Łukasz GOLLY
INSTITUTE OF ELECTRONICS, SILESIAN UNIVERSITY OF TECHNOLOGY,
Akademicka Street 16, 44-100 Gliwice, Poland

Multitask real-time architecture supporting predictable management and memory system

M.Sc. Łukasz GOLLY

Mr. Golly received the M.Sc. degree in electronics from Silesian University of Technology, where he continues his PhD studies. His first works concerned automated generation of symbolic functions. Since 2011 Mr. Golly has started to work on his PhD thesis that covers the field of system level design. Currently his area of interest covers multithreading, real-time systems, time predictable systems, scheduling in multi-threaded systems and timing analysis of highly predictable real-time systems.

e-mail: perk@poczta.onet.pl

Abstract

The paper presents an extension of interleave pipeline PRET architecture. The main emphasize is made on obtaining deterministic, time predicable data and a program memory system which cooperates with the pipeline processor. The idea is to pass control of all IO operations to a special unit called MACU (Memory Access Control Unit). Thread State Controller (TSC), Dynamic Interleave Controller (DIC) and MACU together with a pipeline processor are proposed, to deliver microarchitecture details, which deal with a number of hardware threads working in the system. The paper also discusses several arbitration policies in MACU and DIC. A few experiments are performed to show benefits of the proposed methodology.

Keywords: precision time model; pipeline interleave; system level design; simulation and modeling; real-time electronic embedded systems.

Zarządzanie oraz organizacja systemu pamięci wielozadaniowej przewidywalnej czasowo struktury czasu rzeczywistego

Streszczenie

Praca porusza tematykę przewidywalności czasowej systemów elektronicznych. Problem ten był obiektem badań różnych grup badaczy. Zdaniem autora największym osiągnięciem było przedstawienie architektury potokowej z przeplotem wątków sprzętowych. Główną cechą tego podejścia było usunięcie zjawiska hazardu danych i sterowania. Autor niniejszej pracy zwraca uwagę na kwestie wypracowania schematu zarządzania wielozadaniową architekturą przewidywalną czasowo oraz przewidywalnego czasowo systemu pamięci (pkt. nr 3). Pierwsza kwestia została rozwiązana dzięki zastosowaniu kontrolera kontekstu potoku (pkt. nr 4). Identyfikatory, pamięć stanów wątków (rys. 1), specjalne kolejki do przechowywania identyfikatorów (rys. nr 2) i danych (rys. nr 4) umożliwiły efektywne zarządzanie wątkami sprzętowymi. Zmiana kontekstu potoku (punkt nr 4., rys. 3 oraz 4) następuje, gdy wątek wykonuje rozkaz operacji z pamięcią (rys. nr 5). Dzięki temu podczas wykonywania operacji wejścia/wyjścia, inny wątek będzie mógł realizować swój program. Takie podejście pozwala na ukrycie detali związanych z dostępem do pamięci głównej (implementacja w module MACU) oraz zwiększenie wydajności systemu (pkt. nr 7.). Przedyskutowano także różne schematy arbitrażu przy wprowadzaniu wątków do potoku (algorytmy zaimplementowane w DIC) oraz dostępu do pamięci (algorytmy zaimplementowane w MACU) (pkt. nr 6). Wysunięte tezy potwierdzono wynikami eksperymentalnymi (pkt. nr 6).

Słowa kluczowe: Precyzyjne modele czasowe, przeplot wątków, projektowanie systemowe, symulacja i modelowanie, elektroniczne systemy wbudowane czasu rzeczywistego.

1. Introduction

Time predictability and system dependability are very important factors in modern electronic systems. These groups of architectures must precisely define their timing properties. The development of technology, especially RISC processor introduced many new problems like data and control hazards, caching, timing dependencies etc. Even a small change of cache or branch prediction algorithm can cause totally different timing of the entire system. If the system is applied in industrial, medical or security systems, the consequences could be very serious. It must be emphasized that PRET architectures should not sacrifice their performance for timing predictability. The time predictability paradigm is very important for the present and future real-time systems.

2. Related works

The idea of precision timed machines (PRET) was introduced by Edwards an Lee in [1]. They extended ISA with a special DEADLINE instruction. It was used to control elapsed time over defined code fragment. This inspired other authors [2, 3] to develop predictable hardware platforms. The technique which effectively solves data and control hazards in interleaved pipelines [4] was introduced and analyzed in [5, 6]. Some memory systems used in term of predictable timed machine have been defined [7, 8]. The first paper [7] proposes application of scratchpad memories (SPM) to avoid cache hit/miss problems. The latter one [8] considers architecture with DRAM predictable controller useful when the program does not fit to SPM. An interesting approach of PRET architecture with a dynamically rescheduled system (controller by Dynamic Interleave Controller of Threads - DICT) is presented in [9]. The considered system contains separate data memory for all hardware thread and the access to the global memory controlled by Memory Access Control Unit (MACU). The idea of DICT and MACU is borrowed in this paper. The state of art and future challenges for PRET architectures are introduced in [10].

3. Problem formulation

A. Management of multitask PRET architectures

The idea of predictable interleaved pipeline was only analyzed in the case of small number of working threads (equal to amount of pipeline stages). In real industrial systems a processor very often needs to handle many events (hardware threads). In this case it is very important to develop an architecture which could manage their status without scarifying its time predictable behavior.

B. Time predictable memory system architecture

When IO operation occurs the system becomes time unpredictable because memory transfer time cannot be clearly defined. It is necessary to add empty cycles (pipeline bubble) into the pipeline to wait for IO operation termination. There are also additional cases that hinder formulation of the time predictable memory system. The main of them are:

- cache policy algorithms - using caches can significantly reduce the memory access time but only when a cache hit occurs. Proper algorithm selection is crucial. Its slight modification totally ruins the architecture timing behavior.
- SPM – usage of scratchpad memories increases the system dependency but only if the thread fits into dedicated memory region. If it is not the case, the data must be transferred from the slow main memory to fast SPM [10].

4. Platform structure

Taking into account the above problems a special PRET platform (Fig. 1.) was designed. It implements mechanisms that serve multitask organization of hardware threads. A special module called Thread State Controller (TSC) is responsible for switching pipeline contents in discrete moments of time. Memory Access Control Unit (MACU) stands for all IO transactions. Its main role is to gain access to the global memory for threads that need to exchange data (global memory can be generalized to any IO device).

The concept of a pipeline processor was borrowed from [9]. It consists of six stages (ThId, Fetch, Decode, Execute, Memory and Writeback). The processor has a multiplied register set to serve different hardware threads. In the presented platform they are formed in ten independent banks. The ThId stage is responsible for thread ID identification. The rest of the pipeline stages stand for the classical RISC processor architecture.

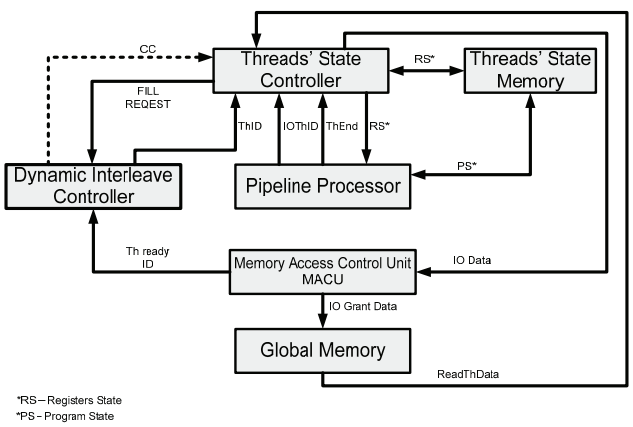


Fig. 1. General model of the platform
Rys. 1. Ogólny model platformy

At this point it is necessary to define three addition terms *the thread state*, *the pipeline state* and *pipeline context change*. ***Thread state*** is the current content of thread registers. It can be held in one of the processor register banks or in Thread State Memory (TSM). The first situation occurs when the thread is in the active mode (it performs program in the pipeline), the latter when it is in the offline mode (it does not perform operation in the pipeline). ***The pipeline state*** stands for current content of the processor register banks. It also defines system IDs of the active threads**. P*ipeline context change*** is a procedure that replaces the thread ID and processor register banks content with the data related to the new thread (threads) introduced to the pipeline.

A. Platform multitask management

As mentioned before, the TSC is responsible for organization of the threads working in the system. For proper maintenance each thread has a unique ID (Fig. 2). The system IDs are stored in the special stream called Main ID Stream (MIDS). The processor contains ten independent register banks so it is necessary, at every pipeline stage, to identify appropriate one for correct data operations.

It uses the IDs stored in the Processor ID Stream (PIDS) for this purpose. The system also maps IDs from MIDS to PIDS for accurate control. If a thread leaves the pipeline (switches to offline mode), its ID in MIDS is replaced with the last position stored in Shadow ID Stream (SIDS). SIDS items are prepared concurrently by the Dynamic Interleave Controller (DIC), to instantly supply a new ID when the context change operation occurs.

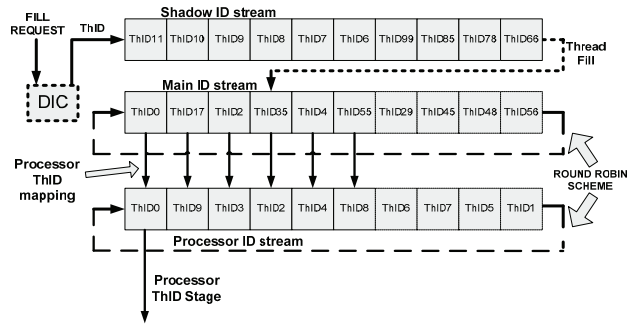


Fig. 2. Streams of threads ID
Rys. 2. Strumienie identyfikatorów wątków

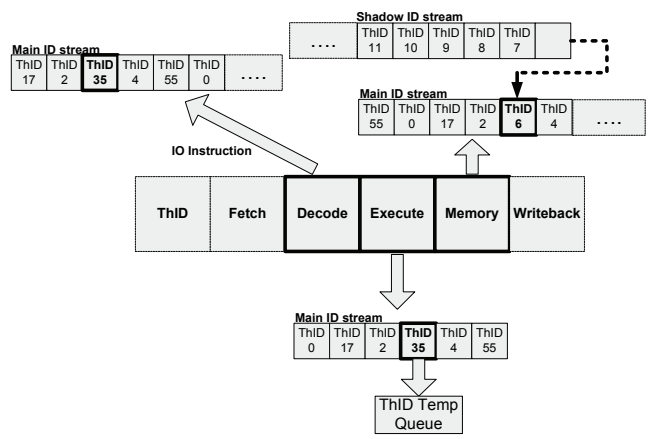


Fig. 3. System ID exchange procedure
Rys. 3. Procedura zmiany identyfikatora systemowego

Figures 3 and 4 show the procedure of pipeline context exchange. Similarly to the idea of storing thread ID, it is essential to prepare the register content for threads that will be introduced to the pipeline. Shadow State Queue (SSQ) is formed for this purpose. The needed data is delivered concurrently (it does not influence the processor operation) from TSM. For correct system management the TSC maps successive SIDS IDs with the elements stored in SSQ. This mechanism guarantees the proper data for the pipeline in the case of context change procedure.

B. Platform time predictable memory system

A platform predictable memory system is based on the thought to change pipeline context when a thread needs to perform an IO operation. Communication with the main memory (DRAM) can take hundreds of cycles. To address this problem, some researchers [6, 9, 10] introduced small local memories for faster data transfers. In this paper the idea of time predictable memory system is based on a simple philosophy: ***use thread IO communication time for operation of other ready to go thread.***

The mechanism of changing pipeline context is used when the thread needs to perform IO operation. The processor informs TSC that the current instruction is a data transfer instruction (if this is the case) at the decode stage (Fig. 3). This starts the pipeline context change procedure. In the following stages the processor does not complete the IO instructions. Instead it prepares necessary information for the data transfer. The platform assumes an ARM architecture (load/store scheme) so it is only necessary to indicate the source registers and operation type (read/write). For this purpose a special register (Fig. 5.) called IO register (IOR) is set.

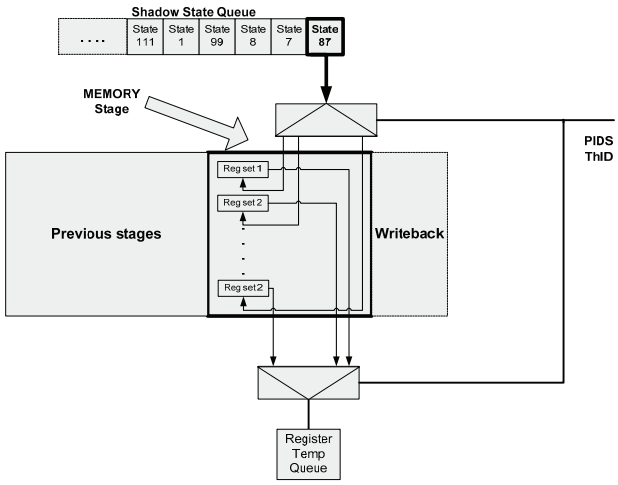


Fig. 4. Register content change
Fig. 4. Zmiana zawartości rejestrów roboczych

According to this information TSC is able to collect the rest of necessary data for IO transfer. During ID number change (Fig. 3) and register content change (Fig. 4) special queues called ThId Temp Queue (IDQ) and Register Temp Queue (RTQ) are formed. This information is needed to save thread status in TSM and collect necessary data for IO transfer. IDQ and RTQ are periodically flushed to provide space for further thread IO data.

When TSC collects all the needed memory transfer information, it prepares a dedicated IO data packet. Successive data packets produce an IO queue (IOQ). These elements are then copied by (IOQ is periodically flushed) to a special local data memory called Data Temporary Memory (DTM).

MACU decides, according to the implemented scheduling policy, threads (waiting in DTM) main memory access order. When the IO transfer is terminated, thread ID is moved to ready to go pool, so it could be rescheduled by DIC. If a read operation was performed, the obtained data needs to be saved in the TSM to update the thread status. It is sent to a special Main Memory Data Queue (MMDQ), and periodically flushed by TSC.

With small modifications the described scheme can be directly applied to a thread termination event. For the appropriate system action the processor ISA was expanded. If "EndTask" instruction is introduced to the pipeline, it means that it is the last thread's instruction.

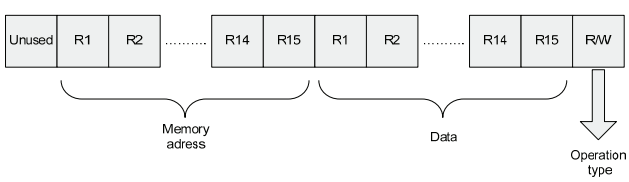


Fig. 5. IOR register
Rys. 5. Rejestr IOR

It is also possible to completely change the pipeline context. If that is the case during ten thread cycles (one thread cycle equals six processor clock cycles), ten new threads are launched into the pipeline. SSQ, IDQ, RTQ are ten position wide so it is possible to save the present pipeline content (queues are flushed/filled before the procedure start). This functionality is very important when DIC implement scheduling policy on fair pipeline access.

The presented memory system hides main memory access transfer details from the proposed architecture performance. As long as there are ready to go threads waiting in the pool, the system is fluently (without any pipeline stall) performing threads tasks.

5. Scheduling strategy

The scheduling strategies are employed in Dynamic Interleave Controller and Memory Access Control Unit. In both cases the main goal of delivered algorithms is to "produce" as many as possible “ready-to-go” threads. In the proposed platform, if this number is greater or even than ten, the main memory access and pipeline context exchange becomes processor transparent (ie. it does not reduce the system performance).

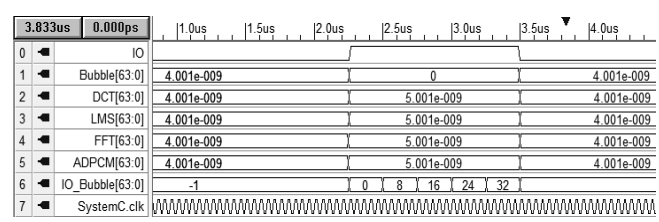
A. MACU scheduling

The algorithm implemented in the platform was inspired by [11, 12]. To obtain the "main goal", it is necessary to first schedule these threads that need to communicate with the main memory very rarely or have very small access transfer. However, this kind of scheduling policy can be used only if threads do not introduce any timing parameter. The platform is also able to use the DEADLINE parameter [1]. In this case gaining main memory access must be some kind of trade off between the threads memory intensiveness, the access time and DEADLINE. It is very important to balance these parameters. If threads with short DEADLINE are more privileged, it could entail small number threads that are ready to go, especially when these threads are memory intensive or have long time access. On the other hand, it would be possible to miss threads' DEADLINE, if memory intensive and access time parameters were more advantaged.

B. DIC scheduling

Dynamic Interleave Controller uses the same criteria to schedule threads to the pipeline. A special ready to go threads queue (RGQ) is prepared to fill the SSQ when needed. Each time the main memory access finish DIC rearranges RGQ according to current thread parameters. DIC however can work in two modes. The first one assumes that the pipeline context change occurs only if one (or more) thread needs to perform IO transfer. Using the second one it is possible to completely change the pipeline context (ie. remove all active threads and introduce new ones in the pipeline).

a)



b)

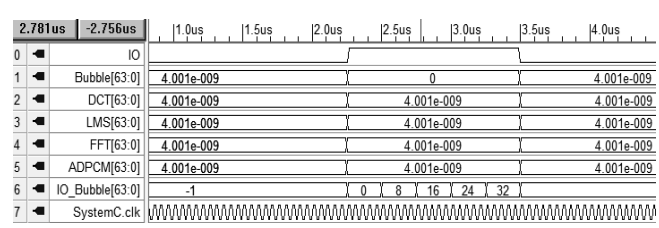


Fig. 6. Sample simulation waveforms:
a) IO time sharing
b) No IO time sharing
Rys. 6. Przykładowa symulacja:
a) z dzieleniem czasu podczas operacji wejścia/wyjścia
b) bez dzielenia czasu podczas operacji wejścia/wyjścia

6. Results

To perform some experiments, a platform designed in [13] was used. Figure 6 a) shows the situation in which if one of the threads starts an IO cycle, the rest can share its time window (they can spend more time in the pipeline).

In the case of Figure 6b) when the IO operation starts all of the active threads keep their current time window. It should be noted that this situation reflects the methodology used in [1, 5, 6, 9].

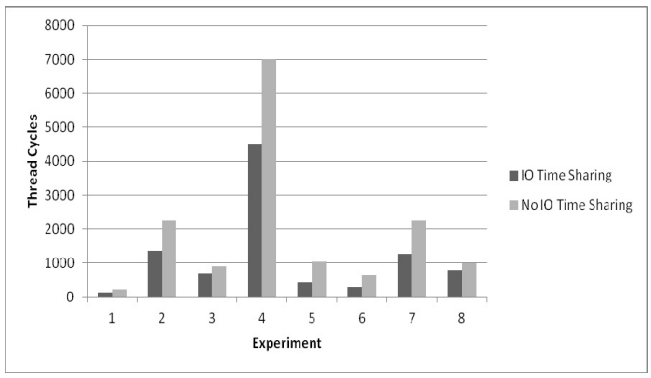


Fig. 7. IO time sharing vs. No IO time sharing
Rys. 7. Wyniki symulacji z dzieleniem i bez dzielenia czasu podczas operacji wejścia/wyjścia

The described methods are compared in Fig. 7. The experiments were performed for five different Mälardalen benchmarks [13] using random input data. The number of IO cycles was constant in both cases. It can be clearly seen that the threads using the proposed methodology needed fewer thread cycles (one thread cycle equals the assigned time window) to complete their work. The total time needed to accomplish the task is shorter because the time period reserved for thread performing IO cycle is dispensed between active threads. It should be also noted that IO transfer is performed concurrently (at the same time period).

7. Conclusions

The presented approach delivers the methodology that implements time predictable architecture for real-time systems. The main benefit is a **platform independent,** time predictable and deterministic memory system. This paper discusses only the usage of global memory as a source of threads data. It should be noted that other "mid-memories" like caches, SPM can be used even if they show unpredictable behavior. This can be explained by the fact that all of memory transfer details are hidden in implementation of Memory Access Control Unit. It should be stressed that in the proposed case the only condition to get complete PRET architecture is to have n-1 ready to go threads (n reflects the number of pipeline stages)

Use of the suggested approach also improves the total system performance. This is achieved by concurrently performed IO access and progress of ready to go threads waiting in the pool (it is also possible to share this time).

The first results shown in Section 6 look very promising. The future work will concentrate on developing scheduling algorithms, refining platform details and performing further experiments.

8. References

[1] Edwards S.A., Lee E.A.: The Case for the Precision Timed (PRET) Machine, Proceedings of Design & Automation Conference, DAC 2007, June 4-8, 2007, ACM, New York, NY, USA, p. 264–265.

[2] Andalam S., Roop P. and Girault A.: Predictable multi-threading of embedded applications using PRET-C. In Proc. MEMOCODE, pages 159–168, 2010.

[3] Schoeberl M.: A Java processor architecture for embedded real-timesystems. Journal of Systems Architecture, 54(1–2):265–286, 2008.

[4] Lee E.E. and Messerschmitt D.: Pipeline interleaved programmable DSP's: Architecture. Acoustics, Speech, and Signal, IEEE Transactions on,35(9):1320–1333, 1987.

[5] Liu, Reineke J., Broman D., Zimmer M. and Lee E. A.: A pret microarchitecture implementation with repeatable timing and competitive performance. In To appear in Proceedings of International Conference on Computer Design (ICCD), October 2012.

[6] Liu: Precision Timed Machines. PhD thesis, EECS Department,University of California, Berkeley, May 2012.

[7] Reineke J., Liu I., Patel H. D., Kim S. and Lee E. A.: PRET DRAM controller: Bank privatization for predictability and temporal isolation. In CODES+ISSS, pages 99–108. ACM, October 2011.

[8] Banakar R., Steinke S., Lee B. S., Balakrishnan M. and Marwedel P.: Scratchpad memory: design alternative for cache on-chip memory in embedded systems. In Proc. CODES, pages 73–78, 2002.

[9] Pułka A., Milik A.: Dynamic Rescheduling of Tasks in Time Predictable Embedded Systems, Proceedings of Programmable Devices and Embedded Systems, PDES 2012, Brno, Czech Republic, Vol. 11, Part 1, May 23-25, 2012, p. 305–310.

[10] David Broman, Michael Zimmer, Yooseong Kim, Hokeun Kim, Jian Cai, Aviral Shrivastava, Stephen A. Edwards, Edward A. Lee: Precision Timed Infrastructure: Design Challenges. In Proceedings of the Electronic System Level Synthesis Conference (ESLsyn), Austin, Texas, USA, May 31-June 1, 2013.

[11] Yoongu Kim , Michael Papamichael , Onur Mutlu , Mor Harchol-Balter: Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior, Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture, p.65-76, December 04-08, 2010 [doi 10.1109/MICRO.2010.51].

[12] Kultursay E., Kandemir M.: Das Addressing End-to-End Memory Access Latency in NoC-Based Multicores Sharifi, C.R. Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on Digital Object Identifier: 10.1109/MICRO.2012.35 Publication Year: 2012 , Page(s): 294 – 304.

[13] Golly Ł., Pułka A.: Time predictable systems based on pipeline processor with interleaving of hardware thread, ELEKTRONIKA - KONSTRUKCJE, TECHNOLOGIE, ZASTOSOWANIA 2013/12.

otrzymano / received: 24.03.2014
przyjęto do druku / accepted: 02.05.2014 *artykuł recenzowany / revised paper*