Columbia University
Department of Electrical Engineering
EECS E4340. Laboratory #5B.
Interface Logic
Due: April 26, 2004

In this lab, you will complete the logic design of the PCI-DDR controller and interface by designing the interface logic that will exist between the DDR controller and the PCI core.

Based on the board design, we need to define some interfaces. At the very top level, your design should have the following interface:

```
entity ddr_pci_interface is
  port( -- PCI ports
    AD        : inout std_logic_vector(63 downto 0);
    CBE       : inout std_logic_vector( 7 downto 0);
    PAR       : inout std_logic;
    PAR64     : inout std_logic;
    FRAME_N   : inout std_logic;
    REQ64_N   : inout std_logic;
    TRDY_N    : inout std_logic;
    IRDY_N    : inout std_logic;
    STOP_N    : inout std_logic;
    DEVSEL_N  : inout std_logic;
    ACK64_N   : inout std_logic;
    IDSEL     : in    std_logic;
    INTR_A    : out   std_logic;
    PERR_N    : inout std_logic;
    SERR_N    : inout std_logic;
    REQ_N     : out   std_logic;
    GNT_N     : in    std_logic;
    RST_N     : in    std_logic;
    PCLK      : in    std_logic;
    -- System Ports
    sys_rst_n : in    std_logic;
    sys_clk   : in    std_logic;
    -- DDR Interface
    ddr_ad    : out   std_logic_vector(15 downto 0);
    ddr_dm    : out   std_logic_vector(1 downto 0);
    ddr_ba    : out   std_logic_vector(1 downto 0);
    ddr_rasb  : out   std_logic;
    ddr_casb  : out   std_logic;
    ddr_web   : out   std_logic;
    ddr_clk   : out   std_logic;
    ddr_clkb  : out   std_logic;
    ddr_dqs   : out   std_logic_vector(1 downto 0);
    ddr_csb   : out   std_logic;
    ddr_cke   : out   std_logic;
    ddr_dq    : inout std_logic_vector(15 downto 0)
    );
```

Your top-level design will instantiate your DDR interface, the PCI core, and some “bridge” or “glue” logic to connect them together. The ports of this glue logic will look something like:

```
entity pci_ddr_bridge is
  port (
    -- system interface
    sys_rst_n : in std_logic;
    fpga_clk  : in std_logic;
    -- pci interface
    pci_clk      : in    std_logic;
    pci_rst      : in    std_logic;
    pci_wrdn     : in    std_logic;
    pci_sdata    : in    std_logic;
    pci_base_hit : in    std_logic_vector(2 downto 0);
    pci_addr     : in    std_logic_vector(31 downto 0);
    pci_addr_vld : in    std_logic;
    pci_data_vld : in    std_logic;
    pci_ready    : out   std_logic;
    pci_term     : out   std_logic;
    pci_abort    : out   std_logic;
    pci_data     : inout std_logic_vector(63 downto 0);
    -- ddr interface
    ddr_ref_ack  : in  std_logic;
    ddr_data_vld : in  std_logic;
    ddr_data_i   : in  std_logic_vector(31 downto 0);
    ddr_data_o   : out std_logic_vector(31 downto 0);
    ddr_cmd      : out std_logic_vector(7 downto 1);
    ddr_addr     : out std_logic_vector(23 downto 0));
end pci_ddr_bridge;
```

`fpga_clk` is coming from the `clk_dlls` component of the DDR reference design and is used to clock the logic of the interface. Please remember that the PCI core (clocked by `pci_clk`) and the DDR interface (clocked by `fpga_clk`) are asynchronous with respect to each other. As a result, you must be careful to synchronize signals moving between these two domains.

Please implement a simple controller that implements (non-burst) read and write transfers. Your controller can (and probably should) incorporate the refresh counter that you previously added onto the DDR reference design. Following reset, you will want the controller to wait for the DLL's to lock, precharge the arrays, and initialize the mode registers.

If you want to be more ambitious, you can try to implement burst transfers, which will probably require you to implement a FIFO buffer. I would recommend that you get the basic read-write transfers working before attempting the burst-mode implementation.