On the Design, Control, and Use of a Reconfigurable Heterogeneous Multi-Core System-on-a-Chip

Tyrone Tai-On Kwok† and Yu-Kwong Kwok†‡∗
†Electrical and Electronic Engineering, The University of Hong Kong, Pokfulam Road, Hong Kong
‡Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523-1373, USA

Abstract
With the continued progress in VLSI technologies, we can integrate numerous cores in a single billion-transistor chip to build a multi-core system-on-a-chip (SoC). This also brings great challenges to traditional parallel programming as to how we can increase the performance of applications with increased number of cores. In this paper, we meet the challenges using a novel approach. Specifically, we propose a reconfigurable heterogeneous multi-core system. Under our proposed system, in addition to conventional processor cores, we introduce dynamically reconfigurable accelerator cores to boost the performance of applications. We have built a prototype of the system using FPGAs. Experimental evaluation demonstrates significant system efficiency of the proposed heterogeneous multi-core system in terms of computation and power consumption.

Keywords: FPGA, heterogeneous multi-core, network-on-chip (NoC), parallel processing, reconfigurable computing, system-on-a-chip (SoC).

1. Introduction
The continued progress in VLSI technologies allows us to put more cores (from dozens to hundreds) on a single chip to build a system-on-a-chip (SoC) system [11]. This kind of SoC system is commonly found in embedded systems which are prevalent in every aspect of our daily life, such as mobile terminals, portable game consoles, personal media players, etc. It is also found in battlefield, for example, unmanned aerial vehicles (UAVs), killer robots, etc. It is gaining popularity that various services are expected to be integrated on an SoC-based embedded system, e.g., a smart phone can handle phone calls, play multimedia files, access to the Internet, etc., simultaneously. On the other hand, the growing development of multimedia applications (e.g., high quality video) and high data rate wireless technologies (e.g., UWB, WiMax) are driving the need for higher computational power in such system. This also requests for sustained long-period operation of the system (e.g., watching a live soccer match using a mobile terminal with high-quality video; long-time monitoring using UAVs). However, high computational requirement and sustained long-period operation are often two contradicting goals. In essence, energy efficient operation of SoC systems is highly required.

More specifically, in a study conducted by MorphICs [13], it is concluded that the algorithmic complexity of future applications grows much faster than the processor performance, which is governed by the Moore’s Law. However, the battery technology grows at a much lower rate. This means that an embedded system such as a mobile terminal is expected to execute dramatically more computation-intensive tasks, but the increase in battery capacity is simply more than offset by the increase in computational requirement. This suggests that, in order to conserve energy and hence to prolong the lifetime of the system, the computational tasks should be executed in an intelligent way.

To meet computational need and at the same time reduce energy requirement, the growing trend is to apply parallel processing by employing a number of relatively less capable processing cores, instead of one big, power-hungry core [2]. In the embedded systems programming survey conducted in 2005 [6], it was reported that nearly 50% of chips used multiple processors and over 100 projects used more than 10 processors. Moreover, nearly two-third of SoC’s were heterogeneous multi-processor. The future trend is to employ heterogeneous multi-core system-on-a-chip.

Compared with a homogeneous multi-core SoC platform like TILE64 [22], a heterogeneous multi-core SoC platform is more preferable for striking a proper balance between energy efficiency and computational requirement. For instance, a computation-intensive task can be offloaded to a single dedicated hardware, instead of distributing the task to several processor cores, and more importantly the dedicated hardware consumes less power than a single processor.
core. In this paper, we illustrate this concept by describing the design, control, and use of our proposed dynamically reconfigurable heterogeneous multi-core system based on Field Programmable Gate Arrays (FPGAs).

The rest of this paper is organized as follows. In the next section, we present a model of heterogeneous multi-core SoC design. Section 3 describes the system architecture of our proposed reconfigurable heterogeneous multi-core system. In Section 4, we present the design and implementation of the proposed system. Evaluation of the system is presented in Section 5. In Section 6, we present the related work. Finally, we conclude in Section 7.

2. Heterogeneous Multi-Core SoC Model

Figure 1 shows the system architecture block diagram a typical heterogeneous multi-core SoC [11, 18]. In such a system, there is a set of heterogeneous cores (processing elements) of different flexibility and computation characteristics allowing optimal execution of different tasks in the system. Specifically, these processing elements can be reconfigured/utilized to save energy in various ways:

1. embedded processor supports dynamic voltage scaling (DVS), on-demand shut-down, or power-up;

2. reconfigurable logic (e.g., Field Programmable Gate Array [1, 26] and Field Programmable Object Array [12]) acts as offloader to implement computation-intensive algorithms in hardware;

3. configurable processor (e.g., [7, 19, 21]) customized to execute specific tasks; and

4. dedicated hardware (e.g., cryptographic cores, DSP cores) for most energy efficient executing of tasks.

For energy efficient operation of the SoC system, the goal is to judiciously reconfigure the processing elements in the system to execute various tasks of applications. This is illustrated in Figure 2. Suppose that there are eight processing elements in the system and each processing element can be reconfigured in a number of ways (e.g., an embedded processor can be reconfigured to work at different operating voltages for different tradeoffs between energy consumption and computational capability), the figure shows the optimal allocation of processing elements to the tasks of two applications. Here, we consider a task allocation is optimal if it can meet the deadline or performance requirement of an application using minimal power consumption. This is in fact a space-time-configuration problem for energy efficient scheduling of tasks to the processing elements, where different configuration at different time is applied to different processing element.

![Diagram](a) multi-core SoC

![Diagram](b) energy efficiency vs. flexibility

Figure 1. Block diagram of a heterogeneous multi-core architecture and the energy efficiency versus flexibility tradeoff of different kind of processing elements.

Figure 2. Optimal resource allocation of tasks to different processing elements using different configurations.
In this paper, however, our goal is not to develop a scheduling algorithm for the resource allocation problem. Instead, our focus is on the design of a reconfigurable heterogeneous multi-core system using FPGAs.

A typical FPGA system consists of a sea of regularly-structured configurable logic blocks for implementing different digital circuits. With these logic blocks, massively parallel circuits can be constructed to accelerate computation-intensive software tasks. The flexibility of FPGAs emerges from the fact that FPGAs can be dynamically reconfigured at runtime. With this ability, a portion of the FPGA can be partially reconfigured without stopping the functionality of the unchanged sections, enabling the FPGA to fully and rapidly adapt to user needs. This also makes it easy for upgrading or changing the functionality of a part of the FPGA.

Under our proposed system, in addition to conventional processor cores, we introduce dynamically reconfigurable accelerator cores to boost the performance of applications. Furthermore, we propose a communication network for interconnecting different cores in the system.

3. Proposed System Architecture

Figure 3 depicts the system block diagram of our proposed reconfigurable heterogeneous multi-core system. As can be seen, there are two types of cores to be implemented on the FPGA—m accelerator cores and n processor cores. The accelerator cores are implemented as reconfigurable modules [27] on the FPGA. They are intended for executing computation-intensive tasks in hardware, such as encryption, compression, encoding, etc. By implementing these cores as reconfigurable modules, at one time a module might be executing an encryption algorithm while at another time, the module might be executing a compression algorithm after it is reconfigured.

One may wonder: Since the accelerator cores can be used to execute computation-intensive tasks for higher performance, then what is the use of those processor cores? There are two main reasons for the incorporation of the n processor cores. First of all, as shown in Figure 3, each accelerator core is tightly coupled to a processor core, which results in a distributed control of the accelerator cores. This is more advantageous when compared to the case where a single processor core takes control of all the accelerator cores. Secondly, the n processor cores provide a building block for those users who are familiar with the traditional parallel programming paradigm. For maximal performance of applications, users can make use of free IP-cores which are available on the Internet [14], such as DES/AES cryptographic core, FFT DSP core, etc. However, if the required IP-cores are not available at hand, they can enhance the performance of applications by adopting parallel processing on the n processor cores.

As illustrated in Figure 3, one of the processor cores is designated for the reconfiguration of the accelerator cores (i.e., the reconfigurable modules). We have addressed the issues involved in reconfiguring the reconfigurable modules in [10]. Specifically, we designed a scheduler to allocate tasks for executing in different regions (modules) of the FPGA. On the other hand, in our earlier work [9], we have implemented a reconfigurable SoPC (System-on-a-Programmable-Chip) based cryptographic engine. The cryptographic engine can be reconfigured on-the-fly (i.e., while the system is in operation) to realize either a DES or AES encryption engine. Moreover, the parameters of the encryption algorithms such as the S-Box can also be dynamically reconfigured. In this paper, we are not focusing on the design and implementation of such reconfigurable modules. Rather, we focus on the control and use of the processor cores, and hence the accelerator cores. For instance, one of our major goals in this study is to develop a communication network for interconnecting the n processor cores.

4. System Design and Implementation

4.1. Design of a Communication Network

In our proposed reconfigurable heterogeneous multi-core system, the communication network design, or network-on-chip (NoC) design, is very crucial to the performance of the system since it is the basis for the distributed cooperation of the accelerator and processor cores.

In this paper, we propose a communication network called the cone network, which is illustrated in Figure 4(a).
Because FPGAs have abundant on-chip interconnect resources, as opposed to the traditional mesh network (see Figure 4(b)) adopted in many multi-core processors, we can add more links between processor cores. Specifically, our proposed cone network shares the properties of a tree network (see Figure 4(c)) and mesh network. For example, broadcasting on a mesh network requires $O(n)$ steps while that on the tree network and hence the cone network requires only $O(\log n)$ steps.

![Diagram](a) proposed cone network (b) mesh network (c) tree network

**Figure 4. Different inter-core communication networks.**

Based on the cone network, as for the $n$ processor cores in our proposed system architecture, we have implemented a multiprocessor subsystem of 15 MicroBlaze\(^1\) [28] processors on the Xilinx Virtex-II Pro XC2VP50 FPGA. The structure of the subsystem is illustrated in Figure 5. As can be seen, each processor is attached with some on-chip memory of 16 KB BlockRAM for storing instruction and data. Each processor communicates with its neighboring processors using some communication FIFOs called fast simplex link (FSL) links, which are 32-bit wide and we can access them through some registers of the MicroBlaze processor. Thus, each node in the cone network is in fact a simple MicroBlaze processor system.

To enable a processor to communicate with other processors apart from its neighbors, we need to develop a communication protocol. First of all, each processor is identified by a two-tuple $(x, y)$, where $x$ is the layer number and $y$ is the node number in the layer. Figure 5 gives an example on how to label the 15 processors.

Afterwards, we have to develop a packet format according to which data transferred from one processor node to another is encapsulated. Figure 6 shows the detail of the packet format.

**Figure 6. Packet format for data communication between processors.**

Finally, a routing scheme is required. By adopting the proposed cone network, the routing method is trivial. Specifically, when a node receives a packet destined to a lower (higher) layer, it then forwards the packet to its parent (left/right child); when a node receives a packet destined to other node of the same layer, it then forwards the packet to its left or right neighbor according to the destination’s node number.

To be more specific, Algorithm 1 details the routing steps in the cone network. We give a numerical example as follows. Suppose node $(1, 1)$ needs to route a packet destined to node $(3, 6)$, then it has to decide whether to forward the packet to its left child or right child. We can make this decision by considering the subtree rooted at node $(1, 1)$ and regarding the corresponding nodes in Level 3 (i.e., nodes $(3, 4)$ to $(3, 7)$) as leaf nodes of the subtree. Afterwards, we can make the packet forwarding decision by determining whether the destination node is within the first half of the leaf nodes or the second half. Specifically, according to Algorithm 1, the smallest node number of the node in the second half is:

$$\text{leftmost node + half leaves} = 1 \times 2^2 + 2^1 = 6.$$

Thus, node $(1, 1)$ will route the packet to its right child.

**4.2. System Implementation**

We implement the proposed system using Xilinx Embedded Development Kit (EDK). To construct the system, we need to create a Microprocessor Hardware Specification (MHS) file specifying how the components (i.e., MicroBlaze processors, system buses, BlockRAMs, FSL links, etc.) are connected. For our system, the MHS file counts nearly 2000 lines. Thus, to automate this tedious and error-prone

---

\(^1\)MicroBlaze is a 32-bit soft processor core optimized for Xilinx FPGA implementation.
Algorithm 1 Packet routing for the proposed cone network.

ConeRouting(src, dst)
1: MASK ← 1111111b/* 7-bit binary mask */
2: y1 ← src & MASK /* node no. of source */
3: x1 ← src >> 7 /* layer no. of source */
4: y2 ← dst & MASK /* node no. of destination */
5: x2 ← dst >> 7 /* layer no. of destination */
6: if (x1 > x2) then
7: return NODE_PARENT
8: else if (x1 == x2) then
9: d ← y2 - y1
10: if ((d > 0 and d < 2x1-1) or
    (d < 0 and d <= -2x1-1)) then
11: return NODE_RIGHT_NEIGHBOR
12: else
13: return NODE_LEFT_NEIGHBOR
14: end if
15: else
16: leftmost_node ← y1 × 2x2-x1
17: half_leaves ← 2x2-x1-1
18: if (y2 < leftmost_node + half_leaves) then
19: return NODE_LEFT_CHILD
20: else
21: return NODE_RIGHT_CHILD
22: end if
23: end if

Figure 5. Structure of a multiprocessor implementation of 15 processor cores.

MHS file creation process, we have developed a Java application to create the file automatically. Figure 7 shows a code segment of the Java application. This Java application eases the effort required for investigating different system parameters, such as the communication network design, number of processors, etc. In fact, this application can be part of a design automation flow [8] for mapping tasks in an application onto various computation cores. In our future work, we plan to develop a GUI interface for this Java application and then release it for public use.

MBSystem mb_0_0 = new MBSystem("MB_0_0");
MBSystem mb_1_0 = new MBSystem("MB_1_0");
MBSystem mb_1_1 = new MBSystem("MB_1_1");
.....
mb_0_0.systemConnect(null, null, mb_1_0, mb_1_1);
mb_1_0.systemConnect(null, mb_1_1, mb_2_0, mb_2_1);
mb_1_0.systemParentConnect(mb_0_0, SystemConstant.NODE_LEFT_CHILD);
mb_1_1.systemConnect(mb_1_0, null, mb_2_2, mb_2_3);
mb_1_1.systemParentConnect(mb_1_0, null, SystemConstant.NODE_RIGHT_CHILD);
.....

Figure 7. Java code segment for creating the MHS file.

After the MHS file is created, we can specify the software program codes for the processors to execute. Specifically, the codes will be compiled and then stored in the

\[^{2}\text{The software will be released at: http://www.eee.hku.hk/~tokwok/JMP/}\]
BlockRAM of each processor. Table 1 shows the major resource utilization of implementing the 15-core multiprocessor subsystem on the Xilinx Virtex-II Pro XC2VP50 FPGA.

In Table 1, \textit{RAMB16}\textsubscript{s} are used for implementing BlockRAMs, while \textit{MULT18X18}\textsubscript{s} and \textit{SLICE}\textsubscript{s} are used for implementing other digital circuits of the multiprocessor subsystem. As can be seen, nearly half of the device resources are used for implementing the subsystem, which means that we can use another half for implementing accelerator cores.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
\textbf{Resource Type} & \textbf{Utilization} \\
\hline
MULT18X18 & 45 out of 232 (19\%) \\
RAMB16 & 120 out of 232 (51\%) \\
SLICE & 11172 out of 23616 (47\%) \\
\hline
\end{tabular}
\caption{Major resource utilization of implementing the 15-core multiprocessor subsystem.}
\end{table}

4.3. Software Control

In our current design, there are three entities in the communication network infrastructure:

1. **Sender.** This is the originator of a data transmission session.

2. **Receiver.** This is the terminator of a data transmission session.

3. **Forwarder.** This is for routing packets between the sender and receiver of a data transmission session. A store-and-forward data transmission paradigm is used.

Each processor node can take on one or more roles of the three entities. Figure 8 shows the function prototypes of the three entities. Specifically, for \textit{sender()} and \textit{receiver()}, \textit{node\_list} specifies a list of processor nodes to send and receive packets, respectively. For \textit{forwarder()}, \textit{neighbor\_list} is a list of neighbors for which the processor node will forward packets for them. On the other hand, \textit{fsl\_state} stores the states (“idle” or “busy”) of the five outgoing FSL links of the processor node. When the processor node sends sufficient number of packets to one of its outgoing FSL link, the state of the link will be set as “busy”. Then if an acknowledgement packet is received from the corresponding incoming FSL link, which means that a packet has been received from the outgoing FSL link by another processor node, the state of the outgoing FSL link will be set as “idle”.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{sender_receiver.png}
\caption{Function prototypes of the three communication entities.}
\end{figure}

Suppose that all the nodes in Level 3 want to send packets to node (0, 0), then Figure 9 illustrates how the 15 processor nodes should be configured according to the three entities mentioned above.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{level_3.png}
\caption{A data transmission scenario.}
\end{figure}

5. Performance Evaluation

In this section, we present the performance evaluation of the proposed system. The three main aspects of evaluation are as follows:

1. **Efficiency of the Communication Network.** We study the performance of the communication network when different number of hops are involved in a data transmission, and when there are different concurrent data transmissions.
2. **Application Performance.** The performance of a data encryption application is studied.

3. **Power Consumption.** We study the power consumption of the data encryption application when different number of processor cores and accelerator cores are used.

The objective of the performance evaluation is to study the scalability of the proposed reconfigurable heterogeneous multi-core system (e.g., the worthiness of using more cores) and to investigate any potential enhancements to the system.

5.1. Efficiency of the Communication Network

First of all, we study the efficiency of the proposed communication network when different number of hops are involved in a data transmission. Figure 10 shows the throughput achieved when different payload sizes are used to transfer a data stream of 8MB. From the graph, we can see that the throughput drops sharply when two or more hops are involved in the data transmission. The reason is that forwarders are involved. Recall that a store-and-forward transfer paradigm is used in forwarders, which incurs significant communication overhead. When there is only one hop in the data transmission, a larger payload size gives a larger throughput because there is less packet handling overhead. In the case where two or more hops are involved, a smaller payload size gives a larger throughput because the data transmission can be much pipelined when the payload size is smaller.

Figure 11 shows the throughput achieved when there are different number of concurrent transmissions. We use the data transmission scenario as depicted in Figure 9, and a 16-byte payload size is used. Specifically, we vary the number of senders with each sender transmitting a data stream of 8MB to the receiver (i.e., node \((0,0)\)). Then we measure the throughput of the \((3,0)-(0,0)\) transmission pair. As illustrated in Figure 11, we can see that the throughput achieved is halved when the number of concurrent transmissions is doubled.

From the above results, it can be concluded that the packet transmission process incurs a significant overhead when it is performed using the processors in the system. Therefore, the packet transmission process should be offloaded to hardware.

5.2. Application Performance

We now study the performance of a data encryption application. Specifically, the DES encryption standard is used, and a data stream of 800KB is to be encrypted. We compare the performance when the processor cores and accelerator cores are used together (hardware approach), and when
only the processor cores are used (software approach). In this data encryption application, the processor cores are executing at 100MHz while the accelerator cores are executing at 33MHz. For the software approach, the encryption process is distributed among the processor cores. For the hardware approach, hardware DES encryption engines are implemented in reconfigurable modules as the DES accelerator cores [9]. In this experiment, only two DES accelerator cores are implemented (due to the complication of the design flow [27] when more reconfigurable modules are implemented) and the two cores are attached via some FSL links to processor nodes (0,0) and (1,0), respectively.

Figure 12 shows the execution time used in different approaches. As expected, the hardware approach can have significant performance gain over the software approach. As shown in Figure 12, despite that adding one more DES accelerator core can result in some performance improvement (i.e., a 20% reduction in time when compared to the case of only one DES accelerator core), the degree of improvement is less than that of the software approach when one more processor core is used. The reason is that there is significant communication overhead for node (0,0) to send 400KB data to node (1,0) and then read back the encrypted data. This also suggests implementing the communication protocol in hardware in order to further enhance the performance of applications.

5.3. Power Consumption

To study the power consumption of different cores during execution, in addition to the two DES accelerator cores, we only implement eight processor cores (i.e., nodes (3,1) to (3,7) are not implemented on the FPGA). Furthermore, in the software approach, the processor cores are busy executing the DES encryption algorithm without data transfer with other processor cores. On the contrary, in the hardware approach, the processor cores merely send data to the DES accelerator cores for encryption and then read back the encrypted data (while the rest of the processor cores in the system are “stalled” at a specific state). We measure the power consumption by taking current and voltage readings of the power supply of the FPGA board.

Figure 13 shows the power consumption of using different approaches for data encryption. As can be seen, besides computation efficiency, the hardware approach is more power-efficient than the software approach. By using the above experiments, we demonstrate that by adopting a suitably cooperative use of heterogeneous cores, e.g., the accelerator and processor cores, it can bring dramatic system efficiency in terms of computation and power consumption.
6. Related Work

In [25], Williams et al. proposed a reconfigurable cluster-on-chip architecture based on FPGAs. However, a processor core is directly connected to every other processor core in the system, which is simply not scalable. Moreover, the system architecture does not consider a reconfigurable architecture using the rest of the FPGA fabric to accelerate applications. On the contrary, under our proposed system, in addition to conventional processor cores, we introduce dynamically reconfigurable accelerator cores to boost the performance of applications. More importantly, we have proposed a scalable communication network for interconnecting different cores in the system.

Patel et al. [17] proposed a multiprocessor system based on FPGAs. They designed a scalable architecture utilizing multiple FPGA chips/boards. A communication interface and an MPI message-passing standard were implemented for developing parallel programs. On the other hand, Saint-Jean et al. [20] proposed a multiprocessor SoC architecture for embedded systems. However, unlike our proposed work, the architecture consists of a set of homogeneous RISC processors. They adopted the traditional multithreaded programming model.

The Berkeley RAMP project [23] uses FPGAs as an accelerator for multiprocessor research (i.e., FPGAs act as a multiprocessor emulator). Using FPGAs as a system prototype, the XMT processor [24] is a 64-core processor for executing parallel random access machine/model (PRAM) algorithms. Different from the RAMP project, we are interested in the design of a platform for accelerating practical applications. Similar to the XMT processor and the work in [25], we would like to build an FPGA-based multiprocessor platform, but we also take advantage of the dynamically reconfigurable property of FPGAs.

As for connecting various processing elements on a chip, many schemes can be applied, such as bus-based communication, circuit switching, and network-on-chip (NoC) [4]. In bus-based communication scheme such as [16], the processing elements communicate via a common bus. Despite that a common bus can reduce the amount of resources required in the system, it is not scalable as it limits the number of concurrent communications. Furthermore, additional delay is caused by bus arbitration. On the other hand, using circuit switching in reconfigurable devices presents some drawbacks such as long communication delay [4]. A more promising way to on-chip communication is to adopt network-on-chip, where the communication among processing elements is achieved by routing packets [5].

Although a plethora of work has been done on network-on-chip design [3, 15], our proposed NoC is compact in the sense that it shares the properties of tree and mesh networks, and allows the implementation of a simple routing algorithm.

7. Conclusions and Future Work

In this paper, we have presented the design and implementation of a reconfigurable heterogeneous multi-core system using FPGAs. Specifically, we have proposed a cone communication network for interconnecting different cores. We have evaluated the proposed system in various aspects—efficiency of communication network, application performance, and power consumption. It is demonstrated that a suitably cooperative use of accelerator and processor cores can bring dramatic system efficiency in terms of computation and power consumption. However, we still need a fast implementation of the communication protocol in order to further enhance the efficiency of the system.

As part of our on-going work, we are now implementing the proposed cone network and its related communication protocol in hardware. Furthermore, we plan to develop a GUI interface for the Java application which eases the construction of a multi-core system and then release it for public use. In addition to this, for our future work we would also like to evaluate the performance of our system using other parallel applications, such as MPEG4 encoding/decoding.

References


Biographies

Tyrone T.-O. Kwok is currently pursuing his Ph.D. degree in the Electrical and Electronic Engineering Department at the University of Hong Kong. He received his B.Eng. degree in computer engineering and M.Phil. degree in electrical and electronic engineering both from the University of Hong Kong in 2003 and 2005, respectively. His research interests include FPGA-based reconfigurable computing, parallel computer architecture, and wireless network security.

Yu-Kwong Kwok (Ricky) is an associate professor in the Electrical and Computer Engineering Department at the
Colorado State University. Prior to joining CSU, he was an associate professor at the University of Hong Kong. He received his Ph.D. degree in computer science from the Hong Kong University of Science and Technology in 1997. In the areas of distributed systems and wireless networking, he has co-authored over 170 technical papers and two textbooks. Ricky currently serves on the Editorial Board of the Journal of Parallel and Distributed Computing for the subject area Peer-to-Peer (P2P) Computing, and the International Journal of Sensor Networks. His current research endeavors are mainly related to game theoretic security and incentive issues for wireless systems, and resource management for dynamically reconfigurable chip multiprocessor systems. Ricky received the Outstanding Young Researcher Award from the University of Hong Kong in November 2004.