Assignment 7 - Priority Interrupt Controller

Due April 3, 1998

Question 1

In this assignment you will write a synthesizeable VHDL description of a priority interrupt controller (PIC) that is a simplified version of the Intel 8259 PIC. This controller differs from a real 8259 in that it:

- uses a clock to latch interrupt requests at clock edges
- has a synchronous reset input
- always operates in the (fully nested) mode used in the IBM PC
- always has all interrupts enabled
- cannot be cascaded
- has a single-cycle interrupt-acknowledge (IACK) cycle
- has no status or configuration registers that can be read or written except for the interrupt type during IACK
- always supplies an interrupt type during IACK of 0x40 plus the interrupt request number (0x40 to 0x47)
- does not have a tri-state data bus output (the data bus is always an output)

The PIC has the following inputs:

`clk` - the CPU clock. Each CPU cycle (idle, write or IACK) takes one clock cycle. The cycle starts on the rising edge of this clock. The three other inputs may be assumed to be valid at the start of a CPU cycle and to meet all required setup times.

`reset` - reset input.

`wr*` - a write strobe which is asserted at the start of a write cycle (when an EOI is being written to the PIC, see below).

`inta*` - an interrupt acknowledge bus signal which is asserted at the start of an IACK cycle.

`ir (7 downto 0)` - interrupt request inputs, latched on rising edge of `clk` at the start of *all* cycles.

The PIC has the following outputs:

`int` - interrupt request. Is asserted when the PIC is requesting an interrupt.

`d(7 downto 0)` - data bus. Outputs the interrupt type for the CPU to read during an IACK cycle.

The states of the various interrupts can be defined by two 8-bit registers, the Interrupt Request Register (IRR) and the In-Service Register (ISR). Bits in the IRR indicate interrupts that have been requested but not acknowledged. Bits in the ISR indicate interrupts that have been acknowledged but for which the CPU has not yet sent the PIC an EOI instruction (i.e. their interrupt service routines have not completed).

Your design should be a synchronous Moore state machine (the outputs and internal state should change only at the rising clock edge and the inputs should only be sampled at these clock edges).

The behaviour of the PIC is as follows:

- both ISR and IRR are cleared if reset is asserted
- an IRR bit is set when its IR input is high
- the highest IRR bit is reset at the start of an IACK cycle
- the ISR bit corresponding to the highest IRR bit is set at the start of an IACK cycle (before the IRR bit is reset)

- the highest ISR bit is reset at the start of a write cycle (it is assumed that all writes to the PIC are EOIs)
- the data bus *always* outputs 0x40 plus the number (0-7) of the highest IRR bit. If no IRR bits are set then the output should be 0x47 (the lowest priority). This output need not be latched.
- the INT output should be asserted when the highest IRR bit is higher than the highest ISR bit. This output need not be latched.

The "highest bit" is the bit corresponding to the highest priority. The 8259's `IR0` line (least significant bit) has the highest priority. You may want to define a separate entity that implements this "highest bit" function and instantiate it in your design.

The above rules ensure that interrupt service routines can be nested but that only an interrupt request of a higher level than the one currently being serviced will cause an interrupt.

Use the following VHDL entity (signals ending in "_n" are active-low):

```
entity pic is
   port ( clk, reset, wr_n, inta_n : in std_logic ;
      ir : in std_logic_vector (7 downto 0) ;
      int : out std_logic ;
      d : out std_logic_vector (7 downto 0) ) ;
end ;
```

Write a synthesizeable VHDL description of this device. Test it using the test bench in `~elec379/asg7tb.vhd`. When it passes the test bench synthesize it using the class library as with previous assignments. Hand in a listing of your code, the simulation log file and a schematic.

Hints: (1) This circuit is sufficiently simple that your design can consist of only a data path with no controller entity. (2) The `and`, `or`, and `not` operations can be applied to `std_logic_vector`s to set and clear bits in registers.