An Efficient Architectural Design of Hardware Interface for Heterogeneous Multi-core System

Xiongli Gu[1], Jie Yang[1], Xiamin Wu[2], Chunming Huang[1], Peng Liu[1,1]

Department of Information Science and Electronic Engineering,

[1]Zhejiang University, Hangzhou, 310027, China

[2]UTStarcom Co.Ltd., Hangzhou, 310053, China

{guxiongli2010, mckeyyang, alonewoo}@gmail.com,

hcm198611@yahoo.com.cn,liupeng@zju.edu.cn

Abstract: How to manage the message passing among inter processor cores with lower overhead is a great challenge when the multi-core system is the contemporary solution to satisfy high performance and low energy demands in general and embedded computing domains. Generally speaking, the networks-on-chip connects the distributed multi-core system. It takes charge of message passing which including data and synchronization message among cores. The size of most data transmission is typically large enough that it remains strongly bandwidth-bound. The synchronization message is very small which is primarily latency bound. Thus the separated networks-on-chip are needed to transmit the above two types of message. In this paper we focus on the network for the transmission of synchronization messages. A hardware module – message passing unit (MPU) is proposed to manage the synchronization message passing for the heterogeneous multi-core system. Compared with the original single network approach, this solution reduces the run-time object scheduling and synchronization overhead effectively, thereby, improving the whole system performance.

Keywords: data flow graph (DFG); multi-core system; parallel programming.

1. Introduction

Nowadays multi-core system becomes a popular solution for obtaining higher performance, short developing period, and low cost in the application system design. The multi-core system has its advantages but it also brings new problems: the run-time concurrency and synchronization among tasks are crucial if the high system performance is pursued [1].

We use the data flow graph (DFG) programming model to guide the parallel programming [2] in our work. The compiled inter-processor tasks are statically

[1] The corresponding author, liupeng@zju.edu.cn

mapped to the distributed processors and the synchronization message passing among inter processors are needed for the parallel running of tasks on different processors. Usually the size of synchronization message is small which should be transmitted among processors with lower latency. But the traditional networks-on-chip is usually designed for the data transmission and is not efficient to manage the synchronization message passing among inter cores.

In this paper we suggest a hardware module called message passing unit (MPU) which transmits the scheduling and synchronization message for the lightweight distributed multi-core system without shared memory. The aim of this work is to reduce the inter processor communication overhead. We consider that NoC should not only play the role of data transmission but also help to manage part of the application scheduling and synchronization work for the multi-core system. The constructs in the application program for handling coordination and synchronization between the threads are transferred to the control signals that shall be sent and/or received by the processors during runtime. The control signals are short and they should be transmitted fast with less overhead. If they are transmitted in the same network as data the average overhead is expensive which is usually unacceptable. So in the NoC design space we specify two sub layers in the link/network layer: the data transmission sub layer manages the data communication. The control sub layer - MPU manages the control signal flow and messages passing for handling the coordination and synchronization among threads which is efficient for the heterogeneous multi-core system.

The rest of the paper is organized as follows: Section 2 gives an overview of related work. Section 3 explains our MPU proposal. Section 4 describes the implementation details of object scheduling and synchronization flow. Section 5 discusses the evaluation methodology and gives the evaluated results. Finally conclusions are made.

2. Related work

In the literature, there are some prior work have been done to accelerate the synchronization message passing among inter cores. In [3], the MultiFlex uses the object request broker (ORB) to connect client task to the server task. The aim is to accelerate the scheduling and synchronization message passing among tasks. In [4], the MLCA adopts universal register file (URF) to exchange the scheduling and synchronization message passing among inter-core tasks. But the ORB and URF have the same problem: they transmit the scheduling and synchronization message in the general NoC. The general NoC is designed for data transmission among inter cores, which usually has large bit-width and the complicated protocol to assure the correct data transmission. As a result, the complicated protocol will bring overheads. Thus the traditional NoC is not efficient to manage the small size synchronization message passing among inter-core tasks. In [5], DMA based message passing mechanism is applied in Cheng's work to transmit the synchronization messages among inter-core tasks which is also inefficient for the transmission of synchronization message.

3. Construct of MPU

3.1 Application model

Before describing the proposed MPU, the application model is firstly introduced. The tasks of a data-driven workload can be modeled as a data flow graph.

Definition 1: A *data flow graph* (*DFG*) [6] is a directed graph DFG($\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{D}$), where each node $n_i \in \boldsymbol{V}$ represents a task and a directed edge $e_k=(n_i, n_j)\in\boldsymbol{E}$ represents the communication between nodes n_i and $n_{j.}$.

The tasks are mapped onto processors statically at compile time. All tasks are executed in a self-timed manner [7] as follows: a task can be invoked if (1) it receives the data from all its predecessors and (2) its output data buffers are valid. The above two operations are based on message passing mechanism. So the MPU should manage the message passing for these two operations of message passing architecture. The tasks are executed atomically and in this paper we define the atomic task as object.

3.2 MPU structure

A hardware/software implementation for the object scheduling, synchronization and data transmission between objects is proposed. Essentially the function of real-time operating system (RTOS) task/thread management is partly realized in hardware, which is named message passing unit (MPU) and situated in the link/network layer of NoC. As described in Section 1, MPU is also the control sub net of NoC, which takes advantage of the multi-core system's characteristics and transmits the control signals and messages for object scheduling and synchronization efficiently.

The task/thread management is the controller of the scheduler. The producer processor informs MPU that its object execution has been completed and then MPU wakes up the consumer objects to start processing the incoming data. This message passing mechanism realized in the control sub net guides the scheduling and synchronization of the different objects. The DMA tasks are also set by MPU to transfer the data between the producer and consumer objects.

3.2.1 MPU hardware architecture

The hardware module of MPU supports four functions, (1)receive and response to the producer object information, (2)wake up the available consumer objects for execution, (3)synchronize the different objects execution, and (4)initialize the DMA tasks for data transmission between the different objects. The hardware block diagram of MPU is illustrated in Fig. 1, which consists of four parts: an object score board, a wakeup logic, an object program counter (PC) array, and a DMA task parameters buffer.

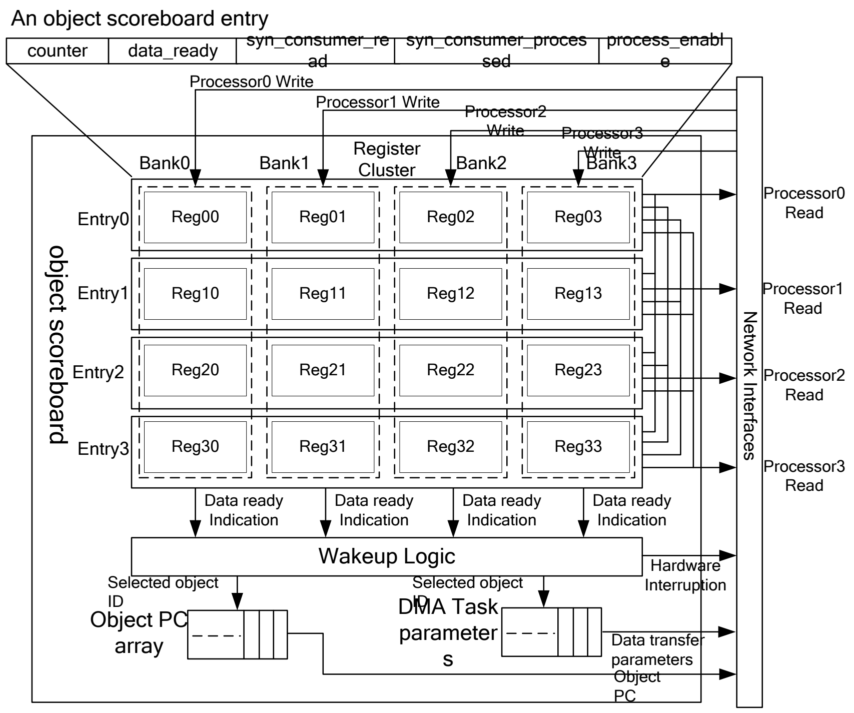

Fig.1. Block diagram of MPU hardware

(1) The object scoreboard records the status of objects which are running or pending in the multi-core system. It is the kernel part for message interchange. Each processor in the multi-core system has one corresponding entry (in Fig. 1 it is supposed that the system has four processors) in the object scoreboard, which records the status of the

objects assigned to this processor. The length of entry is decided by the maximum number of objects that can be mapped to the processor. Each processor or object can write data in one bank of register cluster, and each processor (or object) can read every entry of the register cluster. So the message passing among tasks are realized in this way. Each entry includes five fields: counter, data_ready, syn_consumer_read, syn_consumer_processed, and process_enable.

i) Counter field

The counter field records the input data port number of the objects allocated on this processor.

ii) Data_ready field

This field is set by the related producer objects allocated in the other processors, which is used to inform the consumer objects that the input data has been ready.

iii) Syn_consumer_read field

The syn_consumer_read field is designed to realize the synchronization for data reading.

iv) Syn_consumer_processed field

This field is served as the synchronization point to assure the synergistic working of the producer and consumer objects pair in the multi-core system.

v) Process_enable field

In our execution model the object should be executed atomically. When one object is in its execution, the other objects cannot interrupt it. This is realized by tag set in the process_enable field.

(2) The wakeup logic selects the available objects for execution.

(3) The object program counter array records the starting addresses of objects.

(4) The DMA task parameter buffer is used to store the DMA operation parameters which the global memory address, the local memory address, the data length and the data format are set in its entries. Directors call for the DMA operation when the input data is ready.

3.2.2 Operating system interface for MPU

We design four functions in RTOS for MPU to manage the task/thread running of multi-core system. These functions include *Write_MPU()*, *Syn_consumer_read()*, *Syn_consumer_processed()*, and *Check_MPU()*.

(1) The *Write_MPU* function is used by the director of producer object to set and clear its consumer object’s data ready fields in the entry of object scoreboard.

(2)The *Syn_consumer_read* function is called by the director of consumer object to set its syn_consumer_read field of the object scoreboard.

(3)The *Syn_consumer_processed* function is used by the director of consumer object to clear its syn_consumer_processed field.

(4)The *Check_MPU* function is applied to check the object status in an entry of object scoreboard.

These four functions constitute main part of the task/thread management of RTOS, which handles the objects operation in the multi-core system.

4. Object scheduling and synchronization flow

The scheduling and synergistic synchronization management for the running of the parallel program are partly performed by the hardware MPU and partly by software - the director in RTOS. The hardware MPU manages the object waken-up and part of the operation of synergistic synchronization for the objects execution in the multi-core system. After the object scoreboard entries have been set by the producer object which completed its execution, the DMA operation will be called first to transfer the produced output data to the local memories of the consumer objects. Then the consumer objects which have all the input data ready can be selected for execution by the wakeup logic. The whole flow includes the following steps.

(1) The syn_consumer_processed bit becomes unavailable after the data ready bit has been set. The MPU sends the DMA operation interruption signal to the corresponding consumer object director to initiate the data transfer between the producer and consumer objects. When the DMA operation is over, the director of consumer object will set syn_consumer_read bit and decrease the number by one in the corresponding counter. Later the data ready bit will be cleared by the producer object.

(2) The wakeup logic scans the counters to find out the ready objects for execution. If the processor of the ready object is free (process_enable bit is available) the highest priority ready object will be activated, MPU sends execution interruption signal to the corresponding processor. Then this processor fetches the program from the address indicated by the PC array.

(3) After the scheduled object finishes its execution the result data will be written to the global memory and its consumer objects' entries in the object score board will be set if they have completely processed the former produced data (all the syn_consumer_processed bits of its consumer objects are cleared). Then this producer object's syn_consumer_processed bit is cleared, process_enable is reset to valid and

the counter is reset to its input data number by hardware.

Thus it means that the parallel program has finished its execution for one batch of its input data, the process mentioned above will be repeated cyclically for the successive batches of the input data. The MPU scheduling flow is depicted in Fig. 2.

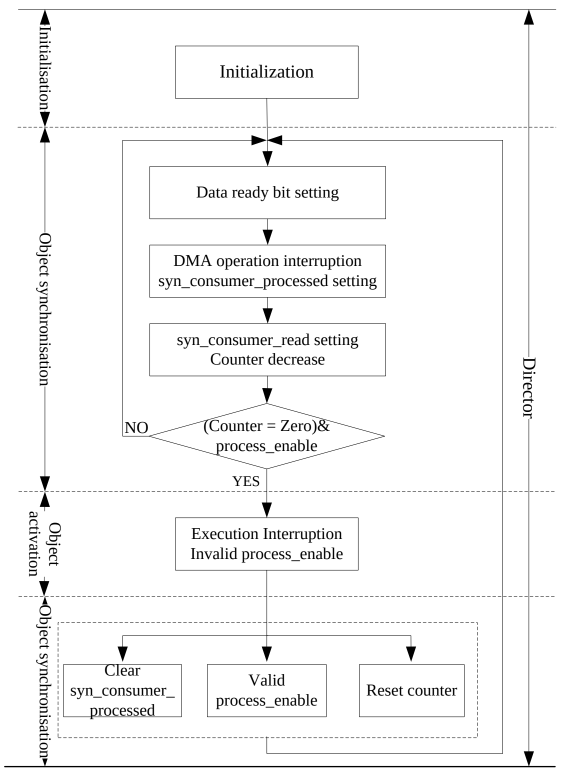

Fig.2. MPU scheduling and synchronization flow

5. Experimental results

5.1 Evaluation methodology

In order to evaluate the proposed MPU performance two applications – FFT program [8] and eigenvalue of matrix – Jacobi algorithm [9] program are adopted in the experiment. The experimental platform integrates one 32-bit integer RISC core - RISC32E [10] and eight 32-bit integer DSPs - MediaDSP3200 [11-12]. A 3x3 mesh topology NoC connects them and the DMA engine is also included. Each core has a local memory and the SDRAM is used as the global data buffer for the application. Each DSP has one allocated director and RTOS is running on the RISC processor. The

MPU and software directors (proxy of RTOS) manage the object scheduling and synchronization for this multi-core system.

5.2 The fast Fourier transform (FFT)

The fast Fourier transform (FFT) is the fast algorithm for DFT and one of the most popular algorithm is the Cooley-Turkey algorithm which is written as below:

$$X[k_1,k_2]=\sum_{n_2=0}^{N_2-1}W_{N_2}^{n_2k_2}\underbrace{\left(W_N^{n_2k_1}\underbrace{\sum_{n_1=0}^{N_1-1}x[n_1,n_2]W_{N_1}^{n_1k_1}}_{N_1\text{- point DFT transfer}}\right)}_{\bar{x}[n_2,k_1]}=\underbrace{\sum_{n_2=0}^{N_2-1}W_{N_2}^{n_2k_2}\bar{x}[n_2,k_1]}_{N_2\text{- point DFT transfer}},$$

$$n=N_2n_1+n_2\begin{cases}0\le n_1\le N_1-1\\0\le n_2\le N_2-1\end{cases},\ k=k_1+N_1k_2\begin{cases}0\le k_1\le N_1-1\\0\le k_2\le N_2-1\end{cases},\ N=N_1N_2\,.$$

In the experiment 64-point FFT is applied as the test algorithm. The 64-point FFT on platform is arranged as $N_1=N_2=8$ and Fig.3 shows its parallel programming process and synchronizations among objects.

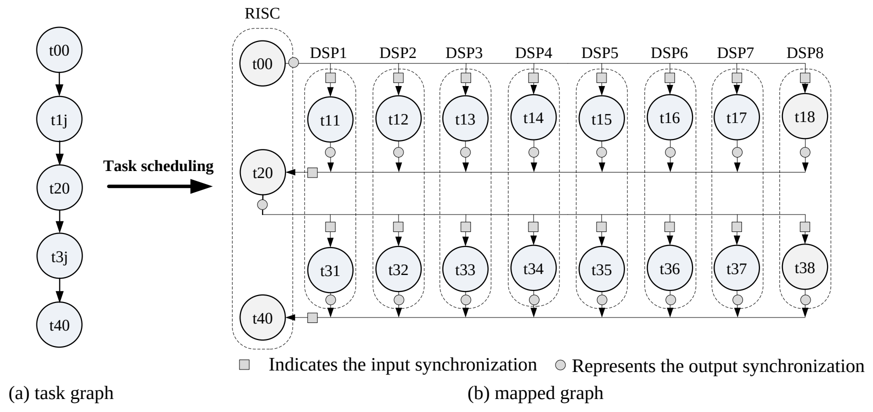

Fig. 3 Parallel programming process and synchronizations in FFT

1) The object t_{00} is the predecessor for the inter-processor objects t_{1j} (j=1…8), which calculates the indexes for original data. It is allocated to the control RISC.

2) The objects t_{1j} (j=1…8) calculate the inner summation algorithm of the FFT $\sum_{n_1=0}^{N_1-1}x[n_1,n_2]W_{N_1}^{n_1k_1}$ by using the FFT algorithm. The objects are allocated to DSPs.

3) The object t_{20} calculates the $W_N^{n_2 k_1}$ which multiplies by input data and finishes the data permutation operation from the order of $x[k_1,n_2]$ to $x[n_2,k_2]$. It is allocated to the control RISC.

4) The objects t_{3j} (j=1…8) calculate the summation algorithm of the FFT $\sum_{n_2=0}^{N_2-1} W_{N_2}^{n_2 k_2} \bar{x}[n_2, k_2]$ by using the FFT algorithm. The objects are allocated to DSPs.

5) The object t_{40} manages the output data permutation to the frequency domain parameter and is allocated in control RISC.

Table 3. FFT experimental result

Overhead / Scheduler	RTOS kernel on RISC(Kbytes)	Director on processor(Kbytes)	Kernel delay/ total execution time (cycles)
Software Director	10	1.884	6716/21299
MPU	9	0.584	1435/16018

The experimental results are shown in Table 3. From Table 3 we can see that the MPU solution has less memory space requirement for RTOS kernel and directors. It takes an average time consumption of 16018 cycles for one time 64-point FFT. The kernel delay for the scheduling and synchronization of MPU approach takes up 8.96% and for software director the percentage is 31.53%. Thus the MPU efficiently reduces the kernel delay for the multi-core system which improves the system efficiency by 24.79%.

5.3 Eigenvalue of matrix

The eigenvalue λ of matrix **A** is defined as $\mathbf{Au} = \lambda\mathbf{u}$, where **A** is an n x n matrix, λ is a real number and ***u*** is the n dimensional characteristic vector of matrix **A.** The Jacobi matrix algorithm is applied to solve this problem. In experiment a 64*64 matrix is applied as the test algorithm. The Fig. 4 shows its parallel programming process and the synchronizations among objects.

1) The object t_{00} assigns the matrix coefficients of **A** to the DSPs, which is the

predecessor of objects t_{1j} (j=1…8). It is allocated to control RISC.

2) The objects t_{1j} (j=1…8) divide the assigned coefficients into upper diagonal and lower diagonal classes. They are the predecessors of object t_{20} which are allocated to DSPs.

3) The thread t_{20} finds the maximum value from the collected biggest values, which is the predecessor of objects t_{60} and t_{3j}. It is allocated to control RISC.

4) The objects t_{3j} (j=1…8) calculate the tangent of rotating angle θ *of* hyper plane (g, h), cos θ and sin θ, which are the predecessors of objects t_{5j} (j=1…8) and are allocated to DSPs.

5) The object t_{40} broadcasts the matrix coefficients of row g and column h to the DSPs, which is the predecessor of objects t_{5j} (j=1…8). It is allocated to control RISC.

6) The objects t_{5j} (j=1…8) performs multiplication of the former result ($\mathbf{P_{gh}}^{(1)}$ $\mathbf{P_{gh}}^{(2)}$…$\mathbf{P_{gh}}^{(k-2)}$) by $\mathbf{P_{gh}}^{(k-1)}$, which are the predecessors of t_{1j} and are allocated to DSPs.

7) The object t_{60} get the final results, which is allocated to control RISC.

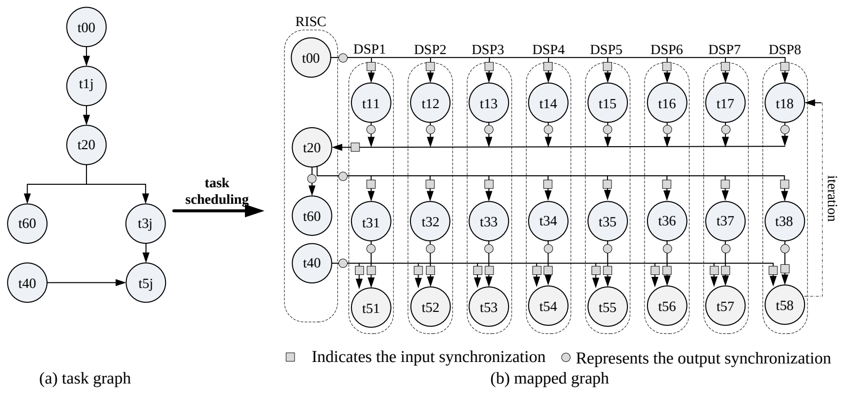

Fig.4 Parallel programming process and synchronizations in Jacobi matrix algorithm

Table 4. Jacobi matrix algorithm experimental result

Overhead / Scheduler	RTOS kernel on RISC(Kbytes)	Director on processor(Kbytes)	Kernel delay/ total execution time (cycles)
Software Director	10.298	2.182	15870/59182
MPU	9.494	0.974	5633/48945

The eigenvalue of matrix's result is shown in Table 4. Similarly, the memory space requirement of MPU is less than the software director’s. The one-time 64*64 Jacobi matrix algorithm calculation needs 48945 cycles so that the kernel delay of MPU approach takes up about 11.51% of the whole time consumption and for software director the percentage is 26.82%. Thus the MPU scheme reduces the kernel delay which improves the system efficiency by 17.30%.

5.4 Physical parameters of MPU

The hardware cost of MPU depends on the scale of multi-core system and the maximum number of threads/objects which are allocated on the processors. Here the scale of MPU is for nine processors multi-core system so that there are nine entries in the object scoreboard. For a limited number of threads exploited in the program it is supposed that each processor has no more than four threads/objects. In the multi-core system each object may have maximum eight producers and eight consumers so every object needs 27-bit for the entry. At the same time a 32-bit program counter and a 128-bit DMA task parameter buffer are needed for one object. So a 187-bit width hardware register is needed for one object in total. The 3x3 mesh topology NoC is used which the channel width is 64-bit and the depth of the FIFO is 8. We synthesize the MPU module and NoC module using Synopsys v-2003.6 and the TMSC90 CMOS process technology. The physical parameters of MPU and NoC are listed in Table 5. From Table 5 we can see that the MPU module can work at the NoC frequency and the area cost of MPU takes up 14.5% of the whole NoC area.

Table 5. Physical parameters of MPU and NoC (TSMC90: worst case, voltage 1.08v, temperature 125℃)

Parameter / Component	Delay (ns)	Frequency (MHz)	Gate count	Dynamic power (mW)
MPU	0.85	1176	97096	6.42
NoC	1.30	769	669634	80.82

6. Conclusion

The hardware/software approach – message passing unit interface offers an efficient way to manage the object scheduling and synchronization for the multi-core system to reduce the communication overhead. Compared with the software director, it not only cut down the average delay for object scheduling and synchronization but also reduces the code size for RTOS and director, which is usually important for the limited memory space of the embedded system. The physical parameters imply that the hardware MPU module has the characteristic of relatively higher frequency and small area. Compared with the software approach the MPU approach improves the system efficiency by 24.79% in FFT application and 17.30% in Jacobi matrix algorithm with the hardware area increase of 14.5%.

Acknowledgment

This work is supported by NSFC under grant 60873112.

References

1. Mignolet, J.-Y., Baert, R., Ashby, T.J., Avasare, P., Hye-On Jang, Jae Cheol Son: MPA: parallelizing an application onto a multicore platform made easy. IEEE MICRO, Vol.29, issue3, pp.31-39 , (2009).
2. Robert, G. and Babb,Ⅱ: Parallel processing with large grain data flow techniques, Computer, Vol.17, No.7, pp.55-61, (1984).
3. Paulin, P.G., Pilkington, C., Langevin, M., Bensoudane, E., Lyonnard, D., Benny, O., Lavigueur, B., Lo, D., Beltrame, G., Gagne, V., Nicolescu, G.: Parallel programming models for a multiprocessor SoC platform applied to networking and multimedia. IEEE Transactions on Very Large Scale Integration (VLSI) systems, Vol.14, No.7, pp.667-680, (2006).
4. Abdelrahman, T., Abdelkhalek, A., Aydonat, U. : The MLCA: a solution paradigm for parallel programmable SoCs. IEEE North-East Workshop on Circuits and Systems, pp.253-253, (2006).
5. Cheng, X.M., Yao, Y.B., Zhang, Y.X., Liu, P., Yao, Q.D.: An object oriented model scheduling for Media-SoC. Journal of Electronics (CHINA), Vol.26, No.2, pp. 244-251, (2009).
6. Sinnen O.: Task scheduling for parallel systems. Hoboken, New Jersey, John Wiley&Sons press, pp.60-1, (2007).
7. Bekooij M, Hoes R, Moreira O, Poplavko, P., Pastrnak, M., Mesman, B., Mol, J., Stuijk, S., Gheorghita, V., Meerbergen, J.V.: Dataflow analysis for real-time embedded multiprocessor system design. In: Dynamic and Robust Streaming in and between Connected Consumer-Electronic Device, Vol. 3. Dordrecht: Springer Netherlands, p. 81-108, (2006).
8. Cooley JW and John WT. An algorithm for the machine calculation of complex Fourier series. Mathematics of Computation; 19(90): 297–301, (1965).
9. Golub GH and van der Vorst HA. : Eigenvalue computation in the 20th century. Journal of Computational and Applied Mathematics, Vol.123, No.1/2, pp.35–65, (2000).
10. Xiao, Z.B., Liu, P., Yao, Y.B., Yao, Q.D.: Optimizing pipeline for a RISC processor with multimedia extension. ISA. Journal of Zhejiang University-Science A, Vol.7, No.2, pp.269–74, (2006).
11. Liu, P., Yao, Q.D., Li, D.X.: 32-bit media digital signal processor. China Patent ZL200410016753.8. (2004).
12. Shi, C., Wang, W.D., Zhou, L., Gao, L., Liu, P., Yao, Q.D.: 32b RISC/DSP media processor: MediaDSP3201. Embedded Processors for Multimedia and Communications II, SPIE Vol.5683, Jan.2005, pp.43-52.