Interfacing IBM's PowerPC 403GC to PCI using V360EPC from V3 Semiconductor

1. Objective

This application note describes the interface between PPC403GC processors from IBM and V360EPC Enhanced PCI Controller (EPC) from V3 Semiconductor. The V360EPC family of devices are pin to pin and software compatible with previous V292PBC RevB2, and provide superior functionality like DMA chaining, demand mode DMA, 3.3V support up to 33MHz (50MHz at 5V), hot swap friendly features and availability in industrial (-40 to 85'C) grades. V292PBC RevB2 PCI Bridge Controller (PBC) may be alternatively used - up to 40MHz - for further cost reduction in applications where the enhanced features of EPC are not required.

2. Overview

Here, we focus around popular DRAM-based PPC403GC embedded applications. Both PCI and local bus "master" and "target" functionalities are supported via V360EPC where the internal DRAM controller of PPC403GC is utilized for DRAM access. To support external local masters - e.g. V360EPC - for DRAM access, extra address multiplexing logic for row/ column address decoding is also required (refer to chapter 3 of PPC403GC embedded controller users manual). The address multiplexing logic will not be a requirement for SRAM-based applications. In the following sections, system interconnect, finite state machine, glue logic and VHDL codes for interfacing V360EPC to PPC403GC are described.

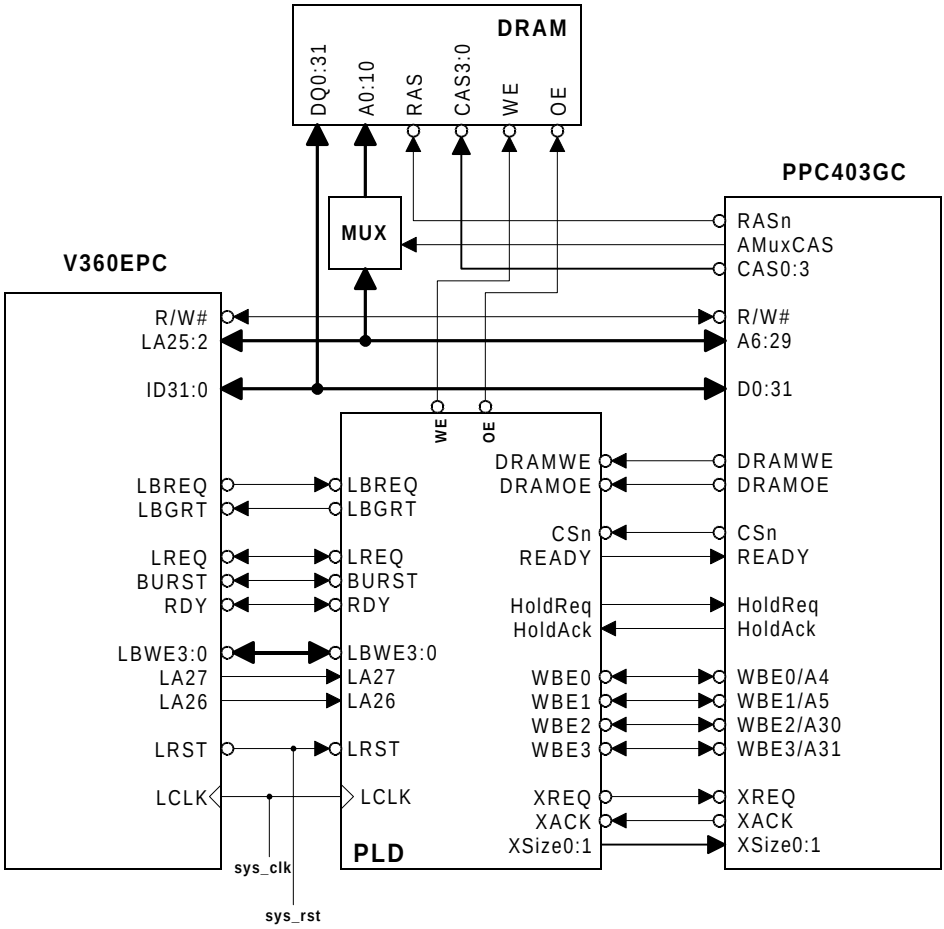
3. System Interconnect

Interconnection of the PPC403GC to the V360EPC (or V292PBC) requires minimal glue logic as indicated in Figure 1. The PLD and MUX blocks contain the necessary glue logic to interface PPC403GC to PCI bus via V360EPC. Notice that the bus ordering convention is reverse on PPC403GC - e.g. the local data bit 0 (ID0) on the V360EPC is equivalent to D31 on the PPC403GC. Furthermore, due to the fact that PPC403GC does not provide the upper 6 address bits A[0:5], the maximum addressing range for each of the eight external memory banks (SRAM or DRAM) is 64MB.

The higher order address bits of V360EPC can be connected to GND via pull-down resistors. This provides 64MB window which can be divided for Aperture_0, Aperture_1 and EPC's internal registers if only one of PPC403GC's bank registers (in SRAM configuration) is used to communicate to PCI.

LREQ#, BURST#, RDY# and XACK# need pull-up resistors for proper operation.

Figure 1- The interface between V360EPC and PPC403GC



4. Functional Description

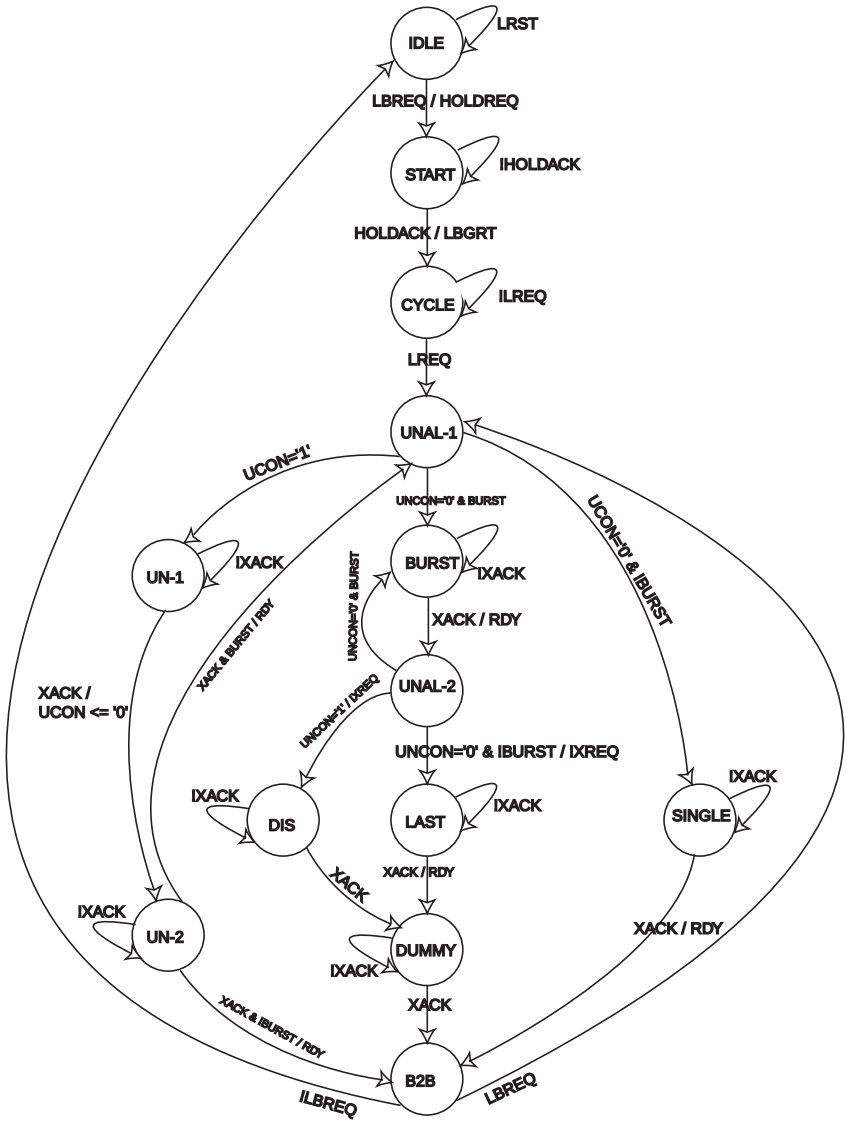
The interface provides the following two main functionalities:

- PCI agents accessing local DRAM controlled by PPC403GC via V360EPC (*master*),
- PPC403GC accessing devices on PCI bus via V360EPC (*target*).

4.1 MASTER INTERFACE

Top level functionality and state machine design of the master interface - V360EPC acting as local bus master - is described in this section. Figure 2 demonstrates the state machine diagram for V360EPC accessing the DRAM via PPC403GC's internal DRAM controller.

Figure 2- Master state machine



Single accesses, burst and unaligned transfers from PCI are all supported. Notice that PPC403GC detects the end of an external burst cycle in a quite different fashion compared to V360EPC. V360EPC relies on BURST# signal being deasserted, which acts similar to FRAME# in PCI. In other words, deassertion of BURST# indicates that the current data being sampled is the last data. On the other hand, PPC403GC protocol expects the last data to arrive one cycle after XREQ# is deasserted - or XSIZE0:1 does not have a "11" value. Since XREQ# is generated from BURST#, an unwanted data will always follow the last data coming from PCI. That's why the master state machine disables DRAM modules to mask off this dummy data at the end of the burst cycles (state DUMMY in Figure 2).

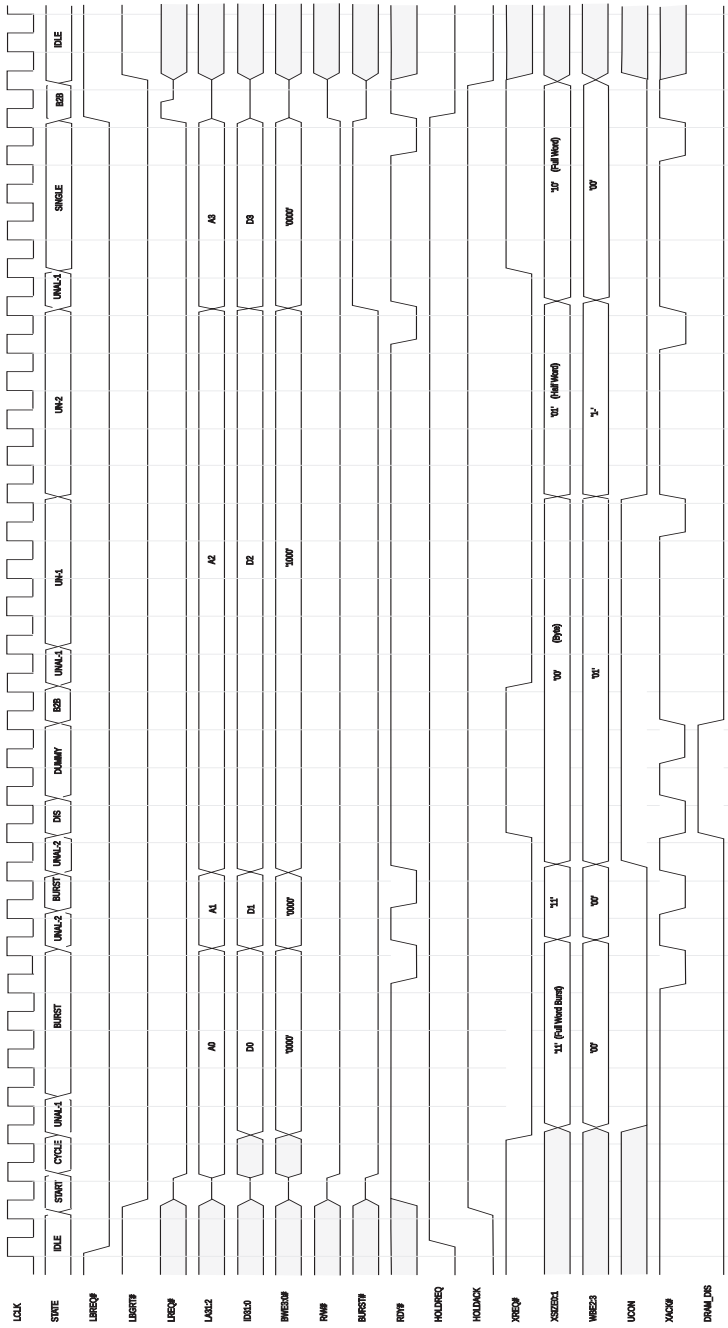
Furthermore, unaligned burst transfers to local memory, even in the middle of a burst, are fully supported by translating the local byte enables (LBWE3:0#) into PPC403GC 8-bit and/ or 16-bit "byte" and "half word" accesses. This is illustrated in Table 1 below:

Table 1- Unaligned transfers broken to two cycles on PPC403GC's local bus (Big Endian)

				First Cycle		**Second Cycle**		
LBWE(3)	LBWE(2)	LBWE(1)	LBWE(0)	Xsize(0:1)	A(30:31)	Xsize(0:1)	A(30:31)	UCON
0	0	0	0	x x	x x	x x	x x	x
0	0	0	1	0 0	1 1	x x	x x	0
0	0	1	0	0 0	1 0	x x	x x	0
0	0	1	1	0 1	1 x	x x	x x	0
0	1	0	0	0 0	0 1	x x	x x	0
0	1	0	1	0 0	0 1	0 0	1 1	1
0	1	1	0	0 0	0 1	0 0	1 0	1
0	1	1	1	0 0	0 1	0 1	1 x	1
1	0	0	0	0 0	0 0	x x	x x	0
1	0	0	1	0 0	0 0	0 0	1 1	1
1	0	1	0	0 0	0 0	0 0	1 0	1
1	0	1	1	0 0	0 0	0 1	1 x	1
1	1	0	0	0 1	0 x	x x	x x	0
1	1	0	1	0 1	0 x	0 0	1 1	1
1	1	1	0	0 1	0 x	0 0	1 0	1
1	1	1	1	1 0	0 0	x x	x x	0

Notice that if an unaligned case is detected in the middle of a burst transfer, then the state machine of Figure 2 will disable the DRAM module for the current data (state DIS) and terminates the burst (state DUMMY) before entering the unaligned transfer states (UN_1 and UN-2). The internal 1-bit counter UCON will be set to 1, as shown in Table 1 above, whenever an unaligned case is detected. Figure 3 illustrates an unaligned PCI cycle in the middle of a burst write to the local memory. Here, when byte enables change during a burst access, the PLD will disable the local DRAM memory to avoid unwanted data corruption (states DIS and Dummy).

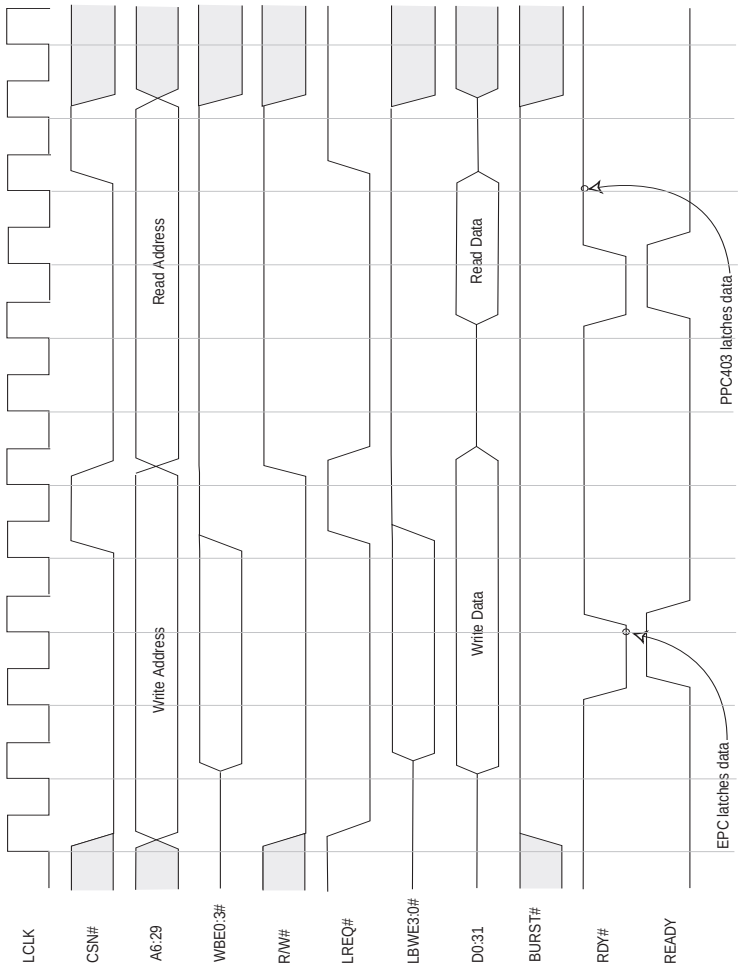
Figure 3 Unaligned burst transfers performed by V360EPC



4.2 TARGET INTERFACE

The PPC403GC can initiate PCI IO, memory and configuration cycles (Type-0 and Type-1) via V360EPC. The internal configuration registers of V360EPC are accessible by both PPC403GC (via LB_IO_BASE register) and host (via PCI_IO_BASE register). There are 8 bank registers in PPC403GC where one or more of them can be programmed (in SRAM mode) for communicating to PCI. For instance, bank register 0, providing 64MB window, can be used to access PCI via Aperture_0, Aperture_1 and also EPC's internal registers. Notice that both PCI memory and IO cycles can be initiated via either of the apertures. Since PPC403GC does not indicate the end of a burst cycle, only "single" cycles can be supported in target mode as demonstrated in Figure 4.

Figure 4- Single Write followed by a Read cycle



The target interface is simple and fully combinational. There is a dedicated internal bit in both V360EPC and V292PBC rev B2 for compatibility in the slave mode, where PPC403GC is performing a read from PCI. Setting bit-11 (PPC_RDY) of the LB_CFG register to '1', results in extending the availability of data for an extra cycle which is compatible with PPC403GC protocol where data is latched one clock cycle following the activation of READY - as illustrated in Figure 4. This eliminates any requirement for latching buffers for the data bus.

4.3 ADDRESS MULTIPLEXER

An external multiplexer is needed to generate row and column addresses for the DRAM modules - refer to block MUX in Figure 1. This is a requirement by PPC403GC's internal DRAM controller to support memory access for other local bus masters. For example, a PPC403GC-based embedded system with 4MB memory, comprised of two 1Mx16 bit DRAM modules, can be addressed as illustrated in Table 2.

Table 2- External address multiplexer for 1Mx16 DRAM modules

DRAM Address (A9-A0)	Row	Column
A9	LA21	LA11
A8	LA20	LA10
A7	LA19	LA9
A6	LA18	LA8
A5	LA17	LA7
A4	LA16	LA6
A3	LA15	LA5
A2	LA14	LA4
A1	LA13	LA3
A0	LA12	LA2

Notice that the internal/external multiplexer bit of the corresponding PPC403GC bank register must also be set to enable DRAM access via the external multiplexer.

5. PLD Equations

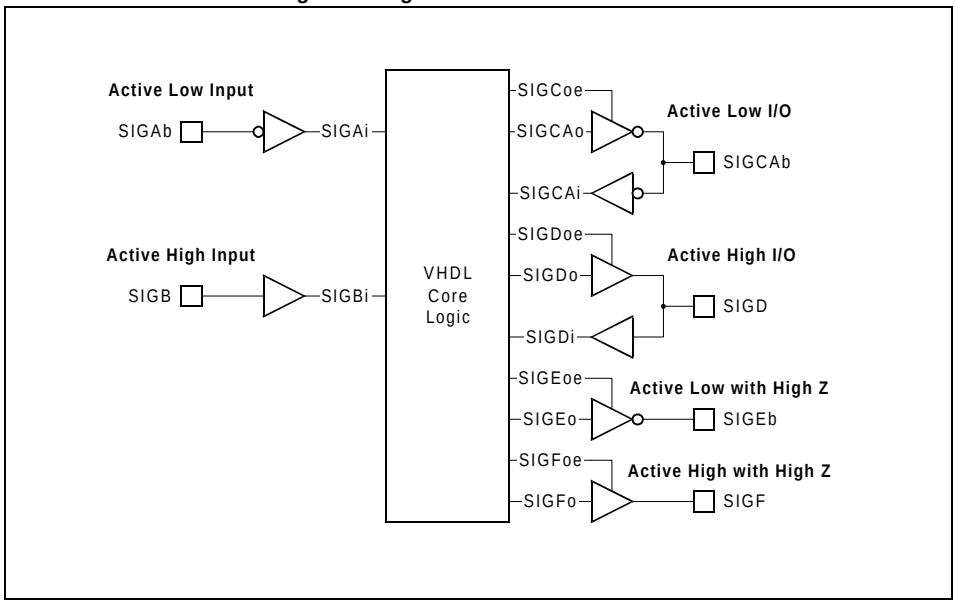
The glue logic source code is written in VHDL and can be targeted at a number of small, low cost PLD devices such as CY7C371i from Cypress Semiconductor. There are a few conventions adopted in writing the code. In the following sections, first the coding convention, then the protocol conversion (both master and target cases) along with the necessary address multiplexing codes are described in details.

5.1 VHDL CONVENTIONS

There are some conventions that have been used in writing the VHDL equations that help to make them more readable:

- Signal names are all in upper case except as outlined below
- Signals that are active low (XACKb, BWEb, BURSTb) are indicated by a lower case 'b' at the end of the signal name. All active low signals are converted to active high when used in the logic generation portion of the VHDL code.
- Top level signals pass through an I/O buffer stage as shown in Figure 5. This converts everything to positive true logic and also converts weak input highs/lows 'H'/'L' to '1'/'0'.
- All VHDL core logic is written in positive true logic. That is, an active low signal (such as $\overline{RDY}$ called RDYb in the VHDL) will be converted to an active high equivalent (as in Figure 5) and then used by the core logic in the active high sense (RDYi = '1' means $\overline{RDY}$ is "active).

Figure 5- Signal Names in Relation to I/O Buffers



5.2 PROTOCOL CONVERSIONS

The source code for conversion of PPC403GC protocol to V360EPC and vice versa is presented below. A hierarchical style is adopted to isolate core from top level logic.


```
--=============================================================================--
-- MODULE << pld >>
-- Description : Top level PLD interfacing PPC403GC to V360EPC
-- By: F. Pourbigharaz
-- (c) Copy Right V3 Semiconductor Corp. June 1998.
--=============================================================================--

library IEEE;
use IEEE.std_logic_1164.all;
entity pld is
    port(
-- system ports
        LRSTb   : in        STD_ULOGIC;
        LCLK    : in        STD_ULOGIC;
-- Interface to V360EPC
        LA27    : in        STD_ULOGIC;
        LA26    : in        STD_ULOGIC;
        LBWEb   : inout     STD_LOGIC_VECTOR(3 downto 0);
        RDYb    : inout     STD_LOGIC;
        BURSTb  : inout     STD_LOGIC;
        LREQb   : inout     STD_LOGIC;
        LBGRTb  : out       STD_ULOGIC;
        LBREQb  : in        STD_ULOGIC;

-- Interface to PPC403GC
        DRAMWEb : in        STD_ULOGIC;
        DRAMOEb : in        STD_ULOGIC;
        CSNb    : in        STD_ULOGIC;
        READY   : out       STD_ULOGIC;
        HOLDREQ : out       STD_ULOGIC;
        HOLDACK : in        STD_ULOGIC;
        WBEb    : inout     STD_LOGIC_VECTOR(0 to 3);
        XREQb   : out       STD_LOGIC;
        XACKb   : in        STD_ULOGIC;
        XSIZE   : out       STD_LOGIC_VECTOR(0 to 1);

-- Output and write enables for DRAM
        OEb: outSTD_ULOGIC;
        WEb: outSTD_ULOGIC
    );

end pld;

architecture BEHAVIOUR of pld is
-------------------------------------------------------------------------------
-- lower level component declaration
-------------------------------------------------------------------------------
component pld_core
    port(
        LRSTi   : in        STD_ULOGIC;
        LCLKi   : in        STD_ULOGIC;
        LA27i   : in        STD_ULOGIC;
        LA26i   : in        STD_ULOGIC;
        LBWEi   : in        STD_ULOGIC_VECTOR(3 downto 0);
        LBWEo   : out       STD_ULOGIC_VECTOR(3 downto 0);
        LBWEoe  : out       STD_ULOGIC;
        RDYi    : in        STD_ULOGIC;
        RDYo    : out       STD_ULOGIC;
        RDYoe   : out       STD_ULOGIC;
        BURSTi  : in        STD_ULOGIC;
        BURSTo  : out       STD_ULOGIC;
        BURSToe : out       STD_ULOGIC;
        LREQi   : in        STD_ULOGIC;
        LREQo   : out       STD_ULOGIC;
        LREQoe  : out       STD_ULOGIC;
```

```
        LBGRTo  : out       STD_ULOGIC;
        LBREQi  : in        STD_ULOGIC;
        DRAMWEi : in        STD_ULOGIC;
        DRAMOEi : in        STD_ULOGIC;
        CSNi    : in        STD_ULOGIC;
        READYo  : out       STD_ULOGIC;
        HOLDREQo: out       STD_ULOGIC;
        HOLDACKi: in        STD_ULOGIC;
        WBEi    : in        STD_ULOGIC_VECTOR(0 to 3);
        WBEo    : out       STD_ULOGIC_VECTOR(0 to 3);
        WBEoe   : out       STD_ULOGIC;
        XREQo   : out       STD_ULOGIC;
        XREQoe  : out       STD_ULOGIC;
        XACKi   : in        STD_ULOGIC;
        XSIZEo   : out      STD_ULOGIC_VECTOR(0 to 1);
        XSIZEoe  : out      STD_ULOGIC;
        OEo      : out      STD_ULOGIC;
        WEo      : out      STD_ULOGIC
    );

end component;
-------------------------------------------------------------------------------
-- Internal signal declarations
-------------------------------------------------------------------------------
signal LRSTi, LCLKi           : STD_ULOGIC;
signal LA27i, LA26i           : STD_ULOGIC;
signal LBWEi, LBWEo           : STD_ULOGIC_VECTOR(3 downto 0);
signal LBWEoe                 : STD_ULOGIC;
signal RDYi, RDYo, RDYoe      : STD_ULOGIC;
signal BURSTi, BURSTo, BURSToe: STD_ULOGIC;
signal LREQi, LREQo, LREQoe   : STD_ULOGIC;
signal LBGRTo, LBREQi         : STD_ULOGIC;
signal DRAMWEi, DRAMOEi, CSNi : STD_ULOGIC;
signal READYo                 : STD_ULOGIC;
signal HOLDREQo, HOLDACKi     : STD_ULOGIC;
signal WBEi, WBEo             : STD_ULOGIC_VECTOR(0 to 3);
signal WBEoe                  : STD_ULOGIC;
signal XREQo, XREQoe, XACKi   : STD_ULOGIC;
signal XSIZEo                 : STD_ULOGIC_VECTOR(0 to 1);
signal XSIZEoe                : STD_ULOGIC;
signal OEo, WEo               : STD_ULOGIC;

begin
-------------------------------------------------------------------------------
-- pld core instantiation
-------------------------------------------------------------------------------
U1 : pld_core
port map ( LRSTi,LCLK,LA27i,LA26i,LBWEi,LBWEo,LBWEoe,RDYi,
           RDYo,RDYoe,BURSTi,BURSTo,BURSToe,LREQi,LREQo,
           LREQoe,LBGRTo,LBREQi,DRAMWEi,DRAMOEi,CSNi,READYo,
           HOLDREQo,HOLDACKi,WBEi,WBEo,WBEoe,XREQo,XREQoe,
           XACKi,XSIZEo,XSIZEoe,OEo,WEo
         );

-------------------------------------------------------------------------------
-- I/O Buffers
-------------------------------------------------------------------------------
-- All signals go to the pld_core level as active high

LCLKi    <= LCLK;
LRSTi    <= not LRSTb;
LA27i    <= To_X01(STD_ULOGIC(LA27));
LA26i    <= To_X01(STD_ULOGIC(LA26));
LBWEb    <= STD_LOGIC_VECTOR(not LBWEo) when (LBWEoe = '1')
            else (others => 'Z');
```



```
LBWEi    <= To_X01(STD_ULOGIC_VECTOR(not LBWEb));
RDYb     <= STD_LOGIC(not RDYo) when (RDYoe='1') else 'Z';
RDYi     <= To_X01(STD_ULOGIC(not RDYb));
BURSTb   <= STD_LOGIC(not BURSTo) when (BURSToe='1') else 'Z';
BURSTi   <= To_X01(STD_ULOGIC(not BURSTb));
LREQb    <= STD_LOGIC(not LREQo) when (LREQoe='1') else 'Z';
LREQi    <= To_X01(STD_ULOGIC(not LREQb));
LBREQi   <= To_X01(not LBREQb);
LBGRTb   <= not LBGRTo;
DRAMWEi  <= To_X01(not DRAMWEb);
DRAMOEi  <= To_X01(not DRAMOEb);
CSNi     <= To_X01(not CSNb);
READY    <= READYo;
HOLDREQ  <= HOLDREQo;
HOLDACKi <= To_X01(HOLDACK);
-- WBE is active high when EPC is the bus master
WBEb     <= STD_LOGIC_VECTOR(WBEo) when (WBEoe='1')
            else (others => 'Z');
-- WBE is active low when ppc403 is the bus master
WBEi     <= To_X01(STD_ULOGIC_VECTOR(not WBEb));
XREQb    <= STD_LOGIC (not XREQo) when (XREQoe = '1') else 'Z';
XACKi    <= To_X01(not XACKb);
XSIZE    <= STD_LOGIC_VECTOR(XSIZEo) when (XSIZEoe = '1')
            else (others => 'Z');
OEb      <= not OEo;
WEb      <= not WEo;


end BEHAVIOUR;
--=============================================================================--
-- END of MODULE << pld >>
--=============================================================================--

--=============================================================================--
-- MODULE << pld_core >>
-- Description : Core level PLD interfacing PPC403GC to V360EPC
-- By: F. Pourbigharaz
-- (c) Copy Right V3 Semiconductor Corp. June 1998.
--=============================================================================--
library IEEE;
use IEEE.std_logic_1164.all;

entity pld_core is
    port(
    LRSTi   : in        STD_ULOGIC;
    LCLKi   : in        STD_ULOGIC;
    LA27i   : in        STD_ULOGIC;
    LA26i   : in        STD_ULOGIC;
    LBWEi   : in        STD_ULOGIC_VECTOR(3 downto 0);
    LBWEo   : out       STD_ULOGIC_VECTOR(3 downto 0);
    LBWEoe  : out       STD_ULOGIC;
    RDYi    : in        STD_ULOGIC;
    RDYo    : out       STD_ULOGIC;
    RDYoe   : out       STD_ULOGIC;
    BURSTi  : in        STD_ULOGIC;
    BURSTo  : out       STD_ULOGIC;
    BURSToe : out       STD_ULOGIC;
    LREQi   : in        STD_ULOGIC;
    LREQo   : out       STD_ULOGIC;
    LREQoe  : out       STD_ULOGIC;
    LBGRTo  : out       STD_ULOGIC;
    LBREQi  : in        STD_ULOGIC;
    DRAMWEi : in        STD_ULOGIC;
    DRAMOEi : in        STD_ULOGIC;
    CSNi    : in        STD_ULOGIC;
```

```
    READYo  : out       STD_ULOGIC;
    HOLDREQo: out       STD_ULOGIC;
    HOLDACKi: in        STD_ULOGIC;
    WBEi    : in        STD_ULOGIC_VECTOR(0 to 3);
    WBEo    : out       STD_ULOGIC_VECTOR(0 to 3);
    WBEoe   : out       STD_ULOGIC;
    XREQo   : out       STD_ULOGIC;
    XREQoe  : out       STD_ULOGIC;
    XACKi   : in        STD_ULOGIC;
    XSIZEo  : out       STD_ULOGIC_VECTOR(0 to 1);
    XSIZEoe : out       STD_ULOGIC;
    OEo     : out       STD_ULOGIC;
    WEo     : out       STD_ULOGIC
    );

end pld_core;

architecture BEHAVIOUR of pld_core is
type STATE_TYPE is (IDLE, START, CYCLE, UNAL_1, BURST, UNAL_2,
                    LAST, DIS, DUMMY, B2B, UN_1, UN_2, SINGLE);
signal STATE    :   STATE_TYPE;

-- DRAM_DIS is an internal signal used for disabling DRAM
signal DRAM_DIS :   STD_ULOGIC;

-- UCON is a one-bit counter for unaligned transfers
signal UCON     :   STD_ULOGIC;

begin
-------------------------
-- Master interface
-------------------------
    process
    begin
        wait until (LCLKi'event and (LCLKi='1'));
            case STATE is
                when IDLE =>
                    if (LBREQi='1') then
                        STATE <= START;
                    end if;
                when START =>
                    if (HOLDACKi='1') then
                        STATE <= CYCLE;
                    end if;
                when CYCLE =>
                    if (LREQi='1') then
                        STATE <= UNAL_1;
                    end if;
                when UNAL_1 =>
                    if (UCON='1') then
                        STATE <= UN_1;
                    elsif (BURSTi='1') then
                        STATE <= BURST;
                    else
                        STATE <= SINGLE;
                    end if;
                when BURST =>
                    if (XACKi='1') then
                        STATE <= UNAL_2;
                    end if;
                when UNAL_2 =>
                    if (UCON='1') then
                        STATE <= DIS;
                    elsif (BURSTi='1') then
                        STATE <= BURST;
```



```
                    else
                        STATE <= LAST;
                    end if;
                when LAST =>
                    if (XACKi='1') then
                        STATE <= DUMMY;
                    end if;
                when DUMMY =>
                    if (XACKi='1') then
                        STATE <= B2B;
                    end if;
                when UN_1 =>
                    if (XACKi='1') then
                        STATE <= UN_2;
                    end if;
                when UN_2 =>
                    if (XACKi='1' and BURSTi='0') then
                        STATE <= B2B;
                    elsif (XACKi='1' and BURSTi='1') then
                        STATE <= UNAL_1;
                    end if;
                when SINGLE =>
                    if (XACKi='1') then
                        STATE <= B2B;
                    end if;
                when B2B =>
                    if (LBREQi='1') then
                        STATE <= UNAL_1;
                    else
                        STATE <= IDLE;
                    end if;
                when DIS =>
                    if (XACKi='1') then
                        STATE <= DUMMY;
                    end if;
            end case;
-- Synchronous Reset
            if (LRSTi='1') then
                STATE <= IDLE;
            end if;
    end process;


-- generate XSIZE, BWE[0:3], and 1-bit unaligned counter UCON
    process (LBWEi, BURSTi,STATE)
    begin
    case LBWEi is
-- Byte transfer (big endian)
        when "1000"=>
            XSIZEo        <= "00";
            WBEo(2 to 3)  <= "00";
            UCON          <= '0';
        when "0100"=>
            XSIZEo        <= "00";
            WBEo(2 to 3)  <= "01";
            UCON          <= '0';
        when "0010"=>
            XSIZEo        <= "00";
            WBEo(2 to 3)  <= "10";
            UCON          <= '0';
        when "0001"=>
            XSIZEo        <= "00";
            WBEo(2 to 3)  <= "11";
            UCON          <= '0';
```

```
-- Half Word transfer
        when "1100"=>
            XSIZEo      <= "01";
            WBEo(2 to 3)<= "00";
            UCON        <= '0';
        when "0011"=>
            XSIZEo      <= "01";
            WBEo(2 to 3)<= "10";
            UCON        <= '0';

-- Full Word transfers
        when "1111" =>
            if (BURSTi = '0') then
                XSIZEo <= "10";       -- Single
            else
                XSIZEo <= "11";       -- Burst
            end if;
            WBEo(2 to 3)<= "00";
            UCON        <= '0';

-- unaligned transfers
        when "0101" =>
            XSIZEo      <= "00";
            WBEo(2 to 3)<= "01";
            UCON        <= '1';
            if (STATE = UN_2) then
                XSIZEo      <= "00";
                WBEo(2 to 3)<= "11";
                UCON        <=  '0';
            end if;
        when "0110" =>
            XSIZEo      <= "00";
            WBEo(2 to 3)<="01";
            UCON        <= '1';
            if (STATE = UN_2) then
                XSIZEo      <= "00";
                WBEo(2 to 3)<= "10";
                UCON        <= '0';
            end if;
        when "0111" =>
            XSIZEo      <= "00";
            WBEo(2 to 3)<= "01";
            UCON        <= '1';
            if (STATE = UN_2) then
                XSIZEo      <= "01";
                WBEo(2 to 3)<= "10";
                UCON        <= '0';
            end if;
        when "1001" =>
                XSIZEo      <= "00";
                WBEo(2 to 3)<= "00";
                UCON        <= '1';
            if (STATE = UN_2) then
                XSIZEo      <= "00";
                WBEo(2 to 3)<= "11";
                UCON        <= '0';
            end if;
        when "1010" =>
                XSIZEo      <= "00";
                WBEo(2 to 3)<= "00";
                UCON        <= '1';
            if (STATE = UN_2) then
                XSIZEo      <= "00";
                WBEo(2 to 3)<= "10";
                UCON        <= '0';
```



```
            end if;
        when "1011" =>
                XSIZEo      <= "00";
                WBEo(2 to 3)<= "00";
                UCON        <= '1';
            if (STATE = UN_2) then
                XSIZEo      <= "01";
                WBEo(2 to 3)<= "10";
                UCON        <= '0';
            end if;
        when "1101" =>
                XSIZEo      <= "01";
                WBEo(2 to 3)<= "00";
                UCON        <= '1';
            if (STATE = UN_2) then
                XSIZEo      <= "00";
                WBEo(2 to 3)<= "11";
                UCON        <= '0';
            end if;
        when "1110" =>
                XSIZEo      <= "01";
                WBEo(2 to 3)<= "00";
                UCON        <= '1';
            if (STATE = UN_2) then
                XSIZEo      <= "00";
                WBEo(2 to 3)<= "10";
                UCON        <= '0';
            end if;
        when others =>
            XSIZEo      <= "10";
            WBEo(2 to 3)<= "00";
            UCON        <= '0';
        end case;
    end process;

-- WBEo(0 to 1) will act like address A4 and A5 of PPC403GC

    WBEo(0) <= LA27i;
    WBEo(1) <= LA26i;

    WBEoe <= HOLDACKi;

    XSIZEoe <= HOLDACKi;

-- arbitration

    HOLDREQo <= LBREQi;
    LBGRTo <= HOLDACKi;

-- wait states imposed by RDYb (XACKb and RDYb need pull-up resistors)

    with STATE select
        RDYo <= XACKi when BURST | SINGLE | UN_2 | LAST,
                '0'   when others;
    RDYoe <= HOLDACKi;

-- DRAM_DIS used for disabling DRAM

    with STATE select
        DRAM_DIS <= '1' when DIS | DUMMY,
                    '0' when others;
    OEo <= DRAMOEi and not DRAM_DIS;
    WEo <= DRAMWEi and not DRAM_DIS;

-- XREQo is used to request an external cycle
```

```
    with STATE select
        XREQo <=    '1' when UNAL_1 | BURST | UNAL_2 | UN_1 | UN_2,
                    '0' when others;
    XREQoe <= HOLDACKi;

--------------------------------------------------------------------------------
-- Slave Interface
--------------------------------------------------------------------------------

-- Ready generated by the V360EPC
    READYo      <= (RDYi and CSNi);

-- Chip select signal from PPC403GC used to access V360EPC
    LREQo       <= CSNi;
    LREQoe      <= not HOLDACKi;

-- This version of the PPC403GC does not provide a "burst" signal
    BURSTo      <= '0';
    BURSToe     <= not HOLDACKi;

-- Byte enables with reverse order from PPC403GC
    LBWEo(3)   <= WBEi(0);
    LBWEo(2)   <= WBEi(1);
    LBWEo(1)   <= WBEi(2);
    LBWEo(0)   <= WBEi(3);
    LBWEoe      <= not HOLDACKi;

end BEHAVIOUR;
--================================================================================--
-- END of MODULE << pld_core >>
--================================================================================--
```

5.3 EXTERNAL MULTIPLEXER CODE

The multiplexer core logic is developed to address 4MB memory composed of two 1Mx16 bit DRAM modules - as illustrated earlier in Table 2. The code is quite simple, however, there is a requirement of 31 external I/O signals.

```
--================================================================================--
-- MODULE << amux >>
-- Description : Address MUX for 1Mx16x2 DRAM
--================================================================================--
library ieee;
use ieee.std_logic_1164.all;

entity amux is
  port(
      LA      : in    std_logic_vector (21 downto 2);
      AMuxCas : in    std_ulogic;
      MA      : out   std_ulogic_vector (9 downto 0)
  );
end amux;

architecture behavior of amux is

  signal LAi     :  std_ulogic_vector (21 downto 2);
  signal AMuxCasi:  std_ulogic;

begin
```



```
    LAi      <= To_StdUlogicVector(LA);
    AMuxCasi <= To_X01(AMuxCas);
process(LAi)
    begin
    if (AMuxCasi='1') then
        MA <= LAi(11 downto 2);  -- col address
    else
        MA <= LAi(21 downto 12); -- row address
    end if;
end process;
end behavior;
```

6. Synthesis

The protocol conversion code can be sysnthesized into a 32 macrocell CPLD (CY7C371i), using WARP™ synthesis compiler, from Cypress Semiconductor. Table 3 demonstrate the utilization of resources within the CPLD.

Table 3- CY7C371i CPLD macrocell utilization

Resources	Used	Max
Dedicated Inputs	3	3
Clock/Inputs	2	2
I/O Macrocells	28	32
Utilization	89%	100%

The address multiplexer code can also be synthesized into a similar CPLD. Although the amout of macrocell utilization would be quite low but notice that there is a requirement of 31 I/O pins for the multiplexer. Alternatively, both protocol convertion and multiplexer codes may be compiled into a larger 64 macrocell CPLD or integrated with existing FPGAs in the system.

7. Conclusion

Detailed interfacing and implementation issues between V360EPC (or V292PBC) from V3 Semiconductor and PPC403GC from IBM were described.

V360EPC provides an efficient and cost-effective solution for interfacing PPC403GC processor to the PCI bus. Higher performance PPC403GCX series may also be interfaced in the same fasion since the external bus interface is the same - internal 2x clock and 8x cache. The protocol conversion circuitry has been described in great details and can be easily realized within a conventional 32-macrocell CPLD.

V3 Semiconductor reserves the right to change and improve the contents of this documents without notice. For availability, samples and pricing please contact "v3info@vcubed.com". For technical support please contact "v3help@vcubed.com".

USA:
2348G Walsh Ave.
Santa Clara CA 95051
Phone: (408)988-1050 Fax: (408)988-2601
Toll Free: (800)488-8410 (Canada and U.S. only)
World Wide Web: http://www.vcubed.com

