

```

--***** receiver.vhd *****
--* VHDL file for Xilinx XC4000E series FPGA on PMC daughter *
--* board of Pamette PCI Board. *
--* In order to receive data from DCM module, and put data *
--* into Pamette's DRAM.. 16X32 Dual Port FIFO w. common *
--* clock is implemented. *
--* Written by: Steve Xu Lin *
--* Date Created: 05/22/97 *
--* Version: 2.51 *
--***** Revision History *****
--* 06/30/97 by Steve Lin
--* Added logic to wirte 5 more words to FIFO after the HOLD is
--* asserted.
--* Added STROBE to wirte data into DRAM.
--*
--* 08/16/97 by Steve Lin
--* Changed all signals to be Std_Logic.
--*
--* 08/21/97 by Steve Lin
--* Hold will be asserted when fifo is 1/2 full, which allows
--* 8 words to be written into fifo.
--* After rd goes low, if there is a last word which did not written
--* into DRAM, but it's on the data bus, a strobe wil be provided.
--* Therefore, Pamette need to at least accept one word after it
--* removes rd.
--* 09/25/97 by Steve Lin
--* Add METAMOR PIN attributes, add signal AEN, add data_strobe
--* process in order to handle the AEN.
--* 10/10/97 by Steve Lin
--* Reduce the logic in process read to make the sythesis faster
--* from few hours to about 15 min with high effort
--* 12/04/97 by Steve Lin
--* Remove NPACKET, and add LAST to indicate the last word of a packet.

library IEEE;
use IEEE.std_logic_1164.all;
--library xblox;
--use xblox.all;

library METAMOR;
use METAMOR.attributes.all;
--***** Signal interface for receiver. last and EOP are signals
--* to indicate begin and end of packet.
--***** entity receiver is
entity receiver is
    port(
        data_in: in std_logic_vector(31 downto 0);
        data_out: out std_logic_vector(31 downto 0);
        empty, hold, EOP, nready, strobe: out std_logic;
        clock, rd, valid, last, reset, AEN: in std_logic);

-- assignment signals to pins

```

```

attribute pinnum of data_in : signal is "p10,p11,p12,p13,
p15,p16,p17,p18,p19,p20,p21,p22,p23,p24,
p27,p28,p29,p30,p31,p32,p33,p34,p35,p36, p38,p39,p40, p41, p42,p43,p44,p45";
attribute pinnum of data_out : signal is
"p150,p149,p148,p147,p146,p145,p144,p143,p141,p140,p139,p138,p137,p136,p135,p134
,p133,p132,p129,p128,p127,p126,p125,p124,p123,p122,p121,p120,p118,p117,p116,p115
";
attribute pinnum of empty      : signal is "P112";
attribute pinnum of hold       : signal is "P8";
attribute pinnum of EOP        : signal is "P113";
attribute pinnum of nready     : signal is "P9";
attribute pinnum of strobe     : signal is "p114";
attribute pinnum of clock      : signal is "p4";
attribute pinnum of rd         : signal is "p111";
attribute pinnum of valid      : signal is "p6";
attribute pinnum of last       : signal is "p7";
attribute pinnum of reset      : signal is "p46";
attribute pinnum of AEN        : signal is "p110";

attribute FAST: boolean;
attribute FAST of data_out: signal is true;

attribute Xilinx_BUFG: boolean;
attribute xilinx_bufg of strobe: signal is true;
attribute xilinx_bufg of AEN: signal is true;
attribute xilinx_bufg of clock: signal is true;
-- attribute xilinx_GSR:boolean;
-- attribute xilinx_GSR of reset: signal is true;
end receiver;

-----*****
architecture behave of receiver is
-----*****
-- component STARTUP
--   port (GSR: in std_logic);
-- end component;

constant height: integer := 15;
constant word_width: integer := 31;

type fifo_array is array(height downto 0) of std_logic_vector(word_width downto
0);
--/* Declare Internal Signals */
signal fifo: fifo_array;
signal direction: std_logic; --determine full or empty
--/* empty_flag is readback of empty signal */
-- signal empty_flag: std_logic;
signal lastwd: integer range 0 to height; --wrptr of the last word in a
pkt
signal last_flag: std_logic;
signal wrptr, rdptr: integer range 0 to height;
signal dmuxout: std_logic_vector(word_width downto 0);

begin
-- U1: STARTUP port map (gsr => reset);
-----*****
--* Initialize FIFO

```

```

--*****  

initial0: process(reset,clock,valid,wrptr,data_in)
begin
    if (reset='1') then
        fifo(0) <=(others =>'0') ;
    elsif ((clock'event)and(clock='1')) then
        if (valid ='1')and (wrptr=0) then
            fifo(0) <= data_in;
        end if;
    end if;
end process;  

  

initial1: process(reset,clock,valid,wrptr,data_in)
begin
    if (reset='1') then
        fifo(1)  <= (others =>'0');
    elsif ((clock'event)and(clock='1')) then
        if (valid ='1')and (wrptr=1) then
            fifo(1) <= data_in;
        end if;
    end if;
end process;  

initial2: process(reset,clock,wrptr, data_in, valid)
begin
    if reset='1' then
        fifo(2)  <= (others =>'0');
    elsif ((clock'event)and(clock='1')) then
        if (valid ='1')and (wrptr=2) then
            fifo(2) <= data_in;
        end if;
    end if;
end process;  

initial3: process(reset,clock,wrptr,valid,data_in)
begin
    if reset='1' then
        fifo(3)  <= (others =>'0');
    elsif ((clock'event)and(clock='1')) then
        if (valid ='1')and (wrptr=3) then
            fifo(3) <= data_in;
        end if;
    end if;
end process;  

initial4: process(reset,clock,wrptr,valid,data_in)
begin
    if reset='1' then
        fifo(4)  <= (others =>'0');
    elsif ((clock'event)and(clock='1')) then
        if (valid ='1')and (wrptr=4) then
            fifo(4) <= data_in;
        end if;
    end if;
end process;  

initial5: process(reset,clock)
begin
    if reset='1' then
        fifo(5)  <= (others =>'0');

```

```

        elsif ((clock'event)and(clock='1')) then
            if (valid ='1')and (wrptr=5) then
                fifo(5) <= data_in;
            end if;
        end if;
    end process;
initial6: process(reset,clock)
begin
    if reset='1' then
        fifo(6)  <= (others =>'0');
    elsif ((clock'event)and(clock='1')) then
        if (valid ='1')and (wrptr=6) then
            fifo(6) <= data_in;
        end if;
    end if;
end process;
initial7: process(reset,clock)
begin
    if reset='1' then
        fifo(7)  <= (others =>'0');
    elsif ((clock'event)and(clock='1')) then
        if (valid ='1')and (wrptr=7) then
            fifo(7) <= data_in;
        end if;
    end if;
end process;
initial8: process(reset,clock)
begin
    if reset='1' then
        fifo(8)  <= (others =>'0');
    elsif ((clock'event)and(clock='1')) then
        if (valid ='1')and (wrptr=8) then
            fifo(8) <= data_in;
        end if;
    end if;
end process;
initial9: process(reset,clock)
begin
    if reset='1' then
        fifo(9)  <= (others =>'0');
    elsif ((clock'event)and(clock='1')) then
        if (valid ='1')and (wrptr=9) then
            fifo(9) <= data_in;
        end if;
    end if;
end process;
initial10: process(reset,clock)
begin
    if reset='1' then
        fifo(10)  <= (others =>'0');
    elsif ((clock'event)and(clock='1')) then
        if (valid ='1')and (wrptr=10) then
            fifo(10) <= data_in;
        end if;
    end if;
end process;
initial11: process(reset,clock)

```

```

begin
    if reset='1' then
        fifo(11) <= (others =>'0');
    elsif ((clock'event)and(clock='1')) then
        if (valid ='1')and (wrptr=11) then
            fifo(11) <= data_in;
        end if;
    end if;
end process;
initial12: process(reset,clock)
begin
    if reset='1' then
        fifo(12) <= (others =>'0');
    elsif ((clock'event)and(clock='1')) then
        if (valid ='1')and (wrptr=12) then
            fifo(12) <= data_in;
        end if;
    end if;
end process;
initial13: process(reset,clock)
begin
    if reset='1' then
        fifo(13) <= (others =>'0');
    elsif ((clock'event)and(clock='1')) then
        if (valid ='1')and (wrptr=13) then
            fifo(13) <= data_in;
        end if;
    end if;
end process;
initial14: process(reset,clock)
begin
    if reset='1' then
        fifo(14) <= (others =>'0');
    elsif ((clock'event)and(clock='1')) then
        if (valid ='1')and (wrptr=14) then
            fifo(14) <= data_in;
        end if;
    end if;
end process;
initial15: process(reset,clock)
begin
    if reset='1' then
        fifo(15) <= (others =>'0');
    elsif ((clock'event)and(clock='1')) then
        if (valid ='1')and (wrptr=15) then
            fifo(15) <= data_in;
        end if;
    end if;
end process;

--***** Check if the end of packet is encountered, and tracking the last word
--* within the packet
--***** Track_lastwd: process(clock, reset, rdptr, last)

```

```

begin
    if reset='1' then
        last_flag <= '0';
    elsif (((clock'event) and (clock='1')) and(last ='1')) then
        last_flag <= '1';
    end if;
    if (rdptr = lastwd+1)and (last='0') then
        last_flag<='0';
    end if;
end process;

--*****
-- Tracking last word
--*****
Last_word: process(clock, reset, last)
begin
    if reset = '1' then
        lastwd <= 15;
    elsif (((clock'event) and (clock='1')) and (last = '1')) then
        lastwd <= wrptr;
    end if;
end process;

--*****
--* Control Logic to determine Hold, Empty, etc.
--*****
control: process(clock, reset, rdptr, wrptr)
begin
    if reset='1' then
        direction<='1';
    elsif ((clock'event)and(clock='1')) then
        if ((rdptr>wrptr)and((rdptr-wrptr)<10)) then
            direction <='1';
        elsif ((wrptr>rdptr)and((wrptr-rdptr)<2)) then
            direction <='0';
        end if;
    end if;
end process;

--*****
--* Increment Read Pointer and generate empty signal
--*****
read_ptr: process(fifo, reset, rd, rdptr, AEN)
begin
    if reset = '1' then
        rdptr <= 0;
    elsif ((AEN'event)and(AEN='0')) then
        if ((rd='1')and(rdptr=height)) then
            rdptr <= 0;
        elsif ((rd ='1')and(rdptr<height)) then
            rdptr <= rdptr+1;
        end if;
    end if;
end process;
--*****
--* Increment the write pointer and generate HOLD logic

```

```

--*****
-- Hold_DCM: process(clock, reset, rdptr, direction, wrptr)
begin
    if reset = '1' then
        hold <= '0';
    elsif ((clock'event)and(clock='0')) then
        if (((rdptr-wrptr)=9)and(direction='1')) or
            (((wrptr-rdptr)=7)and(direction='0')) then
            -- (((wrptr-rdptr)=22) and (direction='0')) then **32 deep
                hold<='1';
        elsif (((wrptr-rdptr)<7)and(wrptr>rdptr)) or
            (((rdptr-wrptr)>9)and(rdptr>wrptr)) then
                hold <='0';
        end if;
    end if;
end process;

--*****
-- Generate HOLD to control data flow
--*****
Write_ptr: process(clock, wrptr, reset)
begin
    if reset = '1' then
        wrptr <= 15;
    elsif ((clock'event) and (clock='0')) then
        if ((valid ='1')and(wrptr=height)) then
            wrptr<=0;
        elsif ((valid ='1')and(wrptr<height)) then
            wrptr <= wrptr+1;
        end if;
    end if;
end process;

--*****
--* output of FIFO
--*****
tri_state: process(rd, dmuxout)
begin
    if (rd='1') then
        data_out <= dmuxout;
    else
        data_out  <= (others => 'X') ;
    end if;
end process;

--*****
--* Error checking
--* if there is no clock or reset=1, nready = 1 (false)
--*****
error_check: process(clock, reset)
begin
    if reset ='1' then
        nready <= '1';
    elsif ((clock'event) and (clock='1')) then
        nready <= '0';
    end if;
end process;

```

```

--***** Generate STROBE signal to write data into PAM DRAM upon receiving
--** the AEN from PAM
--*****
data_strobe: process(AEN, reset)
begin
    if (reset='1') then
        strobe <= '0';
    elsif (AEN='1') then
        strobe <= '1' ; -- after 3 NS;
    else
        strobe <= '0';
    end if;
end process;
--***** Data MUX or tri-state buffer to isolate data from data_out bus
--*****
data_mux: process(rdptra, fifo, reset)
begin
    if reset='1' then
        dmuxout<=fifo(0);
    else
        dmuxout<=fifo(rdptra);
    end if;
end process;
--***** Generate End of Packet logic to inform PAMETTE that the word is last word
--** within the last packet
--*****
END_PKT: process(reset, lastwd, last_flag, rdptra, rd)
begin
    if (reset = '1') then
        EOP <= '0';
    elsif ((rd='1')and(rdptra=lastwd)and(last_flag='1')) then
        EOP <= '1';
    else
        EOP <= '0';
    end if;
end process;
--
--***** Generate EMPTY signa to tell PAM to stop reading
--*
--*****
empty_fifo: process(reset, wrptr, rdptra, AEN, clock)
begin
    if reset = '1' then
        empty <= '1';
    elsif (((clock'event) and(clock='0')) or((AEN'event) and(AEN='0'))) then
        if ((rdptr=wrptr-1) or (rdptr=15+wrptr)) then
            empty<='1';
        elsif ((rdptr=wrptr-2) or (rdptr=14+wrptr)) then

```

```
        empty<= '0';
    end if;
end if;
end process;

end behave;
```