FPGA DESIGN OF AN EFFICIENT AND LOW-COST SMART PHONE INTERRUPT CONTROLLER

M. DE ALBA†, A. ANDRADE†, J. GONZÁLEZ‡, J. GÓMEZ-TAGLE‡ and A. D. GARCÍA†

† *Depto. de Ingeniería Eléctrica y Electrónica, Tecnológico de Monterrey, Campus Estado de México, Atizapán de Zaragoza, México, 52926 México*

{a00464139, marcos.de.alba, garcia.andres}@itesm.mx

‡ *Centro de Diseño Electrónico, Tecnológico de Monterrey, Campus Guadalajara, Zapopan, Jalisco, 45201 México*

{jair, javier.gomeztagle}@itesm.mx

***Abstract*— In this work we have designed and implemented an efficient platform-level interrupt controller for a PXA270 microprocessor-based smart phone. Although current hardware development boards include this type of controllers, for specific applications most of them are costly and include too many interrupt sources that represent a waste for a particular design. For this reason we designed our own interrupt controller which is capable of detecting interrupt sources coming from different devices that request microprocessor service. The developed interrupt controller is efficient and low-cost due to the small number of register and logic gates required for its implementation, as well as for the small number of levels to be traversed in the circuit´s critical execution path.**

***Keywords*— Interrupt controller, Codesign, FPGA, low-cost, effective, PXA270, smart phone.**

I. INTRODUCTION

Specific-purpose microprocessors are involved in the design of mobile digital systems, such as PDAs or Smart Phones. A microprocessor provides flexibility and scalability to the system. The flexibility is due to different programs being implemented with the same hardware. The scalability is due to the integration of additional hardware components by just modifying routines in the application program.

When the system performs several applications, the hardware and software become more complex. On the hardware side, additional devices are needed.

On the software side a manager program, like a real-time operating system is required, otherwise conflicts among different devices may arise. One effective mechanism to manage hardware devices is an interrupt controller.

An interrupt controller is a hardware component capable of detecting request signals coming from different devices. The request signals are generated by devices to indicate that they need some service to continue working. For example, pressing a keyboard key attached to a personal computer sends the equivalent digital code to the CPU. In there, an interrupt controller detects that the keyboard device needs attention, therefore an interrupt service routine is assisted in order to activate the input port associated to the keyboard and read it. The data that was read from the port corresponds to the value of the pressed key. Afterwards, depending on the application being used, for example a word processor, the available data might be displayed in the screen at the current cursor position.

In a more complex microprocessor-based architecture, which contains a high number of peripheral devices, it is desirable to have an interrupt controller to identify which device makes requests at which time. These devices must allow the microprocessor to define the priority level of each interrupt service.

In this work we developed a FPGA platform-level hardware software codesign of an efficient and low-cost interrupt controller for a smart phone to manage requests originated by several devices. The developed interrupt controller involved a hardware software codesign process because the smart phone is being designed for a particular operating system (Windows, 2005) and the interrupt controller features have taken into account the target operating system characteristics, to enhance the development and maintenance of the smart phone. The interrupt controller has been designed for a particular smart phone that is being designed as a cooperative effort between the authors´ institutions with the support of Instituto Tecnológico de Monterrey, Intel Corporation and the Secretaría de Economía de México, among others. The smart phone is to become a commercial product within this year. We have developed this interrupt controller because the available hardware development board of the used microprocessor needs and external interrupt controller in order to service I/O devices hierarchically.

In our design the core microprocessor is an Intel PXA270 (Intel, 2004a; Intel, 2004b; Intel, 2004c; Intel, 2004d; Intel, 2004e; Intel, 2004f; Intel, 2004g). Although the PXA270 has a hardware-level interrupt controller, a platform-level is needed. This type of controller allows handling requests from a device source to the appropriate target device. It also allows having several requests coming from the same device.

The rest of the paper is organized as follows. The Related Work section describes some previous interrupt controller designs. The Implementation of the Platform-level Interrupt Controller section explains our design in detail. The section Evaluation of the Interrupt Controller

describes the methodology that we used to prove the functionality and validation of our design. In the Results section we show views of our controller working. In Future Work and Conclusions is explained what direction we are heading with our design and conclude what are the advantages and disadvantages of the proposed controller.

II. RELATED WORK

Interrupt controllers are used in personal computers to manipulate interrupt requests generated by peripheral devices, one of the most popular is Intel´s 8259 (Intel, 1995). However, since mobile phones contain a smaller number of devices than a personal computer, this type of interrupt controller (IC) is too costly to be used in a smart phone. Although there are programmable interrupt controllers, like Intel´s 8259, to maintain a simple smart phone design we liked to avoid sending commands to the IC. For this reason a non-programmable IC was proposed.

An interrupt controller for a network card is defined in (Nakashima, 2002). It is particularly designed to handle interrupt requests when a set of network packages arrives to the controller.

A self-timed asynchronous IC is described in (De Gloria, 1994). It was specifically designed for the ST9026 micro-architecture. Since these standard solutions can not easily be adapted to our smart-phone, we preferred to design a custom IC.,

The interrupt controller described in this work does not need to be programmed in order to detect interrupts from different sources. This feature makes it simple and fast. In the next section we describe the proposed interrupt controller.

III. IMPLEMENTATION OF A PLATFORM-LLEVEL INTERRUPT CONTROLLER

Figure 1 shows a block diagram of the proposed platfom-level interrupt controller. The design consists of a set of hardware queues to store pending interrupts. When two or more devices make an interrupt request simultaneously, the controller identifies which device has the highest priority. The controller immediately generates an interrupt request to the microprocessor and it indicates the device identification number, ID. The interrupt requests of the devices with smaller priorities are stored in the corresponding queues. Interrupt requests are then serviced from the queues obeying to the priority of each device. Devices are assigned priorities according to the pin where they are connected.

When an interrupt request is sent to the microprocessor it is also sent the ID of the device that generated the interrupt. A *queue selector* always chooses the queue with highest priority.

Another component of the controller is the *interrupts buffer*. This block detects interrupt signals with different frequencies –since different devices might generate requests at different rates-.

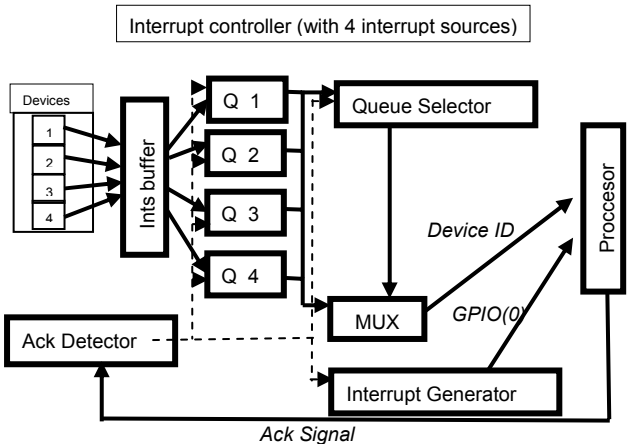

Fig. 1. Block diagram of platform-level interrupt controller

At the time the microprocessor detects an interrupt it sends an acknowledge signal to the controller. The recent serviced interrupt is removed from its queue. Pending interrupts are removed from the queues starting with the highest priority queue and ending with the lowest priority queue. If new requests arrive while interrupts are being serviced, these will be stored in the appropriate queue.

The design of the interrupt controller was developed using VHDL, simulated and verified using Xilinx´s ISE 6.2i and implemented in a Xilinx Spartan II FPGA.

The interrupt controller was implemented using a structural organization. It contains the following main entities:

- `InterruptController`
- `NewInterruptDetection`
- `DeviceIDDeco`
- `AcknowledgeDetection`
- `PendingInterrupts`
- `InterruptGenerator`

The module `InterruptController`, shown in Fig. 2, contains inputs and outputs to the interrupt controller. It has the following input signals: *CLK* clock signal, *n* interrupt sources (one for each device), ACK acknowledge, which indicates that the last interrupt has been serviced, and *RESET*. It has the following output signals: *GPIO[0]*, which connects to the microprocessor *INT* input, and *deviceID* that provides the code of each device. For testing and verification purposes, 4 and 16 interrupt sources where generated. However, the design is scalable to *n* sources. The microprocessor servicing the interrupts has to generate the ACK signal, which is twice as long as the FPGA´s clock signal in order to be detected.

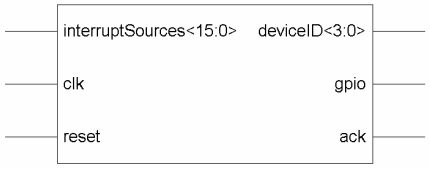

Fig. 2. Interrupt Controller module block diagram

The `NewInterruptDetection` module detects interrupts coming from different sources with different frequencies. It has a register that holds the state of pending interrupts. Interrupts are detected at rising edges. New interrupts are generated only if new rising edges are detected.

The `DeviceIDDeco module` is a digital decoder that generates the identification code of the device that is being serviced by the microprocessor.

`The AcknowledgeDetection module` is a register that holds the value of the last acknowledged interrupt, which is used to control pending interrupts.

`The PendingInterrupts module` is a multiple-entry register. It holds pending interrupt requests for each of the interrupt sources. It updates its state once the ACK signal has been received.

The `InterruptGenerator` component produces the interrupt request signal that is sent to the microprocessor. It also sends a vector ID to the `DeviceIDDeco` module. The period of this signal is related to the period of the FPGA´s clock multiplied by a general purpose input/output duration constant.

The interrupt controller handles interrupts according to priorities. Every device has a different priority depending on the attached pin. The lowest significant bit is assigned to the device with highest priority, and the most significant bit is assigned to the device with lowest priority.

IV. EVALUATION METHODOLOGY

To evaluate the correctness of the proposed interrupt controller, a PXA270 emulator was built based on a personal computer, a microcontroller AT90S8535, and a Xilinx Spartan 3 FPGA with ISE 6.2i and programming interfaces in java.

Figure 3 shows a window of the ISE tool, on the right side of the window part of the interrupt controller VHDL specification is listed.

The developed interrupt controller has been tested and verified using a personal computer. A program that generates interrupt signals (organized as interrupt vectors) is used as a benchmark to validate the correct response of the interrupt controller.

The program generates random interrupt vector requests. These vectors where sent to the FPGA through the serial port. The interrupt controller was implemented in a Xilinx Spartan II FPGA. It detects the requests and processes them according to their priorities. Interrupts are extracted from the queues one by one from highest to lowest priority. The interrupt controller selects an interrupt and sends a request signal to a microcontroller (an Atmel AT90S8535, running at 16MHz). The INT pin of the microcontroller detects the interrupt. The microcontroller emulates the handling of the interrupt request and sends an ACK signal to the interrupt controller. It extracts also the next interrupt request to be processed. For simplicity, we used only one INT line on the microcontroller. However, more than one external interrupt line might be used to emulate the functionality of the interrupt controller.

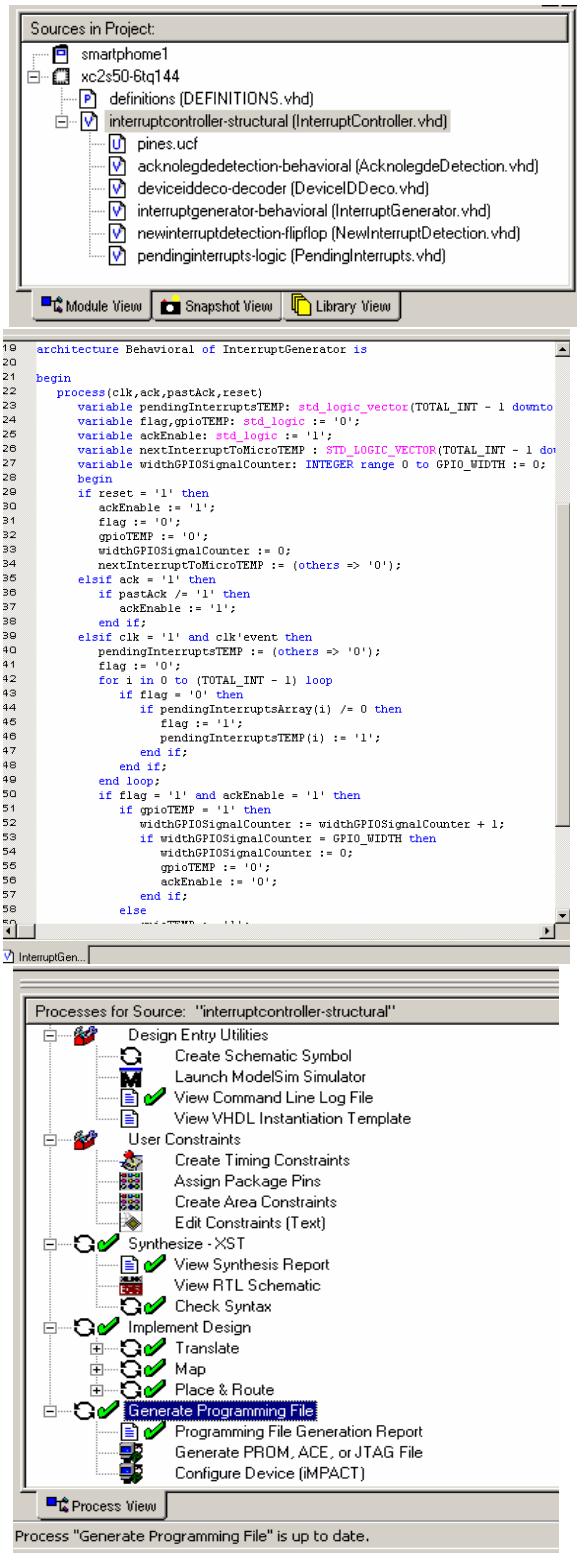

Fig. 3. VHDL specification of the interrupt controller using ISE 6.2i

In order to implement an evaluation, the program running in the personal computer is set to a number of interrupt requests. For example, setting this number to 65,536 (for 16 different devices), the program is started

and interrupt vectors ranging from 0 to 65,535 are generated and sent one by one through the serial port. The FPGA reads the vectors and fills the queues to avoid stalling the processing of requests. Since the requests arrive more rapidly than the processing speed of the microprocessor, the queues get filled in. However, they are emptied as interrupts are serviced.

On the AT90S8535 microcontroller side the 8-bit port B was utilized for proving the correctness of the interrupt controller. Figure 4 shows the bit assignment for port B.

ID[1]	ID[0]	ACK	RESET	IS[3]	IS[2]	IS[1]	IS[0]

Fig. 4. AT90S8535 Port B bit assignment

The bits B0 to B3 are utilized for interrupt sources IS0 to IS3.
Bit B4 is assigned to handle the RESET signal.
Bit B5 is assigned to handle the ACK signal.
Bits B6 and B7 are used for handling the device ID.

Notice that in this implementation, four devices generate interrupts. However, during evaluation we implemented the controller to handle up to 16 different devices and generated up to 64K interrupt requests. As long as interrupt requests were generated, they were handled by the interrupt controller in the FPGA. Interrupt requests were processed according to their priorities. Figure 5 shows an emulation of the interrupt controller. The window on the top to the left shows a set of interrupt request vectors, the window on the top to the right shows the number of pending interrupt requests, as can be seen, the requests at 00 have the highest priority, that is why they have the smallest number of pending interrupts. The window on the bottom shows an exchange of data between the interrupting devices and the microprocessor.

A *Data Sent* message indicates that a data has been sent by the device. The *Data Received* message indicates that the microprocessor has just received the data. Then, an *ACK message* is sent to the personal computer to indicate that the interrupt was handled by the interrupt controller and passed over to the microprocessor hierarchically. All the *Data Received* messages have received a "1", this is because the highest interrupt level which is being serviced has sent a "1". Once all interrupt requests have been serviced, the window of pending interrupt vectors is cleared.

We explored the design space of the interrupt controller. In the implemented emulated system we included 16 different interrupt sources with a 256-entry queue. For such implementation we obtained the FPGA area usage listed in Table 1. In average, a 12.6 % of the FPGA area is used with such design.

We also explored the interrupt controller design space for larger implementations, being the largest one with 32 interrupt sources and a 32-entry queue, for

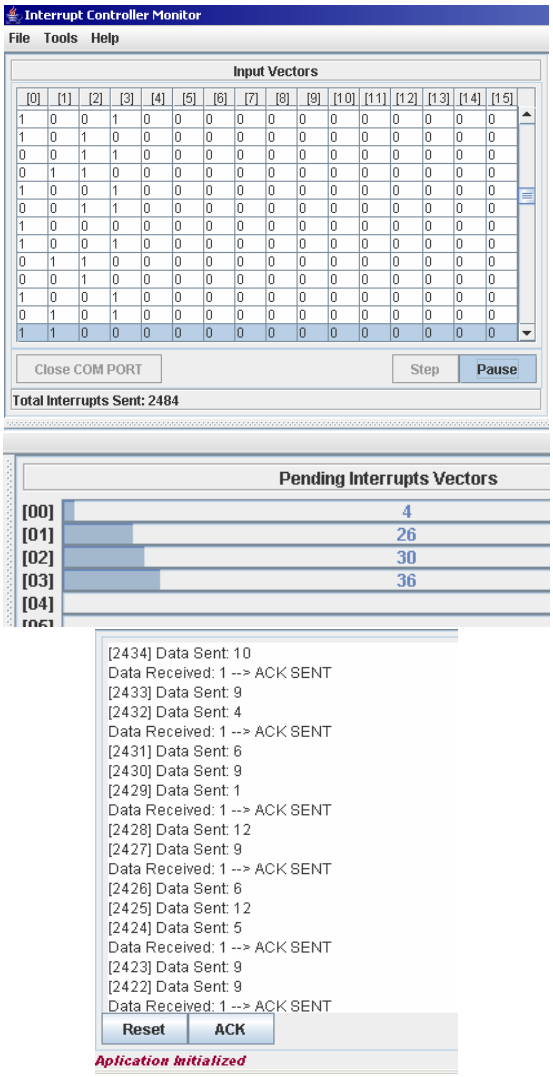

Fig. 5. An emulation of the interrupt controller (interface on the personal computer side)

Table 1. FPGA area usage for implemented interrupt controller

Number of Slices:	112	out of	768	14%
Number of Slice Flip Flops:	47	out of	1536	3%
Number of 4 input LUTs:	198	out of	1536	12%
Number of bonded IOBs:	9	out of	96	9%
Number of GCLKs:	1	out of	4	25%

which we obtained the FPGA area usage listed in table 2. For this new design an average of 52.2 % of FPGA area is used, for the smart phone type of applications no more than 32 devices might be connected, for this reason we found our proposed interrupt controller to be feasible even for larger number of devices in the chosen FPGA.

Table 2. FPGA area usage for 32 interrupt sources and a 32-entry queue.

Number of Slices:	762	out of	768	99%
Number of Slice Flip Flops:	231	out of	1536	15%
Number of 4 input LUTs:	1250	out of	1536	81%
Number of bonded IOBs:	40	out of	96	41%
Number of GCLKs:	1	out of	4	25%

V. CONCLUSIONS AND FUTURE WORK

We designed and evaluated a platform-level interrupt controller for a smart phone based on the PXA270 microprocessor. We integrated a digital system to prove the correct functionality of the system by randomly generating interrupt requests from a personal computer attached to the interrupt controller implemented on a Xilinx Spartan II FPGA. We emulated the PXA270 microprocessor with a microcontroller, we generated interrupt signals within the operating frequency range of the PXA270 microprocessor to guarantee the correctness of our interrupt controller.

The developed interrupt controller uses 25,200 gates of the FPGA, allowing other peripheral modules (needed by the smart phone) to be implemented in this device in order to optimize the PCB area. The use of FPGAs in this kind of projects allows us to upgrade our design by adding new elements to the smart phone.

REFERENCES

Aldec Corp. 8259, *IP Core Programmable Interrupt Controller Datasheet* (2004).

Alatek Inc. Alatek AL8259 *IP Core Application Note.* Dec. (1999).

De Gloria, A., P. Faraboschi, and M. Olivieri, "A Self Timed Interrupt Controller: A Case Study in Asynchronous Micro-Architecture Design," *Proc. of ASIC94,* Rochester, NY, 296-299 (1994).

Intel Corp. 80C186EB/80C188EB *Microprocessor User´s Manual* (1995).

Intel Corp. Intel PXA27x *Processor Developer´s Kit Parts Lis*. (2004a).

Intel Corp. Intel PXA27x *Processor Family Design Guide* (2004b).

Intel Corp. Intel PXA27x *Processor Family Developer´s Manual* (2004c).

Intel Corp. Diagnostics for the Intel PXA27x *Processor Developer´s Kit. User´s Guide* (2004d).

Intel Corp. FPGA Code for the Intel PXA27x *Processor Developer´s Kit Main Board* (2004e).

Intel Corp. Intel PXA27x *Processor Optimization Guide* (2004f).

Intel Corp. Intel PXA27x *Processor Developer´s Kit Schematics* (2004g).

Nakashima, K., S. Kusakabe, H. Taniguchi and M. Amamiya, "Design and Implementation of Interrupt Packaging Mechanism," *Proc. of the International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems,* Los Alamitos, CA, 95-102 (2002).

Windows CE. http://msdn.microsoft.com/embedded/windowsce/default.aspx (2005).

Received: April 14, 2006.
Accepted: September 8, 2006.
Recommended by Special Issue Editors Hilda Larrondo, Gustavo Sutter.