LOADUNLOAD BRAM
VHD

```vhdl
----------------------------------------------------------------------------------
-- Company: University of New Mexico
-- Engineer: Professor Jim Plusquellic, Copyright Univ. of New Mexico
--
-- Create Date:
-- Design Name:
-- Module Name:    LoadUnLoadMem - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------

-- LoadUnLoadMem securely transfers data into or out of PNL_BRAM using GPIO
registers

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;

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

```

```
--
================================================================================
==========
-- State and register logic
```

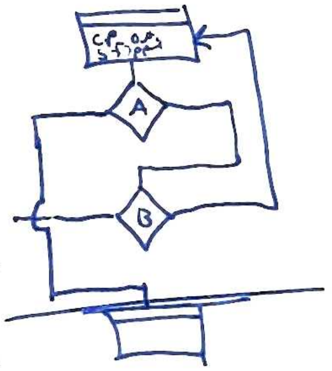
State 1

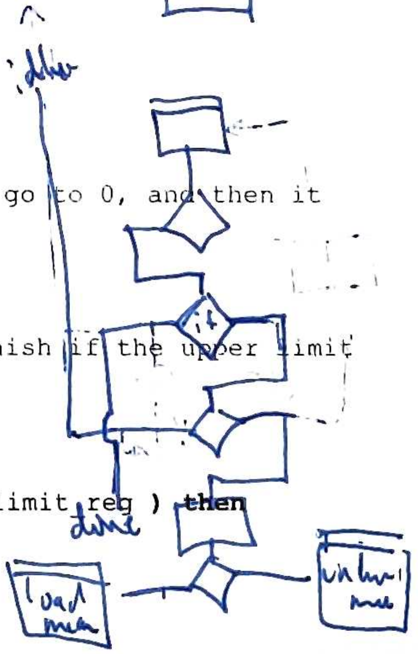
```
--
================================================================================
==========
   process(Clk, RESET)
      begin
      if ( RESET = '1' ) then
         state_reg <= idle;
         ready_reg <= '1';
         PNL_BRAM_addr_reg <= (others=>'0');
         PNL_BRAM_upper_limit_reg <= (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;

--
================================================================================
==========
-- Combo logic
```

State 2

```
--
================================================================================
==========
   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';

      case state_reg is

-- ====================
         when idle =>
            ready_next <= '1';

            if ( start = '1' ) then
               ready_next <= '0';

-- Latch the 'base_address' and 'upper_limit' at the instant 'start' is asserted.
NOTE: These signals MAY BE SET
-- BACK TO all 0's ~~after the 'start' signal is received.~~
         Asn   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;

-- =====================
-- Write value to memory location
         when load_mem =>

-- Signal C program that we are ready to receive a word. Once ready ('continue'
becomes '1'), transfer and complete handshake.
               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;

-- Wait handshake signals
                     state_next <= wait_load_unload;
                  end if;

-- Handle case where C program has nothing to store.
               else
                  state_next <= wait_done;
               end if;

-- =====================
-- Get value at memory location
         when unload_mem =>

-- Put the PNL BRAM word on CP_out_word. Do NOT do this by default for security
reasons.
               CP_out_word <= PNL_BRAM_dout(WORD_SIZE_NB-1 downto 0);

-- Signal C program that we are ready to deliver a word. Once it reads the word,
it sets 'continue' to '1'.
               stopped <= '1';
         A     if ( continue = '1' ) then

-- Wait handshake signals
                  state_next <= wait_load_unload;
               end if;

-- Handle case where C program does not want to read any data.
         B     if ( done = '1' ) then
                  state_next <= wait_done;
               end if;

-- =====================
-- Complete handshake and update addresses
         when wait_load_unload =>

-- C program holds 'continue' at 1 until it sees 'stopped' go to 0, and then it
writes a '0' to continue. It also writes
-- 'done' with a '1' when last transfer is made.
            A0 if ( continue = '0' ) then

-- Done collecting C program transmitted words. Force a finish if the upper limit
has been reached. This will protect the memory
-- from overruns (reading or writing).
               B0 if ( done = '1' ) then
                     state_next <= wait_done;
                  elsif ( PNL_BRAM_addr_reg = PNL_BRAM_upper_limit_reg ) then
                     state_next <= idle;
                  else
```

of the 32 bit GPIO register only store 15:0 as data





```
                  PNL_BRAM_addr_next <= PNL_BRAM_addr_reg + 1;
               C. if ( load_unload = '0' ) then
                     state_next <= load_mem;
                  else
                     state_next <= unload_mem;
               C  end if;
            B  end if;
          A end if;

-- =====================
-- Wait for 'done' to return to 0 before returning to idle, if it was set by the
C program to exit early.
         when wait_done =>
            if ( done = '0' ) then
               state_next <= idle;
            end if;

         end case;
      end process;

-- Use 'look-ahead' signal for BRAM address.
   PNL_BRAM_addr <= std_logic_vector(PNL_BRAM_addr_next);
   ready <= ready_reg;

end beh;
```