DESIGN AND DEVELOPMENT OF AXI BASED MULTI CHANNEL INTERRUPT CONTROLLER2

K.NAGA SAILAGA*

TARANGINI.K**

ABSTRACT:

Processors and peripherals usually communicate with each other with the use of interrupt. Number of peripherals and processors are increasing and the processors have limited interrupt ports which is far less than the total interrupt signals of peripherals and other co-processors in a System-on-Chip (SoC). Flexibility of interrupt controller is more and more concerned with the development of multi-million gate SoC.A configurable multichannel interrupt controller is proposed to solve this problem. Interrupt priority is configurable for processor by accessing registers through AXI (Advance extensible Interface) interface. The AMBA3 AXI is chosen protocol defines a unidirectional channel architecture, which enables the efficient use of register slices to pipeline the connection for higher speeds, or to enable the use of multiple clock domains for low power Combination interrupt is also realized for one Interrupt Service Routing (ISR) to service multiple interrupts at a time. Up to 60 interrupt inputs and 12 interrupt channels are supported in This design and by using this design in many Embedded system design applications and by this in one soc we connected number of peripherals device in one system on chip to increases the performance of the system. Verilog is a hardware description language (HDL) used for simulation of proposed algorithm.

Keywords: SOC, AMBA, ISR, AXI, INTERRUPT

* Assistant Professor, ECE, Vidya Jyothi Institute of Technology, Hyderabad

** Assistant Professor, ECE, SRINIDHI Institute of science andTechnology, Hyderabad

I. INTRODUCTION:

Design of System-on-Chip (SoC) receives a great deal of attention in recent days. The number of peripheral used in one SoC becomes larger and larger, multiprocessor System-on Chip (MPSoC) is also widely used. Processors and peripherals usually communicate with each other with interrupt. With the development of SoC and MPSoC technique, the communication between processors and peripherals becomes a problem as processors have limited interrupt ports which is far less than the total interrupt signals of peripherals and other processors. Interrupt controller is used to manage the numerous interrupts and assert the expected interrupt to processors.

An interrupt controller can capture interrupt signals from peripherals and processors. Interrupt signals are sorted by priority in the interrupt controller. Interrupt with the highest priority level is asserted to processor for interrupt processing. Many structures of interrupt controller s are neither priority configurable nor interrupt-combinable.

An interrupt controller can capture interrupt signals from peripherals and processors. Interrupt signals are sorted by priority in the interrupt controller. Interrupt with the highest priority level is asserted to processor for interrupt processing. Many structures of interrupt controller have been proposed by former papers "A high-precision timing and interrupt controller to support distributed real-time operating systems," and “A self timed interrupt controller a case study in asynchronous micro-architecture design," but they are neither priority configurable nor interrupt-combinable

In this paper a priority-configurable and interrupt combinable interrupt controller is proposed. Priority of interrupts can be configured by software. Combination of up to 16 interrupts is also supported. This interrupt controller has 60 interrupt inputs and 12 interrupt outputs. The processor can access interrupt controller through AHB bus

1.1 Block Diagram and Description:

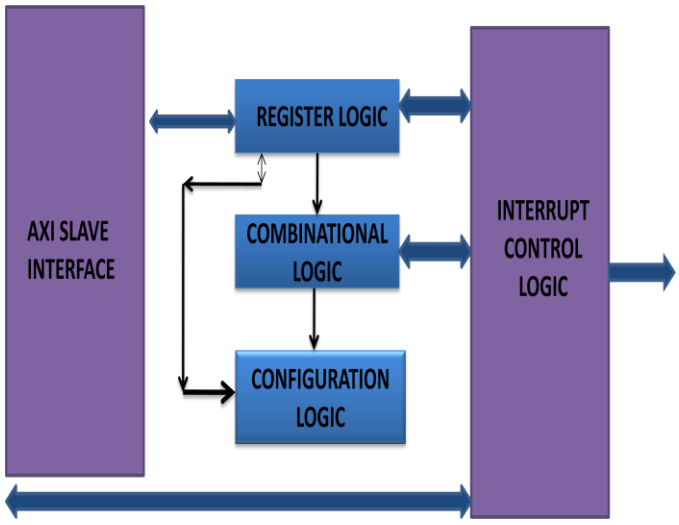


Fig 1.1 Block Diagram of Design of A Configurable Multichannel Interrupt Controller Module

1.2 Module Description:

1.2.1 AXI Slave Interface module:

This module generates necessary control signals to register module address and data width is 32bits and AXI Lite is used

1.2.2 Register Module:

The register module is used to program and control operation of multiple interrupt controllers.

1.2.3 Combinational Logic:

The combinational logic module used to combine the incoming interrupts and its supports combination up to 60 interrupt.

1.2.4 Configuration Logic:

The configuration logic module is used to dynamically change the priority of the interrupts

1.2.5 Interrupt control logic:

This Interrupt control logic it is heart of the core and it generates IRQ from the incoming interrupt and depending upon the controlled signals from the register

2. INTERRUPTS:

2.1 Polled Approach Method:

Sometimes it is necessary to have computer automatically execute one of a collection of the special routine when ever certain conditions exist within a program or in the microcomputer

system. For example, if it is necessary that microcomputer system should give response to devices such as keyboard, sensors and other components when they request for service.

The most common approach method of servicing such device is the **polled approach**. This is where the processor must test each device in sequence and in effect"ask" each one if it needs communication with the processor .It is easy that a large portion of the main program is looping through this continuous polling cycle .Such a method would have a various decremental effect on system throughput, thus limiting the tasks that could be assumed by the microcomputer and reducing the cost effectiveness of using such device

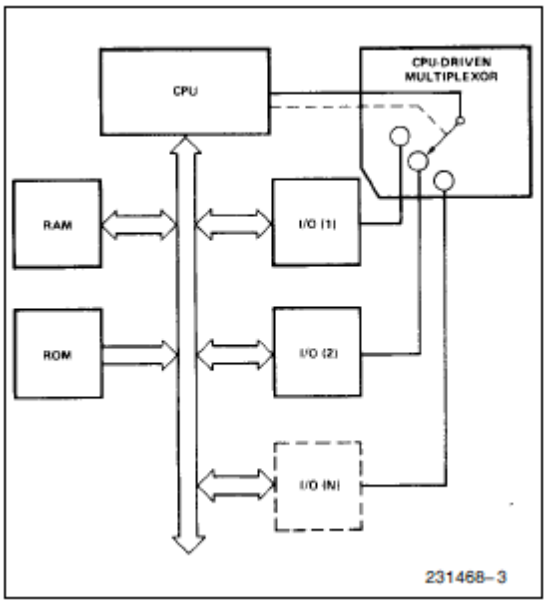


Fig 2.1 polled approach method

2.1.1 Interrupt Method: An interrupt is a signal from a device attached to a computer or from a program within the computer that causes the main program that operates the computer (the operating system) to stop and figure out what to do next. Almost all personal (or larger) computers today are interrupt-driven - that is, they start down the list of computer instruction s in one program (perhaps an application such as a word processor) and keep running the instructions until either (A) they can't go any further or (B) an interrupt signal is sensed. After the interrupt signal is sensed, the computer either resumes running the program it was running or begins running another program.

Basically, a single computer can perform only one computer instruction at a time. But, because it can be interrupted, it can take turns in which programs or sets of instructions that it

performs. This is known as multitasking. It allows the user to do a number of different things at the same time. The computer simply takes turns managing the programs that the user effectively starts. Of course, the computer operates at speeds that make it seem as though all of the user's tasks are being performed at the same time. (The computer's operating system is good at using little pauses in operations and user think time to work on other programs. An operating system usually has some code that is called an interrupt handler. The interrupt handler prioritizes the interrupts and saves them in a queue if more than one is waiting to be handled. The operating system has another little program, sometimes called a scheduler that figures out which program to give control

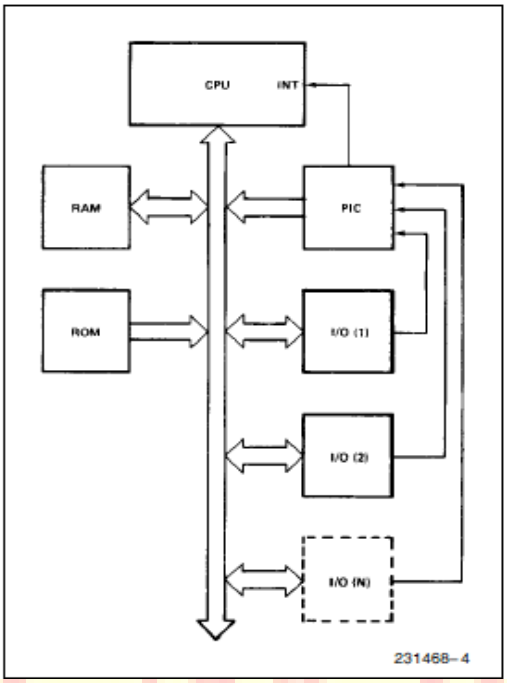


Figure 2.1.1Interrupt method

2.2 Interrupt Mechanism:

At the beginning of each FDE cycle, each bit in the interrupt register is checked in turn. This register is a special register in the CPU that takes note of when an interrupt has happened. Each bit in the register represents a different kind of interrupt. If a bit has been set, that would indicate an interrupt has happened! The CPU has to decide whether to service the interrupt immediately, or leave it till later.

For example, if 2 interrupts have happened at the same time, one of them has to wait! Which one? That depends upon which one is the least important! Some interrupts are more

important than others and so need to be done before others. What about the situation where one interrupt is currently being serviced by the CPU and another happens? Again, it depends on how important the new interrupt is compared to the one already being done. If it is more important, then the CPU will want to service it immediately.

When the CPU decides to service an interrupt, it stops processing the current job, 'pushing' the contents of its registers onto the stack. This would include, for example, the contents of the Program Counter and the accumulator. The CPU is now free to work on another piece of software but can return to what it was doing after the interrupt has been serviced because it has saved where it was. It then transfers control to the interrupt handling software for that type of interrupt using the vectored interrupt mechanism. When it has finished servicing the interrupt, the contents of the stack are 'popped' back into the appropriate registers and the CPU continues from where it left off before the interrupt happened.

2.3 Interrupts:

Hardware interrupts were introduced as a way to avoid wasting the processor's valuable time in polling loops, waiting for external events. With hardware interrupt scheme processor utilize its waiting time for performing some other useful task and this in turn increase processors working ability. They may be implemented in hardware as a distinct system with control lines, or they may be integrated into the memory subsystem.

If implemented in hardware, an interrupt controller circuit such as the IBM PC's Programmable Interrupt Controller (PIC) may be connected between the interrupting device and the processors interrupt pin to multiplex several sources of interrupt onto the one or two CPU lines typically available. If implemented as part of the memory controller, interrupts are mapped into the system's memory address space.

3. ADVANCE EXTENSIBLE BUS PROTOCOL (AXI):

AXI is the high-performance bus in the AMBA family. The architecture defines three write channels and two read channels. The write channels are address, write data, and response. The read channels are address and read data. The address channels include 32-bit address buses, AWADDR and ARADDR, but this could be extended in some implementations. The write and read data buses (WDATA and RDATA) may be defined under the specification as any 2n number, from 8-bit to 1024-bit. With the assumption that both the address and data buses are 32-bit, and that the data buses are 128-bit, the write address, write data, and write response channels

would require 56, 139, and 8 I/O, respectively. The read address and read data channels would require 56 and 137 I/O, respectively. Thus, each 128-bit AXI master has 396 I/O total. AXI masters and slaves are connected together through a central interconnect, which routes master requests and write data to the proper slave, and returning read data to the Requesting master. The interconnect also maintains ordering based on tags if, for example, a single master pipelines read requests to different slaves.

AXI uses a handshake between VALID and READY signals. VALID is driven by the source, and READY is driven by the destination. Transfer of information, either address and control or data, occurs when both VALID and READY are sampled high.

An AXI master begins a read transfer by driving an address, ARADDR, and other transfer qualifiers with ARVALID. In high frequency implementations, the interconnect latches and drives the same signals to the slave, which responds with ARREADY. The slave drives read data, RDATA, with RVALID, and the transfer is made when RVALID and RREADY are sampled active.

Again, in high frequency implementations, it is common for the interconnect to latch and drive the read data back to the requesting master. The last beat of read data is driven with RLAST.

For a write transfer, the AXI master begins by driving an address, AWADDR, and other transfer qualifiers with AWVALID. An interconnect latches and drives the same signals to the slave in high frequency implementations, and the slave responds with AWREADY. When WVALID is active, the first beat of write data is valid. WVALID may be driven with valid data even before or after the address that relates to it. The transfer is made when WVALID and WREADY are sampled active. The AXI master drives the last beat of write data with WLAST. The slave then drives a write response, BRESP, with BVALID back to the master to indicate when the write transfer is complete, along with any applicable error information.

Another family of buses that the Power.org Bus Architectures TSC decided to investigate is the ARM AMBA (Advanced Microcontroller Bus Architecture) set of buses: APB, AHB and AXI. The APB (Advanced Peripheral Bus) is considered a low-performance, peripheral level bus. The AHB (Advanced High-performance Bus) is considered (despite the bus name) a mid-performance bus and the AXI (Advanced extensible Interface) bus is considered a high-

performance bus. The AMBA family of buses was chosen for consideration because of their widespread acceptance in the industry and large amount of existing IP cores.

We note in passing that the APB bus is not always used as a standalone bus. Some AMBA based IP cores will use a higher performing bus like AHB or AXI for the primary function, but use an APB port for access to configuration registers. This gets configuration accesses off of the main bus which can be reserved for higher bandwidth operations. OPB is never used in this way because Core Connect provides an additional bus called DCR whose purpose is to offload configuration accesses. All of the APB IP cores discussed here are standalone devices so that this report will be an apples to apples comparison of the two buses.

3.1 OCP-IP Protocol: The Bus Architectures TSC also investigated the Open Core Protocol (OCP), due to its openness, its definition being driven by participation from its members, its wide range of supported bus features, and its acceptance within several large companies that design their own IP. OCP-IP spans all three levels of performance hierarchy using one architecture specification, but low, mid, and high performance IP are still tailored to suit their individual performance requirements. OCP-IP has been defined since 2001, and has gained acceptance in markets such as consumer electronics and gaming.

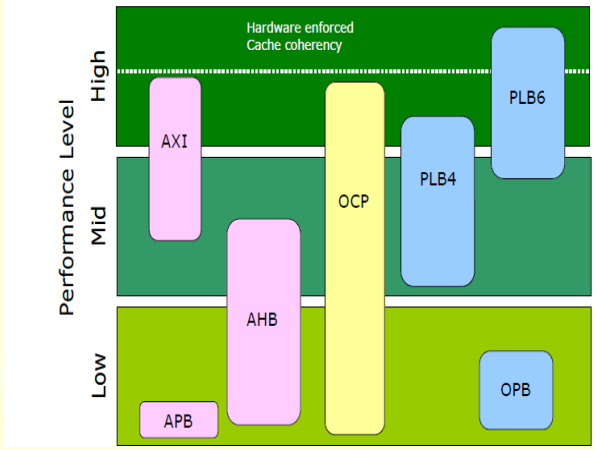


Figure: 3.1 performance evaluation

The AMBA, Core Connect, and OCP buses evaluated are shown below in Figure 1. Note that the features and functions of each bus are usually not neatly contained within one performance level. There is often enough flexibility within the architecture that allows implementations to span across performance levels. Note that the upper half of the high performance level is reserved for

buses that support hardware-enforced cache coherency, and that only one of the buses evaluated, PLB6 will support this feature.

3.2 APB Description:

The APB Bus is the lowest performance bus in the AMBA family. There are separate address (PADDR), write data (PWDATA), and read data (PRDATA) buses, up to 32-bits each. With 7 additional control signals, there can be up to 103 I/O for each APB slave.

There is one APB master, usually the bridge from a higher performance bus that begins a transfer by asserting the appropriate PSELn signal with PADDR. PWRITE is active for a write and inactive on a read. PENABLE is asserted in the second clock, and is held active until PREADY is returned by the slave. The minimum transfer, read or write is two clocks. APB slaves also have the option of inserting wait states for reads or writes by withholding PREADY. There is an optional PSLVERR signal used by the slave to report an error on a read or write with PREADY.

Assuming a frequency of 133MHz, (which should be achievable in a 90nm or smaller process) and the maximum data bus widths of 32-bits, the overall APB bandwidth could be as high as 267MB/s, but that assumes that none of the APB slaves would insert any wait states. Also, the total APB bandwidth must be shared between all of the APB slaves.

4. Architecture of Interrupt controller:

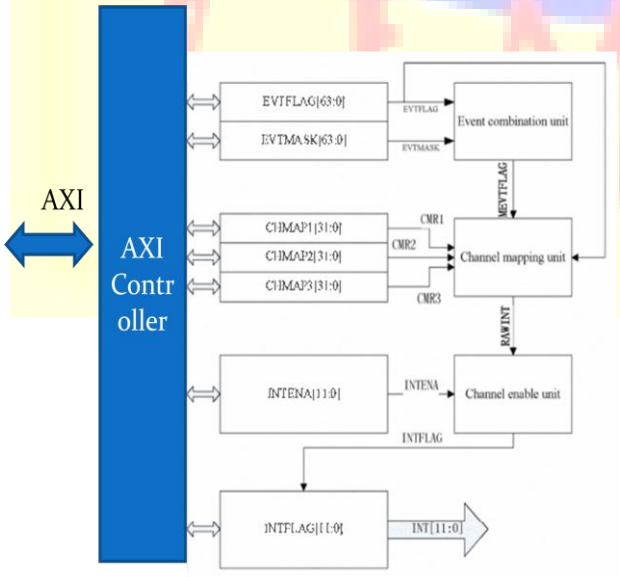


Fig 4.1 Interrupt controller

5. AXI based Embedded System:

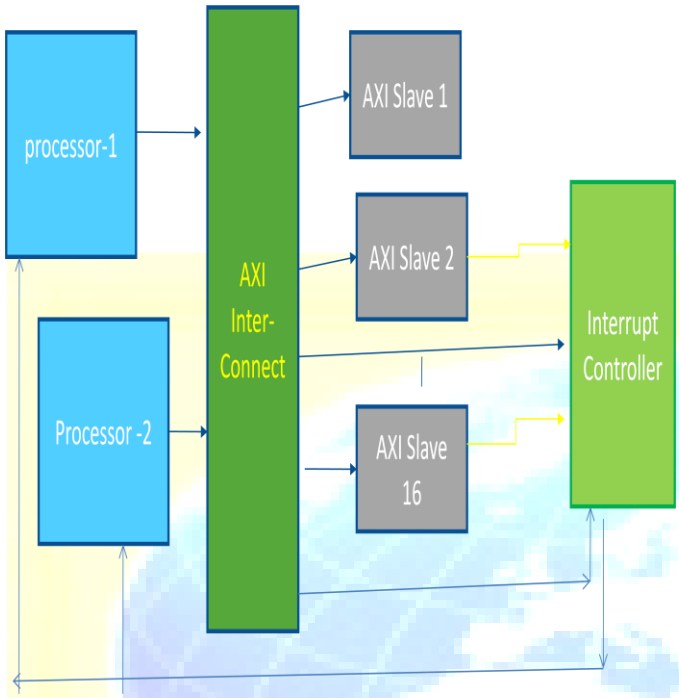


Figure 5.1 AXI based Embedded system

6. DESIGN AND ANALAYSIS:

6.1 Hierarchy of Paper:

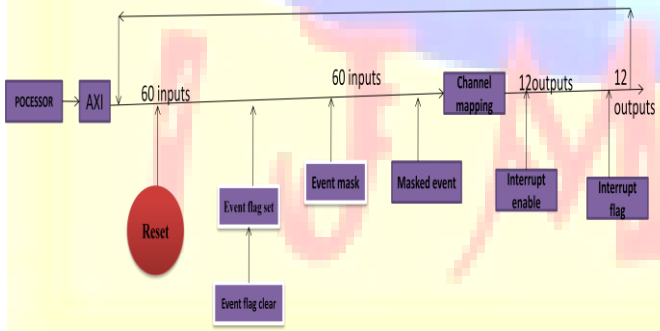


Fig6.1 Hierarchy of Paper

Initial all the interrupts are captured and they sorted according to piority and the given to the processor as per cnfiguerd by the interrupt controller register. Usaually processor and pheripherlas in the register communicate with each other through an interrupt and they connected with high performance bus protocol to incerase the performance of the interrupt controller.And this the hierachy of the project to acesses the interrupt controller and processor.

6.2 Implementation AXI Bus:

To implement the axi bus we should develop by using finite state machine.The state machine gives the operation if the axi and its working .it is linked between the processor and interrupt controller and it is high performance bus protocol and it had different signals to flow of the data .

- Initialy it is ideal state.
- Next state write valid to write the data by the processor.
- If the write valid is logic 1 than processor writing operation is done.
- Write address the processor write the address to operate the corressponding data to acesses.
- If address write valid logic 1 and write valid logic 0 when these both signals having logics than corrressponding values then writing address is done.
- If address write valid logic 1and write valid logic 1 when these both signals having logics corressponding values then done.
- After writing than it send response signal after getting response signal again it reaches to ideal state.

  Thus in this way we implement the axi by writing the verilog code to acesses between the processor and interrupt controller.

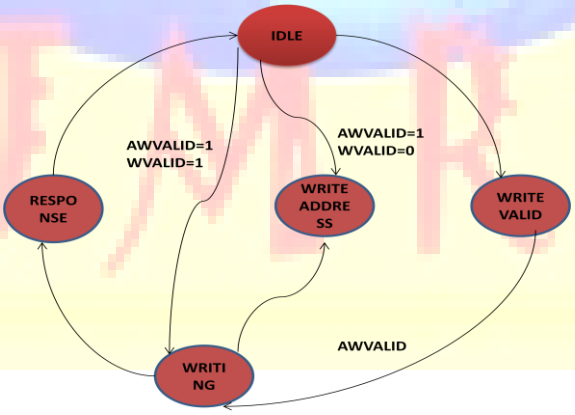
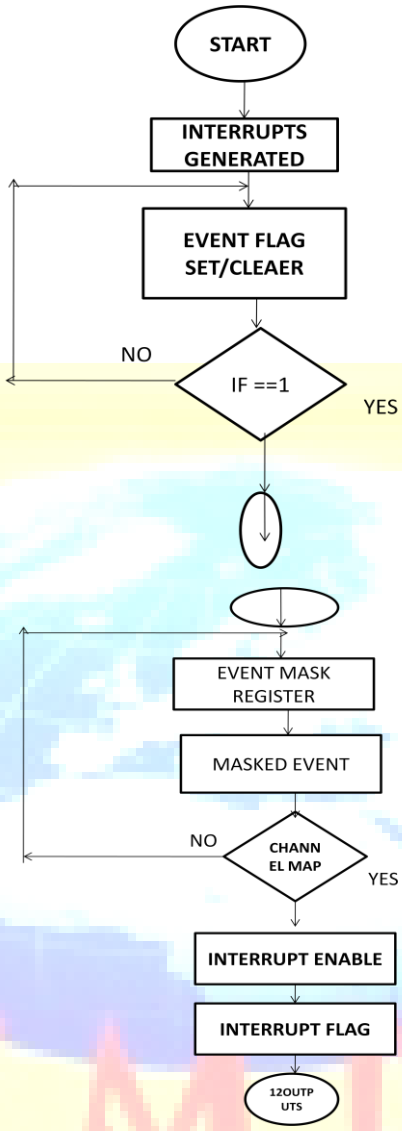


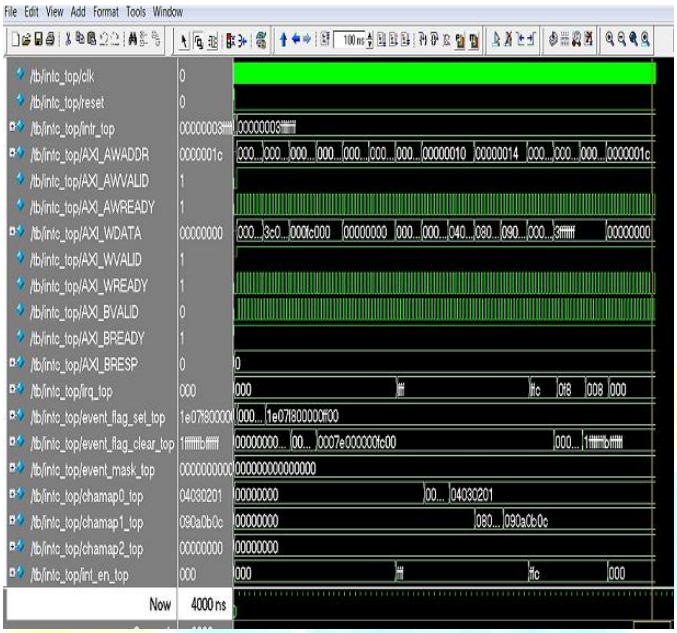
Fig6.2 Finite state machine for AXI

6.4 Flow Chart:

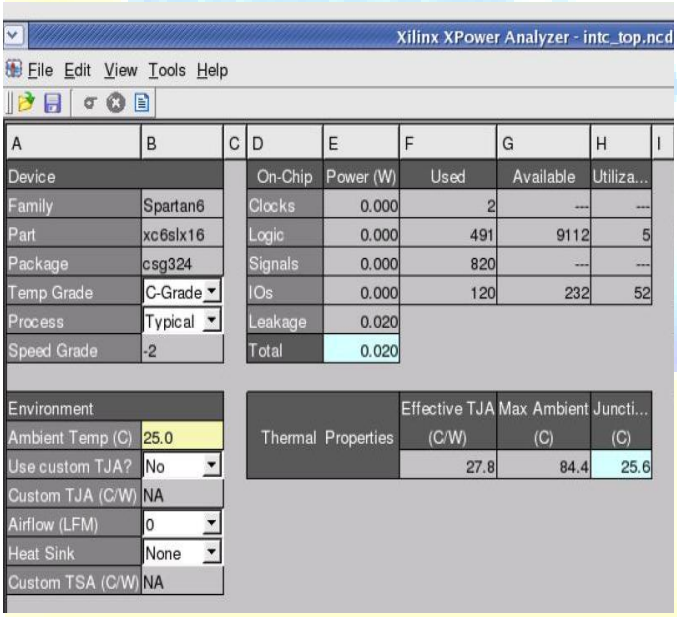


7. SIMULATION RESULTS:

7.1 Simulation Report:



7.2 Power report:



7.3 Design Summary:

INTERRUPT CONTROLLER Project Status			
Project File:	interrupt controller.ise	Current State:	Synthesized
Module Name:	intc_top	• Errors:	No Errors
Target Device:	xc3s1600e-4fg320	• Warnings:	7 Warnings
Product Version:	ISE 9.2i	• Updated:	Fri Nov 18 14:47:51 2011

INTERRUPT CONTROLLER Partition Summary
No partition information was found.

Detailed Reports					
Report Name	Status	Generated	Errors	Warnings	Infos
Synthesis Report	Current	Fri Nov 18 14:43:27 2011	0	7 Warnings	0
Translation Report					
Map Report					
Place and Route Report					
Static Timing Report					
Bitgen Report					

7.4 Synthesis Report:

Release 9.2i - xst J.36

Copyright (c) 1995-2007 Xilinx, Inc. All rights reserved.

--> Parameter TMPDIR set to ./xst/projnav.tmp

CPU : 0.00 / 0.16 s | Elapsed : 0.00 / 1.00 s

--> Parameter xsthdpdir set to ./xst

CPU : 0.00 / 0.16 s | Elapsed : 0.00 / 1.00 s

--> Reading design: intc_top.prj

TABLE OF CONTENTS

1) Synthesis Options Summary

2) HDL Compilation

3) Design Hierarchy Analysis

4) HDL Analysis

5) HDL Synthesis

5.1) HDL Synthesis Report

6) Advanced HDL Synthesis

6.1) Advanced HDL Synthesis Report

7) Low Level Synthesis

8) Partition Report

9) Final Report

9.1) Device utilization summary

9.2) Partition Resource Summary

9.3) TIMING REPORT

********* Synthesis Options Summary**********

---- Source Parameters

Input File Name : "intc_top.prj"

Input Format : mixed

Ignore Synthesis Constraint File : NO

---- Target Parameters

Output File Name : "intc_top"

Output Format : NGC

Target Device : xc3s1600e-4-fg320

---- Source Options

Top Module Name : intc_top

Automatic FSM Extraction : YES

FSM Encoding Algorithm : Auto

Safe Implementation : No

FSM Style : lut

RAM Extraction : Yes

RAM Style : Auto

ROM Extraction : Yes

Mux Style : Auto

Decoder Extraction : YES

Priority Encoder Extraction : YES

Shift Register Extraction : YES

Logical Shifter Extraction : YES

XOR Collapsing : YES

ROM Style : Auto

Mux Extraction : YES

Resource Sharing	: YES
Asynchronous To Synchronous	: NO
Multiplier Style	: auto
Automatic Register Balancing	: No
---- Target Options	
Add IO Buffers	: YES
Global Maximum Fanout	: 500
Add Generic Clock Buffer(BUFG)	: 24
Register Duplication	: YES
Slice Packing	: YES
Optimize Instantiated Primitives	: NO
Use Clock Enable	: Yes
Use Synchronous Set	: Yes
Use Synchronous Reset	: Yes
Pack IO Registers into IOBs	: auto
Equivalent register Removal	: YES
---- General Options	
Optimization Goal	: Speed
Optimization Effort	: 1
Library Search Order	: intc_top.lso
Keep Hierarchy	: NO
RTL Output	: Yes
Global Optimization	: AllClockNets
Read Cores	: YES
Write Timing Constraints	: NO
Cross Clock Analysis	: NO
Hierarchy Separator	: /
Bus Delimiter	: <>
Case Specifier	: maintain
Slice Utilization Ratio	: 100
BRAM Utilization Ratio	: 100

Verilog 2001 : YES

Auto BRAM Packing : NO

Slice Utilization Ratio Delta : 5

*********HDL Compilation **********

WARNING:HDLCompilers:176 - Include directory \New folder (2)\new modified code\ does not exist

Compiling verilog file "F:/New folder (2)/new modified code/axi_top.v" in library work

Compiling verilog file "F:/New folder (2)/new modified code/intc_top.v" in library work

Module <axi_top> compiled

Module <intc_top> compiled

==========

* Design Hierarchy Analysis *

==========

ERROR:HDLCompilers:87 - "F:/New folder (2)/new modified code/intc_top.v" line 60 Could not find module/primitive 'final_top'

-->

Total memory usage is 152100 kilobytes

Number of errors : 1 (0 filtered)

Number of warnings : 7 (0 filtered)

Number of infos : 0 (0 filtered)

8. Conclusion:

In this paper, an interrupt-combinable and priority configurable multichannel interrupt controller is proposed. AXI bus interface is used for communication with processor in this interrupt controller. The interrupt controller supports 60 interrupt inputs and has 12 interrupt channels for interrupt requests. Input interrupts are split into 4 groups; a combination interrupt is generated with each group. In this case, the interrupt controller has 64 interrupt sources. Priority of interrupts is configurable. Any of the 64 interrupt sources can be mapped to any interrupt channel, which makes this interrupt controller more flexible. The synthesis time delay in SMIC O.13um CMOS technology is 1.43ns and the total power is 8.44mW at the switching rate of 100%.

9. Future Scope:

- Can connect more than 60 interrupts
- Can use a different processor Interface other than AXI (next generation bus protocol)
- We can reduce the power consumption.
- We can implement in any embedded systems for real time applications.
- Configurable and combination can be done for large number of interrupt inputs
- We can also increase the number of peripherals on single system on chip by using interrupt controller as multichannel.

10. BIBLIOGRAPHY:

[1] W.A. Halang, M. Wannemacher, C.E. Pereira, "A high-precision timing and interrupt controller to support distributed real-time operating systems ," Circuits and Systems, 1995, pp 9-12.

[2] A. de Gloria, P. Faraboschi, M. Olivieri, "A self timed interrupt controller: a case study in asynchronous micro-architecture design," ASIC Conference and Exhibit, 1994, pp 296-299.

[3] A. Tumeo, M. Branca, L. Camerini, M. Monchiero, G. Palermo, F. Ferrandi, D. Sciuto, "An Interrupt Controller for FPGA-based Multiprocessors," Embedded Computer Systems: Architectures, Modeling and Simulation, 2007, pp 82-87.

[4] D. Hristu-Varsakelis, P.R. Kumar, "Interrupt-based feedback control over a shared communication medium," Decision and Control, 2002, pp 3223- 3228.

[5] C. Panis, J. Hohl, H. Gruenbacher, J. Nurmi, "xICU - in interrupt control unit for a configurable DSP core ," System-on-Chip, 2003, pp 75-78.

[6] Qiurong Wang, "An Interrupt Management Scheme Based on Application in Embedded System ," MultiMedia and Information Technology, 2008, pp 449-452.