Multithreaded Coprocessor Interface for Multi-Core Multimedia SoC

Shih-Hao Ou, *Tay-Jyi Lin*, *Xiang Sheng Deng*, *Zhi Hong Zhuo*, and *Chih Wei Liu*

Department of Electronics Engineering
National Chiao Tung University, Taiwan

Abstract –Modern architectures exploit task level parallelism to improve their performance in a cost-effective manner. However, task synchronization and management is time consuming and wastes computing resources especially on application-specific architectures, such as DSP. In this paper, we propose a smart coprocessor interface that helps to offload the task management job from MPU or DSP. In our simulations, our approach can improve the overall performance of a dual-core platform by 57%. The hardware overhead of the interface is only 1.56% of the DSP core.

I. Introduction

Due to task divergence in most embedded systems, heterogeneous dual-core/multi-core SoC, i.e. MPU + DSP, is accepted as a cost-effective solution for the increasing computation demands in mobile media applications. TI OMAP [1], for example, is one popular dual-core platform, where the DSP, as the slave, performs the computation intensive task sent and requested by the host processor (i.e. RISC). On the other hand, as the complexity of the target applications on dual-(multi-)core processors grows rapidly, the DSP is likely needed by many tasks concurrently. The DSP architects tend to explore more parallelism among tasks to make the architecture more efficient [2]. Thus, DSP task management is required. Conventionally, DSP task management can be done on DSP itself (with an OS or a kernel) or on MPU (as a device driver) respectively. Nevertheless, the former suffers from inefficient processing of intensive programs flow and interrupt handling by utilizing the DSP. Furthermore, the significant context switch overheads and idle DSP-specific functional-units of the DSP cores make this solution infeasible. Fig. 1 depicts the TI OMAP platform where a real-time kernel called DSP/BIOS [3] incorporated with the DSP/BIOS Link (or Linux DSP Gateway) [4] provides software layer abstraction on DSP and complete IPC (Inter-processor communication) mechanism. On the contrary, although the MPU is suitable for task management, the frequent interaction between the MPU and the DSP will significantly affects the DSP utilization due to slow MPU response time incurred by the thick OS layers.

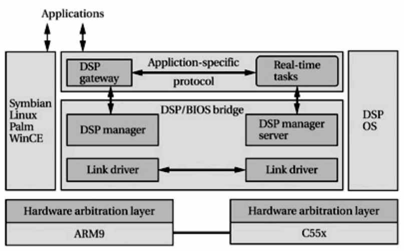

Fig. 1 TI OMAP software hierarchy

In this paper, an intelligent host processor interface (HPI) with dynamic task management capability is proposed to be a hardwired, dedicated controller for task management offloaded from either the DSP or the MPU. The proposed HPI is able to achieve more hardware efficiency and quick response time compared to the DSP or the MPU respectively. Besides, the task loading mechanism of the proposed HPI allows instant task initialization and thus the complexity is greatly reduced. Fig. 2 illustrates the system blocks of a typical dual-core processor incorporated with our proposed HPI. The proposed HPI can be seen as an agent of the DSP subsystem. The HPI is not only responsible for communication between the MPU and the DSP but also schedules those tasks distributed from the MPU to the DSP. Due to the highly optimized and simplified microarchitecture of the proposed HPI, the area cost of the proposed HPI, compared to a classic 5-stage pipelined multithreaded DSP core with 4 threads, increases only by 1.56%. Furthermore in our evaluation, the proposed HPI reduces the total execution time of a JPEG encoding task (to encode a 256x256 image) distributed from the MPU to the DSP from 42,826µs (task management on MPU) and 27,293µs (task management on DSP) to 18,412µs respectively.

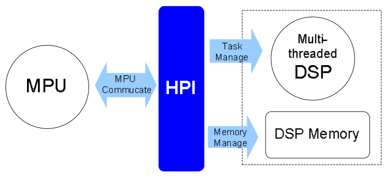

Fig. 2 A typical dual-core processor with the proposed HPI

This paper is organized as follows. Section II details the proposed HPI design and the performance evaluation is available in Section III. Section IV and V describe the implementation and a brief conclusion.

II. Proposed Design

The proposed HPI consists of three interactive controllers (FSMs) including the MPU manager, the DSP manager, and the task manager as depicted in Fig. 3. Communication among those controllers is through command queues. The MPU manager takes care of commands such as data push/pull from the MPU or the DSP. The DSP manager will receive commands from the MPU or the DSP. Some commands such as "free memory page request/release" are handled by the DSP manager itself while others like "dequeue/enqueue" are bypassed to the task manager. The main function of the task manager is to monitor all task status and schedule any ready-to-run task to idle DSP thread for execution.

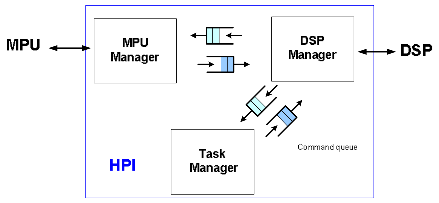

Fig. 3 HPI block diagram

Before going into the details of the proposed coprocessor interface, the data structure used for task management is explained first. Take a JPEG encoding application for example. First, the JPEG encoding task is partitioned into 4 subtasks including the color space transform (CST), the discrete cosine transform (DCT), the quantization (Q), and the variable-length coding (VLC). A unique priority is assigned to each task in descending order from the VLC to the CST, i.e. the VLC has the highest priority while the CST has the lowest priority. The HPI uses two tables depicted in Fig. 4 and Fig. 5 respectively to maintain task and DSP-thread status. The first table is the priority queue where tasks are recorded in descending order. The higher priority of the task means the earlier scheduled-to-run. Besides, the priority queue records the child tasks (destination-entry in priority queue), the number of the available tasks (data sets) and the program address for each task. The other table is the dispatch table which keeps the DSP-thread status. For each DSP-thread, there are two entries recorded. The one is enable signal and the other is a pointer to the corresponding task in priority queue. If there is any idle DSP-thread (enable signal = 0), the HPI checks the priority queue to see if any ready-to-run task (according to the priority and the number of the available data sets) exists. Once the HPI finds an idle DSP-thread and any ready-to-run task, it sets the corresponding enable signal (= 1) in the dispatch table, and writes the address (pointer) of the selected ready-to-run task in priority queue to the dispatch table. In the idle mode, the DSP-thread will cease PC (program counter) counting and the PC points to the NOP instruction. After the enable signal is set by the HPI, the corresponding DSP-thread will start incrementing the PC and execute the initialization code segments like a bootstrape to set up for task execution. Therefore, what the HPI to do for activating a DSP thread is only to set the corresponding enable signal and thus the complexity of the HPI is greatly reduced.

	Type	Destination	Queue pointer		Program address
			Head	Tail	
TCB[0] (Output)	OUT	-	0	0	&Output
TCB[1] (VLC)	G	Output	1	0	&VLC
TCB[2] (Q)	G	VLC	1	1	&Q
TCB[3] (DCT)	G	Q	2	1	&DCT
TCB[4] (CST)	G	DCT	4	2	&CST
TCB[5] (Input)	IN	CST	-	-	&Input

Fig. 4 Priority queue

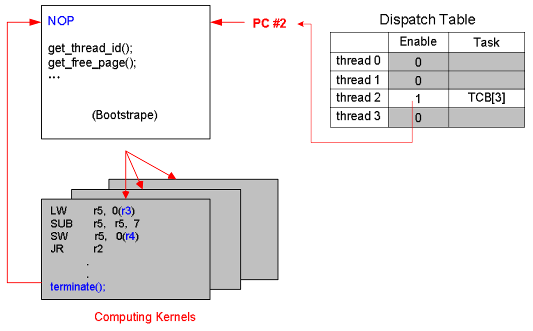

Fig. 5 Task loading mechanism

III. Performance Evaluation

We use the CoWare ESL simulation platform and refer to the TI DaVanci processor to construct a dual-core processor and conduct a JPEG encoding experiment to evaluate the proposed HPI. The target application is 256x256 Lena image JPEG encoding as described in Section II. The simulated dual-core processor is composed of an ARM926 (as MPU) and a multithreaded classic 5-stage pipelined DSP core with 4 threads running at 250MHz and 500MHz respectively. The communication between the ARM and the DSP is through the mailbox (interrupt-driven) and the interconnection AMBA AHB runs at 116MHz. The experiment is to simulate the case that some user application on MPU calls a JPEG encoding function call which will need the DSP acceleration. For the task management on MPU, the MPU where a uClinux is running is responsible for slicing the JPEG encoding tasks into 4 sub-tasks as aforementioned and then distributes each subtask to the DSP. On the other hand, for the task management on DSP, the DSP where a uC-OS II is running will take care of task slicing and scheduling. Finally, our proposed solution uses the hardwired HPI for the task management without any OS or kernel needed. We measure the total execution time of the JPEG encoding tasks of the three simulation conditions. Table 1 gives the performance evaluation. For task management on MPU, due to the thick OS layer with large context switch overhead, the kernel response time plus the IPC overhead contributes almost 54% of the total execution time. On the other hand, for task management on DSP, the kernel response time plus the IPC overhead has been reduced from 23,040µs to 4,609µs compared to the MPU due to lightweight kernel on DSP and reduced interactions between the MPU and the DSP. However, the time spent on thread management increases from 1,430µs to 4,328µs. The reasons are two folded: the one is due to inefficient handling of control-oriented task management by DSP and the other is due to interleaved multithreaded DSP. Finally, for the task management on the proposed HPI, the kernel response time is actually 0 and the IPC overhead is 1µs. Most importantly, the thread management costs only 55µs and the total execution time takes only 18,412µs.

Table 1 Performance evaluation

	MPU	DSP	**HPI**
Data movement	7,963	7,963	**7,963**
Thread management	1,430	4,328	**55**
Kernel response time + IPC overhead	23,040	4,609	**1**
Effective computation	10,393	10,393	**10,393**
Total execution time	42,826	27,293	**18,412**

unit: µs

IV. Implementation

This section gives the FPGA prototyping and cell-based design. First, the complete cell-based design flow has been conducted for the proposed HPI. Table 2 shows the synthesis results. The area cost of the proposed HPI is only 1.56% (6,524 um^2) of that of the multithreaded DSP core. At the same time, the ARM Versatile platform is used to implement the dual-core processor prototype. We utilize the on-board ARM926EJ-S core as host processor and develop a multithreaded DSP core with the proposed HPI on the Xilinx Virtex II 6000 FPGA. The default operating frequency of the ARM, the DSP and the AMBA AHB bus is 210MHz, 35HMz, and 35MHz respectively. The JPEG encoding has been successfully ported on the dual-core platform.

Table 2 Synthesis results

Technology	TSMC 0.13um CMOS cell-library
Frequency (MHz)	125
Area (um^2)	6524

V. Conclusion

This paper presents an intelligent coprocessor interface with dynamic task management capability. Compared to the conventional approach to task management (either on DSP itself or on MPU), the proposed hardwired HPI improves the hardware efficiency and exhibits quick response time. In our performance evaluation, the execution time of the JPEG encoding (of a 256x256 image) can be reduced from 42,826µs (with task management on MPU) and 27,293µs (with task management on DSP) to 18,412µs (with task management on HPI). Furthermore, the area cost of the proposed HPI is only 1.56% of the DSP core due to the highly optimized and simplified architecture design. As the number of integrated cores in popular multi-core design increases rapidly, the proposed hardwired HPI can be seen as an efficient solution to more complex task management approach for future multi-core architecture.

References

[1] *OMAP5910 Dual Core Processor – Technical Reference Manual*, Texas Instruments, Jan. 2003
[2] D. W. Wall, “Limits of instruction-level parallelism,” in Proc. ASPLOS-IV, pp. 176-188, April 1991,
[3] DSP/BIOS Kernel Technical Overview, Texas Instruments. Available online: http://focus.ti.com/lit/an/spra780/spra780.pdf
[4] Linux DSP Gateway Specification, rev. 3.3.1, Nokia, Sep. 2006. Available online: http://dspgateway.sourceforge.net/pub/3.3.1/DSP_Gateway331_spec_a.pdf