PAPER

A Low-Cost and Energy-Efficient Multiprocessor System-on-Chip for UWB MAC Layer

Hao XIAO[†a)], ***Nonmember***, **Tsuyoshi ISSHIKI**[†], ***Member***, **Arif Ullah KHAN**[†], ***Nonmember***, **Dongju LI**[†], ***Member***, **Hiroaki KUNIEDA**[†], ***Fellow***, **Yuko NAKASE**[††], ***and*** **Sadahiro KIMURA**[††], ***Nonmembers***

SUMMARY Ultra-wideband (UWB) technology has attracted much attention recently due to its high data rate and low emission power. Its media access control (MAC) protocol, WiMedia MAC, promises a lot of facilities for high-speed and high-quality wireless communication. However, these benefits in turn involve a large amount of computational load, which challenges the traditional uniprocessor architecture based implementation method to provide the required performance. However, the constrained cost and power budget, on the other hand, makes using commercial multiprocessor solutions unrealistic. In this paper, a low-cost and energy-efficient multiprocessor system-on-chip (MPSoC), which tackles at once the aspects of system design, software migration and hardware architecture, is presented for the implementation of UWB MAC layer. Experimental results show that the proposed MPSoC, based on four simple RISC processors and shared-memory infrastructure, achieves up to 45% performance improvement and 65% power saving, but takes 15% less area than the uniprocessor implementation.

key words: *multiprocessor system-on-chip (MPSoC), ultra-wideband (UWB), medium access control (MAC), shared-memory*

1. Introduction

The ever-growing demand for anytime and anywhere wireless connectivity has driven the emergence of Wireless Personal Area Networks (WPANs) due to their flexibility and increasing capability. Among various WPAN technologies, WiMedia ultra-wideband (UWB) is considered as the best for high-speed wireless applications, due to its high data rate, large bandwidth and low emission power [1], [2]. Besides the favorable physical layer characteristics, WiMedia UWB also has some attractive features in its Media Access Control (MAC) layer. It allows both reservation and contention-based medium access with Quality-of-Service guarantee. Moreover, no existing infrastructure is required in the WiMedia UWB network. All these features make UWB a primary candidate for WPAN applications, such as multimedia [3], [4], health care services and so on [5].

However, the facilities that WiMedia MAC promises do not come for free. From the perspective of implementation, both the zero-infrastructure network and the high data rate require a significant amount of computational effort and extremely short processing delay. It is therefore a challenge to design an efficient architecture for WiMedia MAC in embedded systems because of the stringent requirements of cost, power and flexibility. Previous works from both academia and industry have proposed implementations for WiMedia MAC [6]–[9]. In [6]–[8], implementations of WiMedia MAC using conventional single-core architecture are proposed. However, since we also developed similar uniprocessor architecture before this work to speed time-to-market, the power consumption of this solution is too high for portable devices. In [9], an FPGA based implementation is proposed. The primary drawback of this solution is that it cannot provide the required flexibility to upgrade or adapt the system to different scenarios, and thus, programmable architectures are more desirable. Unlike these previous works, this paper explores MPSoC architectures for WiMedia MAC and evaluates their impact on performance, area and energy efficiency with respect to uniprocessor architecture. To the best of our knowledge, no such work has been done for WiMedia MAC so far.

It is well-known that parallel processing on multiprocessor is an attractive technology to achieve high-performance. However, most of the existing commercial multicore platforms [10], [11] target high-end general purpose applications and usually require much higher cost than a conventional single-core processor, such as the classic ARM9 processor [12]. Thus, for our application-specific and cost-constrained system, these commercial solutions are obviously too expensive. Therefore, this work motives to develop a low-cost and energy-efficient multiprocessor system-on-chip (MPSoC) for WiMedia MAC. Besides the required high-performance, the target MPSoC should also feature competitive area and power even with respect to the uniprocessor SoC. The proposed MPSoC consists of four processor nodes, which are organized by a shared-memory infrastructure. Instead of a general-purpose processor, it employs a simple RISC processing element (PE) for each node to save area. In order to migrate from single-core to multicore scenario, we maintain the legacy multitask software of our previous uniprocessor system and let each individual task execute on a dedicated node in the multiprocessor system. Moreover, dedicated hardware engines are developed to support inter-processor synchronization and communication. Then, by executing a unique thread on each node, the processors can get rid of the multitasking overhead, and hence, achieve performance improvement.

Implementation and evaluation of the proposed MP-

Manuscript received September 21, 2011.
Manuscript revised March 8, 2012.

[†]The authors are with the Department of Communications and Integrated Systems, Tokyo Institute of Technology, Tokyo, 152–8552 Japan.

[††]The authors are with the R&D Group, RICOH Co. LTD., Yokohama-shi, 224–0035 Japan.

a) E-mail: xiaohao@vlsi.ss.titech.ac.jp

DOI: 10.1587/transinf.E95.D.2027

SoC have been carried out at system level by using commercial electronic system-level (ESL) tools [13], [14]. Finally, area and power estimation is carried out by using post-synthesis results. Experimental results show that the proposed MPSoC achieves up to 45% performance improvement and up to 65% power saving, but costs 15% less area when compared to our previous uniprocessor design.

The rest of the paper is organized as follows. In Sect. 2, we introduce some related works and summarize our contributions, which is followed by an introduction of our MPSoC design methodology in Sect. 3. In Sect. 4, we explain the MPSoC hardware architecture. In Sect. 5, the implementation of UWB MAC and the experimental results are presented. Finally, conclusions are drawn in Sect. 6.

2. Related Work and Our Contributions

The emergence of WiMedia MAC has attracted a lot of attention recently and quite a few research works on its implementation have been carried out. In [6], the authors proposed a hardware/software (HW/SW) co-design approach for WiMedia MAC to shorten the time-to-market and achieve flexibility for future upgrade. They discussed some of the system architectural design considerations, especially the HW/SW partition. However, from the perspective of architecture, they still use the conventional embedded design approach, which is based on a single general purpose processor. Moreover, they didn't evaluate the performance in terms of power consumption and area. In [7], an LSI implementation of WiMedia MAC is proposed, which is still based on the similar uniprocessor architecture. In [8], the authors focus on the throughput issues of WiMedia MAC and propose an optimized architecture. However, their exploration is still limited within the general purpose processor based uniprocessor architecture. Before this work, we also developed a similar SoC by employing such uniprocessor architecture. Because of the rich support of programming models, such as RTOS, and debugger tools, this approach may speed time-to-market. However, this design approach in turn leads to a significant amount of performance overhead, which further results in high power dissipation.

In [9], a full hardware implementation based on FPGA is proposed for WiMedia MAC. The primary drawback of this solution is that it cannot provide the required flexibility to upgrade or adapt the system to different scenarios, and thus, programmable architectures are more desirable.

Unlike these prior works on UWB MAC, this work aims to achieve aggressive performance and ultra low power consumption by exploring MPSoC architectures.

Additionally, with the emergence of MPSoC, using HW support for the operating system and its communication primitives is always a hot topic. Due to the inherent characteristics of multiprocessors, some RTOS features, such as synchronization, must rely on hardware support for some sort of atomic *read-modified-write* operation [15]. On the other hand, implementing some RTOS features requiring greater CPU workload in hardware can decrease the system overhead [16]. In [17], a hardware module, the synchronization-operation buffer, is proposed to queue and manage the inter-processor synchronization, *e.g.*, locks and barriers. In [18], A. C. Nacul et al. propose a HW-RTOS for speeding up the scheduling and data handling of the RTOS. Moreover, operating systems completely implemented in the hardware have also been proposed in [19] and [20].

Unlike previous works on HW RTOS, this work focuses on a fixed application, UWB MAC, and intends to improve its performance by removing the RTOS overhead. By distributing the SW tasks to dedicated processors, task scheduling and switching is no longer needed. Furthermore, hardware modules are proposed to support inter-processor communication and synchronization.

The contributions of this paper are summarized as follows.

- We demonstrate that, in uniprocessor architecture, there is significant overhead when using RTOS. In order to deal with this problem, an MPSoC design methodology is proposed for WiMedia MAC. And a complete comparison between the proposed MPSoC and the uniprocessor design is carried out from the aspects of performance, area and power consumption. To the best of our knowledge, no such work has been done for WiMedia MAC so far.
- In our proposed MPSoC, each individual task executes on a dedicated processing node. Hardware engines are developed to support inter-processor synchronization and communication. Thus, processors can get rid of RTOS, which leads to significant performance improvements (up to 45%) with respect to the uniprocessor design
- Because of the distributed workload in MPSoC, we explore the usage of simple RISC PE, instead of general purpose processor. This saves the design cost and also makes the potential for using ASIP possible. Moreover, removing RTOS reduces the required size of instruction and data memory, which further results in area saving. Due to the above contributions, our proposed MPSoC achieves 15% area saving over the uniprocessor design.
- Our proposed MPSoC also achieves significant power saving (up to 65%) with respect to the uniprocessor design, which is due to the following reasons: (i) the overhead reduction makes the MPSoC consume much less energy for program execution; (ii) using MPSoC natively splits the single big memory, which is usually used in the uniprocessor context for instructions and data, into multiple small memories, which features much less energy per access than the bigger one.

3. MPSoC Design Methodology

In this section, we first introduce the WiMedia MAC protocol and the system architecture. Then, the basic idea of our MPSoC design methodology is presented.

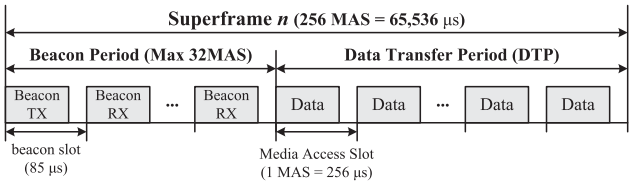

Fig. 1 Superframe format of WiMedia MAC.

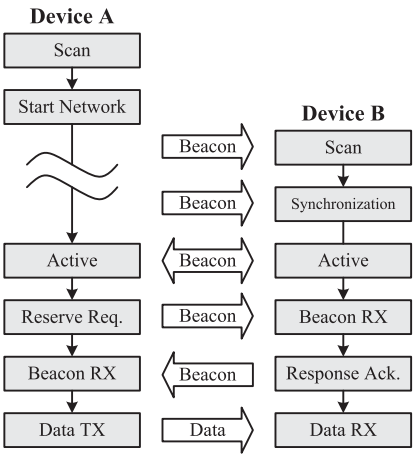

Fig. 2 WiMedia MAC communication flow.

3.1 WiMedia MAC Protocol

In communication systems, MAC layer is mainly used to provide addressing and channel access control mechanisms for terminals within the network. As in other MAC protocols, the medium time of WiMedia MAC is divided into superframes, which describe periodic intervals used to coordinate operation between devices. As shown in Fig. 1, one superframe consists of 256 medium access slots (MASs), where each MAS duration is 256 μs. Each superframe starts with a beacon period (BP) which is followed by a data transfer period (DTP).

Since there is no central coordinator in the network, all the management and control information is exchanged in the form of beacon frames that are transmitted during the BP. As shown in Fig. 2, once a device is turned on, it scans for beacons from other devices. If no beacon is received during the initial scanning, such as device A, it creates its own BP and becomes active. If one or more beacons are received correctly during the initial scanning, such as device B, it synchronizes with the existing network and picks up unoccupied beacon slots to transmit its own beacon frames. Once a device joins a network, it always sends the beacon frame and collects the beacons from its neighbors in the BP.

After the network becomes stable, data communication could be carried out through either Distributed Reservation Protocol (DRP) or Prioritized Channel Access (PCA) protocol. The DRP allows a reserved period of time for transmission during which the reservation owner has exclusive access to the medium. As shown in Fig. 2, the negotiation process starts with the transmitter sending a reservation request in its beacon frame. The request includes the set of MASs that the transmitter intends to reserve for transmission. Upon reception of the request, the receiver analyzes the channel time utilization of its beacon group and responds, using its beacon frame, indicating whether the reservation request is accepted. Once a reservation is successfully negotiated, the reservation is announced in the beacons. Other devices become aware of the reservation by reception of the beacons, and therefore defer access to the medium during the reserved MASs. During the unreserved MASs, frames with different priorities can be transmitted through the contention based PCA protocol. Only the one with the highest priority, which is labeled by the higher layers, can be allowed to access the medium.

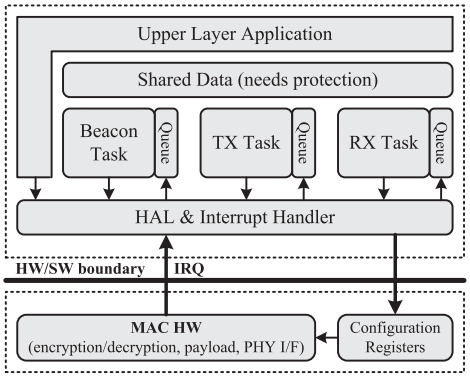

Fig. 3 System architecture of WiMedia MAC.

3.2 System Architecture

We implement the MAC protocol using a hybrid hardware and software (HW/SW) architecture, which is shown in Fig. 3. For flexibility and easy upgrade possible, the MAC functionality is tried to be achieved in SW as far as possible. Only time critical functions, such as data encryption/decryption, payload transfer and PHY layer handshake, are done in HW. A set of configuration registers are provided for the SW to control and manage the HW. On the other hand, the HW triggers the SW processing via interrupt, when any event happens.

According to the MAC protocol, there are three communication scenarios: beacon processing, data frame reception and data frame transmission. Correspondingly, we implement three tasks to deal with these scenarios, including reception (RX) task (receive traffic from wireless medium and notify upper layer), transmission (TX) task (handle the data transfer from upper layer to the wireless medium) and beacon task (receive and transmit beacon frames). Moreover, a fourth task which acts as a upper layer application is designed to control the communication. All these components share the same data resource. Thus, in order to avoid multi-task access, these shared data needs to be protected. Additionally, each task has its own queue to receive the messages from other tasks.

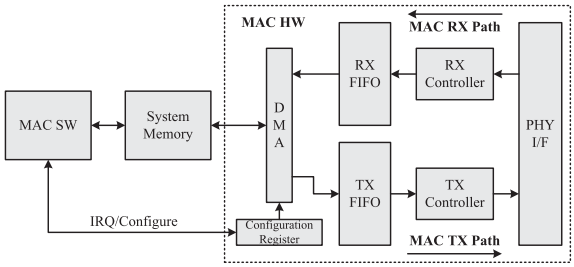

Fig. 4 HW/SW interaction for reception and transmission.

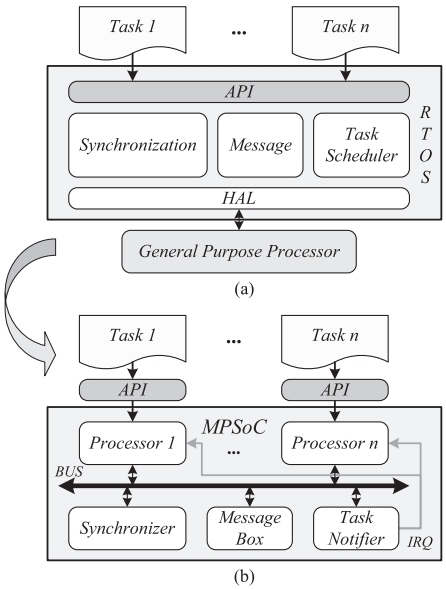

Fig. 5 Migration from single-core to multi-core.

Figure 4 further shows the HW/SW interaction for data frame (or beacon frame) reception and transmission. In case of RX path, the HW first stores the received data in its local FIFO, and then, moves the payload to some memory space which is pre-defined by SW. After getting one complete frame, the HW will generate interrupt to trigger the SW processing. Then, the SW might update the HW through configuration registers and start working on the received data.

In case of TX path, the SW first configures the HW with the pointer of payload, and then, triggers it to start transmitting. Upon startup, the HW copies the data from system memory to its local FIFO, and also, handshakes with the PHY layer to arrange the transmission. After completion of the transmission, HW will also interrupt the SW with the specific event.

3.3 Software Migration

In the uniprocessor scenario, the above mentioned software architecture can be conveniently realized by using RTOS, which is also the most conventional development approach. As shown Fig. 5 (a), most RTOSes offer the capability of intertask message and synchronization. With the RTOS-offered APIs, message and synchronization operations can be implemented by calling the related service routines. Moreover, interrupt handler can be realized by using the RTOS-offered interrupt service routine. With the scheduling of RTOS kernel, the multiple contexts can switch orderly to handle the corresponding event. Then, both application tasks and RTOS execute on a general purpose processor.

However, the drawback of using RTOS is the overhead associated with task scheduling and context switching. In order to handle real-time multitasking, RTOS has to invoke kernel scheduler in each interrupt, synchronization and mailbox operation [21]. And if any task needs to be suspended or resumed, context must be stored and restored respectively. These operations in turn require a large amount of workload and thus lead to a significant overhead [22]. According to our evaluation (reported in Sect. 5), RTOS overhead costs up to 49% workload in the uniprocessor design. These overheads not only shorten the available time for data processing but also pose a great burden to the processor. Hence, in order to achieve real-time processing, the single processor has to execute at a high frequency, which results in high power dissipation.

Therefore, in order to overcome this bottleneck and achieve high-performance, the idea here is to develop MPSoC for MAC SW. In order to minimize the SW design effort and achieve early-stage HW/SW co-simulation, we reuse the legacy multitask software and directly migrate the separated tasks to MPSoC, which is shown in Fig. 5 (b). Each MAC task is assigned to a dedicated processor in the MPSoC system. Since each processor executes a unique thread, multi-task RTOS becomes redundant, and hence, can be removed to relieve the processors' workload. Instead of RTOS, we develop dedicated hardware engines to support the inter-processor synchronization and message-passing. For task scheduling, a first-come-first-served scheduling mechanism is established and interrupt signals are utilized for notification. Then, these hardware features are abstracted as a set of APIs and exposed to programmers.

4. Hardware Architecture

In this section, we first present some considerations for our MPSoC architecture, which will be followed by the overall architecture. Then, the structure of employed hardware engines for inter-processor synchronization and communication are described respectively.

4.1 MPSoC Architecture

The proposed MPSoC, which is based on shared-memory infrastructure, is shown in Fig. 6. It consists of four processors which are connected via a shared bus. Distributed instruction and data memories are attached with each nodes to relieve the shared bus from traffic contention. Because of the usage of on-chip memory, cache is not used to avoid the overhead of cache coherency support. A shared mem-

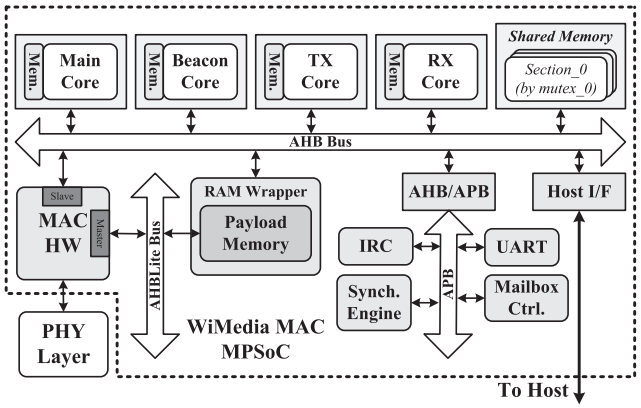

Fig. 6 Proposed MPSoC architecture.

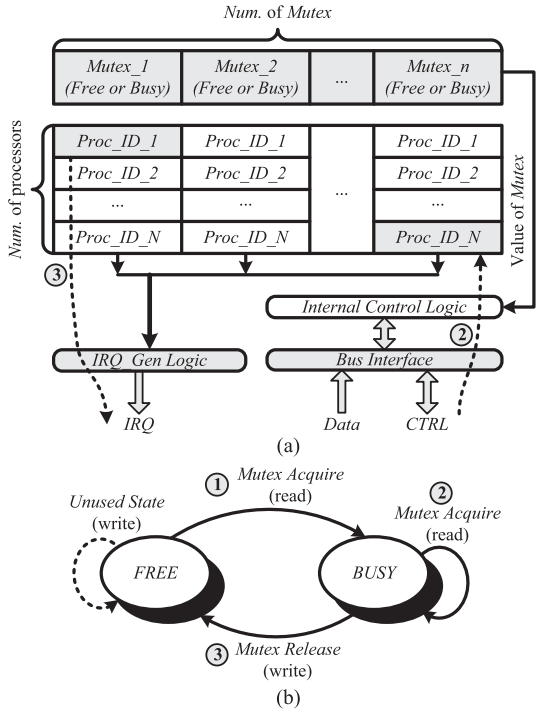

Fig. 7 Architecture of synchronization engine.

ory and a payload memory are attached with the shared bus for parameter and payload storage, respectively. The shared-memory architecture, on the one hand, reduces the latency for frequently parameter and payload access, and on the other hand, saves the on-chip memory space for storing. The MAC HW is also connected with this shared bus and treated as a slave to the processors. In order to have more bandwidth for the data transfer, the MAC HW also occupies a dedicated bus for payload RAM access. Moreover, a set of necessary peripherals are provided to support the application.

In order to synchronize the multiple processors and guarantee the data coherence of the shared memory, we use the *mutex* and *semaphore* mechanism [15] for inter-processor synchronization. According to the data dependencies, the shared memory is divided into several sections, each of which is assigned a dedicated *mutex* to prevent it from multi-accessing. Considering the flexibility and cost-efficiency, we prefer hardware-based implementation [17] and thus develop a synchronization engine to centrally manage the *mutex* and *semaphore* resource. Moreover, a hardware-based mailbox is also developed to support inter-processor message-passing. In the following two subsections, the implementation of the synchronization engine and the mailbox controller will be discussed in detail.

4.2 Synchronization Engine

The synchronization engine, as shown in Fig. 7 (a), is designed to support the *mutex* and *semaphore* operations [15] in our MPSoC. Taking *mutex* for example, there are *n* registers (*n* is the maximum number of supported *mutex*) and each of them is used to store the value of a specified *mutex* (*FREE* or *BUSY*). Correspondingly, each *mutex* has a waiting queue for holding the IDs of waiting processors. This waiting queue is implemented by FIFO which makes the management of processor order easier. The depth of FIFO *N* depends on the total number of processors in the system.

Figure 7 (b) shows the state machine of hardware *mutex*. It has two states, one is called *FREE* and the other is named *BUSY*. Transition 1 means the *mutex* is acquired at its *FREE* state, and thus, its value should be updated from *FREE* to *BUSY*. This is the so called *test-and-set* operation and should be atomic. Since the hardware is connected with a bus, this inherent property guarantees that only one read access is allowed at a time. Thus, even if two acquisitions (or even more) happen at the same time, only one winner can get the *mutex* first, and meanwhile, the hardware logic will set the *mutex* to *BUSY* within the same clock cycle. Then, the loser, who acquires the *mutex* in the following cycle, will always be returned by the updated value. In other words, the atomic *test-and-set* operation is separated into two parts, *test* (or *read*) done by the processor and *set* (or *write*) done by the hardware. They are ensured to finish within the same atomic cycle. If any processor tries to acquire the *mutex* in the *BUSY* state, as shown in transition 2, it should keep waiting until the *mutex* is released by the owner processor. In this case, hardware logic will save the pending processor's ID and notify it through an interrupt when the *mutex* is available for its turn. Thus, during the waiting period, the processor only executes *nop* instruction instead of polling, which doesn't cause any traffic at all. Transition 3 indicates that the processor who occupies the *mutex* should release it by setting it back to *FREE*. Meanwhile, if any processor is waiting for the released *mutex*, as shown in Fig. 7 (a), its ID will be fetched from the head of the waiting queue and an interrupt will be generated accordingly to notify it.

4.3 Mailbox Controller

Mailbox controller, which is shown in Fig. 8, is designed to offer message sending and receiving services for processors.

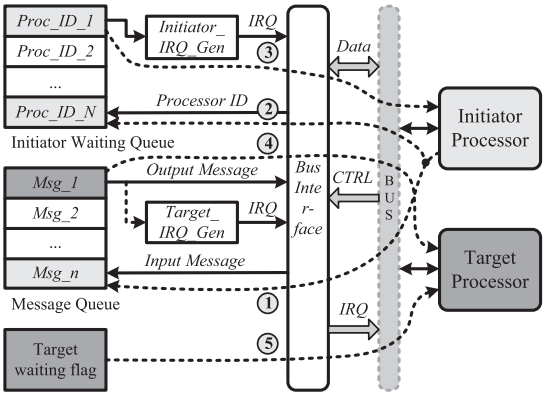

Fig. 8 Architecture of mailbox controller.

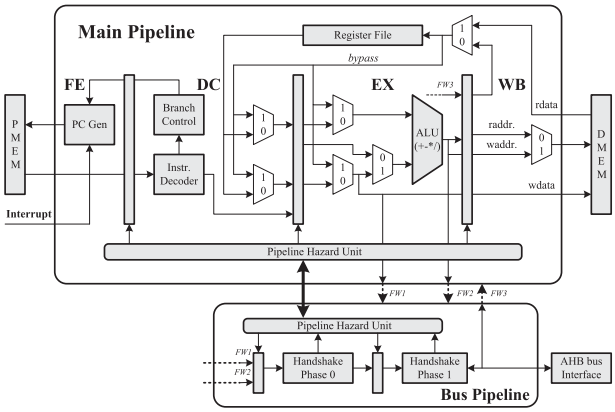

Fig. 9 Block diagram of TCT processor.

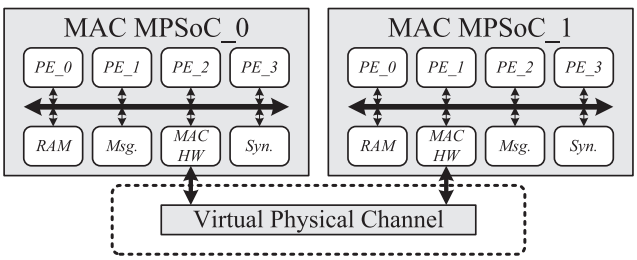

Fig. 10 WiMedia MAC MPSoC test platform.

There are two FIFOs which are used for initiator waiting queue and message queue respectively. The former is used to hold the IDs of waiting initiators who intend to send messages and the latter is used to store the content of messages. Moreover, a target waiting flag is employed to indicate if the target processor, to whom the mailbox belongs, is waiting for a message or not.

In order to send a message, the initiator first checks the availability of the target message queue. If there is enough space, it directly puts the message at the tail of the message queue, which is represented by transition 1. Whereas, if the message queue is full, the initiator has to keep waiting and its ID is pushed into the initiator waiting queue, which is illustrated by transition 2. Until any message is consumed by the target processor and available space appears in the message queue, an interrupt is generated to notify the waiting initiator processor in order. This process is shown in transition 3. Then, the initiator, who has just been woken up by the interrupt, starts to re-send its pending message, which is the same as transition 1.

On the other hand, the target processor that intends to receive a message first checks the availability of its message queue. If any message is available, it directly fetches messages at the head of the message queue, which is shown in transition 4. On the contrary, if the message queue is empty, it is suspended until the availability of a message is signaled by an interrupt, which is described in transition 5, and then retries its pending receive operation as in transition 4.

5. Implementation and Results

In this section, we first explain the simulation setup of the proposed MPSoC. Then, comparisons between our previous uniprocessor SoC and the proposed MPSoC are carried out respectively from the aspects of performance, area and power consumption.

5.1 Processing Element

In the proposed MPSoC, we adopt our in-house TCT processor for each processing node. As shown in Fig. 9, the TCT processor is a 32-bit RISC core with a 4-stage pipeline. It was first proposed in our previous work [23]. However, according to the architecture of this work, we improve this processor by adding the AHB interface and removing the un-used communication module. The processor is modeled by language for instruction-set architectures (LISA) [24], which can be compiled by Synopsys Processor Designer [14] to generate cycle-accurate SystemC model, instruction set simulator (ISS) and register transfer level (RTL) code.

5.2 Simulation Setup

The proposed MPSoC is modeled at system level using commercial ESL tool [13], which provides support for platform creation, simulation and evaluation. In order to verify and evaluate the system, two WiMedia MAC MPSoCs, one as receiver and the other as transmitter, are connected via a virtual physical channel, which is shown in Fig. 10. The virtual channel is modeled by SystemC and used for simulating the behavior of the UWB physical layer and channel.

Table 1 shows the detailed architecture configurations. We explore two multi-core architectures, one is the multi-ARM, which consists of four ARM946E-S, and the other is multi-TCT, which employs our TCT processor. As explained in Sect. 4.1, both multi-ARM and multi-TCT architectures use tightly-coupled memories (TCMs) for local data and instructions. Since they execute the same SW, these two architectures require similar size of memory for data and instruction, which are, respectively, 32 KB and 204 KB in all.

Table 1 Architecture configurations for exploration.

Parameter	Single-ARM		Multi-core	
	TCM	Cache	Multi-ARM	Multi-TCT
Num. of PE	ARM×1	ARM×1	ARM×4	TCT×4
Cache Size(KB)	*None*	8/8	*None*	*None*
Data Mem.(KB)	78	*None*	20/4/4/4*	20/4/4/4*
Inst. Mem.(KB)	230	*None*	72/82/42/8*	72/82/42/8*
Main Mem.	256 KB	564 KB	256 KB	256 KB
Total Mem.	564 KB	564 KB	492 KB	492 KB
HW Support	*None*	*None*	Synch., Mailbox	Synch., Mailbox
Inter-connection	AMBA	AMBA	AMBA	AMBA

**Note*: memory size of each processor

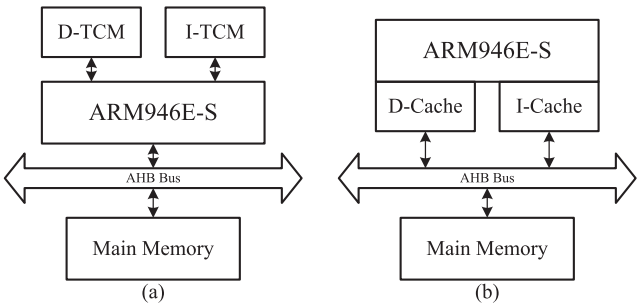

Fig. 11 Memory architecture of single-ARM platform.

Moveover, the size of main memory, for MAC parameter and payload, are 256 KB. Thus, the total memory used in multi-core platform is 492 KB. Due to the limited storage requirement, all the memories are on-chip. Thus, cache is not used to avoid the overhead of cache coherency support. Additionally, the synchronization and mailbox engines are modeled by both transaction-level SystemC and RTL code.

For comparison, we also model the single-core system using only one ARM946E-S processor. As shown in Table 1, we evaluate two conventional single-core architectures, whose memory hierarchies are further shown in Fig. 11. In Fig. 11 (a), TCMs are used for local data and instructions to boost overall performance. In the other architecture shown in Fig. 11 (b), all data and instructions are stored in the main memory, which is accessed through the AHB bus. Then, caches are employed to reduce the access delay. TOPPERS/JSP Kernel [26], a popular RTOS that conforms to the μITRON4.0 specification [21], is adopted in single-core architectures for multitasking. Due to the usage of RTOS, more memory space is required. As shown in Table 1, the total on-chip storage of single-core platform is 564 KB, which is bigger than that of multi-core platform.

5.3 Performance Evaluation

This subsection evaluates the performance of the proposed MPSoC and the uniprocessor architectures. The measured communication scenario consists of two devices, one sends at a fixed data rate and the other receives in a periodic manner. The medium channel is accessed through the DRP protocol [2] with a frame payload of 1024 bytes. According to [2] and [27], a two-device network has a maximum of 254 MASs available for data transmission, where each MAS can further contain a maximum of 8 frames whose payload is 1024 bytes (using burst mode at PHY rate of 480 Mbps). Table 2 shows the relationship between the MAC throughput and the number of frames per MAS. With different inter-frame intervals, the MAC throughput can vary from 0 to 254 Mbps.

Table 2 MAC throughput of a two-device network using DRP protocol.

MAS/SF	254	254	254	254	254	254	254	254
Frame/MAS	1	2	3	4	5	6	7	8
Payload(byte)	1024	1024	1024	1024	1024	1024	1024	1024
Throughput (Mbps)	31.8	63.5	95.3	127	158.8	190.5	222.3	254

For workload profiling, we adopt the approach mentioned in [25], which utilizes a hardware counter (modeled by SystemC) driven by the same clock source of the processors. In order to get the execution cycle of a specific function, cycle stamps can be obtained by reading the counter at the function entrance and return point respectively.

Table 3 reports the execution cycle of some fundamental MAC operations. We measured four architectures, including single-ARM with TCM, single-ARM with cache, multi-ARM and multi-TCT. The workload of each architecture is divided into two components, one is the MAC SW, which is necessary for MAC functionality, and the other is the overhead related to multitasking and interrupt processing. Then, the total workload is the sum of MAC SW and overhead. The execution cycles of cached single-ARM, multi-ARM and multi-TCT are normalized with respect to the execution cycles of single-ARM with TCM.

Thus, combining the data in Table 2 and the workload reported in Table 3, we further calculate the workload of a system maintaining a two-device network under various data throughput using the formulas below:

$$W_{Total} = W_{bcn} + W_{data\_TX/RX}, \quad (1)$$

where W_{Total} is the total workload of a device, W_{bcn} is the workload for handling beacon frames and $W_{data\_TX/RX}$ is workload for transmitting (or receiving) data frames. Since the measured network consists of two devices, each device only needs to receive one beacon from its neighbor. Thus, W_{bcn} can be expressed as:

$$W_{bcn} = W_{bcn\_TX} + W_{bcn\_RX} + W_{bcn\_processing}, \quad (2)$$

where W_{bcn_TX}, W_{bcn_RX} and $W_{bcn\_processing}$ are reported in Table 3. $W_{data\_TX/RX}$, on the other hand, depends on the MAC throughput and can be defined as:

$$W_{data\_TX/RX} = W_{frame} \times N_{MAS} \times N_{frame\_per\_MAS}, \quad (3)$$

where W_{frame} is the execution cycle for TX (or RX) one data frame, N_{MAS} is the number of MAS used for data TX (or RX) and $N_{frame\_per\_MAS}$ is the number of frames within each MAS. Figure 12 shows the workload breakdown of the RX device, which is calculated by using Eq. (1)-(3) at three

Table 3 Execution cycle for WiMedia MAC operations.

Operation	ARM TCM(#cycle)		ARM Cache (#cycle)		Multi-ARM (#cycle)		Multi-TCT (#cycle)	
	MAC SW	Overhead	MAC SW	Overhead	MAC SW	Overhead	MAC SW	Overhead
Beacon frame TX	787	3,931	987	4,207	1,374	754	1,365	764
	4,718 (100%)		**5,194 (110.1%)**		**2,128 (45.1%)**		**2,129 (45.1%)**	
Beacon frame RX	4,095	4,531	5,302	5,097	4,672	784	4,287	7,94
	8,626 (100%)		**10,399 (120.6%)**		**5,456 (63.3%)**		**5,081 (58.9%)**	
Beacon Processing	24,816	26,390	22,678	30,498	27,115	3,526	34,149	3,401
	51,206 (100%)		**53,176 (103.8%)**		**30,641 (59.8%)**		**37,550 (73.3%)**	
TX one data frame	7,398	5,965	7,530	7,913	6,812	898	6,711	878
	13,363 (100%)		**15,443 (115.6%)**		**7,710 (57.7%)**		**7,589 (56.8%)**	
RX one data frame	8,739	9,272	10,061	11,317	9,027	1,060	8,847	970
	18,011 (100%)		**21,378 (118.7%)**		**10,087 (56.0%)**		**9,817 (54.5%)**	

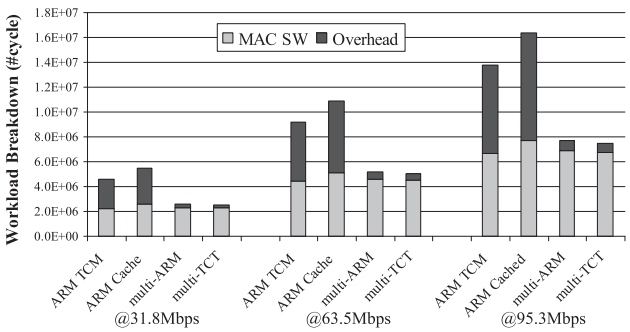

Fig. 12 Workload breakdown of RX device at throughput 31.8 Mbps, 63.5 Mbps and 95.3 Mbps.

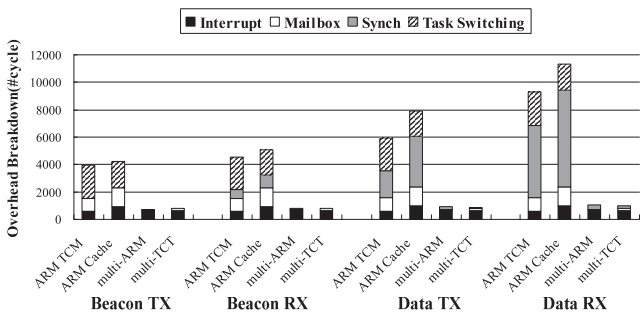

Fig. 13 Software overhead breakdown of four MAC operations.

given throughput of 31.8 Mbps, 63.5 Mbps and 95.3 Mbps.

As shown in Table 3, the overhead, taking RX data frame for example, is reduced from 9272 cycles in the single-ARM with TCM to 1060 cycles in the multi-ARM platform (about 88% reduction). The overhead savings of the other operations are also more than 80%. On the other hand, as mentioned in Sect. 3.1, the software for MAC functionality is maintained during the migration from single-core to multi-core. Thus, the workload of this component should be essentially the same for single-ARM with TCM and multi-ARM. As shown in Table 3, the results confirm this expectation that these two architectures cost similar execution cycles for MAC SW. Finally, as shown in Fig. 12, due to the significant reduction of overhead, multi-ARM platform achieves about 44% workload saving over the single-ARM with TCM. While, in the cached single-ARM architecture, the performance degrades about 17%, which is due to the possibility of cache miss.

Comparing multi-ARM and multi-TCT, since they execute the same software, their workload, regardless of whether it is from the MAC SW or overhead, should be the same. However, due to the different instruction set and compiler (the TCT processor is based on our in-house defined instruction set and compiler [23]), the execution cycles of them are different. The results show that the use of simple processors does not lead to dramatic performance degradation; on the contrary, its performance is even better than that of the general purpose processor in some cases. Finally, multi-TCT platform obtains about 45% workload saving over the single-ARM with TCM, which is similar with multi-ARM.

5.4 Overhead Evaluation

In the previous subsection, we demonstrate that the performance improvement of MPSoC mainly comes from the reduction of overhead. As shown in Fig. 12, the RTOS overhead in single-ARM TCM architecture accounts for approximately 51% of the total workload. However, in the multi-core architectures, the proportion of the workload spent on the overhead diminishes to approximately 11%. This result also confirms the argumentation of another research [22] that using RTOS might result in significant overhead with respect to the useful application.

To examine the overhead reduction in detail, we further break down the software overhead, as shown in Fig. 13, into four categories: *Task Switching* specifies the execution cycles spent on task scheduling and context switching; *Mailbox* and *Synch.* are, respectively, the workload spent for mailbox and synchronization operations; and *Interrupt* denotes the remaining cycles spent on the interrupt handler. As the graph shows, the proposed MPSoCs feature a much smaller overhead than the single-ARM platforms. In more depth, the obtained overhead saving mainly stems from two sources of reduction: (i) overhead savings caused by multi-processor implementation; (ii) acceleration obtained by using hardware engines. As regards the former, using RTOS in the uniprocessor architecture costs a lot of cycles for

Table 4 Area comparison of single-core and multi-core.

Parameter	Single-ARM		Multi-core	
	TCM	Cache	Multi-ARM	Multi-TCT
Technology	90 nm	90 nm	90 nm	90 nm
PE Area[1]	0.303	0.650	0.303×4	0.084×4
HW Area[1]	0	0	0.089	0.089
Area w/o Mem.[1]	0.303	0.650	1.301	0.425
Memory Area[1]	3.291	3.256	2.614	2.614
Total Area[1]	3.594	3.906	3.915	3.040

[1]*Note*: unit of area is mm^2

Table 5 Power estimation parameters of processors and hardware.

Parameter	Single-ARM		Multi-core			
	Non-cache	Cache	ARM	TCT	Synch.	Mailbox
Freq.(MHz)	200	200	100	100	100	100
Technology	90 nm	90 nm	90 nm	90 nm	90 nm	90 nm
Supply Voltage	1.0 V	1.0 V	1.0 V	1.0 V	1.0 V	1.0 V
Static*	0.51	1.10	0.51	0.15	0.10	0.05
Dynamic*	15.20	21.20	7.60	3.32	2.04	1.11
Total Power*	15.71	22.30	8.11	3.48	2.14	1.16

**Note*: unit of power is *mW*

task scheduling and context switching. Taking single-ARM TCM architecture for example, the *Task Switching* portion accounts for a 61% overhead in the Beacon TX task, 53% in the Beacon RX task, 40% in the Data TX task and 26% in the Data RX task. However, in our MPSoCs, each task is statically assigned a dedicated processor, and thus dynamic task scheduling and context switching no longer exist, resulting in the saving of the *Task Switching* portion. As regards the latter, the synchronization and mailbox operations in RTOS are implemented in pure software, which is time-consuming. For instance, in single-ARM TCM architecture, the *Mailbox* and *Synch.* portions together account for a 25% overhead in the Beacon TX task, 35% in the Beacon RX task, 50% in the Data TX task and 68% in the Data RX task. However, in our MPSoCs, the synchronization and communication operations are supported by hardware, reducing the workload of this portion by more than 88%. Finally, due to this two-level acceleration, the multitasking overhead in the proposed MPSoC is reduced significantly.

5.5 Area and Energy Evaluation

This subsection evaluates the area and energy efficiency of the proposed MPSoC. Table 4 and Table 5 list the detailed area and power related parameters used for comparison. The area and power information of the TCT PE, synchronization engine and mailbox controller is obtained by using RTL synthesis and post-synthesis simulation [28] with TSMC 90 nm CMOS technology. For ARM946E-S, its area and power model is derived from officially published data [12]. The area and power model of memory, as summarized in Table 6, is obtained through CACTI v6.5 tool [29] for 90 nm technology.

As shown in Table 4, we account for the area of processors, hardware engines and memories (for local data and instructions). First, comparing the area without memory, multi-ARM times the area of single-ARM due to the duplication of PE and the adoption of hardware engines. However, due to the utilization of simple PEs, multi-TCT costs much less area than single-ARM with cache, but it is still larger than single ARM with TCM. Then, comparing the total area with memory, multi-TCT achieves about 15% area saving with respect to single-ARM with TCM, which is due to the less on-chip memory requirement.

Table 6 Power model of data and instruction memories.

Architecture		Size	Area (mm^2)	Read/Write Energy(nJ)	Static Power(μW)
Single (TCM)	Data	78 KB	0.85	0.057/0.041	10.733
	Inst.	230 KB	2.44	0.109/0.067	31.490
Single (Cache)	Data& Inst.	308 KB	3.26	0.111/0.076	42.162
Multi-core	Data	4 KB	0.05	0.017/0.010	0.589
		4 KB	0.05	0.017/0.010	0.589
		4 KB	0.05	0.017/0.010	0.589
		20 KB	0.23	0.038/0.021	2.898
	Inst.	8 KB	0.09	0.019/0.011	1.159
		42 KB	0.46	0.043/0.025	6.063
		82 KB	0.89	0.056/0.041	11.280
		72 KB	0.79	0.055/0.040	9.911

Power estimation of the single-core and multi-core platforms are also carried out at system level. We account for the power consumption from PE execution (E_{PE}), hardware engines (E_{HW}) and PE's local memories (E_{Mem}).

$$E_{Total} = E_{PE} + E_{HW} + E_{Mem}, \tag{4}$$

The energy of PE execution (E_{PE}) is estimated by:

$$E_{PE} = W_{PE} \times P_{d\_PE} \div f_{PE}, \tag{5}$$

where W_{PE} is the PE execution cycle derived from Eq. (1), P_{d_PE} and f_{PE} are, respectively, the dynamic power and frequency of PE. It is noteworthy that, due to the large amount of workload saving, PEs in multi-core platforms can execute at a lower frequency (100 MHz) than single-core platform (200 MHz). Moreover, the power consumption of hardware engines (E_{HW}) is also taken into account by:

$$E_{HW} = W_{HW} \times P_{d\_HW} \div f_{HW}, \tag{6}$$

where W_{HW} is their active cycles, P_{d_HW} and f_{HW} are, respectively, their dynamic power and frequency. Finally, the energy for memory access (E_{Mem}) is assumed to be:

$$E_{Mem} = N_{Mem} \times E_{Mem/access}, \tag{7}$$

where N_{Mem} is the memory access count obtained from the simulator [13] and $E_{Mem/access}$ is the energy per access [29]. Additionally, we assume that the static power of these components are consumed all the time.

Figure 14 shows the power consumption of a MAC system for maintaining a two-device network under various data throughput. At each given throughput, the power consumption of the single-ARM with TCM, single-ARM

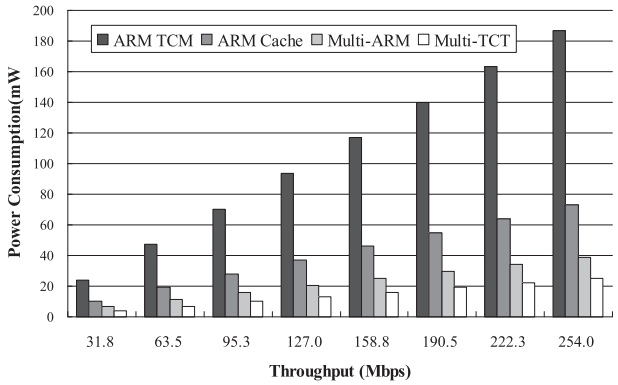

Fig. 14 Power consumption under various data throughput.

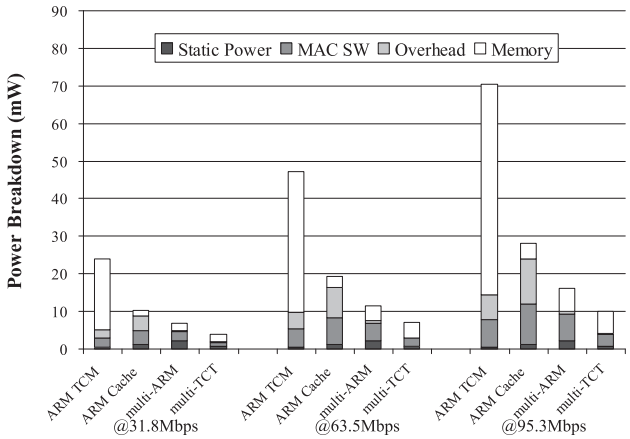

Fig. 15 Power breakdown at throughput 31.8 Mbps and 63.5 Mbps.

with cache, multi-ARM and multi-TCT platforms are reported respectively. The graph shows that, multi-core platforms, including both multi-ARM and multi-TCT, reduce significant amount of power consumption when comparing with the single-ARM counterparts. Comparing with single-ARM with TCM, multi-ARM and multi-TCT obtain about 75% and 85% power saving, respectively. Comparing with single-ARM with cache, multi-ARM and multi-TCT achieve about 45% and 65% power saving, respectively.

Figure 15 further shows the power distribution at three given throughput of 31.8 Mbps, 63.5 Mbps and 95.3 Mbps. The total power of the four architectures is broken down into static power and dynamic power. The dynamic power further contains the power consumption due to the memory accessing and PE execution, which is more finely divided into *MAC SW component* - power consumption due to execution of MAC software, and *overhead component* - power consumption due to multitasking and interrupt processing. In single-ARM with TCM, two large memories, as mentioned in Sect. 5.2, are used for local data and instruction. However, in multi-core platforms, the local memory is split into multiple smaller memories distributed to each node, which features much lower energy per access than the larger memories [29]. On the other hand, the saving of overhead further reduces the amount of memory access. Therefore, the energy for memory accessing is significantly reduced in multi-core platform. Moreover, comparing the power consumption from PE execution, multi-core platforms also achieve lower power dissipation, which is due to the reduction of RTOS overhead. In the single-ARM with cache, the usage of cache saves a significant energy for memory access. However, due to the more execution cycles and the power from cache, this architecture consumes much more energy for PE execution. Thus, its total power is still much higher than multi-core platforms. Finally, comparing multi-ARM with multi-TCT, multi-TCT achieves further more energy saving, which is due to the lower power of the TCT PE when compared to the general purpose ARM.

6. Conclusion

In this paper we presented a low-cost and energy-efficient MPSoC solution for a resource-constrained embedded application, UWB MAC. The proposed MPSoC helps the target application to overcome the performance bottleneck of using the traditional uniprocessor architecture. By distributing the complex multi-task application to multiple processors and executing a unique task on each processor, RTOS can be removed to relieve the processors from multitasking overhead, and thus, achieve performance improvement. Then, due to reduced workload and hardware support for synchronization and communication, simple RISC processor can be used, instead of general purpose one, to save cost and energy. Finally, we show the effectiveness of the proposed MPSoC solution by comparing a four-core design with the uniprocessor design. Results show the proposed MPSoC achieves up to 45% performance improvement and 65% power saving, but costs 15% less area when compared with the single-core architecture which is based on a high-end general-purpose processor.

Moreover, it is noteworthy that the proposed design methodology, including software migration and MPSoC architecture, is independent from the processor. Therefore, we believe it should be not only effective for the WiMedia MAC, but also helpful for other embedded applications to achieve performance improvement and power saving.

As future work, fine-grained power management strategies will be explored to further reduce power dissipation. Moreover, application-specific instructions will also be designed and extended into our base RISC processor to further improve the performance and power efficiency.

References

[1] R. Ruby and J. Pan, "Performance analysis of WiMedia UWB MAC," Proc. 29th IEEE Int. Conf. Distributed Computing Systems, pp.504–510, Montreal, Canada, July 2009.
[2] Standard ECMA-368: High Rate Ultra Wideband PHY and MAC Standard, Dec. 2008.
[3] Y. Jeon, S. Lee, S. Lee, S. Choi, and D.Y. Kim, "High definition video transmission using bluetooth over UWB," IEEE Trans. Consum. Electron., vol.56, no.1, pp.27–33, Feb. 2010.
[4] J. Kim, S. Lee, Y. Jeon, and S. Choi, "Residential HDTV distribution system using UWB and IEEE 1394," IEEE Trans. Consum.

Electron., vol.52, no.1, pp.116–122, Feb. 2006.
[5] L.X. Cai, X. Shen, and J. Mark, "Efficient MAC protocol for ultra-wideband networks," IEEE Commun. Mag., vol.47, no.6, pp.179–185, June 2009.
[6] G. Sen, P. Xiaoming, Q. Xuhong, and T.K. Seng, "System architecture design considerations for distributed WiMedia MAC protocol," Proc. 6th IEEE Consumer Communications and Networking Conference (CCNC 2009), pp.1–5, Las Vegas, United States, Jan. 2009.
[7] K. Sakoda, Y. Morioka, C. Fujita, E. Tanimoto, K. Nishikawa, and M. Suzuki, "Implementation and evaluation of WiMedia MAC LSI," Proc. 3rd Int. Conf. Information Security and Assurance (ISA 2009), pp.438–449, Seoul Korea, June 2009.
[8] S.K. Ganesan, P. Dixit, M.K. Saxena, and E. Bhatnagar, "Architecture enhancement of a CWUSB system for high throughput," Proc. IEEE Symposium on Industrial Electronics & Applications (ISIEA 2010), pp.1–4, Penang, Malaysia, Oct. 2010.
[9] T.Y. Tse, "WiMedia UWB MAC layer implementation in FPGA," Proc. 5th Int. Conf. Wireless and Mobile Communications (ICWMC 2009), pp.234–238, Cannes, La Bocca, Aug. 2009.
[10] "ARM11 MPCore Processor Technical Reference Manual," ARM Inc..
[11] J.A. Kahle, M.N. Day, H.P. Hofstee, C.R. Johns, T.R. Maeurer, and D. Shippy, "Introduction to the Cell multiprocessor," IBM J. Research and Development, vol.49, no.4/5, pp.589–604, July 2005.
[12] ARM946E-S, ARM Inc., http://www.arm.com/
[13] Platform Architect, Synopsys Inc., http://www.synopsys.com/
[14] Processor Designer, Synopsys Inc., http://www.synopsys.com/
[15] D.E. Culler, J.P. Singh, and A. Gupta, Parallel Computer Architecture: A Hardware/Software Approach, pp.334–358, Morgan Kaufmann, 1998.
[16] J. Lee, V.J. Mooney III, A. Daleby, K. Ingstrom, T. Klevin, and L. Lindh, "A comparison of the RTU hardware RTOS with a hardware/software RTOS," IEEE Proc. ASP-DAC 2003, pp.683–688, Jan. 2003.
[17] M. Monchiero, G. Palermo, C. Silvano, and O. Villa, "Efficient synchronization for embedded on-chip multiprocessors," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.14, no.10, pp.1049–1062, Oct. 2006.
[18] A.C. Nacul, F. Regazzoni, and M. Lajolo, "Hardware scheduling support in SMP architectures," Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE'07), pp.1–6, April 2007.
[19] L. Lindh, and F. Stanischewski, "FASTCHART - A fast time deterministic CPU and hardware based real-time-kernel," IEEE Euromicro Workshop on Real-Time Systems, pp.36–40, June 1991.
[20] T. Nakano, A. Utama, M. Itabashi, A. Shiomi, and M. Imai, "Hardware implementation of a real-time operating system," Proc. 12th TRON Project International Symposium, pp.34–42, 1995.
[21] μITRON4.0 Specification Ver.4.00.00, ITRON Committe, TRON Association.
[22] K. Baynes, C. Collins, E. Fiterman, B. Ganesh, P. Kohout, C. Smit, T. Zhang, and B. Jacob, "The performance and energy consumption of embedded real-time operating systems," IEEE Trans. Comput., vol.52, no.11, pp.1454–1469, Nov. 2003.
[23] M.Z. Urfianto, T. Isshiki, A.U. Khan, D. Li, and H. Kunieda, "A multiprocessor SoC architecture with efficient communication infrastructure and advanced compiler support for easy application development," IEICE Trans. Fundamentals, vol.E91-A, no.4, pp.1185–1196, April 2008.
[24] Language for Instruction Set Architectures (LISA 2.0), Synopsys Inc., http://www.synopsys.com/
[25] A. Sinha, N. Ickes, and A.P. Chandrakasan, "Instruction Level and Operating System Profiling for Energy Exposed Software," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.11, no.6, pp.1044–1057, Dec. 2003.
[26] TOPPERS Project, http://www.toppers.jp/
[27] WiMedia MAC White Paper, "WiMedia ultra-wideband: efficiency considerations of the effects of protocol overhead on data throughput," 2009.
[28] Power Compiler, Synopsys Inc., http://www.synopsys.com/
[29] N. Muralimanohar, R. Balasubramonian, and N.P. Jouppi, "CACTI 6.0: A tool to model large caches," HP Laboratories, 2009.

Hao Xiao received his B.E. degree from Zhejiang University and M.S. degree from Fudan University, P.R China, in 2005 and 2009, respectively. Since 2009, he has been studying for his Ph.D. degree in the Department of Communications and Integrated Systems, Tokyo Institute of Technology. His interests include MPSoC hardware/software co-design and VLSI architecture design for communication systems.

Tsuyoshi Isshiki has received his B.E. and M.E. degrees in electrical and electronics engineering from Tokyo Institute of Technology in 1990 and 1992, respectively. He received his Ph.D. degree in computer engineering from University of California at Santa Cruz in 1996. He is currently an Associate Professor at Department of Communications and Integrated Systems in Tokyo Institute of Technology. His research interests include MPSoC programming framework, high-level design methodology for configurable systems, bit-serial synthesis, FPGA architecture, image processing, fingerprint authentication algorithms, computer graphics, and speech synthesis. Dr. Isshiki is a member of IEEE CAS, and IPSJ.

Arif Ullah Khan received his B.Sc. degree in Electrical Engineering from NWFP University of Engineering and Technology Pakistan in 2001 and M.Sc. degree in Information and Communication Engineering from University of Karlsruhe Germany in 2004. He is currently working as a researcher at Department of Communication and Integrated Systems at Tokyo Institute of Technology. His interests include MPSoC design framework, NOC and HW/SW integration in MPSoC. Mr. Arif is a member of IEEE and IPSJ.

Dongju Li received her B.S. degree from LiaoNing University and M.E. degree from Harbin Institute of Technology, China, in 1984 and 1987, respectively. She worked as an IC design engineer in VLSI Design Laboratory of Northeast Micro-electronics Institute, Electronic Industry Burean, China, from 1987-1993. She is currently a Research Associate at Department of Communications and Integrated Systems in Tokyo Institute of Technology. Her current research interests are in embedded fingerprint authentication algorithms, VLSI Architecture and Design Methodology. System on Chip design for multimedia processing including video CODEC. Dr. Li is a member of IEEE CAS.

Hiroaki Kunieda was born in Yokohama in 1951. He received B.E., M.E. and Dr. Eng. degrees from Tokyo Institute of Technology in 1973,1975 and 1978, respectively. He was Research Associate in 1978 and Associate Professor in 1985, at Tokyo Institute of Technology. He is currently Professor at Department of Communications and Integrated Systems in Tokyo Institute of Technology. He has been engaged in researches on Distributed Circuits, Switched Capacitor Circuits, IC Circuit Simulation, VLSI CAD, VLSI Signal Processing and VLSI Design. His current research focuses on fingerprint authentication algorithms, VLSI Multimedia Processing including Video CODEC, Design for System On Chip, VLSI Signal Processing, VLSI Architecture including Reconfigurable Architecture, and VLSI CAD. Dr. Kunieda is a member of IEEE CAS, SP society, and IPSJ.

Yuko Nakase received her B.E. and M.S. degrees from Osaka University in 2002 and 2004, respectively. In 2004, she joined Ricoh Co., Ltd. Since then, she has been engaged in the research and development of wireless communication system.

Sadahiro Kimura received his B.E. degree from Kyoto Sangyo University and M.S. degree from Nara Institute of Science and Technology in 1994 and 1996, respectively. He entered Ricoh in 1996. He is a member of Information Processing Society. He has been engaged in the research and development of wireless communication system with the electronic system level design.