```
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
  generic (
    DEBUG_ENABLE : boolean := false
    );
  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_CLK_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_CLK_OUT    : out std_logic;

    ADC0_LOCKED_OUT      : out std_logic;
    ADC1_LOCKED_OUT      : out std_logic;

    ADC0_SLOPPY_FRAME    : in  std_logic;
    ADC1_SLOPPY_FRAME    : in  std_logic;

    ADC0_ERROR_OUT       : out std_logic;
    ADC1_ERROR_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_ff                 : std_logic_vector(19 downto 0);
  signal q_0_f                  : std_logic_vector(19 downto 0);
  signal q_0                    : std_logic_vector(19 downto 0);
  signal q_1_ff                 : std_logic_vector(19 downto 0);
  signal q_1_f                  : std_logic_vector(19 downto 0);
  signal q_1                    : std_logic_vector(19 downto 0);

  -- ADC Data Handler
  signal adc0_error_status      : std_logic_vector(2 downto 0);
  signal adc1_error_status      : std_logic_vector(2 downto 0);
  signal adc0_error_status_sl   : std_logic_vector(2 downto 0);
  signal adc1_error_status_sl   : std_logic_vector(2 downto 0);

  -- Data Types
  type adc_data_t is array(0 to 3) of std_logic_vector(11 downto 0);

  -- Output
  signal adc0_data_clk_o        : std_logic;
  signal adc0_data_o            : adc_data_t;
  signal adc0_locked_o          : std_logic;
  signal adc0_error_o           : std_logic;

  signal adc1_data_clk_o        : std_logic;
  signal adc1_data_o            : adc_data_t;
  signal adc1_locked_o          : std_logic;
  signal adc1_error_o           : std_logic;

  -- RESET Handler
  type R_STATES is (R_IDLE,
                    R_WAIT_CLKDIV,
                    R_WAIT_RESET_ADC,
                    R_WAIT_RESET_END
                    );
  signal R_STATE : R_STATES;

  signal startup_reset          : std_logic;
  signal timer_reset            : std_logic;
  signal wait_timer_start       : std_logic;
  signal wait_timer_done        : std_logic;
  signal RESET_CLKDIV           : std_logic;
  signal RESET_ADC0             : std_logic;
  signal RESET_ADC1             : std_logic;

  signal RESET_ADC0_CLKD_F      : std_logic;
  signal RESET_ADC0_CLKD        : std_logic;
  signal RESET_ADC1_CLKD_F      : std_logic;
  signal RESET_ADC1_CLKD        : std_logic;

  --
  attribute syn_keep : boolean;

  attribute syn_keep of q_0_ff                : signal is true;
  attribute syn_keep of q_0_f                 : signal is true;
  attribute syn_keep of q_1_ff                : signal is true;
  attribute syn_keep of q_1_f                 : signal is true;

  attribute syn_keep of RESET_ADC0_CLKD_F     : signal is true;
  attribute syn_keep of RESET_ADC0_CLKD       : signal is true;
  attribute syn_keep of RESET_ADC1_CLKD_F     : signal is true;
```

```
  attribute syn_keep of RESET_ADC1_CLKD        : signal is true;

  attribute syn_preserve : boolean;

  attribute syn_preserve of q_0_ff             : signal is true;
  attribute syn_preserve of q_0_f              : signal is true;
  attribute syn_preserve of q_1_ff             : signal is true;
  attribute syn_preserve of q_1_f              : signal is true;

begin

  -------------------------------------------------------------------------------
  -- Debug Handler
  -------------------------------------------------------------------------------

  DFALSE: if (DEBUG_ENABLE = false) generate
    DEBUG_OUT               <= (others => '0');
    --DEBUG_OUT(0)            <= CLK_IN;
    --DEBUG_OUT(1)            <= DDR_DATA_CLK;
    --DEBUG_OUT(2)            <= adc0_write_enable;
    --DEBUG_OUT(3)            <= adc0_fifo_full;
    --DEBUG_OUT(4)            <= adc0_fifo_empty;
    --DEBUG_OUT(5)            <= adc0_data_clk_m;
    --DEBUG_OUT(6)            <= adc0_read_enable;
    --DEBUG_OUT(7)            <= adc0_read_enable_t;
    --DEBUG_OUT(8)            <= adc0_read_enable_tt;
    --DEBUG_OUT(9)            <= adc0_data_clk_o;
    --DEBUG_OUT(10)           <= adc0_error;
    --DEBUG_OUT(11)           <= adc0_frame_locked;
    --DEBUG_OUT(12)           <= adc0_frame_clk_ok;
    --DEBUG_OUT(13)           <= wait_timer_done;
    --DEBUG_OUT(14)           <= RESET_CLKDIV;
    --DEBUG_OUT(15)           <= RESET_ADC0;
  end generate DFALSE;

  DTRUE: if (DEBUG_ENABLE = true) generate

    PROC_DEBUG: process (DEBUG_IN)
    begin
      DEBUG_OUT(0)               <= CLK_IN;
      DEBUG_OUT(1)               <= DDR_DATA_CLK;

      case DEBUG_IN is

        when others =>
          DEBUG_OUT(15 downto 2)  <= (others => '0');

      end case;
    end process PROC_DEBUG;

  end generate DTRUE;

  -------------------------------------------------------------------------------
  -- DDR Generic Interface to ADC
  -------------------------------------------------------------------------------
  adc_ddr_generic_1: entity work.adc_ddr_generic
    port map (
      clk_0          => ADC0_DCLK_IN,
      clk_1          => ADC1_DCLK_IN,
      clkdiv_reset   => RESET_CLKDIV,
      eclk           => CLK_ADCDAT_IN,
      reset_0        => RESET_ADC0,
```



```
      reset_1        => RESET_ADC1,
      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_ff,
      q_1            => q_1_ff
      );

  -- Two FIFOs to relaxe timing
  q_0_f   <= q_0_ff when rising_edge(DDR_DATA_CLK);
  q_0     <= q_0_f  when rising_edge(DDR_DATA_CLK);

  q_1_f   <= q_1_ff when rising_edge(DDR_DATA_CLK);
  q_1     <= q_1_f  when rising_edge(DDR_DATA_CLK);

  -- The ADC Data Handlers
  adc_ad9228_data_handler_1: entity work.adc_ad9228_data_handler
    generic map (
      DEBUG_ENABLE => DEBUG_ENABLE
      )
    port map (
      CLK_IN              => CLK_IN,
      RESET_IN            => RESET_ADC0,
      DDR_DATA_CLK        => DDR_DATA_CLK,
      DDR_DATA_IN         => q_0,
      DATA_A_OUT          => adc0_data_o(0),
      DATA_B_OUT          => adc0_data_o(1),
      DATA_C_OUT          => adc0_data_o(2),
      DATA_D_OUT          => adc0_data_o(3),
      DATA_CLK_OUT        => adc0_data_clk_o,
      SLOPPY_FRAME_IN     => ADC0_SLOPPY_FRAME,
      FRAME_LOCKED_OUT    => adc0_locked_o,
      ERROR_STATUS_OUT    => adc0_error_status,
      ERROR_STATUS_SL_OUT => adc0_error_status_sl,
      DEBUG_OUT           => open
      );

  adc_ad9228_data_handler_2: entity work.adc_ad9228_data_handler
    generic map (
      DEBUG_ENABLE => DEBUG_ENABLE
      )
    port map (
      CLK_IN              => CLK_IN,
      RESET_IN            => RESET_ADC1,
      DDR_DATA_CLK        => DDR_DATA_CLK,
      DDR_DATA_IN         => q_1,
      DATA_A_OUT          => adc1_data_o(0),
      DATA_B_OUT          => adc1_data_o(1),
      DATA_C_OUT          => adc1_data_o(2),
      DATA_D_OUT          => adc1_data_o(3),
      DATA_CLK_OUT        => adc1_data_clk_o,
```

```
    SLOPPY_FRAME_IN     => ADC1_SLOPPY_FRAME,
    FRAME_LOCKED_OUT    => adc1_locked_o,
    ERROR_STATUS_OUT    => open, --ERROR_STATUS_OUT,
    ERROR_STATUS_SL_OUT => open, --ERROR_STATUS_OUT,
    DEBUG_OUT           => open
    );

-------------------------------------------------------------------------------
-- Error Status Handler
-------------------------------------------------------------------------------
PROC_ERROR_STATUS: process(CLK_IN)
begin
  if (rising_edge(CLK_IN)) then
    if (RESET_IN = '1') then
      adc0_error_o       <= '0';
      adc1_error_o       <= '0';
    else
      adc0_error_o       <= '0';
      adc1_error_o       <= '0';

      if (adc0_error_status /= "000" or
          (ADC0_SLOPPY_FRAME = '1' and adc0_error_status_sl /= "000")) then
        adc0_error_o     <= '1';
      end if;

      if (adc1_error_status /= "000" or
          (ADC1_SLOPPY_FRAME = '1' and adc1_error_status_sl /= "000")) then
        adc1_error_o     <= '1';
      end if;
    end if;
  end if;
end process PROC_ERROR_STATUS;

-------------------------------------------------------------------------------
-- Reset Handler
-------------------------------------------------------------------------------

timer_static_RESET_TIMER: timer_static
  generic map (
    CTR_WIDTH => 20,
    CTR_END   => 625000 -- 5ms
    )
  port map (
    CLK_IN         => CLK_IN,
    RESET_IN       => timer_reset,
    TIMER_START_IN => wait_timer_start,
    TIMER_DONE_OUT => wait_timer_done
    );

PROC_DDR_RESET_HANDLER: process(CLK_IN)
begin
  if (rising_edge(CLK_IN)) then
    if (RESET_IN = '1') then
      RESET_CLKDIV      <= '1';
      RESET_ADC0        <= '1';
      RESET_ADC1        <= '1';
      wait_timer_start  <= '0';
      timer_reset       <= '1';
      startup_reset     <= '1';
      R_STATE           <= R_IDLE;
    else
      RESET_CLKDIV      <= '0';
```



```
      RESET_ADC0        <= '0';
      RESET_ADC1        <= '0';
      wait_timer_start  <= '0';
      timer_reset       <= '0';
      startup_reset     <= '0';

      case R_STATE is
        when R_IDLE =>
          if (startup_reset = '1') then
            -- Start Reset
            RESET_CLKDIV      <= '1';
            RESET_ADC0        <= '1';
            RESET_ADC1        <= '1';
            wait_timer_start  <= '1';
            R_STATE           <= R_WAIT_CLKDIV;
          else
            timer_reset       <= '1';
            R_STATE           <= R_IDLE;
          end if;

        when R_WAIT_CLKDIV =>
          if (wait_timer_done = '0') then
            RESET_CLKDIV      <= '1';
            RESET_ADC0        <= '1';
            RESET_ADC1        <= '1';
            R_STATE           <= R_WAIT_CLKDIV;
          else
            -- Release RESET_CLKDIV
            RESET_ADC0        <= '1';
            RESET_ADC1        <= '1';
            wait_timer_start  <= '1';
            R_STATE           <= R_WAIT_RESET_ADC;
          end if;

        when R_WAIT_RESET_ADC =>
          if (wait_timer_done = '0') then
            RESET_ADC0        <= '1';
            RESET_ADC1        <= '1';
            R_STATE           <= R_WAIT_RESET_ADC;
          else
            -- Release reset_adc
            wait_timer_start  <= '1';
            R_STATE           <= R_WAIT_RESET_END;
          end if;

        when R_WAIT_RESET_END =>
          if (wait_timer_done = '0') then
            R_STATE           <= R_WAIT_RESET_END;
          else
            R_STATE           <= R_IDLE;
          end if;
      end case;
    end if;
  end if;
end process PROC_DDR_RESET_HANDLER;

-------------------------------------------------------------------------------
-- Outputs
-------------------------------------------------------------------------------
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_CLK_OUT    <= adc0_data_clk_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_CLK_OUT    <= adc1_data_clk_o;

  ADC0_LOCKED_OUT      <= adc0_locked_o;
  ADC1_LOCKED_OUT      <= adc1_locked_o;

  ADC0_ERROR_OUT        <= adc0_error_o;
  ADC1_ERROR_OUT        <= adc1_error_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 adc_ad9228_data_handler is
  generic (
    DEBUG_ENABLE : boolean := false
    );
  port (
    CLK_IN               : in  std_logic;
    RESET_IN             : in  std_logic;

    DDR_DATA_CLK         : in  std_logic;
    DDR_DATA_IN          : in  std_logic_vector(19 downto 0);

    DATA_A_OUT           : out std_logic_vector(11 downto 0);
    DATA_B_OUT           : out std_logic_vector(11 downto 0);
    DATA_C_OUT           : out std_logic_vector(11 downto 0);
    DATA_D_OUT           : out std_logic_vector(11 downto 0);
    DATA_CLK_OUT         : out std_logic;

    SLOPPY_FRAME_IN      : in  std_logic;
    FRAME_LOCKED_OUT     : out std_logic;
    ERROR_STATUS_OUT     : out std_logic_vector(2 downto 0);
    ERROR_STATUS_SL_OUT  : out std_logic_vector(2 downto 0);
                               -- 2: resync
                               -- 1: BITSHIFTED, fatal
                               -- 0: UNDEF, fatal
    DEBUG_OUT            : out std_logic_vector(15 downto 0)
    );
end adc_ad9228_data_handler;

architecture Behavioral of  adc_ad9228_data_handler is

  -- Frame Lock Handler
  type adc_data_s       is array(0 to 4) of std_logic_vector(13 downto 0);
  type adc_data_t       is array(0 to 3) of std_logic_vector(11 downto 0);

  type BYTE_STATUS is (B_UNDEF,
```



```
                       B_ALIGNED,
                       B_BYTESHIFTED,
                       B_BITSHIFTED
                       );
  signal adc_data_shift           : adc_data_s;

  signal adc_data_c_m             : adc_data_t;
  signal adc_data_clk_c_m         : std_logic;
  signal adc_byte_status_c        : BYTE_STATUS;
  signal adc_byte_status_last_c   : BYTE_STATUS;
  signal adc_frame_clk_ok_c       : std_logic;
  signal adc_frame_clk_ok_hist_c  : std_logic_vector(15 downto 0);
  signal adc_frame_locked_c       : std_logic;
  signal error_status_c           : std_logic_vector(2 downto 0);

  signal adc_data_sl_m            : adc_data_t;
  signal adc_data_clk_sl_m        : std_logic;
  signal adc_byte_status_sl       : BYTE_STATUS;
  signal adc_byte_status_last_sl  : BYTE_STATUS;
  signal adc_frame_clk_ok_sl      : std_logic;
  signal adc_frame_clk_ok_hist_sl : std_logic_vector(15 downto 0);
  signal adc_frame_locked_sl      : std_logic;
  signal error_status_sl          : std_logic_vector(2 downto 0);

  -- Sloppy Multiplexer
  signal adc_data_m               : adc_data_t;
  signal adc_data_clk_m           : std_logic;
  signal adc_byte_status          : BYTE_STATUS;
  signal adc_frame_clk_ok         : std_logic;
  signal adc_frame_locked         : std_logic;

  -- Clock Transfer
  signal adc_fifo_empty           : std_logic;
  signal adc_fifo_full            : std_logic;
  signal adc_write_enable         : std_logic;
  signal adc_read_enable          : std_logic;
  signal adc_read_enable_t        : std_logic;
  signal adc_read_enable_tt       : std_logic;
  signal adc_locked_ff            : std_logic;
  signal adc_locked_f             : std_logic;
  signal adc_locked_o             : std_logic;
  signal adc_error_status_c_ff    : std_logic_vector(2 downto 0);
  signal adc_error_status_c_f     : std_logic_vector(2 downto 0);
  signal adc_error_status_c_o     : std_logic_vector(2 downto 0);
  signal adc_error_status_sl_ff   : std_logic_vector(2 downto 0);
  signal adc_error_status_sl_f    : std_logic_vector(2 downto 0);
  signal adc_error_status_sl_o    : std_logic_vector(2 downto 0);

  signal adc_data                 : adc_data_t;

  -- Output
  signal adc_data_clk_o           : std_logic;
  signal adc_data_o               : adc_data_t;

  -- RESET Handler
  signal RESET_DDR_DATA_CLK_F     : std_logic;
  signal RESET_DDR_DATA_CLK       : std_logic;

  -- Attributes
  attribute syn_keep : boolean;
  attribute syn_keep of RESET_DDR_DATA_CLK_F       : signal is true;
  attribute syn_keep of RESET_DDR_DATA_CLK         : signal is true;
```

```
  attribute syn_keep of adc_locked_ff             : signal is true;
  attribute syn_keep of adc_locked_f              : signal is true;

  attribute syn_keep of adc_error_status_c_ff     : signal is true;
  attribute syn_keep of adc_error_status_c_f      : signal is true;
  attribute syn_keep of adc_error_status_sl_ff    : signal is true;
  attribute syn_keep of adc_error_status_sl_f     : signal is true;

  attribute syn_preserve : boolean;
  attribute syn_preserve of RESET_DDR_DATA_CLK_F  : signal is true;
  attribute syn_preserve of RESET_DDR_DATA_CLK    : signal is true;

  attribute syn_preserve of adc_locked_ff         : signal is true;
  attribute syn_preserve of adc_locked_f          : signal is true;

  attribute syn_preserve of adc_error_status_c_ff  : signal is true;
  attribute syn_preserve of adc_error_status_c_f   : signal is true;
  attribute syn_preserve of adc_error_status_sl_ff : signal is true;
  attribute syn_preserve of adc_error_status_sl_f  : signal is true;

begin

  ------------------------------------------------------------------------------
  RESET_DDR_DATA_CLK_F  <= RESET_IN             when rising_edge(DDR_DATA_CLK);
  RESET_DDR_DATA_CLK    <= RESET_DDR_DATA_CLK_F when rising_edge(DDR_DATA_CLK);

  ------------------------------------------------------------------------------
  -- Debug Handler
  ------------------------------------------------------------------------------

  DEBUG_OUT               <= (others => '0');
  --DEBUG_OUT(0)            <= CLK_IN;
  --DEBUG_OUT(1)            <= DDR_DATA_CLK;
  --DEBUG_OUT(2)            <= adc_write_enable;
  --DEBUG_OUT(3)            <= adc_fifo_full;
  --DEBUG_OUT(4)            <= adc_fifo_empty;
  --DEBUG_OUT(5)            <= adc_data_clk_m;
  --DEBUG_OUT(6)            <= adc_read_enable;
  --DEBUG_OUT(7)            <= adc_read_enable_t;
  --DEBUG_OUT(8)            <= adc_read_enable_tt;
  --DEBUG_OUT(9)            <= adc_data_clk_o;
  --DEBUG_OUT(10)           <= adc_error;
  --DEBUG_OUT(11)           <= adc_frame_locked;
  --DEBUG_OUT(12)           <= adc_frame_clk_ok;
  --DEBUG_OUT(14)           <= RESET_CLKDIV;
  --DEBUG_OUT(15)           <= RESET_ADC;

  ------------------------------------------------------------------------------
  -- Lock to ADC Frame Data
  ------------------------------------------------------------------------------

  PROC_LOCK_TO_ADC_FRAME: process(DDR_DATA_CLK)
  begin
    if (rising_edge(DDR_DATA_CLK)) then
      if (RESET_DDR_DATA_CLK = '1') then
        for I in 0 to 4 loop
          adc_data_shift(I)       <= (others => '0');
        end loop;

        for I in 0 to 3 loop
          adc_data_sl_m(I)        <= (others => '0');
```



```
          adc_data_c_m(I)         <= (others => '0');
        end loop;
        adc_data_clk_sl_m         <= '0';
        adc_data_clk_c_m          <= '0';

        adc_byte_status_c         <= B_UNDEF;
        adc_byte_status_last_c    <= B_UNDEF;
        adc_frame_clk_ok_c        <= '0';
        adc_frame_clk_ok_hist_c   <= (others => '0');
        adc_frame_locked_c        <= '0';
        error_status_c            <= (others => '0');

        adc_byte_status_sl        <= B_UNDEF;
        adc_byte_status_last_sl   <= B_UNDEF;
        adc_frame_clk_ok_sl       <= '0';
        adc_frame_clk_ok_hist_sl  <= (others => '0');
        adc_frame_locked_sl       <= '0';
        error_status_sl           <= (others => '0');
      else
        -- Store new incoming Data in Shift Registers
        for I in 0 to 4 loop
          adc_data_shift(I)(3)             <= DDR_DATA_IN(I + 0);
          adc_data_shift(I)(2)             <= DDR_DATA_IN(I + 5);
          adc_data_shift(I)(1)             <= DDR_DATA_IN(I + 10);
          adc_data_shift(I)(0)             <= DDR_DATA_IN(I + 15);
          adc_data_shift(I)(13 downto  4)  <= adc_data_shift(I)(9 downto 0);
        end loop;

        ------------------------------------------------------------------------
        -- Check Frame Lock and valid Status, Index 4 is THE Frame Clock
        ------------------------------------------------------------------------
        case adc_data_shift(4)(11 downto 0) is
          when "111111000000" =>
            -- Input Data is correct and new Frame is available
            for I in 0 to 3 loop
              adc_data_c_m(I)               <= adc_data_shift(I)(11 downto 0);
            end loop;
            adc_data_clk_c_m                <= '1';
            adc_frame_clk_ok_c              <= '1';
            adc_byte_status_c               <= B_ALIGNED;

          when "111100000011" =>
            -- Input Data is correct and new Frame is available,
            -- but byte shifted by one
            for I in 0 to 3 loop
              adc_data_c_m(I)               <= adc_data_shift(I)(13 downto 2);
            end loop;
            adc_data_clk_c_m                <= '1';
            adc_frame_clk_ok_c              <= '1';
            adc_byte_status_c               <= B_BYTESHIFTED;

          when "110000001111" | "000011111100"    =>
            -- Input Data is correct
            adc_data_clk_c_m                <= '0';
            adc_frame_clk_ok_c              <= '1';
            adc_byte_status_c               <= B_ALIGNED;

          when "000000111111" | "001111110000" =>
            -- Input Data is correct
            adc_data_clk_c_m                <= '0';
            adc_frame_clk_ok_c              <= '1';
            adc_byte_status_c               <= B_BYTESHIFTED;
```

```
          when "000001111110" |
               "000111111000" |
               "011111100000" |
               "111110000001" |
               "111000000111" |
               "100000011111" =>
            adc_data_clk_c_m               <= '0';
            adc_frame_clk_ok_c             <= '0';
            adc_byte_status_c              <= B_BITSHIFTED;

          when others =>
            -- Input Data is invalid, Fatal Error of DDR Data, needs reset.
            adc_data_clk_c_m               <= '0';
            adc_frame_clk_ok_c             <= '0';
            adc_byte_status_c              <= B_UNDEF;

        end case;

        -- Determin ADC Frame Lock Status
        adc_frame_clk_ok_hist_c(0)         <= adc_frame_clk_ok_c;
        adc_frame_clk_ok_hist_c(15 downto 1) <=
          adc_frame_clk_ok_hist_c(14 downto 0);

        if (adc_frame_clk_ok_hist_c = x"ffff") then
          adc_frame_locked_c               <= '1';
        else
          adc_frame_locked_c               <= '0';
        end if;

        -- Error Status
        adc_byte_status_last_c             <= adc_byte_status_c;
        if (adc_byte_status_c /= adc_byte_status_last_c) then
          error_status_c(2)                <= '1';
        else
          error_status_c(2)                <= '0';
        end if;

        if (adc_byte_status = B_BITSHIFTED) then
          error_status_c(1)                <= '1';
        else
          error_status_c(1)                <= '0';
        end if;

        if (adc_byte_status = B_UNDEF) then
          error_status_c(0)                <= '1';
        else
          error_status_c(0)                <= '0';
        end if;

        ------------------------------------------------------------------------
        -- Sloppy Frame Handler
        ------------------------------------------------------------------------
        if (adc_data_shift(4)(6 downto 5) =  "10") then
          -- Input Data is correct and new Frame is available
          for I in 0 to 3 loop
            adc_data_sl_m(I)               <= adc_data_shift(I)(11 downto 0);
          end loop;
          adc_data_clk_sl_m                <= '1';
          adc_frame_clk_ok_sl              <= '1';
          adc_byte_status_sl               <= B_ALIGNED;
```



```
        elsif (adc_data_shift(4)(8 downto 7) =  "10") then
          -- Input Data is correct and new Frame is available,
          -- but byte shifted by one
          for I in 0 to 3 loop
            adc_data_sl_m(I)               <= adc_data_shift(I)(13 downto 2);
          end loop;
          adc_data_clk_sl_m                <= '1';
          adc_frame_clk_ok_sl              <= '1';
          adc_byte_status_sl               <= B_BYTESHIFTED;

        elsif ((adc_data_shift(4)(10 downto 9) =  "10") or
               (adc_data_shift(4)(2 downto 1) =  "10")) then
          -- Input Data is correct
          adc_data_clk_sl_m                <= '0';
          adc_frame_clk_ok_sl              <= '1';
          adc_byte_status_sl               <= B_ALIGNED;

        elsif (((adc_data_shift(4)(11) = '0') and
                (adc_data_shift(4)(0) = '1')) or
               (adc_data_shift(4)(4 downto 2) =  "10")) then
          -- Input Data is correct
          adc_data_clk_sl_m                <= '0';
          adc_frame_clk_ok_sl              <= '1';
          adc_byte_status_sl               <= B_BYTESHIFTED;

        elsif ((adc_data_shift(4)( 1 downto  0) =  "10") or
               (adc_data_shift(4)( 3 downto  2) =  "10") or
               (adc_data_shift(4)( 5 downto  4) =  "10") or
               (adc_data_shift(4)( 7 downto  6) =  "10") or
               (adc_data_shift(4)( 9 downto  8) =  "10") or
               (adc_data_shift(4)(11 downto 10) =  "10")) then
          adc_data_clk_sl_m                <= '0';
          adc_frame_clk_ok_sl              <= '0';
          adc_byte_status_sl               <= B_BITSHIFTED;
        else
          -- Input Data is invalid, Fatal Error of DDR Data, needs reset.
          adc_data_clk_sl_m                <= '0';
          adc_frame_clk_ok_sl              <= '0';
          adc_byte_status_sl               <= B_UNDEF;
        end if;

        -- Determin ADC Frame Lock Status
        adc_frame_clk_ok_hist_sl(0)          <= adc_frame_clk_ok_sl;
        adc_frame_clk_ok_hist_sl(15 downto 1) <=
          adc_frame_clk_ok_hist_sl(14 downto 0);

        if (adc_frame_clk_ok_hist_sl = x"ffff") then
          adc_frame_locked_sl                <= '1';
        else
          adc_frame_locked_sl                <= '0';
        end if;

        -- Error Status
        adc_byte_status_last_sl              <= adc_byte_status_sl;
        if (adc_byte_status_sl /= adc_byte_status_last_sl) then
          error_status_sl(2)                 <= '1';
        else
          error_status_sl(2)                 <= '0';
        end if;

        if (adc_byte_status_sl = B_BITSHIFTED) then
          error_status_sl(1)                 <= '1';
```

```
      else
        error_status_sl(1)                  <= '0';
      end if;

      if (adc_byte_status = B_UNDEF) then
        error_status_sl(0)                  <= '1';
      else
        error_status_sl(0)                  <= '0';
      end if;

    end if;

  end if;
end process PROC_LOCK_TO_ADC_FRAME;

PROC_SLOPPY_MULTIPLEXER: process(SLOPPY_FRAME_IN)
begin
  if (SLOPPY_FRAME_IN = '0')  then
    adc_data_m            <= adc_data_c_m;
    adc_data_clk_m        <= adc_data_clk_c_m;
    adc_frame_clk_ok      <= adc_frame_clk_ok_c;
    adc_frame_locked      <= adc_frame_locked_c;
  else
    adc_data_m            <= adc_data_sl_m;
    adc_data_clk_m        <= adc_data_clk_sl_m;
    adc_frame_clk_ok      <= adc_frame_clk_ok_sl;
    adc_frame_locked      <= adc_frame_locked_sl;
  end if;
end process PROC_SLOPPY_MULTIPLEXER;

-------------------------------------------------------------------------------
-- Domain Tansfer of Data to CLK_IN
-------------------------------------------------------------------------------

fifo_adc_48to48_dc_1: entity work.fifo_adc_48to48_dc
  port map (
    Data(11 downto 0)  => adc_data_m(0),
    Data(23 downto 12) => adc_data_m(1),
    Data(35 downto 24) => adc_data_m(2),
    Data(47 downto 36) => adc_data_m(3),
    WrClock            => DDR_DATA_CLK,
    RdClock            => CLK_IN,
    WrEn               => adc_write_enable,
    RdEn               => adc_read_enable,
    Reset              => RESET_IN,
    RPReset            => RESET_IN,
    Q(11 downto 0)     => adc_data(0),
    Q(23 downto 12)    => adc_data(1),
    Q(35 downto 24)    => adc_data(2),
    Q(47 downto 36)    => adc_data(3),
    Empty              => adc_fifo_empty,
    Full               => adc_fifo_full
    );

-- Readout Handler
adc_write_enable    <= adc_data_clk_m and not adc_fifo_full;
adc_read_enable     <= not adc_fifo_empty;

PROC_ADC_FIFO_READ: process(CLK_IN)
begin
  if (rising_edge(CLK_IN)) then
    adc_read_enable_tt    <= adc_read_enable;
```



```
    if (RESET_IN = '1') then
      adc_read_enable_t   <= '0';
      for I in 0 to 3 loop
        adc_data_o(I)     <= (others => '0');
      end loop;
      adc_data_clk_o      <= '0';
    else
      -- Read enable
      adc_read_enable_t   <= adc_read_enable_tt;

      if (adc_read_enable_t = '1') then
        for I in 0 to 3 loop
          adc_data_o(I)   <= adc_data(I);
        end loop;
        adc_data_clk_o    <= '1';
      else
        adc_data_clk_o    <= '0';
      end if;
    end if;
  end if;
end process PROC_ADC_FIFO_READ;

-------------------------------------------------------------------------------
-- Domain Transfer of Control Signals
-------------------------------------------------------------------------------
adc_locked_ff          <= adc_frame_locked       when rising_edge(CLK_IN);
adc_locked_f           <= adc_locked_ff          when rising_edge(CLK_IN);
adc_locked_o           <= adc_locked_f           when rising_edge(CLK_IN);

adc_error_status_c_ff  <= error_status_c         when rising_edge(CLK_IN);
adc_error_status_c_f   <= adc_error_status_c_ff  when rising_edge(CLK_IN);
adc_error_status_c_o   <= adc_error_status_c_f   when rising_edge(CLK_IN);

adc_error_status_sl_ff <= error_status_sl        when rising_edge(CLK_IN);
adc_error_status_sl_f  <= adc_error_status_sl_ff when rising_edge(CLK_IN);
adc_error_status_sl_o  <= adc_error_status_sl_f  when rising_edge(CLK_IN);

-------------------------------------------------------------------------------
-- Output
-------------------------------------------------------------------------------
DATA_A_OUT           <= adc_data_o(0);
DATA_B_OUT           <= adc_data_o(1);
DATA_C_OUT           <= adc_data_o(2);
DATA_D_OUT           <= adc_data_o(3);
DATA_CLK_OUT         <= adc_data_clk_o;

FRAME_LOCKED_OUT     <= adc_locked_o;
ERROR_STATUS_OUT     <= adc_error_status_c_o;
ERROR_STATUS_SL_OUT  <= adc_error_status_sl_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          : 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
  ---------------------------------------------------------------------------
  -- 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;
  DEBUG_OUT(15 downto 12) <= (others => '0');

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

  -- 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
    );

-- 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

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

  -- Checkerboard
  signal checker_counter    : unsigned(15 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

  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
      debug_line_o           <= checker_counter;
    else
      debug_line_o           <= (others => '1');
    end if;
  end process PROC_MULTIPLEXER;

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

  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(8)              <= almost_empty_o;
  DEBUG_OUT(9)              <= Reset;
  DEBUG_OUT(15 downto 10)    <=
    std_logic_vector(write_ctr - read_ctr)(5 downto 0);

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

  ram_fifo_delay_256x44_1: entity work.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.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
    DATA_IN                 : in  std_logic_vector(43 downto 0);
    DATA_CLK_IN             : in  std_logic;

    DATA_OUT                : out std_logic_vector(43 downto 0);
    DATA_CLK_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
```

```
  -- Input FFs
  signal data_in_f             : std_logic_vector(43 downto 0);
  signal data_clk_in_f         : std_logic;

  -- FIFO Write Handler
  signal fifo_full             : std_logic;
  signal fifo_write_enable     : std_logic;
  signal fifo_reset            : std_logic;

  signal fifo_reset_p          : std_logic;
  signal fifo_reset_l          : std_logic;

  -- FIFO READ
  signal fifo_data_o           : 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 data_o                : std_logic_vector(43 downto 0);
  signal data_clk_o            : std_logic;

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

  -- Frame Rate Counter
  signal rate_timer            : unsigned(27 downto 0);
  signal frame_rate_in_ctr_t   : unsigned(27 downto 0);
  signal frame_rate_out_ctr_t  : unsigned(27 downto 0);
  signal frame_rate_input      : unsigned(27 downto 0);
  signal frame_rate_output     : unsigned(27 downto 0);

  -- Error Status
  signal data_clk_shift         : std_logic_vector(3 downto 0);
  signal frame_dt_error         : std_logic;
  signal frame_dt_error_ctr     : unsigned(15 downto 0);
  signal frame_rate_error       : std_logic;

  signal data_clk_out_shift     : std_logic_vector(3 downto 0);
  signal frame_dt_out_error     : std_logic;
  signal frame_dt_out_error_ctr : unsigned(15 downto 0);
  signal frame_rate_out_error   : std_logic;

  signal error_o                : 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 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)            <= DATA_CLK_IN;
      DEBUG_OUT(2)            <= fifo_reset;
      DEBUG_OUT(3)            <= fifo_full;
      DEBUG_OUT(4)            <= fifo_write_enable;
      DEBUG_OUT(5)            <= fifo_empty;
      DEBUG_OUT(6)            <= fifo_almost_empty;
      DEBUG_OUT(7)            <= fifo_read_enable;
      DEBUG_OUT(8)            <= fifo_read_enable_t;
      DEBUG_OUT(9)            <= fifo_read_enable_tt;
      DEBUG_OUT(10)           <= data_clk_o;
      DEBUG_OUT(15 downto 11) <= (others => '0');
    else
      DEBUG_OUT               <= debug_fifo;
    end if;
  end process PROC_DEBUG_MULTIPLEXER;

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

  data_in_f           <= DATA_IN when rising_edge(CLK_IN);
  data_clk_in_f       <= DATA_CLK_IN when rising_edge(CLK_IN);

  fifo_44_data_delay_my_1: fifo_44_data_delay_my
    port map (
      Data          => data_in_f,
      Clock         => CLK_IN,
      WrEn          => fifo_write_enable,
      RdEn          => fifo_read_enable,
      Reset         => fifo_reset,
      AmEmptyThresh => fifo_delay,
      Q             => fifo_data_o,
      Empty         => fifo_empty,
      Full          => fifo_full,
      AlmostEmpty   => fifo_almost_empty,
      DEBUG_OUT     => debug_fifo
      );

  fifo_read_enable             <= not fifo_almost_empty;
  fifo_reset                   <= RESET_IN or fifo_reset_l;
  fifo_write_enable            <= data_clk_in_f and not fifo_full;

  fifo_reset_p  <= fifo_reset_r or fifo_delay_reset;
  pulse_to_level_FIFO_RESET: pulse_to_level
    generic map (
      NUM_CYCLES => 3
      )
    port map (
      CLK_IN    => CLK_IN,
      RESET_IN  => RESET_IN,
      PULSE_IN  => fifo_reset_p,
      LEVEL_OUT => fifo_reset_l
      );

  -- 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';

      data_o               <= (others => '0');
      data_clk_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
        data_o             <= fifo_data_o;
        data_clk_o         <= '1';
      else
        data_o             <= x"fff_ffff_ffff";
        data_clk_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)) 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;

PROC_CAL_RATES: process (CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if (RESET_IN = '1') then
      rate_timer           <= (others => '0');
      frame_rate_input     <= (others => '0');
      frame_rate_output    <= (others => '0');
      frame_rate_in_ctr_t  <= (others => '0');
      frame_rate_out_ctr_t <= (others => '0');
    else
      if (rate_timer < x"5f5e100") then
        if (DATA_CLK_IN = '1') then
          frame_rate_in_ctr_t             <= frame_rate_in_ctr_t + 1;
        end if;
        if (data_clk_o = '1') then
          frame_rate_out_ctr_t            <= frame_rate_out_ctr_t + 1;
        end if;
        rate_timer                        <= rate_timer + 1;
      else
        frame_rate_input                  <= frame_rate_in_ctr_t;
        frame_rate_in_ctr_t(27 downto 1)  <= (others => '0');
        frame_rate_in_ctr_t(0)            <= DATA_CLK_IN;
```



```
        frame_rate_output                 <= frame_rate_out_ctr_t;
        frame_rate_out_ctr_t(27 downto 1) <= (others => '0');
        frame_rate_out_ctr_t(0)           <= data_clk_o;

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

PROC_DATA_STREAM_DELTA_T: process(CLK_IN)
begin
  if (rising_edge(CLK_IN)) then
    if (RESET_IN = '1') then
      data_clk_shift          <= (others => '0');
      frame_dt_error_ctr      <= (others => '0');
      frame_dt_error          <= '0';
      data_clk_out_shift      <= (others => '0');
      frame_dt_out_error_ctr  <= (others => '0');
      frame_dt_out_error      <= '0';
    else
      -- Frame
      data_clk_shift(0)               <= DATA_CLK_IN;
      data_clk_shift(3 downto 1)      <= data_clk_shift(2 downto 0);

      data_clk_out_shift(0)           <= data_clk_o;
      data_clk_out_shift(3 downto 1)  <= data_clk_out_shift(2 downto 0);

      case data_clk_shift is
        when "1100" | "1110" | "1111" | "0000" =>
          frame_dt_error_ctr          <= frame_dt_error_ctr + 1;
          frame_dt_error              <= '1';

        when others =>
          frame_dt_error              <= '0';

      end case;

      case data_clk_out_shift is
        when "1100" | "1110" | "1111" | "0000" =>
          frame_dt_out_error_ctr      <= frame_dt_out_error_ctr + 1;
          frame_dt_out_error          <= '1';

        when others =>
          frame_dt_out_error          <= '0';

      end case;

    end if;
  end if;
end process PROC_DATA_STREAM_DELTA_T;

------------------------------------------------------------------------------
-- 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';
    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(27 downto 0)  <= frame_rate_input;
            slv_data_o(31 downto 28) <= (others => '0');
            slv_ack_o                <= '1';

          when x"0002" =>
            slv_data_o(27 downto 0)  <= frame_rate_output;
            slv_data_o(31 downto 28) <= (others => '0');
            slv_ack_o                <= '1';

          when x"0003" =>
            slv_data_o(15 downto 0)  <= frame_dt_error_ctr;
            slv_data_o(31 downto 16) <= (others => '0');
            slv_ack_o                <= '1';

          when x"0004" =>
            slv_data_o(15 downto 0)  <= frame_dt_out_error_ctr;
            slv_data_o(31 downto 16) <= (others => '0');
            slv_ack_o                <= '1';

          when x"0005" =>
            slv_data_o(0)            <= debug_r;
            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"0005" =>
            debug_r                  <= 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
  DATA_OUT               <= data_o;
  DATA_CLK_OUT           <= data_clk_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
  generic (
    DEBUG_ENABLE : boolean := false
    );
  port(
    CLK_IN                : in  std_logic;
    RESET_IN              : in  std_logic;
    TRIGGER_IN            : in  std_logic;
    NX_ONLINE_IN          : in  std_logic;
    NX_CLOCK_ON_IN        : in  std_logic;

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

    -- 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
    DATA_OUT              : out std_logic_vector(43 downto 0);
    DATA_CLK_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;

    ADC_TR_ERROR_IN        : in  std_logic;
    DISABLE_ADC_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

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

  -- NX_TIMESTAMP_IN Process
  signal nx_timestamp_delay_f        : unsigned(2 downto 0);
  signal nx_timestamp_delay          : unsigned(2 downto 0);
  signal nx_shift_register_delay     : std_logic_vector(5 downto 0);
  signal nx_frame_word_ff            : std_logic_vector(7 downto 0);
  signal nx_frame_word_f             : std_logic_vector(7 downto 0);

  signal nx_frame_word_t             : std_logic_vector(31 downto 0);
  signal nx_frame_clk_t              : std_logic;

  -- Frame Sync Process
  signal frame_byte_pos              : unsigned(1 downto 0);
  signal nx_frame_word               : std_logic_vector(31 downto 0);
  signal nx_frame_clk                : std_logic;
  signal nx_frame_clk_c100           : std_logic;

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

  -- NX Frame Delay
  signal nx_frame_word_s             : std_logic_vector(31 downto 0);
  signal nx_frame_clk_s              : std_logic;

  -- Clock Domain Transfer NX Data
  signal fifo_nx_reset_i             : std_logic;
  signal fifo_nx_write_enable        : std_logic;
  signal fifo_nx_read_enable         : std_logic;
  signal fifo_nx_empty               : std_logic;
  signal fifo_nx_full                : std_logic;
  signal fifo_nx_data                : std_logic_vector(31 downto 0);
  signal fifo_nx_data_clk_tt         : std_logic;
  signal fifo_nx_data_clk_t          : std_logic;
  signal fifo_nx_data_clk            : std_logic;
  signal nx_data                     : std_logic_vector(31 downto 0);
  signal nx_data_clk                 : std_logic;

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

  -- ADC Ckl Generator
  signal adc_sclk_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_sclk_ok                 : std_logic;
  signal adc_sclk_ok_c100            : std_logic;

  signal pll_adc_sampling_clk_o      : std_logic;
  signal pll_adc_sampling_clk_lock   : std_logic;

  -- PLL ADC Monitor
  signal pll_adc_not_lock            : std_logic;
  signal pll_adc_not_lock_shift      : 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_sclk_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);

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

  -- ADC Handler
  signal ADC_RESET_AD9228            : std_logic;
  signal adc_data                    : std_logic_vector(11 downto 0);
  signal adc_data_clk                : std_logic;
  signal adc_locked                  : std_logic;
  signal adc_notlocked_p             : std_logic;

  signal adc_data_s                  : std_logic_vector(11 downto 0);
  signal adc_data_s_clk              : std_logic;
  signal adc_notlock_ctr             : unsigned(11 downto 0);
  signal adc_sloppy_frame            : std_logic;
  signal ADC_DEBUG                   : std_logic_vector(15 downto 0);

  signal adc_error                   : std_logic;
  signal adc_error_p                 : std_logic;

  -- Merge Data Streams
  signal merge_handler_reset         : std_logic;
  signal merge_status                : std_logic_vector(1 downto 0);
  signal merge_status_error          : std_logic;
  signal merge_timing_ctr            : unsigned(2 downto 0);
  signal merge_timing_error          : std_logic;

  signal merge_error_ctr             : unsigned(11 downto 0);

  signal data_m                      : std_logic_vector(43 downto 0);
  signal data_clk_m                  : std_logic;

  -- Data Output Handler
  signal data_o                      : std_logic_vector(43 downto 0);
  signal data_clk_o                  : std_logic;

  -- ADC Sampling Clock Phase adjustment
  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_err_rate_ctr         : unsigned(27 downto 0);
  signal parity_err_rate             : unsigned(27 downto 0);
  signal rate_timer_ctr              : unsigned(27 downto 0);

  -- Error
  signal error_o                     : std_logic;
  signal error_status_bits           : std_logic_vector(15 downto 0);
  signal adc_notlock_counter         : unsigned(27 downto 0);
  signal adc_error_counter           : unsigned(27 downto 0);
  signal nx_online                   : std_logic;
  signal nx_online_shift             : std_logic_vector(1 downto 0);
  signal reset_after_offline         : std_logic;

  -- Rate Errors
  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;

  -- Events per Second Errors
  signal adc_dt_error_cur            : std_logic;
  signal adc_dt_error                : std_logic;
  signal timestamp_dt_error_cur      : std_logic;
  signal timestamp_dt_error          : std_logic;

  -- Data Stream DeltaT Error Counters
  signal adc_dt_shift_reg            : std_logic_vector(3 downto 0);
  signal timestamp_dt_shift_reg      : std_logic_vector(3 downto 0);
  signal adc_dt_error_ctr            : unsigned(11 downto 0);
  signal timestamp_dt_error_ctr      : unsigned(11 downto 0);

  signal adc_dt_error_p              : std_logic;
  signal adc_dt_error_c100           : std_logic;
  signal timestamp_dt_error_p        : std_logic;
  signal timestamp_dt_error_c100     : std_logic;

  ------------------------------------------------------------------------------
  -- CLK Domain Transfer
  ------------------------------------------------------------------------------

  -- 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_mode                    : std_logic_vector(2 downto 0);
  signal reset_handler_start_r         : std_logic;
  signal johnson_counter_sync_r        : unsigned(1 downto 0);
  signal pll_adc_sample_clk_dphase_r   : unsigned(3 downto 0);
  signal pll_adc_sample_clk_finedelb_r : unsigned(3 downto 0);
  signal nx_timestamp_delay_adjust     : std_logic;
  signal nx_timestamp_delay_r          : unsigned(2 downto 0);
```



```
  signal nx_timestamp_delay_a          : unsigned(2 downto 0);
  signal nx_timestamp_delay_s          : unsigned(2 downto 0);
  signal nx_timestamp_delay_actr       : unsigned(15 downto 0);
  signal nx_frame_word_delay_rr        : unsigned(1 downto 0);
  signal nx_frame_word_delay_r         : unsigned(1 downto 0);
  signal adc_dt_error_ctr_r            : unsigned(11 downto 0);
  signal timestamp_dt_error_ctr_r      : unsigned(11 downto 0);
  signal adc_notlock_ctr_r             : unsigned(11 downto 0);
  signal merge_error_ctr_r             : unsigned(11 downto 0);
  signal nx_frame_synced_rr            : std_logic;
  signal nx_frame_synced_r             : std_logic;
  signal disable_adc_r                 : std_logic;
  signal adc_debug_type_r              : std_logic_vector(3 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;
  signal nx_timestamp_reset_o        : std_logic;
  signal nx_fifo_reset_handler       : std_logic;

  signal reset_handler_trigger       : std_logic_vector(15 downto 0);

  type R_STATES is (R_IDLE,
                    R_START,
                    R_WAIT_0,
                    R_RESET_TIMESTAMP,
                    R_WAIT_1,
                    R_SET_ALL_RESETS,
                    R_WAIT_2,
                    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 pll_adc_sampling_clk_reset  : std_logic;
  signal adc_reset_handler           : 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;

  -- 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_b              : std_logic;
  signal parity_error_c100           : std_logic;
  signal parity_error_counter        : unsigned(11 downto 0);
```

```
  -- Reset Domain Transfers
  signal reset_nx_timestamp_clk_in_ff : std_logic;
  signal reset_nx_timestamp_clk_in_f  : std_logic;
  signal RESET_NX_TIMESTAMP_CLK_IN    : std_logic;

  signal reset_nx_data_clk_in_ff      : std_logic;
  signal reset_nx_data_clk_in_f       : std_logic;

  signal debug_state                  : std_logic_vector(3 downto 0);

  -- Keep FlipFlops, do not change to shift registers -----------

  attribute syn_keep : boolean;

  attribute syn_keep of nx_frame_word_f                   : signal is true;

  attribute syn_keep of reset_nx_timestamp_clk_in_ff      : signal is true;
  attribute syn_keep of reset_nx_timestamp_clk_in_f       : signal is true;

  attribute syn_keep of reset_nx_data_clk_in_ff           : signal is true;
  attribute syn_keep of reset_nx_data_clk_in_f            : signal is true;

  attribute syn_keep of nx_timestamp_delay_f              : signal is true;
  attribute syn_keep of nx_timestamp_delay                : signal is true;

  attribute syn_preserve : boolean;

  attribute syn_preserve of nx_frame_word_f               : signal is true;

  attribute syn_preserve of reset_nx_timestamp_clk_in_ff  : signal is true;
  attribute syn_preserve of reset_nx_timestamp_clk_in_f   : signal is true;

  attribute syn_preserve of reset_nx_data_clk_in_ff       : signal is true;
  attribute syn_preserve of reset_nx_data_clk_in_f        : signal is true;

  attribute syn_preserve of nx_timestamp_delay_f          : signal is true;
  attribute syn_preserve of nx_timestamp_delay            : signal is true;

begin

  DFALSE: if (DEBUG_ENABLE = false) generate
    DEBUG_OUT               <= (others => '0');
  end generate DFALSE;

  DTRUE: if (DEBUG_ENABLE = true) generate
    PROC_DEBUG_MULT: process(debug_mode,
                             adc_data,
                             adc_data_clk,
                             adc_sclk_ok,
                             adc_sclk_skip,
                             adc_reset_sync,
                             adc_reset_sync_s,
                             ADC_RESET_AD9228,
                             nx_frame_clk,
                             adc_reset_ctr,
                             adc_data_s_clk,
                             data_clk_o,
                             nx_frame_synced,
                             rs_sync_reset
                             )
    begin
```



```
      case debug_mode is
        when "001" =>
          -- Reset Handler
          DEBUG_OUT(0)            <= CLK_IN;
          DEBUG_OUT(1)            <= nx_frame_clk_c100;
          DEBUG_OUT(2)            <= adc_data_clk;
          DEBUG_OUT(3)            <= adc_sclk_ok;
          DEBUG_OUT(4)            <= adc_reset_sync;
          DEBUG_OUT(5)            <= adc_reset_handler;
          DEBUG_OUT(6)            <= nx_online;
          DEBUG_OUT(7)            <= pll_adc_not_lock;
          DEBUG_OUT(8)            <= reset_after_offline;
          DEBUG_OUT(9)            <= nx_fifo_reset_handler;
          DEBUG_OUT(10)           <= reset_handler_busy;
          DEBUG_OUT(11)           <= pll_adc_sampling_clk_reset;
          DEBUG_OUT(15 downto 12) <= debug_state;

        when "010" =>
          -- AD9228 Handler Debug output
          DEBUG_OUT               <= ADC_DEBUG;

        when "011" =>
          -- 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)            <= '0';
          DEBUG_OUT(6)            <= nx_frame_rate_error;
          DEBUG_OUT(7)            <= pll_adc_not_lock;
          DEBUG_OUT(8)            <= '0';
          DEBUG_OUT(9)            <= adc_frame_rate_error;
          DEBUG_OUT(10)           <= nx_fifo_reset_handler;
          DEBUG_OUT(11)           <= pll_adc_sampling_clk_reset;
          DEBUG_OUT(12)           <= adc_reset_handler;
          DEBUG_OUT(13)           <= output_handler_reset;
          DEBUG_OUT(14)           <= frame_rate_error;
          DEBUG_OUT(15)           <= reset_timeout_flag;

        when "100" =>
          -- AD9228 Handler Debug output
          DEBUG_OUT(0)            <= CLK_IN;
          DEBUG_OUT(1)            <= '0';
          DEBUG_OUT(2)            <= nx_frame_clk;
          DEBUG_OUT(3)            <= nx_data_clk;
          DEBUG_OUT(4)            <= adc_data_clk;
          DEBUG_OUT(5)            <= '0';
          DEBUG_OUT(6)            <= adc_dt_error_p;
          DEBUG_OUT(9 downto 7)   <= merge_timing_ctr;
          DEBUG_OUT(10)           <= timestamp_dt_error_p;
          DEBUG_OUT(11)           <= merge_status_error;
          DEBUG_OUT(12)           <= merge_timing_error;
          DEBUG_OUT(14 downto 13) <= merge_status;
          DEBUG_OUT(15)           <= data_clk_o;

        when "101" =>
          -- AD9228 Handler Debug output
          DEBUG_OUT(0)            <= CLK_IN;
          DEBUG_OUT(1)            <= '0';
          DEBUG_OUT(2)            <= nx_frame_clk;
          DEBUG_OUT(3)            <= '0';
          DEBUG_OUT(4)            <= '0';
          DEBUG_OUT(5)            <= merge_timing_error;
```

```
          DEBUG_OUT(6)            <= '0';
          DEBUG_OUT(7)            <= '0';
          DEBUG_OUT(9 downto 8)   <= (others => '0');
          DEBUG_OUT(10)           <= '0';
          DEBUG_OUT(11)           <= '0';
          DEBUG_OUT(15 downto 12) <= (others => '0');

        when "110" =>
          DEBUG_OUT(0)            <= nx_frame_clk_s;  --data_clk_o;
          DEBUG_OUT(15 downto 1)  <= nx_frame_word_s(14 downto 0); --data_m(14 d
ownto 0);

        when "111" =>
          DEBUG_OUT(0)            <= nx_data_clk;
          DEBUG_OUT(15 downto 1)  <= nx_data(14 downto 0);

        when others =>
          -- Default
          DEBUG_OUT(0)            <= CLK_IN;
          DEBUG_OUT(1)            <= TRIGGER_IN;
          DEBUG_OUT(2)            <= data_clk_o;
          DEBUG_OUT(3)            <= nx_fifo_reset_handler;
          DEBUG_OUT(4)            <= '0';
          DEBUG_OUT(5)            <= '0';
          DEBUG_OUT(6)            <= '0';
          DEBUG_OUT(7)            <= '0';
          DEBUG_OUT(8)            <= '0';
          DEBUG_OUT(9)            <= nx_frame_clk;
          DEBUG_OUT(10)           <= '0';
          DEBUG_OUT(11)           <= adc_data_s_clk;
          DEBUG_OUT(12)           <= data_clk_o;
          DEBUG_OUT(13)           <= parity_error_c100;
          DEBUG_OUT(14)           <= merge_timing_error;
          DEBUG_OUT(15)           <= nx_frame_synced;

      end case;

    end process PROC_DEBUG_MULT;

  end generate DTRUE;

  ------------------------------------------------------------------------------
  -- Reset Domain Transfer
  ------------------------------------------------------------------------------
  reset_nx_timestamp_clk_in_ff  <= RESET_IN
                                   when rising_edge(NX_TIMESTAMP_CLK_IN);
  reset_nx_timestamp_clk_in_f   <= reset_nx_timestamp_clk_in_ff
                                   when rising_edge(NX_TIMESTAMP_CLK_IN);
  RESET_NX_TIMESTAMP_CLK_IN     <= reset_nx_timestamp_clk_in_f
                                   when rising_edge(NX_TIMESTAMP_CLK_IN);

  ------------------------------------------------------------------------------
  -- PLL Handler
  ------------------------------------------------------------------------------

  PROC_PLL_PHASE_SETUP: process(CLK_IN)
  begin
    if (rising_edge(CLK_IN)) then
      -- Shift dphase to show 0 as optimal value in standard setup
      pll_adc_sample_clk_dphase     <=
        std_logic_vector(13 + pll_adc_sample_clk_dphase_r);
      pll_adc_sample_clk_finedelb   <=
```



```
        std_logic_vector(8 + pll_adc_sample_clk_finedelb_r);
    end if;
  end process  PROC_PLL_PHASE_SETUP;

  pll_adc_sampling_clk_2: entity work.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
      );

  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_shift  <= (others => '0');
        pll_adc_not_lock_ctr    <= (others => '0');
      else
        pll_adc_not_lock_shift(0)  <= pll_adc_not_lock;
        pll_adc_not_lock_shift(1)  <= pll_adc_not_lock_shift(0);

        if (pll_adc_not_lock_shift = "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;


  timer_static_RESET_TIMER: timer_static
    generic map (
      CTR_WIDTH => 20,
      CTR_END   => 500000 -- 5ms
      )
    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 => 32,
      CTR_END   => 2000000000 -- 10s
      )
    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
```

```
    );


-------------------------------------------------------------------------------
-- 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 (adc_sclk_skip = '0') then
      johnson_ff_0     <= not johnson_ff_1;
      johnson_ff_1     <= johnson_ff_0;
    end if;
    adc_sampling_clk   <= not johnson_ff_1;
  end if;
end process PROC_ADC_SAMPLING_CLK_GENERATOR;

-- Adjust johnson_counter_sync to show optimal value at 0

PROC_ADC_SAMPLING_CLK_SYNC: process(NX_TIMESTAMP_CLK_IN)
  variable adc_sclk_state : std_logic_vector(1 downto 0);
begin
  if (rising_edge(NX_TIMESTAMP_CLK_IN)) then
    if (RESET_NX_TIMESTAMP_CLK_IN = '1') then
      johnson_counter_sync  <= (others => '0');
      adc_sclk_skip         <= '0';
      adc_sclk_ok           <= '0';
    else
      johnson_counter_sync <= std_logic_vector(johnson_counter_sync_r);
      adc_sclk_state        := johnson_ff_1 & johnson_ff_0;
      adc_sclk_skip         <= '0';
      if (nx_frame_clk = '1') then
        if (adc_sclk_state /= johnson_counter_sync) then
          adc_sclk_skip     <= '1';
          adc_sclk_ok       <= '0';
        else
          adc_sclk_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_sclk_ok_last    <= '0';
      adc_reset_sync_s    <= '0';
    else
      adc_reset_sync_s    <= '0';
      adc_sclk_ok_last    <= adc_sclk_ok;
      if (adc_sclk_ok_last = '0' and adc_sclk_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 Timestamp Handler
-------------------------------------------------------------------------------

-- First: Use three Input FIFO to relax Timing
nx_frame_word_ff  <= NX_TIMESTAMP_IN   when rising_edge(NX_TIMESTAMP_CLK_IN);
nx_frame_word_f   <= nx_frame_word_ff when rising_edge(NX_TIMESTAMP_CLK_IN);

-- Second: 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
      nx_frame_word_t     <= (others => '0');
      nx_frame_word       <= (others => '0');
      nx_frame_clk_t      <= '0';
      nx_frame_clk        <= '0';
    else
      case frame_byte_pos is
        when "11" => nx_frame_word_t(31 downto 24) <= nx_frame_word_f;
                     nx_frame_clk_t                <= '0';

        when "10" => nx_frame_word_t(23 downto 16) <= nx_frame_word_f;
                     nx_frame_clk_t                <= '0';

        when "01" => nx_frame_word_t(15 downto  8) <= nx_frame_word_f;
                     nx_frame_clk_t                <= '0';

        when "00" => nx_frame_word_t( 7 downto  0) <= nx_frame_word_f;
                     nx_frame_clk_t                <= '1';
      end case;

      -- Output Frame
      if (nx_frame_clk_t = '1') then
        nx_frame_word                              <= nx_frame_word_t;
        nx_frame_clk                               <= '1';
      else
        nx_frame_word                              <= x"0000_0001";
        nx_frame_clk                               <= '0';
      end if;
    end if;
  end if;
end process PROC_8_TO_32_BIT;

-- TS 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_frame_clk_t = '1') then
        case nx_frame_word_t 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') then
      nx_frame_synced     <= '0';
    else
      if (rs_sync_reset = '1') then
        nx_frame_synced   <= '0';
      elsif (rs_sync_set = '1') then
        nx_frame_synced   <= '1';
      end if;
    end if;
  end if;
end process PROC_RS_FRAME_SYNCED;

-- Check Parity Bit
PROC_PARITY_CHECKER: 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_b       <= '0';
    else
      if (nx_frame_clk = '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_b   <= '1';
        else
          parity_error_b   <= '0';
        end if;
      else
        parity_error_b     <= '0';
      end if;
    end if;
  end if;
end process PROC_PARITY_CHECKER;

-- Delay NX Data relative to ADC Data
dynamic_shift_register33x64_1: entity work.dynamic_shift_register33x64
  port map (
    Din(31 downto 0) => nx_frame_word,
    Din(32)          => nx_frame_clk,
    Addr             => nx_shift_register_delay,
    Clock            => NX_TIMESTAMP_CLK_IN,
    ClockEn          => '1',
    Reset            => RESET_NX_TIMESTAMP_CLK_IN,
    Q(31 downto 0)   => nx_frame_word_s,
    Q(32)            => nx_frame_clk_s
    );

-- Timestamp Input Delay relative to ADC
nx_timestamp_delay_f  <= nx_timestamp_delay_s
                         when rising_edge(NX_TIMESTAMP_CLK_IN);
nx_timestamp_delay    <= nx_timestamp_delay_f
                         when rising_edge(NX_TIMESTAMP_CLK_IN);
PROC_NX_SHIFT_REGISTER_DELAY: process(NX_TIMESTAMP_CLK_IN)
begin
  if (rising_edge(NX_TIMESTAMP_CLK_IN)) then
    if(RESET_NX_TIMESTAMP_CLK_IN = '1') then
      nx_shift_register_delay       <= (others => '0');
    else
      case nx_timestamp_delay is
        when "000" =>
          nx_shift_register_delay   <= "001111"; -- 15

        when "001" =>
          nx_shift_register_delay   <= "010011"; -- 19

        when "010" =>
          nx_shift_register_delay   <= "010111"; -- 23

        when "011" =>
          nx_shift_register_delay   <= "011011"; -- 27

        when "100" =>
          nx_shift_register_delay   <= "011111"; -- 31

        when "101" =>
          nx_shift_register_delay   <= "100011"; -- 35

        when "110" =>
          nx_shift_register_delay   <= "100111"; -- 39
```

```
        when "111" =>
          nx_shift_register_delay   <= "101011"; -- 43

      end case;
    end if;
  end if;
end process PROC_NX_SHIFT_REGISTER_DELAY;

--------------------------------------------------------------------------------
-- Clock Domain Transfer Nxyter Data Stream
--------------------------------------------------------------------------------

fifo_nxyter_32to32_dc_1: entity work.fifo_nxyter_32to32_dc
  port map (
    Data    => nx_frame_word_s,
    WrClock => NX_TIMESTAMP_CLK_IN,
    RdClock => CLK_IN,
    WrEn    => fifo_nx_write_enable,
    RdEn    => fifo_nx_read_enable,
    Reset   => fifo_nx_reset_i,
    RPReset => fifo_nx_reset_i,
    Q       => fifo_nx_data,
    Empty   => fifo_nx_empty,
    Full    => fifo_nx_full
    );

fifo_nx_reset_i         <= RESET_IN or nx_fifo_reset_handler;
fifo_nx_write_enable    <= not fifo_nx_full and nx_frame_clk_s;
fifo_nx_read_enable     <= not fifo_nx_empty;

PROC_NX_FIFO_READ_ENABLE: process(CLK_IN)
begin
  if (rising_edge(CLK_IN)) then
    fifo_nx_data_clk_tt    <= fifo_nx_read_enable;
    if(RESET_IN = '1') then
      fifo_nx_data_clk_t    <= '0';
      fifo_nx_data_clk      <= '0';
    else
      -- Delay read signal by two Clock Cycles
      fifo_nx_data_clk_t    <= fifo_nx_data_clk_tt;
      fifo_nx_data_clk      <= fifo_nx_data_clk_t;
    end if;
  end if;
end process PROC_NX_FIFO_READ_ENABLE;

PROC_NX_FIFO_READ_HANDLER: process(CLK_IN)
begin
  if (rising_edge(CLK_IN)) then
    if(RESET_IN = '1') then
      nx_data         <= (others => '0');
      nx_data_clk     <= '0';
    else
      if (fifo_nx_data_clk = '1') then
       nx_data        <= fifo_nx_data;
       nx_data_clk    <= '1';
      else
       nx_data        <= (others => '0');
       nx_data_clk    <= '0';
      end if;
    end if;
  end if;
end process PROC_NX_FIFO_READ_HANDLER;
```



```
--------------------------------------------------------------------------------
-- ADC Input Handler
--------------------------------------------------------------------------------

ADC_RESET_AD9228         <= RESET_IN or adc_reset_handler;

adc_ad9228_1: adc_ad9228
  generic map (
    DEBUG_ENABLE => false
    )
  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       => open,
    ADC0_DATA_C_OUT       => open,
    ADC0_DATA_D_OUT       => open,
    ADC0_DATA_CLK_OUT     => adc_data_clk,

    ADC1_DATA_A_OUT       => open,
    ADC1_DATA_B_OUT       => open,
    ADC1_DATA_C_OUT       => open,
    ADC1_DATA_D_OUT       => open,
    ADC1_DATA_CLK_OUT     => open,

    ADC0_LOCKED_OUT       => adc_locked,
    ADC1_LOCKED_OUT       => open,

    ADC0_SLOPPY_FRAME     => adc_sloppy_frame,
    ADC1_SLOPPY_FRAME     => '0',

    ADC0_ERROR_OUT        => adc_error,
    ADC1_ERROR_OUT        => open,

    DEBUG_IN              => adc_debug_type_r,
    DEBUG_OUT             => ADC_DEBUG
    );

--------------------------------------------------------------------------------
-- Merge Data Streams Timestamps and ADC Value
--------------------------------------------------------------------------------
```

```
merge_handler_reset        <= merge_status_error or
                              merge_timing_error or
                              output_handler_reset;

PROC_DATA_MERGE_HANDLER: process(CLK_IN)
  variable status           : std_logic_vector(3 downto 0);
begin
  if (rising_edge(CLK_IN)) then
    if (RESET_IN = '1' or merge_handler_reset = '1') then
      merge_status         <= (others => '0');
      data_m               <= (others => '0');
      data_clk_m           <= '0';

      merge_status_error   <= '0';
      merge_timing_ctr     <= (others => '0');
      merge_timing_error   <= '0';
    else
      merge_status_error   <= '0';
      merge_timing_error   <= '0';
      data_clk_m           <= '0';

      if (disable_adc_r = '0') then
        status               := adc_data_clk & nx_data_clk & merge_status;
      else
        status               := '0' & nx_data_clk & '1' & merge_status(0);
      end if;

      case status is
        when "0100" =>
          data_m(31 downto 0)  <= nx_data;
          data_m(43 downto 32) <= (others => '0');
          merge_status         <= "01";

        when "0110" =>
          data_m(31 downto 0)  <= nx_data;
          merge_status         <= "00";
          data_clk_m           <= '1';

        when "1000" =>
          data_m(31 downto 0)  <= (others => '0');
          data_m(43 downto 32) <= adc_data;
          merge_status         <= "10";

        when "1001" =>
          data_m(43 downto 32) <= adc_data;
          data_clk_m           <= '1';
          merge_status         <= "00";

        when "1100" =>
          data_m(31 downto 0)  <= nx_data;
          data_m(43 downto 32) <= adc_data;
          data_clk_m           <= '1';
          merge_status         <= "00";

        when "0000" |
          "0001" |
          "0010" =>
          null;

        when others =>
          data_m               <= (others => '0');
```



```
          merge_status         <= (others => '0');
          merge_status_error   <= '1';

      end case;

      -- Check Timing
      if (data_clk_m = '1') then
        if (merge_timing_ctr < x"2" or merge_timing_ctr > x"3") then
          merge_timing_error   <= '1';
        else
          merge_timing_ctr     <= (others => '0');
        end if;
      else
        if (merge_timing_ctr >= x"3") then
          merge_timing_error   <= '1';
        end if;
        merge_timing_ctr       <= merge_timing_ctr + 1;
      end if;

    end if;
  end if;
end process PROC_DATA_MERGE_HANDLER;

-------------------------------------------------------------------------------
-- Signal Domain Transfers
-------------------------------------------------------------------------------
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
    );

pulse_dtrans_nx_frame_clk: pulse_dtrans
  generic map (
    CLK_RATIO => 2
    )
  port map (
    CLK_A_IN    => NX_TIMESTAMP_CLK_IN,
    RESET_A_IN  => RESET_NX_TIMESTAMP_CLK_IN,
    PULSE_A_IN  => nx_frame_clk,
    CLK_B_IN    => CLK_IN,
    RESET_B_IN  => RESET_IN,
    PULSE_B_OUT => nx_frame_clk_c100
    );

pulse_dtrans_parity_error: pulse_dtrans
  generic map (
    CLK_RATIO => 2
    )
  port map (
    CLK_A_IN    => NX_TIMESTAMP_CLK_IN,
    RESET_A_IN  => RESET_NX_TIMESTAMP_CLK_IN,
    PULSE_A_IN  => parity_error_b,
    CLK_B_IN    => CLK_IN,
    RESET_B_IN  => RESET_IN,
    PULSE_B_OUT => parity_error_c100
    );
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
      );

  signal_async_trans_ADC_SCLK_OK: signal_async_trans
    port map (
      CLK_IN      => CLK_IN,
      SIGNAL_A_IN => adc_sclk_ok,
      SIGNAL_OUT  => adc_sclk_ok_c100
      );

  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
      );

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

  level_to_pulse_ADC_NOTLOCKED: level_to_pulse
    port map (
      CLK_IN    => CLK_IN,
      RESET_IN  => RESET_IN,
      LEVEL_IN  => not adc_locked,
      PULSE_OUT => adc_notlocked_p
      );

  level_to_pulse_ADC_ERROR: level_to_pulse
    port map (
      CLK_IN    => CLK_IN,
      RESET_IN  => RESET_IN,
      LEVEL_IN  => adc_error,
      PULSE_OUT => adc_error_p
      );

  -- 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_c100 = '1') then
          parity_error_counter <= parity_error_counter + 1;
        end if;
      end if;
    end if;
  end process PROC_PARITY_ERROR_COUNTER;

  PROC_MERGE_ERROR_COUNTER: process(CLK_IN)
  begin
    if (rising_edge(CLK_IN)) then
      if (RESET_IN = '1') then
        merge_error_ctr          <= (others => '0');
      else
        if (merge_status_error = '1' or merge_timing_error = '1') then
          merge_error_ctr          <= merge_error_ctr + 1;
        end if;
      end if;
    end if;
  end process PROC_MERGE_ERROR_COUNTER;

  ------------------------------------------------------------------------------
  -- 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_err_rate_ctr    <= (others => '0');
        parity_err_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_frame_clk_c100 = '1') then
            nx_frame_rate_ctr              <= nx_frame_rate_ctr + 1;
          end if;

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

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

          if (parity_error_c100 = '1') then
            parity_err_rate_ctr            <= parity_err_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_err_rate                 <= parity_err_rate_ctr;

        nx_frame_rate_ctr(27 downto 1)  <= (others => '0');
        nx_frame_rate_ctr(0)            <= nx_frame_clk_c100;

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

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

        parity_err_rate_ctr(27 downto 1) <= (others => '0');
        parity_err_rate_ctr(0)          <= parity_error_c100;
      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_error       <= '0';
      adc_frame_rate_error      <= '0';
      frame_rate_error          <= '0';
      parity_rate_error         <= '0';
    else
      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_err_rate > 0) then
        parity_rate_error       <= '1';
      else
        parity_rate_error       <= '0';
      end if;
    end if;
```



```
  end if;
end process PROC_RATE_ERRORS;

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

adc_dt_error_c100 <= adc_dt_error_p;
timestamp_dt_error_c100 <= timestamp_dt_error_p;

PROC_EVENT_ERRORS_PER_SECOND: process(CLK_IN)
begin
  if (rising_edge(CLK_IN)) then
    if (RESET_IN = '1' or frame_rates_reset = '1') then
      adc_dt_error_cur        <= '0';
      adc_dt_error            <= '0';
      timestamp_dt_error_cur  <= '0';
      timestamp_dt_error      <= '0';
    else
      if (rate_timer_ctr < x"5f5e100") then
        if (adc_dt_error_c100 = '1') then
          adc_dt_error_cur        <= '1';
        end if;
        if (timestamp_dt_error_c100 = '1') then
          timestamp_dt_error_cur  <= '1';
        end if;
      else
        adc_dt_error              <= adc_dt_error_cur;
        timestamp_dt_error        <= timestamp_dt_error_cur;
        adc_dt_error_cur          <= '0';
        timestamp_dt_error_cur    <= '0';
      end if;
    end if;
  end if;
end process PROC_EVENT_ERRORS_PER_SECOND;

PROC_DATA_STREAM_DELTA_T: process(CLK_IN)
begin
  if (rising_edge(CLK_IN)) then
    if (RESET_IN = '1') then
      adc_dt_shift_reg        <= (others => '0');
      timestamp_dt_shift_reg  <= (others => '0');
      adc_dt_error_ctr        <= (others => '0');
      timestamp_dt_error_ctr  <= (others => '0');
    else

      -- ADC
      adc_dt_shift_reg(0)          <= adc_data_clk;
      adc_dt_shift_reg(3 downto 1) <= adc_dt_shift_reg(2 downto 0);

      case adc_dt_shift_reg is
        when "1100" | "1110" | "1111" | "0000" =>
          adc_dt_error_ctr         <= adc_dt_error_ctr + 1;
          adc_dt_error_p           <= '1';

        when others =>
          adc_dt_error_p           <= '0';

      end case;

      -- TimeStamp
      timestamp_dt_shift_reg(0)    <= nx_data_clk;
      timestamp_dt_shift_reg(3 downto 1)
        <= timestamp_dt_shift_reg(2 downto 0);
```

```
      case timestamp_dt_shift_reg is
        when "1100" | "1110" | "0000" =>
          timestamp_dt_error_ctr   <= timestamp_dt_error_ctr + 1;
          timestamp_dt_error_p     <= '1';

        when others =>
          timestamp_dt_error_p     <= '0';

      end case;

    end if;
  end if;
end process PROC_DATA_STREAM_DELTA_T;

--------------------------------------------------------------------------------
-- Reset Handler
--------------------------------------------------------------------------------

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_handler       <= '0';
      pll_adc_sampling_clk_reset  <= '0';
      adc_reset_p                 <= '0';
      adc_reset_handler           <= '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';
      nx_timestamp_reset_o        <= '0';
      reset_handler_trigger       <= (others => '0');
      R_STATE                     <= R_IDLE;
    else
      frame_rates_reset           <= '0';
      nx_fifo_reset_handler       <= '0';
      pll_adc_sampling_clk_reset  <= '0';
      adc_reset_p                 <= '0';
      adc_reset_handler           <= '0';
      output_handler_reset        <= '0';

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

      debug_state                 <= x"0";

      if (nx_online = '0') then
        -- If Nxyter is 0ffline nothing will happen
        rs_timeout_timer_reset    <= '1';
        reset_timeout_flag        <= '0';
        R_STATE                   <= R_IDLE;
      elsif (reset_handler_start_r = '1') then
```



```
        -- Reset by register always wins, start it
        rs_timeout_timer_reset    <= '1';
        reset_timeout_flag        <= '0';
        reset_handler_trigger(0)  <= '1';
        reset_handler_trigger(15 downto 1) <= (others => '0');
        R_STATE                   <= R_START;
      elsif (rs_timeout_timer_done = '1') then
        -- Reset Timeout, retry RESET
        rs_timeout_timer_reset    <= '1';
        reset_timeout_flag        <= '1';
        reset_handler_trigger(0)  <= '0';
        reset_handler_trigger(1)  <= '1';
        reset_handler_trigger(15 downto 2) <= (others => '0');
        R_STATE                   <= R_START;
      else

        case R_STATE is
          when R_IDLE =>
            if (nx_online = '1') then
              if ((disable_adc_r = '0' and
                   (pll_adc_not_lock     = '1' or
                    adc_reset_sync       = '1' or
                    adc_frame_rate_error = '1' or
                    adc_dt_error         = '1' or
                    adc_sclk_ok_c100     = '0' or
                    adc_locked           = '0'
                    )
                  ) or
                  (
                    nx_frame_rate_error   = '1' or
                    timestamp_dt_error    = '1' or
                    parity_rate_error     = '1' or
                    nx_frame_rate_error   = '1' or
                    reset_after_offline   = '1' or
                    startup_reset         = '1'
                    )
                  ) then

                reset_handler_trigger(1 downto 0) <= (others => '0');
                reset_handler_trigger( 2) <= startup_reset;
                reset_handler_trigger( 3) <= reset_after_offline;
                reset_handler_trigger( 4) <= nx_frame_rate_error;
                reset_handler_trigger( 5) <= parity_rate_error;
                reset_handler_trigger( 6) <= timestamp_dt_error;
                reset_handler_trigger( 7) <= nx_frame_rate_error;
                reset_handler_trigger( 8) <= not adc_locked;
                reset_handler_trigger( 9) <= not adc_sclk_ok_c100;
                reset_handler_trigger(10) <= adc_dt_error;
                reset_handler_trigger(11) <= adc_frame_rate_error;
                reset_handler_trigger(12) <= adc_reset_sync;
                reset_handler_trigger(13) <= pll_adc_not_lock;
                reset_handler_trigger(15 downto 14) <= (others => '0');

                R_STATE                 <= R_START;
              else
                reset_timeout_flag      <= '0';
                rs_timeout_timer_reset  <= '1';
                reset_handler_busy      <= '0';
                R_STATE                 <= R_IDLE;
              end if;
            else
              reset_timeout_flag        <= '0';
```

```
          rs_timeout_timer_reset    <= '1';
          reset_handler_busy        <= '0';
          R_STATE                   <= R_IDLE;
        end if;

        debug_state                 <= x"1";

      when R_START =>
        -- First wait 1mue for NX_MAIN_CLK, have to put lock status here
        -- to check in the future.
        rs_wait_timer_start         <= '1';
        R_STATE                     <= R_WAIT_0;
        debug_state                 <= x"2";

      when R_WAIT_0 =>
        if (rs_wait_timer_done = '0') then
          R_STATE                   <= R_WAIT_0;
        else
          R_STATE                   <= R_RESET_TIMESTAMP ;
        end if;
        debug_state                 <= x"3";

      when R_RESET_TIMESTAMP =>
        -- must reset/resync Timestamp clock and data transmission clock
        -- of nxyter first, afterwards wait a bit to let settle down
        reset_handler_counter       <= reset_handler_counter + 1;
        nx_timestamp_reset_o        <= '1';
        rs_wait_timer_start         <= '1';  -- wait 1mue to settle
        R_STATE                     <= R_WAIT_1;
        debug_state                 <= x"4";

      when R_WAIT_1 =>
        if (rs_wait_timer_done = '0') then
          R_STATE                   <= R_WAIT_1;
        else
          R_STATE                   <= R_SET_ALL_RESETS;
        end if;
        debug_state                 <= x"5";

      when R_SET_ALL_RESETS =>
        -- timer reset should be finished, can we check status,
        -- To be done?
        -- now set reset of all handlers
        frame_rates_reset           <= '1';
        pll_adc_sampling_clk_reset  <= '1';
        adc_reset_p                 <= '1';
        adc_reset_handler           <= '1';
        output_handler_reset        <= '1';
        nx_fifo_reset_handler       <= '1';

        -- give resets 1mue to take effect
        rs_wait_timer_start         <= '1';
        R_STATE                     <= R_WAIT_2;
        debug_state                 <= x"6";

      when R_WAIT_2 =>
        pll_adc_sampling_clk_reset  <= '1';
        adc_reset_handler           <= '1';
        output_handler_reset        <= '1';
        nx_fifo_reset_handler       <= '1';
        if (rs_wait_timer_done = '0') then
          R_STATE                   <= R_WAIT_2;
```



```
        else
          -- now start timeout timer and begin to release resets
          -- step by step
          rs_timeout_timer_start  <= '1';
          R_STATE                 <= R_WAIT_NX_FRAME_RATE_OK;
        end if;
        debug_state               <= x"7";

      when R_WAIT_NX_FRAME_RATE_OK =>
        if (nx_frame_rate_error   = '0') then
          -- Next: Release PLL Reset, i.e. sampling_clk_reset
          adc_reset_handler       <= '1';
          output_handler_reset    <= '1';
          nx_fifo_reset_handler   <= '1';
          if (disable_adc_r = '0') then
            R_STATE               <= R_PLL_WAIT_LOCK;
          else
            R_STATE               <= R_WAIT_DATA_HANDLER_OK;
          end if;
        else
          pll_adc_sampling_clk_reset <= '1';
          adc_reset_handler       <= '1';
          output_handler_reset    <= '1';
          nx_fifo_reset_handler   <= '1';
          R_STATE                 <= R_WAIT_NX_FRAME_RATE_OK;
        end if;
        debug_state               <= x"8";

      when R_PLL_WAIT_LOCK =>
        if (adc_sclk_ok_c100 = '1' and
            pll_adc_not_lock = '0') then
          -- Next: Release ADC Reset
          output_handler_reset    <= '1';
          nx_fifo_reset_handler   <= '1';
          R_STATE                 <= R_WAIT_ADC_OK;
        else
          adc_reset_handler       <= '1';
          output_handler_reset    <= '1';
          nx_fifo_reset_handler   <= '1';
          R_STATE                 <= R_PLL_WAIT_LOCK;
        end if;
        debug_state               <= x"9";

      when R_WAIT_ADC_OK =>
        if (adc_locked = '1' and
            adc_frame_rate_error = '0') then
          -- Next: Release Output Handler and Clock Domain transfer Fifo
          -- Resets
          R_STATE                 <= R_WAIT_DATA_HANDLER_OK;
        else
          output_handler_reset    <= '1';
          nx_fifo_reset_handler   <= '1';
          R_STATE                 <= R_WAIT_ADC_OK;
        end if;
        debug_state               <= x"a";

      when R_WAIT_DATA_HANDLER_OK =>
        if (frame_rate_error  = '0') then
          startup_reset           <= '0';
          reset_timeout_flag      <= '0';
          rs_timeout_timer_reset  <= '1';
          R_STATE                 <= R_IDLE;
```

```
            else
              R_STATE                 <= R_WAIT_DATA_HANDLER_OK;
            end if;
            debug_state               <= x"b";
        end case;
      end if;
    end if;
  end if;
end process PROC_RESET_HANDLER;

------------------------------------------------------------------------------
-- Error Status
------------------------------------------------------------------------------
PROC_ERROR_STATUS: process(CLK_IN)
  variable error_mask : std_logic_vector(15 downto 0);
begin
  if (rising_edge(CLK_IN)) then
    if (RESET_IN = '1') then
      nx_online               <= '0';
      nx_online_shift         <= (others => '0');
      reset_after_offline     <= '0';
      error_status_bits       <= (others => '0');
      error_o                 <= '0';
      adc_notlock_counter     <= (others => '0');
      adc_error_counter       <= (others => '0');
    else
      error_status_bits(0)             <= not nx_online;
      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 nx_frame_synced_r;
      error_status_bits(6)             <= '0';
      error_status_bits(7)             <= pll_adc_not_lock;
      error_status_bits(8)             <= not adc_sclk_ok_c100;
      error_status_bits(9)             <= not adc_locked;
      error_status_bits(10)            <= timestamp_dt_error;
      error_status_bits(11)            <= adc_dt_error;
      error_status_bits(12)            <= reset_handler_busy;
      error_status_bits(15 downto 13)  <= (others => '0');

      nx_online                  <= NX_CLOCK_ON_IN and NX_ONLINE_IN;
      nx_online_shift(0)         <= nx_online;
      nx_online_shift(1)         <= nx_online_shift(0);
      if (nx_online_shift = "01") then
        reset_after_offline      <= '1';
      else
        reset_after_offline      <= '0';
      end if;

      if (disable_adc_r = '1') then
        error_mask := x"f437";
      else
        error_mask := x"0000";
      end if;

      if ((error_status_bits and error_mask) = x"0000") then
        error_o                  <= '0';
      else
        error_o                  <= '1';
      end if;
```



```
      if (adc_notlocked_p = '1') then
       adc_notlock_counter             <= adc_notlock_counter + 1;
      end if;

      if (adc_error_p = '1') then
       adc_error_counter               <= adc_error_counter + 1;
      end if;

    end if;
  end if;
end process PROC_ERROR_STATUS;

PROC_NX_TIMESTAMP_DELAY_ADJUST: process(CLK_IN)
begin
  if (rising_edge(CLK_IN)) then
    if (RESET_IN = '1') then
      nx_timestamp_delay_a          <= (others => '0');
      nx_timestamp_delay_actr       <= (others => '0');
    else
      -- Automatic nx_timestamp_delay adjust
      if (nx_timestamp_delay_adjust = '1' and ADC_TR_ERROR_IN = '1') then
        if (nx_timestamp_delay_a <= "100") then
          nx_timestamp_delay_a      <= nx_timestamp_delay_a + 1;
        else
          nx_timestamp_delay_a      <= (others => '0');
        end if;
        nx_timestamp_delay_actr     <= nx_timestamp_delay_actr + 1;
      end if;

      -- Multiplexer
      if (nx_timestamp_delay_adjust = '1') then
        nx_timestamp_delay_s        <= nx_timestamp_delay_a;
      else
        nx_timestamp_delay_s        <= nx_timestamp_delay_r;
      end if;
    end if;
  end if;
end process PROC_NX_TIMESTAMP_DELAY_ADJUST;

------------------------------------------------------------------------------
-- TRBNet Slave Bus
------------------------------------------------------------------------------
PROC_SLAVE_BUS_BUFFER: process(CLK_IN)
begin
  if (rising_edge(CLK_IN)) then
    nx_frame_synced_rr                <= nx_frame_synced;
    --nx_frame_word_delay_rr            <= nx_frame_word_delay_f;

    if (RESET_IN = '1') then
      nx_frame_synced_r               <= '0';
      adc_dt_error_ctr_r              <= (others => '0');
      timestamp_dt_error_ctr_r        <= (others => '0');
      adc_notlock_ctr_r               <= (others => '0');
      merge_error_ctr_r               <= (others => '0');
    else
      nx_frame_synced_r               <= nx_frame_synced_rr;
      adc_dt_error_ctr_r              <= adc_dt_error_ctr;
      timestamp_dt_error_ctr_r        <= timestamp_dt_error_ctr;
      adc_notlock_ctr_r               <= adc_notlock_ctr;
      merge_error_ctr_r               <= merge_error_ctr;
    end if;
  end if;
```

```
end process PROC_SLAVE_BUS_BUFFER;

-- 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_ack_o                    <= '0';
      slv_unknown_addr_o           <= '0';
      slv_no_more_data_o           <= '0';

      reset_resync_ctr             <= '0';
      reset_parity_error_ctr       <= '0';
      johnson_counter_sync_r       <= "00";
      pll_adc_sample_clk_dphase_r  <= x"5";
      pll_adc_sample_clk_finedelb_r <= (others => '0');
      pll_adc_not_lock_ctr_clear   <= '0';
      nx_timestamp_delay_adjust    <= '1';
      nx_timestamp_delay_r         <= "011";
      reset_handler_start_r        <= '0';
      adc_debug_type_r             <= (others => '0');
      debug_mode                   <= (others => '0');
      disable_adc_r                <= '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';

      if (SLV_READ_IN  = '1') then
        case SLV_ADDR_IN is
          when x"0000" =>
            slv_data_out_o(15 downto 0)   <= error_status_bits;
            slv_data_out_o(31 downto 16)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0001" =>
            slv_data_out_o(0)             <= reset_handler_busy;
            slv_data_out_o(1)             <= reset_timeout_flag;
            slv_data_out_o(15 downto 2)   <= (others => '0');
            slv_data_out_o(31 downto 16)  <= reset_handler_trigger;
            slv_ack_o                     <= '1';

          when x"0002" =>
            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"0003" =>
            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"0004" =>
            slv_data_out_o(27 downto 0)   <= std_logic_vector(adc_frame_rate);
            slv_data_out_o(30 downto 28)  <= (others => '0');
            slv_data_out_o(30)            <= adc_sloppy_frame;
```



```
            slv_data_out_o(31)            <= disable_adc_r;
            slv_ack_o                     <= '1';

          when x"0005" =>
            slv_data_out_o(27 downto 0)   <=
              std_logic_vector(parity_err_rate);
            slv_data_out_o(31 downto 28)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0006" =>
            slv_data_out_o(2 downto 0)    <=
              std_logic_vector(nx_timestamp_delay_s);
            slv_data_out_o(14 downto 3)   <= (others => '0');
            slv_data_out_o(15)            <= nx_timestamp_delay_adjust;
            slv_data_out_o(31 downto 16)  <= nx_timestamp_delay_actr;
            slv_ack_o                     <= '1';

          when x"0007" =>
            slv_data_out_o(3 downto 0)    <=
              std_logic_vector(pll_adc_sample_clk_dphase_r);
            slv_data_out_o(15 downto 4)   <= (others => '0');
            slv_data_out_o(19 downto 16)  <=
              std_logic_vector(pll_adc_sample_clk_finedelb_r);
            slv_data_out_o(31 downto 20)   <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0008" =>
            slv_data_out_o(15 downto 0)   <=
              std_logic_vector(reset_handler_counter);
            slv_data_out_o(31 downto 16)  <= (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(11 downto 0)   <=
              std_logic_vector(adc_notlock_ctr_r);
            slv_data_out_o(31 downto 12)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"000b" =>
            slv_data_out_o(11 downto 0)   <=
              std_logic_vector(merge_error_ctr_r);
            slv_data_out_o(31 downto 12)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"000c" =>
            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"000d" =>
            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"000e" =>
            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"000f" =>
            slv_data_out_o(11 downto 0)   <=
              std_logic_vector(adc_dt_error_ctr_r);
            slv_data_out_o(31 downto 12)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0010" =>
            slv_data_out_o(11 downto 0)   <=
              std_logic_vector(timestamp_dt_error_ctr_r);
            slv_data_out_o(31 downto 12)  <= (others => '0');
            slv_ack_o                     <= '1';

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

          when x"0012" =>
            slv_data_out_o(27 downto 0)   <=
              std_logic_vector(adc_error_counter);
            slv_data_out_o(31 downto 15)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"001d" =>
            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"001e" =>
            slv_data_out_o(2 downto 0)    <= debug_mode;
            slv_data_out_o(31 downto 3)   <= (others => '0');
            slv_ack_o                     <= '1';

          when x"001f" =>
            slv_data_out_o(3 downto 0)    <= adc_debug_type_r;
            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"0001" =>
            reset_handler_start_r         <= '1';
            slv_ack_o                     <= '1';

          when x"0004" =>
            disable_adc_r                 <= SLV_DATA_IN(31);
            adc_sloppy_frame              <= SLV_DATA_IN(30);
            slv_ack_o                     <= '1';

          when x"0006" =>
            nx_timestamp_delay_r          <=
```



```
                unsigned(SLV_DATA_IN(2 downto 0));
              nx_timestamp_delay_adjust     <= SLV_DATA_IN(15);
              slv_ack_o                     <= '1';

          when x"0007" =>
            pll_adc_sample_clk_dphase_r   <=
              unsigned(SLV_DATA_IN(3 downto 0));
            pll_adc_sample_clk_finedelb_r <=
              unsigned(SLV_DATA_IN(19 downto 16));
            reset_handler_start_r         <= '1';
            slv_ack_o                     <= '1';


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

          when x"000d" =>
            reset_parity_error_ctr        <= '1';
            slv_ack_o                     <= '1';

          when x"000e" =>
            pll_adc_not_lock_ctr_clear    <= '1';
            slv_ack_o                     <= '1';

          when x"001d" =>
            johnson_counter_sync_r
              <= unsigned(SLV_DATA_IN(1 downto 0)) + 1;
            reset_handler_start_r         <= '1';
            slv_ack_o                     <= '1';

          when x"001e" =>
            debug_mode                    <= SLV_DATA_IN(2 downto 0);
            slv_ack_o                     <= '1';

          when x"001f" =>
            adc_debug_type_r              <=
              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_SLAVE_BUS;

-- Output Signals
data_o                  <= data_m;
data_clk_o              <= data_clk_m;

NX_TIMESTAMP_RESET_OUT  <= nx_timestamp_reset_o;
DATA_OUT                <= data_o;
DATA_CLK_OUT            <= data_clk_o;
ADC_SCLK_LOCK_OUT       <= pll_adc_sampling_clk_lock;
DISABLE_ADC_OUT         <= disable_adc_r;
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
    DATA_IN                : in  std_logic_vector(43 downto 0);
    DATA_CLK_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_CLK_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;

    ADC_TR_ERROR_OUT       : out std_logic;
    DISABLE_ADC_IN         : in std_logic;
    ERROR_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_clk_o           : std_logic;

  signal nx_token_return_o    : std_logic;
  signal nx_nomore_data_o     : std_logic;

  signal parity_error_ctr     : unsigned(15 downto 0);
  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 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_trigger_ctr_t_nr  : unsigned(31 downto 0);
  signal 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 adc_tr_error_ctr_t   : unsigned(27 downto 0);
  signal adc_tr_update_ctr_t  : unsigned(27 downto 0);
  signal adc_tr_data_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(15 downto 0);
  signal adc_average_sum      : unsigned(31 downto 0);
  signal adc_average          : unsigned(11 downto 0);

  signal adc_data_last        : std_logic_vector(11 downto 0);

  -- Token Return Average
  signal nx_token_return_pipec  : std_logic_vector(9 downto 0);
  signal nx_token_return_pipev  : std_logic_vector(11 downto 0);
  signal adc_tr_value_tmp       : std_logic_vector(11 downto 0);
  signal adc_tr_value           : std_logic_vector(11 downto 0);
  signal adc_tr_data_p          : unsigned(11 downto 0);
  signal adc_tr_data_c          : unsigned(11 downto 0);
  signal adc_tr_data_clk        : std_logic;
  signal adc_tr_limit           : unsigned(11 downto 0);
  signal adc_tr_error_ctr       : unsigned(11 downto 0);
  signal adc_tr_error           : std_logic;
  signal adc_tr_error_status    : std_logic_vector(1 downto 0);
  signal adc_tr_debug_mode      : std_logic;
  signal adc_tr_error_o         : std_logic;

  type TR_STATES is (S_IDLE,
                     S_START,
                     S_END
                     );
  signal TR_STATE : TR_STATES;
```

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

  -- Error Status
  signal new_timestamp_shift  : std_logic_vector(3 downto 0);
  signal frame_dt_error       : std_logic;
  signal frame_dt_error_ctr   : unsigned(15 downto 0);
  signal frame_rate_error     : std_logic;
  signal error_o              : 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 clear_counters       : std_logic;
  signal nx_hit_rate          : unsigned(27 downto 0);
  signal frame_rate           : unsigned(27 downto 0);
  signal nx_pileup_rate       : unsigned(27 downto 0);
  signal nx_overflow_rate     : unsigned(27 downto 0);
  signal adc_tr_error_rate    : unsigned(27 downto 0);
  signal adc_tr_update_rate   : unsigned(27 downto 0);
  signal adc_tr_data_rate     : unsigned(27 downto 0);
  signal invalid_adc          : std_logic;
  signal adc_tr_value_r       : std_logic_vector(11 downto 0);

  signal adc_tr_debug_p       : std_logic;
  signal adc_tr_debug_c       : std_logic;
  signal adc_tr_value_update  : std_logic;
  signal state_debug          : std_logic_vector(1 downto 0);

begin

  -- Debug Line
  DEBUG_OUT(0)             <= CLK_IN;
  DEBUG_OUT(1)             <= DATA_CLK_IN;
  DEBUG_OUT(2)             <= nx_token_return_o;
  DEBUG_OUT(3)             <= nx_nomore_data_o;
  DEBUG_OUT(4)             <= data_clk_o;
  DEBUG_OUT(5)             <= new_timestamp;
  DEBUG_OUT(6)             <= self_trigger_o;
--  DEBUG_OUT(7)               <= invalid_adc;
  DEBUG_OUT(7)             <= adc_tr_data_clk;
  DEBUG_OUT(8)             <= adc_tr_error;
  DEBUG_OUT(10 downto 9)   <= adc_tr_error_status;
  DEBUG_OUT(11)            <= adc_tr_debug_p;
  DEBUG_OUT(12)            <= adc_tr_debug_c;
  DEBUG_OUT(13)            <= adc_tr_value_update;
  DEBUG_OUT(15 downto 14)  <= state_debug;


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

  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 DATA_IN(30 downto 24),
      GRAY_IN( 6 downto 0)  => not DATA_IN(22 downto 16),
      BINARY_OUT            => nx_timestamp
      );

  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    => DATA_IN(14 downto 8),
      BINARY_OUT => nx_channel_id
      );

  -- 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');
        status_bits         <= (others => '0');
        new_timestamp       <= '0';
        adc_data            <= (others => '0');
      else
        if (DATA_CLK_IN = '1') then
          valid_frame_bits(3)       <= DATA_IN(31);
          valid_frame_bits(2)       <= DATA_IN(23);
          valid_frame_bits(1)       <= DATA_IN(15);
          valid_frame_bits(0)       <= DATA_IN(7);
          status_bits               <= DATA_IN(2 downto 1);
          adc_data                  <= DATA_IN(43 downto 32);
          new_timestamp             <= '1';
        else
          valid_frame_bits          <= (others => '0');
          status_bits               <= (others => '0');
          adc_data                  <= (others => '0');
          new_timestamp             <= '0';
        end if;
      end if;
    end if;
  end process PROC_TIMESTAMP_BITS;

  -- Check Parity Bit
  PROC_PARITY_CHECKER: process(CLK_IN)
    variable parity_bits : std_logic_vector(22 downto 0);
    variable parity      : std_logic;
  begin
    if (rising_edge(CLK_IN)) then
      if (RESET_IN = '1') then
        parity_error        <= '0';
      else
        if (DATA_CLK_IN = '1') then
          -- Timestamp Bit #6 is excluded (funny nxyter-bug)
          parity_bits       := DATA_IN(31)           &
                               DATA_IN(30 downto 24) &
                               DATA_IN(21 downto 16) &
                               DATA_IN(14 downto  8) &
                               DATA_IN( 2 downto  1);
          parity            := xor_all(parity_bits);
```

```
        if (parity /= DATA_IN(0)) then
          parity_error    <= '1';
        else
          parity_error    <= '0';
        end if;
      else
        parity_error      <= '0';
      end if;
    end if;
  end if;
end process PROC_PARITY_CHECKER;

-------------------------------------------------------------------------------
-- 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_clk_o            <= '0';
      nx_token_return_o     <= '0';
      nx_nomore_data_o      <= '0';
      trigger_rate_inc      <= '0';
      pileup_rate_inc       <= '0';
      overflow_rate_inc     <= '0';
      parity_error_ctr      <= (others => '0');
      invalid_frame_ctr     <= (others => '0');
      overflow_ctr          <= (others => '0');
      pileup_ctr            <= (others => '0');
      invalid_adc           <= '0';
      adc_tr_data_p         <= (others => '0');
      adc_tr_data_c         <= (others => '0');
      adc_tr_data_clk       <= '0';
      nx_token_return_pipec <= (others => '0');
      nx_token_return_pipev <= (others => '0');
      adc_data_last         <= (others => '0');
      adc_tr_value_tmp      <= (others => '0');
      adc_tr_value_update   <= '0';
      TR_STATE              <= S_IDLE;
      state_debug           <= "00";
    else
      timestamp_o           <= (others => '0');
      channel_o             <= (others => '0');
      timestamp_status_o    <= (others => '0');
      adc_data_o            <= (others => '0');
      data_clk_o            <= '0';
      trigger_rate_inc      <= '0';
      pileup_rate_inc       <= '0';
      overflow_rate_inc     <= '0';
      invalid_adc           <= '0';
      adc_tr_data_clk       <= '0';
      adc_tr_value_update   <= '0';

      if (new_timestamp = '1') then

        adc_data_last                     <= adc_data;
```



```
      if (parity_error = '1') then
        parity_error_ctr <= parity_error_ctr + 1;
      end if;

      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;
          if (adc_tr_debug_mode = '0') then
            adc_data_o                  <= adc_data;
          else
            adc_data_o                  <= std_logic_vector(adc_tr_data_p);
          end if;
          data_clk_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';

          if (nx_token_return_o = '1' and
              nx_token_return_pipec = "1111111111") then
            -- First Data Word after 5 empty Frames
            adc_tr_data_p               <= unsigned(adc_data_last);
            adc_tr_data_c               <= unsigned(adc_data);
            adc_tr_data_clk             <= '1';
          end if;

        when "0000" =>
          -- Token return and nomore_data
          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;

        -- Token Return Check Handler
        case TR_STATE is
          when S_IDLE =>
            if (nx_token_return_pipev(6 downto 0) = "1111111") then
              adc_tr_value_tmp             <= adc_data_last;
              TR_STATE                     <= S_START;
            else
              TR_STATE                     <= S_IDLE;
            end if;
            state_debug                    <= "01";

          when S_START =>
            if (nx_token_return_pipev = "111111111111") then
              TR_STATE                     <= S_END;
            elsif (nx_token_return_pipev( 6 downto 0) = "1111111" or
                   nx_token_return_pipev( 7 downto 0) = "11111111" or
                   nx_token_return_pipev( 8 downto 0) = "111111111" or
                   nx_token_return_pipev( 9 downto 0) = "1111111111" or
                   nx_token_return_pipev(10 downto 0) = "11111111111") then
              TR_STATE                     <= S_START;
            else
              TR_STATE                     <= S_IDLE;
            end if;
            state_debug                    <= "10";

          when S_END =>
            adc_tr_value                   <= adc_tr_value_tmp;
            adc_tr_value_update            <= '1';
            TR_STATE                       <= S_IDLE;
            state_debug                    <= "11";

        end case;

        -- Token Return Pipeline
        nx_token_return_pipec(0)           <= nx_token_return_o;
        for I in 1 to 9 loop
          nx_token_return_pipec(I)         <= nx_token_return_pipec(I - 1);
        end loop;

        if (TR_STATE /= S_END) then
          nx_token_return_pipev(0)         <= nx_token_return_o;
          for I in 1 to 11 loop
            nx_token_return_pipev(I)       <= nx_token_return_pipev(I - 1);
          end loop;
        else
          nx_token_return_pipev            <= (others => '0');
        end if;

      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_rate_timer        <= (others => '0');

      nx_trigger_ctr_t     <= (others => '0');
      frame_ctr_t          <= (others => '0');
      adc_tr_error_ctr_t   <= (others => '0');
      adc_tr_update_ctr_t  <= (others => '0');
      adc_tr_data_ctr_t    <= (others => '0');

      nx_hit_rate          <= (others => '0');
      frame_rate           <= (others => '0');
      adc_tr_error_rate    <= (others => '0');
      adc_tr_update_rate   <= (others => '0');
      adc_tr_data_rate     <= (others => '0');

      nx_trigger_ctr_t_nr  <= (others => '0');
      adc_tr_error_ctr     <= (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;
          nx_trigger_ctr_t_nr          <= nx_trigger_ctr_t_nr + 1;
        end if;
        if (new_timestamp = '1') then
          frame_ctr_t                  <= 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;
        if (adc_tr_error = '1') then
          adc_tr_error_ctr_t           <= adc_tr_error_ctr_t + 1;
          adc_tr_error_ctr             <= adc_tr_error_ctr + 1;
        end if;
        if (adc_tr_value_update = '1') then
          adc_tr_update_ctr_t          <= adc_tr_update_ctr_t + 1;
        end if;
        if (adc_tr_data_clk = '1') then
          adc_tr_data_ctr_t            <= adc_tr_data_ctr_t + 1;
        end if;

        nx_rate_timer                  <= nx_rate_timer + 1;
      else
        nx_hit_rate                    <= nx_trigger_ctr_t;
        frame_rate                     <= frame_ctr_t;
        nx_pileup_rate                 <= nx_pileup_ctr_t;
        nx_overflow_rate               <= nx_overflow_ctr_t;
        adc_tr_error_rate              <= adc_tr_error_ctr_t;
        adc_tr_update_rate             <= adc_tr_update_ctr_t;
        adc_tr_data_rate               <= adc_tr_data_ctr_t;

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

        frame_ctr_t(27 downto 1)       <= (others => '0');
        frame_ctr_t(0)                 <= new_timestamp;
```

```
        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;

        adc_tr_error_ctr_t(27 downto 0)  <= (others => '0');
        adc_tr_error_ctr_t(0)            <= adc_tr_error;

        adc_tr_update_ctr_t(27 downto 0)  <= (others => '0');
        adc_tr_update_ctr_t(0)            <= adc_tr_value_update;

        adc_tr_data_ctr_t(27 downto 0)    <= (others => '0');
        adc_tr_data_ctr_t(0)              <= adc_tr_data_clk;

        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');
    else
      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);
        if (data_clk_o = '1') then
          adc_average_sum(11 downto 0)  <= unsigned(adc_data_o);
          adc_average_sum(31 downto 13) <= (others => '0');
          adc_average_ctr  <= x"0001";
        else
          adc_average_sum  <= (others => '0');
          adc_average_ctr  <= (others => '0');
        end if;
      elsif (data_clk_o = '1') then
        adc_average_sum    <= adc_average_sum + unsigned(adc_data_o);
        adc_average_ctr    <= adc_average_ctr + 1;
      end if;

    end if;
  end if;
end process PROC_ADC_AVERAGE;

PROC_ADC_TOKEN_RETURN: process(CLK_IN)
  variable lower_limit      : unsigned(11 downto 0);
begin
  if (rising_edge(CLK_IN) ) then
    if (RESET_IN = '1') then
      adc_tr_error          <= '0';
      adc_tr_debug_p        <= '0';
      adc_tr_debug_c        <= '0';
      adc_tr_error_status   <= "00";
    else
      lower_limit            := unsigned(adc_tr_value) - adc_tr_limit;
      adc_tr_error           <= '0';
```



```
      if (adc_tr_data_clk = '1') then

        if (adc_tr_data_p    > lower_limit) then
          adc_tr_debug_p     <= '1';
        else
          adc_tr_debug_p     <= '0';
        end if;

        if (adc_tr_data_c    > lower_limit) then
          adc_tr_debug_c     <= '1';
        else
          adc_tr_debug_c     <= '0';
        end if;

        if (adc_tr_data_p    > lower_limit and
            adc_tr_data_c    < lower_limit) then
          adc_tr_error_status  <= "00";
          adc_tr_error         <= '0';

        elsif (adc_tr_data_p > lower_limit and
               adc_tr_data_c > lower_limit) then
          adc_tr_error_status  <= "01";
          adc_tr_error         <= '1';
        else
          adc_tr_error_status  <= "10";
          adc_tr_error         <= '1';
        end if;
      end if;

    end if;
  end if;
end process PROC_ADC_TOKEN_RETURN;

PROC_ERROR_HANDLER: process(CLK_IN)
begin
  if (rising_edge(CLK_IN) ) then
    if (RESET_IN = '1') then
      frame_rate_error <= '0';
      error_o          <= '0';
    else
      if (adc_tr_error_rate > x"0000020" and DISABLE_ADC_IN = '0') then
        error_o               <= '1';
      else
        error_o               <= '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;
    end if;
  end if;
end process PROC_ERROR_HANDLER;

PROC_DATA_STREAM_DELTA_T: process(CLK_IN)
begin
  if (rising_edge(CLK_IN)) then
    if (RESET_IN = '1') then
      new_timestamp_shift     <= (others => '0');
      frame_dt_error_ctr      <= (others => '0');
```

```
      frame_dt_error          <= '0';
    else
      -- Frame
      new_timestamp_shift(0)          <= new_timestamp;
      new_timestamp_shift(3 downto 1) <= new_timestamp_shift(2 downto 0);

      case new_timestamp_shift is
        when "1100" | "1110" | "1111" | "0000" =>
          frame_dt_error_ctr          <= frame_dt_error_ctr + 1;
          frame_dt_error              <= '1';

        when others =>
          frame_dt_error              <= '0';

      end case;

    end if;
  end if;
end process PROC_DATA_STREAM_DELTA_T;

------------------------------------------------------------------------------
-- 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
    adc_tr_value_r         <= adc_tr_value;
    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";

      adc_tr_limit           <= x"064";  -- 100
      adc_tr_debug_mode      <= '0';
    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(27 downto 0)   <=
              std_logic_vector(nx_hit_rate);
            slv_data_out_o(31 downto 28)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0001" =>
            slv_data_out_o(27 downto 0)   <=
              std_logic_vector(frame_rate);
            slv_data_out_o(30 downto 28)  <= (others => '0');
            slv_data_out_o(31)            <= frame_rate_error;
            slv_ack_o                     <= '1';

          when x"0002" =>
```



```
            slv_data_out_o(15 downto 0)
              <= std_logic_vector(parity_error_ctr);
            slv_data_out_o(31 downto 16)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0003" =>
            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"0004" =>
            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"0005" =>
            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 x"0006" =>
            slv_data_out_o(1 downto 0)    <= adc_tr_error_status;
            slv_data_out_o(31 downto 2)   <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0007" =>
            slv_data_out_o(27 downto 0)
              <= std_logic_vector(adc_tr_error_rate);
            slv_data_out_o(31 downto 28)  <= (others => '0');
            slv_ack_o                     <= '1';

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

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

          when x"000b" =>
            slv_data_out_o(11 downto 0)
              <= std_logic_vector(adc_tr_limit);
            slv_data_out_o(31 downto 12)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"000c" =>
            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"000d" =>
            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"000e" =>
            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"000f" =>
            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"0010" =>
            slv_data_out_o(11 downto 0)
              <= std_logic_vector(adc_tr_error_ctr);
            slv_data_out_o(31 downto 12)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0011" =>
            slv_data_out_o(0)             <= adc_tr_debug_mode;
            slv_data_out_o(31 downto 1)   <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0012" =>
            slv_data_out_o                <= nx_trigger_ctr_t_nr;
            slv_ack_o                     <= '1';

          when x"0013" =>
            slv_data_out_o(15 downto 0)   <= frame_dt_error_ctr;
            slv_data_out_o(31 downto 16)  <= (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"000b" =>
            adc_tr_limit
              <= unsigned(SLV_DATA_IN(11 downto 0));
            slv_ack_o                     <= '1';

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

          when x"0011" =>
            adc_tr_debug_mode             <= 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
-------------------------------------------------------------------------------

adc_tr_error_o         <= adc_tr_error;

TIMESTAMP_OUT          <= timestamp_o;
CHANNEL_OUT            <= channel_o;
TIMESTAMP_STATUS_OUT   <= timestamp_status_o;
ADC_DATA_OUT           <= adc_data_o;
DATA_CLK_OUT           <= data_clk_o;
NX_TOKEN_RETURN_OUT    <= nx_token_return_o;
NX_NOMORE_DATA_OUT     <= nx_nomore_data_o;

ADC_TR_ERROR_OUT       <= adc_tr_error_o;
ERROR_OUT              <= error_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: entity work.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') then
        flush_end_enable    <= '0';
      else
        if (flush_end_enable_reset_x = '1') then
          flush_end_enable  <= '0';
        elsif (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_RESET_IN      : in  std_logic;
    TIMESTAMP_RESET_OUT     : out std_logic;
    TRIGGER_IN              : in  std_logic; -- must be in NX_MAIN_CLK_DOMAIN
    TIMESTAMP_HOLD_OUT      : out unsigned(11 downto 0);
    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
  type S_STATES is (S_IDLE,
                    S_RESET,
                    S_RESET_WAIT,
                    S_HOLD
                    );
  signal S_STATE : S_STATES;

  signal wait_timer_start     : std_logic;
  signal wait_timer_done      : std_logic;

  signal timestamp_reset_ff   : std_logic;
  signal timestamp_reset_f    : std_logic;
  signal timestamp_reset      : std_logic;
  signal timestamp_ctr        : unsigned(11 downto 0);

  signal timestamp_hold_o     : std_logic_vector(11 downto 0);
  signal timestamp_trigger_o  : std_logic;
  signal timestamp_reset_o    : std_logic;

  -- Reset
  signal reset_nx_main_clk_in_ff  : std_logic;
  signal reset_nx_main_clk_in_f   : std_logic;
  signal RESET_NX_MAIN_CLK_IN     : std_logic;

  attribute syn_keep : boolean;
  attribute syn_keep of reset_nx_main_clk_in_ff     : signal is true;
  attribute syn_keep of reset_nx_main_clk_in_f      : signal is true;

  attribute syn_keep of timestamp_reset_ff          : signal is true;
  attribute syn_keep of timestamp_reset_f           : signal is true;

  attribute syn_preserve : boolean;
  attribute syn_preserve of reset_nx_main_clk_in_ff : signal is true;
  attribute syn_preserve of reset_nx_main_clk_in_f  : signal is true;

  attribute syn_preserve of timestamp_reset_ff      : signal is true;
  attribute syn_preserve of timestamp_reset_f       : signal is true;

begin
  DEBUG_OUT(0)              <= NX_MAIN_CLK_IN;
  DEBUG_OUT(1)              <= '0';
  DEBUG_OUT(2)              <= TIMESTAMP_RESET_IN;
  DEBUG_OUT(3)              <= '0';
  DEBUG_OUT(4)              <= TRIGGER_IN;
  DEBUG_OUT(5)              <= '0';
  DEBUG_OUT(6)              <= timestamp_reset_ff;
  DEBUG_OUT(7)              <= '0';
  DEBUG_OUT(8)              <= timestamp_reset_f;
  DEBUG_OUT(9)              <= '0';
  DEBUG_OUT(10)             <= timestamp_reset;
  DEBUG_OUT(11)             <= '0';
  DEBUG_OUT(12)             <= timestamp_reset_o;
  DEBUG_OUT(13)             <= '0';
  DEBUG_OUT(14)             <= timestamp_trigger_o;

  DEBUG_OUT(15)             <= '0';
  --timestamp_hold_o(10 downto 0);
```

```
-------------------------------------------------------------------------------
-- Reset Domain Transfer
-------------------------------------------------------------------------------
reset_nx_main_clk_in_ff   <= RESET_IN when rising_edge(NX_MAIN_CLK_IN);
reset_nx_main_clk_in_f    <= reset_nx_main_clk_in_ff
                             when rising_edge(NX_MAIN_CLK_IN);
RESET_NX_MAIN_CLK_IN      <= reset_nx_main_clk_in_f
                             when rising_edge(NX_MAIN_CLK_IN);

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

-- Timestamp Process + Trigger
timestamp_reset_ff         <= TIMESTAMP_RESET_IN
                              when rising_edge(NX_MAIN_CLK_IN);
timestamp_reset_f          <= timestamp_reset_ff
                              when rising_edge(NX_MAIN_CLK_IN);
timestamp_reset            <= timestamp_reset_f
                              when rising_edge(NX_MAIN_CLK_IN);

-- Timer
timer_static_TS_RESET: timer_static
  generic map (
    CTR_WIDTH => 3,
    CTR_END   => 7
    )
  port map (
    CLK_IN         => NX_MAIN_CLK_IN,
    RESET_IN       => RESET_NX_MAIN_CLK_IN,
    TIMER_START_IN => wait_timer_start,
    TIMER_DONE_OUT => wait_timer_done
    );

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
      wait_timer_start         <= '0';
      timestamp_ctr            <= (others => '0');
      timestamp_hold_o         <= (others => '0');
      timestamp_reset_o        <= '0';
      S_STATE                  <=  S_RESET;
    else
      wait_timer_start         <= '0';
      timestamp_trigger_o      <= '0';
      timestamp_reset_o        <= '0';

      case S_STATE is

        when S_IDLE =>
          timestamp_ctr            <= timestamp_ctr + 1;
          if (timestamp_reset = '0' and timestamp_reset_f = '1') then
            S_STATE                <= S_RESET;
          elsif (TRIGGER_IN = '1') then
            S_STATE                <= S_HOLD;
          else
            S_STATE                <= S_IDLE;
          end if;

        when S_RESET =>
```



```
          timestamp_reset_o        <= '1';
          wait_timer_start         <= '1';
          S_STATE                  <=  S_RESET_WAIT;

        when S_RESET_WAIT =>
          if (wait_timer_done = '0') then
            timestamp_reset_o      <= '1';
            S_STATE                <= S_RESET_WAIT;
          else
            timestamp_ctr          <= (others => '0');
            S_STATE                <= S_IDLE;
          end if;

        when S_HOLD =>
          timestamp_ctr            <= timestamp_ctr + 1;
          timestamp_hold_o         <= timestamp_ctr;
          timestamp_trigger_o      <= '1';
          S_STATE                  <=  S_IDLE;

      end case;

    end if;
  end if;
end process PROC_TIMESTAMP_CTR;

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

TIMESTAMP_RESET_OUT       <= timestamp_reset_o;
TIMESTAMP_HOLD_OUT        <= timestamp_hold_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
    );
  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(31 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(31 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,
                    H_ERASE,
                    H_ERASE_CHANNEL
                    );
  signal H_STATE, H_NEXT_STATE : H_STATES;

  signal address_hist_m          : std_logic_vector(BUS_WIDTH - 1 downto 0);
  signal address_hist_m_x        : std_logic_vector(BUS_WIDTH - 1 downto 0);
  signal data_hist_m             : std_logic_vector(31 downto 0);
  signal data_hist_m_x           : std_logic_vector(31 downto 0);

  signal read_data_hist          : std_logic_vector(31 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(31 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(31 downto 0);
  signal write_enable            : std_logic;

  signal channel_write_busy_o    : std_logic;

  signal erase_counter_x         : unsigned(BUS_WIDTH - 1 downto 0);
  signal erase_counter           : unsigned(BUS_WIDTH - 1 downto 0);

  -- Hist Read Handler
  signal read_address            : std_logic_vector(BUS_WIDTH - 1 downto 0);
  signal read_data               : std_logic_vector(31 downto 0);
  signal read_enable_p           : std_logic;
  signal read_enable             : std_logic;
  signal channel_data_o          : std_logic_vector(31 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;

  signal debug_state_x        : std_logic_vector(2 downto 0);
  signal debug_state          : std_logic_vector(2 downto 0);

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(8)              <= RESET_IN;
  DEBUG_OUT(11 downto 9)    <= debug_state;
  DEBUG_OUT(15 downto 12)   <= channel_data_o(3 downto 0);

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

  SMALL: if (BUS_WIDTH = 7) generate

    ram_dp_COUNTER_HIST: entity work.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_RESULT_HIST: entity work.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
        );
  end generate SMALL;


  LARGE: if (BUS_WIDTH = 9) generate

    ram_dp_COUNTER_HIST: entity work.ram_dp_512x40
      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_RESULT_HIST: entity work.ram_dp_512x32
     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
       );
 end generate LARGE;

 -------------------------------------------------------------------------------
 -- 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');
       erase_counter             <= (others => '0');
       H_STATE                   <= H_ERASE;
       debug_state               <= (others => '0');
     else
       address_hist_m            <= address_hist_m_x;
       data_hist_m               <= data_hist_m_x;
       erase_counter             <= erase_counter_x;
       H_STATE                   <= H_NEXT_STATE;
       debug_state               <= debug_state_x;
     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;
   erase_counter_x             <= erase_counter;

   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;
      debug_state_x <= "001";

    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;
      debug_state_x <= "010";

    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;
      debug_state_x <= "011";
```



```
    when H_ERASE =>
      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';
      erase_counter_x         <= erase_counter + 1;
      read_address_hist       <= (others => '0');
      read_enable_hist        <= '0';
      address_hist_m_x        <= std_logic_vector(erase_counter);
      data_hist_m_x           <= (others => '0');
      channel_write_busy_o    <= '1';
      H_NEXT_STATE            <= H_ERASE_CHANNEL;
      debug_state_x <= "100";

    when H_ERASE_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';
      if (erase_counter > 0) then
        H_NEXT_STATE          <= H_ERASE;
      else
        H_NEXT_STATE          <= H_IDLE;
      end if;
      debug_state_x <= "101";

  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_TS_IN         : in  std_logic_vector(8 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;

  -- Timestamp Histogram
  signal ts_num_averages      : unsigned(2 downto 0);
  signal ts_average_enable    : std_logic;
  signal ts_write_busy        : std_logic;
  signal ts_read_busy         : std_logic;

  signal ts_write_id          : std_logic_vector(8 downto 0);
  signal ts_write_data        : std_logic_vector(31 downto 0);
  signal ts_write             : std_logic;
  signal ts_add               : std_logic;

  signal ts_read_id           : std_logic_vector(8 downto 0);
  signal ts_read              : std_logic;
  signal ts_read_data         : std_logic_vector(31 downto 0);
  signal ts_read_data_valid   : std_logic;

  -- Reset Hists
  signal RESET_HISTS          : 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);

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

 RESET_HISTS    <= RESET_IN or RESET_HISTS_IN;

 nx_histogram_hits: nx_histogram
   generic map (
     BUS_WIDTH => 7
     )
   port map (
     CLK_IN                => CLK_IN,
     RESET_IN              => RESET_HISTS,

     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             => open
     );

 nx_histogram_adc: nx_histogram
   generic map (
     BUS_WIDTH => 7
     )
   port map (
     CLK_IN                => CLK_IN,
     RESET_IN              => RESET_HISTS,

     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
   generic map (
     BUS_WIDTH => 7
     )
   port map (
     CLK_IN                => CLK_IN,
     RESET_IN              => RESET_HISTS,

     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
   generic map (
     BUS_WIDTH => 7
     )
   port map (
     CLK_IN                => CLK_IN,
     RESET_IN              => RESET_HISTS,

     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
     );

 nx_histogram_ts: nx_histogram
   generic map (
     BUS_WIDTH => 9
     )
   port map (
     CLK_IN                => CLK_IN,
```

```
    RESET_IN                => RESET_HISTS,

    NUM_AVERAGES_IN         => ts_num_averages,
    AVERAGE_ENABLE_IN       => ts_average_enable,
    CHANNEL_ID_IN           => ts_write_id,
    CHANNEL_DATA_IN         => ts_write_data,
    CHANNEL_ADD_IN          => ts_add,
    CHANNEL_WRITE_IN        => ts_write,
    CHANNEL_WRITE_BUSY_OUT => ts_write_busy,

    CHANNEL_ID_READ_IN      => ts_read_id,
    CHANNEL_READ_IN         => ts_read,
    CHANNEL_DATA_OUT        => ts_read_data,
    CHANNEL_DATA_VALID_OUT => ts_read_data_valid,
    CHANNEL_READ_BUSY_OUT   => ts_read_busy,

    DEBUG_OUT               => DEBUG_OUT
    );

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

PROC_FILL_HISTOGRAMS: process(CLK_IN)
begin
  if (rising_edge(CLK_IN)) then
    if (RESET_HISTS = '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';

      ts_write_id                   <= (others => '0');
      ts_write_data                 <= (others => '0');
      ts_write                      <= '0';
      ts_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';

      ts_write_id                   <= (others => '0');
      ts_write_data                 <= (others => '0');
      ts_write                      <= '0';
      ts_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;

        if (unsigned(CHANNEL_TS_IN) > 0) then
          ts_write_id                 <= CHANNEL_TS_IN;
          ts_write_data               <= x"0000_0001";
          ts_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_HISTS = '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      <= (others => '0');
      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';

      ts_read_id            <= (others => '0');
      ts_read               <= '0';
      ts_num_averages       <= "000";
      ts_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';

      ts_read_id            <= (others => '0');
      ts_read               <= '0';

      if (hit_read_busy    = '1' or
          adc_read_busy    = '1' or
          pileup_read_busy = '1' or
          ovfl_read_busy   = '1' or
          ts_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';
        elsif (ts_read_data_valid = '1') then
```



```
          slv_data_out_o                <= ts_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';
        elsif (unsigned(SLV_ADDR_IN) >= x"0400" and
               unsigned(SLV_ADDR_IN) <= x"05ff") then
          ts_read_id                    <= SLV_ADDR_IN(8 downto 0);
          ts_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 x"0600" =>
              slv_data_out_o(2 downto 0)  <=
                std_logic_vector(ts_num_averages);
              slv_data_out_o(31 downto 3) <= (others => '0');
              slv_ack_o                   <= '1';

            when x"0601" =>
              slv_data_out_o(0)           <= ts_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 x"0600" =>
            ts_num_averages               <= SLV_DATA_IN(2 downto 0);
            slv_ack_o                     <= '1';

          when x"0601" =>
            ts_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_i;
  DEBUG_OUT(15)           <= scl_i;
  --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_register_setup is
  port(
    CLK_IN               : in  std_logic;
    RESET_IN             : in  std_logic;

    I2C_ONLINE_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_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);
    NX_CLOCK_ON_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_register_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_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_3_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;

  -- 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;

  -- Status
  signal nx_clock_on_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_disable_memory; --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_r;
  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_3_clear;
  DEBUG_OUT(11)           <= i2c_command(31);
  DEBUG_OUT(12)           <= i2c_lock_0;
  DEBUG_OUT(13)           <= i2c_lock_1;
  DEBUG_OUT(14)           <= i2c_lock_2;
  DEBUG_OUT(15)           <= i2c_lock_3;

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

  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(3 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_command     <= (others => '0');
      else
        i2c_command     <= (others => '0');
        locks           := 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_command_busy_o = '0') then
        if (nx_i2c_command(31)   = '1'     and
            ((locks and "1110")  = "0000") 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 "1011")       = "0000") 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 "1101")      = "0000") and
               i2c_lock_1_clear         = '0') then
          i2c_command       <= dac_read_i2c_command;
          i2c_lock_1        <= '1';
        elsif (adc_i2c_command(31)  = '1'     and
               ((locks and "0111")  = "0000") and
               i2c_lock_3_clear     = '0') then
          i2c_command       <= adc_i2c_command;
          i2c_lock_3        <= '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_3_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_3_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_3 = '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_3_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_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_IN;
      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_IN;
              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 ON
            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_IN;
            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
-------------------------------------------------------------------------------
nx_clock_on_o           <= i2c_ram(33)(3) when rising_edge(CLK_IN);

I2C_COMMAND_OUT         <= i2c_command_o;
I2C_LOCK_OUT            <= i2c_command_busy_o;

SPI_COMMAND_OUT         <= (others => '0');
SPI_LOCK_OUT            <= '0';

-- Internal Read
INT_ACK_OUT             <= int_ack_o;
INT_DATA_OUT            <= int_data_o;

NX_CLOCK_ON_OUT         <= nx_clock_on_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_status 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;
    PLL_RESET_OUT         : out std_logic;

    -- Signals
    I2C_SM_RESET_OUT      : inout std_logic;
    I2C_REG_RESET_OUT     : out std_logic;
    NX_ONLINE_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_status is

  -- Offline Handler

  signal i2c_sm_reset_i_x        : std_logic;
  signal i2c_sm_reset_i          : std_logic;
  signal i2c_sm_online           : std_logic;
  signal i2c_sm_online_ctr       : unsigned(8 downto 0);

  signal offline_force           : std_logic;
  signal online_o                : std_logic;
  signal online_trigger          : std_logic;
  signal online_last             : std_logic;

  -- Reset Handler
  signal i2c_sm_reset_start      : std_logic;
  signal i2c_reg_reset_start     : std_logic;

  signal i2c_sm_reset_o          : std_logic;
  signal i2c_reg_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
                  );

  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;
  signal pll_reset_p              : std_logic;
  signal pll_reset_o              : 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)            <= '0';
  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_sm_online;
  DEBUG_OUT(11)           <= offline_force;
  DEBUG_OUT(12)           <= online_o;
  DEBUG_OUT(13)           <= i2c_sm_reset_i;
  DEBUG_OUT(14)           <= pll_reset_o;
  DEBUG_OUT(15)           <= online_trigger;
```

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

signal_async_trans_i2c_sm_reset_i: signal_async_trans
  port map (
    CLK_IN      => CLK_IN,
    SIGNAL_A_IN => i2c_sm_reset_i_x,
    SIGNAL_OUT  => i2c_sm_reset_i
    );

PROC_I2C_OFFLINE_SCHMITT_TRIGGER: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      i2c_sm_online       <= '0';
      i2c_sm_online_ctr   <= (others => '0');
    else
      if (i2c_sm_reset_i = '1') then
        if (i2c_sm_online_ctr < x"1ff") then
          i2c_sm_online_ctr  <= i2c_sm_online_ctr + 1;
        end if;
      else
        if (i2c_sm_online_ctr > x"000") then
          i2c_sm_online_ctr  <= i2c_sm_online_ctr - 1;
        end if;
      end if;

      if (i2c_sm_online_ctr > x"1d6") then
        i2c_sm_online        <= '1';
      elsif (i2c_sm_online_ctr < x"01e") then
        i2c_sm_online        <= '0';
      end if;
    end if;
  end if;
end process PROC_I2C_OFFLINE_SCHMITT_TRIGGER;

PROC_NXYTER_OFFLINE: process(CLK_IN)
  variable online_state : std_logic_vector(1 downto 0) := "00";
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      online_trigger    <= '0';
      online_o          <= '1';
      online_last       <= '0';
    else
      if (i2c_sm_online = '1' and
          offline_force = '0' and
          pll_nx_clk_lock = '1') then
```



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

      -- Offline State changes
      online_last       <= online_o;
      online_state      := online_o & online_last;

      case online_state is
        when  "01" | "10" =>
          online_trigger <= '1';

        when others =>
          online_trigger <= '0';
      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';
      STATE              <= S_IDLE;
    else
      i2c_sm_reset_o     <= '0';
      i2c_reg_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;
          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;

      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';
      offline_force              <= '0';
      nx_data_clk_dphase_o       <= x"7";
      nx_data_clk_finedelb_o     <= x"0";
      clear_notlock_counters     <= '0';
      pll_reset_p                <= '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';
      clear_notlock_counters     <= '0';
      pll_reset_p                <= '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" =>
            slv_ack_o                   <= '1';

          when x"0003" =>
            offline_force               <= SLV_DATA_IN(0);
            slv_ack_o                   <= '1';

          when x"0006" =>
            pll_reset_p                 <= '1';
            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"0000" =>
            slv_data_out_o(0)           <= i2c_sm_reset_i;
            slv_data_out_o(31 downto 1) <= (others => '0');
            slv_ack_o                   <= '1';

          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_sm_online;
            slv_data_out_o(31 downto 1) <= (others => '0');
            slv_ack_o                   <= '1';

          when x"0005" =>
            slv_data_out_o(0)           <= online_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;

 ------------------------------------------------------------------------------
 pulse_to_level_1: pulse_to_level
   generic map (
     NUM_CYCLES => 15)
   port map (
     CLK_IN    => CLK_IN,
     RESET_IN  => RESET_IN,
     PULSE_IN  => pll_reset_p,
     LEVEL_OUT => pll_reset_o
     );

 -- Output Signals
 i2c_sm_reset_i_x      <= I2C_SM_RESET_OUT;

 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;

 PLL_RESET_OUT         <= pll_reset_o;
 I2C_SM_RESET_OUT      <= '0' when i2c_sm_reset_o = '1' else 'Z';
 I2C_REG_RESET_OUT     <= i2c_reg_reset_o;
 NX_ONLINE_OUT         <= online_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";
    VERSION_NUMBER            : std_logic_vector(3 downto 0) := x"1"
    );
  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_HEADER,
                    E_READ_NEXT,
                    E_READ,
                    E_NEXT_INDEX,
                    E_TRAILER,
                    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"
 --    );

  -- For the moment just the 4 I2C ADC Values, event must be small
  constant NUM_REGS : integer      := 1;
  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"0080"
     );
  constant reg_addr_end            : reg_addr_t :=
    (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_HEADER;
          else
            E_STATE                 <= E_IDLE;
          end if;
```



```
        when E_HEADER =>
          fee_data_o(25 downto 0)    <= (others => '1');
          fee_data_o(29 downto 26)   <= VERSION_NUMBER;
          fee_data_o(31 downto 30)   <= BOARD_ID;
          fee_data_write_o           <= '1';
          E_STATE                    <= E_NEXT_INDEX;

        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_TRAILER;
          end if;

        when E_TRAILER =>
          fee_data_o                 <= (others => '1');
          fee_data_write_o           <= '1';
          E_STATE                    <= E_END;

        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;

    TRIGGER_BUSY_IN        : in  std_logic;
    EXTERNAL_TRIGGER_OUT   : out std_logic;
    INTERNAL_TRIGGER_OUT   : out std_logic;

    DATA_IN                : in  std_logic_vector(43 downto 0);
    DATA_CLK_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_trigger_generator is

  -- Internal Trigger Generator
  signal pulser_trigger_on         : std_logic;
  signal pulser_trigger_period     : unsigned(27 downto 0);
  signal pulser_trigger            : std_logic;

  -- Self Trigger Generator
  signal data_i_f                  : std_logic_vector(43 downto 0);
  signal data_i                    : std_logic_vector(43 downto 0);
  signal data_clk_i_f              : std_logic;
  signal data_clk_i                : std_logic;

  signal self_trigger_on           : std_logic;
  signal self_trigger              : std_logic;

  -- Trigger Outputs
  signal trigger_output_select     : std_logic;  -- 0: Ext 1: Intern
  signal external_trigger_i        : std_logic;
  signal external_trigger_o        : std_logic;
  signal internal_trigger_o        : std_logic;
  signal trigger                   : std_logic;

  type S_STATES is (S_IDLE,
                    S_BUSY
                    );
  signal S_STATE : S_STATES;

  signal external_trigger_on       : std_logic;
  signal external_trigger_ctr      : unsigned(4 downto 0);
  signal external_trigger_busy     : std_logic;
  signal external_trigger          : std_logic;

  -- Rate Calculation
  signal self_trigger_rate_t       : unsigned(27 downto 0);
  signal self_trigger_rate         : unsigned(27 downto 0);
  signal pulser_trigger_rate_t     : unsigned(27 downto 0);
  signal pulser_trigger_rate       : unsigned(27 downto 0);
  signal trigger_rate              : unsigned(27 downto 0);
  signal 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 pulser_trigger_period_r   : unsigned(27 downto 0);

begin
  -- Debug Line
  DEBUG_OUT(0)             <= CLK_IN;
  DEBUG_OUT(1)             <= '0';
```



```
  DEBUG_OUT(2)             <= DATA_CLK_IN;
  DEBUG_OUT(3)             <= TRIGGER_BUSY_IN;

  DEBUG_OUT(4)             <= self_trigger_on;
  DEBUG_OUT(5)             <= self_trigger;

  DEBUG_OUT(6)             <= pulser_trigger_on;
  DEBUG_OUT(7)             <= pulser_trigger;

  DEBUG_OUT(8)             <= external_trigger_busy;
  DEBUG_OUT(9)             <= external_trigger;

  DEBUG_OUT(10)            <= internal_trigger_o;
  DEBUG_OUT(11)            <= external_trigger_o;
  DEBUG_OUT(12)            <= trigger_output_select;
  DEBUG_OUT(13)            <= trigger;
  DEBUG_OUT(15 downto 14)  <= (others => '0');

  ------------------------------------------------------------------------------
  -- Generate Pulser Trigger
  ------------------------------------------------------------------------------

  timer_PULSER_TRIGGER: timer
    generic map (
      CTR_WIDTH => 28
      )
    port map (
      CLK_IN         => CLK_IN,
      RESET_IN       => RESET_IN,
      TIMER_START_IN => pulser_trigger_on,
      TIMER_END_IN   => pulser_trigger_period,

      TIMER_DONE_OUT => pulser_trigger
      );

  pulser_trigger_period <= (pulser_trigger_period_r - 1)
                           when pulser_trigger_period_r > 10
                           else x"0000009";

  ------------------------------------------------------------------------------
  -- Self Trigger
  ------------------------------------------------------------------------------

  PROC_SELF_TRIGGER: process(CLK_IN)
    variable frame_bits : std_logic_vector(3 downto 0);
  begin
    if( rising_edge(CLK_IN) ) then
      data_i_f      <= DATA_IN;
      data_i        <= data_i_f;
      data_clk_i_f  <= DATA_CLK_IN;
      data_clk_i    <= data_clk_i_f;

      if( RESET_IN = '1' ) then
        self_trigger       <= '0';
      else
        frame_bits      := data_i(31) &
                           data_i(23) &
                           data_i(15) &
                           data_i(7);
        self_trigger       <= '0';

        if (self_trigger_on = '1' and
```

```
          data_clk_i      = '1' and
          frame_bits      = "1000") then
        self_trigger     <= '1';
      else
        self_trigger     <= '0';
      end if;
    end if;
  end if;
end process PROC_SELF_TRIGGER;

-------------------------------------------------------------------------------
-- Trigger Output Handler
-------------------------------------------------------------------------------

PROC_TRIGGER_OUTPUTS: process (CLK_IN)
  variable trigger_signals : std_logic;
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      external_trigger_i     <= '0';
      internal_trigger_o     <= '0';
    else
      trigger_signals        := self_trigger or pulser_trigger;

      if (trigger_output_select = '0') then
        external_trigger_i   <= trigger_signals;
      else
        internal_trigger_o   <= trigger_signals;
      end if;

      -- For Rate Counter
      trigger                <= external_trigger or internal_trigger_o;
    end if;
  end if;
end process PROC_TRIGGER_OUTPUTS;

PROC_EXTERN_TRIGGER_OUT: process(CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if( RESET_IN = '1' ) then
      external_trigger_ctr    <= (others => '0');
      external_trigger_busy   <= '0';
      external_trigger        <= '0';
    else
      external_trigger             <= '0';
      external_trigger_busy        <= '0';

      case S_STATE is
        when S_IDLE =>
          if (TRIGGER_BUSY_IN = '0' and
              external_trigger_i = '1') then
            external_trigger_ctr   <= "10100";  -- 20
            external_trigger       <= '1';
            S_STATE                <= S_BUSY;
          else
            external_trigger_ctr   <= (others => '0');
            S_STATE                <= S_IDLE;
          end if;

        when S_BUSY =>
          if (external_trigger_ctr > 0) then
            external_trigger_ctr   <= external_trigger_ctr  - 1;
```



```
            external_trigger_busy  <= '1';
            S_STATE                <= S_BUSY;
          else
            S_STATE                <= S_IDLE;
          end if;
      end case;

    end if;
  end if;
end process PROC_EXTERN_TRIGGER_OUT;

-- Goes to CTS
pulse_to_level_EXTERNAL_TRIGGER: pulse_to_level
  generic map (
    NUM_CYCLES => 8
    )
  port map (
    CLK_IN    => CLK_IN,
    RESET_IN  => RESET_IN,
    PULSE_IN  => external_trigger,
    LEVEL_OUT => external_trigger_o
    );

-------------------------------------------------------------------------------
--  Rate Counter
-------------------------------------------------------------------------------
PROC_CAL_RATES: process (CLK_IN)
begin
  if( rising_edge(CLK_IN) ) then
    if (RESET_IN = '1') then
      self_trigger_rate_t        <= (others => '0');
      self_trigger_rate          <= (others => '0');
      pulser_trigger_rate_t      <= (others => '0');
      pulser_trigger_rate        <= (others => '0');
      trigger_rate_t             <= (others => '0');
      trigger_rate               <= (others => '0');
      rate_timer                 <= (others => '0');
    else
      if (rate_timer < x"5f5e100") then
        if (self_trigger = '1') then
          self_trigger_rate_t     <= self_trigger_rate_t + 1;
        end if;

        if (pulser_trigger = '1') then
          pulser_trigger_rate_t   <= pulser_trigger_rate_t + 1;
        end if;

        if (trigger = '1') then
          trigger_rate_t          <= trigger_rate_t + 1;
        end if;

        rate_timer                <= rate_timer + 1;
      else
        self_trigger_rate_t(27 downto 1)    <= (others => '0');
        self_trigger_rate_t(0)              <= '0';

        pulser_trigger_rate_t(27 downto 1)  <= (others => '0');
        pulser_trigger_rate_t(0)            <= pulser_trigger;

        trigger_rate_t(27 downto 1)         <= (others => '0');
        trigger_rate_t(0)                   <= trigger;
```

```
        self_trigger_rate                 <= self_trigger_rate_t;
        pulser_trigger_rate               <= pulser_trigger_rate_t;
        trigger_rate                      <= trigger_rate_t;

        rate_timer                        <= (others => '0');
      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';
      trigger_output_select      <= '0';
      self_trigger_on            <= '1';
      pulser_trigger_on          <= '0';
      pulser_trigger_period_r    <= x"00186a0";
    else
      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" =>
            self_trigger_on              <= SLV_DATA_IN(0);
            pulser_trigger_on            <= SLV_DATA_IN(1);
            trigger_output_select        <= SLV_DATA_IN(2);
            slv_ack_o                    <= '1';

          when x"0001" =>
            pulser_trigger_period_r      <=
              unsigned(SLV_DATA_IN(27 downto 0));
            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(0)            <= self_trigger_on;
            slv_data_out_o(1)            <= pulser_trigger_on;
            slv_data_out_o(2)            <= trigger_output_select;
            slv_data_out_o(31 downto 3)  <= (others => '0');
            slv_ack_o                    <= '1';

          when x"0001" =>
```



```
            slv_data_out_o(27 downto 0)   <=
              std_logic_vector(pulser_trigger_period_r);
            slv_data_out_o(31 downto 28)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0002" =>
            slv_data_out_o(27 downto 0)   <=
              std_logic_vector(self_trigger_rate);
            slv_data_out_o(31 downto 28)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0003" =>
            slv_data_out_o(27 downto 0)   <=
              std_logic_vector(pulser_trigger_rate);
            slv_data_out_o(31 downto 28)  <= (others => '0');
            slv_ack_o                     <= '1';

          when x"0004" =>
            slv_data_out_o(27 downto 0)   <=
              std_logic_vector(trigger_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
------------------------------------------------------------------------------

-- Trigger Output
EXTERNAL_TRIGGER_OUT     <= external_trigger_o;
INTERNAL_TRIGGER_OUT     <= internal_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;
    TRIGGER_CALIBRATION_OUT    : out std_logic;
    FAST_CLEAR_OUT             : out std_logic;
    TRIGGER_BUSY_OUT           : out std_logic;

    -- Pulser
    NX_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_ff   : std_logic;
  signal invalid_timing_trigger_f    : std_logic;
  signal invalid_timing_trigger      : std_logic;
  signal invalid_timing_trigger_ctr  : unsigned(15 downto 0);

  signal trigger_busy_ff             : std_logic;
  signal trigger_busy_f              : std_logic;
  signal trigger_busy                : std_logic;

  signal fast_clear_ff               : std_logic;
  signal fast_clear_f                : 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 calibration_trigger_o       : std_logic;
  signal calib_downscale_ctr         : unsigned(15 downto 0);
  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 testpulse_trigger           : std_logic;

  signal testpulse_enable            : std_logic;

  signal timestamp_calib_trigger_c100 : std_logic;
  signal timestamp_calib_trigger_f    : std_logic;
  signal timestamp_calib_trigger_o    : std_logic;

  type STATES is (S_IDLE,
                  S_IGNORE_TRIGGER,
                  S_STATUS_TRIGGER,
                  S_TIMING_TRIGGER,
```

```
                  S_CALIBRATION_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_TIMING,
                       T_STATUS,
                       T_CALIBRATION
                       );
signal TRIGGER_TYPE : TRIGGER_TYPES;


-- Testpulse Handler
type T_STATES is (T_IDLE,
                  T_WAIT_TESTPULE_DELAY,
                  T_SET_TESTPULSE,
                  T_WAIT_TESTPULE_END
                  );

signal T_STATE : T_STATES;

signal start_testpulse             : std_logic;
signal testpulse_delay             : unsigned(11 downto 0);
signal testpulse_length            : unsigned(11 downto 0);
signal testpulse_o                 : std_logic;
signal wait_timer_reset            : std_logic;
signal wait_timer_start            : std_logic;
signal wait_timer_done             : std_logic;
signal wait_timer_end              : unsigned(11 downto 0);
signal internal_trigger_f          : std_logic;
signal internal_trigger            : std_logic;

-- Rate Calculation
signal start_testpulse_ff          : std_logic;
signal start_testpulse_f           : std_logic;

signal accepted_trigger_rate_t     : unsigned(27 downto 0);
signal start_testpulse_clk100      : std_logic;
signal testpulse_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_length        : unsigned(11 downto 0);
signal reg_testpulse_enable        : std_logic;
signal accepted_trigger_rate       : unsigned(27 downto 0);
signal testpulse_rate              : unsigned(27 downto 0);
signal invalid_t_trigger_ctr_clear : std_logic;
signal bypass_all_trigger          : std_logic;
```



```
  signal bypass_physics_trigger      : std_logic;
  signal bypass_status_trigger       : std_logic;
  signal bypass_calibration_trigger  : std_logic;
  signal calibration_downscale       : unsigned(15 downto 0);
  signal physics_trigger_type        : std_logic_vector(3 downto 0);
  signal status_trigger_type         : std_logic_vector(3 downto 0);
  signal calibration_trigger_type    : std_logic_vector(3 downto 0);

  -- Reset
  signal reset_nx_main_clk_in_ff     : std_logic;
  signal reset_nx_main_clk_in_f      : std_logic;
  signal RESET_NX_MAIN_CLK_IN        : std_logic;

  attribute syn_keep : boolean;
  attribute syn_keep of reset_nx_main_clk_in_ff     : signal is true;
  attribute syn_keep of reset_nx_main_clk_in_f      : signal is true;

  attribute syn_keep of trigger_busy_ff             : signal is true;
  attribute syn_keep of trigger_busy_f              : signal is true;

  attribute syn_keep of fast_clear_ff               : signal is true;
  attribute syn_keep of fast_clear_f                : signal is true;

  attribute syn_keep of internal_trigger_f          : signal is true;
  attribute syn_keep of internal_trigger            : signal is true;

  attribute syn_keep of start_testpulse_ff          : signal is true;
  attribute syn_keep of start_testpulse_f           : signal is true;

  attribute syn_keep of timestamp_calib_trigger_f   : signal is true;
  attribute syn_keep of timestamp_calib_trigger_o   : signal is true;

  attribute syn_preserve : boolean;
  attribute syn_preserve of reset_nx_main_clk_in_ff : signal is true;
  attribute syn_preserve of reset_nx_main_clk_in_f  : signal is true;

  attribute syn_preserve of trigger_busy_ff         : signal is true;
  attribute syn_preserve of trigger_busy_f          : signal is true;

  attribute syn_preserve of fast_clear_ff           : signal is true;
  attribute syn_preserve of fast_clear_f            : signal is true;

  attribute syn_preserve of internal_trigger_f      : signal is true;
  attribute syn_preserve of internal_trigger        : signal is true;

  attribute syn_preserve of start_testpulse_ff      : signal is true;
  attribute syn_preserve of start_testpulse_f       : signal is true;

  attribute syn_preserve of timestamp_calib_trigger_f  : signal is true;
  attribute syn_preserve of timestamp_calib_trigger_o  : signal is true;

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)           <= testpulse_trigger;
  DEBUG_OUT(15)           <= testpulse_o;

  ------------------------------------------------------------------------------
  -- Reset Domain Transfer
  ------------------------------------------------------------------------------
  reset_nx_main_clk_in_ff   <= RESET_IN when rising_edge(NX_MAIN_CLK_IN);
  reset_nx_main_clk_in_f    <= reset_nx_main_clk_in_ff
                               when rising_edge(NX_MAIN_CLK_IN);
  RESET_NX_MAIN_CLK_IN      <= reset_nx_main_clk_in_f
                               when rising_edge(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
      );


  -- Signal Domain Transfers to NX Clock
  trigger_busy_ff  <= trigger_busy_o
                      when rising_edge(NX_MAIN_CLK_IN);
  trigger_busy_f   <= trigger_busy_ff
                      when rising_edge(NX_MAIN_CLK_IN);
  trigger_busy     <= trigger_busy_f
                      when rising_edge(NX_MAIN_CLK_IN);

  fast_clear_ff    <= fast_clear_o
                      when rising_edge(NX_MAIN_CLK_IN);
  fast_clear_f     <= fast_clear_ff
                      when rising_edge(NX_MAIN_CLK_IN);
  fast_clear       <= fast_clear_f
                      when rising_edge(NX_MAIN_CLK_IN);

  testpulse_enable <= reg_testpulse_enable when rising_edge(NX_MAIN_CLK_IN);

  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';
        testpulse_trigger          <= '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';
        testpulse_trigger          <= '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
                    testpulse_trigger    <= '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;

-- Relax Timing
invalid_timing_trigger_ff  <= invalid_timing_trigger_n
                              when rising_edge(NX_MAIN_CLK_IN);
invalid_timing_trigger_f   <= invalid_timing_trigger_ff
                              when rising_edge(NX_MAIN_CLK_IN);

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_f,
    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';
      calibration_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';
      timestamp_calib_trigger_c100 <= '0';
      calib_downscale_ctr         <= (others => '0');
      TRIGGER_TYPE                <= T_UNDEF;
      STATE                       <= S_IDLE;
    else
      valid_trigger_o             <= '0';
      timing_trigger_o            <= '0';
      status_trigger_o            <= '0';
      calibration_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';
      timestamp_calib_trigger_c100 <= '0';

      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_TIMING_TRG_IN = '1') then
              -- Timing Trigger IN
              if (NXYTER_OFFLINE_IN          = '1' or
                  bypass_all_trigger         = '1') then

                -- Ignore Trigger for nxyter is or pretends to be offline
                TRIGGER_TYPE                    <= T_IGNORE;
                STATE                           <= S_IGNORE_TRIGGER;
              else
                -- Check Trigger Type
                if (LVL1_TRG_TYPE_IN = physics_trigger_type) then
```

```
                -- Physiks Trigger
                if (bypass_physics_trigger = '1') then
                  TRIGGER_TYPE                <= T_IGNORE;
                  STATE                       <= S_IGNORE_TRIGGER;
                else
                  TRIGGER_TYPE                <= T_TIMING;
                  STATE                       <= S_TIMING_TRIGGER;
                end if;
              else
                -- Unknown Timing Trigger, ignore
                TRIGGER_TYPE                  <= T_IGNORE;
                STATE                         <= S_IGNORE_TRIGGER;
              end if;
            end if;

          elsif (LVL1_VALID_NOTIMING_TRG_IN = '1') then
            -- No Timing Trigger IN
            if (NXYTER_OFFLINE_IN          = '1' or
                bypass_all_trigger         = '1') then

              -- Ignore Trigger for nxyter is or pretends to be offline
              TRIGGER_TYPE                    <= T_IGNORE;
              STATE                           <= S_IGNORE_TRIGGER;
            else
              -- Check Trigger Type
              if (LVL1_TRG_TYPE_IN = calibration_trigger_type) then
                -- Calibration Trigger
                if (bypass_calibration_trigger = '1') then
                  TRIGGER_TYPE                <= T_IGNORE;
                  STATE                       <= S_IGNORE_TRIGGER;
                else
                  if (calib_downscale_ctr >= calibration_downscale) then
                    timestamp_calib_trigger_c100 <= '1';
                    calib_downscale_ctr       <= x"0001";
                    TRIGGER_TYPE              <= T_CALIBRATION;
                    STATE                     <= S_CALIBRATION_TRIGGER;
                  else
                    calib_downscale_ctr       <= calib_downscale_ctr + 1;
                    TRIGGER_TYPE              <= T_IGNORE;
                    STATE                     <= S_IGNORE_TRIGGER;
                  end if;
                end if;

              elsif (LVL1_TRG_TYPE_IN = status_trigger_type) then
                -- Status Trigger
                if (bypass_status_trigger = '1') then
                  TRIGGER_TYPE                <= T_IGNORE;
                  STATE                       <= S_IGNORE_TRIGGER;
                else
                  -- Status Trigger
                  status_trigger_o            <= '1';
                  TRIGGER_TYPE                <= T_STATUS;
                  STATE                       <= S_STATUS_TRIGGER;
                end if;

              else
                -- Some other Trigger, ignore it
                TRIGGER_TYPE                  <= T_IGNORE;
                STATE                         <= S_IGNORE_TRIGGER;
              end if;

            end if;
```



```
          else
            -- No Trigger IN, Nothing to do, Sleep Well
            trigger_busy_o        <= '0';
            TRIGGER_TYPE          <= T_UNDEF;
            STATE                 <= S_IDLE;
          end if;

        when S_TIMING_TRIGGER =>
          valid_trigger_o         <= '1';
          timing_trigger_o        <= '1';
          STATE                   <= S_WAIT_TRG_DATA_VALID;

        when S_CALIBRATION_TRIGGER =>
          calibration_trigger_o   <= '1';
          valid_trigger_o         <= '1';
          timing_trigger_o        <= '1';
          STATE                   <= S_WAIT_TRG_DATA_VALID;

        when S_WAIT_TRG_DATA_VALID | S_STATUS_TRIGGER | S_IGNORE_TRIGGER =>
          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_TYPE = T_TIMING or
                TRIGGER_TYPE = T_CALIBRATION)
               and TRIGGER_BUSY_0_IN = '1')
              or
              (TRIGGER_TYPE = T_STATUS  and
               TRIGGER_BUSY_1_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 =>
      fee_data_o                 <= (others => '0');
      fee_data_write_o           <= '0';

    when T_TIMING | T_CALIBRATION =>
      fee_data_o                 <= FEE_DATA_0_IN;
      fee_data_write_o           <= FEE_DATA_WRITE_0_IN;

    when T_STATUS =>
      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   => wait_timer_end,
    TIMER_DONE_OUT => wait_timer_done
    );

testpulse_delay     <= reg_testpulse_delay when rising_edge(NX_MAIN_CLK_IN);
testpulse_length    <= reg_testpulse_length when rising_edge(NX_MAIN_CLK_IN);

internal_trigger_f  <= INTERNAL_TRIGGER_IN or
                       calibration_trigger_o when rising_edge(NX_MAIN_CLK_IN);
internal_trigger    <= internal_trigger_f  when rising_edge(NX_MAIN_CLK_IN);

start_testpulse     <= testpulse_trigger or
                       internal_trigger;

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';
      testpulse_o          <= '0';
      T_STATE              <= T_IDLE;
```



```
    else
      wait_timer_start     <= '0';
      wait_timer_reset     <= '0';
      testpulse_o          <= '0';

      if (fast_clear = '1') then
        wait_timer_reset   <= '1';
        T_STATE            <= T_IDLE;
      else
        case T_STATE is

          when T_IDLE =>
            if (start_testpulse = '1') then
              if (reg_testpulse_delay > 0) then
                wait_timer_end    <= testpulse_delay;
                wait_timer_start  <= '1';
                T_STATE           <= T_WAIT_TESTPULE_DELAY;
              else
                T_STATE           <= T_SET_TESTPULSE;
              end if;
            else
              T_STATE             <= T_IDLE;
            end if;

          when T_WAIT_TESTPULE_DELAY =>
            if (wait_timer_done = '0') then
              T_STATE             <= T_WAIT_TESTPULE_DELAY;
            else
              T_STATE             <= T_SET_TESTPULSE;
            end if;

          when T_SET_TESTPULSE =>
            testpulse_o           <= '1';
            wait_timer_end        <= testpulse_length;
            wait_timer_start      <= '1';
            T_STATE               <= T_WAIT_TESTPULE_END;

          when T_WAIT_TESTPULE_END =>
            if (wait_timer_done = '0') then
              testpulse_o         <= '1';
              T_STATE             <= T_WAIT_TESTPULE_END;
            else
              T_STATE             <= T_IDLE;
            end if;

        end case;
      end if;
    end if;
  end if;
end process PROC_TESTPULSE_HANDLER;

-- Relax Timing
  start_testpulse_ff <= start_testpulse    when rising_edge(NX_MAIN_CLK_IN);
  start_testpulse_f  <= start_testpulse_ff when rising_edge(NX_MAIN_CLK_IN);

  pulse_dtrans_TESTPULSE_RATE: 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  => start_testpulse_f,
      CLK_B_IN    => CLK_IN,
      RESET_B_IN  => RESET_IN,
      PULSE_B_OUT => start_testpulse_clk100
      );

  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');
        testpulse_rate_t            <= (others => '0');
        testpulse_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;

          if (start_testpulse_clk100 = '1') then
            testpulse_rate_t                  <= testpulse_rate_t + 1;
          end if;
          rate_timer                          <= rate_timer + 1;
        else
          rate_timer                          <= (others => '0');
          accepted_trigger_rate               <= accepted_trigger_rate_t;
          testpulse_rate                      <= testpulse_rate_t;

          accepted_trigger_rate_t             <= (others => '0');
          testpulse_rate_t                    <= (others => '0');
        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_length          <= x"064";
        reg_testpulse_enable          <= '0';
        invalid_t_trigger_ctr_clear   <= '1';
        bypass_all_trigger            <= '0';
        bypass_physics_trigger        <= '0';
        bypass_status_trigger         <= '1';
        bypass_calibration_trigger    <= '1';
        calibration_downscale         <= x"0001";
        physics_trigger_type          <= x"1";
        calibration_trigger_type      <= x"9";
        status_trigger_type           <= x"e";
      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   <= '0';

        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"0002" =>
              reg_testpulse_length        <=
                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 x"0006" =>
              bypass_physics_trigger      <= SLV_DATA_IN(0);
              bypass_status_trigger       <= SLV_DATA_IN(1);
              bypass_calibration_trigger  <= SLV_DATA_IN(2);
              bypass_all_trigger          <= SLV_DATA_IN(3);
              slv_ack_o                   <= '1';

            when x"0007" =>
              if (unsigned(SLV_DATA_IN(15 downto 0)) > x"0000") then
                calibration_downscale     <=
                  unsigned(SLV_DATA_IN(15 downto 0));
              end if;
              slv_ack_o                   <= '1';

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

            when x"0009" =>
              status_trigger_type         <= SLV_DATA_IN(3 downto 0);
              slv_ack_o                   <= '1';

            when x"000a" =>
              calibration_trigger_type    <= SLV_DATA_IN(3 downto 0);
              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(11 downto 0)  <=
              std_logic_vector(reg_testpulse_length);
            slv_data_out_o(31 downto 12) <= (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 x"0004" =>
            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"0005" =>
            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 x"0006" =>
            slv_data_out_o(0)            <= bypass_physics_trigger;
            slv_data_out_o(1)            <= bypass_status_trigger;
            slv_data_out_o(2)            <= bypass_calibration_trigger;
            slv_data_out_o(3)            <= bypass_all_trigger;
            slv_data_out_o(31 downto 4)  <= (others => '0');
            slv_ack_o                    <= '1';

          when x"0007" =>
            slv_data_out_o(15 downto 0)  <= calibration_downscale;
            slv_data_out_o(31 downto 16) <= (others => '0');
            slv_ack_o                    <= '1';

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

          when x"0009" =>
            slv_data_out_o(3 downto 0)   <= status_trigger_type;
            slv_data_out_o(31 downto 4)  <= (others => '0');
            slv_ack_o                    <= '1';

          when x"000a" =>
            slv_data_out_o(3 downto 0)   <= calibration_trigger_type;
            slv_data_out_o(31 downto 4)  <= (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
-------------------------------------------------------------------------------

  timestamp_calib_trigger_f  <= timestamp_calib_trigger_c100
                                when rising_edge(NX_MAIN_CLK_IN);

  timestamp_calib_trigger_o  <= timestamp_calib_trigger_f
                                when rising_edge(NX_MAIN_CLK_IN);

-- Trigger Output
  VALID_TRIGGER_OUT          <= valid_trigger_o;
  TIMESTAMP_TRIGGER_OUT      <= timestamp_trigger_o or timestamp_calib_trigger_o;
  TRIGGER_TIMING_OUT         <= timing_trigger_o;
  TRIGGER_STATUS_OUT         <= status_trigger_o;
  TRIGGER_CALIBRATION_OUT    <= calibration_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;

  NX_TESTPULSE_OUT           <= 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";
    VERSION_NUMBER           : std_logic_vector(3 downto 0) := x"1"
    );
  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_CALIBRATION_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_RESET_OUT    : out 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_TS_OUT       : out std_logic_vector(8 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 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_nr      : unsigned(31 downto 0);
  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;
  signal trigger_calibration   : 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_ff     : unsigned(11 downto 0);
  signal timestamp_fpga_f      : 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_ts_o        : std_logic_vector(8 downto 0);
  signal histogram_pileup_o    : std_logic;
  signal histogram_ovfl_o      : std_logic;

  signal histogram_ts_range    : std_logic_vector(2 downto 0);

  -- 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;

  signal histogram_trig_filter : std_logic;
  signal histogram_limits      : std_logic;
  signal histogram_lower_limit : unsigned(13 downto 0);
  signal histogram_upper_limit : unsigned(13 downto 0);
  signal reset_hists           : std_logic;
  signal reset_hists_o         : std_logic;
```



```
  -- Timestamp Trigger Window Settings
  signal nxyter_cv_time            : unsigned(11 downto 0);
  signal cts_trigger_delay         : unsigned(11 downto 0);
  signal trigger_calibration_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);

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

  attribute syn_keep : boolean;
  attribute syn_keep of timestamp_fpga_ff     : signal is true;
  attribute syn_keep of timestamp_fpga_f      : signal is true;

  attribute syn_preserve : boolean;
  attribute syn_preserve of timestamp_fpga_ff : signal is true;
  attribute syn_preserve of timestamp_fpga_f  : signal is true;

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)           <= TIMESTAMP_STATUS_IN(S_PARITY);
  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 cts_trigger_delay_tmp     : unsigned(11 downto 0);
  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 histTStore                : unsigned(8 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');

      histogram_fill_o        <= '0';
      histogram_bin_o         <= (others => '0');
      histogram_adc_o         <= (others => '0');
      histogram_ts_o          <= (others => '0');
      histogram_pileup_o      <= '0';
      histogram_ovfl_o        <= '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_ts_o          <= (others => '0');
      histogram_pileup_o      <= '0';
      histogram_ovfl_o        <= '0';

      ----------------------------------------------------------------------
      -- Calculate Thresholds and values for FIFO Delay
      ----------------------------------------------------------------------

      cts_trigger_delay_tmp  := cts_trigger_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_tmp + ts_window_offset_unsigned;
      else
        -- Offset is positive
        window_lower_thr       :=
          cts_trigger_delay_tmp - 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;

            -- Fill Histogram
            if (histogram_trig_filter = '1') then
              case histogram_ts_range is
```



```
                when  "000"  =>
                  histTStore        := deltaTStore( 8 downto 0);
                when  "001"  =>
                  histTStore        := deltaTStore( 9 downto 1);
                when  "010"  =>
                  histTStore        := deltaTStore(10 downto 2);
                when  "011"  =>
                  histTStore        := deltaTStore(11 downto 3);
                when  "100"  =>
                  histTStore        := deltaTStore(12 downto 4);
                when  "101" =>
                  histTStore        := deltaTStore(13 downto 5);
                when others =>
                  histTStore        := deltaTStore(12 downto 4);
              end case;

              if (histogram_limits = '1') then
                if (deltaTStore >= histogram_lower_limit and
                    deltaTStore <= histogram_upper_limit) then
                  histogram_fill_o     <= '1';
                  histogram_bin_o      <= CHANNEL_IN;
                  histogram_adc_o      <= ADC_DATA_IN;
                  histogram_ts_o       <= histTStore;
                  histogram_pileup_o   <= TIMESTAMP_STATUS_IN(S_PILEUP);
                  histogram_ovfl_o     <= TIMESTAMP_STATUS_IN(S_OVFL);
                end if;
              else
                histogram_fill_o       <= '1';
                histogram_bin_o        <= CHANNEL_IN;
                histogram_adc_o        <= ADC_DATA_IN;
                histogram_ts_o         <= histTStore;
                histogram_pileup_o     <= TIMESTAMP_STATUS_IN(S_PILEUP);
                histogram_ovfl_o       <= TIMESTAMP_STATUS_IN(S_OVFL);
              end if;
            end if;
          end if;

          if (out_of_window_error_ctr_clear = '1') then
            out_of_window_error_ctr          <= (others => '0');
          end if;
        end if;

        -- Fill Histogram
        if (histogram_trig_filter = '0') then
          histogram_fill_o                   <= '1';
          histogram_bin_o                    <= CHANNEL_IN;
          histogram_adc_o                    <= ADC_DATA_IN;
          histogram_ts_o                     <= (others => '0');
          histogram_pileup_o                 <= TIMESTAMP_STATUS_IN(S_PILEUP);
          histogram_ovfl_o                   <= TIMESTAMP_STATUS_IN(S_OVFL);
        end if;

      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_h_ctr      <= (others => '0');

        when "01"=>                   -- Start validation
          out_of_window_l_ctr      <= (others => '0');
          window_hit_ctr           <= (others => '0');
          out_of_window_h_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_h_ctr_r    <= out_of_window_h_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_h = '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_nr       <= (others => '0');
      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;
          data_rate_ctr_nr        <= data_rate_ctr_nr + 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;

timestamp_fpga_ff <= TIMESTAMP_FPGA_IN when rising_edge(CLK_IN);
timestamp_fpga_f  <= timestamp_fpga_ff  when rising_edge(CLK_IN);

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
    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');
      trigger_calibration         <= '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 + 280; --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;

      if (trigger_calibration = '1') then
        min_validation_time       :=
          min_validation_time + resize(trigger_calibration_delay, 20);
      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');
            trigger_calibration   <= TRIGGER_CALIBRATION_IN;
            STATE                 <= S_TRIGGER;
          else
            trigger_calibration   <= '0';
            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
            -- If Self-Trigger-Mode active set TS Ref to zero
            if (self_trigger_mode = '1') then
              timestamp_fpga      <= (others => '0');
            else
              timestamp_fpga      <=
                timestamp_fpga_f + fpga_timestamp_offset;
            end if;

            if (trigger_calibration = '1') then
              timestamp_fpga      <=
                timestamp_fpga_f + trigger_calibration_delay;
            end if;
            STATE                 <= S_WRITE_HEADER;
          end if;

        when S_WRITE_HEADER =>
          state_d                 <= "10";
          timestamp_ref           <= timestamp_fpga;
```

```
          t_data_o(11 downto 0)         <= timestamp_fpga;
          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);  -- convertion time in 4n steps
  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               <= "0001100100"; -- 100  = 400ns
      cts_trigger_delay             <= x"019";       -- 25   = 100ns
      readout_mode_r                <= "0000";
      readout_time_max              <= x"3e8";       -- 1000 = 10mus
      histogram_trig_filter         <= '1';
      fpga_timestamp_offset         <= (others => '0');
      out_of_window_error_ctr_clear <= '0';
      skip_wait_for_data            <= '0';
      nxyter_cv_time                <= x"190";       -- 400ns

      histogram_lower_limit         <= (others => '0');
      histogram_upper_limit         <= (others => '1');
      reset_hists                   <= '0';
      histogram_limits              <= '0';
      histogram_trig_filter         <= '0';
      histogram_ts_range            <= "100";
      trigger_calibration_delay     <= 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';
      reset_hists                      <= '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(15 downto 10)    <= (others => '0');
            slv_data_out_o(27 downto 16)    <=
              std_logic_vector(trigger_calibration_delay);
```

```
            slv_data_out_o(31 downto 28)    <= (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 x"0020" =>
            slv_data_out_o(13 downto 0)     <=
              std_logic_vector(histogram_lower_limit);
            slv_data_out_o(28 downto 15)    <=
              std_logic_vector(histogram_upper_limit);
            slv_data_out_o(29)              <= '0';
            slv_data_out_o(30)              <= histogram_limits;
            slv_data_out_o(31)              <= histogram_trig_filter;
            slv_ack_o                       <= '1';

          when x"0021" =>
            slv_data_out_o(2  downto 0)     <= histogram_ts_range;
            slv_data_out_o(31 downto 3)     <= (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));
            trigger_calibration_delay       <=
              unsigned(SLV_DATA_IN(27 downto 16));
            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 x"0020" =>
            histogram_lower_limit           <= SLV_DATA_IN(13 downto 0);
            histogram_upper_limit           <= SLV_DATA_IN(28 downto 15);
            histogram_limits                <= SLV_DATA_IN(30);
            histogram_trig_filter           <= SLV_DATA_IN(31);
            reset_hists                     <= '1';
            slv_ack_o                       <= '1';

          when x"0021" =>
            reset_hists                     <= '1';
            histogram_ts_range              <= SLV_DATA_IN(2 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
  ------------------------------------------------------------------------------

  pulse_to_level_RESET_HISTS: pulse_to_level
    generic map (
      NUM_CYCLES => 15
      )
    port map (
      CLK_IN    => CLK_IN,
      RESET_IN  => RESET_IN,
      PULSE_IN  => reset_hists,
      LEVEL_OUT => reset_hists_o
      );

  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_RESET_OUT    <= reset_hists_o;
  HISTOGRAM_FILL_OUT     <= histogram_fill_o;
  HISTOGRAM_BIN_OUT      <= histogram_bin_o;
  HISTOGRAM_ADC_OUT      <= histogram_adc_o;
  HISTOGRAM_TS_OUT       <= histogram_ts_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;
      PLL_RESET_OUT               : out   std_logic;
      TRIGGER_OUT                 : out   std_logic;
      I2C_SDA_INOUT               : inout std_logic;
      I2C_SCL_INOUT               : inout std_logic;
      I2C_SM_RESET_OUT            : inout 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_TIMESTAMP_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_ACK_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
  generic (
    DEBUG_ENABLE : boolean);
  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_CLK_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_CLK_OUT   : out std_logic;
    ADC0_LOCKED_OUT     : out std_logic;
    ADC1_LOCKED_OUT     : out std_logic;
    ADC0_SLOPPY_FRAME   : in  std_logic;
    ADC1_SLOPPY_FRAME   : in  std_logic;
    ADC0_ERROR_OUT      : out std_logic;
    ADC1_ERROR_OUT      : out std_logic;
    DEBUG_IN            : in  std_logic_vector(3 downto 0);
    DEBUG_OUT           : out std_logic_vector(15 downto 0)
    );
end component;

------------------------------------------------------------------------------
-- TRBNet Registers
------------------------------------------------------------------------------
```



```
component nx_register_setup
  port (
    CLK_IN               : in  std_logic;
    RESET_IN             : in  std_logic;
    I2C_ONLINE_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_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);
    NX_CLOCK_ON_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_status
  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;
    PLL_RESET_OUT          : out std_logic;
    I2C_SM_RESET_OUT       : inout std_logic;
    I2C_REG_RESET_OUT      : out std_logic;
    NX_ONLINE_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 fifo_data_stream_44to44_dc
  port (
    Data    : in  std_logic_vector(43 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(43 downto 0);
    Empty   : out std_logic;
    Full    : out std_logic
    );
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 nx_data_receiver
  generic (
    DEBUG_ENABLE : boolean
    );
  port (
    CLK_IN                : in  std_logic;
    RESET_IN              : in  std_logic;
    TRIGGER_IN            : in  std_logic;
    NX_ONLINE_IN          : in  std_logic;
    NX_CLOCK_ON_IN        : in  std_logic;
    NX_TIMESTAMP_CLK_IN   : in  std_logic;
    NX_TIMESTAMP_IN       : in  std_logic_vector (7 downto 0);
    NX_TIMESTAMP_RESET_OUT : out std_logic;
    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;
    DATA_OUT              : out std_logic_vector(43 downto 0);
    DATA_CLK_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;
    ADC_TR_ERROR_IN       : in  std_logic;
    DISABLE_ADC_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;
    DATA_IN              : in  std_logic_vector(43 downto 0);
    DATA_CLK_IN          : in  std_logic;
    DATA_OUT             : out std_logic_vector(43 downto 0);
    DATA_CLK_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;
    DATA_IN                 : in  std_logic_vector(43 downto 0);
    DATA_CLK_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_CLK_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;
    ADC_TR_ERROR_OUT        : out std_logic;
    DISABLE_ADC_IN          : in  std_logic;
    ERROR_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);
    VERSION_NUMBER : std_logic_vector(3 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_CALIBRATION_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_RESET_OUT    : out 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_TS_OUT       : out std_logic_vector(8 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);
    VERSION_NUMBER : std_logic_vector(3 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_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
    );
  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(31 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(31 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_TS_IN        : in  std_logic_vector(8 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 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 5
    );
  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;

component nx_fpga_timestamp
  port (
    CLK_IN                  : in  std_logic;
    RESET_IN                : in  std_logic;
    NX_MAIN_CLK_IN          : in  std_logic;
```

```
    TIMESTAMP_RESET_IN       : in  std_logic;
    TIMESTAMP_RESET_OUT      : out std_logic;
    TRIGGER_IN               : in  std_logic;
    TIMESTAMP_HOLD_OUT       : out unsigned(11 downto 0);
    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;
    TRIGGER_CALIBRATION_OUT  : out std_logic;
    FAST_CLEAR_OUT           : out std_logic;
    TRIGGER_BUSY_OUT         : out std_logic;
    NX_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;
    TRIGGER_BUSY_IN      : in  std_logic;
    EXTERNAL_TRIGGER_OUT : out std_logic;
    INTERNAL_TRIGGER_OUT : out std_logic;
    DATA_IN              : in  std_logic_vector(43 downto 0);
    DATA_CLK_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;

------------------------------------------------------------------------------
-- 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;
    PLL_RESET_OUT             : out 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          : inout 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_TIMESTAMP_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
  -- Data Format Version
  constant VERSION_NUMBER       : std_logic_vector(3 downto 0) := x"1";

------------------------------------------------------------------------------
-- 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 nx_timestamp_reset     : std_logic;
  signal nx_timestamp_reset_o   : std_logic;
  signal i2c_reg_reset_o        : std_logic;
  signal nxyter_online          : 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_clock_on        : std_logic;

  -- SPI Interface ADC
  signal spi_sdi                : std_logic;
  signal spi_sdo                : std_logic;

  -- Data Receiver
  signal data_recv              : std_logic_vector(43 downto 0);
  signal data_clk_recv          : std_logic;
  signal pll_sadc_clk_lock      : std_logic;
  signal disable_adc_receiver   : std_logic;

  -- Data Delay
  signal data_delayed           : std_logic_vector(43 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_clk               : std_logic;
  signal adc_tr_error           : 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_ts     : std_logic_vector(8 downto 0);
  signal trigger_validate_pileup : std_logic;
  signal trigger_validate_ovfl   : std_logic;
  signal reset_hists             : 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_calibration    : std_logic;
  signal trigger_busy           : std_logic;
  signal fast_clear             : std_logic;
  signal fee_trg_release_o      : std_logic;

  -- FPGA Timestamp
  signal timestamp_hold         : unsigned(11 downto 0);

  -- Trigger Generator
  signal internal_trigger       : std_logic;

  -- Error
  signal error_all              : std_logic_vector(7 downto 0);
  signal error_data_receiver    : std_logic;
  signal error_data_validate    : 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_data_validate;
  error_all(2)          <= error_event_buffer;
  error_all(3)          <= not nxyter_online;
  error_all(7 downto 4) <= (others => '0');

------------------------------------------------------------------------------
-- Port Maps
------------------------------------------------------------------------------

  THE_BUS_HANDLER: trb_net16_regio_bus_handler
    generic map(
      PORT_NUMBER         => NUM_PORTS,

      PORT_ADDRESSES      => (  0 => x"0100",    -- NX Status Handler
                                1 => x"0040",    -- I2C Master
                                2 => x"0500",    -- Data Receiver
                                3 => x"0080",    -- Event 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 Register Setup
                               10 => x"0800",    -- NX Histograms
                               11 => x"0020",    -- Debug Handler
                               12 => x"0000",    -- Data Delay
                                others => x"0000"
                                ),

      PORT_ADDR_MASK      => (  0 => 4,          -- NX Status Handler
                                1 => 1,          -- I2C master
                                2 => 5,          -- Data Receiver
                                3 => 3,          -- Event Buffer
                                4 => 0,          -- SPI Master
                                5 => 3,          -- Trigger Generator
                                6 => 5,          -- Data Validate
                                7 => 4,          -- Trigger Handler
                                8 => 6,          -- Trigger Validate
                                9 => 9,          -- NX Register Setup
                               10 => 11,         -- NX Histograms
                               11 => 0,          -- Debug Handler
                               12 => 3,          -- 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_status_1: nx_status
    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,
      PLL_RESET_OUT             => PLL_RESET_OUT,

      I2C_SM_RESET_OUT          => I2C_SM_RESET_OUT,
      I2C_REG_RESET_OUT         => i2c_reg_reset_o,
      NX_ONLINE_OUT             => nxyter_online,

      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_register_setup_1: nx_register_setup
    port map (
      CLK_IN               => CLK_IN,
      RESET_IN             => RESET_IN,
      I2C_ONLINE_IN        => nxyter_online,
      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_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,
      NX_CLOCK_ON_OUT      => nxyter_clock_on,
      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_ACK_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_RESET_IN      => nx_timestamp_reset,
      TIMESTAMP_RESET_OUT     => nx_timestamp_reset_o,
      TRIGGER_IN              => timestamp_trigger,
      TIMESTAMP_HOLD_OUT      => timestamp_hold,
      TIMESTAMP_TRIGGER_OUT   => NX_TIMESTAMP_TRIGGER_OUT,
      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         => not nxyter_online,

      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       => internal_trigger,

      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,
      TRIGGER_CALIBRATION_OUT   => trigger_calibration,
      FAST_CLEAR_OUT            => fast_clear,
      TRIGGER_BUSY_OUT          => trigger_busy,

      NX_TESTPULSE_OUT          => NX_TESTPULSE_OUT,

      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,

      TRIGGER_BUSY_IN     => trigger_busy,
      EXTERNAL_TRIGGER_OUT => TRIGGER_OUT,
      INTERNAL_TRIGGER_OUT => internal_trigger,

      DATA_IN             => data_recv,
      DATA_CLK_IN         => data_clk_recv,

      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
    generic map (
      DEBUG_ENABLE => true
      )
    port map (
      CLK_IN                => CLK_IN,
      RESET_IN              => RESET_IN,
      TRIGGER_IN            => trigger_timing,  -- for debugging only
      NX_ONLINE_IN          => nxyter_online,
      NX_CLOCK_ON_IN        => nxyter_clock_on,

      NX_TIMESTAMP_CLK_IN   => NX_TIMESTAMP_CLK_IN,
      NX_TIMESTAMP_IN       => NX_TIMESTAMP_IN,
      NX_TIMESTAMP_RESET_OUT => nx_timestamp_reset,

      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,

      DATA_OUT              => data_recv,
      DATA_CLK_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),
```

```
      ADC_TR_ERROR_IN        => adc_tr_error,
      DISABLE_ADC_OUT        => disable_adc_receiver,
      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,

      DATA_IN              => data_recv,
      DATA_CLK_IN          => data_clk_recv,

      DATA_OUT             => data_delayed,
      DATA_CLK_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,

      DATA_IN                => data_delayed,
      DATA_CLK_IN            => data_clk_delayed,

      TIMESTAMP_OUT          => timestamp,
      CHANNEL_OUT            => timestamp_channel_id,
      TIMESTAMP_STATUS_OUT   => timestamp_status,
      ADC_DATA_OUT           => adc_data,
      DATA_CLK_OUT           => data_clk,

      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),

      ADC_TR_ERROR_OUT       => adc_tr_error,
      DISABLE_ADC_IN         => disable_adc_receiver,
      ERROR_OUT              => error_data_validate,
      DEBUG_OUT              => debug_line(9)
      );

-------------------------------------------------------------------------------
-- NX Trigger Validate
-------------------------------------------------------------------------------

  nx_trigger_validate_1: nx_trigger_validate
    generic map (
      BOARD_ID       => BOARD_ID,
      VERSION_NUMBER => VERSION_NUMBER
      )
    port map (
      CLK_IN                   => CLK_IN,
      RESET_IN                 => RESET_IN,

      DATA_CLK_IN              => data_clk,
      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_CALIBRATION_IN   => trigger_calibration,
      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_RESET_OUT      => reset_hists,
      HISTOGRAM_FILL_OUT       => trigger_validate_fill,
      HISTOGRAM_BIN_OUT        => trigger_validate_bin,
      HISTOGRAM_ADC_OUT        => trigger_validate_adc,
      HISTOGRAM_TS_OUT         => trigger_validate_ts,
      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         => not nxyter_online,

      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,
      VERSION_NUMBER  => VERSION_NUMBER
      )
    port map (
      CLK_IN                  => CLK_IN,
      RESET_IN                => RESET_IN,
      NXYTER_OFFLINE_IN       => not nxyter_online,
      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            => reset_hists,
      CHANNEL_FILL_IN           => trigger_validate_fill,
      CHANNEL_ID_IN             => trigger_validate_bin,
      CHANNEL_ADC_IN            => trigger_validate_adc,
      CHANNEL_TS_IN             => trigger_validate_ts,
      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_RESET_OUT         <= not nx_timestamp_reset_o;

--------------------------------------------------------------------------------
-- I2C Signals
--------------------------------------------------------------------------------

  I2C_REG_RESET_OUT    <= not i2c_reg_reset_o;

--------------------------------------------------------------------------------
-- Others
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
-- 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)
      );
```

```
  --DEBUG_LINE_OUT <= (others => '0');

-------------------------------------------------------------------------------
-- 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;

  attribute syn_keep : boolean;
  attribute syn_keep of pulse_ff      : signal is true;

  attribute syn_preserve : boolean;
  attribute syn_preserve of pulse_ff  : signal is true;

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;

  attribute syn_keep : boolean;
  attribute syn_keep of signal_ff      : signal is true;

  attribute syn_preserve : boolean;
  attribute syn_preserve of signal_ff  : signal is true;

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;
```