VHDL for DataTypes_pkg.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.all;

package DataTypes_pkg is
   constant PN_NUMBITS_NB: integer := 12;
   constant PN_PRECISION_NB: integer := 4;
   constant PN_SIZE_LB: integer := 4;
   constant PN_SIZE_NB: integer := PN_NUMBITS_NB + PN_PRECISION_NB;

   constant BYTE_SIZE_LB: integer := 3;
   constant BYTE_SIZE_NB: integer := 8;

   constant WORD_SIZE_LB: integer := 4;
   constant WORD_SIZE_NB: integer := 16;

   constant PNL_BRAM_ADDR_SIZE_NB: integer := 13;
   constant PNL_BRAM_DBITS_WIDTH_LB: integer := PN_SIZE_LB;
   constant PNL_BRAM_DBITS_WIDTH_NB: integer := PN_SIZE_NB;
   constant PNL_BRAM_NUM_WORDS_NB: integer := 2**PNL_BRAM_ADDR_SIZE_NB;
   constant LARGEST_POS_VAL: integer := 16383;
   constant LARGEST_NEG_VAL: integer := -16383;
   constant PN_BRAM_BASE: integer := PNL_BRAM_NUM_WORDS_NB/2;
   constant PN_UPPER_LIMIT: integer := PNL_BRAM_NUM_WORDS_NB;
   constant DIST_BRAM_OFFSET: integer := (PNL_BRAM_NUM_WORDS_NB/2)/2;
end DataTypes_pkg;
```

VHDL for Top.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.all;

library work;
use work.DataTypes_pkg.all;

entity Top is
   port (
      Clk: in std_logic;
      PS_RESET_N: in std_logic;
      GPIO_Ins: in std_logic_vector(31 downto 0);
      GPIO_Outs: out std_logic_vector(31 downto 0);
      PNL_BRAM_addr: out std_logic_vector (PNL_BRAM_ADDR_SIZE_NB-1 downto 0);
      PNL_BRAM_din: out std_logic_vector (PNL_BRAM_DBITS_WIDTH_NB-1 downto 0);
      PNL_BRAM_dout: in std_logic_vector (PNL_BRAM_DBITS_WIDTH_NB-1 downto 0);
      PNL_BRAM_we: out std_logic_vector (0 to 0);
      DEBUG_IN: in std_logic;
      DEBUG_OUT: out std_logic
      );
end Top;
```

VHDL for Top.vhd

```
architecture beh of Top is

-- GPIO INPUT BIT ASSIGNMENTS
   constant IN_CP_RESET: integer := 31;
   constant IN_CP_START: integer := 30;
   constant IN_CP_LM_ULM_LOAD_UNLOAD: integer := 26;
   constant IN_CP_LM_ULM_DONE: integer := 25;
   constant IN_CP_HANDSHAKE: integer := 24;

-- GPIO OUTPUT BIT ASSIGNMENTS
   constant OUT_SM_READY: integer := 31;
   constant OUT_SM_HANDSHAKE: integer := 28;

-- Signal declarations
   signal RESET: std_logic;

   signal LM_ULM_start, LM_ULM_ready: std_logic;
   signal LM_ULM_stopped, LM_ULM_continue: std_logic;
   signal LM_ULM_done: std_logic;
   signal LM_ULM_base_address: std_logic_vector(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);
   signal LM_ULM_upper_limit: std_logic_vector(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);
   signal LM_ULM_load_unload: std_logic;

   signal DataIn: std_logic_vector(WORD_SIZE_NB-1 downto 0);
   signal DataOut: std_logic_vector(WORD_SIZE_NB-1 downto 0);
```

VHDL for Top.vhd

```
    begin
    DEBUG_OUT <= LM_ULM_ready;
    RESET <= GPIO_Ins(IN_CP_RESET) or not PS_RESET_N;

-- =====================
-- INPUT control and status signals
    LM_ULM_start <= GPIO_Ins(IN_CP_START);
    LM_ULM_load_unload <= GPIO_Ins(IN_CP_LM_ULM_LOAD_UNLOAD);
    LM_ULM_done <= GPIO_Ins(IN_CP_LM_ULM_DONE);
    LM_ULM_continue <= GPIO_Ins(IN_CP_HANDSHAKE);
    DataIn <= GPIO_Ins(WORD_SIZE_NB-1 downto 0);

-- =====================
-- OUTPUT control and status signals
    GPIO_Outs(OUT_SM_READY) <= LM_ULM_ready;
    GPIO_Outs(OUT_SM_HANDSHAKE) <= LM_ULM_stopped;
    GPIO_Outs(WORD_SIZE_NB-1 downto 0) <= DataOut;

-- =====================
-- Setup memory base and upper_limit
    LM_ULM_base_address <=
       std_logic_vector(to_unsigned(PN_BRAM_BASE, PNL_BRAM_ADDR_SIZE_NB));
    LM_ULM_upper_limit <=
       std_logic_vector(to_unsigned(PNL_BRAM_NUM_WORDS_NB -1,
          PNL_BRAM_ADDR_SIZE_NB));
```

VHDL for Top.vhd

```
-- Secure BRAM access control module
   LoadUnLoadMemMod: entity work.LoadUnLoadMem(beh)
      port map(Clk=>Clk, RESET=>RESET, start=>LM_ULM_start, ready=>LM_ULM_ready,
         load_unload=>LM_ULM_load_unload, stopped=>LM_ULM_stopped,
         continue=>LM_ULM_continue, done=>LM_ULM_done,
         base_address=>LM_ULM_base_address, upper_limit=>LM_ULM_upper_limit,
         CP_in_word=>DataIn, CP_out_word=>DataOut,
         PNL_BRAM_addr=>PNL_BRAM_addr, PNL_BRAM_din=>PNL_BRAM_din,
         PNL_BRAM_dout=>PNL_BRAM_dout, PNL_BRAM_we=>PNL_BRAM_we);
end beh;
```

VHDL for LoadUnloadMem.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.all;
library work;
use work.DataTypes_pkg.all;

entity LoadUnLoadMem is
   port(
      Clk: in std_logic;
      RESET: in std_logic;
      start: in std_logic;
      ready: out std_logic;
      load_unload: in std_logic;
      stopped: out std_logic;
      continue: in std_logic;
      done: in std_logic;
      base_address: in std_logic_vector(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);
      upper_limit: in std_logic_vector(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);
      CP_in_word: in std_logic_vector(WORD_SIZE_NB-1 downto 0);
      CP_out_word: out std_logic_vector(WORD_SIZE_NB-1 downto 0);
      PNL_BRAM_addr: out std_logic_vector(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);
      PNL_BRAM_din: out std_logic_vector(PNL_BRAM_DBITS_WIDTH_NB-1 downto 0);
      PNL_BRAM_dout: in std_logic_vector(PNL_BRAM_DBITS_WIDTH_NB-1 downto 0);
      PNL_BRAM_we: out std_logic_vector(0 to 0)
      );
end LoadUnLoadMem;
```

VHDL for LoadUnloadMem.vhd

```
architecture beh of LoadUnLoadMem is
   type state_type is (idle, load_mem, unload_mem, wait_load_unload, wait_done);
   signal state_reg, state_next: state_type;

   signal ready_reg, ready_next: std_logic;

   signal PNL_BRAM_addr_reg, PNL_BRAM_addr_next:
      unsigned(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);
   signal PNL_BRAM_upper_limit_reg, PNL_BRAM_upper_limit_next:
      unsigned(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);

   begin
   process(Clk, RESET)
      begin
      if ( RESET = '1' ) then
         state_reg <= idle;
         ready_reg <= '1';
         PNL_BRAM_addr_reg <= (others=>'0');
         PNL_BRAM_upper_limit_reg <= (others=>'0');
      elsif ( Clk'event and Clk = '1' ) then
         state_reg <= state_next;
         ready_reg <= ready_next;
         PNL_BRAM_addr_reg <= PNL_BRAM_addr_next;
         PNL_BRAM_upper_limit_reg <= PNL_BRAM_upper_limit_next;
      end if;
   end process;
```

VHDL for LoadUnloadMem.vhd

```
process (state_reg, start, ready_reg, load_unload, PNL_BRAM_addr_reg,
   PNL_BRAM_upper_limit_reg, PNL_BRAM_dout, CP_in_word, continue, base_address,
   upper_limit, done)
   begin
   state_next <= state_reg;
   ready_next <= ready_reg;

   PNL_BRAM_addr_next <= PNL_BRAM_addr_reg;
   PNL_BRAM_upper_limit_next <= PNL_BRAM_upper_limit_reg;

   PNL_BRAM_we <= "0";
   PNL_BRAM_din <= (others=>'0');
   CP_out_word <= (others=>'0');

   stopped <= '0';
```

VHDL for LoadUnloadMem.vhd

```
        case state_reg is

    -- =====================
            when idle =>
               ready_next <= '1';
               if ( start = '1' ) then
                  ready_next <= '0';
                  PNL_BRAM_addr_next <= unsigned(base_address);
                  PNL_BRAM_upper_limit_next <= unsigned(upper_limit);

                  if ( load_unload = '0' ) then
                     state_next <= load_mem;
                  else
                     state_next <= unload_mem;
                  end if;
               end if;
```

VHDL for LoadUnloadMem.vhd

```
-- =====================
          when load_mem =>
             stopped <= '1';
             if ( done = '0' ) then
                if ( continue = '1' ) then
                   PNL_BRAM_we <= "1";
                   PNL_BRAM_din <= (PNL_BRAM_DBITS_WIDTH_NB-1 downto WORD_SIZE_NB =>
                     '0') & CP_in_word;
                   state_next <= wait_load_unload;
                end if;
             else
                state_next <= wait_done;
             end if;

-- =====================
          when unload_mem =>
             CP_out_word <= PNL_BRAM_dout(WORD_SIZE_NB-1 downto 0);
             stopped <= '1';
             if ( continue = '1' ) then
                state_next <= wait_load_unload;
             end if;

             if ( done = '1' ) then
                state_next <= wait_done;
             end if;
```

VHDL for LoadUnloadMem.vhd

```
-- =====================
          when wait_load_unload =>
             if ( continue = '0' ) then
                if ( done = '1' ) then
                   state_next <= wait_done;
                elsif ( PNL_BRAM_addr_reg = PNL_BRAM_upper_limit_reg ) then
                   state_next <= idle;
                else
                   PNL_BRAM_addr_next <= PNL_BRAM_addr_reg + 1;
                   if ( load_unload = '0' ) then
                      state_next <= load_mem;
                   else
                      state_next <= unload_mem;
                   end if;
                end if;
             end if;

-- =====================
          when wait_done =>
             if ( done = '0' ) then
                state_next <= idle;
             end if;
       end case;
    end process;
    PNL_BRAM_addr <= std_logic_vector(PNL_BRAM_addr_next);
    ready <= ready_reg;
end beh;
```