



PRIORITY INTERRUPT CONTROLLER

FEATURES

- Any number of **interrupts**
- Configurable **priority levels**
- **Intel x86** compatible.
- **Edge** and **Level** triggered interrupts
- **Specific** and **Non-specific EOI** Commands

DESCRIPTION

Microcomputer system design requires that I/O devices such as keyboards, displays, sensors and other components receive servicing in an efficient manner so that large amounts of total system tasks can be assumed by the processor with little or no effect on the throughput.

The most common method of servicing such devices is the *Polled* approach. This approach requires the processor to test each device in sequence and in effect "ask" each one if it needs servicing. A program structured for polling, loops through a large section of code, without doing anything most of the time. The scan rate (or throughput) of the program is thus low and the number of tasks it performs are fewer. The cost effectiveness of using such devices is thus lowered.

A more desirable approach is one that allows the processor to execute its main program and stop to service peripheral devices only when requested by the device itself. This method requires an asynchronous input to the processor informing it to complete its current instruction and execute a routine that will service the requesting/interrupting device. On completing this routine the processor returns exactly where it left off.

This method is called interrupt. The P_INC core described here, functions as a manager in an Interrupt-Driven system environment. It accepts requests from peripheral devices, determines which of these is of highest priority, ascertains whether the incoming request has a higher priority than the one being serviced and issues an interrupt to the CPU. The core is a **dynamic** design, requiring a clock input in contrast to a **static** design that doesn't require a clock input. The clock input along with a separate bus for the interrupt vector make it especially suited for an **SOPC** implementation in a **CPLD** or **FPGA**.

APPLICATION

FPGA or ASIC based embedded systems for applications such as DRAM refresh, interfacing with communication ports, keyboard, mouse, hard disk, RTCC chip, etc.



VHDL Component Declaration:

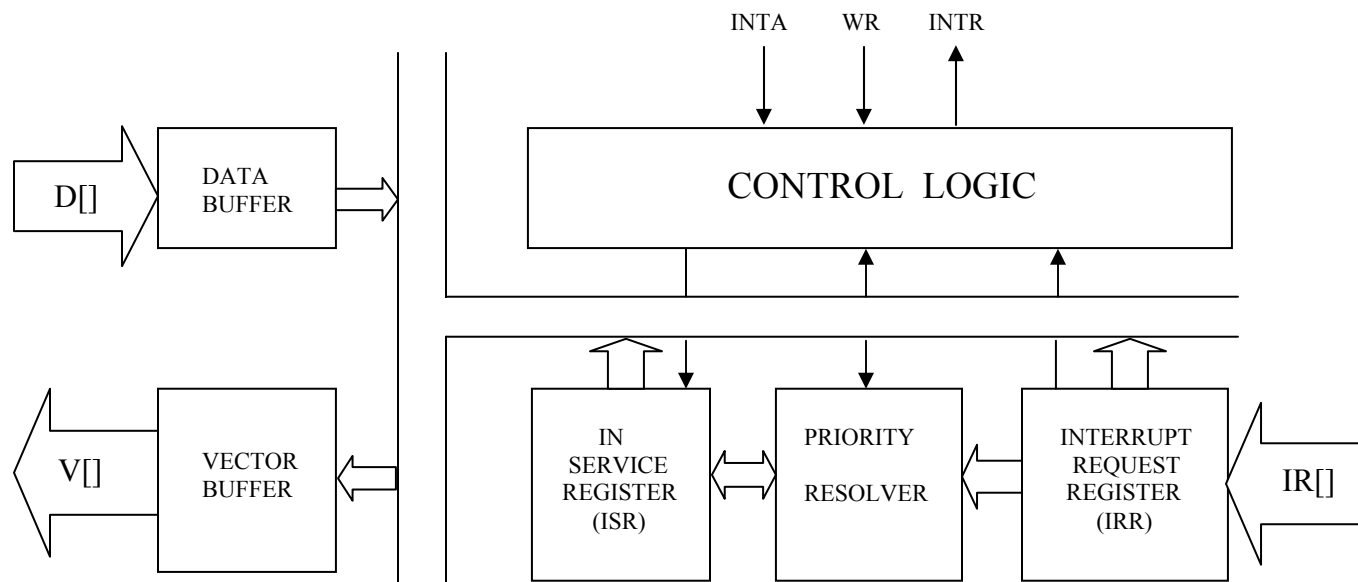
```
COMPONENT P_INC
  GENERIC (
    NUM    : INTEGER :=0;
    EDGE   : INTEGER :=1;
    FWD    : INTEGER :=1;
    DBW    : INTEGER :=0
  );
  PORT (
    IR     : IN      BUS1D(NTI(NUM) DOWNT0 0);
    D      : IN      BUS1D(NTI(DBW) DOWNT0 0);
    WR     : IN      NODE;
    RST    : IN      NODE;
    CLKI   : IN      NODE;
    INTA   : IN      NODE;
    V      : BUFFER  BUS1D(NTI(DBW) DOWNT0 0);
    INTR   : BUFFER  NODE
  );
END COMPONENT; COMPONENT M_PAD IS
```

FILES YOU GET

i) FUNC.DOC	-	Documentation of functions & data types used in the core.
ii) README.DOC	-	Compile and licensing information.
iii) PINC.DOC	-	This document
a) MYLIB.VHD	-	PACKAGE
b) P_INC.vhd	-	TOP HIERARCHY DESIGN FILE
c) S_JKF.vhd	-	DESIGN FILE BELOW TOP HIERARCHY
d) S_DFF.vhd	-	-DO-
e) R_SL.vhd	-	-DO-
f) D_EC0D.VHD	-	-DO-
g) P_AD.vhd	-	-DO-
h) M_DFF.vhd	-	-DO-
i) N_TOP.vhd	-	-DO-
j) D_EMUX.vhd	-	-DO-
k) B_DFF.vhd	-	-DO-
l) M_YMUX.vhd	-	-DO-
m) M_JKF.vhd	-	-DO-



BLOCK DIAGRAM



PRIORITY RESOLVER

This logic block determines the priorities of the bits set in IRR. The highest priority is selected and strobed into the corresponding bit of the ISR at the end of the first INTA pulse.

INTERRUPT REQUEST REGISTER (IRR) AND INTERRUPT SERVICE REGISTER (ISR)

Interrupts at the IR[] inputs are handled by these two registers in cascade. The IRR is used to store all the interrupt levels requesting service and the ISR is used to store all the interrupt levels being serviced.

INTERRUPT REQUESTS

An Interrupt request is an event on the Interrupt Request input lines IR[]. These may be edge or level sensitive, depending on the setting of the EDGE parameter.

Edge Triggered

- ❑ Clocks into IRR only once, on its rising edge.
- ❑ Is recorded irrespective of its pulse width.

Level Triggered

- ❑ Clocks into IRR at the rising edge of every CLKI input.
- ❑ Must remain high until the end of the first INTA pulse. If it drops off prematurely, one of two scenarios are possible :-
 - ❑ If IR[] width is less than one CLKI pulse, INTR will stay low and the event on the IR[] input will be ignored by the CPU.
 - ❑ If IR[] is more than 1 CLKI pulse wide but drops off before the rising edge of the first INTA, INTR will go high, If the priority criteria of this IR[] input are met, and the interrupt will not be ignored by the CPU. However P_INC will send a pointer value of zero on the V[] outputs without affecting the ISR and IRR registers. IR[0] input may thus be left connected to GND, reserved for spurious interrupts, with an empty subroutine.



INTERRUPT SEQUENCE

- 1> An event occurs on one or more of the Interrupt Request lines, as described above, setting the corresponding IRR bits.
- 2> The P_INC core evaluates these requests and sends an INTR to the CPU, if appropriate, one or two CLKI pulses later.
- 3> The CPU acknowledges the INTR and responds with an INTA pulse
- 4> On receiving an INTA, the highest priority ISR bit is set and the corresponding ISR bit is reset. The V[] output is "DON'T CARE" in this cycle.
- 5> The CPU will initiate a second INTA pulse. During this pulse, the P_INC core releases a pointer on the V[] outputs, from where it is read by the CPU. At the end of this pulse the INTR output is made low.
- 6> This completes the interrupt cycle. The ISR bit remains set until an appropriate EOI (End Of Interrupt) command is issued at the end of the interrupt subroutine.

END OF INTERRUPT (EOI) COMMAND

The EOI command is sent by the interrupt subroutine to indicate its end. It is sent on the data bus with the decoded port write signal (WR). Each bit in the EOI command corresponds to a bit in the ISR register and the IR[] input port. A hi bit in the EOI command will reset the corresponding bit in the ISR. The EOI command is 'Specific' if only one bit is hi and 'Non-specific' if all bits are hi.

PORTS AND PARAMETERS

INPUT PORTS – Table 1

NAME	DESCRIPTION	WIDTH	COMMENTS
IR[]	Interrupt request	NUM	Active high . See " INTERRUPT REQUESTS" above
D[]	Data bus	DBW	Decoded EOI command. See "EOI Command" above
WR	Write enable	1	Active hi. Decoded I/O write signal for EOI command. "EOI Command" above
RST	Reset	1	Active lo. Asynchronous, resets all internal logic
CLKI	Clock	1	CPU clock or synchronous to the CPU clock. Positive edge triggered. Synchronizes all internal operations
INTA	Interrupt Acknowledge	1	Active low. Interrupt acknowledge from CPU. Synchronous to CLKI. Minimum width = 1CLKI pulse. Two pulses must follow every INTR output.

OUTPUT PORTS– Table 2

NAME	DESCRIPTION	WIDTH	COMMENTS
V[]	Interrupt Vector	DBW	Offset in memory, of interrupt subroutine
INTR	Interrupt to CPU	1	Active hi. Synchronous to CLKI. As described above INTR goes hi 1 or 2 CLKI pulses after an appropriate interrupt request. After the end of an interrupt cycle (as described above) the INTR line will go high only for higher priority interrupts, until the EOI command. One CLKI pulse after an EOI command it will go hi for any pending interrupt (in IRR), irrespective of its priority. INTR goes low at the falling edge of the second INTA from the CPU.

PARAMETERS – Table 3 (all INTEGER type, except where specifically mentioned)

NAME	MIN	DESCRIPTION
NUM	0	Width of IR[] input port.
EDGE	0	IR[] inputs are level (0) or edge (1) triggered. See " INTERRUPT REQUESTS" above
FWD	0	Decides order of priority FWD=1 => IR[0] has highest and IR[NUM-1] lowest priority FWD=0 => IR[0] has lowest and IR[NUM-1] highest priority



DBW	0	Width of D[] input port and V[] output port.
-----	---	--

SAMPLE DESIGN-CODING

```
LIBRARY IEEE;  
USE IEEE.STD_LOGIC_1164.ALL;  
USE IEEE.STD_LOGIC_UNSIGNED.ALL;  
USE IEEE.STD_LOGIC_ARITH.ALL;
```

```
LIBRARY MYLIB;  
USE MYLIB.MYLIB.ALL;
```

```
ENTITY METOU IS
```

```
    PORT (IR           :IN      BUS1D(2 DOWNTO 0);  
          D           :IN      BUS1D(15 DOWNTO 0);  
          WR          :IN      NODE;  
          RST         :IN      NODE;  
          CLKI        :IN      NODE;  
          INTA        :IN      NODE;  
          V           :BUFFER  BUS1D(15 DOWNTO 0);  
          INTR        :BUFFER  NODE);
```

```
END METOU;
```

```
ARCHITECTURE METOU OF METOU IS
```

```
BEGIN
```

```
A0: M_PAD GENERIC MAP (NUM=>3, EDGE=>0, FWD=>1, DBW=16)
```

```
--BASE ADDRESS VALUE
```

```
    PORT      MAP (IR, D, WR, RST, CLKI, INTA, V, INTR);
```

```
END METOU;
```

SAMPLE DESIGN-TIMING SIMULATION

