Verification Of Interrupts in a SOC's Core Based Subsystem

Ramki Ganji MS in VLSI Engineering, Veda IIT, Guntur, ramkiganji15@gmail.com
Murali Mohan Kilari, SoCtronics Guntur, muralimohan.kilari@soctronics.com
Sivaji Satya Ganesh Nandigama, Dept of VLSI, Veda IIT, Guntur,
sivajisatyaganesh.nandigama@vedaiit.com

Abstract : Interrupt Handling is one of the key design aspects to be handled at SOC level & Interrupt verification for SOC level scope is very important covering the various interrupt sources originating in the SOC. There are several interrupts originated from different set of IPs and Interrupt controllers at different stages are to be placed to decentralize and optimize the overall interrupt data path handling. Various interrupt sources are finally routed to the CPU for servicing and the APIC local to the core or residing in the Northbridge is typically the final interrupt controller destination in the SOC. X2 APIC is a recent technology advancement in Interrupt controllers. The major improvements of the x2APIC address the number of supported CPUs and performance of the interface. This project covers the interrupt verification aspects in a Core based subsystem that involves Core, Northbridge and Memory Controller. From one generation to another generation SOC feature set increases & process technology scales down. This paper is to bring up with the different optimized verification scenarios involved with the x2 APIC interrupt controller which is primarily used to handle all the interrupts from the different peripherals and prioritize them at SOC level. Scope covers different interrupts scenarios covering inter-processor interrupts & IO side interrupts that are routed to CPU.

Keywords : *x2apic; interrupt; microprocessor; multithreads; Core Based Subsystem*

1. INTRODUCTION

In early years of computing, processor has to wait for the signal for processing. So processor has to check each and every hardware and software program in the system if it has any signal to process. This method of checking the signal in the system for processing is called Polling Method. In this method, the problem is that the processor has to waste number of clock cycles just for checking the signal in the system, by this processor will become busy unnecessarily. If any signal came for the process, processor will take some time to process the signal due to the polling process in action. So system performance also will be degraded and response time of the system will also decrease.

So to overcome this problem, engineers introduced a new mechanism. In this mechanism processor will not check for any signal from hardware or software but instead hardware/software will only send the signal to the processor for processing. The signal from hardware or software should have highest priority because processor should leave the current process and process the signal of hardware or software. This mechanism of processing the signal is called Interrupt of the system.

What is an Interrupt?

Interrupt is a signal which has highest priority from hardware or software which processor should process its signal immediately.

Types of Interrupts:

Although interrupts have highest priority than other signals, there are many type of interrupts but basic type of interrupts are

1 Hardware Interrupts:
If the signal for the processor is from external device or hardware is called hardware interrupts. Example: from keyboard we will press the key to do some action this pressing of key in keyboard will generate a signal which is given to the processor to do action, such interrupts are called hardware interrupts. Hardware interrupts can be classified into two types they are
1.1 Maskable Interrupt: The hardware interrupts which can be delayed when a much highest priority interrupt has occurred to the processor.
1.2 Non Maskable Interrupt: The hardware which cannot be delayed and should process by the processor immediately.
2.Software Interrupts: Software interrupt can also divided in to two types. They are

2.1 Normal Interrupts: the interrupts which are caused by the software instructions are called software instructions.
2.2 Exception: unplanned interrupts while executing a program is called Exception. For example: while executing a program if we got a value which should be divided by zero is called a exception.

2. 8259 Programmable Interrupt Controller

- PIC 8259 is a Programmable Interrupt Controller that can work with 8085, 8086 etc.
- It is used to increase the number of interrupts.
- A single 8259 provides 8 interrupts while a cascaded configuration of 1 master 8259 and 8 slave 8259s can provide up to64 interrupts.
- 8259 can handle edge as well as level triggered interrupts.
- 8259 has a flexible priority structure.
- In 8259 interrupts can be masked individually.7)The Vector address of the interrupts is programmable.
- 8259 has to be compulsorily initialized by giving commands, to decide several properties such as Vector Numbers, Priority, Masking, Triggering etc.
- In a cascaded configuration, each 8259 has to be individually initialized, master as well as each slave.

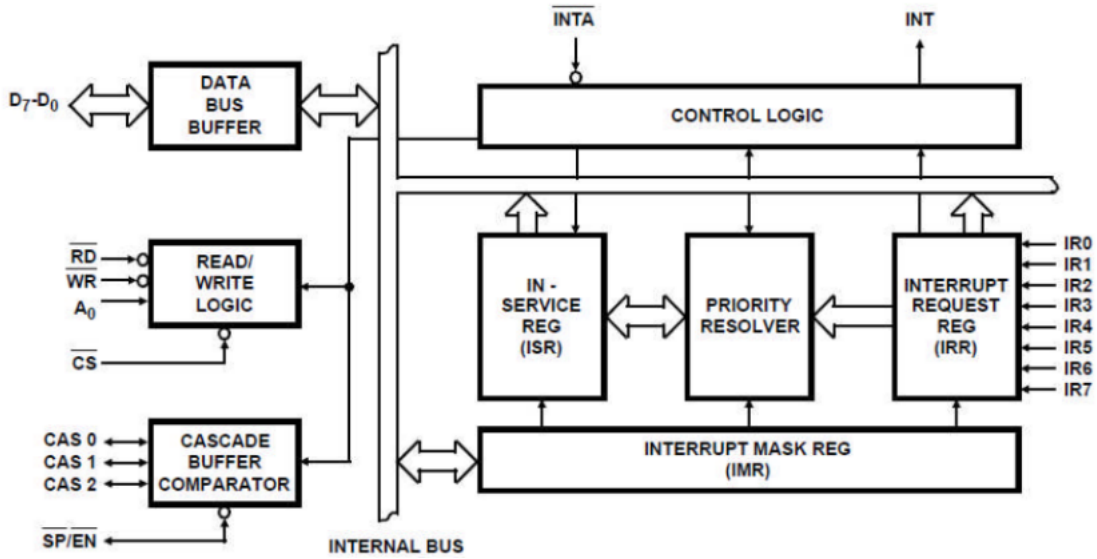


Fig 1.1 Architecture of 8259

3. X2APIC

3.1 X2APIC Enhancements

The key enhancements provided by the x2APIC architecture over xAPIC are the following:

- Support for two modes of operation to provide backward compatibility and extensibility for future platform innovations.
- Increased range of processor addressability in x2APIC mode:
- More efficient MSR interface to access APIC registers.
- The semantics for accessing APIC registers have been revised to simplify the programming of frequently-used APIC registers by system software. Specifically
- the software semantics for using the Interrupt Command Register (ICR)
- And End Of Interrupt (EOI) registers have been modified to allow for more efficient delivery and dispatching of interrupts.

3.2 Interrupt Command Register

In x2APIC mode, The lower 32 bits of ICR in x2APIC mode is identical to the lower half of the ICR in xAPIC mode, except bit 12 (Delivery Status) is not used since it is not needed in X2APIC mode. The destination ID field is expanded to 32 bits in x2APIC mode.

3.3 SELF IPI register

SELF IPIs are used extensively by some system software. The xAPIC architecture provided a mechanism for sending an IPI to the current local APIC short-hand in the interrupt command register .using the "self-IPI"s The x2APIC architecture introduces a new register interface. This new resgister is dedicated to the purpose of sending self-IPIs with the intent of enabling a highly optimized path for sending self IPIs.

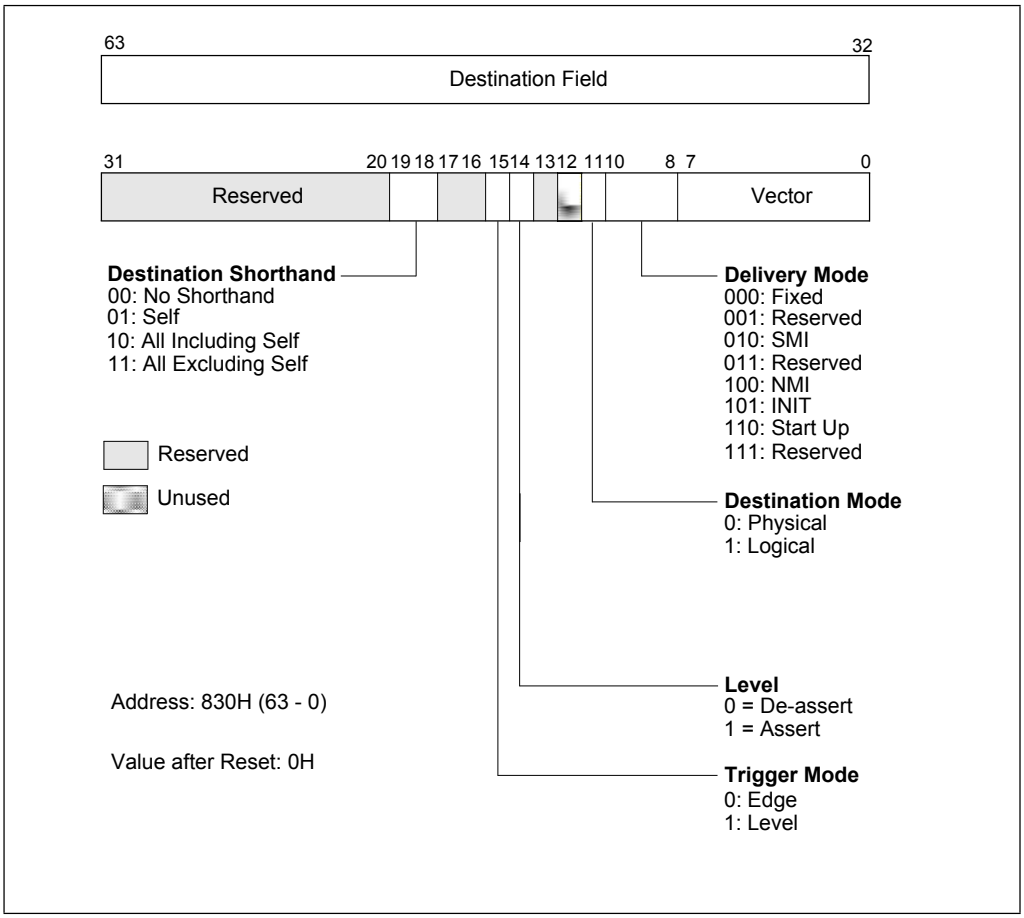


Fig3.1 Interrupt Command Register

3.4 Local APIC ID Register

In x2APIC mode, the local APIC ID register is a read-only register to system software and will be initialized by hardware. It is accessed via the RDMSR instruction reading the MSR at address 0802H. Fig. provides the layout of the Local x2APIC ID register.

3.5 Logical Destination Register

In x2APIC mode, the Logical Destination Register (LDR) is increased to 32 bits wide. It is a read-only register to system software. This 32-bit value is referred to as "logical x2APIC ID". System software accesses this register via the RDMSR instruction reading the MSR at address 80DH. Figure provides the layout of the Logical Destination Register in x2APIC mode

3.6 Interrupt Handling in a SOC

At the system level, APIC consists of two parts (Figure 2.0)—one residing in the I/O subsystem (called the **IOAPIC**) and the other in the CPU (called the **Local APIC**). The local APIC and the IOAPIC communicate over a dedicated APIC bus. The IOAPIC bus interface consists of two bi-directional data signals (APICD[1:0]) and a clock input (APICCLK).

The CPU's Local APIC Unit contains the necessary intelligence to determine whether or not its processor should accept interrupts broadcast on the APIC bus. The Local Unit also provides local pending of interrupts, nesting and masking of interrupts, and handles all interactions with its local processor

(e.g., the INTR/INTA/EOI protocol). The Local Unit further provides inter-processor interrupts and a timer, to its local processor. The register level interface of a processor to its local APIC is identical for every processor.

The CPU's Local APIC Unit contains the necessary intelligence to determine whether or not its processor should accept interrupts broadcast on the APIC bus. The Local Unit also provides local pending of interrupts, nesting and masking of interrupts, and handles all interactions with its local processor (e.g., the INTR/INTA/EOI protocol). The Local Unit further provides inter-processor interrupts and a timer, to its local processor. The register level interface of a processor to its local APIC is identical for every processor.

The IOAPIC Unit consists of a set of interrupt input signals, a 24-entry by 64-bit Interrupt Redirection Table, programmable registers, and a message unit for sending and receiving APIC messages over the APIC bus. I/O devices inject interrupts into the system by asserting one of the interrupt lines to the IOAPIC. The IOAPIC selects the corresponding entry in the Redirection Table and uses the information in that entry to format an interrupt request message. Each entry in the Redirection Table can be individually programmed to indicate edge/level sensitive interrupt signals, the interrupt vector and priority, the destination processor, and how the processor is selected (statically or dynamically). The information in the table is used to transmit a message to other APIC units (via the APIC bus).

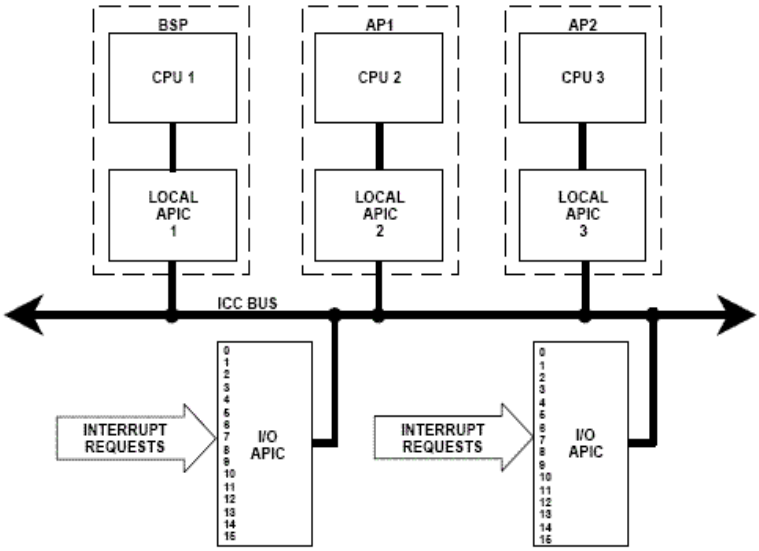


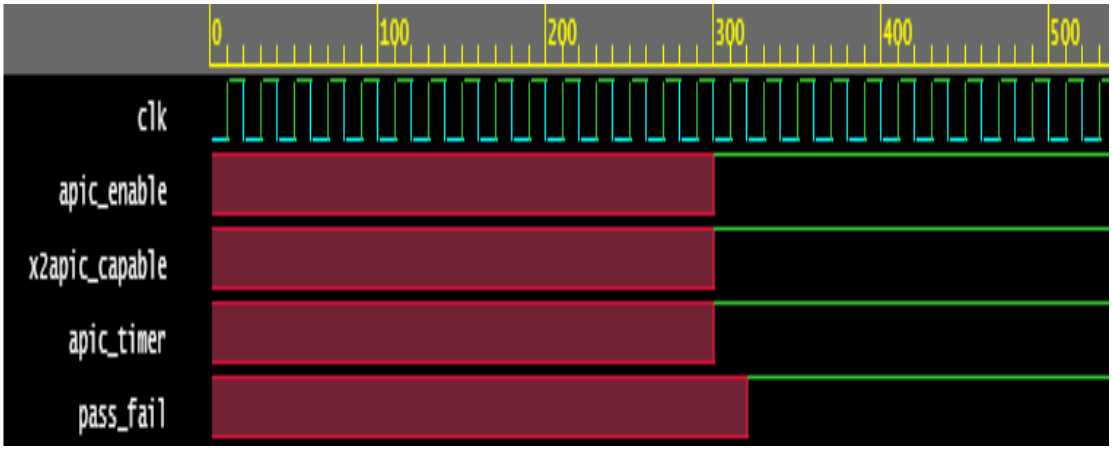
Fig 3.5 Block Diagram of a Typical APIC Implementation in a SOC

4.1 Basic X2APIC Check

This is a basic sanity check to determine whether this core supports the x2apic feature .This is done by executing the CPUID Functions

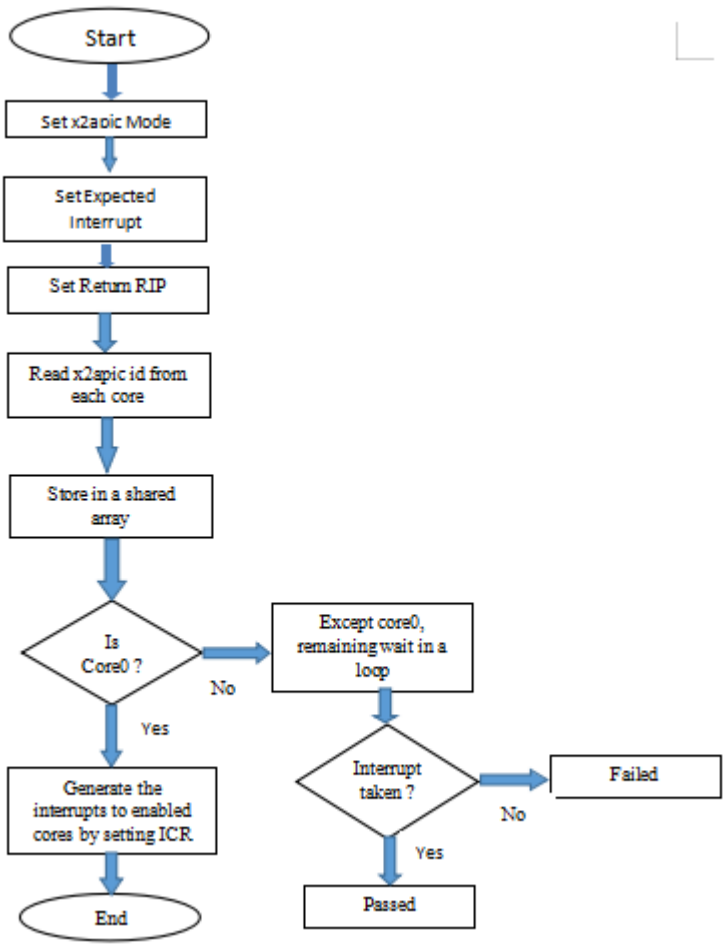
- Function 00000001_edx[9] represent apic is enable
- Function 00000006_eax[2] will represent apic timer enable.
- Function 00000001_ecx[21] represent x2apic capability
- Set x2apic mode by setting the bit 10 and 11 of apic_base_msr.

Results



4.2 Inter Processor Interrupts

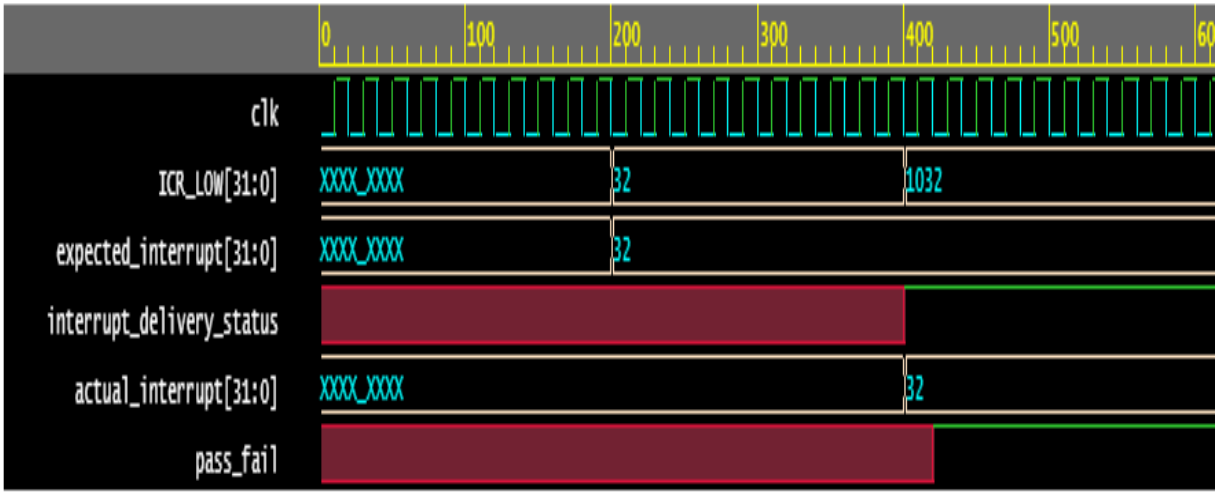
Flow Chart



Intention of this scenario is to check Inter processor interrupt from one core to another core. In this scenario we are going to send the interrupt to all the enabled cores from core0. and verifying that the interrupt received by other cores or not. If the interrupt is received by other cores we are going to make test passed else failed.

We will set the x2apic mode and set the expected interrupt vector to be taken and set the return rip to come back from the interrupt.Then, read all x2apic id's from each core and store it in a shared array(access among all cores). except core0, all remaining cores should go to wait loop for interrupt. Core0 will generate the interrupts to all enabled cores by setting the ICR(Interrupt Command Register). Then we will check all the remaining cores got the interrupt or not by checking the interrupt delivery status bit. Else failed.

Results



4.3 Spurious Interrupt Check

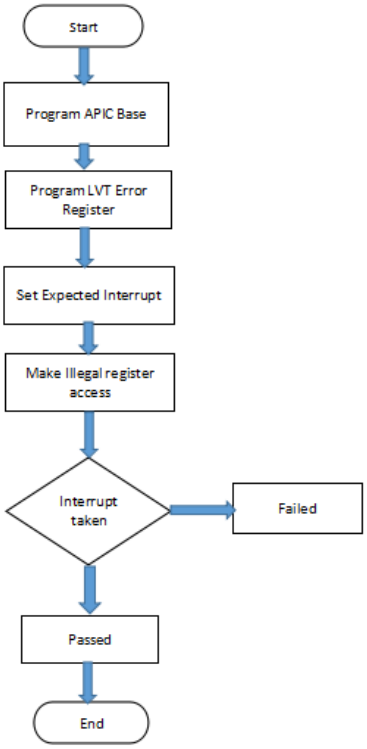
Intention of this scenario is to check Spurious Interrupt when detect the error while handling the interrupts.

The Local APIC provides an error status register(ESR) that uses to record the errors that it detects when handling interrupts. An APIC error interrupt is generated when the local apic sets one of the error bits in the ESR. Error Status Register(ESR) bit 7 represents the illegal register access The LVT error register allows the selection of interrupt to be delivered to the processor core when apic error is detected.

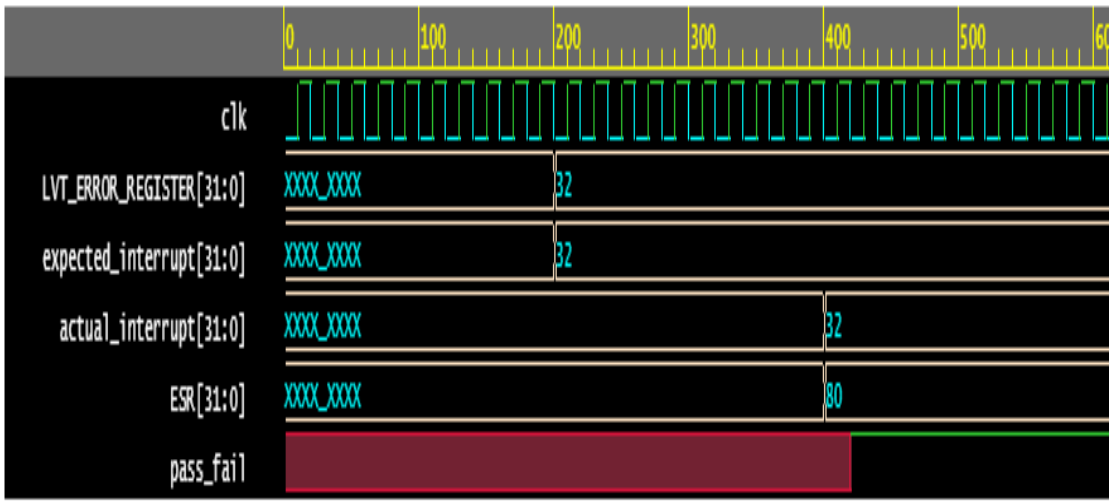
- Program the apic base

- Program the LVT error register
- Make an illegal register access
- Make sure APIC interrupt to be taken else failed.

Flow Chart



Results



Conclusion

Interrupt Handling is one of the key design aspects to be handled at SOC level & Interrupt verification for SOC level scope is very important covering the various interrupt sources originating in the SOC. This paper is to bring up the interrupt verification scenarios in a system on chip. These scenario's will cover the basic x2apic capability check and interrupts from one core to another core and spurious interrupts.

References

[1] An Interrupt Controller for the RAW by Anant K. Saraswat MASSACHUSETTS INSTITUTE OF TECHNOLOGY February 2003
[2] ARM Generic Interrupt Controller Architecture Specification
[3] Intel 64 Architecture x2APIC Specification
[4] "Improving Interrupt Handling in the nMPRA" 12th International Conference on DEVELOPMENT AND APPLICATION SYSTEMS, Suceava, Romania, May 15-17, 2014