An FPGA-Based Experiment Platform for Multi-Core System

Jianguo Xing, Wenmin Zhao and Hua Hu
College of Computer Science and Information Engineering
Zhejiang Gongshang University
Hangzhou 310018,China
{jgxing, zhaowm, huhua}@zjgsu.edu.cn

Abstract

The undergoing radical change of microprocessor architecture, from single-core design to multi-core design on a chip, has posed great challenge to computer system education. In this paper, we introduce an educational experiment platform for multi-core system design with commercial off-the-shelf FPGA chips and free open IP cores. From this simple platform, student can learn some common design patterns and techniques in multi-core system which can be a good start for real application design.

Keywords: multi-core system design, parallel computing, FPGA, experiment platform

1. Introduction

Multi-core or many-core[1] may be the only viable way to improve processor performance under current silicon technology due to physical limitations and power constraint. The main manufacturers have already released several dual-/quad-core desktop CPUs in past few years and more ambitious research plan such as 80-core or more are also undergoing. Taking advantage of multiple processor cores requires more system-level design cooperation between software and hardware designers. Unlike the ILP and TLP[2][3] which are heavily exploited with advance compiler and superscalar technology and transparent to most software programmers, the designer must explicitly know the inside hardware structure of multi-core or many-core system and be careful with their software which needs some experiences different from those learned from single-core programming.

But unfortunately, for most of us, especially at university, are still not ready for this. To unleash the power of multi-core, the designer has to delve into the detail of the hardware and software partition, interconnection of modules, processor features, memory organization, task communication, synchronization and etc., which are traditionally hidden by languages and compilers. Experiences from sequential program is outdated, traditional tools to design, debug and profile are not work well. And the designer's experience and rules of thumb prevail in the multi-core design flow.

The undergoing radical change of microprocessor architecture, from single-core design to multi-core and many-core design on a chip, has posed a great challenge to computer system education, which involves several different fields which taught independently for students in computer science. Although the main processor manufactures have turned to multi-core designs attempting to extract ever more performance. However, little is known on how to build, program, or manage systems with multi-core processors, and lacks basic curriculum and experiment tools at universities. We argue that there is a strong need to provide a new integrated course and lab for multi-core system study.

In recent years, the reprogrammability of FPGA makes them ideal for educational purposes because it allows students to design and implement a hardware prototype system easily and quickly. For example, with the help of EDA tools students can design a small RISC CPU design and validate it on an FPGA system in just a few minutes. Some commercial boards provide flexible solution for researcher or student implementing their design. Some EDA providers also support multi-core solutions in their EDA products. But currently there is still no systematically tutorial and platform for this purpose.

In this paper, we introduce an educational experiment platform for multi-core system design with commercial off-the-shelf FPGA chips and free open IP cores. With this simple platform, student can learn some common design patterns and techniques which can be a good start for real application design. And we believe such a system will inspire student interests across the fields for multiple processors: hardware/software co-design, parallel systems, operating systems, compilers, programming languages, and so on. It is our goal to give the students an integrated view about multi-core design from both hardware and software perspective, especially for embedded application.

2. Multi-Core Design Patterns

There are several reasons for developing a system using more than one processor which must be kept in mind for our educational purpose. Also there are some distinct design patterns which occurred recurrently in past experiences. Following is a brief summary of the typical scenarios and patterns of multi-core design.

2.1. Typical Scenarios for Multi-core Design

The reasons for developing a multi-core system may simply be performance consideration. In others, it is functionality, modularity, and such concerns.

2.1.1. Multiple Independent Functions/Modules. The design may have multiple, independent set of processing tasks to be performed. An attractive way to solve the problem could just mean creating various processing modules that are completely independent, each dedicated to its own processing task. Each processing module is assigned a unique processor and peripheral set.

2.1.2. Load Balance. A common design scenario is the presence of data intensive or protocol processing tasks. In these cases, some slave processors are used to offload the data intensive or number crunching tasks, while the master processor performs overall co-ordination, setting up of computations and host interface. The slave processors may contain specialized functions or interfaces to allow it to meet computation performance requirements. Some examples of this scenario include, network offload, streaming media processing, security algorithm, etc.

2.1.3. Stream processing. For handling stream-oriented computation, processors may be arranged to act upon the data stream in a pipeline fashion. Each stage in the multiprocessor pipeline, acts upon one portion of the computation before passing it on to the next processor. All the processors act concurrently. This solution is typically used to increase the throughput of a solution.

2.1.4. Reliability and redundancy. A processing system may be replicated multiple times to provide for reliability and redundancy. For example, Triple Mode Redundancy (TMR) is a related concept, which is outside the scope of this paper.

2.2. Multi-Core Design Patterns

2.2.1. Master/Slave (or Processor/Co-processor) Pattern. Such system has one general purpose CPU as the master processor which allocates and dispatches tasks to other slave processors. Slave processors are designed to process some special algorithms which usually accelerated with hardware. In its classic telecom and media processing forms, this pattern's advantage comes from running data-heavy algorithms on a specialized DSP or network processor, while keeping the rest of the system software on a general-purpose CPU. This specialization means that these systems are normally implemented on purpose-built hardware —— the high-level design of software will usually follow from the structure of the hardware it runs on. Since these systems have only one general purpose CPU core, conventional single-core tools, operating systems and design techniques can be used for design and debug of the master processor of the system.

2.2.2. SMP Pattern. In a symmetric multiprocessing (SMP) system, a single OS runs on top of two or more essentially identical processing cores that share memory. SMP pattern systems can dynamically balance tasks over a pool of processor cores. This ability to redistribute processing power to match the tasks is a major strength of SMP systems, and it is what makes them the most flexible and adaptable among the three patterns. With modern SMP RTOS, this flexibility comes with no cost in determinism or real-time responsiveness. An SMP pattern system requires two things: a symmetric, multicore shared-memory processor platform and an SMP-capable OS.

2.2.3. Distributed Processor Pattern. This pattern is a variation of the master/slave pattern and covers any multiprocessor system made up of a collection of fully independent, networked nodes. To use this pattern, designers are required to first partition their system and then find a suitable inter-node communication system. Once the system is partitioned, design and implementation of each individual node proceeds just like a standalone system..

3. Experiment Platform

3.1. Hardware Resources

Figure 1 shows the FPGA board which is used as our experiment platform. The on board hardware resources includes an EP2C35F672 Cyclone II device, 16MB DDR2 SDRAM, 16MB Flash, 1MB SRAM, a serial configuration device(EPCS16), led & keys and a JTAG interface connector. The target system is downloaded to the board with an USB-blaster Cable.

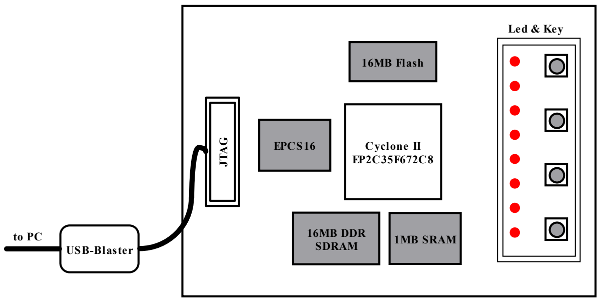

Figure 1. FPGA Board

3.2. Basic Design Component

There is no need to reinvent the wheel because Altera's EDA toolkits (Quartus II and SOPC Builder)[4][5] have provided rich IP components and related software drivers which can be used as basic blocks for high-level design. Such IP cores range from CPU cores to all kinds of peripherals. Some open IP cores such as those from www.opencores.org can also be freely available.

Altera Avalon bus is a standard interface to glue the NiosII cores and other components. For some open source IP cores WISHBONE bus is preferred. NOC (network on chip) interconnect is also used for many-cores or some complex cases.

Altera also provides some IPC (Inter-process Communication) mechanisms to simplify multi-process design, which includes: Mutex and Mailbox.

The mutex core provides a hardware-based atomic test-and-set operation, allowing software in a multiprocessor environment to determine which processor owns the mutex. The mutex core can be used in conjunction with shared memory to implement additional inter-processor coordination features, such as mailboxes and software mutexes.

Mailbox core is designed for use in Avalon-based processor systems, Multiprocessor environments can use the mailbox core with Avalon interface to send messages between processors.

A JTAG UART core is used as standard input and output. The system has only one UART and connect to a processor. Other process or tasks read or write message through this processor.

3.3. Design Flow

A typical design flow of application which runs on multi-core system can be summarized as follows:
a. Partitioning the computation into small tasks.
b. Analyzing the communication needed to coordinate task execution and choose communication structure.
c. Mapping the tasks into hardware and determine the processor number, function and interconnection pattern.
d. Implementing and verifying the design on hardware.

4. Laboratory Design

Here we select four labs as examples to illustrate the typical hardware/software co-design method of multi-core system. They are: producer-consumer problem, leader election problem, Jacobi finite difference method and edge detection.

4.1. Producer-Consumer Problem

Producer-Consumer problem (also known as bounded-buffer problem) is a classical example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, who share a common, fixed-size buffer. The producer put data into the buffer while the consumer is remove data from the buffer at the same time.

Two independent Nios II processors are added to the system. Each processor has its own code and data memory (on chip memory) and a shared memory as the buffer (Figure 2).

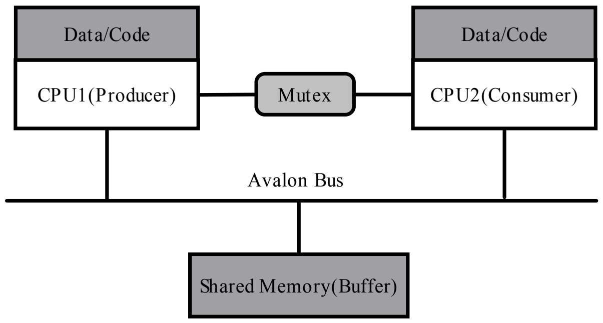

Figure 2. Producer-consumer problem

To prevent the two process access the buffer at same time, here we can use mutex component provided by SOPC Builder. The main code of producer is as following:

```
while(1)
{
  altera_avalon_mutex_lock( mutex, 1 );
  if(!isFull(buffer))
       put_item(buffer,data);
  altera_avalon_mutex_unlock( mutex );
}
```

The code of consumer is similar. From this lab, we can know how to access shared resources safely with mutex for multi-process application.

4.2. Leader Election Problem

The Leader Election Problem is to choose a leader from a set of candidates. Initially, node n_i is only aware of its own identification and a totally order is defined on the set of all identifications.

Here we use a logic ring structure to connect the node with mailbox as figure 3.

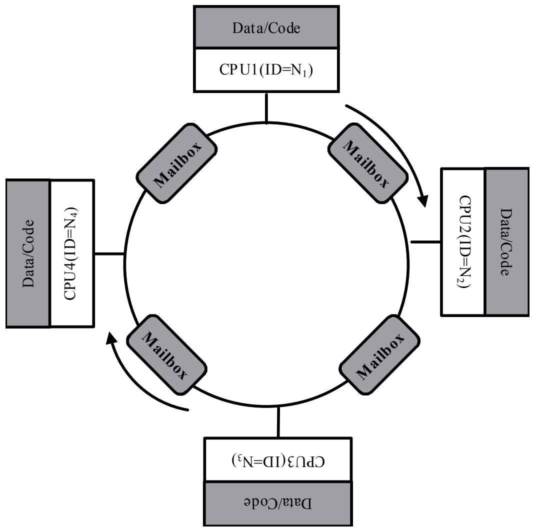

Figure 3. Leader election problem

Node i receive message (the leader id accepted by the sender) from its neighbor (i-1) %n, send message(the leader id) to its neighbor (i+1)%n. The algorithm is as following:

```
leader_id = altera_avalon_mailbox_get(mailbox);
if(leader_id < my_id)
  Leader_id = my_id;
altera_avalon_mailbox_post(mailbox_next,leader
_id);
```

And at round n, each node will know which node is the leader. In this example, we use mailbox to exchange message between different processes.

4.3. Jacobi Finite Difference Method

In this example (see [6]), we consider a 2D numerical solver named Jocobi finite difference method. State of node $X_{i,j}^{(t)}$ at time step t+1 is updated using equation 1 (Figure 4).

$$X_{i,j}^{(t+1)} = \frac{4X_{i,j}^{(t)} + X_{i-1,j}^{(t)} + X_{i+1,j}^{(t)} + X_{i,j-1}^{(t)} + X_{i,j+1}^{(t)}}{8} \qquad (1)$$

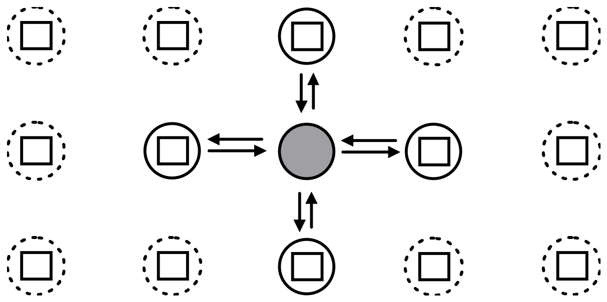

Figure 4. Jocobi Finite Difference Method

Each node connects with its four neighbors using message channel and executes the following logic:

```
for(t=0;t < T;t++)
{
send $X_{i,j}^{(t)}$ to each neighbor
receive $X_{i-1,j}^{(t)}$, $X_{i+1,j}^{(t)}$, $X_{i,j+1}^{(t)}$, $X_{i,j-1}^{(t)}$ from neighbors
compute $X_{i,j}^{(t+1)}$ using equation (1)
}
```

4.4. Edge Detection

This example illustrates typical stream-oriented application design.

Edge detection is the basis of feature detection and extraction in image processing and computer vision which requires a lot of computation. Here we use three processors work concurrently to form a pipeline (Figure 5). Processor 1 reduces the noise in raw image with a Gaussian Filter. Processor 2 calculates the intensity gradient of the image with Sobel operators. And Processor 3 applies a threshold to mark out the edges. FIFOs are used as a synchronization mechanism for stream applications.

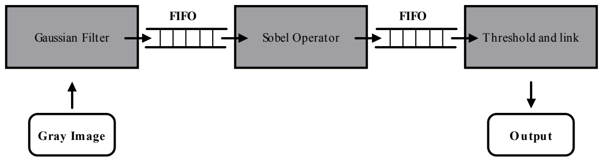

Figure 5. Edge Detection

5. Discussion and Future work

Multi-core processors appear to be the correct solution to support Moore's law, but its design is still strange to most software engineers, who have grown up with sequential codes. To exploit multi-core parallelism in some fashion and get the desired performance, designer is required to know not only application requirement, but also hardware and software structure that is best for that application. In this paper we show some common design patterns of multi-core system and give some examples which can be easily tested and verified on FPGA system. With modern EDA tools, the FPGA has shown its flexibility in rapid prototyping and we show that it is an ideal educational platform for multi-core system.

But here we intentionally skip some important parts of multiple process design, such as operating system and programming language, which are still immature for application. And limited by the FPGA recourses, our platform is restricted to few cores which could not emulate the many-core systems.

References

[1] Alex Krasnov, Andrew Schultz, John Wawrzynek, Greg Gibeling, Pierre-Yves Droz, "RAMP Blue: A Message-Passing Manycore System In FPGAs", *Proceedings of International Conference on Field Programmable Logic and Applications*, Amsterdam, The Netherlands, August 2007, PP. 54 – 61.

[2] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, 2006.

[3] D. Patterson and J. Hennessy, *Computer Organization & Design: The Hardware/Software Interface*, Morgan Kaufmann Publishers, 2007.

[4]Altera Corp., *Nios II Processor Reference Handbook, ver 8.0*, 2008

[5]Altera Corp., *Nios II Software Developer's Handbook, ver 8.0*, 2008

[6] Ian Foster, *Designing and Building Parallel Programs, Concepts and Tools for Parallel Software Engineering*, Addison Wesley, 1995