VHDL Implementation of Interrupt Controller

Aathira.R[1] Chithra.M[2] Jamshitha.M[3] J.S.Devi[4]
[1,2,3,4]Student
[1,2,3,4]Department of Electronics and Communication Engineering
[1,2,3,4] Nehru College of Engineering and Research Centre

Abstract— This project aims at the implementation of Interrupt Controller using VHDL (Very High Speed Integrated Circuit Hardware Description Language). The Interrupt Controller is an integral part of the computer system. It accepts all the interrupts generated, selects the interrupt with the highest priority and generates the appropriate redirected location. The 8259A is one of the most popular interrupt controllers in use today. The 8259A manages 8interrupts according to the instructions written into its control registers. It is designed to minimize the software and real time overhead in handling multi-level priority interrupts. It has several modes, permitting optimization for a variety of system requirements. In this project, the 8259A is designed to handle 15 vectored interrupts using VHDL.VHDL is chosen as the synthesis methodology for the ASIC (application-specific integrated circuit) design. The key advantage of VHDL, when used for systems design, is that it allows the behaviour of the required system to be described (modelled) and verified (simulated) before synthesis tools translate the design into real hardware (gates and wires). Another benefit is that VHDL allows the description of a concurrent system. VHDL is a dataflow language, unlike procedural computing languages such as BASIC, C, and assembly code, which all run sequentially, one instruction at a time. Through this project, the Interrupt Controller is intended to be designed in VHDL and tested using simulation in Model Sim.

Key words: VHDL, Interrupt controller, Model Sim, Interrupts.

I. INTRODUCTION

When a digital computer operates, certain rare events or exception conditions such as an addition overflow, a parity error, a request from an I/O operation, or a signal from a real-time input may occur. These events and conditions are quite infrequent and unpredictable. However, whenever they happen, they must be attended to by the CPU as quickly as possible. In the early days, a program loop was used for testing and waiting for an event to occur; this resulted in great inefficiency of hardware utilization. This has now been replaced by interrupts. When an interrupt occurs, the event is handled partly by hardware and partly by software. As the time period for handling the interrupts should be very short, so that next interrupt can be handled with little delay, a separate entity the "Interrupt Controller" is utilized. The interrupt controller generates and handles the interrupts that occur in a digital computer system.

In this paper an interrupt controller handling 15 vectored interrupts is modeled using VHDL. VHDL is chosen as the language to implement the system design as it is a powerful and versatile hardware description language with readily available tools. The interrupt controller has been implemented as two sub-blocks – the interrupt generator and the interrupt handler. The interrupt generator receives the interrupt signals sent to the processor and the interrupt handler utilizes a priority encoder, to encode the interrupts that are pending to the respective sub-routine addresses. The controller blocks are compiled and tested using test benches and by using Model Sim, it is simulated to obtain the desired functionality of the interrupt controller.

II. LITERATURE SURVEY

[1] The description of complex systems during the design procedure requires hierarchically structured abstractions of a system with its components that allow different views of the system. The different levels are characterized by an increasing variety of details and decreasing transparency the higher they are in hierarchical order. In the design process the system goes through behavioural, structural and physical forms of description and graphical representation. The behavioural description shows how operations or transformations are applied to input data (input information), how these input data are transferred into "intermediate data" internally and on which way they are represented in output data (output information). The structural description shows the topological structures in a system. The bigger components are assembled by smaller structures and modules.

[2] Interrupt is an event external to the currently executing process that causes a change in the normal flow of instruction execution; usually generated by hardware devices external to the CPU. Interrupts give each device a wire (interrupt line) that it can use to signal the processor. When interrupt is signalled, processor executes a routine called an interrupt handler to deal with the interrupt and no overhead when no requests are pending.

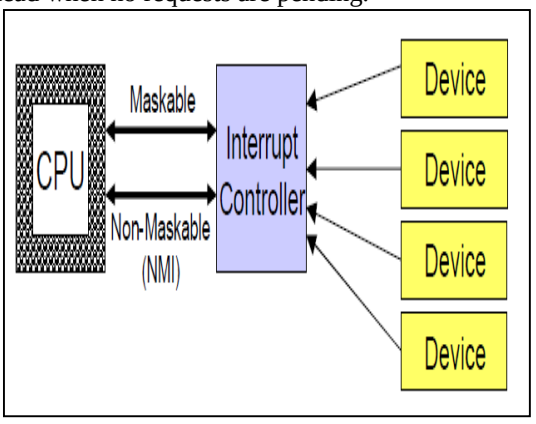

Fig. 1: Interrupt controller

Interrupt is a signal send by an external device to the processor, to the processor to perform a particular task or work. Mainly in the microprocessor based system the interrupts are used for data transfer between the peripheral and the microprocessor. When a peripheral is ready for data transfer, it interrupts the processor by sending an appropriate signal to the interrupt pin of the processor. If the processor accepts the interrupt then the processor suspends its current

activity and executes an interrupt service subroutine to complete the data transfer between the peripheral and processor. After executing the interrupt service routine the processor resumes its current activity. This type of data transfer scheme is called interrupt driven data transfer scheme.

A. VHDL syntax: [3]

1) VHDL is case-insensitive. There are many capitalization styles. I prefer all lower-case. You may use whichever style you wish as long as you are consistent.
2) Everything following two dashes "--" on a line is a comment and is ignored.
3) Statements can be split across any number of lines. A semicolon ends each statement. Indentation styles vary but an "end" should be indented the same as its corresponding "begin"
4) Entity and signal names begin with a letter followed by letters, digits or underscore (" ") characters.

III. DESIGN OVERVIEW

The logic design of 8259A depicts the total blue print of the proposed project. The total essence and functioning of the project is represented in a single design block.

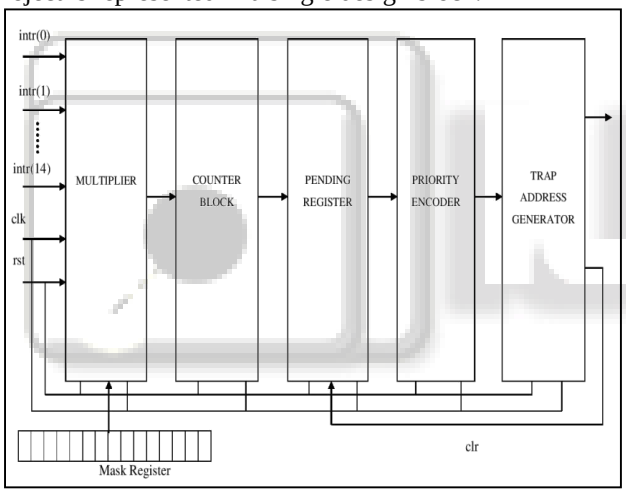

Fig. 2: Logic diagram

A. Logic Diagram Description

1) Mask Register

The interrupt mask register can be used to enable or mask out individually the interrupt request input. It contains 15 bits identified by mask (0) through mask (14). These bits correspond to interrupt request inputs intr(0) through intr (14) respectively. Logic 0 in mask register bit position mask it out and logic 1 enables the corresponding interrupt input.

2) Multiplier

The inputs of this block are incoming interrupts identified by INT0 to INT14 and their corresponding mask register bits from the mask register. The operation of this block determines the interrupt to be masked. For this AND operation is carried out between interrupts and mask bits.

The output of this block is interrupt mask out (0) to interrupt mask out (14). Logic 0 indicates that the corresponding interrupt is masked and logic 1 enables the interrupt.

3) Counter Block

A signal is treated as interrupt only if it remains active for three or more clock pulses. This reduces the possibility of accepting momentary glitches as interrupts. For this purpose counter block is introduced. This block checks whether the interrupts are high for atleast three clock pulses.

4) Pending Register

The main purpose of pending register is to store interrupts. All the interrupts that are high for atleast three clock cycles are stored in pending register as pend (0) to pend (14). The inputs of this block are interrupts and clear signals. The clear signals clr (0) to clr (14) are generated by trap address generator. Once the interrupt is serviced, its corresponding clear signal clears the respective pend bit.

5) Priority Encoder

Priority encoder is used to encode the interrupts based on its priority. Here priority increases with the interrupt bit index, that is, interrupt 15 is given more priority than interrupt 14.

6) Trap Address Generator

The interrupt to be serviced is selected by the priority encoder and is given to trap address generator as the input. The trap address generator generates the corresponding address from the memory map along with the corresponding clear signal which serves as input to the pending register.

IV. FUNCTIONAL DESCRIPTION

The interrupt controller has been modeled as:

a) Interrupt generator
b) Interrupt handler

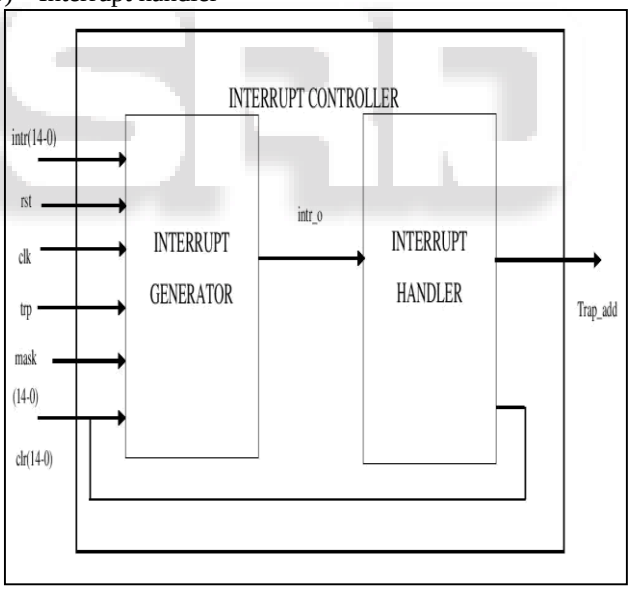

Fig. 3: Interrupt Controller Internal Working

The input signals are that of reset, clock , interrupt signals-intr, mask register-mask and clear register-clr.

The output of the interrupt controller is the address or pointer to the location of the service routine for the interrupt detected.

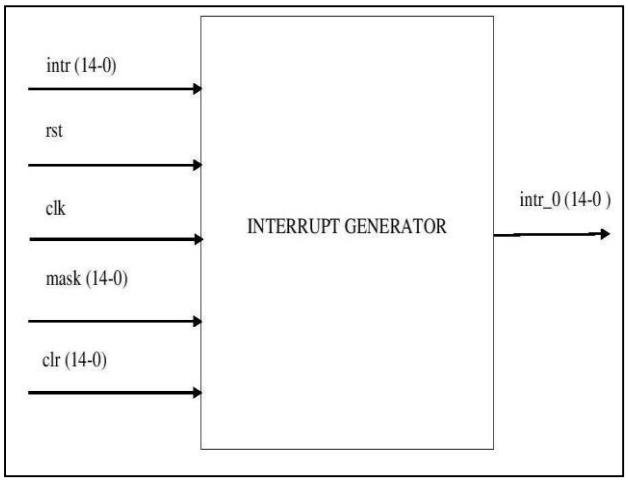

Fig. 4: Interrupt Generator I

The interrupt generator accepts the INTA signals from the processor. These are represented as the IRR_INP vector, one bit corresponding to each of the 16 interrupts. The interrupt generator enables the interrupt request only if the interrupt signal remains HIGH for at least three clock pulses. The major problem face in interrupt controllers is that most of the times a glitch or temporary change in the interrupt signals from the processor, due to noise, causes the interrupt request to be enabled. This causes unnecessary interrupt procedure, thereby consuming time and width of the processor's bus. In order to eliminate false requests, the interrupt generator observes the interrupt request signals IRR_INP from the processor for three clock pulses. If the signal remains HIGH then it is taken as a valid request and the IRR bit corresponding to the bit in the IRR_INP is SET. The interrupt request is now compared with the mask register. The mask register, IMR, contains 16 bits, according to which the interrupts are either masked or not. In this particular design of the interrupt generator, the interrupt corresponding to mask bit=0 is neglected, and that with mask bit=1 is considered valid.

The IMR and IRR registers are compared and the resulting register is then subjected to the priority encoder.

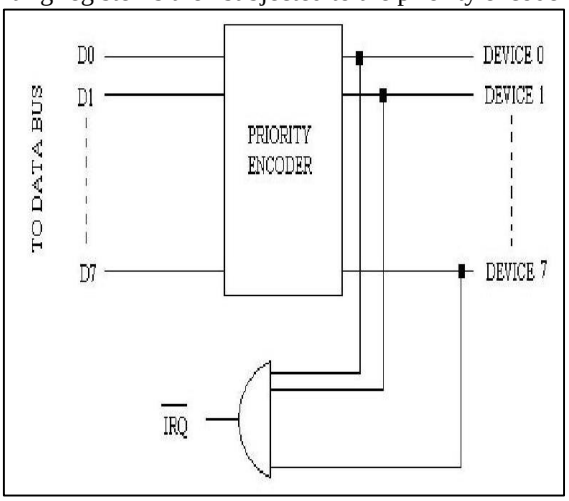

Fig. 5: Interrupt Generator II

The priority encoder checks each of the interrupts in the order of the priority, from the highest to the lowest and thereby generates the Interrupt service register (ISR).This is the output of the first stage of the interrupt generator.

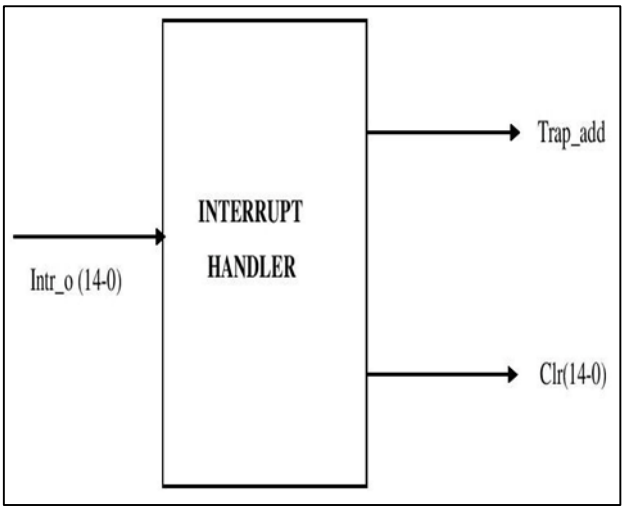

Fig. 6: Trap Handler

The trap handler must generate the vectored location based on the interrupt that has been generated in the previous stage. The vectored location is generated from the interrupt vector table. In the project, 16 interrupts have been taken into consideration. The main memory map implemented is given as follows:

INTERRUPTS	ADDRESS
intr(0)	0000100000000000
intr(1)	0101010101010101
intr(2)	1111111111111111
intr(3)	0000011111000001
intr(4)	0000011111001001
intr(5)	0000011111100001
intr(6)	0000011111111001
intr(7)	0000011111000001
intr(8)	0000011111111111
intr(9)	1000011111000001
intr(10)	1100011111000001
intr(11)	1110011111000001
intr(12)	1111011111000001
intr(13)	1111111111000001
intr(14)	0000011001000001

Table 1: interrupt vector table

After generating the 8 bit address, the bit corresponding to the interrupt that was serviced is set to 1 in the clear register and sent as a feedback to the interrupt generator so that the interrupt process is terminated and the bus control is given back to the processor. The two modules are then integrated in to a top level file where together they function as the interrupt controller.

V. SOFTWARE DETAILS

ModelSim is a multi-language HDL simulation environment by Mentor Graphics, for simulation of hardware description languages such as VHDL, Verilog and SystemC, and includes a built-in C debugger. ModelSim can be used independently, or in conjunction with Altera Quartus or Xilinx ISE. Simulation is performed using the graphical user interface (GUI), or automatically using scripts.

A. Editions

ModelSim is offered in multiple editions, such as ModelSim PE, ModelSim SE, and ModelSim XE. ModelSim SE offers high-performance and advanced debugging capabilities, while ModelSim PE is the entry-level simulator for hobbyists and students. ModelSim SE is used in large multi-million gate designs, and is supported on Microsoft Windows and Linux, in 32-bit and 64-bit architectures. ModelSim XE stands for Xilinx Edition, and is specially designed for integration with Xilinx ISE. ModelSim XE enables testing of HDL programs written for Xilinx Virtex/Spartan series FPGA's without needed physical hardware. ModelSim can also be used with MATLAB/Simulink, using Link for ModelSim. Link for ModelSim is a fast bidirectional co-simulation interface between Simulink and ModelSim. For such designs, MATLAB provides a numerical simulation toolset, while ModelSim provides tools to verify the hardware implementation & timing characteristics of the design. ModelSim uses a unified kernel for simulation of all supported languages, and the method of debugging embedded C code is the same as VHDL or Verilog. ModelSim enables simulation, verification and debugging for the following languages:

1) VHDL
2) Verilog
3) Verilog 2001

B. Basic Simulation Flow

The following diagram shows the basic steps for simulating a design in ModelSim.

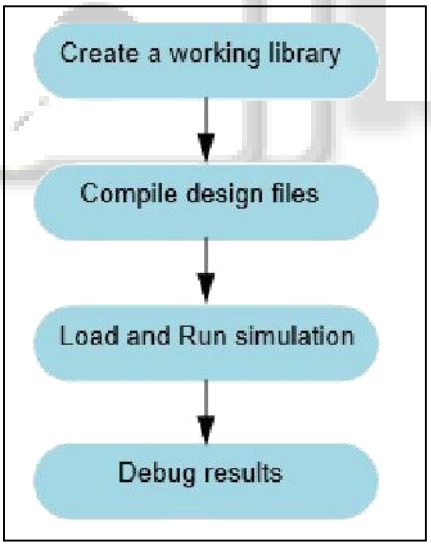

Fig. 7: Basic Simulation Flow

1) Creating the Working Library
2) Create a working library
3) Compile design files
4) Load and Run simulation
5) Debug results

C: In ModelSim, all designs are compiled into a library. You typically start a new simulation in ModelSim by creating a working library called "work," which is the default library name used by the compiler as the default destination for compiled design units.

1) Compiling Your Design

After creating the working library, you compile your design units into it. The ModelSim library format is compatible across all supported platforms. You can simulate your design on any platform without having to recompile your design.

2) Loading the Simulator with Your Design and Running the Simulation With the design compiled,

you load the simulator with your design by invoking the simulator on a top-level module (Verilog) or a configuration or entity/architecture pair (VHDL). Assuming the design loads successfully, the simulation time is set to zero, and you enter a run command to begin simulation

3) Debugging Your Results

If you don't get the results you expect, you can use ModelSim's robust debugging environment to track down the cause of the problem.

VI. Applications

The guidance system of a rocket includes very sophisticated sensors, on-board computers, radars, and communication equipment. The guidance system has two main roles during the launch of a rocket; to provide stability for the rocket, and to control the rocket during maneuvers.

Making a rocket stable requires some form of control system. Controls on rockets can either be active or passive. Passive controls are fixed devices that keep rockets stabilized by their very presence on the rocket's exterior. Active controls are the ones performed from the earth station while the rocket is in flight to stabilize and steer the craft. The changes in the centre of gravity and the flight map of desired path are parameters according to which the flight of the rocket must be controlled. The information of the sensors after interpretation has to be enforced on the rocket. This is done through interrupts. The interrupt signal sent to the on-flight communications chip, is directed to the interrupt controller which effectively generates the sequence for maneuvering the craft. The interrupt controller is a very effective system as it saves the processor the time and bandwidth required to generate and handle interrupts up in space.

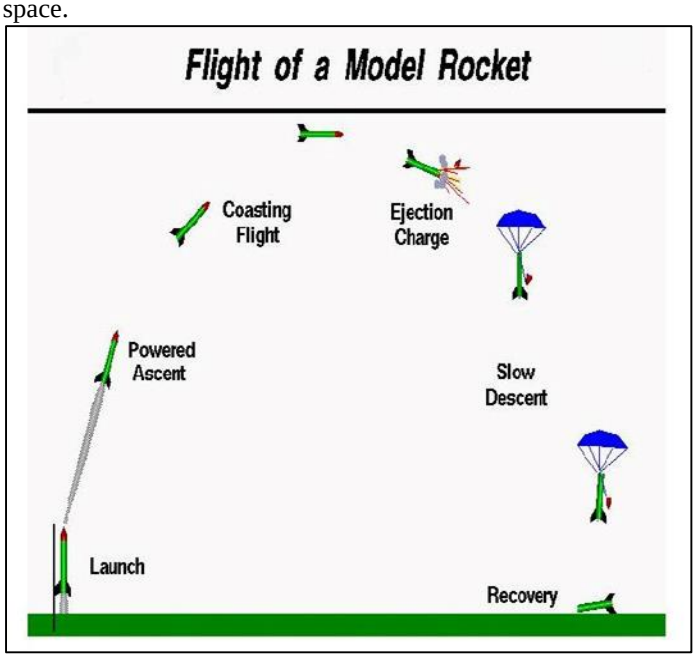

Fig. 8: Flight of A Model Rocket

The same is used to reposition satellites. The use of interrupts and controllers will greatly increase with the need of reusable space craft's that require the crafts to be recovered from space. The flight control is effectively interrupt control, the better the interrupt control system, closer the craft follows the actual flight plan through external interrupts from the ground station as well as timed software interrupts.

VII. CONCLUSION

The logical implementation of interrupt controller is done using VHDL. Our code is able to take 15 interrupts and select the most prior interrupt among them. It also generates its corresponding address from the memory map.

VIII. FUTURE SCOPE

The project may be used for the FPGA implementation of 8259A interrupt controller. In our project we implemented non- nested mode of 8259A interrupt controller, by making sufficient changes and advancements in the code we could implement nested mode of 8259 A.

REFERENCES

[1] D.Tavangarian, "VHDL-Based Simulation of Electronic Circuits and Systems", Proceedings of the 3.EUROCHIP Workshop on VLSI Design Training, Grenoble, vol 11-No.5, May 1992.
[2] V.Udayashankara, M.S.Mallikarjunaswamy, "Microcontroller-Hardware, Software and Applications", Tata McGraw Hill Publications, New Delhi.
[3] Phil Croucher, "The Bios Companion", Electrocution Technical Publishers, vol 2, May 2000.

