This lab is a two-week lab where you will design a bus controller and memory access controller. During the first week, the basic goal of this lab is to design logic to design a state machine for an on-chip bus controller. During the second week, you will design modules that use the on-chip bus to access an off-chip SRAM. The rest of this handout describes the bus controller and the memory access portion will be given next week.

**Bus Controller**

The bus is modeled somewhat after PCI and the Wishbone SoC bus. The signals on the bus are as follows:

ADDR(31:0)
DATA(31:0)
REQ(N:0)
GNT(N:0)
FRAME
TRDY
IRDY
WRITE
CLK
RESET

Each module that is connected to the bus will have the following entity description. Additional signals to connect to the FPGA pins can be added if necessary.

```vhdl
entity bus280module
  port ( clk : in std_logic;
         reset : in std_logic;
         addr : inout std_logic_vector(31 downto 0);
         data : inout std_logic_vector(31 downto 0);
         ireq : out std_logic;
         ignt : in std_logic;
         irdy_i : in std_logic;
         irdy_o : out std_logic;
         trdy_o : out std_logic;
         trdy_i : in std_logic;
         tselect : in std_logic;
         write : inout std_logic;
```
The bus controller which you will need to design will have the following entity description:

```vhdl
entity bus280controller
  port ( clk : in std_logic;
         reset : in std_logic;
         addr : inout std_logic_vector(31 downto 0);
         data : inout std_logic_vector(31 downto 0);
         ireq : in std_logic_vector(N-1 downto 0);
         ignt : out std_logic_vector(N-1 downto 0);
         irdy_o : in std_logic_vector(N-1 downto 0);
         irdy : out std_logic;
         tselect : in std_logic_vector(N-1 downto 0);
         trdy_o : in std_logic_vector(N-1 downto 0);
         trdy : out std_logic;
         write : in std_logic);
end entity;
```

Notes:

N corresponds to the number of modules on the bus. For the purposes of this lab, assume there are only 4 modules, but it will be to your advantage to design a generic bus controller that can handle arbitrary numbers of modules. The `write`, `addr`, and `data` lines are shared lines. Thus, they must be driven to 'Z' when not in use by a module. This includes during a reset. The `trdy` and `irdy` lines are driven by the bus controller and connected to the `trdy_i` and `irdy_i` inputs on the modules.

The basic operation of the bus is as follows. When a module requests the bus, it will assert the `ireq` signal. A module that requests the bus is known as the `initiator`. The bus controller will then assert the `ignt` signal corresponding to the device that requested it. In the clock cycle following the `ignt`, the module can use the bus. Note, that the bus grant is done while another module may have control of the bus. This allows a module to start using the bus immediately after another module is done with it. The module can start using the bus by first checking `irdy`, and making sure that no one else is still using the bus. If the bus is clear, the module grabs control of the bus by asserting `irdy_o` and placing an address on the `addr` lines and setting the `write` line to '0' or '1' depending on if it’s a read or write respectively. If it is a write, the data should be placed on the `data` lines at this time. The bus controller will interpret the `addr` lines and assert the `tselect` bit corresponding to the `target` module referenced in the `addr`. The high order bits in `addr` are used to select the target. For example, if there are four modules in the system, you would use the top two bits to select the target. Thus, if the address is 0xC0001234, the top two bits identify module 3 as the target, and thus bit 3 of the `tselect` lines would be asserted. The `tselect` line is connected to the target module which lets it know that it should perform the requested operation. When it is done, it will assert `trdy_o` and place the data on the bus if it was a read. The initiator can then release the bus by deasserting `irdy_o` and `ireq`, or it can continue using the bus. However, it should check if `ignt` is still asserted, because the bus controller may have removed the grant to prevent a module from parking on the bus.

The bus controller functions by taking in `ireq` signals and asserting `ignt` signals on a round robin basis. If an `ireq` is asserted, the controller should assert the corresponding `ignt` signal. It
should then wait for the _irdy_ signal to go high, indicating that the initiator has started using the bus. At this point, the bus controller can re-arbitrate and check if any other module is requesting the bus. If so, it should assert the corresponding _ign_ signal. This should alert the current module on the bus that it should release the bus by deasserting its _irdy_o_ line. The module can still retain its _ireq_ signal if it wants to still use the bus.

Example of request and grant procedure.