPEED
      )
    port map (
      CLK_IN            => CLK_IN,
      RESET_IN          => RESET_IN,
      START_IN          => sendbyte_seq_start,
      BYTE_IN           => sendbyte_byte,
      SEQUENCE_DONE_OUT => sendbyte_done,
```

```
    SDA_OUT           => sda_sendbyte,
    SCL_OUT           => scl_sendbyte,
    SDA_IN            => sda,
    SCL_IN            => scl,
    ACK_OUT           => sendbyte_ack
    );

nx_i2c_readbyte_1: nx_i2c_readbyte
  generic map (
    I2C_SPEED => I2C_SPEED
    )
  port map (
    CLK_IN            => CLK_IN,
    RESET_IN          => RESET_IN,
    START_IN          => readbyte_seq_start,
    NUM_BYTES_IN      => i2c_num_bytes,
    BYTE_OUT          => readbyte_byte,
    SEQUENCE_DONE_OUT => readbyte_done,
    SDA_OUT           => sda_readbyte,
    SCL_OUT           => scl_readbyte,
    SDA_IN            => sda
    );

 -- Sync I2C Lines
sda_i <= SDA_INOUT;
scl_i <= SCL_INOUT;

PROC_I2C_LINES_SYNC: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      sda_x <= '1';
      sda   <= '1';

      scl_x <= '1';
      scl   <= '1';
    else
      sda_x <= sda_i;
      sda   <= sda_x;

      scl_x <= scl_i;
      scl   <= scl_x;
    end if;
  end if;
end process PROC_I2C_LINES_SYNC;

PROC_I2C_MASTER_TRANSFER: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      i2c_busy              <= '1';
      startstop_select      <= '0';
      startstop_seq_start   <= '0';
      sendbyte_seq_start    <= '0';
      readbyte_seq_start    <= '0';
      sendbyte_byte         <= (others => '0');
      i2c_data              <= (others => '0');
      i2c_bytes             <= (others => '0');
      read_seq_ctr          <= '0';
      STATE                 <= S_RESET;
    else
      i2c_busy              <= i2c_busy_x;
```



```
      startstop_select      <= startstop_select_x;
      startstop_seq_start   <= startstop_seq_start_x;
      sendbyte_seq_start    <= sendbyte_seq_start_x;
      readbyte_seq_start    <= readbyte_seq_start_x;
      sendbyte_byte         <= sendbyte_byte_x;
      i2c_data              <= i2c_data_x;
      i2c_bytes             <= i2c_bytes_x;
      read_seq_ctr          <= read_seq_ctr_x;
      STATE                 <= NEXT_STATE;
    end if;
  end if;
end process PROC_I2C_MASTER_TRANSFER;


PROC_I2C_MASTER: process(STATE,
                         i2c_start,
                         startstop_done,
                         read_seq_ctr,
                         sendbyte_done,
                         sendbyte_ack,
                         readbyte_done,
                         startstop_done
                         )

begin
  -- Defaults
  sda_master                <= '1';
  scl_master                <= '1';
  i2c_busy_x                <= '1';
  startstop_select_x        <= '0';
  startstop_seq_start_x     <= '0';
  sendbyte_seq_start_x      <= '0';
  sendbyte_byte_x           <= (others => '0');
  readbyte_seq_start_x      <= '0';
  i2c_data_x                <= i2c_data;
  i2c_bytes_x               <= i2c_bytes;
  read_seq_ctr_x            <= read_seq_ctr;

  case STATE is

    when S_RESET =>
      i2c_data_x              <= (others => '0');
      i2c_bytes_x             <= (others => '0');
      NEXT_STATE              <= S_IDLE;

    when S_IDLE =>
      if (i2c_start = '1') then
        i2c_data_x            <= x"8000_0000"; -- Set Running, clear all
                                                -- other bits
        NEXT_STATE            <= S_START;
      else
        i2c_busy_x            <= '0';
        i2c_data_x            <= i2c_data and x"7fff_ffff";  -- clear running
                                                            -- bit;
        read_seq_ctr_x        <= '0';
        NEXT_STATE            <= S_IDLE;
      end if;

      -- I2C START Sequence
    when S_START =>
      startstop_select_x      <= '1';
      startstop_seq_start_x   <= '1';
```

```
        NEXT_STATE              <= S_START_WAIT;

      when S_START_WAIT =>
        if (startstop_done = '0') then
          NEXT_STATE            <= S_START_WAIT;
        else
          sda_master            <= '0';
          scl_master            <= '0';
          NEXT_STATE            <= S_SEND_CHIP_ID;
        end if;

        -- I2C SEND ChipId Sequence
      when S_SEND_CHIP_ID =>
        scl_master                   <= '0';
        sendbyte_byte_x(7 downto 1) <= i2c_chipid;
        if (read_seq_ctr = '0') then
          sendbyte_byte_x(0)    <= '0';
        else
          sendbyte_byte_x(0)    <= '1';
        end if;
        sendbyte_seq_start_x    <= '1';
        NEXT_STATE              <= S_SEND_CHIP_ID_WAIT;

      when S_SEND_CHIP_ID_WAIT =>
        if (sendbyte_done = '0') then
          NEXT_STATE            <= S_SEND_CHIP_ID_WAIT;
        else
          scl_master            <= '0';
          if (sendbyte_ack = '0') then
            i2c_data_x          <= i2c_data or x"0100_0000";
            NEXT_STATE          <= S_STOP;
          else
            if (read_seq_ctr = '0') then
              read_seq_ctr_x    <= '1';
              NEXT_STATE        <= S_SEND_REGISTER;
            else
              NEXT_STATE        <= S_GET_DATA;
            end if;
          end if;
        end if;

        -- I2C SEND RegisterId
      when S_SEND_REGISTER =>
        scl_master              <= '0';
        sendbyte_byte_x         <= i2c_registerid;
        sendbyte_seq_start_x    <= '1';
        NEXT_STATE              <= S_SEND_REGISTER_WAIT;

      when S_SEND_REGISTER_WAIT =>
        if (sendbyte_done = '0') then
          NEXT_STATE            <= S_SEND_REGISTER_WAIT;
        else
          scl_master            <= '0';
          if (sendbyte_ack = '0') then
            i2c_data_x          <= i2c_data or x"0200_0000";
            NEXT_STATE          <= S_STOP;
          else
            if (i2c_rw_bit = '0') then
              NEXT_STATE        <= S_SEND_DATA;
            else
              NEXT_STATE        <= S_START;
            end if;
```



```
          end if;
        end if;

        -- I2C SEND DataWord
      when S_SEND_DATA =>
        scl_master              <= '0';
        sendbyte_byte_x         <= i2c_register_data;
        sendbyte_seq_start_x    <= '1';
        NEXT_STATE              <= S_SEND_DATA_WAIT;

      when S_SEND_DATA_WAIT =>
        if (sendbyte_done = '0') then
          NEXT_STATE            <= S_SEND_DATA_WAIT;
        else
          scl_master            <= '0';
          if (sendbyte_ack = '0') then
            i2c_data_x          <= i2c_data or x"0400_0000";
          end if;
          NEXT_STATE            <= S_STOP;
        end if;

        -- I2C GET DataWord
      when S_GET_DATA =>
        scl_master              <= '0';
        readbyte_seq_start_x    <= '1';
        NEXT_STATE              <= S_GET_DATA_WAIT;

      when S_GET_DATA_WAIT =>
        if (readbyte_done = '0') then
          NEXT_STATE            <= S_GET_DATA_WAIT;
        else
          scl_master                    <= '0';
          i2c_data_x(7 downto 0)<= readbyte_byte(7 downto 0);
          i2c_bytes_x           <= readbyte_byte;
          NEXT_STATE            <= S_STOP;
        end if;

        -- I2C STOP Sequence
      when S_STOP =>
        sda_master              <= '0';
        scl_master              <= '0';
        startstop_select_x      <= '0';
        startstop_seq_start_x   <= '1';
        NEXT_STATE              <= S_STOP_WAIT;

      when S_STOP_WAIT =>
        if (startstop_done = '0') then
          NEXT_STATE            <= S_STOP_WAIT;
        else
          i2c_data_x            <= i2c_data or x"4000_0000"; -- Set DONE Bit
          NEXT_STATE            <= S_IDLE;
        end if;

    end case;
  end process PROC_I2C_MASTER;

  PROC_I2C_DATA_MULTIPLEXER: process(CLK_IN)
  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1' ) then
        i2c_data_internal_o       <= (others => '0');
        i2c_data_internal_bytes_o <= (others => '0');
```

```
      i2c_data_slave           <= (others => '0');
      command_busy_o           <= '0';
    else
      if (internal_command = '0' and internal_command_d = '0') then
        i2c_data_slave           <= i2c_data;
      else
        i2c_data_internal_o      <= i2c_data;
        i2c_data_internal_bytes_o <= i2c_bytes;
      end if;
    end if;
    command_busy_o               <= i2c_busy;
  end if;
end process PROC_I2C_DATA_MULTIPLEXER;

------------------------------------------------------------------------------
-- TRBNet Slave Bus
------------------------------------------------------------------------------
--
--   Write bit definition
--   ====================
--
--   D[31]    I2C_GO          0 => don't do anything on I2C,
--                            1 => start I2C access
--   D[30]    I2C_ACTION      0 => write byte, 1 => read byte
--   D[29:27] RESERVED        set all to '0'
--   D[26:24] I2C_NUM_BYTES   number of bytes to be read 1..4
--   D[23:16] I2C_ADDRESS     address of I2C chip
--   D[15:8]  I2C_REG_ADDRESS command byte for access
--   D[7:0]   I2C_DATA        data to be written
--
--   Read bit definition
--   ===================
--
--   D[31]    RUNNING         whatever
--   D[30]    I2C_DONE        whatever
--   D[29]    ERROR_RADDACK   no acknowledge for repeated address byte
--   D[28]    ERROR_RSTART    generation of repeated START condition failed
--   D[27]    ERROR_DATACK    no acknowledge for data byte
--   D[26]    ERROR_CMDACK    no acknowledge for command byte
--   D[25]    ERROR_ADDACK    no acknowledge for address byte
--   D[24]    ERROR_START     generation of START condition failed
--   D[23:21] reserved        reserved
--   D[20:16] debug           subject to change, don't use
--   D[15:8]  reserved        reserved
--   D[7:0]   I2C_DATA        result of I2C read operation
--

PROC_SLAVE_BUS: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      slv_data_out_o     <= (others => '0');
      slv_no_more_data_o <= '0';
      slv_unknown_addr_o <= '0';
      slv_ack_o          <= '0';
      i2c_start          <= '0';
      internal_command   <= '0';
      internal_command_d <= '0';

      i2c_chipid             <= (others => '0');
      i2c_rw_bit             <= '0';
      i2c_num_bytes          <= "001";
```



```
      i2c_registerid          <= (others => '0');
      i2c_register_data       <= (others => '0');
      i2c_register_value_read <= (others => '0');

    else
      slv_unknown_addr_o     <= '0';
      slv_no_more_data_o     <= '0';
      slv_data_out_o         <= (others => '0');
      i2c_start              <= '0';

      internal_command_d     <= internal_command;

      if (i2c_busy = '0' and internal_command_d = '1') then
        internal_command     <= '0';
        slv_ack_o            <= '0';

      elsif (i2c_busy = '0' and INTERNAL_COMMAND_IN(31) = '1') then
        -- Internal Interface Command
        i2c_rw_bit           <= INTERNAL_COMMAND_IN(30);
        i2c_num_bytes        <= unsigned(INTERNAL_COMMAND_IN(26 downto 24));
        i2c_chipid           <= INTERNAL_COMMAND_IN(22 downto 16);
        i2c_registerid       <= INTERNAL_COMMAND_IN(15 downto 8);
        i2c_register_data    <= INTERNAL_COMMAND_IN(7 downto 0);
        i2c_start            <= '1';
        internal_command     <= '1';
        slv_ack_o            <= '0';

      elsif (SLV_WRITE_IN  = '1') then
        case SLV_ADDR_IN is
          when x"0000" =>
            if (internal_command = '0' and
                I2C_LOCK_IN      = '0' and
                i2c_busy         = '0' and
                SLV_DATA_IN(31)  = '1') then
              i2c_rw_bit         <= SLV_DATA_IN(30);
              if (SLV_DATA_IN(29 downto 24) = "111111") then
                i2c_num_bytes    <= "001";
              else
                i2c_num_bytes    <= unsigned(SLV_DATA_IN(26 downto 24));
              end if;
              i2c_chipid         <= SLV_DATA_IN(22 downto 16);
              i2c_registerid     <= SLV_DATA_IN(15 downto 8);
              i2c_register_data  <= SLV_DATA_IN(7 downto 0);
              i2c_start          <= '1';
              slv_ack_o          <= '1';
            else
              slv_no_more_data_o <= '1';
              slv_ack_o          <= '0';
            end if;

          when others =>
            slv_unknown_addr_o   <= '1';
            slv_ack_o            <= '0';
        end case;

      elsif (SLV_READ_IN = '1') then
        case SLV_ADDR_IN is
          when x"0000" =>
            if (internal_command = '0' and
                I2C_LOCK_IN      = '0' and
                i2c_busy         = '0') then
              slv_data_out_o     <= i2c_data_slave;
```

```
              slv_ack_o          <= '1';
            else
              slv_data_out_o     <= (others => '0');
              slv_no_more_data_o <= '1';
              slv_ack_o          <= '0';
            end if;

          when x"0001" =>
            slv_data_out_o       <= i2c_bytes;
            slv_ack_o            <= '1';

          when others =>
            slv_unknown_addr_o   <= '1';
            slv_ack_o            <= '0';
        end case;

      else
        slv_ack_o            <= '0';
      end if;

    end if;
  end if;
end process PROC_SLAVE_BUS;


-------------------------------------------------------------------------------
-- Output Signals
-------------------------------------------------------------------------------

-- I2C Outputs
sda_o                <= (sda_master    and
                         sda_startstop and
                         sda_sendbyte  and
                         sda_readbyte
                         );
SDA_INOUT            <= '0' when (sda_o = '0') else 'Z';

scl_o                <= (scl_master    and
                         scl_startstop and
                         scl_sendbyte  and
                         scl_readbyte
                         );
SCL_INOUT            <= '0' when (scl_o = '0') else 'Z';

COMMAND_BUSY_OUT     <= command_busy_o;
I2C_DATA_OUT         <= i2c_data_internal_o;
I2C_DATA_BYTES_OUT   <= i2c_data_internal_bytes_o;

-- Slave Bus
SLV_DATA_OUT         <= slv_data_out_o;
SLV_NO_MORE_DATA_OUT <= slv_no_more_data_o;
SLV_UNKNOWN_ADDR_OUT <= slv_unknown_addr_o;
SLV_ACK_OUT          <= slv_ack_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.nxyter_components.all;
```



```
entity nx_i2c_readbyte is
  generic (
    I2C_SPEED : unsigned(11 downto 0) := x"3e8"
    );
  port(
    CLK_IN               : in  std_logic;
    RESET_IN             : in  std_logic;

    START_IN             : in  std_logic;
    NUM_BYTES_IN         : in  unsigned(2 downto 0);
    BYTE_OUT             : out std_logic_vector(31 downto 0);
    SEQUENCE_DONE_OUT    : out std_logic;

    -- I2C connections
    SDA_OUT              : out std_logic;
    SCL_OUT              : out std_logic;
    SDA_IN               : in  std_logic
    );
end entity;

architecture Behavioral of nx_i2c_readbyte is

  -- Send Byte
  signal sda_o              : std_logic;
  signal scl_o              : std_logic;
  signal i2c_start          : std_logic;

  signal sequence_done_o    : std_logic;
  signal i2c_data           : unsigned(31 downto 0);
  signal bit_ctr            : unsigned(3 downto 0);
  signal i2c_ack_o          : std_logic;
  signal byte_ctr           : unsigned(2 downto 0);
  signal wait_timer_start   : std_logic;
  signal wait_timer_init    : unsigned(11 downto 0);

  signal sequence_done_o_x  : std_logic;
  signal i2c_data_x         : unsigned(31 downto 0);
  signal bit_ctr_x          : unsigned(3 downto 0);
  signal i2c_ack_o_x        : std_logic;
  signal byte_ctr_x         : unsigned(2 downto 0);
  signal wait_timer_start_x : std_logic;
  signal wait_timer_init_x  : unsigned(11 downto 0);

  type STATES is (S_IDLE,
                  S_INIT,
                  S_INIT_WAIT,

                  S_READ_BYTE,
                  S_UNSET_SCL1,
                  S_SET_SCL1,
                  S_GET_BIT,
                  S_SET_SCL2,
                  S_UNSET_SCL2,
                  S_NEXT_BIT,

                  S_ACK_SET,
                  S_ACK_SET_SCL,
                  S_ACK_UNSET_SCL,

                  S_NACK_SET,
                  S_NACK_SET_SCL,
```

```
                  S_NACK_UNSET_SCL
                  );
  signal STATE, NEXT_STATE : STATES;

  -- Wait Timer
  signal wait_timer_done    : std_logic;

begin

  -- Timer
  timer_1: timer
    generic map(
      CTR_WIDTH => 12
      )
    port map (
      CLK_IN         => CLK_IN,
      RESET_IN       => RESET_IN,
      TIMER_START_IN => wait_timer_start,
      TIMER_END_IN   => wait_timer_init,
      TIMER_DONE_OUT => wait_timer_done
      );


  PROC_READ_BYTE_TRANSFER: process(CLK_IN)
  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1' ) then
        sequence_done_o  <= '0';
        i2c_data         <= (others => '0');
        bit_ctr          <= (others => '0');
        i2c_ack_o        <= '0';
        byte_ctr         <= (others => '0');
        wait_timer_start <= '0';
        wait_timer_init  <= (others => '0');
        STATE            <= S_IDLE;
      else
        sequence_done_o  <= sequence_done_o_x;
        i2c_data         <= i2c_data_x;
        bit_ctr          <= bit_ctr_x;
        i2c_ack_o        <= i2c_ack_o_x;
        byte_ctr         <= byte_ctr_x;
        wait_timer_start <= wait_timer_start_x;
        wait_timer_init  <= wait_timer_init_x;
        STATE            <= NEXT_STATE;
      end if;
    end if;
  end process PROC_READ_BYTE_TRANSFER;

  PROC_READ_BYTE: process(STATE,
                          START_IN,
                          wait_timer_done,
                          bit_ctr
                          )
  begin
    sda_o             <= '1';
    scl_o             <= '1';
    sequence_done_o_x <= '0';
    i2c_data_x        <= i2c_data;
    bit_ctr_x         <= bit_ctr;
    i2c_ack_o_x       <= i2c_ack_o;
    byte_ctr_x        <= byte_ctr;
    wait_timer_init_x <= wait_timer_init;
```



```
    wait_timer_start_x <= '0';

    case STATE is
      when S_IDLE =>
        if (START_IN = '1') then
          sda_o              <= '0';
          scl_o              <= '0';
          i2c_data_x         <= (others => '0');
          byte_ctr_x         <= (others => '0');
          NEXT_STATE         <= S_INIT;
        else
          NEXT_STATE         <= S_IDLE;
        end if;

        -- INIT
      when S_INIT =>
        sda_o                <= '0';
        scl_o                <= '0';
        wait_timer_start_x   <= '1';
        wait_timer_init_x    <= I2C_SPEED srl 1;
        NEXT_STATE           <= S_INIT_WAIT;

      when S_INIT_WAIT =>
        sda_o                <= '0';
        scl_o                <= '0';
        if (wait_timer_done = '0') then
          NEXT_STATE         <= S_INIT_WAIT;
        else
          NEXT_STATE         <= S_READ_BYTE;
        end if;

        -- I2C Read byte
      when S_READ_BYTE =>
        scl_o                <= '0';
        bit_ctr_x            <= x"7";
        byte_ctr_x           <= byte_ctr + 1;
        wait_timer_start_x   <= '1';
        wait_timer_init_x    <= I2C_SPEED srl 2;
        NEXT_STATE           <= S_UNSET_SCL1;

      when S_UNSET_SCL1 =>
        scl_o <= '0';
        if (wait_timer_done = '0') then
          NEXT_STATE         <= S_UNSET_SCL1;
        else
          wait_timer_start_x <= '1';
          wait_timer_init_x  <= I2C_SPEED srl 2;
          NEXT_STATE         <= S_SET_SCL1;
        end if;

      when S_SET_SCL1 =>
        if (wait_timer_done = '0') then
          NEXT_STATE         <= S_SET_SCL1;
        else
          wait_timer_start_x <= '1';
          wait_timer_init_x  <= I2C_SPEED srl 2;
          NEXT_STATE         <= S_GET_BIT;
        end if;

      when S_GET_BIT =>
        i2c_data_x(0)           <= SDA_IN;
        i2c_data_x(31 downto 1) <= i2c_data(30 downto 0);
```

```
        NEXT_STATE              <= S_SET_SCL2;

      when S_SET_SCL2 =>
        if (wait_timer_done = '0') then
          NEXT_STATE <= S_SET_SCL2;
        else
          wait_timer_start_x <= '1';
          wait_timer_init_x  <= I2C_SPEED srl 2;
          NEXT_STATE         <= S_UNSET_SCL2;
        end if;

      when S_UNSET_SCL2 =>
        scl_o                <= '0';
        if (wait_timer_done = '0') then
          NEXT_STATE         <= S_UNSET_SCL2;
        else
          NEXT_STATE         <= S_NEXT_BIT;
        end if;

      when S_NEXT_BIT =>
        scl_o                <= '0';
        if (bit_ctr > 0) then
          bit_ctr_x          <= bit_ctr - 1;
          wait_timer_start_x <= '1';
          wait_timer_init_x  <= I2C_SPEED srl 2;
          NEXT_STATE         <= S_UNSET_SCL1;
        else
          if (byte_ctr < NUM_BYTES_IN) then
            wait_timer_start_x <= '1';
            wait_timer_init_x  <= I2C_SPEED srl 2;
            NEXT_STATE         <= S_ACK_SET;
          else
            wait_timer_start_x <= '1';
            wait_timer_init_x  <= I2C_SPEED srl 2;
            NEXT_STATE         <= S_NACK_SET;
          end if;
        end if;

        -- I2C Send ACK (ACK) Sequence to tell client to read next byte
      when S_ACK_SET =>
        sda_o                <= '0';
        scl_o                <= '0';
        if (wait_timer_done = '0') then
          NEXT_STATE         <= S_ACK_SET;
        else
          wait_timer_start_x <= '1';
          wait_timer_init_x  <= I2C_SPEED srl 1;
          NEXT_STATE         <= S_ACK_SET_SCL;
        end if;

      when S_ACK_SET_SCL =>
        sda_o                <= '0';
        if (wait_timer_done = '0') then
          NEXT_STATE         <= S_ACK_SET_SCL;
        else
          wait_timer_start_x <= '1';
          wait_timer_init_x  <= I2C_SPEED srl 2;
          NEXT_STATE         <= S_ACK_UNSET_SCL;
        end if;

      when S_ACK_UNSET_SCL =>
        sda_o                <= '0';
```



```
        scl_o                <= '0';
        if (wait_timer_done = '0') then
          NEXT_STATE         <= S_ACK_UNSET_SCL;
        else
          NEXT_STATE         <= S_READ_BYTE;
        end if;

        -- I2C Send NOT_ACK (NACK) Sequence to tell client to release the bus
      when S_NACK_SET =>
        scl_o                <= '0';
        if (wait_timer_done = '0') then
          NEXT_STATE         <= S_NACK_SET;
        else
          wait_timer_start_x <= '1';
          wait_timer_init_x  <= I2C_SPEED srl 1;
          NEXT_STATE         <= S_NACK_SET_SCL;
        end if;

      when S_NACK_SET_SCL =>
        if (wait_timer_done = '0') then
          NEXT_STATE         <= S_NACK_SET_SCL;
        else
          wait_timer_start_x <= '1';
          wait_timer_init_x  <= I2C_SPEED srl 2;
          NEXT_STATE         <= S_NACK_UNSET_SCL;
        end if;

      when S_NACK_UNSET_SCL =>
        scl_o                <= '0';
        if (wait_timer_done = '0') then
          NEXT_STATE         <= S_NACK_UNSET_SCL;
        else
          sequence_done_o_x  <= '1';
          NEXT_STATE         <= S_IDLE;
        end if;

    end case;
  end process PROC_READ_BYTE;

  ------------------------------------------------------------------------------
  -- Output Signals
  ------------------------------------------------------------------------------

  SEQUENCE_DONE_OUT <= sequence_done_o;
  BYTE_OUT          <= i2c_data;

  -- I2c Outputs
  SDA_OUT <= sda_o;
  SCL_OUT <= scl_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.nxyter_components.all;


entity nx_i2c_sendbyte is
  generic (
    I2C_SPEED : unsigned(11 downto 0) := x"3e8"
```

```
    );
  port(
    CLK_IN               : in  std_logic;
    RESET_IN             : in  std_logic;

    START_IN             : in  std_logic;
    BYTE_IN              : in  std_logic_vector(7 downto 0);
    SEQUENCE_DONE_OUT    : out std_logic;

    -- I2C connections
    SDA_OUT              : out std_logic;
    SCL_OUT              : out std_logic;
    SDA_IN               : in  std_logic;
    SCL_IN               : in  std_logic;
    ACK_OUT              : out std_logic
    );
end entity;

architecture Behavioral of nx_i2c_sendbyte is

  -- Send Byte
  signal sda_o              : std_logic;
  signal scl_o              : std_logic;
  signal i2c_start          : std_logic;

  signal sequence_done_o    : std_logic;
  signal i2c_byte           : unsigned(7 downto 0);
  signal bit_ctr            : unsigned(3 downto 0);
  signal i2c_ack_o          : std_logic;
  signal wait_timer_start   : std_logic;
  signal wait_timer_init    : unsigned(11 downto 0);
  signal stretch_timeout    : unsigned(19 downto 0);

  signal sequence_done_o_x  : std_logic;
  signal i2c_byte_x         : unsigned(7 downto 0);
  signal bit_ctr_x          : unsigned(3 downto 0);
  signal i2c_ack_o_x        : std_logic;
  signal wait_timer_start_x : std_logic;
  signal wait_timer_init_x  : unsigned(11 downto 0);
  signal stretch_timeout_x  : unsigned(19 downto 0);

  type STATES is (S_IDLE,
                  S_INIT,
                  S_INIT_WAIT,

                  S_SEND_BYTE,
                  S_SET_SDA,
                  S_SET_SCL,
                  S_UNSET_SCL,
                  S_NEXT_BIT,

                  S_ACK_UNSET_SCL,
                  S_ACK_SET_SCL,
                  S_STRETCH_CHECK_SCL,
                  S_STRETCH_WAIT_SCL,
                  S_STRETCH_PAUSE,
                  S_ACK_STORE,
                  S_ACK_UNSET_SCL2
                  );
  signal STATE, NEXT_STATE : STATES;

  -- Wait Timer
```



```
  signal wait_timer_done    : std_logic;

begin

  -- Timer
  timer_1: timer
    generic map (
      CTR_WIDTH => 12
      )
    port map (
      CLK_IN         => CLK_IN,
      RESET_IN       => RESET_IN,
      TIMER_START_IN => wait_timer_start,
      TIMER_END_IN   => wait_timer_init,
      TIMER_DONE_OUT => wait_timer_done
      );


  PROC_SEND_BYTE_TRANSFER: process(CLK_IN)
  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1' ) then
        sequence_done_o  <= '0';
        bit_ctr          <= (others => '0');
        i2c_ack_o        <= '0';
        wait_timer_start <= '0';
        wait_timer_init  <= (others => '0');
        stretch_timeout  <= (others => '0');
        STATE            <= S_IDLE;
      else
        sequence_done_o  <= sequence_done_o_x;
        i2c_byte         <= i2c_byte_x;
        bit_ctr          <= bit_ctr_x;
        i2c_ack_o        <= i2c_ack_o_x;
        wait_timer_start <= wait_timer_start_x;
        wait_timer_init  <= wait_timer_init_x;
        stretch_timeout  <= stretch_timeout_x;
        STATE            <= NEXT_STATE;
      end if;
    end if;
  end process PROC_SEND_BYTE_TRANSFER;

  PROC_SEND_BYTE: process(STATE,
                          START_IN,
                          wait_timer_done,
                          bit_ctr
                          )
  begin
    sda_o             <= '1';
    scl_o             <= '1';
    sequence_done_o_x <= '0';
    i2c_byte_x        <= i2c_byte;
    bit_ctr_x         <= bit_ctr;
    i2c_ack_o_x       <= i2c_ack_o;
    wait_timer_start_x <= '0';
    wait_timer_init_x  <= wait_timer_init;
    stretch_timeout_x  <= stretch_timeout;

    case STATE is
      when S_IDLE =>
        if (START_IN = '1') then
          sda_o               <= '0';
```

```
        scl_o                <= '0';
        i2c_byte_x           <= BYTE_IN;
        NEXT_STATE           <= S_INIT;
      else
        NEXT_STATE           <= S_IDLE;
      end if;

      -- INIT
    when S_INIT =>
      sda_o                  <= '0';
      scl_o                  <= '0';
      wait_timer_start_x     <= '1';
      wait_timer_init_x      <= I2C_SPEED srl 1;
      NEXT_STATE             <= S_INIT_WAIT;

    when S_INIT_WAIT =>
      sda_o                  <= '0';
      scl_o                  <= '0';
      if (wait_timer_done = '0') then
        NEXT_STATE           <= S_INIT_WAIT;
      else
        NEXT_STATE           <= S_SEND_BYTE;
      end if;

      -- I2C Send byte
    when S_SEND_BYTE =>
      sda_o                  <= '0';
      scl_o                  <= '0';
      bit_ctr_x              <= x"7";
      wait_timer_start_x     <= '1';
      wait_timer_init_x      <= I2C_SPEED srl 2;
      NEXT_STATE             <= S_SET_SDA;

    when S_SET_SDA =>
      sda_o                  <= i2c_byte(7);
      scl_o                  <= '0';
      if (wait_timer_done = '0') then
        NEXT_STATE           <= S_SET_SDA;
      else
        wait_timer_start_x   <= '1';
        wait_timer_init_x    <= I2C_SPEED srl 1;
        NEXT_STATE           <= S_SET_SCL;
      end if;

    when S_SET_SCL =>
      sda_o                  <= i2c_byte(7);
      if (wait_timer_done = '0') then
        NEXT_STATE           <= S_SET_SCL;
      else
        wait_timer_start_x   <= '1';
        wait_timer_init_x    <= I2C_SPEED srl 2;
        NEXT_STATE           <= S_UNSET_SCL;
      end if;

    when S_UNSET_SCL =>
      sda_o                  <= i2c_byte(7);
      scl_o                  <= '0';
      if (wait_timer_done = '0') then
        NEXT_STATE           <= S_UNSET_SCL;
      else
        NEXT_STATE           <= S_NEXT_BIT;
      end if;
```



```
    when S_NEXT_BIT =>
      sda_o                  <= i2c_byte(7);
      scl_o                  <= '0';
      if (bit_ctr > 0) then
        bit_ctr_x            <= bit_ctr - 1;
        i2c_byte_x           <= i2c_byte sll 1;
        wait_timer_start_x   <= '1';
        wait_timer_init_x    <= I2C_SPEED srl 2;
        NEXT_STATE           <= S_SET_SDA;
      else
        wait_timer_start_x   <= '1';
        wait_timer_init_x    <= I2C_SPEED srl 2;
        NEXT_STATE           <= S_ACK_UNSET_SCL;
      end if;

      -- Get Slave ACK bit
    when S_ACK_UNSET_SCL =>
      scl_o                  <= '0';
      if (wait_timer_done = '0') then
        NEXT_STATE           <= S_ACK_UNSET_SCL;
      else
        wait_timer_start_x   <= '1';
        wait_timer_init_x    <= I2C_SPEED srl 2;
        NEXT_STATE           <= S_ACK_SET_SCL;
      end if;

    when S_ACK_SET_SCL =>
      if (wait_timer_done = '0') then
        NEXT_STATE           <= S_ACK_SET_SCL;
      else
        NEXT_STATE           <= S_STRETCH_CHECK_SCL;
      end if;

      -- Check for Clock Stretching
    when S_STRETCH_CHECK_SCL =>
      if (SCL_IN = '1') then
        wait_timer_start_x   <= '1';
        wait_timer_init_x    <= I2C_SPEED srl 2;
        NEXT_STATE           <= S_ACK_STORE;
      else
        stretch_timeout_x    <= (others => '0');
        NEXT_STATE           <= S_STRETCH_WAIT_SCL;
      end if;

     when S_STRETCH_WAIT_SCL =>
      if (SCL_IN = '0') then
        if (stretch_timeout < x"30d40") then
          stretch_timeout_x  <= stretch_timeout + 1;
          NEXT_STATE         <= S_STRETCH_WAIT_SCL;
        else
          i2c_ack_o_x        <= '0';
          wait_timer_start_x <= '1';
          wait_timer_init_x  <= I2C_SPEED srl 2;
          NEXT_STATE         <= S_ACK_UNSET_SCL;
        end if;
      else
        wait_timer_start_x   <= '1';
        wait_timer_init_x    <= I2C_SPEED srl 2;
        NEXT_STATE           <= S_STRETCH_PAUSE;
      end if;
```

```
        when S_STRETCH_PAUSE =>
        if (wait_timer_done = '0') then
          NEXT_STATE           <= S_STRETCH_PAUSE;
        else
          wait_timer_start_x   <= '1';
          wait_timer_init_x    <= I2C_SPEED srl 2;
          NEXT_STATE           <= S_ACK_STORE;
        end if;

        -- Read ACK Bit
      when  S_ACK_STORE =>
        if (wait_timer_done = '0') then
          NEXT_STATE           <= S_ACK_STORE;
        else
          i2c_ack_o_x          <= not SDA_IN;
          wait_timer_start_x   <= '1';
          wait_timer_init_x    <= I2C_SPEED srl 2;
          NEXT_STATE           <= S_ACK_UNSET_SCL2;
        end if;

      when S_ACK_UNSET_SCL2 =>
        scl_o                  <= '0';
        if (wait_timer_done = '0') then
          NEXT_STATE <= S_ACK_UNSET_SCL2;
        else
          sequence_done_o_x    <= '1';
          NEXT_STATE           <= S_IDLE;
        end if;

    end case;
  end process PROC_SEND_BYTE;

  ------------------------------------------------------------------------------
  -- Output Signals
  ------------------------------------------------------------------------------

  SEQUENCE_DONE_OUT <= sequence_done_o;
  ACK_OUT           <= i2c_ack_o;

  -- I2c Outputs
  SDA_OUT <= sda_o;
  SCL_OUT <= scl_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.nxyter_components.all;

entity nx_i2c_startstop is
  generic (
    I2C_SPEED : unsigned(11 downto 0) := x"3e8"
    );
  port(
    CLK_IN               : in  std_logic;
    RESET_IN             : in  std_logic;

    START_IN             : in  std_logic;  -- Start Sequence
    SELECT_IN            : in  std_logic;  -- '1' -> Start, '0'-> Stop
    SEQUENCE_DONE_OUT    : out std_logic;
```



```
    -- I2C connections
    SDA_OUT              : out std_logic;
    SCL_OUT              : out std_logic;
    NREADY_OUT           : out std_logic
    );
end entity;

architecture Behavioral of nx_i2c_startstop is

  -- I2C Bus
  signal sda_o              : std_logic;
  signal scl_o              : std_logic;
  signal sequence_done_o    : std_logic;
  signal wait_timer_start   : std_logic;

  signal wait_timer_start_x : std_logic;
  signal sequence_done_o_x  : std_logic;

  type STATES is (S_IDLE,
                  S_START,
                  S_WAIT_START_1,
                  S_WAIT_START_2,
                  S_WAIT_START_3,

                  S_STOP,
                  S_WAIT_STOP_1,
                  S_WAIT_STOP_2,
                  S_WAIT_STOP_3
                  );
  signal STATE, NEXT_STATE : STATES;

  -- I2C Timer
  signal wait_timer_done    : std_logic;

begin

  -- Timer
  timer_static_1: timer_static
    generic map (
      CTR_WIDTH => 12,
      CTR_END   => to_integer(I2C_SPEED srl 1)
      )
    port map (
      CLK_IN         => CLK_IN,
      RESET_IN       => RESET_IN,
      TIMER_START_IN => wait_timer_start,
      TIMER_DONE_OUT => wait_timer_done
      );

  PROC_START_STOP_TRANSFER: process(CLK_IN)
  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1' ) then
        wait_timer_start <= '0';
        sequence_done_o  <= '0';
        STATE <= S_IDLE;
      else
        wait_timer_start <= wait_timer_start_x;
        sequence_done_o  <= sequence_done_o_x;
        STATE            <= NEXT_STATE;
      end if;
```

```
  end if;
end process PROC_START_STOP_TRANSFER;

PROC_START_STOP: process(STATE,
                         START_IN,
                         SELECT_IN,
                         wait_timer_done
                         )
begin
  sda_o              <= '1';
  scl_o              <= '1';
  sequence_done_o_x  <= '0';
  wait_timer_start_x <= '0';

  case STATE is
    when S_IDLE =>
      if (START_IN = '1') then
        if (SELECT_IN = '1') then
          NEXT_STATE <= S_START;
        else
          sda_o      <= '0';
          scl_o      <= '0';
          NEXT_STATE <= S_STOP;
        end if;
      else
        NEXT_STATE <= S_IDLE;
      end if;

      -- I2C START Sequence
    when S_START =>
      wait_timer_start_x   <= '1';
      NEXT_STATE           <= S_WAIT_START_1;

    when S_WAIT_START_1 =>
      if (wait_timer_done = '0') then
        NEXT_STATE         <= S_WAIT_START_1;
      else
        wait_timer_start_x <= '1';
        NEXT_STATE         <= S_WAIT_START_2;
      end if;

    when S_WAIT_START_2 =>
      sda_o                <= '0';
      if (wait_timer_done = '0') then
        NEXT_STATE <= S_WAIT_START_2;
      else
        wait_timer_start_x <= '1';
        NEXT_STATE         <= S_WAIT_START_3;
      end if;

    when S_WAIT_START_3 =>
      sda_o                <= '0';
      scl_o                <= '0';
      if (wait_timer_done = '0') then
        NEXT_STATE         <= S_WAIT_START_3;
      else
        sequence_done_o_x  <= '1';
        NEXT_STATE         <= S_IDLE;
      end if;

      -- I2C STOP Sequence
    when S_STOP =>
```



```
        sda_o                <= '0';
        scl_o                <= '0';
        wait_timer_start_x   <= '1';
        NEXT_STATE           <= S_WAIT_STOP_1;

      when S_WAIT_STOP_1 =>
        sda_o                <= '0';
        scl_o                <= '0';
        if (wait_timer_done = '0') then
          NEXT_STATE         <= S_WAIT_STOP_1;
        else
          wait_timer_start_x <= '1';
          NEXT_STATE         <= S_WAIT_STOP_2;
        end if;

      when S_WAIT_STOP_2 =>
        sda_o                <= '0';
        if (wait_timer_done = '0') then
          NEXT_STATE         <= S_WAIT_STOP_2;
        else
          wait_timer_start_x <= '1';
          NEXT_STATE         <= S_WAIT_STOP_3;
        end if;

      when S_WAIT_STOP_3 =>
        if (wait_timer_done = '0') then
          NEXT_STATE <= S_WAIT_STOP_3;
        else
          sequence_done_o_x <= '1';
          NEXT_STATE <= S_IDLE;
        end if;

    end case;
  end process PROC_START_STOP;



  -------------------------------------------------------------------------------
  -- Output Signals
  -------------------------------------------------------------------------------

  SEQUENCE_DONE_OUT <= sequence_done_o;
  SDA_OUT           <= sda_o;
  SCL_OUT           <= scl_o;
  NREADY_OUT        <= '0';

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.trb_net_std.all;
use work.trb_net_components.all;
use work.nxyter_components.all;

entity nx_setup is
  port(
    CLK_IN               : in  std_logic;
    RESET_IN             : in  std_logic;

    I2C_COMMAND_OUT      : out std_logic_vector(31 downto 0);
```

```
    I2C_COMMAND_BUSY_IN  : in  std_logic;
    I2C_DATA_IN          : in  std_logic_vector(31 downto 0);
    I2C_DATA_BYTES_IN    : in  std_logic_vector(31 downto 0);
    I2C_LOCK_OUT         : out std_logic;
    I2C_ONLINE_OUT       : out std_logic;
    I2C_REG_RESET_IN     : in  std_logic;

    SPI_COMMAND_OUT      : out std_logic_vector(31 downto 0);
    SPI_COMMAND_BUSY_IN  : in  std_logic;
    SPI_DATA_IN          : in  std_logic_vector(31 downto 0);
    SPI_LOCK_OUT         : out std_logic;

    -- Internal Register Read
    INT_READ_IN          : in  std_logic;
    INT_ADDR_IN          : in  std_logic_vector(15 downto 0);
    INT_ACK_OUT          : out std_logic;
    INT_DATA_OUT         : out std_logic_vector(31 downto 0);

    -- Slave bus
    SLV_READ_IN          : in  std_logic;
    SLV_WRITE_IN         : in  std_logic;
    SLV_DATA_OUT         : out std_logic_vector(31 downto 0);
    SLV_DATA_IN          : in  std_logic_vector(31 downto 0);
    SLV_ADDR_IN          : in  std_logic_vector(15 downto 0);
    SLV_ACK_OUT          : out std_logic;
    SLV_NO_MORE_DATA_OUT : out std_logic;
    SLV_UNKNOWN_ADDR_OUT : out std_logic;

    -- Debug Line
    DEBUG_OUT            : out std_logic_vector(15 downto 0)
    );
end entity;

architecture Behavioral of nx_setup is

  -- I2C Command Multiplexer
  signal i2c_lock_0      : std_logic;
  signal i2c_lock_1      : std_logic;
  signal i2c_lock_2      : std_logic;
  signal i2c_lock_3      : std_logic;
  signal i2c_lock_4      : std_logic;
  signal i2c_command     : std_logic_vector(31 downto 0);

  -- Send I2C Command
  type I2C_STATES is (I2C_IDLE,
                      I2C_WAIT_BUSY_HIGH,
                      I2C_WAIT_BUSY_LOW
                      );

  signal I2C_STATE : I2C_STATES;

  signal i2c_command_o           : std_logic_vector(31 downto 0);
  signal i2c_command_busy_o      : std_logic;
  signal i2c_command_done        : std_logic;
  signal i2c_error               : std_logic;
  signal i2c_data                : std_logic_vector(31 downto 0);
  signal i2c_data_bytes          : std_logic_vector(31 downto 0);

  -- I2C Register Ram
  type i2c_ram_t is array(0 to 45) of std_logic_vector(7 downto 0);
  signal i2c_ram                 : i2c_ram_t;
```



```
  type register_access_type_t is array(0 to 45) of std_logic_vector(1 downto 0);
  constant register_access_type : register_access_type_t :=
    ("11", "11", "11", "11", "11", "11", "11", "11",    --  0 ->  7
     "11", "11", "11", "11", "11", "11", "11", "11",    --  8 -> 15
     "11", "11", "11", "11", "11", "11", "11", "11",    -- 16 -> 23
     "11", "11", "11", "11", "11", "11", "00", "00",    -- 24 -> 31
     "11", "11", "10", "10", "10", "10", "11", "11",    -- 32 -> 39
     "00", "00", "00", "11", "11", "11"                 -- 40 -> 45
     );

  -- I2C RAM Handler
  signal ram_index_0             : integer;
  signal ram_index_1             : integer;
  signal ram_data_0              : std_logic_vector(7 downto 0);
  signal ram_data_1              : std_logic_vector(7 downto 0);
  signal ram_write_0             : std_logic;
  signal ram_write_1             : std_logic;
  signal do_write                : std_logic;

  -- DAC Trim FIFO RAM Handler
  type dac_ram_t is array(0 to 128) of std_logic_vector(5 downto 0);
  signal dac_ram                 : dac_ram_t;
  signal dac_ram_write_0         : std_logic;
  signal dac_ram_write_1         : std_logic;
  signal dac_ram_index_0         : integer;
  signal dac_ram_index_1         : integer;
  signal dac_ram_data_0          : std_logic_vector(5 downto 0);
  signal dac_ram_data_1          : std_logic_vector(5 downto 0);
  signal do_dac_write            : std_logic;

  -- ADC RAM Handler
  type adc_ram_t is array(0 to 3) of std_logic_vector(12 downto 0);
  signal adc_ram                 : adc_ram_t;

  -- Token Handler
  signal i2c_read_token          : std_logic_vector(45 downto 0);
  signal i2c_write_token         : std_logic_vector(45 downto 0);

  -- I2C Registers IO Handler
  type T_STATES is (T_IDLE_TOKEN,
                    T_WRITE_I2C_REGISTER,
                    T_WAIT_I2C_WRITE_DONE,
                    T_READ_I2C_REGISTER,
                    T_WAIT_I2C_READ_DONE,
                    T_READ_I2C_STORE_MEM,
                    T_NEXT_TOKEN
                    );

  signal T_STATE  : T_STATES;


  signal nx_i2c_command          : std_logic_vector(31 downto 0);
  signal token_ctr               : unsigned(5 downto 0);
  signal next_token              : std_logic;
  signal read_token_clear        : std_logic_vector(45 downto 0);
  signal write_token_clear       : std_logic_vector(45 downto 0);
  signal i2c_lock_0_clear        : std_logic;

  -- DAC Token Handler
  signal dac_read_token          : std_logic_vector(128 downto 0);
  signal dac_write_token         : std_logic_vector(128 downto 0);
```

```
  -- Read DAC I2C Registers
  type DR_STATES is (DR_IDLE,
                     DR_REGISTER,
                     DR_WRITE_BACK,
                     DR_NEXT_REGISTER,
                     DR_WAIT_DONE
                    );

  signal DR_STATE, DR_STATE_RETURN : DR_STATES;

  signal dac_read_i2c_command    : std_logic_vector(31 downto 0);
  signal r_fifo_ctr              : unsigned(7 downto 0);
  signal dac_read_token_clear    : std_logic_vector(128 downto 0);
  signal next_token_dac_r        : std_logic;
  signal i2c_lock_1_clear        : std_logic;

  -- Write DAC I2C Registers
  type DW_STATES is (DW_IDLE,
                     DW_REGISTER,
                     DW_WRITE_BACK,
                     DW_NEXT_REGISTER,
                     DW_WAIT_DONE
                     );
  signal DW_STATE, DW_STATE_RETURN : DW_STATES;

  signal dac_write_i2c_command   : std_logic_vector(31 downto 0);
  signal w_fifo_ctr              : unsigned(7 downto 0);
  signal dac_write_token_clear   : std_logic_vector(128 downto 0);
  signal next_token_dac_w        : std_logic;
  signal i2c_lock_2_clear        : std_logic;

  -- ADC Token Handler
  signal adc_read_token          : std_logic_vector(3 downto 0);

  -- Read ADC Registers
  type ADC_STATES is (ADC_IDLE_TOKEN,
                      ADC_READ_I2C_REGISTER,
                      ADC_WAIT_I2C_READ_DONE,
                      ADC_READ_I2C_STORE_MEM,
                      ADC_NEXT_TOKEN
                     );

  signal ADC_STATE : ADC_STATES;

  signal adc_i2c_command         : std_logic_vector(31 downto 0);
  signal adc_token_ctr           : unsigned(1 downto 0);
  signal adc_read_token_clear    : std_logic_vector(3 downto 0);
  signal next_token_adc          : std_logic;
  signal i2c_lock_4_clear        : std_logic;

  -- I2C Online Check
  type R_STATES is (R_TIMER_RESTART,
                    R_IDLE,
                    R_READ_DUMMY,
                    R_WAIT_DONE
                    );

  signal R_STATE : R_STATES;

  signal wait_timer_start        : std_logic;
  signal wait_timer_done         : std_logic;
  signal i2c_online_command      : std_logic_vector(31 downto 0);
```



```
  signal i2c_lock_3_clear        : std_logic;
  signal i2c_online_o            : std_logic;

  -- I2C Status
  signal i2c_online_t            : std_logic_vector(7 downto 0);
  signal i2c_update_memory_p     : std_logic;
  signal i2c_update_memory       : std_logic;
  signal i2c_disable_memory      : std_logic;
  signal i2c_reg_reset_in_s      : std_logic;
  signal i2c_reg_reset_clear     : std_logic;

  -- Internal Register Read
  signal int_data_o              : std_logic_vector(31 downto 0);
  signal int_ack_o               : std_logic;

  -- TRBNet Slave Bus
  signal slv_data_out_o          : std_logic_vector(31 downto 0);
  signal slv_no_more_data_o      : std_logic;
  signal slv_unknown_addr_o      : std_logic;
  signal slv_ack_o               : std_logic;

  signal i2c_read_token_r        : std_logic_vector(45 downto 0);
  signal i2c_write_token_r       : std_logic_vector(45 downto 0);

  signal dac_read_token_r        : std_logic_vector(128 downto 0);
  signal dac_write_token_r       : std_logic_vector(128 downto 0);

  signal adc_read_token_r        : std_logic_vector(3 downto 0);

  signal nxyter_polarity         : std_logic_vector(1 downto 0);  -- 0: negative
  signal nxyter_testpulse        : std_logic_vector(1 downto 0);
  signal nxyter_testtrigger      : std_logic_vector(1 downto 0);
  signal nxyter_clock            : std_logic_vector(1 downto 0);
  signal nxyter_testchannels     : std_logic_vector(2 downto 0);
  signal i2c_update_memory_r     : std_logic;

begin

  ------------------------------------------------------------------------------
  -- DEBUG
  ------------------------------------------------------------------------------

  DEBUG_OUT(0)            <= CLK_IN;
  DEBUG_OUT(1)            <= I2C_COMMAND_BUSY_IN;
  DEBUG_OUT(2)            <= i2c_command_busy_o;
  DEBUG_OUT(3)            <= i2c_error;
  DEBUG_OUT(4)            <= i2c_command_done;
  DEBUG_OUT(5)            <= next_token_dac_r or
                             next_token_dac_w;
  DEBUG_OUT(6)            <= i2c_update_memory;
  DEBUG_OUT(7)            <= i2c_lock_0_clear;
  DEBUG_OUT(8)            <= i2c_lock_1_clear;
  DEBUG_OUT(9)            <= i2c_lock_2_clear;
  DEBUG_OUT(10)           <= i2c_lock_4_clear;
  DEBUG_OUT(11)           <= i2c_online_o;
  DEBUG_OUT(12)           <= i2c_lock_0;
  DEBUG_OUT(13)           <= i2c_lock_1;
  DEBUG_OUT(14)           <= i2c_lock_2;
  DEBUG_OUT(15)           <= i2c_lock_4;

  ------------------------------------------------------------------------------
```

```
PROC_I2C_RAM: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      i2c_write_token_r   <= (others => '0');
      do_write            <= '0';
    else
      i2c_write_token_r   <= (others => '0');
      do_write            <= '0';

      if (ram_write_0 = '1' and
          register_access_type(ram_index_0)(0) = '1') then
        i2c_ram(ram_index_0)           <= ram_data_0;
        i2c_write_token_r(ram_index_0) <= '1';
        do_write                       <= '1';
      elsif (ram_write_1 = '1'                          and
             register_access_type(ram_index_1)(0) = '1' and
             i2c_write_token(ram_index_1) = '0') then
        i2c_ram(ram_index_1)           <= ram_data_1;
        do_write                       <= '1';
      elsif (nxyter_polarity(1) = '1') then
        i2c_ram(33)(2)                 <= nxyter_polarity(0);
        i2c_ram(32)(2)                 <= not nxyter_polarity(0);
        i2c_write_token_r(33)          <= '1';
        i2c_write_token_r(32)          <= '1';
        do_write                       <= '1';
      elsif (nxyter_clock(1) = '1') then
        i2c_ram(33)(3)                 <= nxyter_clock(0);
        i2c_write_token_r(33)          <= '1';
        do_write                       <= '1';
      elsif (nxyter_testtrigger(1) = '1') then
        i2c_ram(32)(3)                 <= nxyter_testtrigger(0);
        i2c_write_token_r(32)          <= '1';
        do_write                       <= '1';
      elsif (nxyter_testpulse(1) = '1') then
        i2c_ram(32)(0)                 <= nxyter_testpulse(0);
        i2c_write_token_r(32)          <= '1';
        do_write                       <= '1';
      elsif (nxyter_testchannels(2) = '1') then
        i2c_ram(33)(1 downto 0)        <= nxyter_testchannels(1 downto 0);
        i2c_write_token_r(33)          <= '1';
        do_write                       <= '1';
      end if;
    end if;
  end if;
end process PROC_I2C_RAM;

PROC_DAC_RAM: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      dac_write_token_r   <= (others => '0');
      do_dac_write        <= '0';
    else
      dac_write_token_r   <= (others => '0');
      do_dac_write        <= '0';

      if (dac_ram_write_0 = '1') then
        dac_ram(dac_ram_index_0)           <= dac_ram_data_0;
        dac_write_token_r(dac_ram_index_0) <= '1';
        do_dac_write                       <= '1';
      elsif (dac_ram_write_1 = '1' and
```



```
             dac_write_token(dac_ram_index_1) = '0') then
        dac_ram(dac_ram_index_1)           <= dac_ram_data_1;
        do_dac_write                       <= '1';
      end if;
    end if;
  end if;
end process PROC_DAC_RAM;

--------------------------------------------------------------------------------

PROC_I2C_COMMAND_MULTIPLEXER: process(CLK_IN)
  variable locks : std_logic_vector(4 downto 0) := (others => '0');
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      i2c_lock_0       <= '0';
      i2c_lock_1       <= '0';
      i2c_lock_2       <= '0';
      i2c_lock_3       <= '0';
      i2c_lock_4       <= '0';
      i2c_command      <= (others => '0');
    else
      i2c_command      <= (others => '0');
      locks            := i2c_lock_4 & i2c_lock_3 &
                          i2c_lock_2 & i2c_lock_1 &
                          i2c_lock_0;

      -- Clear Locks
      if (i2c_lock_0_clear = '1') then
        i2c_lock_0          <= '0';
      end if;
      if (i2c_lock_1_clear = '1') then
        i2c_lock_1          <= '0';
      end if;
      if (i2c_lock_2_clear = '1') then
        i2c_lock_2          <= '0';
      end if;
      if (i2c_lock_3_clear = '1') then
        i2c_lock_3          <= '0';
      end if;
      if (i2c_lock_4_clear = '1') then
        i2c_lock_4          <= '0';
      end if;

      if (i2c_command_busy_o = '0') then
        if (nx_i2c_command(31)   = '1'      and
            ((locks and "11110") = "00000") and
            i2c_lock_0_clear     = '0') then
          i2c_command       <= nx_i2c_command;
          i2c_lock_0        <= '1';
        elsif (dac_write_i2c_command(31) = '1'      and
               ((locks and "11011")      = "00000") and
               i2c_lock_2_clear          = '0') then
          i2c_command       <= dac_write_i2c_command;
          i2c_lock_2        <= '1';
        elsif (dac_read_i2c_command(31) = '1'      and
               ((locks and "11101")     = "00000") and
               i2c_lock_1_clear         = '0') then
          i2c_command       <= dac_read_i2c_command;
          i2c_lock_1        <= '1';
        elsif (i2c_online_command(31) = '1'      and
               ((locks and "10111")   = "00000") and
```

```
                i2c_lock_3_clear       = '0') then
            i2c_command       <= i2c_online_command;
            i2c_lock_3        <= '1';
          elsif (adc_i2c_command(31)  = '1'      and
                 ((locks and "01111") = "00000") and
                 i2c_lock_4_clear     = '0') then
            i2c_command       <= adc_i2c_command;
            i2c_lock_4        <= '1';
          end if;
        end if;
      end if;
    end if;
  end process PROC_I2C_COMMAND_MULTIPLEXER;

  PROC_SEND_I2C_COMMAND: process(CLK_IN)
  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1' ) then
        i2c_command_o       <= (others => '0');
        i2c_command_busy_o  <= '0';
        i2c_command_done    <= '0';
        i2c_error           <= '0';
        i2c_data            <= (others => '0');
        i2c_data_bytes      <= (others => '0');
        I2C_STATE           <= I2C_IDLE;
      else
        i2c_command_o       <= (others => '0');
        i2c_command_busy_o  <= '1';
        i2c_command_done    <= '0';
        i2c_error           <= '0';

        case I2C_STATE is

          when I2C_IDLE =>
            if (i2c_command(31) = '1') then
              i2c_command_o       <= i2c_command;
              I2C_STATE           <= I2C_WAIT_BUSY_HIGH;
            else
              i2c_command_busy_o  <= '0';
              I2C_STATE           <= I2C_IDLE;
            end if;

          when I2C_WAIT_BUSY_HIGH =>
            if (I2C_COMMAND_BUSY_IN = '0') then
              i2c_command_o       <= i2c_command_o;
              I2C_STATE           <= I2C_WAIT_BUSY_HIGH;
            else
              I2C_STATE           <= I2C_WAIT_BUSY_LOW;
            end if;

          when I2C_WAIT_BUSY_LOW =>
            if (I2C_COMMAND_BUSY_IN = '1') then
              I2C_STATE           <= I2C_WAIT_BUSY_LOW;
            else
              if (i2c_data(29 downto 24) ="000000") then
                i2c_error         <= '0';
              else
                i2c_error         <= '1';
              end if;
              i2c_data            <= I2C_DATA_IN;
              i2c_data_bytes      <= I2C_DATA_BYTES_IN;
              i2c_command_done    <= '1';
```



```
              I2C_STATE           <= I2C_IDLE;
            end if;
        end case;

      end if;
    end if;
  end process PROC_SEND_I2C_COMMAND;

  ------------------------------------------------------------------------------

  PROC_I2C_TOKEN_HANDLER: process(CLK_IN)
    variable read_token_mask : std_logic_vector(45 downto 0) := (others => '1');
  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1' ) then
        i2c_read_token      <= (others => '0');
        i2c_write_token     <= (others => '0');
      else
        if (i2c_ram(32)(3) = '1') then
          read_token_mask(15 downto 0)  := (others => '0');
          read_token_mask(45 downto 16) := (others => '1');
        else
          read_token_mask               := (others => '1');
        end if;

        -- Write Token
        if (unsigned(i2c_write_token_r)    /= 0) then
          i2c_write_token   <= i2c_write_token or i2c_write_token_r;
        elsif (unsigned(write_token_clear) /= 0) then
          i2c_write_token   <= i2c_write_token and (not write_token_clear);
        end if;

        -- Read Token
        if (i2c_update_memory = '1') then
          i2c_read_token    <= read_token_mask;
        elsif (unsigned(i2c_read_token_r) /= 0) then
          i2c_read_token    <= (i2c_read_token or i2c_read_token_r) and
                               read_token_mask;
        elsif (unsigned(read_token_clear) /= 0) then
          i2c_read_token    <= i2c_read_token and (not read_token_clear);
        end if;
      end if;
    end if;
  end process PROC_I2C_TOKEN_HANDLER;

  PROC_DAC_TOKEN_HANDLER: process(CLK_IN)
  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1' ) then
        dac_read_token      <= (others => '0');
        dac_write_token     <= (others => '0');
      else
        -- Write Token
        if (unsigned(dac_write_token_r) /= 0) then
          dac_write_token   <= dac_write_token or dac_write_token_r;
        elsif (unsigned(dac_write_token_clear) /= 0) then
          dac_write_token   <= dac_write_token and (not dac_write_token_clear);
        end if;

        -- Read Token
        if (i2c_update_memory = '1') then
          dac_read_token    <= (others => '1');
```

```
      elsif (unsigned(dac_read_token_r) /= 0) then
        dac_read_token    <= dac_read_token or dac_read_token_r;
      elsif (unsigned(dac_read_token_clear) /= 0) then
        dac_read_token    <= dac_read_token and (not dac_read_token_clear);
      end if;
    end if;
  end if;
end process PROC_DAC_TOKEN_HANDLER;


PROC_ADC_TOKEN_HANDLER: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      adc_read_token      <= (others => '0');
    else
      -- Read Token
      if (i2c_update_memory = '1') then
        adc_read_token    <= (others => '1');
      elsif (unsigned(adc_read_token_r) /= 0) then
        adc_read_token    <= (adc_read_token or adc_read_token_r);
      elsif (unsigned(adc_read_token_clear) /= 0) then
        adc_read_token    <= adc_read_token and (not adc_read_token_clear);
      end if;
    end if;
  end if;
end process PROC_ADC_TOKEN_HANDLER;

--------------------------------------------------------------------------------


PROC_I2C_REGISTERS_HANDLER: process(CLK_IN)
  variable index  : integer := 0;
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      nx_i2c_command           <= (others => '0');
      token_ctr                <= (others => '0');
      next_token               <= '0';
      read_token_clear         <= (others => '0');
      write_token_clear        <= (others => '0');
      ram_write_1              <= '0';
      i2c_lock_0_clear         <= '0';
      T_STATE                  <= T_IDLE_TOKEN;
    else
      index                    :=  to_integer(unsigned(token_ctr));
      nx_i2c_command           <= (others => '0');
      next_token               <= '0';
      read_token_clear         <= (others => '0');
      write_token_clear        <= (others => '0');
      ram_write_1              <= '0';
      i2c_lock_0_clear         <= '0';

      case T_STATE is

        when T_IDLE_TOKEN =>
          if (register_access_type(index)(0) = '1') then
            if (i2c_write_token(index) = '1') then
              T_STATE                        <= T_WRITE_I2C_REGISTER;
            elsif (i2c_read_token(index) = '1') then
              T_STATE                        <= T_READ_I2C_REGISTER;
            else
              T_STATE                        <= T_NEXT_TOKEN;
```



```
            end if;
          else
            read_token_clear(index)          <= '1';
            write_token_clear(index)         <= '1';
            T_STATE                          <= T_NEXT_TOKEN;
          end if;

          -- Write I2C Register
        when T_WRITE_I2C_REGISTER =>
          nx_i2c_command(31 downto 16)       <= x"8008";
          nx_i2c_command(15 downto 14)       <= (others => '0');
          nx_i2c_command(13 downto  8)       <= token_ctr;
          nx_i2c_command( 7 downto  0)       <= i2c_ram(index);
          if (i2c_lock_0 = '0') then
            T_STATE                          <= T_WRITE_I2C_REGISTER;
          else
            write_token_clear(index)         <= '1';
            T_STATE                          <= T_WAIT_I2C_WRITE_DONE;
          end if;

        when T_WAIT_I2C_WRITE_DONE =>
          if (i2c_command_done = '0') then
            T_STATE                          <= T_WAIT_I2C_WRITE_DONE;
          else
            i2c_lock_0_clear                 <= '1';
            T_STATE                          <= T_NEXT_TOKEN;
          end if;

          -- Read I2C Register
        when T_READ_I2C_REGISTER =>
          nx_i2c_command(31 downto 16)       <= x"c108";
          nx_i2c_command(15 downto 14)       <= (others => '0');
          nx_i2c_command(13 downto  8)       <= token_ctr;
          nx_i2c_command( 7 downto  0)       <= (others => '0');
          if (i2c_lock_0 = '0') then
            T_STATE                          <= T_READ_I2C_REGISTER;
          else
            read_token_clear(index)          <= '1';
            T_STATE                          <= T_WAIT_I2C_READ_DONE;
          end if;

        when T_WAIT_I2C_READ_DONE =>
          if (i2c_command_done = '0') then
            T_STATE                          <= T_WAIT_I2C_READ_DONE;
          else
            T_STATE                          <= T_READ_I2C_STORE_MEM;
          end if;

        when T_READ_I2C_STORE_MEM =>
          ram_index_1                        <= index;
          ram_data_1                         <= i2c_data(7 downto 0);
          ram_write_1                        <= '1';
          i2c_lock_0_clear                   <= '1';
          T_STATE                            <= T_NEXT_TOKEN;

          -- Next Token
        when T_NEXT_TOKEN =>
          if (token_ctr < x"2e") then
            token_ctr                        <= token_ctr + 1;
          else
            token_ctr                        <= (others => '0');
          end if;
```

```
          next_token                          <= '1';
          T_STATE                             <= T_IDLE_TOKEN;

      end case;
    end if;
  end if;
end process PROC_I2C_REGISTERS_HANDLER;

--------------------------------------------------------------------------------

PROC_READ_DAC_REGISTERS: process(CLK_IN)
  variable index : integer := 0;
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      dac_read_i2c_command   <= (others => '0');
      dac_ram_write_1        <= '0';
      dac_ram_index_1        <= 0;
      dac_ram_data_1         <= (others => '0');
      r_fifo_ctr             <= (others => '0');
      dac_read_token_clear   <= (others => '0');
      next_token_dac_r       <= '0';
      i2c_lock_1_clear       <= '0';
      DR_STATE_RETURN        <= DR_IDLE;
      DR_STATE               <= DR_IDLE;
    else
      dac_read_i2c_command   <= (others => '0');
      dac_ram_write_1        <= '0';
      dac_ram_index_1        <= 0;
      dac_ram_data_1         <= (others => '0');
      dac_read_token_clear   <= (others => '0');
      next_token_dac_r       <= '0';
      i2c_lock_1_clear       <= '0';
      index                  := to_integer(r_fifo_ctr);

      case DR_STATE is
        when DR_IDLE =>
          if (unsigned(dac_read_token) /= 0) then
            DR_STATE                          <= DR_REGISTER;
          else
            DR_STATE                          <= DR_IDLE;
          end if;
          r_fifo_ctr                          <= (others => '0');

        when DR_REGISTER =>
          dac_read_i2c_command(31 downto 16)  <= x"c108";
          dac_read_i2c_command(15 downto 8)   <= x"2a";  -- DAC Reg 42
          dac_read_i2c_command(7 downto 0)    <= (others => '0');
          if (i2c_lock_1 = '0') then
            DR_STATE                           <= DR_REGISTER;
          else
            dac_read_token_clear(index)        <= '1';
            DR_STATE_RETURN                    <= DR_WRITE_BACK;
            DR_STATE                           <= DR_WAIT_DONE;
          end if;

        when DR_WRITE_BACK =>
          -- Store FIFO Entry
          dac_ram_data_1                       <= i2c_data(5 downto 0);
          dac_ram_index_1                      <= index;
          dac_ram_write_1                      <= '1';
```



```
          -- Write Data Back to FIFO
          dac_read_i2c_command(31 downto 16)   <= x"8008";
          dac_read_i2c_command(15 downto 8)    <= x"2a";  -- DAC Reg 42
          dac_read_i2c_command(5 downto 0)     <= i2c_data(5 downto 0);
          dac_read_i2c_command(7 downto 6)     <= (others => '0');
          DR_STATE_RETURN                      <= DR_NEXT_REGISTER;
          DR_STATE                             <= DR_WAIT_DONE;

        when DR_NEXT_REGISTER =>
          if (r_fifo_ctr < x"80") then
            r_fifo_ctr                         <= r_fifo_ctr + 1;
            next_token_dac_r                   <= '1';
            DR_STATE                           <= DR_REGISTER;
          else
            i2c_lock_1_clear                   <= '1';
            DR_STATE                           <= DR_IDLE;
          end if;

        when DR_WAIT_DONE =>
          if (i2c_command_done = '0') then
            DR_STATE                           <= DR_WAIT_DONE;
          else
            DR_STATE                           <= DR_STATE_RETURN;
          end if;
      end case;

    end if;
  end if;
end process PROC_READ_DAC_REGISTERS;

PROC_WRITE_DAC_REGISTERS: process(CLK_IN)
  variable index : integer := 0;
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      dac_write_i2c_command  <= (others => '0');
      w_fifo_ctr             <= (others => '0');
      dac_write_token_clear  <= (others => '0');
      next_token_dac_w       <= '0';
      i2c_lock_2_clear       <= '0';
      DW_STATE_RETURN        <= DW_IDLE;
      DW_STATE               <= DW_IDLE;
    else
      dac_write_i2c_command  <= (others => '0');
      dac_write_token_clear  <= (others => '0');
      next_token_dac_w       <= '0';
      i2c_lock_2_clear       <= '0';

      index                  := to_integer(w_fifo_ctr);
      case DW_STATE is
        when DW_IDLE =>
          if (unsigned(dac_write_token) /= 0) then
            DW_STATE                           <= DW_REGISTER;
          else
            DW_STATE                           <= DW_IDLE;
          end if;
          w_fifo_ctr                           <= (others => '0');

        when DW_REGISTER =>
          dac_write_i2c_command(31 downto 16)  <= x"c108";
          dac_write_i2c_command(15 downto 8)   <= x"2a";  -- DAC Reg 42
          dac_write_i2c_command(7 downto 0)    <= (others => '0');
```

```
          dac_write_token_clear(index)          <= '1';
          if (i2c_lock_2 = '0') then
            DW_STATE                            <= DW_REGISTER;
          else
            dac_write_token_clear(index)        <= '1';
            DW_STATE_RETURN                     <= DW_WRITE_BACK;
            DW_STATE                            <= DW_WAIT_DONE;
          end if;

        when DW_WRITE_BACK =>
          -- Write Data Back to FIFO
          dac_write_i2c_command(31 downto 16)   <= x"8008";
          dac_write_i2c_command(15 downto 8)    <= x"2a";  -- DAC Reg 42
          dac_write_i2c_command(7 downto 6)     <= (others => '0');
          dac_write_i2c_command(5 downto 0)     <= dac_ram(index);
          DW_STATE_RETURN                       <= DW_NEXT_REGISTER;
          DW_STATE                              <= DW_WAIT_DONE;

        when DW_NEXT_REGISTER =>
          if (w_fifo_ctr < x"80") then
            w_fifo_ctr                          <= w_fifo_ctr + 1;
            next_token_dac_w                    <= '1';
            DW_STATE                            <= DW_REGISTER;
          else
            i2c_lock_2_clear                    <= '1';
            DW_STATE                            <= DW_IDLE;
          end if;

        when DW_WAIT_DONE =>
          if (i2c_command_done = '0') then
            DW_STATE                            <= DW_WAIT_DONE;
          else
            DW_STATE                            <= DW_STATE_RETURN;
          end if;

      end case;
    end if;
  end if;
end process PROC_WRITE_DAC_REGISTERS;

-------------------------------------------------------------------------------

PROC_ADC_REGISTERS_HANDLER: process(CLK_IN)
  variable index  : integer := 0;
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      adc_i2c_command           <= (others => '0');
      adc_token_ctr             <= (others => '0');
      next_token_adc            <= '0';
      adc_read_token_clear      <= (others => '0');
      i2c_lock_4_clear          <= '0';
      ADC_STATE                 <= ADC_IDLE_TOKEN;
    else
      index                     := to_integer(unsigned(adc_token_ctr));
      adc_i2c_command           <= (others => '0');
      next_token_adc            <= '0';
      adc_read_token_clear      <= (others => '0');
      i2c_lock_4_clear          <= '0';

      case ADC_STATE is
```



```
        when ADC_IDLE_TOKEN =>
          if (adc_read_token(index) = '1') then
            ADC_STATE                           <= ADC_READ_I2C_REGISTER;
          else
            ADC_STATE                           <= ADC_NEXT_TOKEN;
          end if;

          -- Read I2C Register
        when ADC_READ_I2C_REGISTER =>
          adc_i2c_command(31 downto 16)         <= x"c229";
          case adc_token_ctr is
            when "00" => adc_i2c_command(15 downto  8) <= x"10";
            when "01" => adc_i2c_command(15 downto  8) <= x"20";
            when "10" => adc_i2c_command(15 downto  8) <= x"40";
            when "11" => adc_i2c_command(15 downto  8) <= x"80";
          end case;
          adc_i2c_command(7 downto  0)          <= (others => '0');
          if (i2c_lock_4 = '0') then
            ADC_STATE                           <= ADC_READ_I2C_REGISTER;
          else
            adc_read_token_clear(index)         <= '1';
            ADC_STATE                           <= ADC_WAIT_I2C_READ_DONE;
          end if;

        when ADC_WAIT_I2C_READ_DONE =>
          if (i2c_command_done = '0') then
            ADC_STATE                           <= ADC_WAIT_I2C_READ_DONE;
          else
            ADC_STATE                           <= ADC_READ_I2C_STORE_MEM;
          end if;

        when ADC_READ_I2C_STORE_MEM =>
          if (i2c_data_bytes(13 downto 12) =
              std_logic_vector(adc_token_ctr)) then
            adc_ram(index)(11 downto 0)         <= i2c_data_bytes(11 downto 0);
            adc_ram(index)(12)                  <= '0';
          else
            adc_ram(index)                      <= (others => '1');
          end if;
          i2c_lock_4_clear                      <= '1';
          ADC_STATE                             <= ADC_NEXT_TOKEN;

          -- Next Token
        when ADC_NEXT_TOKEN =>
          if (adc_token_ctr < "11") then
            adc_token_ctr                       <= adc_token_ctr + 1;
          else
            adc_token_ctr                       <= (others => '0');
          end if;
          next_token_adc                        <= '1';
          ADC_STATE                             <= ADC_IDLE_TOKEN;

      end case;
    end if;
  end if;
end process PROC_ADC_REGISTERS_HANDLER;

-------------------------------------------------------------------------------

timer_static_1: timer_static
  generic map (
    CTR_WIDTH => 32,
```

```
    CTR_END   => 500000000    --5S
    )
  port map (
    CLK_IN         => CLK_IN,
    RESET_IN       => RESET_IN,
    TIMER_START_IN => wait_timer_start,
    TIMER_DONE_OUT => wait_timer_done
    );

PROC_I2C_ONLINE: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      i2c_online_command     <= (others => '0');
      i2c_online_o           <= '0';
      i2c_lock_3_clear       <= '0';
      wait_timer_start       <= '0';
      R_STATE                <= R_TIMER_RESTART;
    else
      i2c_online_command     <= (others => '0');
      i2c_lock_3_clear       <= '0';
      wait_timer_start       <= '0';

      case R_STATE is

        when R_TIMER_RESTART =>
          wait_timer_start                  <= '1';
          R_STATE                           <= R_IDLE;

        when R_IDLE =>
          if (wait_timer_done = '1') then
            R_STATE                         <= R_READ_DUMMY;
          else
            R_STATE                         <= R_IDLE;
          end if;

        when R_READ_DUMMY =>
          i2c_online_command(31 downto 16)  <= x"c108";
          i2c_online_command(15 downto 8)   <= x"1f";  -- Dummy register
          i2c_online_command(7 downto 0)    <= (others => '0');
          if (i2c_lock_3 = '0') then
            R_STATE                         <= R_READ_DUMMY;
          else
            R_STATE                         <= R_WAIT_DONE;
          end if;

        when R_WAIT_DONE =>
          if (i2c_command_done = '0') then
            R_STATE                         <= R_WAIT_DONE;
          else
            i2c_online_o                    <= not i2c_error;
            i2c_lock_3_clear                <= '1';
            R_STATE                         <= R_TIMER_RESTART;
          end if;

      end case;

    end if;
  end if;
end process PROC_I2C_ONLINE;

PROC_I2C_STATUS: process(CLK_IN)
```



```
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      i2c_update_memory_p   <= '0';
      i2c_disable_memory    <= '0';
      i2c_online_t          <= (others => '0');
      i2c_reg_reset_clear   <= '0';
    else
      i2c_reg_reset_clear   <= '0';

      -- Shift Online
      i2c_online_t(0)       <= i2c_online_o;
      for I  in 1 to 7 loop
        i2c_online_t(I)     <= i2c_online_t(I - 1);
      end loop;

      if (i2c_update_memory_r = '1') then
        i2c_update_memory_p       <= '1';
        i2c_disable_memory        <= '0';
      else

        case i2c_online_t(7 downto 6) is

          when "00" =>
            i2c_update_memory_p   <= '0';
            i2c_disable_memory    <= '1';

          when "10" =>
            i2c_update_memory_p   <= '0';
            i2c_disable_memory    <= '1';

          when "01" =>
            i2c_update_memory_p   <= '1';
            i2c_disable_memory    <= '0';

          when "11" =>
            if (i2c_reg_reset_in_s = '1' and I2C_REG_RESET_IN = '0') then
              i2c_update_memory_p <= '1';
              i2c_reg_reset_clear <= '1';
            else
              i2c_update_memory_p <= '0';
            end if;
            i2c_disable_memory    <= '0';

        end case;
      end if;
    end if;
  end if;
end process PROC_I2C_STATUS;

pulse_delay_1: pulse_delay
  generic map (
    DELAY => 1000000
    )
  port map (
    CLK_IN    => CLK_IN,
    RESET_IN  => RESET_IN,
    PULSE_IN  => i2c_update_memory_p,
    PULSE_OUT => i2c_update_memory
    );

PROC_REG_RESET: process(CLK_IN)
```

```
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      i2c_reg_reset_in_s    <= '0';
    else
      if (i2c_reg_reset_clear = '1') then
        i2c_reg_reset_in_s  <= '0';
      elsif(I2C_REG_RESET_IN = '1') then
        i2c_reg_reset_in_s  <= '1';
      end if;
    end if;
  end if;
end process PROC_REG_RESET;

--------------------------------------------------------------------------------

PROC_INTERNAL_REG_READ: process(CLK_IN)
  variable index       : integer  := 0;
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      int_data_o          <= (others => '0');
      int_ack_o           <= '0';
    else
      int_data_o          <= (others => '0');
      int_ack_o           <= '0';

      if (INT_READ_IN = '1') then
        if (INT_ADDR_IN >= x"0000" and INT_ADDR_IN <= x"002d") then
          index := to_integer(unsigned(INT_ADDR_IN(5 downto 0)));
          if (i2c_disable_memory = '0') then
            int_data_o(7 downto 0)      <= i2c_ram(index);
            int_data_o(28 downto 8)     <= (others => '0');
            int_data_o(29)              <=
              not register_access_type(index)(0);
            int_data_o(30)              <= i2c_read_token(index);
            int_data_o(31)              <= i2c_write_token(index);
          else
            int_data_o(31 downto 0)     <= (others => '1');
          end if;
          int_ack_o                     <= '1';

        elsif (INT_ADDR_IN >= x"0100" and INT_ADDR_IN <= x"0180") then
          index := to_integer(unsigned(INT_ADDR_IN(7 downto 0)));
          if (i2c_disable_memory = '0') then
            int_data_o(5 downto 0)      <= dac_ram(index);
            int_data_o(29 downto 6)     <= (others => '0');
            int_data_o(30)              <= dac_read_token(index);
            int_data_o(31)              <= dac_write_token(index);
          else
            int_data_o(31 downto 0)     <= (others => '1');
          end if;
          int_ack_o                     <= '1';

        elsif (INT_ADDR_IN >= x"0080" and INT_ADDR_IN <= x"0083") then
          index := to_integer(unsigned(INT_ADDR_IN(1 downto 0)));
          if (i2c_disable_memory = '0') then
            int_data_o(12 downto 0)     <= adc_ram(index);
            int_data_o(31 downto 13)    <= (others => '0');
          else
            int_data_o(31 downto 0)     <= (others => '1');
          end if;
```



```
          int_ack_o                     <= '1';
        else
          case INT_ADDR_IN is
            when x"0050" =>
              -- Nxyter Clock
              if (i2c_disable_memory = '0') then
                int_data_o(0)           <= i2c_ram(33)(3);
                int_data_o(31 downto 1) <= (others => '0');
              else
                int_data_o(31 downto 0) <= (others => '1');
              end if;
              int_ack_o                  <= '1';

            when x"0051" =>
              -- Nxyter Polarity
              if (i2c_disable_memory = '0') then
                int_data_o(0)           <= i2c_ram(33)(2);
                int_data_o(31 downto 1) <= (others => '0');
              else
                int_data_o(31 downto 0) <= (others => '1');
              end if;
              int_ack_o                 <= '1';

            when x"0052" =>
              -- Nxyter Testpulse Polarity
              if (i2c_disable_memory = '0') then
                int_data_o(0)           <= i2c_ram(32)(2);
                int_data_o(31 downto 1) <= (others => '0');
              else
                int_data_o(31 downto 0) <= (others => '1');
              end if;
              int_ack_o                 <= '1';

            when x"0053" =>
              -- Nxyter Testpulse
              if (i2c_disable_memory = '0') then
                int_data_o(0)           <= i2c_ram(32)(0);
                int_data_o(31 downto 1) <= (others => '0');
              else
                int_data_o(31 downto 0) <= (others => '1');
              end if;
              int_ack_o                 <= '1';

            when x"0054" =>
              -- Nxyter Testtrigger
              if (i2c_disable_memory = '0') then
                int_data_o(0)           <= i2c_ram(32)(3);
                int_data_o(31 downto 1) <= (others => '0');
              else
                int_data_o(31 downto 0) <= (others => '1');
              end if;
              int_ack_o                 <= '1';

            when x"0055" =>
              -- Nxyter Testpulse Channels
              if (i2c_disable_memory = '0') then
                int_data_o(1 downto 0)  <= i2c_ram(33)(1 downto 0);
                int_data_o(31 downto 2) <= (others => '0');
              else
                int_data_o(31 downto 0) <= (others => '1');
              end if;
              int_ack_o                 <= '1';
```

```
            when x"0056" =>
              -- I2C Online
              int_data_o(0)             <= i2c_online_o;
              int_data_o(31 downto 2)   <= (others => '0');
              int_ack_o                 <= '1';

            when others =>
              int_data_o(31 downto 0)   <= (others => '1');
              int_ack_o                 <= '1';
          end case;
        end if;
      end if;
    end if;
  end if;
end process PROC_INTERNAL_REG_READ;

-------------------------------------------------------------------------------

PROC_SLAVE_BUS: process(CLK_IN)
  variable index       : integer   := 0;
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      slv_data_out_o         <= (others => '0');
      slv_no_more_data_o     <= '0';
      slv_unknown_addr_o     <= '0';
      slv_ack_o              <= '0';

      ram_data_0             <= (others => '0');
      ram_index_0            <= 0;
      ram_write_0            <= '0';
      i2c_read_token_r       <= (others => '0');

      dac_ram_data_0         <= (others => '0');
      dac_ram_index_0        <= 0;
      dac_ram_write_0        <= '0';
      dac_read_token_r       <= (others => '0');
      adc_read_token_r       <= (others => '0');
      i2c_update_memory_r    <= '0';
      nxyter_clock           <= (others => '0');
      nxyter_polarity        <= (others => '0');
      nxyter_testtrigger     <= (others => '0');
      nxyter_testpulse       <= (others => '0');
      nxyter_testchannels    <= (others => '0');
    else
      slv_unknown_addr_o     <= '0';
      slv_no_more_data_o     <= '0';

      ram_data_0             <= (others => '0');
      ram_index_0            <= 0;
      ram_write_0            <= '0';
      i2c_read_token_r       <= (others => '0');

      dac_ram_data_0         <= (others => '0');
      dac_ram_index_0        <= 0;
      dac_ram_write_0        <= '0';
      dac_read_token_r       <= (others => '0');

      adc_read_token_r       <= (others => '0');

      i2c_update_memory_r    <= '0';
```



```
      nxyter_clock           <= (others => '0');
      nxyter_polarity        <= (others => '0');
      nxyter_testtrigger     <= (others => '0');
      nxyter_testpulse       <= (others => '0');
      nxyter_testchannels    <= (others => '0');

      if (SLV_WRITE_IN = '1') then
        if (SLV_ADDR_IN >= x"0000" and SLV_ADDR_IN <= x"002d") then
          index := to_integer(unsigned(SLV_ADDR_IN(5 downto 0)));
          if (i2c_disable_memory = '0') then
            ram_index_0               <= index;
            ram_data_0                <= SLV_DATA_IN(7 downto 0);
            ram_write_0               <= '1';
          end if;
          slv_ack_o                   <= '1';

        elsif (SLV_ADDR_IN >= x"0100" and SLV_ADDR_IN <= x"0180") then
          -- Write value to ram
          index   := to_integer(unsigned(SLV_ADDR_IN(7 downto 0)));
          if (index = 0) then
            index := 128;
          else
            index := index - 1;
          end if;

          if (i2c_disable_memory = '0') then
            dac_ram_index_0           <= index;
            dac_ram_data_0            <= SLV_DATA_IN(5 downto 0);
            dac_ram_write_0           <= '1';
          end if;
          slv_ack_o                   <= '1';

        else
          case SLV_ADDR_IN is
            when x"0050" =>
              -- Nxyter Clock
              if (i2c_disable_memory = '0') then
                nxyter_clock(0)       <= SLV_DATA_IN(0);
                nxyter_clock(1)       <= '1';
              end if;
              slv_ack_o               <= '1';

            when x"0051" =>
              -- Nxyter Polarity
              if (i2c_disable_memory = '0') then
                nxyter_polarity(0)    <= SLV_DATA_IN(0);
                nxyter_polarity(1)    <= '1';
              end if;
              slv_ack_o               <= '1';

            when x"0053" =>
              -- Nxyter Testpulse
              if (i2c_disable_memory = '0') then
                nxyter_testpulse(0)   <= SLV_DATA_IN(0);
                nxyter_testpulse(1)   <= '1';
              end if;
              slv_ack_o               <= '1';

            when x"0054" =>
              -- Nxyter Testtrigger
              if (i2c_disable_memory = '0') then
                nxyter_testtrigger(0) <= SLV_DATA_IN(0);
```

```
              nxyter_testtrigger(1)  <= '1';
            end if;
            slv_ack_o                <= '1';

          when x"0055" =>
            -- Nxyter Testtrigger
            if (i2c_disable_memory = '0') then
              nxyter_testchannels(1 downto 0) <= SLV_DATA_IN(1 downto 0);
              nxyter_testchannels(2) <= '1';
            end if;
            slv_ack_o                <= '1';

          when x"0060" =>
            if (i2c_disable_memory = '0') then
              i2c_read_token_r       <= (others => '1');
            end if;
            slv_ack_o                <= '1';

          when x"0061" =>
            if (i2c_disable_memory = '0') then
              dac_read_token_r       <= (others => '1');
            end if;
            slv_ack_o                <= '1';

          when x"0062" =>
            if (i2c_disable_memory = '0') then
              i2c_update_memory_r    <= '1';
            end if;
            slv_ack_o                <= '1';

          when others =>
            slv_unknown_addr_o       <= '1';
            slv_ack_o                <= '0';

        end case;
      end if;

    elsif (SLV_READ_IN = '1') then
      if (SLV_ADDR_IN >= x"0000" and SLV_ADDR_IN <= x"002d") then
        index := to_integer(unsigned(SLV_ADDR_IN(5 downto 0)));
        if (i2c_disable_memory = '0') then
          slv_data_out_o(7 downto 0)      <= i2c_ram(index);
          slv_data_out_o(28 downto 8)     <= (others => '0');
          slv_data_out_o(29)              <=
            not register_access_type(index)(0);
          slv_data_out_o(30)              <= i2c_read_token(index);
          slv_data_out_o(31)              <= i2c_write_token(index);
        else
          slv_data_out_o(31 downto 0)     <= (others => '1');
        end if;
        slv_ack_o                         <= '1';

      elsif (SLV_ADDR_IN >= x"0100" and SLV_ADDR_IN <= x"0180") then
        index   := to_integer(unsigned(SLV_ADDR_IN(7 downto 0)));
        if (index = 0) then
          index := 128;
        else
          index := index - 1;
        end if;

        if (i2c_disable_memory = '0') then
          slv_data_out_o(5 downto 0)      <= dac_ram(index);
```



```
          slv_data_out_o(29 downto 6)     <= (others => '0');
          slv_data_out_o(30)              <= dac_read_token(index);
          slv_data_out_o(31)              <= dac_write_token(index);
        else
          slv_data_out_o(31 downto 0)     <= (others => '1');
        end if;
        slv_ack_o                         <= '1';

      elsif (SLV_ADDR_IN >= x"0080" and SLV_ADDR_IN <= x"0083") then
        index := to_integer(unsigned(SLV_ADDR_IN(1 downto 0)));
        if (i2c_disable_memory = '0') then
          slv_data_out_o(12 downto 0)     <= adc_ram(index);
          slv_data_out_o(31 downto 13)    <= (others => '0');
          adc_read_token_r(index)         <= '1';
        else
          slv_data_out_o(31 downto 0)     <= (others => '1');
        end if;
        slv_ack_o                         <= '1';

      else
        case SLV_ADDR_IN is
          when x"0050" =>
            -- Nxyter Clock
            if (i2c_disable_memory = '0') then
              slv_data_out_o(0)           <= i2c_ram(33)(3);
              slv_data_out_o(31 downto 1) <= (others => '0');
            else
              slv_data_out_o(31 downto 0) <= (others => '1');
            end if;
            slv_ack_o                     <= '1';

          when x"0051" =>
            -- Nxyter Polarity
            if (i2c_disable_memory = '0') then
              slv_data_out_o(0)           <= i2c_ram(33)(2);
              slv_data_out_o(31 downto 1) <= (others => '0');
            else
              slv_data_out_o(31 downto 0) <= (others => '1');
            end if;
            slv_ack_o                     <= '1';

          when x"0052" =>
            -- Nxyter Testpulse Polarity
            if (i2c_disable_memory = '0') then
              slv_data_out_o(0)           <= i2c_ram(32)(2);
              slv_data_out_o(31 downto 1) <= (others => '0');
            else
              slv_data_out_o(31 downto 0) <= (others => '1');
            end if;
            slv_ack_o                     <= '1';

          when x"0053" =>
            -- Nxyter Testpulse
            if (i2c_disable_memory = '0') then
              slv_data_out_o(0)           <= i2c_ram(32)(0);
              slv_data_out_o(31 downto 1) <= (others => '0');
            else
              slv_data_out_o(31 downto 0) <= (others => '1');
            end if;
            slv_ack_o                     <= '1';

          when x"0054" =>
```

```
            -- Nxyter Testtrigger
            if (i2c_disable_memory = '0') then
              slv_data_out_o(0)           <= i2c_ram(32)(3);
              slv_data_out_o(31 downto 1) <= (others => '0');
            else
              slv_data_out_o(31 downto 0) <= (others => '1');
            end if;
            slv_ack_o                     <= '1';

          when x"0055" =>
            -- Nxyter Testpulse Channels
            if (i2c_disable_memory = '0') then
              slv_data_out_o(1 downto 0)  <= i2c_ram(33)(1 downto 0);
              slv_data_out_o(31 downto 2) <= (others => '0');
            else
              slv_data_out_o(31 downto 0) <= (others => '1');
            end if;
            slv_ack_o                     <= '1';

          when x"0056" =>
            -- I2C Online
            slv_data_out_o(0)             <= i2c_online_o;
            slv_data_out_o(31 downto 2)   <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0060" =>
            -- Update Register I2C Status
            if (unsigned(i2c_read_token) = 0) then
              slv_data_out_o              <= (others => '0');
            else
              slv_data_out_o              <= x"0000_0001";
            end if;
            slv_ack_o                     <= '1';

          when x"0061" =>
            -- Update Register DAC Status
            if (unsigned(dac_read_token) = 0) then
              slv_data_out_o              <= (others => '0');
            else
              slv_data_out_o              <= x"0000_0001";
            end if;
            slv_ack_o                     <= '1';

          when x"0062" =>
            -- Update Register I2C and DAC Status
            if (unsigned(i2c_read_token) = 0 and
                unsigned(dac_read_token) = 0) then
              slv_data_out_o              <= (others => '0');
            else
              slv_data_out_o              <= x"0000_0001";
            end if;
            slv_ack_o                     <= '1';

          when x"0070" =>
            -- WriteToken
            slv_data_out_o                <= i2c_write_token(31 downto 0);
            slv_ack_o                     <= '1';
          when x"0071" =>
            -- WriteToken
            slv_data_out_o(13 downto 0)   <= i2c_write_token(45 downto 32);
            slv_data_out_o(31 downto 14)  <= (others => '0');
            slv_ack_o                     <= '1';
```



```
          when x"0072" =>
            -- ReadToken
            slv_data_out_o                <= i2c_read_token(31 downto 0);
            slv_ack_o                     <= '1';
          when x"0073" =>
            -- ReadToken
            slv_data_out_o(13 downto 0)   <= i2c_read_token(45 downto 32);
            slv_data_out_o(31 downto 14)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0074" =>
            -- WriteTokenDAC
            slv_data_out_o                <= dac_write_token(31 downto 0);
            slv_ack_o                     <= '1';
          when x"0075" =>
            -- WriteTokenDAC
            slv_data_out_o                <= dac_write_token(63 downto 32);
            slv_ack_o                     <= '1';
          when x"0076" =>
            -- WriteTokenDAC
            slv_data_out_o                <= dac_write_token(95 downto 64);
            slv_ack_o                     <= '1';
          when x"0077" =>
            -- WriteTokenDAC
            slv_data_out_o                <= dac_write_token(127 downto 96);
            slv_ack_o                     <= '1';
          when x"0078" =>
            -- WriteTokenDAC
            slv_data_out_o(0)             <= dac_write_token(128);
            slv_data_out_o(31 downto 1)   <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0079" =>
            -- ReadTokenDAC
            slv_data_out_o                <= dac_read_token(31 downto 0);
            slv_ack_o                     <= '1';
          when x"007a" =>
            -- ReadTokenDAC
            slv_data_out_o                <= dac_read_token(63 downto 32);
            slv_ack_o                     <= '1';
          when x"007b" =>
            -- ReadTokenDAC
            slv_data_out_o                <= dac_read_token(95 downto 64);
            slv_ack_o                     <= '1';
          when x"007c" =>
            -- ReadTokenDAC
            slv_data_out_o                <= dac_read_token(127 downto 96);
            slv_ack_o                     <= '1';
          when x"007d" =>
            -- ReadTokenDAC
            slv_data_out_o(0)             <= dac_read_token(128);
            slv_data_out_o(31 downto 1)   <= (others => '0');
            slv_ack_o                     <= '1';

          when others =>
            slv_unknown_addr_o            <= '1';
            slv_ack_o                     <= '0';
        end case;

      end if;
    else
```

```
          slv_ack_o                          <= '0';
        end if;

      end if;
    end if;
  end process PROC_SLAVE_BUS;

  ------------------------------------------------------------------------------
  -- Output Signals
  ------------------------------------------------------------------------------

  I2C_COMMAND_OUT         <= i2c_command_o;
  I2C_LOCK_OUT            <= i2c_command_busy_o;
  I2C_ONLINE_OUT          <= i2c_online_o;

  SPI_COMMAND_OUT         <= (others => '0');
  SPI_LOCK_OUT            <= '0';

  -- Internal Read
  INT_ACK_OUT             <= int_ack_o;
  INT_DATA_OUT            <= int_data_o;

  -- Slave Bus
  SLV_DATA_OUT            <= slv_data_out_o;
  SLV_NO_MORE_DATA_OUT    <= slv_no_more_data_o;
  SLV_UNKNOWN_ADDR_OUT    <= slv_unknown_addr_o;
  SLV_ACK_OUT             <= slv_ack_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.nxyter_components.all;
use work.trb3_components.all;

entity nx_status_event is
  generic (
    BOARD_ID : std_logic_vector(1 downto 0) := "11"
    );
  port (
    CLK_IN                    : in  std_logic;
    RESET_IN                  : in  std_logic;
    NXYTER_OFFLINE_IN         : in  std_logic;

    -- Trigger
    TRIGGER_IN                : in  std_logic;
    FAST_CLEAR_IN             : in  std_logic;
    TRIGGER_BUSY_OUT          : out std_logic;

    --Response from FEE
    FEE_DATA_OUT              : out std_logic_vector(31 downto 0);
    FEE_DATA_WRITE_OUT        : out std_logic;
    FEE_DATA_ALMOST_FULL_IN   : in  std_logic;

    -- Interface to NX Setup
    INT_READ_OUT              : out std_logic;
    INT_ADDR_OUT              : out std_logic_vector(15 downto 0);
    INT_ACK_IN                : in  std_logic;
    INT_DATA_IN               : in  std_logic_vector(31 downto 0);
```



```
    DEBUG_OUT                 : out std_logic_vector(15 downto 0)
    );

end entity;

architecture Behavioral of nx_status_event is

  --Data channel
  signal trigger_busy_o       : std_logic;
  signal event_write_start    : std_logic;

  type STATES is (S_IDLE,
                  S_EVT_WRITE_WAIT
                  );

  signal STATE    : STATES;

  -- Event Write
  type E_STATES is (E_IDLE,
                    E_READ_NEXT,
                    E_READ,
                    E_NEXT_INDEX,
                    E_END
                    );

  signal E_STATE  : E_STATES;

  constant NUM_REGS : integer      := 3;
  type reg_addr_t is array(0 to NUM_REGS - 1) of std_logic_vector(15 downto 0);
  constant reg_addr_start          : reg_addr_t :=
    (x"0000",
     x"0100",
     x"0080"
     );
  constant reg_addr_end            : reg_addr_t :=
    (x"002d",
     x"0180",
     x"0083"
     );

  signal index_ctr                 : unsigned(3 downto 0);
  signal register_addr             : unsigned(15 downto 0);
  signal int_read_o                : std_logic;
  signal int_addr_o                : std_logic_vector(15 downto 0);
  signal fee_data_o                : std_logic_vector(31 downto 0);
  signal fee_data_write_o          : std_logic;
  signal event_write_done          : std_logic;

begin

  DEBUG_OUT(0)           <= CLK_IN;
  DEBUG_OUT(1)           <= TRIGGER_IN;
  DEBUG_OUT(2)           <= FAST_CLEAR_IN;
  DEBUG_OUT(3)           <= FEE_DATA_ALMOST_FULL_IN;
  DEBUG_OUT(4)           <= trigger_busy_o;
  DEBUG_OUT(5)           <= event_write_start;
  DEBUG_OUT(6)           <= event_write_done;
  DEBUG_OUT(10 downto 7) <= index_ctr;
  DEBUG_OUT(11)          <= int_read_o;
  DEBUG_OUT(12)          <= INT_ACK_IN;
  DEBUG_OUT(13)          <= fee_data_write_o;
  DEBUG_OUT(14)          <= '0';
```

```
  DEBUG_OUT(15)          <= NXYTER_OFFLINE_IN;

  ------------------------------------------------------------------------------
  --
  ------------------------------------------------------------------------------

  PROC_DATA_HANDLER: process(CLK_IN)
  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1' ) then
        event_write_start    <= '0';
        trigger_busy_o       <= '0';
        STATE                <= S_IDLE;
      else
        event_write_start    <= '0';
        trigger_busy_o       <= '1';

        if (FAST_CLEAR_IN = '1') then
          STATE                      <= S_IDLE;
        else
          case STATE is
            when S_IDLE =>
              if (NXYTER_OFFLINE_IN = '1') then
                trigger_busy_o             <= '0';
                STATE                      <= S_IDLE;
              elsif (TRIGGER_IN = '1') then
                event_write_start          <= '1';
                STATE                      <= S_EVT_WRITE_WAIT;
              else
                trigger_busy_o             <= '0';
                STATE                      <= S_IDLE;
              end if;

            when S_EVT_WRITE_WAIT =>
              if (event_write_done = '0') then
                STATE                      <= S_EVT_WRITE_WAIT;
              else
                STATE                      <= S_IDLE;
              end if;

          end case;
        end if;
      end if;
    end if;
  end process PROC_DATA_HANDLER;


  PROC_WRITE_EVENT: process(CLK_IN)
    variable index  : integer   := 0;
  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1' ) then
        index_ctr            <= (others => '0');
        register_addr        <= (others => '0');
        int_read_o           <= '0';
        int_addr_o           <= (others => '0');
        fee_data_o           <= (others => '0');
        fee_data_write_o     <= '0';
        event_write_done     <= '0';
        E_STATE              <= E_IDLE;
      else
        index                := to_integer(index_ctr);
```



```
        int_read_o           <= '0';
        int_addr_o           <= (others => '0');
        fee_data_o           <= (others => '0');
        fee_data_write_o     <= '0';
        event_write_done     <= '0';

        case E_STATE is
          when E_IDLE =>
            index_ctr                <= (others => '0');
            if (event_write_start = '1') then
              E_STATE                <= E_NEXT_INDEX;
            else
              E_STATE                <= E_IDLE;
            end if;

          when E_READ_NEXT =>
            if (register_addr <= unsigned(reg_addr_end(index))) then
              int_addr_o             <= register_addr;
              int_read_o             <= '1';
              E_STATE                <= E_READ;
            else
              index_ctr              <= index_ctr + 1;
              E_STATE                <= E_NEXT_INDEX;
            end if;

          when E_READ =>
            if (INT_ACK_IN = '1') then
              fee_data_o(15 downto 0)  <= INT_DATA_IN(15 downto 0);
              fee_data_o(31 downto 16) <= register_addr;
              fee_data_write_o         <= '1';
              register_addr            <= register_addr + 1;
              E_STATE                  <= E_READ_NEXT;
            else
              E_STATE                  <= E_READ;
            end if;

          when E_NEXT_INDEX =>
            if (index_ctr < NUM_REGS) then
              register_addr            <= reg_addr_start(index);
              E_STATE                  <= E_READ_NEXT;
            else
              E_STATE                  <= E_END;
            end if;

          when E_END =>
            event_write_done           <= '1';
            E_STATE                    <= E_IDLE;

        end case;
      end if;
    end if;
  end process PROC_WRITE_EVENT;

  -- Output Signals

  TRIGGER_BUSY_OUT       <= trigger_busy_o;
  FEE_DATA_OUT           <= fee_data_o;
  FEE_DATA_WRITE_OUT     <= fee_data_write_o;

  INT_READ_OUT           <= int_read_o;
  INT_ADDR_OUT           <= int_addr_o;
```

```
end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.nxyter_components.all;

entity nxyter_timestamp_sim is
  port(
    CLK_IN              : in  std_logic;  -- Clock 128MHz
    RESET_IN            : in  std_logic;

    TIMESTAMP_OUT       : out std_logic_vector(7 downto 0);
    CLK128_OUT          : out std_logic
    );
end entity;

architecture Behavioral of nxyter_timestamp_sim is

  signal timestamp_n     : std_logic_vector(7 downto 0);
  signal timestamp_g     : std_logic_vector(7 downto 0);
  signal counter         : unsigned(1 downto 0);
  signal counter2        : unsigned(3 downto 0);
  signal counter3        : unsigned(1 downto 0);

begin

  PROC_NX_TIMESTAMP: process(CLK_IN)
  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1' ) then
        timestamp_n <= (others => '0');
        counter     <= (others => '0');
        counter2    <= (others => '0');
        counter3    <= (others => '0');

      else

        if (counter3 /= 0) then
          case counter is
            when "11" => timestamp_n <= x"06";
                         counter3 <= counter3 + 1;

            when "10" => timestamp_n <= x"7f";

            when "01" => timestamp_n <= x"7f";

            when "00" => timestamp_n <= x"7f";
          end case;

        else
          case counter is
            when "11" =>
              timestamp_n(7)          <= '0';
              timestamp_n(6 downto 4)  <= (others => '0');
              timestamp_n(3 downto 0)  <= counter2;
              counter3 <= counter3 + 1;

            when "10" =>
              timestamp_n(7)          <= '0';
              timestamp_n(6 downto 4)  <= (others => '0');
```



```
              timestamp_n(3 downto 0)  <= counter2;

            when "01" =>
              timestamp_n(7)          <= '0';
              timestamp_n(6 downto 4)  <= (others => '0');
              timestamp_n(3 downto 0)  <= counter2;

            when "00" =>
              timestamp_n(7)          <= '0';
              timestamp_n(6 downto 4)  <= (others => '0');
              timestamp_n(3 downto 0)  <= counter2;

          end case;
          counter2 <= counter2 + 1;
        end if;

        counter  <= counter + 1;
      end if;
    end if;
  end process PROC_NX_TIMESTAMP;

--   gray_Encoder_1: gray_Encoder
--     generic map (
--       WIDTH => 8
--       )
--     port map (
--       CLK_IN    => CLK_IN,
--       RESET_IN  => RESET_IN,
--       BINARY_IN => timestamp_n,
--       GRAY_OUT  => timestamp_g
--       );
--
--  timestamp_g <= timestamp_n;


-- Output Signals
  TIMESTAMP_OUT <= timestamp_n;
  CLK128_OUT    <= CLK_IN;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.nxyter_components.all;

entity nx_trigger_generator is
  port (
    CLK_IN               : in  std_logic;
    RESET_IN             : in  std_logic;
    NX_MAIN_CLK_IN       : in  std_logic;

    TRIGGER_IN           : in  std_logic;  -- must be in NX_MAIN_CLK_DOMAIN
    TRIGGER_OUT          : out std_logic;
    TS_RESET_OUT         : out std_logic;
    TESTPULSE_OUT        : out std_logic;

    TIMESTAMP_IN         : in  std_logic_vector(31 downto 0);
    ADC_DATA_IN          : in  std_logic_vector(11 downto 0);
    NEW_DATA_IN          : in  std_logic;
    SELF_TRIGGER_OUT     : out std_logic;
```

```
    -- Slave bus
    SLV_READ_IN          : in  std_logic;
    SLV_WRITE_IN         : in  std_logic;
    SLV_DATA_OUT         : out std_logic_vector(31 downto 0);
    SLV_DATA_IN          : in  std_logic_vector(31 downto 0);
    SLV_ADDR_IN          : in  std_logic_vector(15 downto 0);
    SLV_ACK_OUT          : out std_logic;
    SLV_NO_MORE_DATA_OUT : out std_logic;
    SLV_UNKNOWN_ADDR_OUT : out std_logic;

    -- Debug Line
    DEBUG_OUT            : out std_logic_vector(15 downto 0)
    );
end entity;

architecture Behavioral of nx_trigger_generator is
  signal start_cycle         : std_logic;
  signal trigger_cycle_ctr   : unsigned(7 downto 0);
  signal wait_timer_start    : std_logic;
  signal wait_timer_init     : unsigned(11 downto 0);
  signal wait_timer_done     : std_logic;
  signal trigger_o           : std_logic;
  signal ts_reset_o          : std_logic;
  signal testpulse_o         : std_logic;
  signal testpulse_o_b       : std_logic;
  signal testpulse_p         : std_logic;
  signal extern_trigger      : std_logic;

  type STATES is (S_IDLE,
                  S_WAIT_TESTPULSE_END
                  );
  signal STATE : STATES;

  -- Rate Calculation
  signal testpulse              : std_logic;
  signal testpulse_rate_t       : unsigned(27 downto 0);
  signal rate_timer             : unsigned(27 downto 0);

  -- Self Trigger
  signal self_trigger           : std_logic;
  signal self_trigger_o         : std_logic;

  -- TRBNet Slave Bus
  signal slv_data_out_o         : std_logic_vector(31 downto 0);
  signal slv_no_more_data_o     : std_logic;
  signal slv_unknown_addr_o     : std_logic;
  signal slv_ack_o              : std_logic;

  signal reg_trigger_period     : unsigned(15 downto 0);
  signal reg_testpulse_length   : unsigned(11 downto 0);
  signal reg_trigger_num_cycles : unsigned(7 downto 0);
  signal reg_ts_reset_on        : std_logic;
  signal testpulse_rate         : unsigned(27 downto 0);

  -- Reset
  signal RESET_NX_MAIN_CLK_IN : std_logic;

begin
  -- Debug Line
  DEBUG_OUT(0)           <= CLK_IN;
  DEBUG_OUT(1)           <= NEW_DATA_IN;
```



```
  DEBUG_OUT(2)           <= start_cycle;
  DEBUG_OUT(3)           <= ts_reset_o;
  DEBUG_OUT(4)           <= testpulse_o_b;
  DEBUG_OUT(5)           <= testpulse;
  DEBUG_OUT(6)           <= self_trigger;
  DEBUG_OUT(7)           <= self_trigger_o;
  DEBUG_OUT(15 downto 8) <= (others => '0');
  ------------------------------------------------------------------------------
  -- Reset Domain Transfer
  ------------------------------------------------------------------------------
  signal_async_trans_RESET_IN: signal_async_trans
    port map (
      CLK_IN      => NX_MAIN_CLK_IN,
      SIGNAL_A_IN => RESET_IN,
      SIGNAL_OUT  => RESET_NX_MAIN_CLK_IN
    );

  ------------------------------------------------------------------------------

  -- Timer
  timer_1: timer
    generic map (
      CTR_WIDTH => 12
      )
    port map (
      CLK_IN         => NX_MAIN_CLK_IN,
      RESET_IN       => RESET_NX_MAIN_CLK_IN,
      TIMER_START_IN => wait_timer_start,
      TIMER_END_IN   => wait_timer_init,
      TIMER_DONE_OUT => wait_timer_done
      );
  wait_timer_init   <= reg_testpulse_length - 1;

  ------------------------------------------------------------------------------
  -- Generate Trigger
  ------------------------------------------------------------------------------

  PROC_TESTPULSE_OUT: process(NX_MAIN_CLK_IN)
  begin
    if( rising_edge(NX_MAIN_CLK_IN) ) then
      if (RESET_NX_MAIN_CLK_IN = '1') then
        trigger_o         <= '0';
        testpulse_o       <= '0';
        testpulse_p       <= '0';
        ts_reset_o        <= '0';
        wait_timer_start  <= '0';
        trigger_cycle_ctr <= (others => '0');
        extern_trigger    <= '0';
        STATE             <= S_IDLE;
      else
        trigger_o         <= '0';
        testpulse_o       <= '0';
        testpulse_p       <= '0';
        ts_reset_o        <= '0';
        wait_timer_start  <= '0';

        case STATE is
          when  S_IDLE =>
            if (TRIGGER_IN = '1') then
              extern_trigger                <= '1';
              testpulse_o                   <= '1';
              testpulse_p                   <= '1';
```

```
            if (reg_testpulse_length > 0) then
              wait_timer_start              <= '1';
              STATE                         <= S_WAIT_TESTPULSE_END;
            else
              STATE                         <= S_IDLE;
            end if;
          else
            extern_trigger                  <= '0';
            STATE                           <= S_IDLE;
          end if;

        when S_WAIT_TESTPULSE_END =>
          if (WAIT_TIMER_DONE = '0') then
            testpulse_o                     <= '1';
            STATE                           <= S_WAIT_TESTPULSE_END;
          else
            STATE                           <= S_IDLE;
          end if;

      end case;
    end if;
  end if;
end process PROC_TESTPULSE_OUT;

-- Transfer testpulse_p to CLK_IN Domain
pulse_dtrans_TESTPULSE: pulse_dtrans
  generic map (
    CLK_RATIO => 6
    )
  port map (
    CLK_A_IN    => NX_MAIN_CLK_IN,
    RESET_A_IN  => RESET_NX_MAIN_CLK_IN,
    PULSE_A_IN  => testpulse_p,
    CLK_B_IN    => CLK_IN,
    RESET_B_IN  => RESET_IN,
    PULSE_B_OUT => testpulse
    );

PROC_CAL_RATES: process (CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if (RESET_IN = '1') then
      testpulse_rate_t          <= (others => '0');
      testpulse_rate            <= (others => '0');
      rate_timer                <= (others => '0');
    else
      if (rate_timer < x"5f5e100") then
        if (testpulse = '1') then
          testpulse_rate_t            <= testpulse_rate_t + 1;
        end if;
        rate_timer                    <= rate_timer + 1;
      else
        testpulse_rate                <= testpulse_rate_t;

        testpulse_rate_t(27 downto 1) <= (others => '0');
        testpulse_rate_t(0)           <= testpulse;

        rate_timer                    <= (others => '0');
      end if;
    end if;
  end if;
end process PROC_CAL_RATES;
```



```
------------------------------------------------------------------------------
-- Self Trigger
------------------------------------------------------------------------------

PROC_SELF_TRIGGER: process(CLK_IN)
  variable frame_bits : std_logic_vector(3 downto 0);

begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      self_trigger       <= '0';
    else
      frame_bits := TIMESTAMP_IN(31) &
                    TIMESTAMP_IN(23) &
                    TIMESTAMP_IN(15) &
                    TIMESTAMP_IN(7);
      if (NEW_DATA_IN = '1' and
          frame_bits = "1000") then
        self_trigger     <= '1';
      else
        self_trigger     <= '0';
      end if;
    end if;
  end if;
end process PROC_SELF_TRIGGER;

pulse_to_level_SELF_TRIGGER: pulse_to_level
  generic map (
    NUM_CYCLES => 2
    )
  port map (
    CLK_IN    => CLK_IN,
    RESET_IN  => RESET_IN,
    PULSE_IN  => self_trigger,
    LEVEL_OUT => self_trigger_o
    );

------------------------------------------------------------------------------
-- TRBNet Slave Bus
------------------------------------------------------------------------------

PROC_SLAVE_BUS: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      reg_trigger_period     <= x"00ff";
      reg_trigger_num_cycles <= x"01";
      reg_testpulse_length   <= x"064";
      reg_ts_reset_on        <= '0';
      slv_data_out_o         <= (others => '0');
      slv_no_more_data_o     <= '0';
      slv_unknown_addr_o     <= '0';
      start_cycle            <= '0';
      slv_ack_o              <= '0';
    else
      slv_unknown_addr_o <= '0';
      slv_no_more_data_o <= '0';
      slv_data_out_o     <= (others => '0');
      start_cycle        <= '0';
```

```
        if (SLV_WRITE_IN  = '1') then
          case SLV_ADDR_IN is
            when x"0000" =>
              if (unsigned(SLV_DATA_IN(11 downto 0)) > 0) then
                reg_testpulse_length       <=
                  unsigned(SLV_DATA_IN(11 downto 0));
              end if;
              slv_ack_o                    <= '1';

            when others =>
              slv_unknown_addr_o           <= '1';
              slv_ack_o                    <= '0';
          end case;

        elsif (SLV_READ_IN = '1') then
          case SLV_ADDR_IN is
            when x"0000" =>
              slv_data_out_o(11 downto 0)  <=
                std_logic_vector(reg_testpulse_length);
              slv_data_out_o(31 downto 12) <= (others => '0');
              slv_ack_o                    <= '1';

            when x"0001" =>
              slv_data_out_o(27 downto 0)  <= std_logic_vector(testpulse_rate);
              slv_data_out_o(31 downto 28) <= (others => '0');
              slv_ack_o                    <= '1';

            when others =>
              slv_unknown_addr_o           <= '1';
              slv_ack_o                    <= '0';
          end case;

        else
          slv_ack_o                        <= '0';
        end if;
      end if;
    end if;
  end process PROC_SLAVE_BUS;

  -------------------------------------------------------------------------------
  -- Output Signals
  -------------------------------------------------------------------------------

  -- Buffer for timing
  testpulse_o_b        <= testpulse_o when rising_edge(NX_MAIN_CLK_IN);

  -- Trigger Output
  TRIGGER_OUT          <= trigger_o;
  TS_RESET_OUT         <= ts_reset_o;
  TESTPULSE_OUT        <= testpulse_o_b;
  SELF_TRIGGER_OUT     <= self_trigger_o;

  -- Slave Bus
  SLV_DATA_OUT         <= slv_data_out_o;
  SLV_NO_MORE_DATA_OUT <= slv_no_more_data_o;
  SLV_UNKNOWN_ADDR_OUT <= slv_unknown_addr_o;
  SLV_ACK_OUT          <= slv_ack_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```



```
library work;
use work.nxyter_components.all;

entity nx_trigger_handler is
  port (
    CLK_IN                     : in  std_logic;
    RESET_IN                   : in  std_logic;
    NX_MAIN_CLK_IN             : in  std_logic;
    NXYTER_OFFLINE_IN          : in  std_logic;

    --Input Triggers
    TIMING_TRIGGER_IN          : in std_logic; -- The raw timing Trigger Signal
    LVL1_TRG_DATA_VALID_IN     : in std_logic; -- Data Trigger is valid
    LVL1_VALID_TIMING_TRG_IN   : in std_logic; -- Timin Trigger is valid
    LVL1_VALID_NOTIMING_TRG_IN : in std_logic; -- calib trigger w/o ref time
    LVL1_INVALID_TRG_IN        : in std_logic;

    LVL1_TRG_TYPE_IN           : in std_logic_vector(3 downto 0);
    LVL1_TRG_NUMBER_IN         : in std_logic_vector(15 downto 0);
    LVL1_TRG_CODE_IN           : in std_logic_vector(7 downto 0);
    LVL1_TRG_INFORMATION_IN    : in std_logic_vector(23 downto 0);
    LVL1_INT_TRG_NUMBER_IN     : in std_logic_vector(15 downto 0);

    --Response from FEE
    FEE_DATA_OUT               : out std_logic_vector(31 downto 0);
    FEE_DATA_WRITE_OUT         : out std_logic;
    FEE_DATA_FINISHED_OUT      : out std_logic;
    FEE_TRG_RELEASE_OUT        : out std_logic;
    FEE_TRG_STATUSBITS_OUT     : out std_logic_vector(31 downto 0);

    FEE_DATA_0_IN              : in  std_logic_vector(31 downto 0);
    FEE_DATA_WRITE_0_IN        : in  std_logic;
    FEE_DATA_1_IN              : in  std_logic_vector(31 downto 0);
    FEE_DATA_WRITE_1_IN        : in  std_logic;

    -- Internal FPGA Trigger
    INTERNAL_TRIGGER_IN        : in  std_logic;

    -- Trigger FeedBack
    TRIGGER_VALIDATE_BUSY_IN   : in  std_logic;
    TRIGGER_BUSY_0_IN          : in  std_logic;
    TRIGGER_BUSY_1_IN          : in  std_logic;

    -- OUT
    VALID_TRIGGER_OUT          : out std_logic;
    TIMESTAMP_TRIGGER_OUT      : out std_logic;
    TRIGGER_TIMING_OUT         : out std_logic;
    TRIGGER_STATUS_OUT         : out std_logic;
    FAST_CLEAR_OUT             : out std_logic;
    TRIGGER_BUSY_OUT           : out std_logic;

    -- Pulser
    TRIGGER_TESTPULSE_OUT      : out std_logic;

    -- Slave bus
    SLV_READ_IN                : in  std_logic;
    SLV_WRITE_IN               : in  std_logic;
    SLV_DATA_OUT               : out std_logic_vector(31 downto 0);
    SLV_DATA_IN                : in  std_logic_vector(31 downto 0);
    SLV_ADDR_IN                : in  std_logic_vector(15 downto 0);
    SLV_ACK_OUT                : out std_logic;
```

```
    SLV_NO_MORE_DATA_OUT       : out std_logic;
    SLV_UNKNOWN_ADDR_OUT       : out std_logic;

    -- Debug Line
    DEBUG_OUT                  : out std_logic_vector(15 downto 0)
    );
end entity;

architecture Behavioral of nx_trigger_handler is

  -- Timing Trigger Handler
  constant NUM_FF                   : integer := 10;
  signal timing_trigger_ff_p        : std_logic_vector(1 downto 0);
  signal timing_trigger_ff          : std_logic_vector(NUM_FF - 1 downto 0);
  signal timing_trigger_l           : std_logic;
  signal timing_trigger             : std_logic;
  signal timing_trigger_set         : std_logic;
  signal timestamp_trigger_o        : std_logic;

  signal invalid_timing_trigger_n   : std_logic;
  signal invalid_timing_trigger     : std_logic;
  signal invalid_timing_trigger_ctr : unsigned(15 downto 0);

  signal trigger_busy               : std_logic;
  signal fast_clear                 : std_logic;

  type TS_STATES is (TS_IDLE,
                     TS_WAIT_VALID_TIMING_TRIGGER,
                     TS_INVALID_TRIGGER,
                     TS_WAIT_TRIGGER_END
                     );
  signal TS_STATE : TS_STATES;

  signal ts_wait_timer_reset        : std_logic;
  signal ts_wait_timer_start        : std_logic;
  signal ts_wait_timer_done         : std_logic;

  -- Trigger Handler
  signal valid_trigger_o            : std_logic;
  signal timing_trigger_o           : std_logic;
  signal status_trigger_o           : std_logic;
  signal fast_clear_o               : std_logic;
  signal trigger_busy_o             : std_logic;
  signal fee_data_o                 : std_logic_vector(31 downto 0);
  signal fee_data_write_o           : std_logic;
  signal fee_data_finished_o        : std_logic;
  signal fee_trg_release_o          : std_logic;
  signal fee_trg_statusbits_o       : std_logic_vector(31 downto 0);
  signal send_testpulse             : std_logic;

  type STATES is (S_IDLE,
                  S_CTS_TRIGGER,
                  S_WAIT_TRG_DATA_VALID,
                  S_WAIT_TIMING_TRIGGER_DONE,
                  S_FEE_TRIGGER_RELEASE,
                  S_WAIT_FEE_TRIGGER_RELEASE_ACK,
                  S_INTERNAL_TRIGGER,
                  S_WAIT_TRIGGER_VALIDATE_ACK,
                  S_WAIT_TRIGGER_VALIDATE_DONE
                  );
  signal STATE : STATES;
```



```
  type TRIGGER_TYPES is (T_UNDEF,
                         T_IGNORE,
                         T_INTERNAL,
                         T_TIMING,
                         T_SETUP
                         );
  signal TRIGGER_TYPE : TRIGGER_TYPES;


  -- Testpulse Handler
  type T_STATES is (T_IDLE,
                    T_WAIT_TIMER,
                    T_SET_TESTPULSE
                    );

  signal T_STATE : T_STATES;

  signal trigger_testpulse_o         : std_logic;
  signal wait_timer_reset            : std_logic;
  signal wait_timer_start            : std_logic;
  signal wait_timer_done             : std_logic;

  -- Rate Calculation
  signal accepted_trigger_rate_t     : unsigned(27 downto 0);
  signal rate_timer                  : unsigned(27 downto 0);

  -- TRBNet Slave Bus
  signal slv_data_out_o              : std_logic_vector(31 downto 0);
  signal slv_no_more_data_o          : std_logic;
  signal slv_unknown_addr_o          : std_logic;
  signal slv_ack_o                   : std_logic;

  signal reg_testpulse_delay         : unsigned(11 downto 0);
  signal reg_testpulse_enable        : std_logic;
  signal accepted_trigger_rate       : unsigned(27 downto 0);
  signal invalid_t_trigger_ctr_clear : std_logic;

  -- Reset
  signal RESET_NX_MAIN_CLK_IN        : std_logic;

begin

  -- Debug Line
  DEBUG_OUT(0)            <= CLK_IN;
  DEBUG_OUT(1)            <= TIMING_TRIGGER_IN;
  DEBUG_OUT(2)            <= invalid_timing_trigger;
  DEBUG_OUT(3)            <= LVL1_VALID_TIMING_TRG_IN;
  DEBUG_OUT(4)            <= LVL1_TRG_DATA_VALID_IN;
  DEBUG_OUT(5)            <= fee_data_write_o;
  DEBUG_OUT(6)            <= TRIGGER_VALIDATE_BUSY_IN;
  DEBUG_OUT(7)            <= TRIGGER_BUSY_0_IN;
  DEBUG_OUT(8)            <= valid_trigger_o;
  DEBUG_OUT(9)            <= timing_trigger_o;
  DEBUG_OUT(10)           <= fee_data_finished_o;
  DEBUG_OUT(11)           <= fee_trg_release_o;
  DEBUG_OUT(12)           <= trigger_busy_o;
  DEBUG_OUT(13)           <= timestamp_trigger_o;
  DEBUG_OUT(14)           <= send_testpulse;
  DEBUG_OUT(15)           <= trigger_testpulse_o;

  ------------------------------------------------------------------------------
  -- Reset Domain Transfer
```

```
--------------------------------------------------------------------------------
signal_async_trans_RESET_IN: signal_async_trans
  port map (
    CLK_IN      => NX_MAIN_CLK_IN,
    SIGNAL_A_IN => RESET_IN,
    SIGNAL_OUT  => RESET_NX_MAIN_CLK_IN
  );

--------------------------------------------------------------------------------
-- Trigger Handler
--------------------------------------------------------------------------------

PROC_TIMING_TRIGGER_HANDLER: process(NX_MAIN_CLK_IN)
  constant pattern : std_logic_vector(NUM_FF - 1 downto 0)
  := (others => '1');
begin
  if( rising_edge(NX_MAIN_CLK_IN) ) then
    timing_trigger_ff_p(1)                  <= TIMING_TRIGGER_IN;
    if (RESET_NX_MAIN_CLK_IN = '1') then
      timing_trigger_ff_p(0)                <= '0';
      timing_trigger_ff(NUM_FF - 1 downto 0) <= (others => '0');
      timing_trigger_l                      <= '0';
    else
      timing_trigger_ff_p(0)                <= timing_trigger_ff_p(1);
      timing_trigger_ff(NUM_FF - 1)         <= timing_trigger_ff_p(0);

      for I in NUM_FF - 2 downto 0 loop
        timing_trigger_ff(I)                <= timing_trigger_ff(I + 1);
      end loop;

      if (timing_trigger_ff = pattern) then
        timing_trigger_l                    <= '1';
      else
        timing_trigger_l                    <= '0';
      end if;
    end if;
  end if;
end process PROC_TIMING_TRIGGER_HANDLER;

level_to_pulse_1: level_to_pulse
  port map (
    CLK_IN    => NX_MAIN_CLK_IN,
    RESET_IN  => RESET_NX_MAIN_CLK_IN,
    LEVEL_IN  => timing_trigger_l,
    PULSE_OUT => timing_trigger
    );

-- Timer
timer_static_2: timer_static
  generic map (
    CTR_WIDTH => 8,
    CTR_END   => 32   -- 128ns
    )
  port map (
    CLK_IN         => NX_MAIN_CLK_IN,
    RESET_IN       => ts_wait_timer_reset,
    TIMER_START_IN => ts_wait_timer_start,
    TIMER_DONE_OUT => ts_wait_timer_done
    );

PROC_TIMING_TRIGGER_HANDLER: process(NX_MAIN_CLK_IN)
begin
```



```
  if( rising_edge(NX_MAIN_CLK_IN) ) then
    if (RESET_NX_MAIN_CLK_IN = '1') then
      invalid_timing_trigger_n   <= '1';
      ts_wait_timer_start        <= '0';
      ts_wait_timer_reset        <= '1';
      send_testpulse             <= '0';
      timestamp_trigger_o        <= '0';
      TS_STATE                   <= TS_IDLE;
    else
      invalid_timing_trigger_n   <= '0';
      ts_wait_timer_start        <= '0';
      ts_wait_timer_reset        <= '0';
      send_testpulse             <= '0';
      timestamp_trigger_o        <= '0';

      if (fast_clear = '1') then
        ts_wait_timer_reset      <= '1';
        TS_STATE                 <= TS_IDLE;
      else
        case TS_STATE is
          when  TS_IDLE =>
            -- Wait for Timing Trigger synced to NX_MAIN_CLK_DOMAIN
            if (timing_trigger = '1') then
              if (trigger_busy = '1') then
                -- If busy is set --> Error
                TS_STATE               <= TS_INVALID_TRIGGER;
              else
                if (reg_testpulse_enable = '1') then
                  send_testpulse       <= '1';
                end if;
                timestamp_trigger_o    <= '1';
                ts_wait_timer_start    <= '1';
                TS_STATE               <= TS_WAIT_VALID_TIMING_TRIGGER;
              end if;
            else
              TS_STATE                 <= TS_IDLE;
            end if;

          when TS_WAIT_VALID_TIMING_TRIGGER =>
            -- Wait and test if CLK_IN Trigger Handler does accepted Trigger
            if (trigger_busy = '1') then
              -- Trigger has been accepted, stop timer and wait trigger end
              ts_wait_timer_reset      <= '1';
              TS_STATE                 <= TS_WAIT_TRIGGER_END;
            else
              if (ts_wait_timer_done = '1') then
                -- Timeout after 128ns --> Invalid Trigger Error
                TS_STATE               <= TS_INVALID_TRIGGER;
              else
                TS_STATE               <= TS_WAIT_VALID_TIMING_TRIGGER;
              end if;
            end if;

          when TS_INVALID_TRIGGER =>
            invalid_timing_trigger_n   <= '1';
            TS_STATE                   <= TS_IDLE;

          when TS_WAIT_TRIGGER_END =>
            if (trigger_busy = '0') then
              TS_STATE                 <= TS_IDLE;
            else
              TS_STATE                 <= TS_WAIT_TRIGGER_END;
```

```
            end if;

        end case;
      end if;
    end if;
  end if;
end process PROC_TIMING_TRIGGER_HANDLER;

PROC_TIMING_TRIGGER_COUNTER: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if (RESET_IN = '1') then
      invalid_timing_trigger_ctr    <= (others => '0');
    else
      if (invalid_t_trigger_ctr_clear = '1') then
        invalid_timing_trigger_ctr  <= (others => '0');
      elsif (invalid_timing_trigger = '1') then
        invalid_timing_trigger_ctr  <= invalid_timing_trigger_ctr + 1;
      end if;
    end if;
  end if;
end process PROC_TIMING_TRIGGER_COUNTER;

signal_async_trans_TRIGGER_BUSY: signal_async_trans
  port map (
    CLK_IN      => NX_MAIN_CLK_IN,
    SIGNAL_A_IN => trigger_busy_o,
    SIGNAL_OUT  => trigger_busy
    );

signal_async_to_pulse_FAST_CLEAR: signal_async_to_pulse
  generic map (
    NUM_FF => 2
    )
  port map (
    CLK_IN     => NX_MAIN_CLK_IN,
    RESET_IN   => RESET_NX_MAIN_CLK_IN,
    PULSE_A_IN => fast_clear_o,
    PULSE_OUT  => fast_clear
    );

pulse_dtrans_INVALID_TIMING_TRIGGER: pulse_dtrans
  generic map (
    CLK_RATIO => 4
    )
  port map (
    CLK_A_IN    => NX_MAIN_CLK_IN,
    RESET_A_IN  => RESET_NX_MAIN_CLK_IN,
    PULSE_A_IN  => invalid_timing_trigger_n,
    CLK_B_IN    => CLK_IN,
    RESET_B_IN  => RESET_IN,
    PULSE_B_OUT => invalid_timing_trigger
    );

-------------------------------------------------------------------------------

PROC_TRIGGER_HANDLER: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if (RESET_IN = '1') then
      valid_trigger_o      <= '0';
      timing_trigger_o     <= '0';
```



```
      status_trigger_o     <= '0';
      fee_data_finished_o  <= '0';
      fee_trg_release_o    <= '0';
      fee_trg_statusbits_o <= (others => '0');
      fast_clear_o         <= '0';
      trigger_busy_o       <= '0';
      TRIGGER_TYPE         <= T_UNDEF;
      STATE                <= S_IDLE;
    else
      valid_trigger_o      <= '0';
      timing_trigger_o     <= '0';
      status_trigger_o     <= '0';
      fee_data_finished_o  <= '0';
      fee_trg_release_o    <= '0';
      fee_trg_statusbits_o <= (others => '0');
      fast_clear_o         <= '0';
      trigger_busy_o       <= '1';

      if (LVL1_INVALID_TRG_IN = '1') then
        -- There was no valid Timing Trigger at CTS, do a fast clear
        fast_clear_o               <= '1';
        fee_trg_release_o          <= '1';
        STATE                      <= S_IDLE;
      else
        case STATE is
          when  S_IDLE =>
            if (LVL1_VALID_NOTIMING_TRG_IN = '1') then
              -- Calibration Trigger .. ignore
              TRIGGER_TYPE         <= T_IGNORE; --T_SETUP;
              STATE                <= S_WAIT_TRG_DATA_VALID;

            elsif (LVL1_VALID_TIMING_TRG_IN = '1') then
              if (NXYTER_OFFLINE_IN = '0') then
                -- Normal Trigger
                TRIGGER_TYPE       <= T_TIMING;
                STATE              <= S_CTS_TRIGGER;
              else
                -- Ignore Trigger for nxyter is offline
               TRIGGER_TYPE        <= T_IGNORE;
               STATE               <= S_WAIT_TRG_DATA_VALID;
              end if;
            elsif (INTERNAL_TRIGGER_IN = '1') then
              -- Internal Trigger, not defined yet
              TRIGGER_TYPE         <= T_INTERNAL;
              STATE                <= S_INTERNAL_TRIGGER;
            else
              trigger_busy_o       <= '0';
              TRIGGER_TYPE         <= T_UNDEF;
              STATE                <= S_IDLE;
            end if;

          when S_CTS_TRIGGER =>
            valid_trigger_o        <= '1';
            timing_trigger_o       <= '1';
            STATE                  <= S_WAIT_TRG_DATA_VALID;

          when S_WAIT_TRG_DATA_VALID =>
            if (LVL1_TRG_DATA_VALID_IN = '0') then
              STATE                <= S_WAIT_TRG_DATA_VALID;
            else
              STATE                <= S_WAIT_TIMING_TRIGGER_DONE;
            end if;
```

```
          when S_WAIT_TIMING_TRIGGER_DONE =>
            if (TRIGGER_BUSY_0_IN = '1') then
              STATE                <= S_WAIT_TIMING_TRIGGER_DONE;
            else
              fee_data_finished_o  <= '1';
              STATE                <= S_FEE_TRIGGER_RELEASE;
            end if;

          when S_FEE_TRIGGER_RELEASE =>
            fee_trg_release_o      <= '1';
            STATE                  <= S_WAIT_FEE_TRIGGER_RELEASE_ACK;

          when S_WAIT_FEE_TRIGGER_RELEASE_ACK =>
            if (LVL1_TRG_DATA_VALID_IN = '1') then
              STATE                <= S_WAIT_FEE_TRIGGER_RELEASE_ACK;
            else
              STATE                <= S_IDLE;
            end if;

            -- Internal Trigger Handler
          when S_INTERNAL_TRIGGER =>
            valid_trigger_o        <= '1';
            STATE                  <= S_WAIT_TRIGGER_VALIDATE_ACK;

          when S_WAIT_TRIGGER_VALIDATE_ACK =>
            if (TRIGGER_VALIDATE_BUSY_IN = '0') then
              STATE                <= S_WAIT_TRIGGER_VALIDATE_ACK;
            else
              STATE                <= S_WAIT_TRIGGER_VALIDATE_DONE;
            end if;

          when S_WAIT_TRIGGER_VALIDATE_DONE =>
            if (TRIGGER_VALIDATE_BUSY_IN = '1') then
              STATE                <= S_WAIT_TRIGGER_VALIDATE_DONE;
            else
              STATE                <= S_IDLE;
            end if;

        end case;
      end if;
    end if;
  end if;
end process PROC_TRIGGER_HANDLER;

PROC_EVENT_DATA_MULTIPLEXER: process(TRIGGER_TYPE)
begin
  case TRIGGER_TYPE is
    when  T_UNDEF | T_IGNORE | T_INTERNAL =>
      fee_data_o                   <= (others => '0');
      fee_data_write_o             <= '0';

    when T_TIMING =>
      fee_data_o                   <= FEE_DATA_0_IN;
      fee_data_write_o             <= FEE_DATA_WRITE_0_IN;

    when T_SETUP =>
      fee_data_o                   <= FEE_DATA_1_IN;
      fee_data_write_o             <= FEE_DATA_WRITE_1_IN;

  end case;
end process PROC_EVENT_DATA_MULTIPLEXER;
```



```
timer_1: timer
  generic map (
    CTR_WIDTH => 12
    )
  port map (
    CLK_IN         => NX_MAIN_CLK_IN,
    RESET_IN       => wait_timer_reset,
    TIMER_START_IN => wait_timer_start,
    TIMER_END_IN   => reg_testpulse_delay,
    TIMER_DONE_OUT => wait_timer_done
    );

PROC_TESTPULSE_HANDLER: process (NX_MAIN_CLK_IN)
begin
  if( rising_edge(NX_MAIN_CLK_IN) ) then
    if (RESET_NX_MAIN_CLK_IN = '1') then
      wait_timer_start     <= '0';
      wait_timer_reset     <= '1';
      trigger_testpulse_o  <= '0';
      T_STATE              <= T_IDLE;
    else
      trigger_testpulse_o  <= '0';
      wait_timer_start     <= '0';
      wait_timer_reset     <= '0';

      if (fast_clear = '1') then
        wait_timer_reset   <= '1';
        T_STATE            <= T_IDLE;
      else
        case T_STATE is

          when T_IDLE =>
            if (send_testpulse = '1') then
              if (reg_testpulse_delay > 0) then
                wait_timer_start <= '1';
                T_STATE          <= T_WAIT_TIMER;
              else
                T_STATE          <= T_SET_TESTPULSE;
              end if;
            else
              T_STATE            <= T_IDLE;
            end if;

          when T_WAIT_TIMER =>
            if (wait_timer_done = '0') then
              T_STATE            <= T_WAIT_TIMER;
            else
              T_STATE            <= T_SET_TESTPULSE;
            end if;

          when T_SET_TESTPULSE =>
            trigger_testpulse_o  <= '1';
            T_STATE              <= T_IDLE;
        end case;
      end if;
    end if;
  end if;
end process PROC_TESTPULSE_HANDLER;

PROC_CAL_RATES: process (CLK_IN)
begin
```

```
    if( rising_edge(CLK_IN) ) then
      if (RESET_IN = '1') then
        accepted_trigger_rate_t     <= (others => '0');
        accepted_trigger_rate       <= (others => '0');
        rate_timer                  <= (others => '0');
      else
        if (rate_timer < x"5f5e100") then
          if (timing_trigger_o = '1') then
            accepted_trigger_rate_t           <= accepted_trigger_rate_t + 1;
          end if;
          rate_timer                          <= rate_timer + 1;
        else
          rate_timer                          <= (others => '0');
          accepted_trigger_rate               <= accepted_trigger_rate_t;

          accepted_trigger_rate_t(27 downto 0) <= (others => '0');
          accepted_trigger_rate_t(0)          <= timing_trigger_o;
        end if;
      end if;
    end if;
  end process PROC_CAL_RATES;

  ------------------------------------------------------------------------------
  -- TRBNet Slave Bus
  ------------------------------------------------------------------------------

  PROC_SLAVE_BUS: process(CLK_IN)
  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1' ) then
        slv_data_out_o                <= (others => '0');
        slv_no_more_data_o            <= '0';
        slv_unknown_addr_o            <= '0';
        slv_ack_o                     <= '0';
        reg_testpulse_delay           <= (others => '0');
        reg_testpulse_enable          <= '0';
        invalid_t_trigger_ctr_clear   <= '1';
      else
        slv_unknown_addr_o            <= '0';
        slv_no_more_data_o            <= '0';
        slv_data_out_o                <= (others => '0');
        slv_ack_o                     <= '0';
        invalid_t_trigger_ctr_clear   <= '1';

        if (SLV_WRITE_IN  = '1') then
          case SLV_ADDR_IN is
            when x"0000" =>
              reg_testpulse_enable        <= SLV_DATA_IN(0);
              slv_ack_o                   <= '1';

            when x"0001" =>
              reg_testpulse_delay       <=
                unsigned(SLV_DATA_IN(11 downto 0));
              slv_ack_o                   <= '1';

            when x"0003" =>
              invalid_t_trigger_ctr_clear  <= '1';
              slv_ack_o                   <= '1';

            when others =>
              slv_unknown_addr_o          <= '1';
```



```
          end case;

        elsif (SLV_READ_IN = '1') then
          case SLV_ADDR_IN is

            when x"0000" =>
              slv_data_out_o(0)           <= reg_testpulse_enable;
              slv_data_out_o(31 downto 1) <= (others => '0');
              slv_ack_o                   <= '1';

            when x"0001" =>
              slv_data_out_o(11 downto 0)  <=
                std_logic_vector(reg_testpulse_delay);
              slv_data_out_o(31 downto 12) <= (others => '0');
              slv_ack_o                   <= '1';

            when x"0002" =>
              slv_data_out_o(27 downto 0)  <=
                std_logic_vector(accepted_trigger_rate);
              slv_data_out_o(31 downto 28) <= (others => '0');
              slv_ack_o                   <= '1';

            when x"0003" =>
              slv_data_out_o(15 downto 0)  <=
                std_logic_vector(invalid_timing_trigger_ctr);
              slv_data_out_o(31 downto 26) <= (others => '0');
              slv_ack_o                   <= '1';

            when others =>
              slv_unknown_addr_o          <= '1';

          end case;

        end if;
      end if;
    end if;
  end process PROC_SLAVE_BUS;

  ------------------------------------------------------------------------------
  -- Output Signals
  ------------------------------------------------------------------------------

  -- Trigger Output
  VALID_TRIGGER_OUT         <= valid_trigger_o;
  TIMESTAMP_TRIGGER_OUT     <= timestamp_trigger_o;
  TRIGGER_TIMING_OUT        <= timing_trigger_o;
  TRIGGER_STATUS_OUT        <= status_trigger_o;
  FAST_CLEAR_OUT            <= fast_clear_o;
  TRIGGER_BUSY_OUT          <= trigger_busy_o;

  FEE_DATA_OUT              <= fee_data_o;
  FEE_DATA_WRITE_OUT        <= fee_data_write_o;
  FEE_DATA_FINISHED_OUT     <= fee_data_finished_o;
  FEE_TRG_RELEASE_OUT       <= fee_trg_release_o;
  FEE_TRG_STATUSBITS_OUT    <= fee_trg_statusbits_o;

  TRIGGER_TESTPULSE_OUT     <= trigger_testpulse_o;

  -- Slave Bus
  SLV_DATA_OUT              <= slv_data_out_o;
  SLV_NO_MORE_DATA_OUT      <= slv_no_more_data_o;
  SLV_UNKNOWN_ADDR_OUT      <= slv_unknown_addr_o;
```

```
  SLV_ACK_OUT               <= slv_ack_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.nxyter_components.all;

entity nx_trigger_validate is
  generic (
    BOARD_ID : std_logic_vector(1 downto 0) := "11"
    );
  port (
    CLK_IN                 : in  std_logic;
    RESET_IN               : in  std_logic;

    -- Inputs
    DATA_CLK_IN            : in  std_logic;
    TIMESTAMP_IN           : in  std_logic_vector(13 downto 0);
    CHANNEL_IN             : in  std_logic_vector(6 downto 0);
    TIMESTAMP_STATUS_IN    : in  std_logic_vector(2 downto 0);  -- 2: Parity Err
    ADC_DATA_IN            : in  std_logic_vector(11 downto 0); -- 1: Pileup
    NX_TOKEN_RETURN_IN     : in  std_logic;                     -- 0: Ovfl
    NX_NOMORE_DATA_IN      : in  std_logic;

    TRIGGER_IN             : in  std_logic;
    TRIGGER_BUSY_IN        : in  std_logic;
    FAST_CLEAR_IN          : in  std_logic;
    TRIGGER_BUSY_OUT       : out std_logic;
    TIMESTAMP_FPGA_IN      : in  unsigned(11 downto 0);
    DATA_FIFO_DELAY_OUT    : out std_logic_vector(7 downto 0);

    -- Event Buffer I/O
    DATA_OUT               : out std_logic_vector(31 downto 0);
    DATA_CLK_OUT           : out std_logic;
    NOMORE_DATA_OUT        : out std_logic;
    EVT_BUFFER_CLEAR_OUT   : out std_logic;
    EVT_BUFFER_FULL_IN     : in  std_logic;

    -- Histogram
    HISTOGRAM_FILL_OUT     : out std_logic;
    HISTOGRAM_BIN_OUT      : out std_logic_vector(6 downto 0);
    HISTOGRAM_ADC_OUT      : out std_logic_vector(11 downto 0);
    HISTOGRAM_PILEUP_OUT   : out std_logic;
    HISTOGRAM_OVERFLOW_OUT : out std_logic;

    -- Slave bus
    SLV_READ_IN            : in  std_logic;
    SLV_WRITE_IN           : in  std_logic;
    SLV_DATA_OUT           : out std_logic_vector(31 downto 0);
    SLV_DATA_IN            : in  std_logic_vector(31 downto 0);
    SLV_ADDR_IN            : in  std_logic_vector(15 downto 0);
    SLV_ACK_OUT            : out std_logic;
    SLV_NO_MORE_DATA_OUT   : out std_logic;
    SLV_UNKNOWN_ADDR_OUT   : out std_logic;

    DEBUG_OUT              : out std_logic_vector(15 downto 0)
    );

end entity;
```



```
architecture Behavioral of nx_trigger_validate is
  constant VERSION_NUMBER      : std_logic_vector(3 downto 0) := x"1";

  constant S_PARITY            : integer := 2;
  constant S_PILEUP            : integer := 1;
  constant S_OVFL              : integer := 0;

  -- Process Channel_Status
  signal channel_index         : std_logic_vector(6 downto 0);
  signal channel_wait          : std_logic_vector(127 downto 0);
  signal channel_done          : std_logic_vector(127 downto 0);
  signal channel_hit           : std_logic_vector(127 downto 0);
  signal channel_all_done      : std_logic;

  signal channel_done_r        : std_logic_vector(127 downto 0);
  signal channel_wait_r        : std_logic_vector(127 downto 0);
  signal channel_hit_r         : std_logic_vector(127 downto 0);
  signal channel_all_done_r    : std_logic;
  signal token_update          : std_logic;

  -- Channel Status Commands
  type CS_CMDS is (CS_RESET,
                   CS_CLEAR_WAIT,
                   CS_TOKEN_UPDATE,
                   CS_SET_WAIT,
                   CS_SET_HIT,
                   CS_SET_DONE,
                   CS_NONE
                  );
  signal channel_status_cmd    : CS_CMDS;

  -- Process Calculate Trigger Window
  signal fifo_delay_time       : unsigned(11 downto 0);

  -- Process Timestamp
  signal d_data_o              : std_logic_vector(31 downto 0);
  signal d_data_clk_o          : std_logic;
  signal out_of_window_l       : std_logic;
  signal out_of_window_h       : std_logic;
  signal window_hit            : std_logic;
  signal out_of_window_error   : std_logic;
  signal ch_status_cmd_pr      : CS_CMDS;

  -- Window Status Counter
  signal out_of_window_l_ctr   : unsigned(15 downto 0);
  signal window_hit_ctr        : unsigned(15 downto 0);
  signal out_of_window_h_ctr   : unsigned(15 downto 0);
  signal out_of_window_l_ctr_r : unsigned(15 downto 0);
  signal window_hit_ctr_r      : unsigned(15 downto 0);
  signal out_of_window_h_ctr_r : unsigned(15 downto 0);
  signal validation_busy       : std_logic_vector(1 downto 0);

  -- Rate Calculations
  signal data_rate_ctr         : unsigned(27 downto 0);
  signal data_rate             : unsigned(27 downto 0);
  signal rate_timer_ctr        : unsigned(27 downto 0);

  -- Self Trigger Mode
  signal self_trigger_mode     : std_logic;

  -- Process Trigger Handler
```

```
  signal store_to_fifo         : std_logic;
  signal trigger_busy_o        : std_logic;
  signal nomore_data_o         : std_logic;
  signal wait_timer_start      : std_logic;
  signal wait_timer_start_ns   : std_logic;
  signal wait_timer_init_ns    : unsigned(19 downto 0);
  signal token_return_last     : std_logic;
  signal token_return_first    : std_logic;
  signal ch_status_cmd_tr      : CS_CMDS;
  signal wait_for_data_time_r  : std_logic_vector(19 downto 0);
  signal min_validation_time_r : std_logic_vector(19 downto 0);
  signal skip_wait_for_data    : std_logic;

  type STATES is (S_TEST_SELF_TRIGGER,
                  S_IDLE,
                  S_TRIGGER,
                  S_WAIT_DATA,
                  S_WRITE_HEADER,
                  S_PROCESS_START,
                  S_WAIT_PROCESS_END,
                  S_WRITE_TRAILER,
                  S_SET_NOMORE_DATA
                  );
  signal STATE : STATES;

  signal t_data_o              : std_logic_vector(31 downto 0);
  signal t_data_clk_o          : std_logic;
  signal busy_time_ctr         : unsigned(11 downto 0);
  signal wait_timer_reset_all  : std_logic;
  signal min_val_time_expired  : std_logic;
  signal event_counter         : unsigned(9 downto 0);
  signal out_of_window_error_ctr : unsigned(15 downto 0);

  signal readout_mode          : std_logic_vector(3 downto 0);
  signal timestamp_fpga_i      : unsigned(11 downto 0);
  signal timestamp_fpga        : unsigned(11 downto 0);
  signal timestamp_ref         : unsigned(11 downto 0);
  signal busy_time_ctr_last    : unsigned(11 downto 0);
  signal evt_buffer_clear_o    : std_logic;

  -- Timers
  signal timer_reset           : std_logic;
  signal wait_timer_done       : std_logic;
  signal wait_timer_done_ns    : std_logic;

  -- Histogram
  signal histogram_fill_o      : std_logic;
  signal histogram_bin_o       : std_logic_vector(6 downto 0);
  signal histogram_adc_o       : std_logic_vector(11 downto 0);
  signal histogram_pileup_o    : std_logic;
  signal histogram_ovfl_o      : std_logic;

  -- Data FIFO Delay
  signal data_fifo_delay_o     : unsigned(7 downto 0);

  -- Output
  signal data_clk_o            : std_logic;
  signal data_o                : std_logic_vector(31 downto 0);

  -- Slave Bus
  signal slv_data_out_o        : std_logic_vector(31 downto 0);
  signal slv_no_more_data_o    : std_logic;
```



```
  signal slv_unknown_addr_o    : std_logic;
  signal slv_ack_o             : std_logic;

  signal readout_mode_r        : std_logic_vector(3 downto 0);
  signal out_of_window_error_ctr_clear : std_logic;

  -- Timestamp Trigger Window Settings
  signal nxyter_cv_time        : unsigned(11 downto 0);
  signal cts_trigger_delay     : unsigned(11 downto 0);
  signal ts_window_offset      : signed(11 downto 0);
  signal ts_window_width       : unsigned(9 downto 0);
  signal readout_time_max      : unsigned(11 downto 0);
  signal fpga_timestamp_offset : unsigned(11 downto 0);

  signal      state_d          : std_logic_vector(1 downto 0);

begin

  -- Debug Line
  DEBUG_OUT(0)            <= CLK_IN;
  DEBUG_OUT(1)            <= TRIGGER_IN;
  DEBUG_OUT(2)            <= trigger_busy_o;
  DEBUG_OUT(3)            <= DATA_CLK_IN;
  DEBUG_OUT(4)            <= out_of_window_l;
  DEBUG_OUT(5)            <= out_of_window_h;
  DEBUG_OUT(6)            <= NX_TOKEN_RETURN_IN;
  DEBUG_OUT(7)            <= NX_NOMORE_DATA_IN;
  DEBUG_OUT(8)            <= channel_all_done;
  DEBUG_OUT(9)            <= store_to_fifo;
  DEBUG_OUT(10)           <= data_clk_o;
  DEBUG_OUT(11)           <= out_of_window_error or EVT_BUFFER_FULL_IN;
  DEBUG_OUT(12)           <= token_update; --TRIGGER_BUSY_IN; --wait_timer_done;
  DEBUG_OUT(13)           <= min_val_time_expired;
  DEBUG_OUT(14)           <= token_update;
  DEBUG_OUT(15)           <= nomore_data_o;

  -- Timer
  timer_1: timer
    generic map(
      CTR_WIDTH => 12
      )
    port map (
      CLK_IN         => CLK_IN,
      RESET_IN       => timer_reset,
      TIMER_START_IN => wait_timer_start,
      TIMER_END_IN   => readout_time_max,
      TIMER_DONE_OUT => wait_timer_done
      );

  timer_2: timer
    generic map(
      CTR_WIDTH => 20,
      STEP_SIZE => 10
      )
    port map (
      CLK_IN         => CLK_IN,
      RESET_IN       => timer_reset,
      TIMER_START_IN => wait_timer_start_ns,
      TIMER_END_IN   => wait_timer_init_ns,
      TIMER_DONE_OUT => wait_timer_done_ns
      );
```

```
timer_reset <= RESET_IN or wait_timer_reset_all;

------------------------------------------------------------------------------
-- Filter only valid events
------------------------------------------------------------------------------

PROC_FILTER_TIMESTAMPS: process (CLK_IN)
  variable ts_window_offset_unsigned : unsigned(11 downto 0);
  variable window_lower_thr          : unsigned(11 downto 0);
  variable window_upper_thr          : unsigned(11 downto 0);
  variable ts_window_check_value     : unsigned(11 downto 0);
  variable deltaTStore               : unsigned(13 downto 0);
  variable store_data                : std_logic;
begin
  if( rising_edge(CLK_IN) ) then
    if (RESET_IN = '1') then
      d_data_o               <= (others => '0');
      d_data_clk_o           <= '0';
      out_of_window_l        <= '0';
      out_of_window_h        <= '0';
      window_hit             <= '0';
      out_of_window_error    <= '0';
      fifo_delay_time        <= (others => '0');
      out_of_window_error_ctr <= (others => '0');
    else
      d_data_o               <= (others => '0');
      d_data_clk_o           <= '0';
      out_of_window_l        <= '0';
      out_of_window_h        <= '0';
      window_hit             <= '0';
      out_of_window_error    <= '0';
      fifo_delay_time        <= (others => '0');
      ch_status_cmd_pr       <= CS_NONE;

      histogram_fill_o       <= '0';
      histogram_bin_o        <= (others => '0');
      histogram_adc_o        <= (others => '0');
      histogram_pileup_o     <= '0';
      histogram_ovfl_o       <= '0';

      ------------------------------------------------------------------------
      -- Calculate Thresholds and values for FIFO Delay
      ------------------------------------------------------------------------

      if (ts_window_offset(11) = '1') then
        -- Offset is negative
        ts_window_offset_unsigned :=
          (unsigned(ts_window_offset) xor x"fff") + 1;
        window_lower_thr       :=
          cts_trigger_delay + ts_window_offset_unsigned;
      else
        -- Offset is positive
        window_lower_thr       :=
          cts_trigger_delay - unsigned(ts_window_offset);
      end if;

      -- Calculate FIFO Delay
      if (window_lower_thr(11) = '0') then
        fifo_delay_time        <= window_lower_thr;  -- unit is 4ns
      else
        fifo_delay_time        <= (others => '0');
      end if;
```



```
      -- Final lower Threshold value relative to TS Reference TS
      window_lower_thr         := timestamp_fpga - window_lower_thr;

      window_upper_thr         :=
        window_lower_thr + resize(ts_window_width, 12);
      ts_window_check_value    :=
        unsigned(TIMESTAMP_IN(13 downto 2)) - window_lower_thr;

      -- Timestamp to be stored
      deltaTStore(13 downto 2) := ts_window_check_value;
      deltaTStore( 1 downto 0) := unsigned(TIMESTAMP_IN(1 downto 0));

      ------------------------------------------------------------------------
      -- Validate incoming Data
      ------------------------------------------------------------------------
      if (DATA_CLK_IN = '1') then

        if (store_to_fifo = '1' and EVT_BUFFER_FULL_IN = '0') then
          store_data                       := '0';

          -- TS Window Check
          if (ts_window_check_value(11) = '1') then
            -- TS below Window: Set WAIT Bit in LUT and discard Data
            channel_index                  <= CHANNEL_IN;
            ch_status_cmd_pr               <= CS_SET_WAIT;
            out_of_window_l                <= '1';
            store_data                     := '0';
          elsif (ts_window_check_value > ts_window_width) then
            -- TS above Window: Set DONE Bit in LUT and discard Data
            channel_index                  <= CHANNEL_IN;
            ch_status_cmd_pr               <= CS_SET_DONE;
            out_of_window_h                <= '1';
            store_data                     := '0';
          elsif ((ts_window_check_value >= 0) and
                 (ts_window_check_value <= ts_window_width)) then
            -- TS in between Window: Set WAIT Bit in LUT and Take Data
            channel_index                  <= CHANNEL_IN;
            ch_status_cmd_pr               <= CS_SET_HIT;
            window_hit                     <= '1';
            store_data                     := '1';
          else
            -- TS Window Error condition, do nothing
            out_of_window_error            <= '1';
            store_data                     := '0';
            if (out_of_window_error_ctr_clear = '0') then
              out_of_window_error_ctr      <= out_of_window_error_ctr + 1;
            end if;
          end if;

          --TS Window Disabled, always store data
          if (readout_mode(2)   = '1' or
              self_trigger_mode = '1') then
            store_data                     := '1';
          end if;

          if (store_data = '1') then

            case readout_mode(1 downto 0) is

              when "00" =>
                -- Default Mode
```

```
                if (TIMESTAMP_STATUS_IN(S_PARITY) = '0') then
                  d_data_o(10 downto  0)     <= deltaTStore(10 downto  0);
                  d_data_o(22 downto 11)     <= ADC_DATA_IN;
                  d_data_o(23)               <= TIMESTAMP_STATUS_IN(S_OVFL);
                  d_data_o(24)               <= TIMESTAMP_STATUS_IN(S_PILEUP);
                  d_data_o(31 downto 25)     <= CHANNEL_IN;
                  d_data_clk_o               <= '1';
                end if;

              when "01" =>
                -- Extended Timestamp Mode 12Bit
                if (TIMESTAMP_STATUS_IN(S_PARITY) = '0') then
                  d_data_o(11 downto  0)     <= deltaTStore(11 downto  0);
                  d_data_o(22 downto 12)     <= ADC_DATA_IN(11 downto 1);
                  d_data_o(23)               <= TIMESTAMP_STATUS_IN(S_OVFL);
                  d_data_o(24)               <= TIMESTAMP_STATUS_IN(S_PILEUP);
                  d_data_o(31 downto 25)     <= CHANNEL_IN;
                  d_data_clk_o               <= '1';
                end if;

              when "10" =>
                -- Extended Timestamp Mode 14Bit
                if (TIMESTAMP_STATUS_IN(S_PARITY) = '0') then
                  d_data_o(13 downto  0)     <= deltaTStore;
                  d_data_o(22 downto 14)     <= ADC_DATA_IN(11 downto 3);
                  d_data_o(23)               <= TIMESTAMP_STATUS_IN(S_OVFL);
                  d_data_o(24)               <= TIMESTAMP_STATUS_IN(S_PILEUP);
                  d_data_o(31 downto 25)     <= CHANNEL_IN;
                  d_data_clk_o               <= '1';
                end if;

              when "11" =>
                if (TIMESTAMP_STATUS_IN(S_PARITY) = '0') then
                  d_data_o(13 downto  0)     <= deltaTStore;
                  d_data_o(24 downto 14)     <= ADC_DATA_IN(11 downto 1);
                  d_data_o(31 downto 25)     <= CHANNEL_IN;
                  d_data_clk_o               <= '1';
                end if;

            end case;
          end if;

          if (out_of_window_error_ctr_clear = '1') then
            out_of_window_error_ctr          <= (others => '0');
          end if;
        end if;

        -- Fill Histogram
        histogram_fill_o                  <= '1';
        histogram_bin_o                   <= CHANNEL_IN;
        histogram_adc_o                   <= ADC_DATA_IN;
        histogram_pileup_o                <= TIMESTAMP_STATUS_IN(S_PILEUP);
        histogram_ovfl_o                  <= TIMESTAMP_STATUS_IN(S_OVFL);

      end if;
    end if;
  end if;
end process PROC_FILTER_TIMESTAMPS;

PROC_WINDOW_STATE_CTR: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
```



```
    if (RESET_IN = '1') then
      out_of_window_l_ctr     <= (others => '0');
      window_hit_ctr          <= (others => '0');
      out_of_window_h_ctr     <= (others => '0');
      out_of_window_l_ctr_r   <= (others => '0');
      window_hit_ctr_r        <= (others => '0');
      out_of_window_h_ctr_r   <= (others => '0');
      validation_busy         <= (others => '0');
    else
      validation_busy(0)           <= store_to_fifo;
      validation_busy(1)           <= validation_busy(0);

      case validation_busy is
        when "00"=>                   -- No validation
          out_of_window_l_ctr      <= (others => '0');
          window_hit_ctr           <= (others => '0');
          out_of_window_l_ctr      <= (others => '0');

        when "01"=>                   -- Start validation
          out_of_window_l_ctr      <= (others => '0');
          window_hit_ctr           <= (others => '0');
          out_of_window_l_ctr      <= (others => '0');

        when "10"=>                   -- End validation
          out_of_window_l_ctr_r    <= out_of_window_l_ctr;
          window_hit_ctr_r         <= window_hit_ctr;
          out_of_window_l_ctr_r    <= out_of_window_l_ctr;

        when "11" =>                  -- Validation
          if (out_of_window_l = '1') then
            out_of_window_l_ctr    <= out_of_window_l_ctr + 1;
          end if;

          if (window_hit = '1') then
            window_hit_ctr         <= window_hit_ctr + 1;
          end if;

          if (out_of_window_l = '1') then
            out_of_window_h_ctr     <= out_of_window_h_ctr + 1;
          end if;

      end case;
    end if;
  end if;
end process PROC_WINDOW_STATE_CTR;

PROC_RATE_COUNTER: process(CLK_IN)
begin
  if (rising_edge(CLK_IN) ) then
    if (RESET_IN = '1') then
      data_rate_ctr          <= (others => '0');
      data_rate              <= (others => '0');
      rate_timer_ctr         <= (others => '0');
    else
      if (rate_timer_ctr < x"5f5e100") then
        rate_timer_ctr             <= rate_timer_ctr + 1;

        if (d_data_clk_o = '1') then
          data_rate_ctr            <= data_rate_ctr + 1;
        end if;
      else
        rate_timer_ctr             <= (others => '0');
```

```
        data_rate                   <= data_rate_ctr;

        data_rate_ctr(27 downto 0) <= (others => '0');
        data_rate_ctr(0)           <= d_data_clk_o;
      end if;
    end if;
  end if;
end process PROC_RATE_COUNTER;

-------------------------------------------------------------------------------
-- Trigger Handler
-------------------------------------------------------------------------------

-- Set Self Trigger Mode Toggle Handler
PROC_SELF_TRIGGER: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if (RESET_IN = '1') then
      self_trigger_mode    <= '0';
    else
      if (trigger_busy_o = '0') then
        if (readout_mode_r(3) = '1') then
          self_trigger_mode  <= '1';
        else
          self_trigger_mode  <= '0';
        end if;
      end if;
    end if;
  end if;
end process PROC_SELF_TRIGGER;

PROC_TRIGGER_HANDLER: process(CLK_IN)
  variable wait_for_data_time    : unsigned(19 downto 0);
  variable min_validation_time   : unsigned(19 downto 0);
begin
  if( rising_edge(CLK_IN) ) then
    timestamp_fpga_i              <= TIMESTAMP_FPGA_IN;
    if (RESET_IN = '1' or FAST_CLEAR_IN = '1') then
      store_to_fifo               <= '0';
      trigger_busy_o              <= '0';
      nomore_data_o               <= '0';
      wait_timer_start            <= '0';
      wait_timer_start_ns         <= '0';
      wait_timer_reset_all        <= '0';
      min_val_time_expired        <= '0';
      t_data_o                    <= (others => '0');
      t_data_clk_o                <= '0';
      busy_time_ctr               <= (others => '0');
      busy_time_ctr_last          <= (others => '0');
      token_return_last           <= '0';
      token_return_first          <= '0';
      ch_status_cmd_tr            <= CS_RESET;
      event_counter               <= (others => '0');
      readout_mode                <= (others => '0');
      timestamp_fpga              <= (others => '0');
      timestamp_ref               <= (others => '0');
      evt_buffer_clear_o          <= '0';
      wait_for_data_time_r        <= (others => '0');
      min_validation_time_r       <= (others => '0');
      STATE                       <= S_TEST_SELF_TRIGGER;
    else
      store_to_fifo               <= '0';
```



```
      wait_timer_start            <= '0';
      wait_timer_start_ns         <= '0';
      wait_timer_reset_all        <= '0';
      trigger_busy_o              <= '1';
      nomore_data_o               <= '0';
      t_data_o                    <= (others => '0');
      t_data_clk_o                <= '0';
      ch_status_cmd_tr            <= CS_NONE;
      evt_buffer_clear_o          <= '0';

      -- Wait for Data and minimum Validation Time calculation
      min_validation_time         := resize(ts_window_width * 4, 20);
      wait_for_data_time          :=
        resize(nxyter_cv_time, 20) + data_fifo_delay_o * 32 + 320;

      if (skip_wait_for_data = '1') then
        min_validation_time       :=
          min_validation_time + wait_for_data_time;
        wait_for_data_time        := x"00001";
      end if;
      min_validation_time_r       <= min_validation_time;
      wait_for_data_time_r        <= wait_for_data_time;

      -- Check Token Return
      token_return_last           <= NX_TOKEN_RETURN_IN;
      if (store_to_fifo      = '1' and  -- min_val_time handled by TK-UPDATE
          NX_TOKEN_RETURN_IN = '1' and
          token_return_last  = '0') then
        if (min_val_time_expired = '1') then
          if (token_return_first = '1') then
            ch_status_cmd_tr        <= CS_TOKEN_UPDATE;
          else
            token_return_first      <= '1';
            ch_status_cmd_tr        <= CS_CLEAR_WAIT;
          end if;
        else
          ch_status_cmd_tr          <= CS_CLEAR_WAIT;
        end if;
      end if;

      case STATE is

        when S_TEST_SELF_TRIGGER =>
          state_d <= "00";

          if (self_trigger_mode = '1') then
            -- Wait End of LVL2 Trigger Cycle
            if (TRIGGER_BUSY_IN = '1') then
              STATE                 <= S_TEST_SELF_TRIGGER;
            else
              readout_mode          <= readout_mode_r;
              timestamp_ref         <= (others => '0');
              STATE                 <= S_WRITE_HEADER;
            end if;
          else
            wait_timer_reset_all    <= '1';
            min_val_time_expired    <= '0';
            STATE                   <= S_IDLE;
          end if;

        when S_IDLE =>
          state_d <= "01";
```

```
          if (TRIGGER_IN = '1') then
            busy_time_ctr               <= (others => '0');
            STATE                       <= S_TRIGGER;
          else
            trigger_busy_o              <= '0';
            min_val_time_expired        <= '0';
            if (self_trigger_mode = '1') then
              ch_status_cmd_tr          <= CS_RESET;
              store_to_fifo             <= '1';
            end if;
            STATE                       <= S_IDLE;
          end if;

        when S_TRIGGER =>
          if (self_trigger_mode = '0') then
            readout_mode                <= readout_mode_r;

            -- wait for data arrival and clear evt buffer
            wait_timer_start_ns         <= '1';
            wait_timer_init_ns          <= wait_for_data_time;
            evt_buffer_clear_o          <= '1';
            STATE                       <= S_WAIT_DATA;
          else
            STATE                       <= S_WRITE_TRAILER;
          end if;

        when S_WAIT_DATA =>
          if (wait_timer_done_ns = '0') then
            STATE                       <= S_WAIT_DATA;
          else
            timestamp_fpga              <=
              timestamp_fpga_i + fpga_timestamp_offset;
            timestamp_ref               <= timestamp_fpga;
            STATE                       <= S_WRITE_HEADER;
          end if;

        when S_WRITE_HEADER =>
          state_d                       <= "10";

          t_data_o(11 downto 0)         <= timestamp_ref;
          t_data_o(21 downto 12)        <= event_counter;
          -- Readout Mode Mapping
          -- Bit #3: self Triger mode
          -- Bit #2: 0: activate TS Selection Window
          --         1: disable TS Selection Window, i.e.
          --            data will be written to disk as long as
          --            Readout Time Max (Reg.: 0x8184) is valid
          --
          -- Bit #1..0: 00: Standard
          --            01: UNDEF
          --            10: UNDEF
          --            11: UNDEF
          t_data_o(25 downto 22)        <= readout_mode;
          t_data_o(29 downto 26)        <= VERSION_NUMBER;
          t_data_o(31 downto 30)        <= BOARD_ID;
          t_data_clk_o                  <= '1';

          event_counter                 <= event_counter + 1;
          if (self_trigger_mode = '0') then
            STATE                       <= S_PROCESS_START;
          else
```



```
            STATE                       <= S_IDLE;
          end if;

        when S_PROCESS_START =>
          wait_timer_start              <= '1';
          wait_timer_start_ns           <= '1';
          wait_timer_init_ns            <= min_validation_time;
          token_return_first            <= '0';
          ch_status_cmd_tr              <= CS_RESET;
          store_to_fifo                 <= '1';
          STATE                         <= S_WAIT_PROCESS_END;

        when S_WAIT_PROCESS_END =>
          -- Check minimum validation time
          if (wait_timer_done_ns = '1') then
            min_val_time_expired        <= '1';
          end if;

          -- Always Exit in case of maximum validation time has expired
          if (wait_timer_done     = '1') then
            wait_timer_reset_all        <= '1';
            STATE                       <= S_WRITE_TRAILER;
          elsif (readout_mode(2)      = '0' and
                 min_val_time_expired = '1' and
                 (channel_all_done    = '1' or
                  NX_NOMORE_DATA_IN   = '1')
                 ) then
            wait_timer_reset_all        <= '1';
            STATE                       <= S_WRITE_TRAILER;
          else
            -- Continue Validation
            store_to_fifo               <= '1';
            STATE                       <= S_WAIT_PROCESS_END;
          end if;

          when S_WRITE_TRAILER =>
            state_d                     <= "11";
            t_data_o                    <= (others => '1');
            t_data_clk_o                <= '1';
            STATE                       <= S_SET_NOMORE_DATA;

          when S_SET_NOMORE_DATA =>
            nomore_data_o               <= '1';
            busy_time_ctr_last          <= busy_time_ctr;
            STATE                       <= S_TEST_SELF_TRIGGER;

        end case;

        if (STATE /= S_IDLE) then
          busy_time_ctr                 <= busy_time_ctr + 1;
        end if;

      end if;
    end if;
  end process PROC_TRIGGER_HANDLER;

  ------------------------------------------------------------------------------
  -- Channel Status Handler
  ------------------------------------------------------------------------------

  PROC_CHANNEL_STATUS_CMD: process(ch_status_cmd_tr,
                                   ch_status_cmd_pr)
```

```
  begin
    if (ch_status_cmd_tr /= CS_NONE) then
      channel_status_cmd   <= ch_status_cmd_tr;
    elsif (ch_status_cmd_pr /= CS_NONE) then
      channel_status_cmd   <= ch_status_cmd_pr;
    else
      channel_status_cmd   <= CS_NONE;
    end if;
  end process PROC_CHANNEL_STATUS_CMD;

  PROC_CHANNEL_STATUS: process(CLK_IN)
    constant all_one : std_logic_vector(127 downto 0) := (others => '1');

  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1') then
        channel_wait          <= (others => '0');
        channel_done          <= (others => '0');
        channel_hit           <= (others => '0');
        channel_done_r        <= (others => '0');
        channel_wait_r        <= (others => '0');
        channel_hit_r         <= (others => '0');
        channel_all_done      <= '0';
        channel_all_done_r    <= '0';
        token_update          <= '0';
      else
        token_update          <= '0';
        -- Check done status
        if (channel_status_cmd /= CS_RESET ) then
          if (channel_done = all_one) then
            channel_all_done  <= '1';
          end if;
        else
          channel_all_done    <= '0';
          channel_all_done_r  <= channel_all_done;
        end if;

        -- Process Command
        case channel_status_cmd is

          when CS_RESET =>
            channel_wait      <= (others => '0');
            channel_done      <= (others => '0');
            channel_hit       <= (others => '0');
            channel_done_r    <= channel_done;
            channel_hit_r     <= channel_hit;
            channel_wait_r    <= channel_wait;

          when CS_CLEAR_WAIT =>
            channel_wait    <= (others => '0');

          when CS_TOKEN_UPDATE =>
            channel_done    <= channel_done or (not channel_wait);
            token_update    <= '1';
            channel_wait    <= (others => '0');

          when CS_SET_WAIT =>
            channel_wait(to_integer(unsigned(channel_index))) <= '1';

          when CS_SET_HIT =>
            channel_hit(to_integer(unsigned(channel_index)))  <= '1';
            channel_wait(to_integer(unsigned(channel_index))) <= '1';
```



```
          when CS_SET_DONE =>
            channel_done(to_integer(unsigned(channel_index))) <= '1';

          when CS_NONE => null;

        end case;
      end if;
    end if;
  end process PROC_CHANNEL_STATUS;

  PROC_DATA_FIFO_DELAY: process(CLK_IN)
    variable nx_cvt     : unsigned(11 downto 0);
    variable fifo_delay : unsigned(11 downto 0);
  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1') then
        data_fifo_delay_o       <= x"01";
      else
        -- nxyter delay assumed to be 400ns
         nx_cvt                 := nxyter_cv_time / 4;
        if (fifo_delay_time > nx_cvt and fifo_delay_time < 1000) then
          fifo_delay            := (fifo_delay_time - nx_cvt) / 8;
          data_fifo_delay_o     <= fifo_delay(7 downto 0);
        else
          data_fifo_delay_o     <= x"01";
        end if;
      end if;
    end if;
  end process PROC_DATA_FIFO_DELAY;

  ------------------------------------------------------------------------------
  -- TRBNet Slave Bus
  ------------------------------------------------------------------------------

  -- Give status info to the TRB Slow Control Channel
  PROC_SLAVE_BUS: process(CLK_IN)

  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1' ) then
        slv_data_out_o                <= (others => '0');
        slv_ack_o                     <= '0';
        slv_unknown_addr_o            <= '0';
        slv_no_more_data_o            <= '0';

        ts_window_offset              <= (others => '0');
        ts_window_width               <= "0000110010";  -- 50
        cts_trigger_delay             <= x"0c8";
        readout_mode_r                <= "0000";
        readout_time_max              <= x"3e8";
        fpga_timestamp_offset         <= (others => '0');
        out_of_window_error_ctr_clear <= '0';
        skip_wait_for_data            <= '0';
        nxyter_cv_time                <= x"190";  -- 400ns
      else
        slv_data_out_o                <= (others => '0');
        slv_unknown_addr_o            <= '0';
        slv_no_more_data_o            <= '0';

        cts_trigger_delay(11 downto 10)  <= (others => '0');
        readout_time_max(11 downto 10)   <= (others => '0');
```

```
      out_of_window_error_ctr_clear    <= '0';

      if (SLV_READ_IN  = '1') then
        case SLV_ADDR_IN is
          when x"0000" =>
            slv_data_out_o( 3 downto  0)    <= readout_mode_r;
            slv_data_out_o(31 downto  5)    <= (others => '0');
            slv_ack_o                       <= '1';

          when x"0001" =>
            slv_data_out_o(11 downto  0)    <=
              std_logic_vector(ts_window_offset(11 downto 0));
            --if (ts_window_offset(11) = '1') then
            --  slv_data_out_o(31 downto 12)  <= (others => '1');
            --else
              slv_data_out_o(31 downto 12)  <= (others => '0');
            --end if;
            slv_ack_o                       <= '1';

          when x"0002" =>
            slv_data_out_o(9 downto  0)     <=
              std_logic_vector(ts_window_width);
            slv_data_out_o(31 downto 10)    <= (others => '0');
            slv_ack_o                       <= '1';

          when x"0003" =>
            slv_data_out_o(9 downto  0)     <=
              std_logic_vector(cts_trigger_delay(9 downto 0));
            slv_data_out_o(31 downto 10)    <= (others => '0');
            slv_ack_o                       <= '1';

          when x"0004" =>
            slv_data_out_o(9 downto  0)     <=
              std_logic_vector(readout_time_max(9 downto 0));
            slv_data_out_o(31 downto 10)    <= (others => '0');
            slv_ack_o                       <= '1';

          when x"0005" =>
            slv_data_out_o(11 downto  0)    <=
              std_logic_vector(fpga_timestamp_offset);
            slv_data_out_o(31 downto 12)    <= (others => '0');
            slv_ack_o                       <= '1';

          when x"0006" =>
            slv_data_out_o(11 downto  0)    <=
              std_logic_vector(busy_time_ctr_last);
            slv_data_out_o(31 downto 12)    <= (others => '0');
            slv_ack_o                       <= '1';

          when x"0007" =>
            slv_data_out_o(11 downto  0)    <= timestamp_ref;
            slv_data_out_o(31 downto 12)    <= (others => '0');
            slv_ack_o                       <= '1';

          when x"0008" =>
            slv_data_out_o(11 downto  0)    <= fifo_delay_time;
            slv_data_out_o(31 downto 12)    <= (others => '0');
            slv_ack_o                       <= '1';

          when x"0009" =>
            slv_data_out_o(15 downto  0)    <= out_of_window_error_ctr;
            slv_data_out_o(31 downto 16)    <= (others => '0');
```



```
            slv_ack_o                       <= '1';

          when x"000a" =>
            slv_data_out_o(7 downto  0)     <=
              std_logic_vector(data_fifo_delay_o);
            slv_data_out_o(31 downto  8)    <= (others => '0');
            slv_ack_o                       <= '1';

          -- 4x Channel WAIT

          when x"000b" =>
            slv_data_out_o                  <=
              std_logic_vector(channel_wait_r(31 downto 0));
            slv_ack_o                       <= '1';

          when x"000c" =>
            slv_data_out_o                  <=
              std_logic_vector(channel_wait_r(63 downto 32));
            slv_ack_o                       <= '1';

          when x"000d" =>
            slv_data_out_o                  <=
              std_logic_vector(channel_wait_r(95 downto 64));
            slv_ack_o                       <= '1';

          when x"000e" =>
            slv_data_out_o                  <=
              std_logic_vector(channel_wait_r(127 downto 96));
            slv_ack_o                       <= '1';

            -- 4x Channel HIT

          when x"000f" =>
            slv_data_out_o                  <=
              std_logic_vector(channel_hit_r(31 downto 0));
            slv_ack_o                       <= '1';

          when x"0010" =>
            slv_data_out_o                  <=
              std_logic_vector(channel_hit_r(63 downto 32));
            slv_ack_o                       <= '1';

          when x"0011" =>
            slv_data_out_o                  <=
              std_logic_vector(channel_hit_r(95 downto 64));
            slv_ack_o                       <= '1';

          when x"0012" =>
            slv_data_out_o                  <=
              std_logic_vector(channel_hit_r(127 downto 96));
            slv_ack_o                       <= '1';

            -- 4x Channel DONE

          when x"0013" =>
            slv_data_out_o                  <=
              std_logic_vector(channel_done_r(31 downto 0));
            slv_ack_o                       <= '1';

          when x"0014" =>
            slv_data_out_o                  <=
              std_logic_vector(channel_done_r(63 downto 32));
```

```
          slv_ack_o                       <= '1';

        when x"0015" =>
          slv_data_out_o                  <=
            std_logic_vector(channel_done_r(95 downto 64));
          slv_ack_o                       <= '1';

        when x"0016" =>
          slv_data_out_o                  <=
            std_logic_vector(channel_done_r(127 downto 96));
          slv_ack_o                       <= '1';

        when x"0017" =>
          slv_data_out_o(0)               <= channel_all_done_r;
          slv_data_out_o(31 downto  1)    <= (others => '0');
          slv_ack_o                       <= '1';

        when x"0018" =>
          slv_data_out_o(0)               <= EVT_BUFFER_FULL_IN;
          slv_data_out_o(31 downto  1)    <= (others => '0');
          slv_ack_o                       <= '1';

        when x"0019" =>
          slv_data_out_o(19 downto 0)     <= wait_for_data_time_r;
          slv_data_out_o(30 downto  20)   <= (others => '0');
          slv_data_out_o(31)              <= skip_wait_for_data;
          slv_ack_o                       <= '1';

        when x"001a" =>
          slv_data_out_o(11 downto 0)     <=
            std_logic_vector(nxyter_cv_time);
          slv_data_out_o(31 downto  12)   <= (others => '0');
          slv_ack_o                       <= '1';

        when x"001b" =>
          slv_data_out_o(19 downto 0)     <=
            std_logic_vector(min_validation_time_r);
          slv_data_out_o(31 downto 20)    <= (others => '0');
          slv_ack_o                       <= '1';

        when x"001c" =>
          slv_data_out_o(15 downto 0)     <=
            std_logic_vector(out_of_window_l_ctr_r);
          slv_data_out_o(31 downto 16)    <= (others => '0');
          slv_ack_o                       <= '1';

        when x"001d" =>
          slv_data_out_o(15 downto 0)     <=
            std_logic_vector(window_hit_ctr_r);
          slv_data_out_o(31 downto 16)    <= (others => '0');
          slv_ack_o                       <= '1';

        when x"001e" =>
          slv_data_out_o(15 downto 0)     <=
            std_logic_vector(out_of_window_h_ctr_r);
          slv_data_out_o(31 downto 16)    <= (others => '0');
          slv_ack_o                       <= '1';

        when x"001f" =>
          slv_data_out_o(27 downto 0)     <= std_logic_vector(data_rate);
          slv_data_out_o(31 downto 28)    <= (others => '0');
          slv_ack_o                       <= '1';
```



```
          when others  =>
            slv_unknown_addr_o            <= '1';
            slv_ack_o                     <= '0';

        end case;

      elsif (SLV_WRITE_IN  = '1') then
        case SLV_ADDR_IN is
          when x"0000" =>
            readout_mode_r                <= SLV_DATA_IN(3 downto 0);
            slv_ack_o                     <= '1';

          when x"0001" =>
            if ((signed(SLV_DATA_IN(11 downto 0)) > -2048) and
                (signed(SLV_DATA_IN(11 downto 0)) <  2048)) then
              ts_window_offset(11 downto 0) <=
                signed(SLV_DATA_IN(11 downto 0));
            end if;
            slv_ack_o                     <= '1';

          when x"0002" =>
            ts_window_width               <=
              unsigned(SLV_DATA_IN(9 downto 0));
            slv_ack_o                     <= '1';

          when x"0003" =>
            cts_trigger_delay(9 downto 0)   <=
              unsigned(SLV_DATA_IN(9 downto 0));
            slv_ack_o                     <= '1';

          when x"0004" =>
            if (unsigned(SLV_DATA_IN(9 downto 0)) >= 1) then
              readout_time_max(9 downto 0)  <=
                unsigned(SLV_DATA_IN(9 downto 0));
            end if;
            slv_ack_o                     <= '1';

          when x"0005" =>
            fpga_timestamp_offset(11 downto 0) <=
              unsigned(SLV_DATA_IN(11 downto 0));
            slv_ack_o                     <= '1';

          when x"0009" =>
            out_of_window_error_ctr_clear   <= '1';
            slv_ack_o                     <= '1';

          when x"0019" =>
            skip_wait_for_data            <= SLV_DATA_IN(31);
            slv_ack_o                     <= '1';

          when x"001a" =>
            nxyter_cv_time                <=
              unsigned(SLV_DATA_IN(11 downto 0));
            slv_ack_o                     <= '1';

          when others  =>
            slv_unknown_addr_o            <= '1';
            slv_ack_o                     <= '0';
        end case;
      else
        slv_ack_o                         <= '0';
```

```
        end if;
      end if;
    end if;
  end process PROC_SLAVE_BUS;

  ------------------------------------------------------------------------------
  -- Output Signals
  ------------------------------------------------------------------------------

  data_clk_o <= d_data_clk_o or t_data_clk_o;
  data_o     <= d_data_o or t_data_o;

  TRIGGER_BUSY_OUT       <= trigger_busy_o;
  DATA_OUT               <= data_o or t_data_o;
  DATA_CLK_OUT           <= data_clk_o;
  NOMORE_DATA_OUT        <= nomore_data_o;
  DATA_FIFO_DELAY_OUT    <= std_logic_vector(data_fifo_delay_o);
  EVT_BUFFER_CLEAR_OUT   <= evt_buffer_clear_o;

  HISTOGRAM_FILL_OUT     <= histogram_fill_o;
  HISTOGRAM_BIN_OUT      <= histogram_bin_o;
  HISTOGRAM_ADC_OUT      <= histogram_adc_o;
  HISTOGRAM_PILEUP_OUT   <= histogram_pileup_o;
  HISTOGRAM_OVERFLOW_OUT <= histogram_ovfl_o;

  -- Slave
  SLV_DATA_OUT           <= slv_data_out_o;
  SLV_NO_MORE_DATA_OUT   <= slv_no_more_data_o;
  SLV_UNKNOWN_ADDR_OUT   <= slv_unknown_addr_o;
  SLV_ACK_OUT            <= slv_ack_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package nxyter_components is

------------------------------------------------------------------------------
-- TRBNet interfaces
------------------------------------------------------------------------------

  component nXyter_FEE_board
    generic (
      BOARD_ID : std_logic_vector(1 downto 0));
    port (
      CLK_IN                    : in    std_logic;
      RESET_IN                  : in    std_logic;
      CLK_NX_MAIN_IN            : in    std_logic;
      CLK_ADC_IN                : in    std_logic;
      PLL_NX_CLK_LOCK_IN        : in    std_logic;
      PLL_ADC_DCLK_LOCK_IN      : in    std_logic;
      NX_DATA_CLK_TEST_IN       : in    std_logic;
      TRIGGER_OUT               : out   std_logic;
      I2C_SDA_INOUT             : inout std_logic;
      I2C_SCL_INOUT             : inout std_logic;
      I2C_SM_RESET_OUT          : out   std_logic;
      I2C_REG_RESET_OUT         : out   std_logic;
      SPI_SCLK_OUT              : out   std_logic;
      SPI_SDIO_INOUT            : inout std_logic;
      SPI_CSB_OUT               : out   std_logic;
      NX_DATA_CLK_IN            : in    std_logic;
```



```
      NX_TIMESTAMP_IN           : in    std_logic_vector (7 downto 0);
      NX_RESET_OUT              : out   std_logic;
      NX_TESTPULSE_OUT          : out   std_logic;
      NX_TIMESTAMP_TRIGGER_OUT  : out   std_logic;
      ADC_FCLK_IN               : in    std_logic_vector(1 downto 0);
      ADC_DCLK_IN               : in    std_logic_vector(1 downto 0);
      ADC_SAMPLE_CLK_OUT        : out   std_logic;
      ADC_A_IN                  : in    std_logic_vector(1 downto 0);
      ADC_B_IN                  : in    std_logic_vector(1 downto 0);
      ADC_NX_IN                 : in    std_logic_vector(1 downto 0);
      ADC_D_IN                  : in    std_logic_vector(1 downto 0);
      TIMING_TRIGGER_IN         : in    std_logic;
      LVL1_TRG_DATA_VALID_IN    : in    std_logic;
      LVL1_VALID_TIMING_TRG_IN  : in    std_logic;
      LVL1_VALID_NOTIMING_TRG_IN: in    std_logic;
      LVL1_INVALID_TRG_IN       : in    std_logic;
      LVL1_TRG_TYPE_IN          : in    std_logic_vector(3 downto 0);
      LVL1_TRG_NUMBER_IN        : in    std_logic_vector(15 downto 0);
      LVL1_TRG_CODE_IN          : in    std_logic_vector(7 downto 0);
      LVL1_TRG_INFORMATION_IN   : in    std_logic_vector(23 downto 0);
      LVL1_INT_TRG_NUMBER_IN    : in    std_logic_vector(15 downto 0);
      FEE_TRG_RELEASE_OUT       : out   std_logic;
      FEE_TRG_STATUSBITS_OUT    : out   std_logic_vector(31 downto 0);
      FEE_DATA_OUT              : out   std_logic_vector(31 downto 0);
      FEE_DATA_WRITE_OUT        : out   std_logic;
      FEE_DATA_FINISHED_OUT     : out   std_logic;
      FEE_DATA_ALMOST_FULL_IN   : in    std_logic;
      REGIO_ADDR_IN             : in    std_logic_vector(15 downto 0);
      REGIO_DATA_IN             : in    std_logic_vector(31 downto 0);
      REGIO_DATA_OUT            : out   std_logic_vector(31 downto 0);
      REGIO_READ_ENABLE_IN      : in    std_logic;
      REGIO_WRITE_ENABLE_IN     : in    std_logic;
      REGIO_TIMEOUT_IN          : in    std_logic;
      REGIO_DATAREADY_OUT       : out   std_logic;
      REGIO_WRITE_ACK_OUT       : out   std_logic;
      REGIO_NO_MORE_DATA_OUT    : out   std_logic;
      REGIO_UNKNOWN_ADDR_OUT    : out   std_logic;
      DEBUG_LINE_OUT            : out   std_logic_vector(15 downto 0)
      );
  end component;

------------------------------------------------------------------------------
-- nXyter I2C Interface
------------------------------------------------------------------------------


component nx_i2c_master
  generic (
    I2C_SPEED : unsigned(11 downto 0)
    );
  port (
    CLK_IN               : in    std_logic;
    RESET_IN             : in    std_logic;
    SDA_INOUT            : inout std_logic;
    SCL_INOUT            : inout std_logic;
    INTERNAL_COMMAND_IN  : in    std_logic_vector(31 downto 0);
    COMMAND_BUSY_OUT     : out   std_logic;
    I2C_DATA_OUT         : out   std_logic_vector(31 downto 0);
    I2C_DATA_BYTES_OUT   : out   std_logic_vector(31 downto 0);
    I2C_LOCK_IN          : in    std_logic;
    SLV_READ_IN          : in    std_logic;
    SLV_WRITE_IN         : in    std_logic;
```

```
    SLV_DATA_OUT         : out   std_logic_vector(31 downto 0);
    SLV_DATA_IN          : in    std_logic_vector(31 downto 0);
    SLV_ADDR_IN          : in    std_logic_vector(15 downto 0);
    SLV_ACK_OUT          : out   std_logic;
    SLV_NO_MORE_DATA_OUT : out   std_logic;
    SLV_UNKNOWN_ADDR_OUT : out   std_logic;
    DEBUG_OUT            : out   std_logic_vector(15 downto 0)
    );
end component;

component nx_i2c_startstop
  generic (
    I2C_SPEED : unsigned(11 downto 0)
    );
  port (
    CLK_IN            : in  std_logic;
    RESET_IN          : in  std_logic;
    START_IN          : in  std_logic;  -- Start Sequence
    SELECT_IN         : in  std_logic;  -- '1' -> Start, '0'-> Stop
    SEQUENCE_DONE_OUT : out std_logic;
    SDA_OUT           : out std_logic;
    SCL_OUT           : out std_logic;
    NREADY_OUT        : out std_logic
    );
end component;

component nx_i2c_sendbyte
  generic (
    I2C_SPEED : unsigned(11 downto 0)
    );
  port (
    CLK_IN            : in  std_logic;
    RESET_IN          : in  std_logic;
    START_IN          : in  std_logic;
    BYTE_IN           : in  std_logic_vector(7 downto 0);
    SEQUENCE_DONE_OUT : out std_logic;
    SDA_OUT           : out std_logic;
    SCL_OUT           : out std_logic;
    SDA_IN            : in  std_logic;
    SCL_IN            : in  std_logic;
    ACK_OUT           : out std_logic
    );
end component;

component nx_i2c_readbyte
  generic (
    I2C_SPEED : unsigned(11 downto 0)
    );
  port (
    CLK_IN            : in  std_logic;
    RESET_IN          : in  std_logic;
    START_IN          : in  std_logic;
    NUM_BYTES_IN      : in  unsigned(2 downto 0);
    BYTE_OUT          : out std_logic_vector(31 downto 0);
    SEQUENCE_DONE_OUT : out std_logic;
    SDA_OUT           : out std_logic;
    SCL_OUT           : out std_logic;
    SDA_IN            : in  std_logic
    );
end component;

--------------------------------------------------------------------------------
```



```
-- ADC SPI Interface
--------------------------------------------------------------------------------

component adc_spi_master
  generic (
    SPI_SPEED : unsigned(7 downto 0)
    );
  port (
    CLK_IN               : in    std_logic;
    RESET_IN             : in    std_logic;
    SCLK_OUT             : out   std_logic;
    SDIO_INOUT           : inout std_logic;
    CSB_OUT              : out   std_logic;
    INTERNAL_COMMAND_IN  : in    std_logic_vector(31 downto 0);
    COMMAND_BUSY_OUT     : out   std_logic;
    SPI_DATA_OUT         : out   std_logic_vector(31 downto 0);
    SPI_LOCK_IN          : in    std_logic;
    SLV_READ_IN          : in    std_logic;
    SLV_WRITE_IN         : in    std_logic;
    SLV_DATA_OUT         : out   std_logic_vector(31 downto 0);
    SLV_DATA_IN          : in    std_logic_vector(31 downto 0);
    SLV_ACK_OUT          : out   std_logic;
    SLV_NO_MORE_DATA_OUT : out   std_logic;
    SLV_UNKNOWN_ADDR_OUT : out   std_logic;
    DEBUG_OUT            : out   std_logic_vector(15 downto 0)
    );
end component;

component adc_spi_sendbyte
  generic (
    SPI_SPEED : unsigned(7 downto 0)
    );
  port (
    CLK_IN            : in  std_logic;
    RESET_IN          : in  std_logic;
    START_IN          : in  std_logic;
    BYTE_IN           : in  std_logic_vector(7 downto 0);
    SEQUENCE_DONE_OUT : out std_logic;
    SCLK_OUT          : out std_logic;
    SDIO_OUT          : out std_logic
    );
end component;

component adc_spi_readbyte
  generic (
    SPI_SPEED : unsigned(7 downto 0)
    );
  port (
    CLK_IN            : in  std_logic;
    RESET_IN          : in  std_logic;
    START_IN          : in  std_logic;
    BYTE_OUT          : out std_logic_vector(7 downto 0);
    SEQUENCE_DONE_OUT : out std_logic;
    SDIO_IN           : in  std_logic;
    SCLK_OUT          : out std_logic
    );
end component;

--------------------------------------------------------------------------------
-- ADC Data Handler
--------------------------------------------------------------------------------
```

```
component adc_ad9228
  port (
    CLK_IN               : in  std_logic;
    RESET_IN             : in  std_logic;
    CLK_ADCDAT_IN        : in  std_logic;

    ADC0_SCLK_IN         : in  std_logic;
    ADC0_SCLK_OUT        : out std_logic;
    ADC0_DATA_A_IN       : in  std_logic;
    ADC0_DATA_B_IN       : in  std_logic;
    ADC0_DATA_C_IN       : in  std_logic;
    ADC0_DATA_D_IN       : in  std_logic;
    ADC0_DCLK_IN         : in  std_logic;
    ADC0_FCLK_IN         : in  std_logic;

    ADC1_SCLK_IN         : in  std_logic;
    ADC1_SCLK_OUT        : out std_logic;
    ADC1_DATA_A_IN       : in  std_logic;
    ADC1_DATA_B_IN       : in  std_logic;
    ADC1_DATA_C_IN       : in  std_logic;
    ADC1_DATA_D_IN       : in  std_logic;
    ADC1_DCLK_IN         : in  std_logic;
    ADC1_FCLK_IN         : in  std_logic;

    ADC0_DATA_A_OUT      : out std_logic_vector(11 downto 0);
    ADC0_DATA_B_OUT      : out std_logic_vector(11 downto 0);
    ADC0_DATA_C_OUT      : out std_logic_vector(11 downto 0);
    ADC0_DATA_D_OUT      : out std_logic_vector(11 downto 0);
    ADC0_DATA_VALID_OUT  : out std_logic;

    ADC1_DATA_A_OUT      : out std_logic_vector(11 downto 0);
    ADC1_DATA_B_OUT      : out std_logic_vector(11 downto 0);
    ADC1_DATA_C_OUT      : out std_logic_vector(11 downto 0);
    ADC1_DATA_D_OUT      : out std_logic_vector(11 downto 0);
    ADC1_DATA_VALID_OUT  : out std_logic;

    ADC0_NOTLOCK_COUNTER : out unsigned(7 downto 0);
    ADC1_NOTLOCK_COUNTER : out unsigned(7 downto 0);

    ERROR_ADC0_OUT       : out std_logic;
    ERROR_ADC1_OUT       : out std_logic;
    DEBUG_IN             : in std_logic_vector(3 downto 0);
    DEBUG_OUT            : out std_logic_vector(15 downto 0)
    );
end component;

component adc_ddr_generic
  port (
    clk_0        : in  std_logic;
    clk_1        : in  std_logic;
    clkdiv_reset : in  std_logic;
    eclk         : in  std_logic;
    reset_0      : in  std_logic;
    reset_1      : in  std_logic;
    sclk         : out std_logic;
    datain_0     : in  std_logic_vector(4 downto 0);
    datain_1     : in  std_logic_vector(4 downto 0);
    q_0          : out std_logic_vector(19 downto 0);
    q_1          : out std_logic_vector(19 downto 0)
    );
end component;
```



```
component ddr_generic_single
  port (
    clk_0        : in  std_logic;
    clkdiv_reset : in  std_logic;
    eclk         : in  std_logic;
    reset_0      : in  std_logic;
    sclk         : out std_logic;
    datain_0     : in  std_logic_vector(4 downto 0);
    q_0          : out std_logic_vector(19 downto 0)
    );
end component;

component fifo_adc_48to48_dc
  port (
    Data    : in  std_logic_vector(47 downto 0);
    WrClock : in  std_logic;
    RdClock : in  std_logic;
    WrEn    : in  std_logic;
    RdEn    : in  std_logic;
    Reset   : in  std_logic;
    RPReset : in  std_logic;
    Q       : out std_logic_vector(47 downto 0);
    Empty   : out std_logic;
    Full    : out std_logic
    );
end component;

-----------------------------------------------------------------------------
-- TRBNet Registers
-----------------------------------------------------------------------------

component nx_setup
  port (
    CLK_IN               : in  std_logic;
    RESET_IN             : in  std_logic;
    I2C_COMMAND_OUT      : out std_logic_vector(31 downto 0);
    I2C_COMMAND_BUSY_IN  : in  std_logic;
    I2C_DATA_IN          : in  std_logic_vector(31 downto 0);
    I2C_DATA_BYTES_IN    : in  std_logic_vector(31 downto 0);
    I2C_LOCK_OUT         : out std_logic;
    I2C_ONLINE_OUT       : out std_logic;
    I2C_REG_RESET_IN     : in  std_logic;
    SPI_COMMAND_OUT      : out std_logic_vector(31 downto 0);
    SPI_COMMAND_BUSY_IN  : in  std_logic;
    SPI_DATA_IN          : in  std_logic_vector(31 downto 0);
    SPI_LOCK_OUT         : out std_logic;
    INT_READ_IN          : in  std_logic;
    INT_ADDR_IN          : in  std_logic_vector(15 downto 0);
    INT_ACK_OUT          : out std_logic;
    INT_DATA_OUT         : out std_logic_vector(31 downto 0);
    SLV_READ_IN          : in  std_logic;
    SLV_WRITE_IN         : in  std_logic;
    SLV_DATA_OUT         : out std_logic_vector(31 downto 0);
    SLV_DATA_IN          : in  std_logic_vector(31 downto 0);
    SLV_ADDR_IN          : in  std_logic_vector(15 downto 0);
    SLV_ACK_OUT          : out std_logic;
    SLV_NO_MORE_DATA_OUT : out std_logic;
    SLV_UNKNOWN_ADDR_OUT : out std_logic;
    DEBUG_OUT            : out std_logic_vector(15 downto 0)
    );
end component;
```

```
component nx_control
  port (
    CLK_IN                : in  std_logic;
    RESET_IN              : in  std_logic;
    PLL_NX_CLK_LOCK_IN    : in  std_logic;
    PLL_ADC_DCLK_LOCK_IN  : in  std_logic;
    PLL_ADC_SCLK_LOCK_IN  : in  std_logic;
    I2C_SM_RESET_OUT      : out std_logic;
    I2C_REG_RESET_OUT     : out std_logic;
    NX_TS_RESET_OUT       : out std_logic;
    I2C_ONLINE_IN         : in  std_logic;
    OFFLINE_OUT           : out std_logic;
    ERROR_ALL_IN          : in  std_logic_vector(7 downto 0);
    SLV_READ_IN           : in  std_logic;
    SLV_WRITE_IN          : in  std_logic;
    SLV_DATA_OUT          : out std_logic_vector(31 downto 0);
    SLV_DATA_IN           : in  std_logic_vector(31 downto 0);
    SLV_ADDR_IN           : in  std_logic_vector(15 downto 0);
    SLV_ACK_OUT           : out std_logic;
    SLV_NO_MORE_DATA_OUT  : out std_logic;
    SLV_UNKNOWN_ADDR_OUT  : out std_logic;
    DEBUG_OUT             : out std_logic_vector(15 downto 0)
    );
end component;

component clock10MHz
  port (
    CLK   : in  std_logic;
    CLKOP : out std_logic;
    LOCK  : out std_logic
    );
end component;

component fifo_ts_32to32_dc
  port (
    Data          : in  std_logic_vector(31 downto 0);
    WrClock       : in  std_logic;
    RdClock       : in  std_logic;
    WrEn          : in  std_logic;
    RdEn          : in  std_logic;
    Reset         : in  std_logic;
    RPReset       : in  std_logic;
    Q             : out std_logic_vector(31 downto 0);
    Empty         : out std_logic;
    Full          : out std_logic
    );
end component;

component ram_fifo_delay_256x44
  port (
    WrAddress : in  std_logic_vector(7 downto 0);
    RdAddress : in  std_logic_vector(7 downto 0);
    Data      : in  std_logic_vector(43 downto 0);
    WE        : in  std_logic;
    RdClock   : in  std_logic;
    RdClockEn : in  std_logic;
    Reset     : in  std_logic;
    WrClock   : in  std_logic;
    WrClockEn : in  std_logic;
    Q         : out std_logic_vector(43 downto 0)
    );
end component;
```



```
component fifo_44_data_delay_my
  port (
    Data          : in  std_logic_vector(43 downto 0);
    Clock         : in  std_logic;
    WrEn          : in  std_logic;
    RdEn          : in  std_logic;
    Reset         : in  std_logic;
    AmEmptyThresh : in  std_logic_vector(7 downto 0);
    Q             : out std_logic_vector(43 downto 0);
    Empty         : out std_logic;
    Full          : out std_logic;
    AlmostEmpty   : out std_logic;
    DEBUG_OUT     : out std_logic_vector(15 downto 0)
    );
end component;

component fifo_32_data
  port (
    Data         : in  std_logic_vector(31 downto 0);
    Clock        : in  std_logic;
    WrEn         : in  std_logic;
    RdEn         : in  std_logic;
    Reset        : in  std_logic;
    AmFullThresh : in  std_logic_vector(10 downto 0);
    Q            : out std_logic_vector(31 downto 0);
    Empty        : out std_logic;
    Full         : out std_logic;
    AlmostFull   : out std_logic
    );
end component;

component nx_data_receiver
  port (
    CLK_IN                : in  std_logic;
    RESET_IN              : in  std_logic;
    NX_DATA_CLK_TEST_IN   : in  std_logic;
    TRIGGER_IN            : in  std_logic;
    NX_TIMESTAMP_CLK_IN   : in  std_logic;
    NX_TIMESTAMP_IN       : in  std_logic_vector (7 downto 0);
    ADC_CLK_DAT_IN        : in  std_logic;
    ADC_FCLK_IN           : in  std_logic_vector(1 downto 0);
    ADC_DCLK_IN           : in  std_logic_vector(1 downto 0);
    ADC_SAMPLE_CLK_OUT    : out std_logic;
    ADC_A_IN              : in  std_logic_vector(1 downto 0);
    ADC_B_IN              : in  std_logic_vector(1 downto 0);
    ADC_NX_IN             : in  std_logic_vector(1 downto 0);
    ADC_D_IN              : in  std_logic_vector(1 downto 0);
    ADC_SCLK_LOCK_OUT     : out std_logic;
    NX_TIMESTAMP_OUT      : out std_logic_vector(31 downto 0);
    ADC_DATA_OUT          : out std_logic_vector(11 downto 0);
    NEW_DATA_OUT          : out std_logic;
    SLV_READ_IN           : in  std_logic;
    SLV_WRITE_IN          : in  std_logic;
    SLV_DATA_OUT          : out std_logic_vector(31 downto 0);
    SLV_DATA_IN           : in  std_logic_vector(31 downto 0);
    SLV_ADDR_IN           : in  std_logic_vector(15 downto 0);
    SLV_ACK_OUT           : out std_logic;
    SLV_NO_MORE_DATA_OUT  : out std_logic;
    SLV_UNKNOWN_ADDR_OUT  : out std_logic;
    ERROR_OUT             : out std_logic;
    DEBUG_OUT             : out std_logic_vector(15 downto 0)
```

```
    );
end component;

component nx_data_delay
  port (
    CLK_IN               : in  std_logic;
    RESET_IN             : in  std_logic;
    NX_FRAME_IN          : in  std_logic_vector(31 downto 0);
    ADC_DATA_IN          : in  std_logic_vector(11 downto 0);
    NEW_DATA_IN          : in  std_logic;
    NX_FRAME_OUT         : out std_logic_vector(31 downto 0);
    ADC_DATA_OUT         : out std_logic_vector(11 downto 0);
    NEW_DATA_OUT         : out std_logic;
    FIFO_DELAY_IN        : in  std_logic_vector(7 downto 0);
    SLV_READ_IN          : in  std_logic;
    SLV_WRITE_IN         : in  std_logic;
    SLV_DATA_OUT         : out std_logic_vector(31 downto 0);
    SLV_DATA_IN          : in  std_logic_vector(31 downto 0);
    SLV_ADDR_IN          : in  std_logic_vector(15 downto 0);
    SLV_ACK_OUT          : out std_logic;
    SLV_NO_MORE_DATA_OUT : out std_logic;
    SLV_UNKNOWN_ADDR_OUT : out std_logic;
    DEBUG_OUT            : out std_logic_vector(15 downto 0)
    );
end component;

component nx_data_validate
  port (
    CLK_IN               : in  std_logic;
    RESET_IN             : in  std_logic;
    NX_TIMESTAMP_IN      : in  std_logic_vector(31 downto 0);
    ADC_DATA_IN          : in  std_logic_vector(11 downto 0);
    NEW_DATA_IN          : in  std_logic;
    TIMESTAMP_OUT        : out std_logic_vector(13 downto 0);
    CHANNEL_OUT          : out std_logic_vector(6 downto 0);
    TIMESTAMP_STATUS_OUT : out std_logic_vector(2 downto 0);
    ADC_DATA_OUT         : out std_logic_vector(11 downto 0);
    DATA_VALID_OUT       : out std_logic;
    NX_TOKEN_RETURN_OUT  : out std_logic;
    NX_NOMORE_DATA_OUT   : out std_logic;
    SLV_READ_IN          : in  std_logic;
    SLV_WRITE_IN         : in  std_logic;
    SLV_DATA_OUT         : out std_logic_vector(31 downto 0);
    SLV_DATA_IN          : in  std_logic_vector(31 downto 0);
    SLV_ADDR_IN          : in  std_logic_vector(15 downto 0);
    SLV_ACK_OUT          : out std_logic;
    SLV_NO_MORE_DATA_OUT : out std_logic;
    SLV_UNKNOWN_ADDR_OUT : out std_logic;
    DEBUG_OUT            : out std_logic_vector(15 downto 0)
    );
end component;

component nx_trigger_validate
  generic (
    BOARD_ID : std_logic_vector(1 downto 0)
    );
  port (
    CLK_IN                : in  std_logic;
    RESET_IN              : in  std_logic;
    DATA_CLK_IN           : in  std_logic;
    TIMESTAMP_IN          : in  std_logic_vector(13 downto 0);
    CHANNEL_IN            : in  std_logic_vector(6 downto 0);
```



```
    TIMESTAMP_STATUS_IN    : in  std_logic_vector(2 downto 0);
    ADC_DATA_IN            : in  std_logic_vector(11 downto 0);
    NX_TOKEN_RETURN_IN     : in  std_logic;
    NX_NOMORE_DATA_IN      : in  std_logic;
    TRIGGER_IN             : in  std_logic;
    TRIGGER_BUSY_IN        : in  std_logic;
    FAST_CLEAR_IN          : in  std_logic;
    TRIGGER_BUSY_OUT       : out std_logic;
    TIMESTAMP_FPGA_IN      : in  unsigned(11 downto 0);
    DATA_FIFO_DELAY_OUT    : out std_logic_vector(7 downto 0);
    DATA_OUT               : out std_logic_vector(31 downto 0);
    DATA_CLK_OUT           : out std_logic;
    NOMORE_DATA_OUT        : out std_logic;
    EVT_BUFFER_CLEAR_OUT   : out std_logic;
    EVT_BUFFER_FULL_IN     : in  std_logic;
    HISTOGRAM_FILL_OUT     : out std_logic;
    HISTOGRAM_BIN_OUT      : out std_logic_vector(6 downto 0);
    HISTOGRAM_ADC_OUT      : out std_logic_vector(11 downto 0);
    HISTOGRAM_PILEUP_OUT   : out std_logic;
    HISTOGRAM_OVERFLOW_OUT : out std_logic;
    SLV_READ_IN            : in  std_logic;
    SLV_WRITE_IN           : in  std_logic;
    SLV_DATA_OUT           : out std_logic_vector(31 downto 0);
    SLV_DATA_IN            : in  std_logic_vector(31 downto 0);
    SLV_ADDR_IN            : in  std_logic_vector(15 downto 0);
    SLV_ACK_OUT            : out std_logic;
    SLV_NO_MORE_DATA_OUT   : out std_logic;
    SLV_UNKNOWN_ADDR_OUT   : out std_logic;
    DEBUG_OUT              : out std_logic_vector(15 downto 0)
    );
end component;

component nx_event_buffer
  generic (
    BOARD_ID : std_logic_vector(1 downto 0)
    );
  port (
    CLK_IN                  : in  std_logic;
    RESET_IN                : in  std_logic;
    RESET_DATA_BUFFER_IN    : in  std_logic;
    NXYTER_OFFLINE_IN       : in  std_logic;
    DATA_IN                 : in  std_logic_vector(31 downto 0);
    DATA_CLK_IN             : in  std_logic;
    EVT_NOMORE_DATA_IN      : in  std_logic;
    TRIGGER_IN              : in  std_logic;
    FAST_CLEAR_IN           : in  std_logic;
    TRIGGER_BUSY_OUT        : out std_logic;
    EVT_BUFFER_FULL_OUT     : out std_logic;
    FEE_DATA_OUT            : out std_logic_vector(31 downto 0);
    FEE_DATA_WRITE_OUT      : out std_logic;
    FEE_DATA_ALMOST_FULL_IN : in  std_logic;
    SLV_READ_IN             : in  std_logic;
    SLV_WRITE_IN            : in  std_logic;
    SLV_DATA_OUT            : out std_logic_vector(31 downto 0);
    SLV_DATA_IN             : in  std_logic_vector(31 downto 0);
    SLV_ADDR_IN             : in  std_logic_vector(15 downto 0);
    SLV_ACK_OUT             : out std_logic;
    SLV_NO_MORE_DATA_OUT    : out std_logic;
    SLV_UNKNOWN_ADDR_OUT    : out std_logic;
    ERROR_OUT               : out std_logic;
    DEBUG_OUT               : out std_logic_vector(15 downto 0)
    );
```

```
end component;

component nx_status_event
  generic (
    BOARD_ID : std_logic_vector(1 downto 0));
  port (
    CLK_IN                 : in  std_logic;
    RESET_IN               : in  std_logic;
    NXYTER_OFFLINE_IN      : in  std_logic;
    TRIGGER_IN             : in  std_logic;
    FAST_CLEAR_IN          : in  std_logic;
    TRIGGER_BUSY_OUT       : out std_logic;
    FEE_DATA_OUT           : out std_logic_vector(31 downto 0);
    FEE_DATA_WRITE_OUT     : out std_logic;
    FEE_DATA_FINISHED_OUT  : out std_logic;
    FEE_DATA_ALMOST_FULL_IN : in  std_logic;
    INT_READ_OUT           : out std_logic;
    INT_ADDR_OUT           : out std_logic_vector(15 downto 0);
    INT_ACK_IN             : in  std_logic;
    INT_DATA_IN            : in  std_logic_vector(31 downto 0);
    DEBUG_OUT              : out std_logic_vector(15 downto 0)
    );
end component;

-------------------------------------------------------------------------------

component nx_histogram
  generic (
    BUS_WIDTH  : integer;
    DATA_WIDTH : integer
    );
  port (
    CLK_IN                : in  std_logic;
    RESET_IN              : in  std_logic;
    NUM_AVERAGES_IN       : in  unsigned(2 downto 0);
    AVERAGE_ENABLE_IN     : in  std_logic;
    CHANNEL_ID_IN         : in  std_logic_vector(BUS_WIDTH - 1 downto 0);
    CHANNEL_DATA_IN       : in  std_logic_vector(DATA_WIDTH - 1 downto 0);
    CHANNEL_ADD_IN        : in  std_logic;
    CHANNEL_WRITE_IN      : in  std_logic;
    CHANNEL_WRITE_BUSY_OUT : out std_logic;
    CHANNEL_ID_READ_IN    : in  std_logic_vector(BUS_WIDTH - 1 downto 0);
    CHANNEL_READ_IN       : in  std_logic;
    CHANNEL_DATA_OUT      : out std_logic_vector(DATA_WIDTH - 1 downto 0);
    CHANNEL_DATA_VALID_OUT : out std_logic;
    CHANNEL_READ_BUSY_OUT  : out std_logic;
    DEBUG_OUT             : out std_logic_vector(15 downto 0));
end component;

component nx_histograms
  port (
    CLK_IN              : in  std_logic;
    RESET_IN            : in  std_logic;
    RESET_HISTS_IN      : in  std_logic;
    CHANNEL_FILL_IN     : in  std_logic;
    CHANNEL_ID_IN       : in  std_logic_vector(6 downto 0);
    CHANNEL_ADC_IN      : in  std_logic_vector(11 downto 0);
    CHANNEL_PILEUP_IN   : in  std_logic;
    CHANNEL_OVERFLOW_IN : in  std_logic;
    SLV_READ_IN         : in  std_logic;
    SLV_WRITE_IN        : in  std_logic;
    SLV_DATA_OUT        : out std_logic_vector(31 downto 0);
```



```
    SLV_DATA_IN         : in  std_logic_vector(31 downto 0);
    SLV_ADDR_IN         : in  std_logic_vector(15 downto 0);
    SLV_ACK_OUT         : out std_logic;
    SLV_NO_MORE_DATA_OUT : out std_logic;
    SLV_UNKNOWN_ADDR_OUT : out std_logic;
    DEBUG_OUT           : out std_logic_vector(15 downto 0)
    );
end component;

component ram_dp_128x40
  port (
    WrAddress : in  std_logic_vector(6 downto 0);
    RdAddress : in  std_logic_vector(6 downto 0);
    Data      : in  std_logic_vector(39 downto 0);
    WE        : in  std_logic;
    RdClock   : in  std_logic;
    RdClockEn : in  std_logic;
    Reset     : in  std_logic;
    WrClock   : in  std_logic;
    WrClockEn : in  std_logic;
    Q         : out std_logic_vector(39 downto 0)
    );
end component;

component ram_dp_128x32
  port (
    WrAddress   : in  std_logic_vector(6 downto 0);
    RdAddress   : in  std_logic_vector(6 downto 0);
    Data        : in  std_logic_vector(31 downto 0);
    WE          : in  std_logic;
    RdClock     : in  std_logic;
    RdClockEn   : in  std_logic;
    Reset       : in  std_logic;
    WrClock     : in  std_logic;
    WrClockEn   : in  std_logic;
    Q           : out std_logic_vector(31 downto 0)
    );
end component;

-------------------------------------------------------------------------------

component level_to_pulse
  port (
    CLK_IN         : in  std_logic;
    RESET_IN       : in  std_logic;
    LEVEL_IN       : in  std_logic;
    PULSE_OUT      : out std_logic
    );
end component;

component pulse_to_level
  generic (
    NUM_CYCLES : integer range 2 to 15
    );
  port (
    CLK_IN    : in  std_logic;
    RESET_IN  : in  std_logic;
    PULSE_IN  : in  std_logic;
    LEVEL_OUT : out std_logic
    );
end component;
```

```
component signal_async_to_pulse
  generic (
    NUM_FF : integer range 2 to 4
    );
  port (
    CLK_IN     : in  std_logic;
    RESET_IN   : in  std_logic;
    PULSE_A_IN : in  std_logic;
    PULSE_OUT  : out std_logic
    );
end component;

component signal_async_trans
  generic (
    NUM_FF : integer range 2 to 4
    );
  port (
    CLK_IN      : in  std_logic;
    SIGNAL_A_IN : in  std_logic;
    SIGNAL_OUT  : out std_logic
    );
end component;

component bus_async_trans
  generic (
    BUS_WIDTH : integer range 2 to 32;
    NUM_FF    : integer range 2 to 4);
  port (
    CLK_IN      : in  std_logic;
    RESET_IN    : in  std_logic;
    SIGNAL_A_IN : in  std_logic_vector(BUS_WIDTH - 1 downto 0);
    SIGNAL_OUT  : out std_logic_vector(BUS_WIDTH - 1 downto 0)
    );
end component;

component pulse_dtrans
  generic (
    CLK_RATIO : integer range 2 to 15
    );
  port (
    CLK_A_IN    : in  std_logic;
    RESET_A_IN  : in  std_logic;
    PULSE_A_IN  : in  std_logic;
    CLK_B_IN    : in  std_logic;
    RESET_B_IN  : in  std_logic;
    PULSE_B_OUT : out std_logic
    );
end component;

component Gray_Decoder
  generic (
    WIDTH : integer range 2 to 32
    );
  port (
    CLK_IN     : in  std_logic;
    RESET_IN   : in  std_logic;
    GRAY_IN    : in  std_logic_vector(WIDTH - 1 downto 0);
    BINARY_OUT : out std_logic_vector(WIDTH - 1 downto 0)
    );
end component;

component Gray_Encoder
```



```
  generic (
    WIDTH : integer range 2 to 32
    );
  port (
    CLK_IN    : in  std_logic;
    RESET_IN  : in  std_logic;
    BINARY_IN : in  std_logic_vector(WIDTH - 1 downto 0);
    GRAY_OUT  : out std_logic_vector(WIDTH - 1 downto 0)
    );
end component;

component pulse_delay
  generic (
    DELAY : integer range 2 to 16777216);
  port (
    CLK_IN    : in  std_logic;
    RESET_IN  : in  std_logic;
    PULSE_IN  : in  std_logic;
    PULSE_OUT : out std_logic
    );
end component;

-------------------------------------------------------------------------------
-- PLLs
-------------------------------------------------------------------------------

component pll_nx_clk250
  port (
    CLK   : in  std_logic;
    CLKOP : out std_logic;
    CLKOK : out std_logic;
    LOCK  : out std_logic
    );
end component;

component pll_adc_clk
  port (
    CLK   : in  std_logic;
    CLKOP : out std_logic;
    LOCK  : out std_logic
    );
end component;

component pll_adc_sampling_clk
  port (
    CLK       : in  std_logic;
    RESET     : in  std_logic;
    FINEDELB0 : in  std_logic;
    FINEDELB1 : in  std_logic;
    FINEDELB2 : in  std_logic;
    FINEDELB3 : in  std_logic;
    DPHASE0   : in  std_logic;
    DPHASE1   : in  std_logic;
    DPHASE2   : in  std_logic;
    DPHASE3   : in  std_logic;
    CLKOP     : out std_logic;
    CLKOS     : out std_logic;
    LOCK      : out std_logic
    );
end component;

component nx_fpga_timestamp
```

```
  port (
    CLK_IN                  : in  std_logic;
    RESET_IN                : in  std_logic;
    NX_MAIN_CLK_IN          : in  std_logic;
    TIMESTAMP_SYNC_IN       : in  std_logic;
    TRIGGER_IN              : in  std_logic;
    TIMESTAMP_CURRENT_OUT   : out unsigned(11 downto 0);
    TIMESTAMP_HOLD_OUT      : out unsigned(11 downto 0);
    TIMESTAMP_SYNCED_OUT    : out std_logic;
    TIMESTAMP_TRIGGER_OUT   : out std_logic;
    SLV_READ_IN             : in  std_logic;
    SLV_WRITE_IN            : in  std_logic;
    SLV_DATA_OUT            : out std_logic_vector(31 downto 0);
    SLV_DATA_IN             : in  std_logic_vector(31 downto 0);
    SLV_ACK_OUT             : out std_logic;
    SLV_NO_MORE_DATA_OUT    : out std_logic;
    SLV_UNKNOWN_ADDR_OUT    : out std_logic;
    DEBUG_OUT               : out std_logic_vector(15 downto 0)
    );
end component;

component nx_trigger_handler
  port (
    CLK_IN                    : in  std_logic;
    RESET_IN                  : in  std_logic;
    NX_MAIN_CLK_IN            : in  std_logic;
    NXYTER_OFFLINE_IN         : in  std_logic;
    TIMING_TRIGGER_IN         : in  std_logic;
    LVL1_TRG_DATA_VALID_IN    : in  std_logic;
    LVL1_VALID_TIMING_TRG_IN  : in  std_logic;
    LVL1_VALID_NOTIMING_TRG_IN : in  std_logic;
    LVL1_INVALID_TRG_IN       : in  std_logic;
    LVL1_TRG_TYPE_IN          : in  std_logic_vector(3 downto 0);
    LVL1_TRG_NUMBER_IN        : in  std_logic_vector(15 downto 0);
    LVL1_TRG_CODE_IN          : in  std_logic_vector(7 downto 0);
    LVL1_TRG_INFORMATION_IN   : in  std_logic_vector(23 downto 0);
    LVL1_INT_TRG_NUMBER_IN    : in  std_logic_vector(15 downto 0);
    FEE_DATA_OUT              : out std_logic_vector(31 downto 0);
    FEE_DATA_WRITE_OUT        : out std_logic;
    FEE_DATA_FINISHED_OUT     : out std_logic;
    FEE_TRG_RELEASE_OUT       : out std_logic;
    FEE_TRG_STATUSBITS_OUT    : out std_logic_vector(31 downto 0);
    FEE_DATA_0_IN             : in  std_logic_vector(31 downto 0);
    FEE_DATA_WRITE_0_IN       : in  std_logic;
    FEE_DATA_1_IN             : in  std_logic_vector(31 downto 0);
    FEE_DATA_WRITE_1_IN       : in  std_logic;
    INTERNAL_TRIGGER_IN       : in  std_logic;
    TRIGGER_VALIDATE_BUSY_IN  : in  std_logic;
    TRIGGER_BUSY_0_IN         : in  std_logic;
    TRIGGER_BUSY_1_IN         : in  std_logic;
    VALID_TRIGGER_OUT         : out std_logic;
    TIMESTAMP_TRIGGER_OUT     : out std_logic;
    TRIGGER_TIMING_OUT        : out std_logic;
    TRIGGER_STATUS_OUT        : out std_logic;
    FAST_CLEAR_OUT            : out std_logic;
    TRIGGER_BUSY_OUT          : out std_logic;
    TRIGGER_TESTPULSE_OUT     : out std_logic;
    SLV_READ_IN               : in  std_logic;
    SLV_WRITE_IN              : in  std_logic;
    SLV_DATA_OUT              : out std_logic_vector(31 downto 0);
    SLV_DATA_IN               : in  std_logic_vector(31 downto 0);
    SLV_ADDR_IN               : in  std_logic_vector(15 downto 0);
```



```
    SLV_ACK_OUT               : out std_logic;
    SLV_NO_MORE_DATA_OUT      : out std_logic;
    SLV_UNKNOWN_ADDR_OUT      : out std_logic;
    DEBUG_OUT                 : out std_logic_vector(15 downto 0)
    );
end component;

component nx_trigger_generator
  port (
    CLK_IN              : in  std_logic;
    RESET_IN            : in  std_logic;
    NX_MAIN_CLK_IN      : in  std_logic;
    TRIGGER_IN          : in  std_logic;
    TRIGGER_OUT         : out std_logic;
    TS_RESET_OUT        : out std_logic;
    TESTPULSE_OUT       : out std_logic;
    TIMESTAMP_IN        : in  std_logic_vector(31 downto 0);
    ADC_DATA_IN         : in  std_logic_vector(11 downto 0);
    NEW_DATA_IN         : in  std_logic;
    SELF_TRIGGER_OUT    : out std_logic;
    SLV_READ_IN         : in  std_logic;
    SLV_WRITE_IN        : in  std_logic;
    SLV_DATA_OUT        : out std_logic_vector(31 downto 0);
    SLV_DATA_IN         : in  std_logic_vector(31 downto 0);
    SLV_ADDR_IN         : in  std_logic_vector(15 downto 0);
    SLV_ACK_OUT         : out std_logic;
    SLV_NO_MORE_DATA_OUT : out std_logic;
    SLV_UNKNOWN_ADDR_OUT : out std_logic;
    DEBUG_OUT           : out std_logic_vector(15 downto 0)
    );
end component;

------------------------------------------------------------------------------
-- Misc Tools
------------------------------------------------------------------------------

component timer
  generic (
    CTR_WIDTH : integer range 2 to 32;
    STEP_SIZE : integer range 1 to 100
    );
  port (
    CLK_IN         : in  std_logic;
    RESET_IN       : in  std_logic;
    TIMER_START_IN : in  std_logic;
    TIMER_END_IN   : in  unsigned(CTR_WIDTH - 1 downto 0);
    TIMER_DONE_OUT : out std_logic
    );
end component;

component timer_static
  generic (
    CTR_WIDTH : integer range 2 to 32;
    CTR_END   : integer;
    STEP_SIZE : integer range 1 to 100
    );
  port (
    CLK_IN         : in  std_logic;
    RESET_IN       : in  std_logic;
    TIMER_START_IN : in  std_logic;
    TIMER_DONE_OUT : out std_logic
    );
```

```
end component;

-------------------------------------------------------------------------------
-- Simulations
-------------------------------------------------------------------------------

component nxyter_timestamp_sim
  port (
    CLK_IN        : in  std_logic;
    RESET_IN      : in  std_logic;
    TIMESTAMP_OUT : out std_logic_vector(7 downto 0);
    CLK128_OUT    : out std_logic
    );
end component;

type debug_array_t is array(integer range <>) of std_logic_vector(15 downto 0);

component debug_multiplexer
  generic (
    NUM_PORTS : integer range 1 to 32
    );
  port (
    CLK_IN               : in  std_logic;
    RESET_IN             : in  std_logic;
    DEBUG_LINE_IN        : in  debug_array_t(0 to NUM_PORTS-1);
    DEBUG_LINE_OUT       : out std_logic_vector(15 downto 0);
    SLV_READ_IN          : in  std_logic;
    SLV_WRITE_IN         : in  std_logic;
    SLV_DATA_OUT         : out std_logic_vector(31 downto 0);
    SLV_DATA_IN          : in  std_logic_vector(31 downto 0);
    SLV_ADDR_IN          : in  std_logic_vector(15 downto 0);
    SLV_ACK_OUT          : out std_logic;
    SLV_NO_MORE_DATA_OUT : out std_logic;
    SLV_UNKNOWN_ADDR_OUT : out std_logic
    );
end component;

end package;
-------------------------------------------------------------------------------
--
-- One  nXyter FEB
--
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.trb_net_std.all;
use work.trb_net_components.all;
use work.trb3_components.all;
use work.nxyter_components.all;

entity nXyter_FEE_board is
  generic (
    BOARD_ID : std_logic_vector(1 downto 0) := "11"
    );
  port (
    CLK_IN                    : in  std_logic;
    RESET_IN                  : in  std_logic;
    CLK_NX_MAIN_IN            : in  std_logic;
    CLK_ADC_IN                : in  std_logic;
```



```
    PLL_NX_CLK_LOCK_IN        : in  std_logic;
    PLL_ADC_DCLK_LOCK_IN      : in  std_logic;
    NX_DATA_CLK_TEST_IN       : in  std_logic;
    TRIGGER_OUT               : out std_logic;

    -- I2C Ports
    I2C_SDA_INOUT             : inout std_logic;  -- nXyter I2C fdata line
    I2C_SCL_INOUT             : inout std_logic;  -- nXyter I2C Clock line
    I2C_SM_RESET_OUT          : out std_logic;    -- reset nXyter I2C SMachine
    I2C_REG_RESET_OUT         : out std_logic;    -- reset I2C registers

    -- ADC SPI
    SPI_SCLK_OUT              : out std_logic;
    SPI_SDIO_INOUT            : inout std_logic;
    SPI_CSB_OUT               : out std_logic;

    -- nXyter Timestamp Ports
    NX_DATA_CLK_IN            : in  std_logic;
    NX_TIMESTAMP_IN           : in  std_logic_vector (7 downto 0);
    NX_RESET_OUT              : out std_logic;
    NX_TESTPULSE_OUT          : out std_logic;
    NX_TIMESTAMP_TRIGGER_OUT  : out std_logic;

    -- ADC nXyter Pulse Hight Ports
    ADC_FCLK_IN               : in  std_logic_vector(1 downto 0);
    ADC_DCLK_IN               : in  std_logic_vector(1 downto 0);
    ADC_SAMPLE_CLK_OUT        : out std_logic;
    ADC_A_IN                  : in  std_logic_vector(1 downto 0);
    ADC_B_IN                  : in  std_logic_vector(1 downto 0);
    ADC_NX_IN                 : in  std_logic_vector(1 downto 0);
    ADC_D_IN                  : in  std_logic_vector(1 downto 0);

    -- Input Triggers
    TIMING_TRIGGER_IN         : in  std_logic;
    LVL1_TRG_DATA_VALID_IN    : in  std_logic;
    LVL1_VALID_TIMING_TRG_IN  : in  std_logic;
    LVL1_VALID_NOTIMING_TRG_IN : in  std_logic; -- Status + Info TypE
    LVL1_INVALID_TRG_IN       : in  std_logic;

    LVL1_TRG_TYPE_IN          : in  std_logic_vector(3 downto 0);
    LVL1_TRG_NUMBER_IN        : in  std_logic_vector(15 downto 0);
    LVL1_TRG_CODE_IN          : in  std_logic_vector(7 downto 0);
    LVL1_TRG_INFORMATION_IN   : in  std_logic_vector(23 downto 0);
    LVL1_INT_TRG_NUMBER_IN    : in  std_logic_vector(15 downto 0);

    --Response from FEE
    FEE_TRG_RELEASE_OUT       : out std_logic;
    FEE_TRG_STATUSBITS_OUT    : out std_logic_vector(31 downto 0);
    FEE_DATA_OUT              : out std_logic_vector(31 downto 0);
    FEE_DATA_WRITE_OUT        : out std_logic;
    FEE_DATA_FINISHED_OUT     : out std_logic;
    FEE_DATA_ALMOST_FULL_IN   : in  std_logic;

    -- TRBNet RegIO Port for the slave bus
    REGIO_ADDR_IN             : in  std_logic_vector(15 downto 0);
    REGIO_DATA_IN             : in  std_logic_vector(31 downto 0);
    REGIO_DATA_OUT            : out std_logic_vector(31 downto 0);
    REGIO_READ_ENABLE_IN      : in  std_logic;
    REGIO_WRITE_ENABLE_IN     : in  std_logic;
    REGIO_TIMEOUT_IN          : in  std_logic;
    REGIO_DATAREADY_OUT       : out std_logic;
    REGIO_WRITE_ACK_OUT       : out std_logic;
```

```
    REGIO_NO_MORE_DATA_OUT     : out std_logic;
    REGIO_UNKNOWN_ADDR_OUT     : out std_logic;

    -- Debug Signals
    DEBUG_LINE_OUT             : out   std_logic_vector(15 downto 0)
    );

end entity;


architecture Behavioral of nXyter_FEE_board is

-------------------------------------------------------------------------------
-- Signals
-------------------------------------------------------------------------------

  -- Bus Handler
  constant NUM_PORTS            : integer := 13;

  signal slv_read               : std_logic_vector(NUM_PORTS-1 downto 0);
  signal slv_write              : std_logic_vector(NUM_PORTS-1 downto 0);
  signal slv_no_more_data       : std_logic_vector(NUM_PORTS-1 downto 0);
  signal slv_ack                : std_logic_vector(NUM_PORTS-1 downto 0);
  signal slv_addr               : std_logic_vector(NUM_PORTS*16-1 downto 0);
  signal slv_data_rd            : std_logic_vector(NUM_PORTS*32-1 downto 0);
  signal slv_data_wr            : std_logic_vector(NUM_PORTS*32-1 downto 0);
  signal slv_unknown_addr       : std_logic_vector(NUM_PORTS-1 downto 0);

  -- TRB Register
  signal i2c_sm_reset_o         : std_logic;
  signal nx_ts_reset_1          : std_logic;
  signal nx_ts_reset_2          : std_logic;
  signal nx_ts_reset_o          : std_logic;
  signal i2c_reg_reset_o        : std_logic;
  signal nxyter_offline         : std_logic;

  -- NX Register Access
  signal i2c_lock               : std_logic;
  signal i2c_command            : std_logic_vector(31 downto 0);
  signal i2c_command_busy       : std_logic;
  signal i2c_data               : std_logic_vector(31 downto 0);
  signal i2c_data_bytes         : std_logic_vector(31 downto 0);
  signal spi_lock               : std_logic;
  signal spi_command            : std_logic_vector(31 downto 0);
  signal spi_command_busy       : std_logic;
  signal spi_data               : std_logic_vector(31 downto 0);
  signal nxyter_online_i2c      : std_logic;

  -- SPI Interface ADC
  signal spi_sdi                : std_logic;
  signal spi_sdo                : std_logic;

  -- Data Receiver
  signal timestamp_recv         : std_logic_vector(31 downto 0);
  signal adc_data_recv          : std_logic_vector(11 downto 0);
  signal data_clk_recv          : std_logic;
  signal self_trigger           : std_logic;
  signal pll_sadc_clk_lock      : std_logic;

  -- Data Delay
  signal timestamp_delayed      : std_logic_vector(31 downto 0);
  signal adc_data_delayed       : std_logic_vector(11 downto 0);
```



```
  signal data_clk_delayed       : std_logic;
  signal data_fifo_delay        : std_logic_vector(7 downto 0);

  -- Data Validate
  signal timestamp              : std_logic_vector(13 downto 0);
  signal timestamp_channel_id   : std_logic_vector(6 downto 0);
  signal timestamp_status       : std_logic_vector(2 downto 0);
  signal adc_data               : std_logic_vector(11 downto 0);
  signal data_valid             : std_logic;

  signal nx_token_return        : std_logic;
  signal nx_nomore_data         : std_logic;

  -- Trigger Validate
  signal trigger_data           : std_logic_vector(31 downto 0);
  signal trigger_data_clk       : std_logic;
  signal event_buffer_clear     : std_logic;
  signal trigger_validate_busy  : std_logic;
  signal validate_nomore_data   : std_logic;

  signal trigger_validate_fill   : std_logic;
  signal trigger_validate_bin    : std_logic_vector(6 downto 0);
  signal trigger_validate_adc    : std_logic_vector(11 downto 0);
  signal trigger_validate_pileup : std_logic;
  signal trigger_validate_ovfl   : std_logic;

  -- Event Buffer
  signal fee_data_o_0           : std_logic_vector(31 downto 0);
  signal fee_data_write_o_0     : std_logic;

  signal trigger_evt_busy_0     : std_logic;
  signal evt_buffer_full        : std_logic;
  signal fee_trg_statusbits_o   : std_logic_vector(31 downto 0);
  signal fee_data_o             : std_logic_vector(31 downto 0);
  signal fee_data_write_o       : std_logic;
  signal fee_data_finished_o    : std_logic;
  signal fee_almost_full_i      : std_logic;

  -- Calib Event
  signal fee_data_o_1           : std_logic_vector(31 downto 0);
  signal fee_data_write_o_1     : std_logic;
  signal trigger_evt_busy_1     : std_logic;

  signal int_read               : std_logic;
  signal int_addr               : std_logic_vector(15 downto 0);
  signal int_ack                : std_logic;
  signal int_data               : std_logic_vector(31 downto 0);

  -- Trigger Handler
  signal trigger                : std_logic;
  signal timestamp_trigger      : std_logic;
  signal trigger_timing         : std_logic;
  signal trigger_status         : std_logic;
  signal trigger_busy           : std_logic;
  signal fast_clear             : std_logic;
  signal fee_trg_release_o      : std_logic;
  signal trigger_testpulse      : std_logic;

  -- FPGA Timestamp
  signal timestamp_current      : unsigned(11 downto 0);
  signal timestamp_hold         : unsigned(11 downto 0);
  signal nx_timestamp_sync      : std_logic;
```

```
  signal nx_timestamp_trigger_o : std_logic;

  -- Trigger Generator
  signal trigger_intern         : std_logic;
  signal nx_testpulse_o         : std_logic;

  -- Error
  signal error_all              : std_logic_vector(7 downto 0);
  signal error_data_receiver    : std_logic;
  signal error_event_buffer     : std_logic;

  -- Debug Handler
  constant DEBUG_NUM_PORTS      : integer := 14;
  signal debug_line             : debug_array_t(0 to DEBUG_NUM_PORTS-1);

begin

--------------------------------------------------------------------------------
-- DEBUG
--------------------------------------------------------------------------------
  -- DEBUG_LINE_OUT(0)           <= CLK_IN;
  -- DEBUG_LINE_OUT(15 downto 0) <= (others => '0');
  -- See Multiplexer

--------------------------------------------------------------------------------
-- Errors
--------------------------------------------------------------------------------
  error_all(0)          <= error_data_receiver;
  error_all(1)          <= error_event_buffer;
  error_all(7 downto 2) <= (others => '0');

--------------------------------------------------------------------------------
-- Port Maps
--------------------------------------------------------------------------------

  THE_BUS_HANDLER: trb_net16_regio_bus_handler
    generic map(
      PORT_NUMBER         => NUM_PORTS,

      PORT_ADDRESSES      => (  0 => x"0100",    -- NX Control Handler
                                1 => x"0040",    -- I2C Master
                                2 => x"0500",    -- Data Receiver
                                3 => x"0600",    -- Data Buffer
                                4 => x"0060",    -- SPI Master
                                5 => x"0140",    -- Trigger Generator
                                6 => x"0120",    -- Data Validate
                                7 => x"0160",    -- Trigger Handler
                                8 => x"0400",    -- Trigger Validate
                                9 => x"0200",    -- NX Setup
                               10 => x"0800",    -- NX Histograms
                               11 => x"0020",    -- Debug Handler
                               12 => x"0130",    -- Data Delay
                                others => x"0000"
                                ),

      PORT_ADDR_MASK      => (  0 => 4,          -- NX Control Handler
                                1 => 1,          -- I2C master
                                2 => 5,          -- Data Receiver
                                3 => 3,          -- Data Buffer
                                4 => 0,          -- SPI Master
                                5 => 3,          -- Trigger Generator
                                6 => 4,          -- Data Validate
```



```
                                7 => 4,          -- Trigger Handler
                                8 => 5,          -- Trigger Validate
                                9 => 9,          -- NX Setup
                               10 => 10,         -- NX Histograms
                               11 => 0,          -- Debug Handler
                               12 => 2,          -- Data Delay
                                others => 0
                                ),

      PORT_MASK_ENABLE           => 1
      )
    port map(
      CLK                        => CLK_IN,
      RESET                      => RESET_IN,

      DAT_ADDR_IN                => REGIO_ADDR_IN,
      DAT_DATA_IN                => REGIO_DATA_IN,
      DAT_DATA_OUT               => REGIO_DATA_OUT,
      DAT_READ_ENABLE_IN         => REGIO_READ_ENABLE_IN,
      DAT_WRITE_ENABLE_IN        => REGIO_WRITE_ENABLE_IN,
      DAT_TIMEOUT_IN             => REGIO_TIMEOUT_IN,
      DAT_DATAREADY_OUT          => REGIO_DATAREADY_OUT,
      DAT_WRITE_ACK_OUT          => REGIO_WRITE_ACK_OUT,
      DAT_NO_MORE_DATA_OUT       => REGIO_NO_MORE_DATA_OUT,
      DAT_UNKNOWN_ADDR_OUT       => REGIO_UNKNOWN_ADDR_OUT,

      -- All NXYTER Ports
      BUS_READ_ENABLE_OUT        => slv_read,
      BUS_WRITE_ENABLE_OUT       => slv_write,
      BUS_DATA_OUT               => slv_data_wr,
      BUS_DATA_IN                => slv_data_rd,
      BUS_ADDR_OUT               => slv_addr,
      BUS_TIMEOUT_OUT            => open,
      BUS_DATAREADY_IN           => slv_ack,
      BUS_WRITE_ACK_IN           => slv_ack,
      BUS_NO_MORE_DATA_IN        => slv_no_more_data,
      BUS_UNKNOWN_ADDR_IN        => slv_unknown_addr,

      -- DEBUG
      STAT_DEBUG                 => open
      );

--------------------------------------------------------------------------------
-- Registers
--------------------------------------------------------------------------------
  nx_control_1: nx_control
    port map (
      CLK_IN                  => CLK_IN,
      RESET_IN                => RESET_IN,

      PLL_NX_CLK_LOCK_IN      => PLL_NX_CLK_LOCK_IN,
      PLL_ADC_DCLK_LOCK_IN    => PLL_ADC_DCLK_LOCK_IN,
      PLL_ADC_SCLK_LOCK_IN    => pll_sadc_clk_lock,

      I2C_SM_RESET_OUT        => i2c_sm_reset_o,
      I2C_REG_RESET_OUT       => i2c_reg_reset_o,
      NX_TS_RESET_OUT         => nx_ts_reset_1,
      I2C_ONLINE_IN           => nxyter_online_i2c,
      OFFLINE_OUT             => nxyter_offline,

      ERROR_ALL_IN            => error_all,
```

```
      SLV_READ_IN             => slv_read(0),
      SLV_WRITE_IN            => slv_write(0),
      SLV_DATA_OUT            => slv_data_rd(0*32+31 downto 0*32),
      SLV_DATA_IN             => slv_data_wr(0*32+31 downto 0*32),
      SLV_ADDR_IN             => slv_addr(0*16+15 downto 0*16),
      SLV_ACK_OUT             => slv_ack(0),
      SLV_NO_MORE_DATA_OUT    => slv_no_more_data(0),
      SLV_UNKNOWN_ADDR_OUT    => slv_unknown_addr(0),

      DEBUG_OUT               => debug_line(0)
      );

  nx_setup_1: nx_setup
    port map (
      CLK_IN               => CLK_IN,
      RESET_IN             => RESET_IN,
      I2C_COMMAND_OUT      => i2c_command,
      I2C_COMMAND_BUSY_IN  => i2c_command_busy,
      I2C_DATA_IN          => i2c_data,
      I2C_DATA_BYTES_IN    => i2c_data_bytes,
      I2C_LOCK_OUT         => i2c_lock,
      I2C_ONLINE_OUT       => nxyter_online_i2c,
      I2C_REG_RESET_IN     => i2c_reg_reset_o,
      SPI_COMMAND_OUT      => spi_command,
      SPI_COMMAND_BUSY_IN  => spi_command_busy,
      SPI_DATA_IN          => spi_data,
      SPI_LOCK_OUT         => spi_lock,
      INT_READ_IN          => int_read,
      INT_ADDR_IN          => int_addr,
      INT_ACK_OUT          => int_ack,
      INT_DATA_OUT         => int_data,
      SLV_READ_IN          => slv_read(9),
      SLV_WRITE_IN         => slv_write(9),
      SLV_DATA_OUT         => slv_data_rd(9*32+31 downto 9*32),
      SLV_DATA_IN          => slv_data_wr(9*32+31 downto 9*32),
      SLV_ADDR_IN          => slv_addr(9*16+15 downto 9*16),
      SLV_ACK_OUT          => slv_ack(9),
      SLV_NO_MORE_DATA_OUT => slv_no_more_data(9),
      SLV_UNKNOWN_ADDR_OUT => slv_unknown_addr(9),

      DEBUG_OUT            => debug_line(1)
      );

------------------------------------------------------------------------------
-- I2C master block for accessing the nXyter
------------------------------------------------------------------------------

  nx_i2c_master_1: nx_i2c_master
    generic map (
      I2C_SPEED => x"3e8"
      )
    port map (
      CLK_IN               => CLK_IN,
      RESET_IN             => RESET_IN,
      SDA_INOUT            => I2C_SDA_INOUT,
      SCL_INOUT            => I2C_SCL_INOUT,
      INTERNAL_COMMAND_IN  => i2c_command,
      COMMAND_BUSY_OUT     => i2c_command_busy,
      I2C_DATA_OUT         => i2c_data,
      I2C_DATA_BYTES_OUT   => i2c_data_bytes,
      I2C_LOCK_IN          => i2c_lock,
      SLV_READ_IN          => slv_read(1),
```



```
      SLV_WRITE_IN         => slv_write(1),
      SLV_DATA_OUT         => slv_data_rd(1*32+31 downto 1*32),
      SLV_DATA_IN          => slv_data_wr(1*32+31 downto 1*32),
      SLV_ADDR_IN          => slv_addr(1*16+15 downto 1*16),
      SLV_ACK_OUT          => slv_ack(1),
      SLV_NO_MORE_DATA_OUT => slv_no_more_data(1),
      SLV_UNKNOWN_ADDR_OUT => slv_unknown_addr(1),

      DEBUG_OUT            => debug_line(2)
      );

------------------------------------------------------------------------------
-- SPI master block to access the ADC
------------------------------------------------------------------------------

  adc_spi_master_1: adc_spi_master
    generic map (
      SPI_SPEED => x"c8"
      )
    port map (
      CLK_IN               => CLK_IN,
      RESET_IN             => RESET_IN,
      SCLK_OUT             => SPI_SCLK_OUT,
      SDIO_INOUT           => SPI_SDIO_INOUT,
      CSB_OUT              => SPI_CSB_OUT,
      INTERNAL_COMMAND_IN  => spi_command,
      COMMAND_BUSY_OUT     => spi_command_busy,
      SPI_DATA_OUT         => spi_data,
      SPI_LOCK_IN          => spi_lock,
      SLV_READ_IN          => slv_read(4),
      SLV_WRITE_IN         => slv_write(4),
      SLV_DATA_OUT         => slv_data_rd(4*32+31 downto 4*32),
      SLV_DATA_IN          => slv_data_wr(4*32+31 downto 4*32),
      SLV_ACK_OUT          => slv_ack(4),
      SLV_NO_MORE_DATA_OUT => slv_no_more_data(4),
      SLV_UNKNOWN_ADDR_OUT => slv_unknown_addr(4),

      DEBUG_OUT            => debug_line(3)
      );

------------------------------------------------------------------------------
-- FPGA Timestamp
------------------------------------------------------------------------------

  nx_fpga_timestamp_1: nx_fpga_timestamp
    port map (
      CLK_IN                   => CLK_IN,
      RESET_IN                 => RESET_IN,
      NX_MAIN_CLK_IN           => CLK_NX_MAIN_IN,
      TIMESTAMP_SYNC_IN        => nx_ts_reset_o,
      TRIGGER_IN               => timestamp_trigger,
      TIMESTAMP_CURRENT_OUT    => timestamp_current,
      TIMESTAMP_HOLD_OUT       => timestamp_hold,
      TIMESTAMP_SYNCED_OUT     => nx_timestamp_sync,
      TIMESTAMP_TRIGGER_OUT    => nx_timestamp_trigger_o,
      SLV_READ_IN              => open,
      SLV_WRITE_IN             => open,
      SLV_DATA_OUT             => open,
      SLV_DATA_IN              => open,
      SLV_ACK_OUT              => open,
      SLV_NO_MORE_DATA_OUT     => open,
      SLV_UNKNOWN_ADDR_OUT     => open,
```

```
      DEBUG_OUT                => debug_line(4)
      );

-------------------------------------------------------------------------------
-- Trigger Handler
-------------------------------------------------------------------------------

  nx_trigger_handler_1: nx_trigger_handler
    port map (
      CLK_IN                     => CLK_IN,
      RESET_IN                   => RESET_IN,
      NX_MAIN_CLK_IN             => CLK_NX_MAIN_IN,
      NXYTER_OFFLINE_IN          => nxyter_offline,

      TIMING_TRIGGER_IN          => TIMING_TRIGGER_IN,
      LVL1_TRG_DATA_VALID_IN     => LVL1_TRG_DATA_VALID_IN,
      LVL1_VALID_TIMING_TRG_IN   => LVL1_VALID_TIMING_TRG_IN,
      LVL1_VALID_NOTIMING_TRG_IN => LVL1_VALID_NOTIMING_TRG_IN,
      LVL1_INVALID_TRG_IN        => LVL1_INVALID_TRG_IN,

      LVL1_TRG_TYPE_IN           => LVL1_TRG_TYPE_IN,
      LVL1_TRG_NUMBER_IN         => LVL1_TRG_NUMBER_IN,
      LVL1_TRG_CODE_IN           => LVL1_TRG_CODE_IN,
      LVL1_TRG_INFORMATION_IN    => LVL1_TRG_INFORMATION_IN,
      LVL1_INT_TRG_NUMBER_IN     => LVL1_INT_TRG_NUMBER_IN,

      FEE_DATA_OUT               => FEE_DATA_OUT,
      FEE_DATA_WRITE_OUT         => FEE_DATA_WRITE_OUT,
      FEE_DATA_FINISHED_OUT      => FEE_DATA_FINISHED_OUT,
      FEE_TRG_RELEASE_OUT        => FEE_TRG_RELEASE_OUT,
      FEE_TRG_STATUSBITS_OUT     => FEE_TRG_STATUSBITS_OUT,

      FEE_DATA_0_IN              => fee_data_o_0,
      FEE_DATA_WRITE_0_IN        => fee_data_write_o_0,
      FEE_DATA_1_IN              => fee_data_o_1,
      FEE_DATA_WRITE_1_IN        => fee_data_write_o_1,
      INTERNAL_TRIGGER_IN        => trigger_intern,

      TRIGGER_VALIDATE_BUSY_IN   => trigger_validate_busy,
      TRIGGER_BUSY_0_IN          => trigger_evt_busy_0,
      TRIGGER_BUSY_1_IN          => trigger_evt_busy_1,

      VALID_TRIGGER_OUT          => trigger,
      TIMESTAMP_TRIGGER_OUT      => timestamp_trigger,
      TRIGGER_TIMING_OUT         => trigger_timing,
      TRIGGER_STATUS_OUT         => trigger_status,
      FAST_CLEAR_OUT             => fast_clear,
      TRIGGER_BUSY_OUT           => trigger_busy,

      TRIGGER_TESTPULSE_OUT      => trigger_testpulse,

      SLV_READ_IN                => slv_read(7),
      SLV_WRITE_IN               => slv_write(7),
      SLV_DATA_OUT               => slv_data_rd(7*32+31 downto 7*32),
      SLV_DATA_IN                => slv_data_wr(7*32+31 downto 7*32),
      SLV_ADDR_IN                => slv_addr(7*16+15 downto 7*16),
      SLV_ACK_OUT                => slv_ack(7),
      SLV_NO_MORE_DATA_OUT       => slv_no_more_data(7),
      SLV_UNKNOWN_ADDR_OUT       => slv_unknown_addr(7),

      DEBUG_OUT                  => debug_line(5)
```



```
      );

-------------------------------------------------------------------------------
-- NX Trigger Generator
-------------------------------------------------------------------------------

  nx_trigger_generator_1: nx_trigger_generator
    port map (
      CLK_IN                => CLK_IN,
      RESET_IN              => RESET_IN,
      NX_MAIN_CLK_IN        => CLK_NX_MAIN_IN,
      TRIGGER_IN            => trigger_testpulse,
      TRIGGER_OUT           => trigger_intern,
      TS_RESET_OUT          => nx_ts_reset_2,
      TESTPULSE_OUT         => nx_testpulse_o,

      TIMESTAMP_IN          => timestamp_recv,
      ADC_DATA_IN           => adc_data_recv,
      NEW_DATA_IN           => data_clk_recv,
      SELF_TRIGGER_OUT      => self_trigger,

      SLV_READ_IN           => slv_read(5),
      SLV_WRITE_IN          => slv_write(5),
      SLV_DATA_OUT          => slv_data_rd(5*32+31 downto 5*32),
      SLV_DATA_IN           => slv_data_wr(5*32+31 downto 5*32),
      SLV_ADDR_IN           => slv_addr(5*16+15 downto 5*16),
      SLV_ACK_OUT           => slv_ack(5),
      SLV_NO_MORE_DATA_OUT  => slv_no_more_data(5),
      SLV_UNKNOWN_ADDR_OUT  => slv_unknown_addr(5),

      DEBUG_OUT             => debug_line(6)
      );

-------------------------------------------------------------------------------
-- nXyter Data Receiver
-------------------------------------------------------------------------------

  nx_data_receiver_1: nx_data_receiver
    port map (
      CLK_IN                => CLK_IN,
      RESET_IN              => RESET_IN,
      NX_DATA_CLK_TEST_IN   => NX_DATA_CLK_TEST_IN,
      TRIGGER_IN            => trigger_timing,

      NX_TIMESTAMP_CLK_IN   => NX_DATA_CLK_IN,
      NX_TIMESTAMP_IN       => NX_TIMESTAMP_IN,

      ADC_CLK_DAT_IN        => CLK_ADC_IN,
      ADC_FCLK_IN           => ADC_FCLK_IN,
      ADC_DCLK_IN           => ADC_DCLK_IN,
      ADC_SAMPLE_CLK_OUT    => ADC_SAMPLE_CLK_OUT,
      ADC_A_IN              => ADC_A_IN,
      ADC_B_IN              => ADC_B_IN,
      ADC_NX_IN             => ADC_NX_IN,
      ADC_D_IN              => ADC_D_IN,
      ADC_SCLK_LOCK_OUT     => pll_sadc_clk_lock,

      NX_TIMESTAMP_OUT      => timestamp_recv,
      ADC_DATA_OUT          => adc_data_recv,
      NEW_DATA_OUT          => data_clk_recv,

      SLV_READ_IN           => slv_read(2),
```

```
      SLV_WRITE_IN         => slv_write(2),
      SLV_DATA_OUT         => slv_data_rd(2*32+31 downto 2*32),
      SLV_DATA_IN          => slv_data_wr(2*32+31 downto 2*32),
      SLV_ADDR_IN          => slv_addr(2*16+15 downto 2*16),
      SLV_ACK_OUT          => slv_ack(2),
      SLV_NO_MORE_DATA_OUT => slv_no_more_data(2),
      SLV_UNKNOWN_ADDR_OUT => slv_unknown_addr(2),
      ERROR_OUT            => error_data_receiver,
      DEBUG_OUT            => debug_line(7)
      );

-------------------------------------------------------------------------------
-- NX and ADC Data Delay FIFO
-------------------------------------------------------------------------------
  nx_data_delay_1: nx_data_delay
    port map (
      CLK_IN               => CLK_IN,
      RESET_IN             => RESET_IN,

      NX_FRAME_IN          => timestamp_recv,
      ADC_DATA_IN          => adc_data_recv,
      NEW_DATA_IN          => data_clk_recv,

      NX_FRAME_OUT         => timestamp_delayed,
      ADC_DATA_OUT         => adc_data_delayed,
      NEW_DATA_OUT         => data_clk_delayed,

      FIFO_DELAY_IN        => data_fifo_delay,

      SLV_READ_IN          => slv_read(12),
      SLV_WRITE_IN         => slv_write(12),
      SLV_DATA_OUT         => slv_data_rd(12*32+31 downto 12*32),
      SLV_DATA_IN          => slv_data_wr(12*32+31 downto 12*32),
      SLV_ADDR_IN          => slv_addr(12*16+15 downto 12*16),
      SLV_ACK_OUT          => slv_ack(12),
      SLV_NO_MORE_DATA_OUT => slv_no_more_data(12),
      SLV_UNKNOWN_ADDR_OUT => slv_unknown_addr(12),

      DEBUG_OUT            => debug_line(8)
      );

-------------------------------------------------------------------------------
-- Timestamp Decoder and Valid Data Filter
-------------------------------------------------------------------------------

  nx_data_validate_1: nx_data_validate
    port map (
      CLK_IN                => CLK_IN,
      RESET_IN              => RESET_IN,

      NX_TIMESTAMP_IN       => timestamp_delayed,
      ADC_DATA_IN           => adc_data_delayed,
      NEW_DATA_IN           => data_clk_delayed,

      TIMESTAMP_OUT         => timestamp,
      CHANNEL_OUT           => timestamp_channel_id,
      TIMESTAMP_STATUS_OUT  => timestamp_status,
      ADC_DATA_OUT          => adc_data,
      DATA_VALID_OUT        => data_valid,

      NX_TOKEN_RETURN_OUT   => nx_token_return,
      NX_NOMORE_DATA_OUT    => nx_nomore_data,
```



```
      SLV_READ_IN          => slv_read(6),
      SLV_WRITE_IN         => slv_write(6),
      SLV_DATA_OUT         => slv_data_rd(6*32+31 downto 6*32),
      SLV_DATA_IN          => slv_data_wr(6*32+31 downto 6*32),
      SLV_ADDR_IN          => slv_addr(6*16+15 downto 6*16),
      SLV_ACK_OUT          => slv_ack(6),
      SLV_NO_MORE_DATA_OUT => slv_no_more_data(6),
      SLV_UNKNOWN_ADDR_OUT => slv_unknown_addr(6),

      DEBUG_OUT            => debug_line(9)
      );

-------------------------------------------------------------------------------
-- NX Trigger Validate
-------------------------------------------------------------------------------

  nx_trigger_validate_1: nx_trigger_validate
    generic map (
      BOARD_ID => BOARD_ID
      )
    port map (
      CLK_IN                    => CLK_IN,
      RESET_IN                  => RESET_IN,

      DATA_CLK_IN               => data_valid,
      TIMESTAMP_IN              => timestamp,
      CHANNEL_IN                => timestamp_channel_id,
      TIMESTAMP_STATUS_IN       => timestamp_status,
      ADC_DATA_IN               => adc_data,
      NX_TOKEN_RETURN_IN        => nx_token_return,
      NX_NOMORE_DATA_IN         => nx_nomore_data,

      TRIGGER_IN                => trigger,
      TRIGGER_BUSY_IN           => trigger_busy,
      FAST_CLEAR_IN             => fast_clear,
      TRIGGER_BUSY_OUT          => trigger_validate_busy,
      TIMESTAMP_FPGA_IN         => timestamp_hold,
      DATA_FIFO_DELAY_OUT       => data_fifo_delay,

      DATA_OUT                  => trigger_data,
      DATA_CLK_OUT              => trigger_data_clk,
      NOMORE_DATA_OUT           => validate_nomore_data,
      EVT_BUFFER_CLEAR_OUT      => event_buffer_clear,
      EVT_BUFFER_FULL_IN        => evt_buffer_full,

      HISTOGRAM_FILL_OUT        => trigger_validate_fill,
      HISTOGRAM_BIN_OUT         => trigger_validate_bin,
      HISTOGRAM_ADC_OUT         => trigger_validate_adc,
      HISTOGRAM_PILEUP_OUT      => trigger_validate_pileup,
      HISTOGRAM_OVERFLOW_OUT    => trigger_validate_ovfl,

      SLV_READ_IN               => slv_read(8),
      SLV_WRITE_IN              => slv_write(8),
      SLV_DATA_OUT              => slv_data_rd(8*32+31 downto 8*32),
      SLV_DATA_IN               => slv_data_wr(8*32+31 downto 8*32),
      SLV_ADDR_IN               => slv_addr(8*16+15 downto 8*16),
      SLV_ACK_OUT               => slv_ack(8),
      SLV_NO_MORE_DATA_OUT      => slv_no_more_data(8),
      SLV_UNKNOWN_ADDR_OUT      => slv_unknown_addr(8),

      DEBUG_OUT                 => debug_line(10)
```

```
     );

-------------------------------------------------------------------------------
-- Data Buffer FIFO
-------------------------------------------------------------------------------

  nx_event_buffer_1: nx_event_buffer
    generic map (
      BOARD_ID => BOARD_ID
      )
    port map (
      CLK_IN                    => CLK_IN,
      RESET_IN                  => RESET_IN,
      RESET_DATA_BUFFER_IN      => event_buffer_clear,
      NXYTER_OFFLINE_IN         => nxyter_offline,

      DATA_IN                   => trigger_data,
      DATA_CLK_IN               => trigger_data_clk,
      EVT_NOMORE_DATA_IN        => validate_nomore_data,

      TRIGGER_IN                => trigger_timing,
      FAST_CLEAR_IN             => fast_clear,
      TRIGGER_BUSY_OUT          => trigger_evt_busy_0,
      EVT_BUFFER_FULL_OUT       => evt_buffer_full,

      FEE_DATA_OUT              => fee_data_o_0,
      FEE_DATA_WRITE_OUT        => fee_data_write_o_0,
      FEE_DATA_ALMOST_FULL_IN   => FEE_DATA_ALMOST_FULL_IN,

      SLV_READ_IN               => slv_read(3),
      SLV_WRITE_IN              => slv_write(3),
      SLV_DATA_OUT              => slv_data_rd(3*32+31 downto 3*32),
      SLV_DATA_IN               => slv_data_wr(3*32+31 downto 3*32),
      SLV_ADDR_IN               => slv_addr(3*16+15 downto 3*16),
      SLV_ACK_OUT               => slv_ack(3),
      SLV_NO_MORE_DATA_OUT      => slv_no_more_data(3),
      SLV_UNKNOWN_ADDR_OUT      => slv_unknown_addr(3),

      ERROR_OUT                 => error_event_buffer,
      DEBUG_OUT                 => debug_line(11)
      );

  nx_status_event_1: nx_status_event
    generic map (
      BOARD_ID => BOARD_ID
      )
    port map (
      CLK_IN                  => CLK_IN,
      RESET_IN                => RESET_IN,
      NXYTER_OFFLINE_IN       => nxyter_offline,
      TRIGGER_IN              => trigger_status,
      FAST_CLEAR_IN           => fast_clear,
      TRIGGER_BUSY_OUT        => trigger_evt_busy_1,
      FEE_DATA_OUT            => fee_data_o_1,
      FEE_DATA_WRITE_OUT      => fee_data_write_o_1,
      FEE_DATA_ALMOST_FULL_IN => FEE_DATA_ALMOST_FULL_IN,
      INT_READ_OUT            => int_read,
      INT_ADDR_OUT            => int_addr,
      INT_ACK_IN              => int_ack,
      INT_DATA_IN             => int_data,
      DEBUG_OUT               => debug_line(13)
      );
```



```
  nx_histograms_1: nx_histograms
    port map (
      CLK_IN                    => CLK_IN,
      RESET_IN                  => RESET_IN,

      RESET_HISTS_IN            => '0',
      CHANNEL_FILL_IN           => trigger_validate_fill,
      CHANNEL_ID_IN             => trigger_validate_bin,
      CHANNEL_ADC_IN            => trigger_validate_adc,
      CHANNEL_PILEUP_IN         => trigger_validate_pileup,
      CHANNEL_OVERFLOW_IN       => trigger_validate_ovfl,

      SLV_READ_IN               => slv_read(10),
      SLV_WRITE_IN              => slv_write(10),
      SLV_DATA_OUT              => slv_data_rd(10*32+31 downto 10*32),
      SLV_DATA_IN               => slv_data_wr(10*32+31 downto 10*32),
      SLV_ADDR_IN               => slv_addr(10*16+15 downto 10*16),
      SLV_ACK_OUT               => slv_ack(10),
      SLV_NO_MORE_DATA_OUT      => slv_no_more_data(10),
      SLV_UNKNOWN_ADDR_OUT      => slv_unknown_addr(10),

      DEBUG_OUT                 => debug_line(12)
      );

-------------------------------------------------------------------------------
-- nXyter Signals
-------------------------------------------------------------------------------
  nx_ts_reset_o     <= nx_ts_reset_1 or nx_ts_reset_2;
  NX_RESET_OUT      <= not nx_ts_reset_o;
  NX_TESTPULSE_OUT  <= nx_testpulse_o;

-------------------------------------------------------------------------------
-- I2C Signals
-------------------------------------------------------------------------------

  I2C_SM_RESET_OUT  <= not i2c_sm_reset_o;
  I2C_REG_RESET_OUT <= not i2c_reg_reset_o;

-------------------------------------------------------------------------------
-- Others
-------------------------------------------------------------------------------
  NX_TIMESTAMP_TRIGGER_OUT <= nx_timestamp_trigger_o;
  TRIGGER_OUT              <= self_trigger;

-------------------------------------------------------------------------------
-- DEBUG Line Select
-------------------------------------------------------------------------------
  debug_multiplexer_1: debug_multiplexer
    generic map (
      NUM_PORTS => DEBUG_NUM_PORTS
      )
    port map (
      CLK_IN              => CLK_IN,
      RESET_IN            => RESET_IN,
      DEBUG_LINE_IN       => debug_line,
      DEBUG_LINE_OUT      => DEBUG_LINE_OUT,
      SLV_READ_IN         => slv_read(11),
      SLV_WRITE_IN        => slv_write(11),
      SLV_DATA_OUT        => slv_data_rd(11*32+31 downto 11*32),
      SLV_DATA_IN         => slv_data_wr(11*32+31 downto 11*32),
      SLV_ADDR_IN         => slv_addr(11*16+15 downto 11*16),
```

```
      SLV_ACK_OUT          => slv_ack(11),
      SLV_NO_MORE_DATA_OUT => slv_no_more_data(11),
      SLV_UNKNOWN_ADDR_OUT => slv_unknown_addr(11)
      );

-------------------------------------------------------------------------------
-- END
-------------------------------------------------------------------------------

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.nxyter_components.all;

entity pulse_delay is
  generic (
    DELAY : integer range 2 to 16777216 := 100
    );
  port (
    CLK_IN        : in  std_logic;
    RESET_IN      : in  std_logic;

    PULSE_IN      : in  std_logic;
    PULSE_OUT     : out std_logic
    );

end entity;

architecture Behavioral of pulse_delay is
  signal start_timer_x  : std_logic;

  signal start_timer    : std_logic;
  signal timer_done     : std_logic;
  signal pulse_o        : std_logic;

  type STATES is (IDLE,
                  WAIT_TIMER
                  );
  signal STATE, NEXT_STATE : STATES;

begin

  timer_static_1: timer_static
    generic map (
      CTR_WIDTH => 24,
      CTR_END   => (DELAY - 1)
      )
    port map (
      CLK_IN          => CLK_IN,
      RESET_IN        => RESET_IN,
      TIMER_START_IN  => start_timer,
      TIMER_DONE_OUT  => timer_done
      );

  PROC_CONVERT_TRANSFER: process(CLK_IN)
  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1' ) then
        start_timer    <= '0';
        STATE          <= IDLE;
```



```
      else
        start_timer    <= start_timer_x;
        STATE          <= NEXT_STATE;
      end if;
    end if;
  end process PROC_CONVERT_TRANSFER;

  PROC_CONVERT: process(STATE,
                        PULSE_IN,
                        timer_done
                        )

  begin
    pulse_o               <= '0';
    case STATE is
      when IDLE =>
        if (PULSE_IN = '1') then
          start_timer_x    <= '1';
          pulse_o          <= '0';
          NEXT_STATE       <= WAIT_TIMER;
        else
          start_timer_x    <= '0';
          pulse_o               <= '0';
          NEXT_STATE       <= IDLE;
        end if;

      when WAIT_TIMER =>
        start_timer_x      <= '0';
        if (timer_done = '0') then
          pulse_o          <= '0';
          NEXT_STATE       <= WAIT_TIMER;
        else
          pulse_o          <= '1';
          NEXT_STATE       <= IDLE;
        end if;

    end case;
  end process PROC_CONVERT;

  -- Output Signals
  PULSE_OUT   <= pulse_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.nxyter_components.all;

entity pulse_dtrans is
  generic (
    CLK_RATIO : integer range 2 to 15 := 4
    );
  port (
    CLK_A_IN     : in  std_logic;
    RESET_A_IN   : in  std_logic;
    PULSE_A_IN   : in  std_logic;
    CLK_B_IN     : in  std_logic;
    RESET_B_IN   : in  std_logic;
    PULSE_B_OUT  : out std_logic
    );
```

```
end entity;

architecture Behavioral of pulse_dtrans is
  attribute HGROUP : string;
  attribute HGROUP of Behavioral : architecture is "PULSE_DTRANS";

  signal pulse_a_l      : std_logic;
  signal pulse_b_o      : std_logic;

begin
  ------------------------------------------------------------------------------
  -- Clock A Domain
  ------------------------------------------------------------------------------
  pulse_to_level_1: pulse_to_level
    generic map (
      NUM_CYCLES => CLK_RATIO
      )
    port map (
      CLK_IN     => CLK_A_IN,
      RESET_IN   => RESET_A_IN,
      PULSE_IN   => PULSE_A_IN,
      LEVEL_OUT  => pulse_a_l
      );

  ------------------------------------------------------------------------------
  -- Clock B Domain
  ------------------------------------------------------------------------------

  signal_async_to_pulse_1: signal_async_to_pulse
    generic map (
      NUM_FF => 2
      )
    port map (
      CLK_IN     => CLK_B_IN,
      RESET_IN   => RESET_B_IN,
      PULSE_A_IN => pulse_a_l,
      PULSE_OUT  => pulse_b_o
      );

  -- Outputs
  PULSE_B_OUT   <= pulse_b_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.nxyter_components.all;

entity pulse_to_level is
  generic (
    NUM_CYCLES    : integer range 2 to 15 := 4
    );
  port (
    CLK_IN        : in  std_logic;
    RESET_IN      : in  std_logic;

    PULSE_IN      : in  std_logic;
    LEVEL_OUT     : out std_logic
    );

end entity;
```



```

architecture Behavioral of pulse_to_level is
  attribute HGROUP : string;
  attribute HGROUP of Behavioral : architecture is "PULSE_TO_LEVEL";

  signal start_timer_x  : std_logic;

  signal start_timer    : std_logic;
  signal timer_done     : std_logic;
  signal level_o        : std_logic;

  type STATES is (IDLE,
                  WAIT_TIMER
                  );
  signal STATE, NEXT_STATE : STATES;

begin

  timer_static_1: timer_static
    generic map (
      CTR_WIDTH => 5,
      CTR_END   => NUM_CYCLES
      )
    port map (
      CLK_IN          => CLK_IN,
      RESET_IN        => RESET_IN,
      TIMER_START_IN  => start_timer,
      TIMER_DONE_OUT  => timer_done
      );

  PROC_LEVEL_OUT_TRANSFER: process(CLK_IN)
  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1' ) then
        start_timer    <= '0';
        STATE          <= IDLE;
      else
        start_timer    <= start_timer_x;
        STATE          <= NEXT_STATE;
      end if;
    end if;
  end process PROC_LEVEL_OUT_TRANSFER;

  PROC_LEVEL_OUT: process(STATE,
                          PULSE_IN,
                          timer_done
                          )
  begin

    case STATE is
      when IDLE =>
        if (PULSE_IN = '1') then
          level_o          <= '1';
          start_timer_x    <= '1';
          NEXT_STATE       <= WAIT_TIMER;
        else
          level_o          <= '0';
          start_timer_x    <= '0';
          NEXT_STATE       <= IDLE;
        end if;

      when WAIT_TIMER =>
```

```
        start_timer_x      <= '0';
        if (timer_done = '0') then
          level_o          <= '1';
          NEXT_STATE       <= WAIT_TIMER;
        else
          level_o          <= '0';
          NEXT_STATE       <= IDLE;
        end if;

    end case;
  end process PROC_LEVEL_OUT;

  -- Output Signals
  LEVEL_OUT   <= level_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.nxyter_components.all;

entity signal_async_to_pulse is
  generic (
    NUM_FF : integer range 2 to 4 := 2
    );
  port (
    CLK_IN       : in  std_logic;
    RESET_IN     : in  std_logic;
    PULSE_A_IN   : in  std_logic;
    PULSE_OUT    : out std_logic
    );

end entity;

architecture Behavioral of signal_async_to_pulse is
  attribute HGROUP : string;
  attribute HGROUP of Behavioral : architecture is "SIGNAL_ASYNC_TO_PULSE";

  signal pulse_ff      : std_logic_vector(NUM_FF - 1 downto 0);
  signal pulse_o       : std_logic;

begin

  ------------------------------------------------------------------------------
  -- Clock CLK_IN Domain
  ------------------------------------------------------------------------------

  PROC_SYNC_PULSE: process(CLK_IN)
  begin
    if( rising_edge(CLK_IN) ) then
      pulse_ff(NUM_FF - 1)            <= PULSE_A_IN;
      for i in NUM_FF - 2 downto 0 loop
        pulse_ff(i)                  <= pulse_ff(i + 1);
      end loop;
    end if;
  end process PROC_SYNC_PULSE;

  level_to_pulse_1: level_to_pulse
    port map (
      CLK_IN    => CLK_IN,
      RESET_IN  => RESET_IN,
```



```
      LEVEL_IN  => pulse_ff(0),
      PULSE_OUT => pulse_o
      );

  -- Outputs
  PULSE_OUT     <= pulse_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity signal_async_trans is
  generic (
    NUM_FF : integer range 2 to 5 := 2
    );
  port (
    CLK_IN       : in  std_logic;
    SIGNAL_A_IN  : in  std_logic;
    SIGNAL_OUT   : out std_logic
    );

end entity;

architecture Behavioral of signal_async_trans is
  type signal_ff_t is array(0 to NUM_FF - 1) of std_logic;

  signal signal_ff      : signal_ff_t;

begin

  ------------------------------------------------------------------------------
  -- Clock CLK_IN Domain
  ------------------------------------------------------------------------------

  PROC_SYNC_SIGNAL: process(CLK_IN)
  begin
    if( rising_edge(CLK_IN) ) then
      signal_ff(NUM_FF - 1)   <= SIGNAL_A_IN;
      for i in NUM_FF - 2 downto 0 loop
        signal_ff(i)          <= signal_ff(i + 1);
      end loop;
    end if;
  end process PROC_SYNC_SIGNAL;

-- Output Signals
  SIGNAL_OUT      <= signal_ff(0);

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity timer is
  generic (
    CTR_WIDTH : integer range 2 to 32  := 12;
    STEP_SIZE : integer range 1 to 100 := 1
    );
  port(
    CLK_IN              : in  std_logic;
    RESET_IN            : in  std_logic;
```

```
    TIMER_START_IN       : in  std_logic;
    TIMER_END_IN         : in  unsigned(CTR_WIDTH - 1 downto 0);
    TIMER_DONE_OUT       : out std_logic
    );
end entity;

architecture Behavioral of timer is
  attribute HGROUP : string;
  attribute HGROUP of Behavioral : architecture is "NX_TIMER";

  -- Timer
  signal timer_ctr_x     : unsigned(CTR_WIDTH - 1 downto 0);

  signal timer_ctr       : unsigned(CTR_WIDTH - 1 downto 0);
  signal timer_done_o    : std_logic;

  type STATES is (S_IDLE,
                  S_COUNT
                  );
  signal STATE, NEXT_STATE : STATES;

begin

  PROC_TIMER_TRANSFER: process(CLK_IN)
  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1' ) then
        timer_ctr      <= (others => '0');
        STATE          <= S_IDLE;
      else
        timer_ctr      <= timer_ctr_x;
        STATE          <= NEXT_STATE;
      end if;
    end if;
  end process PROC_TIMER_TRANSFER;

  PROC_TIMER: process(STATE,
                      TIMER_START_IN,
                      TIMER_END_IN,
                      timer_ctr
                      )
  begin

    case STATE is
      when S_IDLE =>
        timer_done_o      <= '0';
        if (TIMER_START_IN = '1' and TIMER_END_IN > 0) then
          timer_ctr_x     <= TIMER_END_IN - 1;
          NEXT_STATE      <= S_COUNT;
        else
          timer_ctr_x     <= (others => '0');
          NEXT_STATE      <= S_IDLE;
        end if;

      when S_COUNT =>
        if (timer_ctr > to_unsigned(STEP_SIZE - 1, CTR_WIDTH)) then
          timer_ctr_x     <= timer_ctr - to_unsigned(STEP_SIZE, CTR_WIDTH);
          timer_done_o    <= '0';
          NEXT_STATE      <= S_COUNT;
        else
          timer_ctr_x     <= (others => '0');
          timer_done_o    <= '1';
```



```
          NEXT_STATE      <= S_IDLE;
        end if;

    end case;

  end process PROC_TIMER;

  -----------------------------------------------------------------------------
  -- Output Signals
  -----------------------------------------------------------------------------

  TIMER_DONE_OUT <= timer_done_o;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity timer_static is
  generic (
    CTR_WIDTH  : integer range 2 to 32   := 12;
    CTR_END    : integer range 2 to 4000 := 10;
    STEP_SIZE  : integer range 1 to 100  := 1
    );
  port(
    CLK_IN               : in  std_logic;
    RESET_IN             : in  std_logic;

    TIMER_START_IN       : in  std_logic;
    TIMER_DONE_OUT       : out std_logic
    );
end entity;

architecture Behavioral of timer_static is
  attribute HGROUP : string;
  attribute HGROUP of Behavioral : architecture is "NX_TIMER_STATIC";

  -- Timer
  constant ctr_limit     : unsigned(CTR_WIDTH - 1 downto 0)
    := to_unsigned(CTR_END - 1, CTR_WIDTH);
  signal timer_ctr_x     : unsigned(CTR_WIDTH - 1 downto 0);

  signal timer_ctr       : unsigned(CTR_WIDTH - 1 downto 0);
  signal timer_done_o    : std_logic;

  type STATES is (S_IDLE,
                  S_COUNT
                  );
  signal STATE, NEXT_STATE : STATES;

begin

  PROC_TIMER_TRANSFER: process(CLK_IN)
  begin
    if( rising_edge(CLK_IN) ) then
      if( RESET_IN = '1' ) then
        timer_ctr      <= (others => '0');
        STATE          <= S_IDLE;
      else
        timer_ctr      <= timer_ctr_x;
        STATE          <= NEXT_STATE;
      end if;
```

```
    end if;
  end process PROC_TIMER_TRANSFER;

  PROC_TIMER: process(STATE,
                      TIMER_START_IN,
                      timer_ctr
                      )
  begin

    case STATE is
      when S_IDLE =>
        timer_done_o      <= '0';
        if (TIMER_START_IN = '1') then
          timer_ctr_x     <= ctr_limit - 1;
          NEXT_STATE      <= S_COUNT;
        else
          timer_ctr_x     <= (others => '0');
          NEXT_STATE      <= S_IDLE;
        end if;

      when S_COUNT =>
        if (timer_ctr > to_unsigned(STEP_SIZE - 1, CTR_WIDTH)) then
          timer_ctr_x     <= timer_ctr - to_unsigned(STEP_SIZE, CTR_WIDTH);
          timer_done_o    <= '0';
          NEXT_STATE      <= S_COUNT;
        else
          timer_ctr_x     <= (others => '0');
          timer_done_o    <= '1';
          NEXT_STATE      <= S_IDLE;
        end if;

    end case;

  end process PROC_TIMER;

  ------------------------------------------------------------------------------
  -- Output Signals
  ------------------------------------------------------------------------------

  TIMER_DONE_OUT <= timer_done_o;

end Behavioral;
```