Design and Implementation of Multiprocessor System on a Chip (MPSoC) Based on FPGA

Chia-Ying Tseng and Yen-Chih Chen
Department of Computer Science and Engineering
Tatung University
cytseng@ttu.edu.tw and jeter2048@yahoo.com.tw

Abstract- *With the growing of multimedia codec types, the huge amount of produced computing can not be handled by a single processor now. Therefore, we hope that the programs which include many computations can be processed by multiprocessors. In addition, the core operated in embedded system platform also gradually becomes multiprocessor from a single processor.*

In this paper, we design a four-processor system using NiosII soft-core and implement our MPSoC architecture (includes 4K I-cache, 1MB SRAM, 32MB SDRAM, 16MB Flash) and design the executable programs running in multiprocessor via hardwire Mutex element. We use the hardwire Mutex core to access the shared memory in the program. The implemented result shows that the quad-core system architecture that we proposed can execute the program concurrently at the same time.

Keywords: FPGA, Nios II, MPSoC, Soft-core, Multi-core.

1. Introduction

The design goal of microprocessor is gradually changing from higher-frequency computation to lower-power consumption in recent years. For satisfying the requests of better performance and not increasing too much power consumption, another obvious difference of microprocessor is that the design goal transfers from increasing frequency to adjusting multi-core architecture [1].

Utilizing the characteristics and techniques of multi-core processors can flexibly implement a small-size, low-cost, high-performance, and reliable systems in many embedded applications.

Nowadays, the integrated multi-core processors are changing the architectures of embedded system. The new processors which aim at embedded applications can implement a high-performance system design that can increase the computation density at specific frequency.

The development of SoC (System on a Chip) is gradually changing from uni-processor to multi-processor in recent years. However, the advantages of using soft-core processor in designing and implementing system on a chip are avoiding the restriction of manufacturing process in semi-conductor and more flexible than hard-core processor [2][3].

At present, the development of multi-media application in embedded system is requiring to integrate a large number of programs and functions into a single platform. By designing these applications on a reconfigurable chip, we can not only reach the goal of time-to-market [4] but also carry out reusable soft-core in hardware architecture. If the architecture we design initially will be used in relative design process later, we can refer to which by means of calling it directly. Another advantage of adopting FPGA (Field Programmable Gate Array) is that we can constantly reconfigure the hardware architecture that we want to design and implement it for some specific applications. Hence, the importance of FPGA in embedded system is gradually rising with no doubt.

In this paper, we design a quad-core multiprocessor system on Cyclone II FPGA chip (4K I-cache, 512KB SRAM, 8 MB SDRAM and 4MB Flash are included) by adopting the soft-core processor and the tool chain offers by Altera® Corporation. And we use hardware Mutex core to design the memory-sharing mechanism in quad-core multiprocessor architecture. Finally we also design the executable test programs running in the quad-core multiprocessor by implementing the Mutex core components [5][6].

2. Background

The most significant benefit of multi-processor system is the improvement of performance, but the complexity of designing the system will also increase. Hence, the architectures of multi-processor system are often restricted in Workstation or complicated High-end PC. For multi-processor systems, there are various ways for categorizing, and we divide this system into two types in the paper. One is Autonomous Multiprocessor System and the other is Non-Autonomous Multiprocessor System. The manner for identifying them is that if the processors in each system are sharing the same resources.

The meaning of Autonomous Multiprocessor Systems is that the execution among processors is absolutely independent without any synchronization or communication. This type of system is less complicated and fewer problems while designing it. The main reason is that every processor work solely and cannot interfere the other processors' operations.

In the Non-Autonomous Multiprocessor system, resources can be shared among processors. It is very useful to adopt resource-sharing mechanism in multiprocessor architectures, but it should be noticed the time which resource will be shared and how to cooperate each other among different processors while sharing the resources.

3. Design of Multiprocessor System on a Chip (MPSoC)

3.1. System Architecture

To evaluate the characteristics of high-performance and low-cost about multiprocessor system in embedded SoC, we improve the multiprocessor system architectures from dual-core to quad-core and integrate our system into a FPGA. Each processor has its own 4KB I-Cache but has no D-Cache. Figure 1 shows the block diagram of our system architecture.

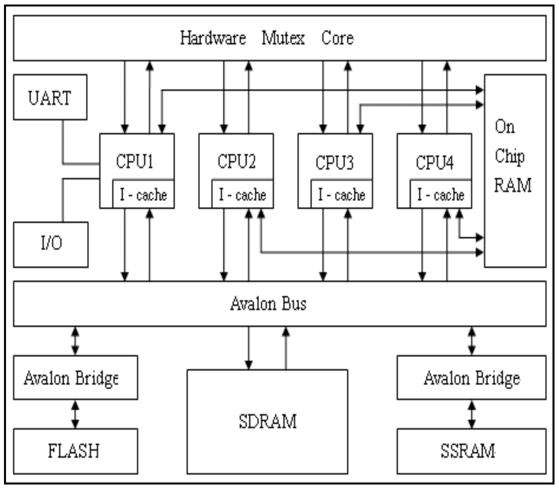


Figure 1. System block diagram

The system provides an independent timer for every processor and the memory has 4MB FLASH, 8 MB SDRAM, and 512KB SSRAM; these memories are allowed to be shared by four processors in the system. However, SSRAM is only permitted to be accessed by the data stream of each processor, FLASH and SDRAM allow the data stream and instruction stream in each processor pass through. We utilize a Mutex Core to implement the shared memory component in multiprocessor system; the data stream in each processor will pass it through but instruction stream would not. By adopting this component, we can design a multiprocessor system to access memory devices.

3.2. Hardware/Software Specification

3.2.1. Hardware Specification

The development board that we use is Cyclone II FPGA Nios II Starter Board [7-9], and the hardware specification is as follows:

■ Altera Cyclone EP2C35F672C6 FPGA
● 35K logic elements (LEs)
● 105 M4K memory blocks
■ Memory subsystem
● 512-Kbyte standard synchronous SRAM
● 4-Mbyte Flash ROM
● 8-Mbyte DDR SDRAM

3.2.2. Software Specification

The software we use is a series of development suites Altera® Corporation offers, and the software specification is as follows:

■ Altera® Quartus® II v7.2 sp2
■ SoPC Builder tools
■ Nios II Embedded Design Suite v7.2 sp2
■ MegaCore® IP Library v7.2 sp2

4. Implementation

4.1. System Design Flow

A complete system design flow can be divided into three parts, which are hardware design [10], system design [11], and software design respectively. At first, the section will describe how to design hardware. We initially establish a project via Quartus® II, and then we use SoPC Builder [12] to build the system that we would like. The overall hardware design also belongs to the system integrated design. With extra parameters or instructions, we can merge the required functions to the system we design originally by means of SoPC Builder. We import the completed system into the project established in Quartus® II, and set certain required parameters for some devices at this time. The design flow chart of development board is shown on Figure 2.

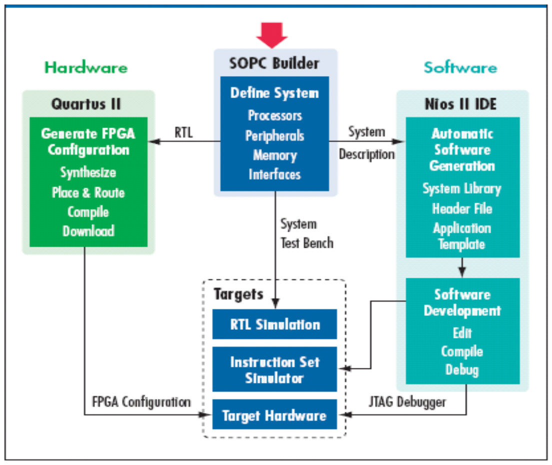


Figure 2. Design flow chart of development board

When we want to attach some customized hardware, we use the compiler Quartus® II offers to develop the hardware we would like to implement. After the simulation of hardware is verified with no errors, we synthesize the hardware with the completed system developed early in Quartus® II. The last step is deploying all external pins and compiling of overall system. Until now the bottom system design has come to an end and the section we describe and build in Quartus® II above is called the integrated design of hardware and system.

In the following section we use Nios II Embedded Design Suite to design the software we need and the software will be placed in our system for execution. In the system design phase, we can debug our programs by adding a JTAG UART; its function is showing the execution result of our programs in the control screen of PC terminal. That there is one thing remained to be paid attention to is which Debug Module you choose for debugging and it must be decided in processor configuration. When the design phase of hardware, system and software is finished, we burn the system file in the target board via the programmer function Quartus® II offers; however, before this step, we must set up the name and type of FPGA (the target board that we use).

When the process of burning is over, Nios II Embedded Design Suite offers the execution tool for us to execute the software which is placed in the system on the development board. The last execution result will be displayed in our computer screen.

4.2. System Implementation

There is a Message Buffer implemented by our designed On-Chip RAM to receive the Message generated by each processor in this system. When processor grabs the hardwire Mutex core, it can write the Message to Message Buffer. We have four processors in this system. Then, the sequence of their writing is the sequence of their grab. Because UART is only connected to CPU1, the CPU1 is responsible for entire system output. We will detect the Message Buffer through CPU1 in our design. If the Message Buffer has some new Message write in, the CPU1 will show it in PC console. Figure 3 is our program scheme.

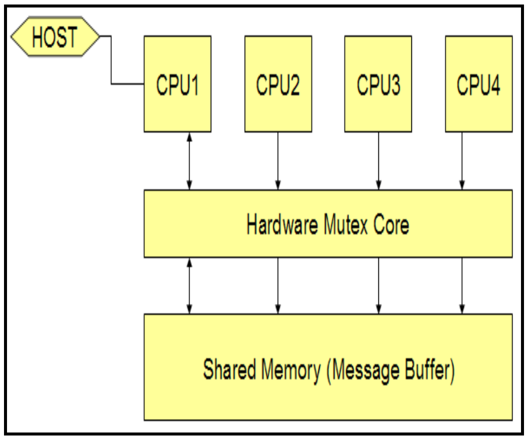
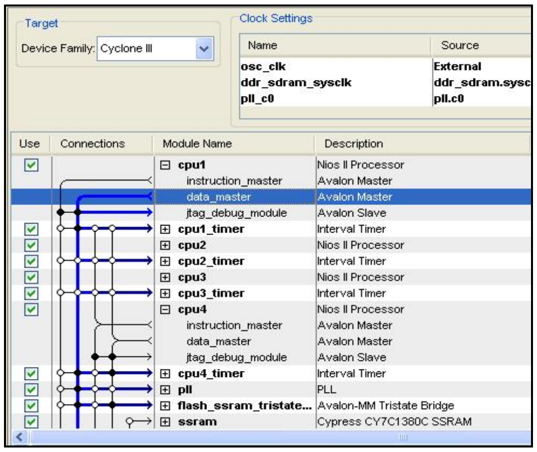


Figure 3. The program scheme

At first, we establish a new project via Quartus® II (illustrated in Figure 4) and use SoPC Builder to design the overall system; after the system architecture is finished, we import it into the project established early and modify required pins via Top Module. After compilation, our system is already established.

Now we will test if our system works normally. We design every project executed on each processor via Nios II Embedded Design Suite. Because our system has four processors, we must establish four different programs placed on separate processor for execution. Figure 5 shows the design program with Nios II EDS.

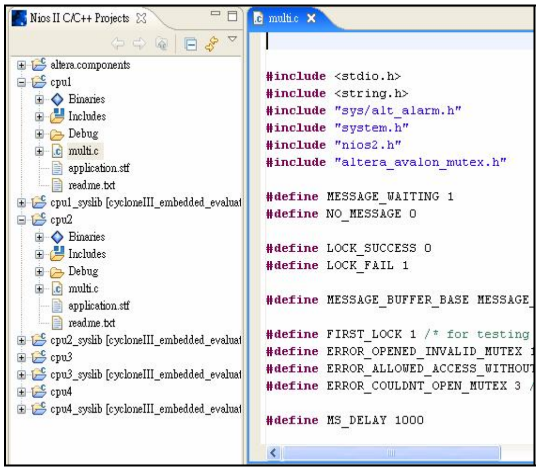


Device Family: Cyclone III

Name	Source
osc_clk	External
ddr_sdram_sysclk	ddr_sdram.sysc
pll_c0	pll.c0

Use	Module Name	Description
✓	cpu1	Nios II Processor
	instruction_master	Avalon Master
	data_master	Avalon Master
	jtag_debug_module	Avalon Slave
✓	cpu1_timer	Interval Timer
✓	cpu2	Nios II Processor
✓	cpu2_timer	Interval Timer
✓	cpu3	Nios II Processor
✓	cpu3_timer	Interval Timer
✓	cpu4	Nios II Processor
	instruction_master	Avalon Master
	data_master	Avalon Master
	jtag_debug_module	Avalon Slave
✓	cpu4_timer	Interval Timer
✓	pll	PLL
✓	flash_ssram_tristate...	Avalon-MM Tristate Bridge
✓	ssram	Cypress CY7C1380C SSRAM

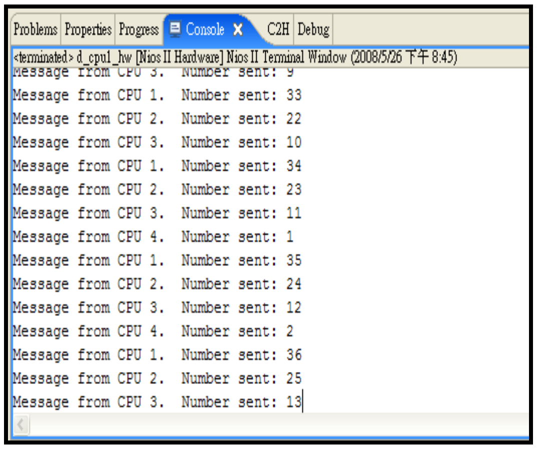
Figure 4. Design system using SoPC Builder



```
#include <stdio.h>
#include <string.h>
#include "sys/alt_alarm.h"
#include "system.h"
#include "nios2.h"
#include "altera_avalon_mutex.h"

#define MESSAGE_WAITING 1
#define NO_MESSAGE 0

#define LOCK_SUCCESS 0
#define LOCK_FAIL 1

#define MESSAGE_BUFFER_BASE MESSAGE_

#define FIRST_LOCK 1 /* for testing
#define ERROR_OPENED_INVALID_MUTEX 1
#define ERROR_ALLOWED_ACCESS_WITHOUT
#define ERROR_COULDNT_OPEN_MUTEX 3 /

#define MS_DELAY 1000
```

Figure 5. Design program with Nios II EDS

The program we execute is accessing an On-Chip RAM via a Mutex Core, and we design a shared data area in the On-Chip RAM. Every processor that owned the Mutex Core will update the count value (count++) and only the first one processor is connected to JTAG_UART. As soon as the data in shared data area change, the first one processor will display it in the screen via the JTAG_UART it connects. The operating principle of processors is estimating different processors by catching the *cpuid* that every processor owns. Which processor can own the Mutex Core is decided by its order of writing their *cupid* to the owner of Mutex Core. The last execution result will be displayed in the PC Nios II EDS. console .The program result shows in Figure 6**.**



```
<terminated> d_cpu1_hw [Nios II Hardware] Nios II Terminal Window (2008/5/26 下午 8:45)
Message from CPU 3.  Number sent: 9
Message from CPU 1.  Number sent: 33
Message from CPU 2.  Number sent: 22
Message from CPU 3.  Number sent: 10
Message from CPU 1.  Number sent: 34
Message from CPU 2.  Number sent: 23
Message from CPU 3.  Number sent: 11
Message from CPU 4.  Number sent: 1
Message from CPU 1.  Number sent: 35
Message from CPU 2.  Number sent: 24
Message from CPU 3.  Number sent: 12
Message from CPU 4.  Number sent: 2
Message from CPU 1.  Number sent: 36
Message from CPU 2.  Number sent: 25
Message from CPU 3.  Number sent: 13
```

Figure 6. The program result in the PC console

5. Performance Evaluation

We design two benchmarks via Nios II EDS tools. The goal of our benchmark is evaluating and analyzing the system performance.

5.1. Variable Computation in Order

We use single variables as operand in this benchmark. The benchmark runs three variables computation. They have some dependent computation and must communicate in sequence. Mutex core will decide who comes to the share memory and accesses the share parameters. The program is executed on six million times loops consisting of eight multiplication instructions. After these multiplications, we have some additions or subtractions. The purpose of this way is to hope that the processors execute the program in order.

The program designed in mutex area gives three flags to share memory. Assume that the CPU order is two, three, four then one, and the flags default are zero. When the CPU2 finish its work and grab the Mutex Core, it will set the ”one” to flag2. The CPU3 and CPU4 do the some work. After the CPU1 grabs the Mutex, it will check three flags whether they are equal to zero or not. The method ensures that the processor communicate will execute in order. The experiment result is shown in Figure 7. The execution time for one processor system is 99.78 seconds, two is 70.75 seconds, three is 58.26 seconds and four is 54.95 seconds.

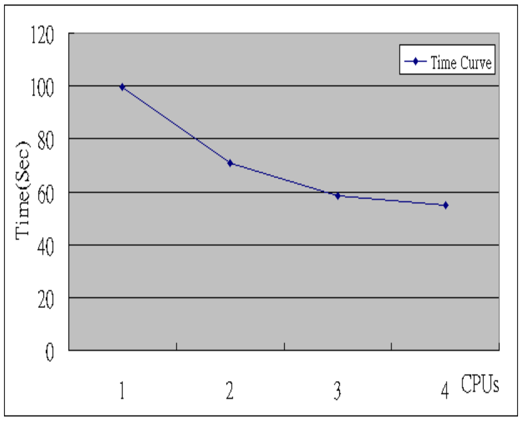


Figure 7. The execution time of VAR benchmark

5.2. Array Computation Out of Order

There are two arrays to deal some operations with each other. It was designed by mixing some simple arithmetic operations (such as: plus, minus, multiply, and divide). Each array has twenty thousand elements and assigned with initial values. The total operations contained about two million instructions. In order to increase the amount of array's data storage we replace the On-Chip RAM with SRAM in this experiment. Since the On-Chip RAM only has 1K Bytes free memory space and the SRAM has 512K Bytes.

In this case, the processors communicate out of order. When CPU2, CPU3, and CPU4 finish their work, they will grab the Mutex. For any one of them grabs the Mutex then it will access the share memory. We only need to ensure that the CPU1 access share memory last.

The experiment result for the array computation out of order is shown in Figure 8. The execution time for one processor system is 63.01 seconds, two is 43.6 seconds, three is 34.32 seconds, and four is 29.27 seconds.

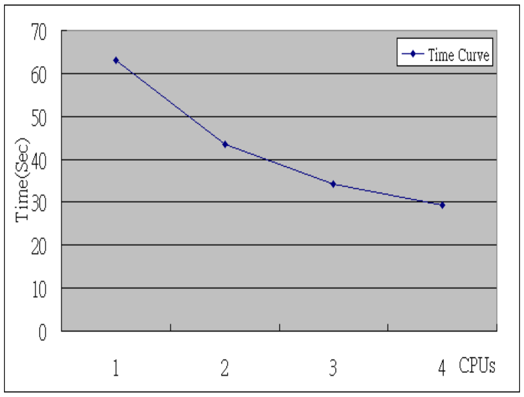


Figure 8. The execution time of Array benchmark

5.3. Performance Evaluation and Analysis

In our experiment, we use one, two, three, and four processors system running the benchmark program to measure the speedup. At the beginning, we run benchmark program to one processor system and test its execution time. Then, distribute our benchmark program for two, three, and four processors system independently. The speedup of two benchmark programs is shown in Figure 9. The figure shows that the slope of these two lines becomes gradually small.

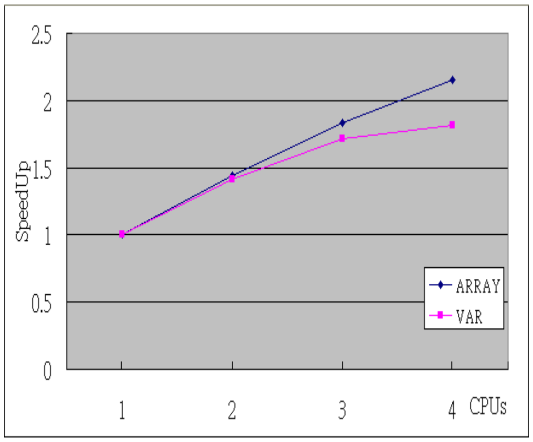


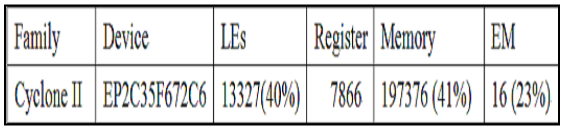
Figure 9. The speedup of two benchmarks

In VAR experiment, we make the one processor system as the standard system. Its execution time is about 99.78 seconds. When VAR benchmark runs on two processors system, it spends 70.75 seconds. The speedup for two processors system is 1.41. When program runs in three and four processors system, it spends 58.26 and 54.95 seconds. Their speedups are 1.71 and 1.82.

In Array experiment result, the execution time for one processor system is 63.01 seconds, two is 43.6 seconds, three is 34.32 seconds and four is 29.27 seconds. The speedups are 1, 1.445, 1.836, and 2.152.

The speedup of VAR experiment result is increasing slower than Array's. The reason of this situation results from the processors communication time. Because of communicating in order, it will spend more execution time. The VAR experiment is belong to this condition but Array experiment is in other way round.

The resource utilization in our system is shown in Figure 10. In our design for four processors uses 13327 LEs, 197376 bits memory, and 16 embedded multiply blocks. The percentages of these are about 40%, 41%, and 23%.



Family	Device	LEs	Register	Memory	EM
Cyclone II	EP2C35F672C6	13327(40%)	7866	197376 (41%)	16 (23%)

Figure 10. The resource utilization in our system

6. Conclusion and future works

We proposed a customized multiprocessor system via Tool Chain that Altera® Corporation offers and had clear understanding in system design process. Then we implemented two benchmark programs for our multiprocessor system. The result of our experiment shows that the speedup in the non order case increases rapidly.

For utilizing the shared memory resource in multiprocessor system without causing any system or data errors, we employ hardwire Mutex Core to achieve the goal; however, though the mechanism can protect multiprocessors from mutually accessing memory, there are some problems remained for us to discuss Mutex.

We discover that the Mutex spends a lot of time in single communication. The later research tends to how to communicate quickly and securely at the same time. The improvement of accessing mechanism of shared resource in multiprocessor system will still be one of main research topics in MPSoC field.

Another problem is that the same physical program memory sharing by multiprocessors. In multiprocessor system, the stored program memory is in the same memory chip. The memory only has one pair read/write port. When the code is increasing, it may cause the performance going down.

The future works we aim at are that the operating system of multitasking and multithreading, and how to utilize the characteristics of multiprocessor to design relative codec multimedia applications. We will also attempt to port uCLinux or uC/OS-II in our proposed multiprocessor system.

7. Acknowledgement

Financial support of this research by Tatung University, Taipei, Taiwan, under the grant B96-I02-028 is gratefully acknowledged.

References

[1] L. Benini and G. de Micheli, “Networks on chips: A new SoC paradigm,” Proceedings of the IEEE Computer, vol. 35, No. 8, pp. 70-78, Jan. 2002.

[2] Sheldon, D. Kumar, R. Vahid, F. Tullsen, D. Lysecky, “Conjoining Soft-Core FPGA Processors,” Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, Pages 694-701, Nov. 2006.

[3] K. Compton, “Reconfigurable Computing: A Survey of Systems and Software,” Proceedings of the ACM Computing Surveys, vol. 34, No. 2, Pages171-210, June 2002.

[4] A. Jerraya and W. Wolf, “Guest Editors' Introduction: Multiprocessor Systems-on-Chips,” Proceedings of the IEEE Computer, vol. 38, No. 7, Pages 36-40, Jul. 2005.

[5] Chia-Ying Tseng, Liang-Teh Lee, Chun-Hung Chen, and Yen-Chih Chen, “A Soft-Core Based Reconfigurable Multiprocessor System,” Proceedings of 2007 National Computer Symposium, Vol. 2, pp. 437-443.

[6] Olli Lehtoranta, Erno Salminen, Ari Kulmala, Marko Hännikäinen, and Timo D. Hämäläinen, “A parallel MPEG-4 encoder for FPGA based multiprocessor SoC,” International Conference on Field Programmable Logic and Applications, 2005.

[7] Altera, “Cyclone II FPGA Starter Board Reference Manual,” Altera Corporation, October 2007.

[8] Altera, “Nios II Processor Reference Handbook,” Altera Corporation, October 2007.

[9] Altera, “Profiling Nios II Systems,” Altera Corporation, February 2006.

[10]Altera, “Nios II Hardware Development Tutorial,” Altera Corporation, October 2007.

[11]Altera, “Creating Multiprocessor Nios II Systems Tutorial,” Altera Corporation, December 2007.

[12]Altera, “Quartus II Handbook Volume 4: SOPC Builder,” Altera Corporation, October 2007.