2010 10th International Conference on Quality Software

Building Multi-Kernel Embedded System on PAC Multi-Core Platform

Jing Chen[*]
jchen@mail.ncku.edu.tw

Chung-Ping Young[#]
cpyoung@mail.ncku.edu.tw

Da-Wei Chang[#]
davidchang@csie.ncku.edu.tw

Guan-Ying Huang[*]
q3697106@mail.ncku.edu.tw

Chung-Yuan Ke[*]
q3697123@mail.ncku.edu.tw

Shih-Tun Yen[#]
shihtun@gmail.com

Tsang-Shuo Kuo[#]
p76981340@mail.ncku.edu.tw

[*]Department of Electrical Engineering
[#]Department of Computer Science and Information Engineering
National Cheng Kung University
No. 1 University Road, Tainan City, Taiwan, R. O. C.

***Abstract*—It is common nowadays that consumer embedded system products are built on platforms with System-On-a-Chip (SOC) in which two or more processor cores, which are not necessarily of the same type, are put into a single chip and form the architecture of Chip-level Multi-Processor (CMP). Although such platform is capable of achieving high performance at relatively low cost, the system architecture of CMP brings new design challenges as well as increased complexity in developing embedded software especially at the level of kernel or operating system software. This paper presents our experience and some preliminary results from the project of building a multi-kernel embedded system platform for application software running in the environment of a newly developed multi-core SOC, namely PAC Duo SOC, which is the latest product from the PAC (short for Parallel Architecture Core) Project initiated by the Industry Technology Research Institute (ITRI) in Taiwan. PAC Duo SOC is a chip-level heterogeneous multi-processor SOC composed of one ARM926 core serving as the general purpose processor (GPP for short) and two ITRI PAC DSP cores serving as the special purpose processors (SPP). We ported Linux operating system to run on the ARM926 processor and ported μC/OS-II real-time kernel to run on one PAC DSP core, leaving the other PAC DSP core with the option, for flexibility, of running either μC/OS-II or a different kernel. In addition, an inter-processor communication (IPC) mechanism is developed which not only takes application-specific requirements into account but also takes advantages of hardware features.**

***Keywords*- embedded system; multicore; operating system; Linux; μC/OS-II**

I. INTRODUCTION

As modern consumer embedded system products, such as portable media players (PMP), personal navigation devices (PND), PDA, smart phones, are required to have sophisticated functionality with high performance yet to keep power consumption low and small in size, two trends have emerged. One is adopting operating system in the embedded software, and the other is deploying SOC (System-On-a-Chip) as the main hardware component. There are more and more products using an SOC in which two or more processor cores are put into a single silicon chip. The SOC typically consists of one GPP (General Purpose Processor) and one SPP (Special purpose Processor) to co-operate, in addition to performing their dedicated functions. OMAP [10] and DaVinci [11] of Texas Instruments are well-known examples, which have been used in many consumer embedded system products. Both OMAP and DaVinci are composed of an ARM9 core and a DSP core to form a platform of chip-level heterogeneous multiprocessor architecture.

While embedded system products with multi-core SOC can effectively meet the requirements imposed from some possibly conflicting restrictions, such as high performance and low cost, the architecture of chip-level heterogeneous multiprocessor brings not only new challenges but also increased complexity in developing embedded software, especially at the level of kernel or operating system software. In this paper, we present our experience and some preliminary results from a three-year project of building a multi-kernel embedded system on the platform which is equipped with a multi-core SOC, namely PAC Duo SOC [28-31] (PAC stands for Parallel Architecture Core), for running application software. PAC Duo SOC, the successor of PAC Solo [13, 14] , is a chip-level heterogeneous multiprocessor SOC composed of an ARM926 core as the GPP and two ITRI PAC DSP cores as the SPP [14]. It is the latest product from the PAC Project initiated by the Industry Technology Research Institute (ITRI) in Taiwan [13-15].

Aside from following the common approach of developing operating systems for multi-processor system [1-7, 9], in which the SPP is usually treated as a special device of the GPP, the option of building multi-kernel embedded system environment on the PAC multi-core platform is adopted by this project, in which there is one kernel running on each processor core. The project started from porting Linux operating system to the ARM926 processor and porting μC/OS-II real-time kernel [16] to one of the two PAC DSP cores. The objective is to provide a

1550-6002/10 $26.00 © 2010 IEEE
DOI 10.1109/QSIC.2010.65

basic execution environment at the earliest stage so that existing application software might be ported to run on the platform. For the other DSP core of PAC Duo SOC, we plan to built another kernel so that, in addition to making the best of PAC's processing capability, software developers are provided with flexibility in software configurations to help meet various application requirements. The implementation of a dataflow kernel on the PAC DSP is therefore included in building our multi-kernel embedded system on the PAC multi-core platform. The rationale is that software developers may choose a suitable configuration from the combinations of running μC/OS-II, or the dataflow kernel, or DSP-as-a-device on each PAC DSP core. In order to make the GPP and the SPP co-operate closely and efficiently, an inter-processor communication (IPC) mechanism is developed in this project, which not only takes into account application-specific requirements, such as transferring large blocks of data in multimedia applications, but also takes advantages of the hardware features supported by PAC SOC, e.g. shared memory and mailbox [14]. Building this multi-kernel environment provided challenging training for students, as PAC SOC is one relatively new product, and the experience would be valuable for developing software on this platform.

This paper is organized as follows. After this introduction, Section II brings the background information and discusses some related works. The PAC multi-core platform is described in Section III. The building of this multi-kernel environment on the platform is described in Section IV, while Section V presents the performance evaluation. Section VI presents demo examples of running applications of multimedia processing in this multi-kernel embedded system for evaluation. Finally, Section VII gives a summary to conclude this paper.

II. Background and Related Works

For a modern embedded system with rich multimedia and communication capabilities, an execution environment with operating system can simplify the development of application software, while a multiprocessor platform can achieve high performance with less cost in power consumption. In terms of processor types, multiprocessor architecture in general can be classified into homogeneous and heterogeneous. Homogeneous multiprocessor consists of multiple processors of the same type, and it is more popular in personal computers or servers for general purpose usage, either desktops or notebooks, like Intel Core 2 Duo or Quad, AMD Athlon or Phenom, and Sun UltraSPARC, etc. Heterogeneous multiprocessor is composed of different types of processors, so it usually aims at some specific applications, such as IBM Cell, Texas Instrument OMAP or DaVinci, and ITRI PAC Solo or PAC Duo, etc.

In this paper, we describe building a multi-kernel embedded system on the ITRI PAC platform. The platform is equipped with a PAC Duo SOC which is an MPSoC (Multiprocessor System-on-Chip), a single chip with multiple built-in processor cores [33]. Existing approaches related to supporting dual or multiple operating system kernels in a single computing system are described in the following.

Virtualization is a common approach for executing multiple operating systems on a computing platform, even the one with a single processor [17]-[23]. However, the virtualization layer, also called the virtual machine monitor (VMM), could hurt the real-time performance of the DSP applications, and thus is not a suitable approach in the PAC environment.

Similar to our work, the Sigma system proposed a dual kernel architecture that runs the operating system kernels natively on the physical processors instead of the ones virtualized by the VMM [24]. However, it was targeted on an SMP platform, and the implementation was done on an Intel Core2 Duo PC. By contrast, we target on a heterogeneous multiprocessor architecture used in embedded systems.

Some recent research efforts proposed new operating system architectures for many-core systems. Corey [25] is a library operating system that allows the applications to control the sharing of resources such as address ranges and kernel cores. Tessellation [26] is a new operating system that divides the system resources such as cores, cache, and network bandwidth based on space-time partitioning so as to maximize the performance of parallel applications. The Multikernel [27] system addresses the scalability issues of traditional operating systems. Inspired by the concept of distributed systems, the Multikernel operating system regards a many-core system as a network of numerous independent cores and divides the operating system functionality into different modules running on different cores. Inter-communication is achieved via MPI (Message Passing Interface). Unlike [32-34], we implement a simple MPI for the communication between processor cores. In addition, different from these research efforts that target mainly on the scalability issues, our multi-kernel architecture aims at satisfying the real-time requirement of DSP applications.

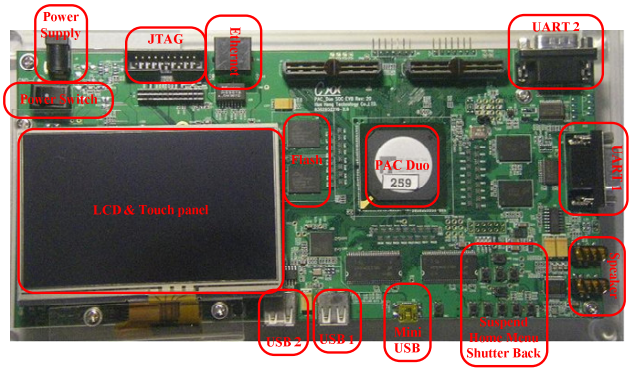

Figure 1. The PAC Duo EVM

III. Hardware Architecture of PAC

Figure 1 shows a picture of PAC Duo EVM which demonstrates connecting the rich set of devices to PAC Duo SOC, which are common in modern embedded system products. Figure 2 depicts the hardware architecture of the PAC Duo SOC [31]. As mentioned earlier, PAC Duo SOC is a chip-level heterogeneous multiprocessor SOC which is composed of an ARM926EJS (the MPU at the center left in Figure 2) and two PAC DSP (Digital Signal Processing) cores of the same architecture (DSP1 and DSP2 at the upper left in Figure 2). The ARM926EJS serves as the general purpose processor (GPP)

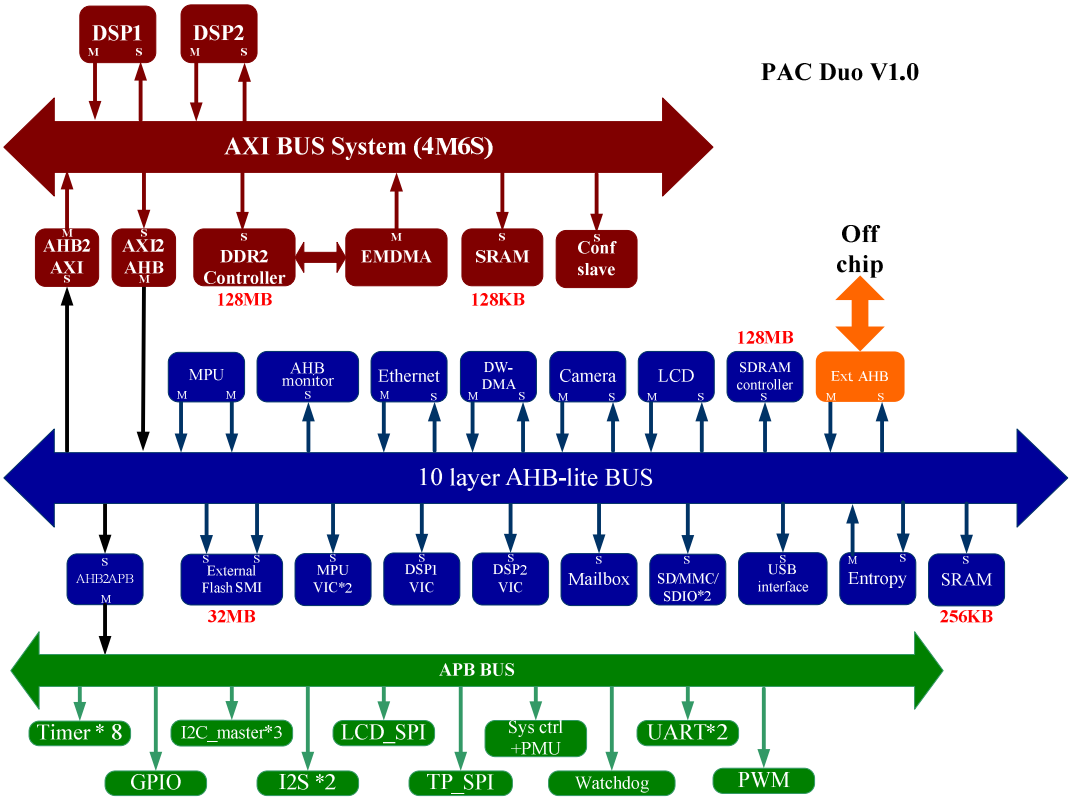

Figure 2. The architecture of PAC Duo SOC [31]

while DSP1 and DSP2 can be treated as special purpose processors (SPP) to co-operate with the GPP.

PAC DSP is a five-way VLIW DSP core [28]. Its architecture includes a scalar unit, two load/store units, and two arithmetic units. It uses distributed register file with low access latency and power consumption, and it utilizes variable-length operation encoding techniques to achieve increasing the code density [13, 14]. In addition, PAC DSP supports Dynamic Frequency Scaling (DFS) with the frequencies ranging from 24MHz to 550MHz [13].

Each DSP core in PAC Duo has a 64KB local memory and resides on the 32-bit AXI bus. Communication with the ARM processor can be achieved through an AXI-AHB bridge since the ARM processor resides on the 32-bit AHB bus. PAC Duo supports inter-processor communication at hardware level through hardware mailbox mechanisms and shared memory [13]. The former is interrupt-driven, allowing a processor to send interrupts to another processor for event notification. The latter allows the processors to share data or states. There are four banks of shared memory on the platform. Two banks of shared memory (128KB SRAM and 128MB DDR2 DRAM) reside on the AXI bus while the other two (256KB SRAM and 128MB SDRAM) reside on the AHB bus. Figure 3 shows the memory map of PAC Duo [31].

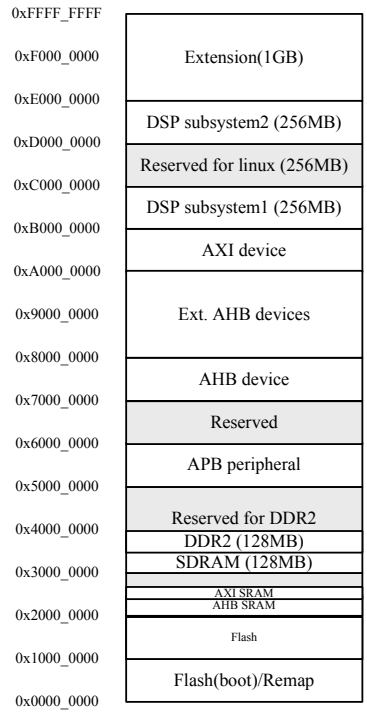

Figure 3. The memory map of PAC Duo [31]

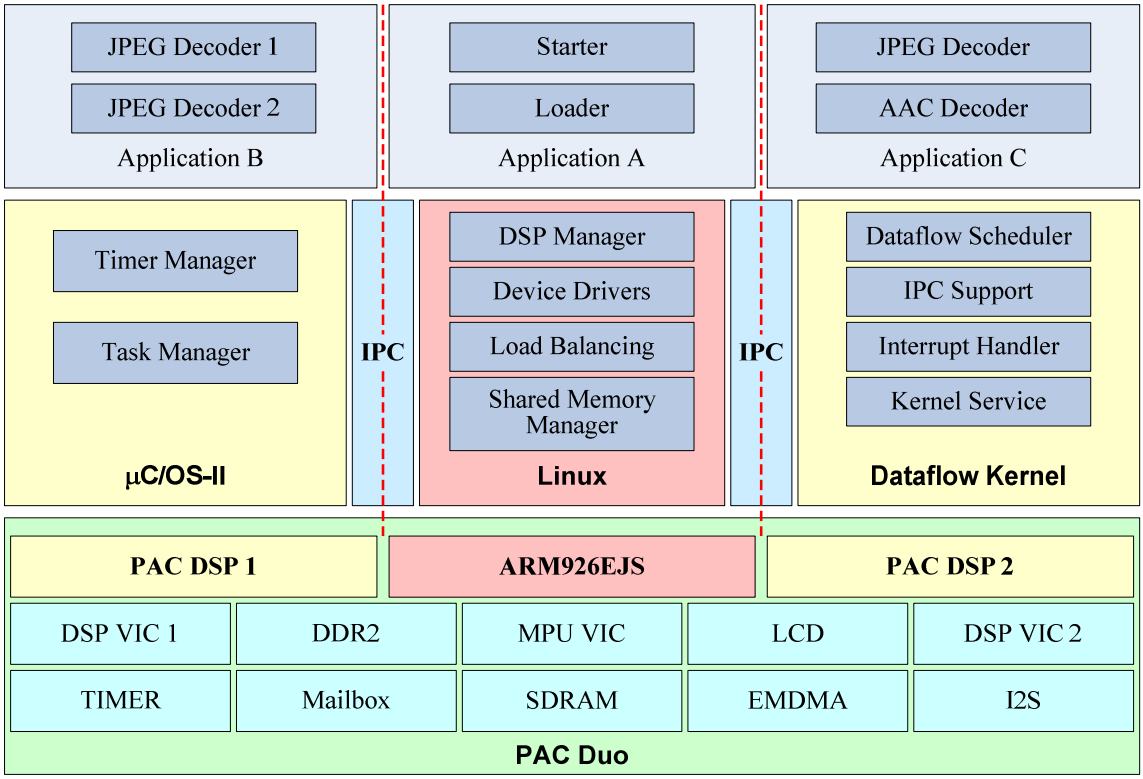

Figure 4. Software architecture of the multi-kernel embedded system on the PAC multi-core platform

IV. The Multi-Kernel Embedded System

This section presents our approach of building a multi-kernel embedded system on PAC multi-core platform. An overview which describes the whole picture of our design and implementation comes first. The descriptions of important parts then follow. Since the two PAC DSP cores in PAC Duo are of the same type, unless otherwise needed, in the following, we do not distinguish them by using PAC DSP as their generic name. When it is necessary, the names DSP1 and DSP2 will be used.

A. Overview

Figure 4 depicts the software architecture of our intended multi-kernel embedded system on the PAC multi-core platform. There are three kernels running on the three processor cores of PAC Duo. In accordance with the processor types, the kernels can be categorized into the one on the ARM side and the ones on the PAC DSP side. Communication among the kernels is achieved through an inter-processor communication (IPC) mechanism. The main parts of this multi-kernel architecture are thus the ARM side, the PAC DSP side, and the IPC.

In Figure 4, a Linux kernel is shown to run on the ARM side as the ARM926EJS is the GPP and Linux is a general purpose operating system. On the PAC DSP side, the real-time kernel μC/OS-II runs on DSP1 and a dataflow kernel runs on DSP2. This multi-kernel architecture allows an application to be separated into real-time (RT) tasks and non-RT tasks. The former ones can run on the DSP side supported by the μC/OS-II real-time kernel and the latter ones can run on the ARM side supported by the Linux kernel. As an example, an MP3 player application can be divided into the decoder engine task, the user interface task, and the file system management task. The former task can be allocated to run on the PAC DSP side for satisfying its real-time requirement, while the latter two tasks can be allocated to execute on the ARM side, leveraging the rich functionality of Linux. Further, there is a dataflow kernel on the PAC DSP side which can support multimedia tasks or share the loading allocated to the PAC DSP side. Note that, on the PAC DSP side, μC/OS-II can run on either DSP1 or DSP2, or both PAC DSP cores, although DSP1 is the host in Figure 4. This is true for the dataflow kernel.

The blocks of IPC (Inter-Processor Communication) shown in Figure 4 are intended to support efficient communication and data sharing among software running on these processor cores. They stand for separately designed functions achieving the purposes. The implementation of IPC will be integrated into each kernel.

B. ARM side

As the GPP in PAC Duo is the processor core on the ARM side, the Linux with kernel version 2.6.27 is ported to run on the ARM926EJS in order to provide general operating system functionalities and support running non-RT tasks. This Linux kernel manages the ARM processor and the I/O devices such as

LCD, Ethernet, UART, etc. and storages. It also manages the execution of applications on the ARM side just as an ordinary Linux operating system.

In order to support hardware and software on the PAC DSP side, functionalities of managing PAC DSP, managing shared memory, IPC, and load balancing support are added into the Linux. These functions are separately implemented as Linux loadable kernel modules in order to run in kernel mode and maintain good modularity of the whole system.

The ARM side is also responsible for handling the jobs of initializing PAC DSP and activating the kernel, either µC/OS-II or the dataflow kernel, on PAC DSP. Specifically, these jobs are achieved by the DSP loader on the ARM side, which is implemented as an application of Linux. The flow and the main steps of initializing a PAC DSP and activating µC/OS-II on its host PAC DSP is shown in Figure 5. During the first step, the pre-built image of µC/OS-II together with the wrapped tasks is loaded into the memory shared by the ARM processor and the host PAC DSP. This includes loading the code section into one shared memory bank and loading the data section into the local memory of the host PAC DSP. Then, the cache of the host PAC DSP is flushed to guarantee a clean start. In the third step, the program starting address of the host PAC DSP, which holds the address of the entry point of µC/OS-II, is loaded into the corresponding register. Finally, the host PAC DSP is started by setting a bit in its PSTART register and the execution of µC/OS-II is thus trigged. To start running the dataflow kernel or other guest kernel on PAC DSP assumes the procedure of similar steps. The initialization of a PAC DSP and starting the kernel on it can be done at any time after the kernel on the ARM side is ready to run a user-level application although in general this is done as the first job when the Linux kernel starts.

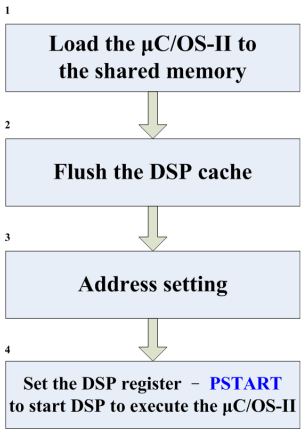

Figure 5. The flow of initializing PAC DSP and starting µC/OS-II

C. PAC DSP side

The PAC DSP side is designated to run tasks with timing constraints or to support tasks with particular requirements such as multimedia processing. In achieving this purpose and avoiding implementing a kernel from scratch on PAC DSP core, the real-time kernel µC/OS-II is ported to run on PAC DSP side. The porting is briefly described below.

µC/OS-II defined a processor dependent layer to achieve its portability [16]. Porting µC/OS-II to another platform therefore mainly involves modifying the source code of the layer which is implemented through three files, namely OS_CPU_C.c with its header file OS_CPU.h, and OS_CPU_A.asm. Important data types, constants and macros used by the kernel are defined in the header file OS_CPU.h. In the file OS_CPU_C.c, a function, namely OSTaskStkInit(), must be modified in accordance with the stack configuration of PAC DSP to initialize the stack of a task. The file OS_CPU_A.asm contains four functions that need modification: OSTickISR(), OSIntCtxSw(), OSCtxSw(), and OSStartHighRdy(). The function OSStartHighRdy() runs the ready task that has the highest priority by restoring the context of the task. The function OSTickISR() implements the timer ISR. Context switches could occur in the timer ISR if the corresponding timer interrupts change the state of the run queue, e.g. a high priority task is waked up by the timer ISR. The functions OSCtxSw() and OSIntCtxSw() perform context switch in the process and interrupt context, respectively. In general, a context switch involves saving the context of a task and loading the context of another task. The context in PAC DSP is defined by five types of registers including the registers of general purpose scalar, ping-pong, accumulator, address, and control flags [28].

As mentioned earlier, the production system of PAC multi-core platform can be configured to run one µC/OS-II on each PAC DSP, or run µC/OS-II on either one of the PAC DSP and run the dataflow kernel on the other PAC DSP. This notion of including a dataflow kernel in the multi-kernel architecture was conceived from [36-38]. A dataflow kernel supports dataflow software architecture in which application control is composed of elementary control objects and data channels [37]. It works in a fire-on-input manner which is different from the traditional tick-control model as in µC/OS-II. This project considers a dataflow kernel as an alternative at PAC DSP side in thinking of complementing the features of µC/OS-II, as well as making the best of PAC Duo's processing capability. By choosing the suitable one from the combinations of running µC/OS-II, or the dataflow kernel, or DSP-as-a-device on each PAC DSP core, software developers have the flexibility in configurations to help meet various application needs. The dataflow kernel mentioned is currently under development. Due to the space limit here, the details are omitted.

D. The Inter-Process Communication (IPC) Mechanism

In order to achieve co-operation between the tasks running on the ARM side and those running on the PAC DSP side, a mechanism realizing such IPC is definitely required. In many multiprocessor systems, the IPC is usually implemented via software approach with, possibly, some hardware supporting features. Since Linux kernel and µC/OS-II are both developed initially to run in single processor environment, this project therefore includes developing an IPC mechanism, of which the implementation is integrated into each kernel in this multi-kernel architecture.

Aside from the concerns of efficiency and flexibility, the IPC is developed with these considerations in mind: taking advantages of hardware features, fitting the kernel structures,

meeting application-specific needs. As described earlier, PAC Duo SOC, at hardware level, supports IPC by shared memory and mailbox [13]. This suggests that the IPC be implemented as function primitives to wrap up the operation details such as reading/writing shared memory and managing the usage of the mailbox. In fitting the kernel structures, the IPC at the ARM side is designed to be realized as a loadable kernel module of Linux, while the implemented at the PAC DSP side provides a function library. To satisfy application-specific needs, it is observed that transferring a large block of data is common in multimedia processing and the performance impact can not be overlooked. Using DMA in the transfer is naturally the choice.

Based on the above, the IPC is designed assuming a two-layer structure. The bottom layer consists of function primitives to transfer data through shared memory. Its implementation is represented by the function interfaces that support exchanging data between the ARM side and the DSP side. Due to the differences in memory re-mapping and hardware structure, these function interfaces are divided into two groups, as listed in Table I and Table II respectively, one for the ARM side and the other for the PAC DSP side.

TABLE I. IPC FUNCTION PRIMITIVES FOR ARM SIDE

Function Name	Operation
AXI_Read_SRAM	Read data from SRAM on AXI bus
AXI_Write_SRAM	Write data into SRAM on AXI bus
AXI_Read_DRAM	Read data from DRAM on AXI bus
AXI_Write_DRAM	Write data into DRAM on AXI bus
AHB_Read_SRAM	Read data from SRAM on AHB bus
AHB_Write_SRAM	Write data into SRAM on AHB bus
AHB_Read_SDRAM	Read data from SDRAM on AHB bus
AHB_Write_SDRAM	Write data into SDRAM on AHB bus
ARM_Send	Send a message via mailbox
ARM_Receive	Receive a message from mailbox
DMA_Transfer	Transfer a block of data using DMA

TABLE II. IPC FUNCTION PRIMITIVES FOR DSP SIDE

Function Name	Operation
DSP_Write_Byte	Write one byte data into shared memory
DSP_Write_Block	Write a block of data into shared memory
DSP_Read_Byte	Read one byte data from shared memory
DSP_Read_Block	Read a block of data from shared memory
DSP_Send	Send a message via mailbox
DSP_Receive	Receive a message from mailbox

The upper layer of the IPC is composed of application level message passing interfaces which include functions of sending and receiving messages, functions of status query, and some supporting functions. Moreover, both blocked and non-blocked operations are supported by the functions of sending/receiving messages so that synchronization can be achieved via message passing. The functions of upper layer are listed in Table III. Their implementation provides symmetry in terms of function interfaces.

TABLE III. MESSAGE PASSING INTERFACES OF IPC

Function Name	Operation
PAC_Send_Blocked	Blocked sending a message
PAC_Send	Non-blocked sending a message
PAC_Receive_Blocked	Blocked receiving a message
PAC_Receive	Non-blocked receiving a message
PAC_Send_and_Wait	Send a message and wait for response
PAC_Send_Response	Send response for a received message
PAC_Query_Sent	Enquire the status of sent message
PAC_MsqQ_Reset	Reset the message queue
PAC_MsgQ_Status	Enquire the status of message queue
PAC_MsgQ_Peek	Receive a message without dequeuing
PAC_MsgQ_Discard	Discard a message from message queue

V. EVALUATION

We tested all the kernel services of the μC/OS-II kernel on the PAC DSP as well as the IPC between the ARM and the DSP cores to verify the correctness of the porting. We also measured the latencies of the DSP initialization (including the loading of the μC/OS-II kernel) as well as the performance of the main functions in the μC/OS-II kernel, and the results are shown in Table IV. From the table, loading the μC/OS-II kernel and starting the DSP requires a relatively long time. This is due to the loading of the μC/OS-II kernel image from the flash storage.

TABLE IV. PERFORMANCE OF MAIN μC/OS-II KERNEL FUNCTIONS AND THE DSP LOADER

Functions	Latencies
Context switch	5.7 us
Task creation	13.5 us
Task deletion	17.7 us
Load μC/OS-II kernel & start DSP	113 ms

In addition, we also evaluated the performance of shared memory and message passing. To measure the performance of shared memory access, we copied 100K bytes of data (via the ARM processor) from SDRAM on the AHB bus to different shared memory banks, and the throughputs (in Mbytes/second) are shown in the left four bars of Figure 6. As shown in the figure, there is no much performance difference among the four banks. The results show that copying a small block of data in the PAC platform is not efficient since the throughput is only about 10 Mbytes/second. Figure 6 also shows the performance of copying a larger block of data (i.e., 512K bytes) as well as copying data via DMA. From the figure, higher performance can be achieved when a larger data block is copied. Moreover, DMA is definitely a better choice for transferring large chunk of data. Specifically, DMA transfer could achieve an 8-fold throughput improvement than moving data via the processor on the PAC platform.

Table V shows the performance of message passing. In the table, the *message sending* row gives the measured time of sending a message from the ARM processor to one of the PAC

DSPs, via the interrupt-based mailbox mechanism supported by the PAC platform, and inserting the message in the message queue of the target task running on the PAC DSP, while the *message receiving* row gives the measured time of removing a message from the target message queue. Since the latter mainly involves simple manipulation on data structure of message queue and triggers no interrupts, it takes less time. Sending and receiving a message from one PAC DSP demonstrates similar performance and thus the results are not shown in this paper.

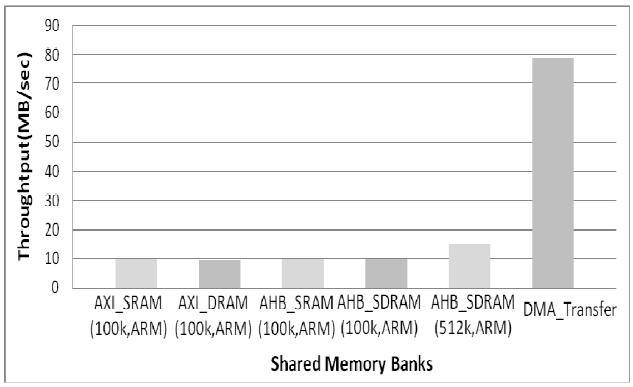

Figure 6. Throughput of Shared Memory Access (MB/s)

TABLE V. PERFORMANCE OF MESSAGE PASSING

Message Passing Functions	Time
Message Sending	6.4 us
Message Receiving	3.8 us

Finally, we measure the memory overhead. The code section of the μC/OS-II kernel image is about 144 Kbytes. Typically, this code section can be placed on the shared memory and thus does not put much pressure in terms of memory consumption on the PAC platform. However, PAC DSP requires that the data section of DSP program (in our case, the μC/OS-II together with the RT tasks) to be fitted in the 64KB DSP local memory. In our current implementation, the measured size of the data section of μC/OS-II is only 8.1 KB, leaving about 87% of the data space for the RT tasks. This overhead should be acceptable when compared to the benefit it brings in simplifying the RT application development.

VI. APPLICATION EXAMPLES

Two example applications were built to demonstrate the multimedia implementation based on multi-kernel embedded software on this PAC multi-core platform. One implemented JPEG decoding and the other one implemented H.264 decoding.

The JPEG decoding application, functioning like a digital picture frame, manages and displays 30 JPEG files. The files were stored in three folders and each folder holds ten pictures. At the start, the cover picture of each folder is displayed in a round-robin manner by DSP1. If a folder is chosen by pressing the touch screen when its cover picture is shown on the touch screen, the pictures in that folder are then displayed one after one for 1 seconds by DSP2.

The H.264 decoding application was implemented for demonstrating the cooperation between two PAC DSP cores to smoothly decode H.264 frames. During the booting procedure, the ARM processor is first booted, all peripheral modules, LCD display, memory, and the two PAC DSP cores are initialized, and then the DSP code and data are loaded into DSP local memory to start up. When executing decoding, each PAC DSP decodes alternating frames and stores the decoded data in shared memory. When each frame is decoded, ARM receives the IPC message from DSP and shows the decoded results on LCD display. Figure 7 shows the implementation of H.264 decoding on the PAC Duo platform.

Figure 7. The H.264 decoding application

VII. CONCLUSIONS

We present in this paper the experience and preliminary results in building a multi-kernel embedded system on PAC embedded multi-core platform. The multi-kernel architecture comes from running one kernel on each processor core in PAC Duo SOC which is composed of an ARM926 processor core and two PAC DSP cores. A Linux kernel is ported to run on the ARM core and μC/OS-II is ported to run on either or both of the PAC DSP cores. The kernels can co-operate not only effectively but also efficiently, with support from a separately developed IPC mechanism, by exchanging data and passing messages. Nevertheless, the PAC DSP cores can be configured to run another kernel so that software development can benefit from the flexibility in choosing a suitable configuration. The results from measuring the cost of running tasks at PAC DSP side show satisfactory performance and the efficiency of IPC mechanism. In addition, two example applications running on this platform demonstrated the achievement of this project. The experience obtained in the project not only showed challenging training for students, but also is valuable for developing new kernel or operating system on this platform.

Improvements on this multi-kernel embedded system are on-going. One enhancement is the fully functional dataflow kernel which integrates the current IPC. It is also expected that, based on this architecture, other application-specific kernel can be hosted to run on PAC DSP core. Therefore a framework is

needed to help port or develop a guest kernel. One interesting yet challenging issue in developing this framework would be the seamless integration of the functionality for application level message passing from current IPC mechanism. Another one is developing a sophisticated load sharing task model on this platform so that, in addition to co-operating by carrying out different functions, the GPP and the SPP may share the same workload such as processing multimedia data of the same data stream. As embedded system platform with multi-core SOC becomes prevailing in consumer products, it is hoped that this multi-kernel architecture can help make the best of hardware capability as well as provide a better runtime environment for embedded software.

ACKNOWLEDGMENT

This work is partially sponsored by the National Science Council (NSC) of Republic of China under the contract NSC98-2220-E-006-019.

REFERENCES

[1] Andrew S. Tanenbaum, Albert S. Woodhull, "Operating Systems: Design and Implementation", *2nd ed.*, Prentice Hall, 1997, ISBN 0136301959.

[2] Andrew S. Tanenbaum, "Modern Operating Systems**"**, *2nd ed*., Prentice Hall, 2001, ISBN 0130926418.

[3] B. Bershad, *et al*, "Operating System Support for High-Performance Parallel I/O Systems", *Scalable I/O Initiative Working Paper No. 4*, 1994.

[4] Charels Crowley, "Operating Systems: a Design-Oriented Approach", Irwin Book Team, 1997, ISBN 0256151512.

[5] DSPLinux, http://www.cadenux.com/ridgerun.

[6] Eran Gabber, *et al*, "The Pebble Component-Based Operating System", *Proc. USENIX Annual Technical Conference*, June, 1999.

[7] Gordon McNutt, Todd Fischer, "Using Linux to Control DSP Processors in Mixed-Processor Systems", Embedded Edge, October, 2001.

[8] Nucleus IPC, http://www.mentor.com/products/embedded_software/nucleus_rtos/nucleus_ipc/.

[9] Steve Muir, Jonathan Smith, "AsyMOS - An Asymmetric Multiprocessor Operating System", *Proc. OPENARCH '98*, April, 1998.

[10] Texas Instruments, "OMAP5910 Dual-Core Processor (Rev. D)", August 2004.

[11] Spectrum Digital Inc., "DaVinci EVM Reference Manual", 2007.

[12] William Stallings, "Operating Systems: Internals and Design Principles", *4th ed.*, Prentice Hall, 2001, ISBN: 0130319996.

[13] T. J. Lin, C. N. Liu, S. Y. Tseng, Y. H. Chu and A.Y. Wu, "Overview of ITRI PAC project - from VLIW DSP processor to multicore computing platform", *Proc. IEEE International Symposium on VLSI Design, Automation and Test (VLSI-DAT 2008)*, 2008, pp. 188-191.

[14] Chang, David Chih-Wei; Liao, I-Tao; Tseng, Shau-Yin; Jen, Chein-Wen, "PAC DSP Core and Its Applications", *Proc. 2006 IEEE Asian Solid-State Circuits Conference (ASSCC 2006)*, Nov. 2006, pp. 19-22.

[15] David Chih-Wei Chang; I-Tao Liao; Jenq-Kuen Lee; Wen-Feng Chen; Shau-Yin Tseng; Chein-Wei Jen, "PAC DSP Core and Application Processors", *Proc. IEEE International Conference on Multimedia and Expo*, July 2006, pp. 289-292.

[16] Jean J. Labrosse, "Micro-C/OS-II", *2nd ed.*, CMP Books, 2002, ISBN 1578201039.

[17] Rosenblum, M. and Garfinkel, T., "Virtual machine monitors: Current technology and future trends", *Computer*, 38(5), 39–47, 2005.

[18] Sugerman, J., Venkitachalam, G., and Lim, B., "Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor", *Proc. USENIX Annual Technical Conference*, pp. 1–14, 2001.

[19] Waldspurger, C., "Memory resource management in VMware ESX server", *ACM SIGOPS Operating Systems Review*, 26(si), 181, 2002.

[20] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., et al., "Xen and the Art of Virtualization", *Proc. 19th ACM Symposium on Operating Systems Principles*, pp. 164–177, 2004.

[21] Goldberg, R., "Survey of Virtual Machine Research", *IEEE Computer*, 7(6), 34–45, 1974.

[22] Joshua LeVasseur, Volkmar Uhlig, Yaowei Yang, Matthew Chapman, Peter Chubb, Ben Leslie and Gernot Heiser, "Pre-virtualization: Soft Layering for Virtual Machines", *Proc. 13th IEEE Asia-Pacific Computer Systems Architecture Conference*, Aug. 2008.

[23] Wataru Kanda, Yu Yumura, Yuki Kinebuchi, K Makijima, and Tatsuo Nakajima, "SPUMONE: Lightweight CPU Virtualization Layer for Embedded Systems", *Proc. IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, Dec. 2008.

[24] Wataru Kanda, Yu Murata and Tatsuo Nakajima, "SIGMA System: A Multi-OS Environment for Embedded Systems", *Journal of Signal Processing Systems*, Sept. 2008.

[25] Silas Boyd-Wickizer, Haibo Chen, Rong Chen, Yandong Mao, Frans Kaashoek, Robert Morris, Aleksey Pesterev, Lex Stein, Ming Wu, Yuehua Dai, Yang Zhang, Zheng Zhang, "Corey: An Operating System for Many Cores", *Proc. 8th USENIX Symposium on Operating Systems Design and Implementation,* Dec. 2008.

[26] R. Liu, K. Klues, S. Bird, S. Hofmeyr, K. Asanovic, and J. Kubiatowicz, "Tessellation: Space-Time Partitioning in a Manycore Client OS", *Proc. First USENIX Workshop on Hot Topics in Parallelism* (HotPar'09), March 2009.

[27] Andrew Baumann, Paul Barhamy, Pierre-Evariste Dagandz, Tim Harrisy, Rebecca Isaacsy, Simon Peter, Timothy Roscoe, Adrian Schupbach, and Akhilesh Singhania, "The Multikernel: A New OS Architecture for Scalable Multicore Systems", *Proc. 2009 ACM Symposium on Operating Systems Principles*, Oct. 2009.

[28] Industrial Technology Research Institute, "PACDSP3S0001-Processor Architecture", 2008.

[29] Industrial Technology Research Institute (ITRI), "PACDSP3S0003-Programming Guide", 2008.

[30] Industrial Technology Research Institute (ITRI), "PACDSP V3 Software Developer's Manual-Volume 1: Processor Architecture ", 2009.

[31] Industrial Technology Research Institute (ITRI), "PAC Duo Programming's Reference", 2009.

[32] Graham, R. L., G. M. Shipman, *et al*. "Open MPI: A High-Performance, Heterogeneous MPI", *Proc. IEEE International Conference on Cluster Computing*, 2006.

[33] Gropp, W., E. Lusk, *et al*. "A high-performance, portable implementation of the MPI message passing interface standard." Parallel Computing 22(6): 789-828, 1996.

[34] Robinson, J., S. H. Russ, *et al*. "A task migration implementation of the Message-Passing Interface", *Proceedings of 5th IEEE International Symposium on High Performance Distributed Computing*, 1996.

[35] Wolf, W. "Multiprocessor Systems-on-Chips", *Proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, IEEE Computer Society**:** 4, 2006.

[36] Barry Kauler, "A Tiny Microcontroller Dataflow Kernel", *Microprocessors and Microsystems*, Vol. 20, No. 2, 1996, pp. 105-110.

[37] Kuljeet Singh, "Design and Evaluation of an Embedded Real-Time Micro-Kernel", CPES, Blacksburg, Virginia, October 2002.

[38] Youngsoo Kim and Suleyman. Sair, "Designing Real-Time H.264 Decoders with Dataflow architectures. In *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pp. 291-296, 2005.