Hardware implementation of inter-processor communication in MPSoCs for multimedia applications

Moonmo Koo*, Soo-Ik Chae**
School of Electrical Engineering and Computer Sciences
Seoul National University, Seoul, Korea
*jeffley@soc.snu.ac.kr, **chae@sdgroup.snu.ac.kr

Abstract – *In this paper we present a scalable and flexible architecture that implements inter-processor communication (IPC) synchronization among FIFO channels for multimedia applications. We also compare it to the simple mail-box architecture, especially for tasks of finer granularity. With experimental results we confirmed the proposed architecture is suitable for various cases including a Motion JPEG example.*

1 Introduction

Now various multimedia products such as high-definition TVs, set-top boxes, 3D game players, digital camcorders, and digital cameras are prevailing. Their architectures are increasingly required to be open and flexible to accommodate more functions and multiple standards. Therefore, employing multiple processors in designing complex SoCs became a viable alternative to meet tight time-to-market by maximizing design reuse and providing flexibility [1]. In MPSoCs, however, an efficient IPC mechanism should be provided. Multimedia applications have parallelism at various levels of granularity, which can easily be modeled by a Kahn process network (KPN). The main purpose of this paper is to find a suitable hardware architecture based on FIFO channels for IPC synchronization in MPSOCs for multimedia applications.

2 Previous Work

A mail-box has widely been adopted in many implementations because of its simplicity in hardware architecture, which is a reasonable solution when the granularity of a task is coarse (e.g. synchronization per frame). It is shown in [3] that the combination of mail-boxes and OS synchronization primitives is sufficient in designing an MPEG-4 video encoder with its communication on the per-frame basis. In the Philips's approach [2] and in the SoCBase-DE [4], a FIFO channel controller, which manages the tokens for put/get operations, is employed for enhancing the performance and reduction of the synchronization overhead. However, a dedicated hardware FIFO should be instantiated for each channel, which increases the hardware overhead.

The objective of our work is to propose an efficient hardware accelerator for IPC synchronization that is flexible and scalable, which is suitable to complex multi-processor systems for multimedia streaming applications. In section 3, after describing a simple mail-box hardware solution for IPC and its limitations, we propose a new IPC hardware architecture to alleviate the problem. In section 4, experiment results are presented, which is followed by the conclusion and future work in section 5.

3 New IPC architecture

3.1 A simple mail-box

Because only storing the messages to be sent and their signaling are supported with a mail-box, software should transport the messages and protect the FIFO control data. As granularity of tasks is finer, their synchronization overhead gets higher while their synchronization buffers are small enough to be allocated in on-chip memory. Moreover, using tasks of finer granularity enable us to find more parallelism easily, which is more flexible and cost-effective in mapping the tasks on an MPSoC architecture.

Table 1 shows the synchronization overhead in message token passing for MJPEG decoding of QCIF 10 frames. For a solution with three CPUs, the overhead of sending and receiving message tokens on the 8x8 block basis, is more than 65% in our measurement, and the performance is degraded to a half to that of the solution with a single CPU. To solve this problem by reducing SW overhead at the finer granularity execution, we must employ dedicated FIFO channels.

Communication unit	Synchronization overhead	# of msg. transport	Perf. Improv. due to 3 CPUs
Frame	2.75 %	30	46.7 %
8x8 block	**65.60 %**	5940	**-98.5 %**

Table 1. Synchronization overhead for a simple mail-box

3.2 Our proposed IPC architecture

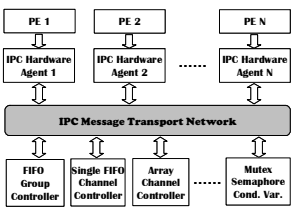

Figure 1. The overall architecture

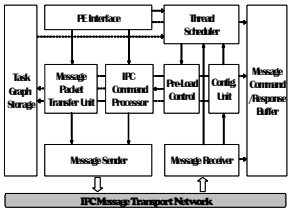

Figure 2. IPC hardware agent

In implementing a FIFO channel and its associated PE interfaces, a centralized structure is not scalable although it can reduce latency and internal synchronization overhead. Therefore, we propose a distributed architecture, shown in Fig.1 for scalability, which is composed of flexible *IPC hardware agent* (Fig.2) that interface PEs, and the *FIFO grouped channel controller* (Fig.3) that configures the number of channels. The IPC hardware agent accesses FIFO channels by processing **thread-level commands** on behalf of its corresponding PE. Multiple channel controllers can be integrated in this framework.

Once connectivity among the tasks is established, the tasks can be executed in parallel with PEs to get tokens to or from FIFO channels to push or pop. This functionality of connectivity **pre-loading** is implemented with low hardware overhead in the IPC hardware agent.

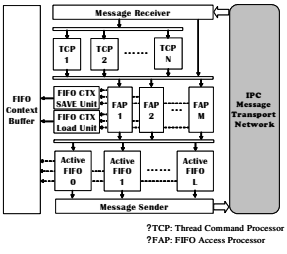

Figure 3. FIFO grouped channel

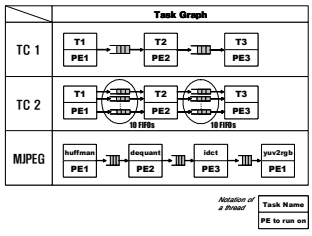

Table 2. Test scenarios

A block diagram for a grouped FIFO that can be shared with multiple FIFO channel controllers is shown in Fig.3. An *active FIFO* means one that is currently accessed, which can be used by another logical FIFO. The context of a FIFO channel is to be saved into or restored from the *FIFO context buffer*. The maximum size of this buffer limits the number of FIFO channels to be shared together in a grouped FIFO. The *FIFO Access Processor (FAP)* finds an active FIFO, or makes a request to save or restore a proper active FIFO. The *Thread Command Processor (TCP)* is responsible for accepting a thread command request and issuing FIFO

channel access commands to FAPs one by one. If any event for an active FIFO is occurred (e.g. the condition which is able to put or get, is changed), or there is a need to return responses to a IPC hardware agent, these messages are transported via the dedicated *IPC Message Transport Network*.

3.3 Implementation result

Table 3 shows synthesis results for the component previous mentioned. Each component consists of several functional sub-blocks to enhance its flexibility and extensibility in implementing a wide range of applications. These results are obtained with 0.18 um technology assuming that clock frequency is lower than 100MHz, which excludes the on-chip buffers such as the command/response buffers, the FIFO context buffers. The hardware overhead for implementing the thread-level command is substantial, while the area overhead of the pre-load functionality is negligible.

	Configuration		Gate Counts
IPC Hardware Agent	Thread-Level Command	Pre-Load Functionality	
	O	O	14.6 K
	O	X	14.5 K
	X	N/A	8.1 K
	Configuration		Gate Counts
IPC FIFO Group	TCP #	FAP #	
	1	1	33.1 K
	3	2	40.0 K
	3	4	42.5 K
	N/A	1	26.6 K

Table 3. Synthesis results of the IPC hardware agent and the IPC grouped FIFO

4 Experimental Results

To confirm feasibility of the scheme proposed in this paper, several experiments for Motion JPEG were performed on three test scenarios (Table 2) to represent the pipelined execution pattern which is common in the streaming multimedia applications. A FPGA prototyping board was used in our experiments, where three ARM7TDMI soft-cores and the proposed IPC architecture that has three IPC hardware agents and one FIFO group are configured.

4.1 Test scenarios

In TC1, all the tasks communicate with other tasks via a single FIFO channel. In TC2, the synchronization overhead is substantial because 10 FIFO channels are used in each connection. The average computation granularity of MJPEG is approximately **10,000** cycles, and the maximal parallelism ignoring synchronization overhead can be achieved is **1.89** (the same to **47%** cycles counts reduction) in the task partition shown in Table 3 (huffman, dequant, idct, yuv2rgb).

4.2 Synchronization overhead

Testbench	Comp. granularity	Sync. overhead (%)	**pre-load** sync. overhead reduction	**thread-level command** sync. cost reduction
TC1	938 cycles	7.8 %	23.2 %	62.3 %
	3928 cycles	**2.0 %**	23.1 %	62.2 %
	9127 cycles	0.9 %	23.1 %	62.1 %
TC2	938 cycles	56.2 %	16.1 %	60.4 %
	3928 cycles	24.3 %	12.0 %	60.3 %
	9128 cycles	**12.5 %**	9.3 %	60.3 %

Table 4. Synchronization overhead and the effects of each function

As shown above, the synchronization overhead is lowered under 2.0 % in the TC1 case when computation granularity becomes larger than 4000 cycles. The TC2, in spite of massive synchronization, the overhead of synchronization approaches to 10% as the granularity is coming up to 10,000. The overhead reduction of hardware acceleration of the thread-level command, is more than 60%, so it can be justified when the hardware overhead shown in Table 3, is acceptable.

4.3 Motion JPEG running profiles on 3 CPUs

Testbench Desc.	# of CPUs	yuv2rgb & Display	Cent. vs. Dist.	Cycle Counts	# of Thread Switch	Cycles reduction
※ MJPEG QCIF 10 Frames, ※ Sync. per 8x8 block basis ※ Round-Robin Thread Scheduling ※ 32 FIFO depth	1	off	N/A	109.8 M	N/A	N/A
	1	on	N/A	132.4 M	N/A	N/A
	3	off	C	58.5 M	1587	46.7 %
	3	**off**	**D**	**62.6 M**	**1530**	**43.0 %**
	3	on	C	77.6 M	1566	41.4 %
	3	**on**	**D**	**78.9 M**	**1676**	**40.4 %**

Table 5. The Motion JPEG execution profile data on 3 CPUs

The last column of Table 5 means the reduction of cycle counts compared to that of the execution on one CPU. Inclusion or exclusion of the yuv2rgb block and display has been also considered, because the yuv2rgb block in the current implementation is done on the frame basis so as to disturb the parallel execution. In spite of the inefficiency at the expense of the implementation of the distributed architecture, the performance is comparable to the centralized one and the ideal case, with the help of the novel architectural functionality such as the thread-level command and the pre-load capability. It is difficult to extract the pure synchronization cycles involved, because the thread scheduling and context switch overhead under a conventional RTOS are unavoidable and the schedule of multi-threads is not optimal. Moreover, these effects of the RTOS make the analysis much more complicated. Therefore, we will implement a hardware accelerator for a RTOS kernels closely related to the multi-thread execution in the near future.

5 Conclusion and Future Work

In this paper, we proposed a new IPC architecture that is efficient especially for more frequent synchronization among smaller tasks, compared to a conventional mail-box approach. To provide scalability and flexibility, we employed a distributed architecture that can easily be adapted to requirements of each channel and its corresponding threads. Both an IPC hardware agent and a grouped FIFO can be configured dynamically or statically. From the area estimation and experimental results for three test scenarios including MJPEG, we confirmed that the propose architecture is effective, compared to the mailbox approach. We plan to integrate the proposed IPC accelerator into SoCBase-DE [4] developed in Seoul National University, which is a refinement-based SoC design environment that covers various abstraction levels from transaction level to register transfer level, and provides mixed-level simulation for incremental refinement.

References

[1] Grant Martin, "Overview of the MPSoC Design Challenge", *Proc. of 43rd Design Automation Conference(DAC)*, San Francisco, July 2006.

[2] Gangwal, O. P., A. K. Nieuwland, and P. E. R. Lippens, "A scalable and flexible data synchronization scheme for embedded HW-SW shared-memory systems", *Proceeding of the International Symposium on System Synthesis*, pp.1-6, October, 2001

[3] M-W. Youssef et al., " Debugging HW/SW Interface for MPSoC: Video Encoder System Design Case Study," *Proc. 41st Design Automation Conf.* (DAC 04), IEEE CS Press, 2004, pp.909-913

[4] Sanggyu Park, Sangyong Yoon, Soo-Ik Chae, "A Mixed-Level Virtual Prototyping Environment for Refinement-based Design Environment", *Asia and South Pacific Design Automation Conference*, pages 588-593, January, 2006