System Time Issues for the ARM Cortex A8 Processor

Irina Fedotova, Eduard Siemens
Anhalt University of Applied Sciences - Faculty of Electrical, Mechanical and Industrial Engineering
Bernburger Str. 57, 06366 Koethen, Germany
E-mail: {i.fedotova, e.siemens}@emw.hs-anhalt.de

***Abstract*—The use of system-on-chip (SoC) platforms has emerged as an important integrated circuit design trend for communication and industrial control applications. At the same time, requirements of stable, efficient and precise processing time values are growing rapidly. Since the ARM processor doesn't possess known timers of PC-platforms such as TSC counter or HPET timer, the common way to obtain time values on ARM-architecture is still only through Linux system calls which are mapped to ARM specific time counters. Indeed, direct access to hardware can help to reduce costs down to 200 nanoseconds against 2-1 microseconds of the time acquisition via Unix system call interface. However, designing this approach is a challenging task. This paper describes specific issues and features of tracking system time on the ARM Cortex A8 processor under Linux OS.**

***Keywords:* timestamp precision; time-keeping; embedded Linux; ARM architecture.**

I. Introduction

Over the last few years, the ARM architecture has become the most pervasive 32-bit architecture in the world, with wide range of integrated circuits available from various manufacturers. ARM processors are embedded in products ranging from cell/mobile phones to control systems of wind-turbines. Depending on the specific products [1][2][3], requirements of meeting timing constrains vary from reaction in a predefined time frame (as in the hard-real time system) and not critical timely responsiveness (as in the soft-real time system). However, any real-time requirements on ARM embedded projects can’t be met without reliable development platform.

The Linux kernel offers interrupt latency less than 100 nanoseconds in most cases on today's fast processors. Although, the non-deterministic nature of task scheduling may make it unsuitable for some hard real-time systems, the wealth of utility programs included with Linux makes it ideal for such tasks as report generation as well as networking and addressing interoperability issues [4][5]. Though the Linux OS, running on diverse ARM architectures becomes increasingly popular in embedded systems, often exactly the lack of high-precision time control pushes system designers to decisions against Linux-based, towards more costly FPGA- or ASIC-based solutions

Modern PC-platforms come with different hardware timers having different attributes. The most popular in the PC domain is a 64-bit TSC counter which represents relative time values and counts CPU cycles from the power on or reset of the computer. The HPET device provides multiple timers, each consisting of a timeout register that is compared with its central counter. Meanwhile, the timing mechanism of ARM-architecture appears more complex. The time performance on ARM is limited by a 32-bit register with a relatively low frequency of 41 ns and hence, the overflow occurs in less than every 3 minutes. Additionally, ARM specific restrictions of memory management allow accessing to registers only from kernel space.

The remainder of the paper is organized as follows. In Section II, related work is described. Section III shows the specific details of each initial time source within the ARM Cortex A8 processor. Some experimental results of appropriate timer sources along with their performance characteristics are shown in Section IV. Finally, Section V describes next steps and future work in our effort to develop a tool for efficient high-performance measurements.

II. Related Work

Since the essence and importance of time acquisition in embedded systems has become apparent, several research projects have suggested to design timing mechanism for real-time application [6][7]. However, these researches consider handling of timer on the outdated platforms, such as, for example, AVR microcontrollers. In other proposals, the entire time capturing process is integrated into dedicated hardware devices [8]. For PC-platforms based on x86 and x86-64 processors, the idea of the invention single structure providing access to different hardware timers has already been suggested [9][10]. Nevertheless, our investigations directed to specific timers of ARM-architecture with the potential purpose of designing the new accurate real-time system

III. ARM Cortex A8 Specific Clocks and Timers

All current investigations are being performed on the *BeagleBone* credit-card-sized Linux computer with single-core AM335x Cortex A8 ARM processor under the Debian GNU/Linux 7.0 with the stable kernel version 3.8.10.

On the given platform, maximum performance and operation timer for user satisfaction is ensured by the special power, reset, and clock management (PRCM) module. This module provides a centralized control for the generation, distribution and gating of most clocks in the device. PRCM gathers external clocks and internally generated clocks for distribution to the model in the device. Moreover, PRCM manages the system clock generation.

Following to the processor's technical manual [11, Chapter 8], the device has two reference clocks which are generated by on-chip oscillators or externally. These are for the main clock tree and RTC block, respectively. In the case of an external oscillator, the 32-KHz crystal oscillator is controlled and is configurable by RTC. This device also contains an on-chip oscillator. This oscillator is not configurable and is always on. The main oscillator on the device produces the master high frequency clock.

Therefore, we assume that there are oscillators with at least two different frequencies. The first one is widespread 32,768 kHz frequency, which is exactly 2^{15} cycles per second, a convenient rate to use with simple binary counter circuits. In this case, the time resolution is 31,25 µsec The second value, so called master frequency, is equaled to 25 MHz with 40 nsec resolution, which allows performing comparatively fast measurements. Table 1 gives more detailed description of available clocks.

TABLE I
CLOCKS RESOLUTION AND MAXIMUM RANGE

Clock	Prescaler	Resolution	Interrupt Period Range	Wraps every
32.768 KHz	1 (min)	31.25 us	31.25 us to ~36h 35m	37 h
	256 (max)	8 ms	8 ms to ~391d 22h 48m	
25 MHz	1 (min)	40 ns	40 ns to ~171.8s	2, 983 m
	256 (max)	10.24 us	~20.5 us to ~24h 32m	

A prescaler is an electronic counting circuit, which allows the timer to be clocked at the rate user desires. The prescaler takes the basic timer clock frequency and divides it by some value before feeding it to the timer. For the given clocks, prescaler's divisor varies from 1 to 256 and accordingly provides 8 different values (divisible by two) of clock resolution. So, with every increase of clock range, the resolution is decreased.

According to the processor specification [11, Chapter 20], four possible timers exist on this processor:

− Dual Mode Timer (DMTimer);
− Dual Mode Timer 1 ms (DMTimer 1 ms);
− Real Time Clock Subsystem (RTS_SS);
− WATCHDOG.

The peripheral DMTimer is 32-bit timer and the module contains a free running upward counter with auto reload capability on overflow. The timer counter can be read and written in real-time (while counting) and configured for 32-bit or 16-bit operation. DMTimer can be configured in three modes of operation: timer mode, capture and compare mode. The compare logic allows an interrupt event on a programmable counter matching value. The capture mode allows capturing of the timer value in a separate register based on an input signal. By default, after core reset, the capture and compare modes are disabled.

In fact, DMTimer provides eight multiple timers:

− The DMTimer0 can only be clocked from the internal RC oscillator of 32.768 KHz.
− The DMTimer1 is implemented using the DMTimer_1ms module, which is capable of generating an accurate 1 ms tick using a 32.768 KHz clock. During low power modes, the master oscillator is disabled. Hence, in this scenario for generating the OS 1ms tick and timer based wakeup, it is sourced from the 32K RC oscillator.
− Each functional clock of DMTimer[2-7] is selected using the associated register from 3 possible sources: the 24-MHz system clock, the 32.768 KHz clock (see Table 1) or the external timer input clock. The availability of particular DMTimer[2-7] depends on the platform implementation. The Linux kernel already contains *dmtimer* driver, which allows using system time through accessing to the DMTimer2. Therefore, this timer is considered as preferable.

Additionally, as any other electronic device, the given processor possesses RTC clock, which keeps time of day, and have an alternate source of power, so that it works even with system power off. The RTC supports external 32.768-kHz crystal or external clock source of the same frequency.

The processor also contains a WATCHDOG timer based on an upward 32-bit counter coupled with a prescaler. It causes the system to be reset if it is not poked periodically. After reset generation, the counter is automatically reloaded with the value stored in the watchdog load register, the prescaler is reset and the timer counter begins incrementing again.

IV. TIME FETCHING PERFORMANCE RESULTS

On ARM platforms, all I/O access is memory-mapped. That means that timers' registers have their specific address stored in a known private memory region and access to them potentially possible only from kernel space. The *dmtimer* driver available in Linux also does not provide high-level abstractions. It is merely a small library supplying some functions for the clients of DMTimers to reserve and program the timers. So, the clients have to manage all the low-level programming and interrupt handling themselves. Accordingly, this driver is considered not used by many.

So, the suggested way of avoiding this obstacle for user would be writing a character device driver and implement *mmap()* function and the read-only access to the timer. It will allow mapping of the physical address of the DMTimer2 main counter to a virtual address in the program memory space.

The idea of timers map is as follows: firstly, the user process invokes an appropriate *mmap()* function and the kernel calls the device driver passing all necessary parameters. The driver validates the request and executes a function to map the necessary range of physical addresses into the address space of the user space. The driver returns an exit code to the kernel and the kernel re-dispatches the user process. Therefore, the user process accesses data of timing hardware by accessing the virtual address returned to it from *mmap()* call. Herewith, inaccurate offset in timer register from user space is handled in the open device operation in kernel space and in the case of error, returns user appropriate message.

Table 2 shows the performance results of getting the time values using direct access to hardware versus using a system call. Initially hardware timer, DMTimer2 with 25 MHz frequency was chosen. For the system call, *clock_gettime()* was issued. For this investigation, time was fetched in a loop of 10 million consecutive runs. The values are not filtered.

TABLE II
THE COSTS OF OBTAINING TIME VALUES ON ARM PROCESSOR

Time source	**Mean, μsec**	**Standard deviation, μsec**
System Call	1.025	2.201
Direct Access to DMTimer	0.201	0.800

Additionally, during these tests, CPU system time was estimated relatively real execution time of the program. In fact, CPU costs value of system call is more than 10 times greater than direct access to hardware.

TABLE III
CPU COSTS ON ARM CORTEX A8 PROCESSOR

Time source	**Real time of test execution**	**System time**	**Percentage ratio**
System Call	2m 59.203s	0m 20.846s	11.632%
Direct Access to DMTimer	2m 15.723s	0m 1.303s	0.960%

The chart in Fig. 1 demonstrates more detailed results of this experiment. The range of samples was decreased to track the behavior of timers more carefully. It shows the clear similarity of both timers' behaviors and proves that the given system interacts with this particular DMTimer2 hardware.

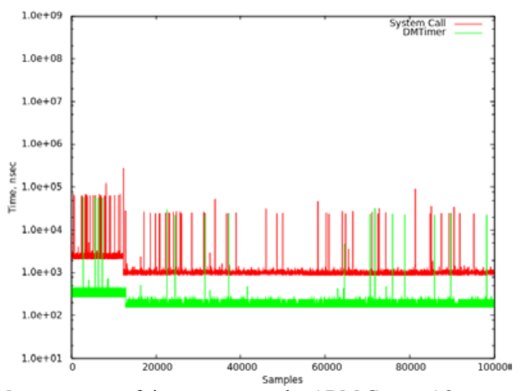

Fig. 1. Measurements of timers costs on the ARM Cortex A8 processor

V. CONCLUSION AND FUTURE WORK

With the rapid development of the field of industrial process control and the wide range of embedded systems, it is necessary to make a higher demand of the time accuracy and reliability of the control system. The embedded ARM Cortex A8 platforms can adapt to strict requirements of the time acquisition and potentially become a basis for building new real-time systems.

Nevertheless, primarily the number of significant issues must be satisfied. Firstly, the reliable interface allowing interacting with system timer must be provided, wherein the benefits of hardware access (such as low cost of obtaining values and low CPU costs) are saved. Secondly, a protection from timer 2.9 min overflow should guarantee meeting real-time constraints. Therefore, at this stage accomplishment of all above challenges is in progress. Moreover, to the next steps, the better support of the ARM Cortex A9 processor will be addressed. The potential benefits of multi-core processors for real-time embedded systems are enormous, but however have even more dangerous drawbacks.

REFERENCES

[1] M. Manivannan and N. Kumaresan, "Embedded web server & GPRS based advanced industrial automation using Linux RTOS," International Journal of Engineering Science and Technology, vol. 2 (11), no. 8, pp. 6074-6081, 2010.

[2] D. Wiklund and D.Liu, "SoCBUS: Cwitched network on chip for hard real time embedded systems," Proc. of the 17st Intl. Symposium on Parallel and Distributed Processing, Los Alamitos, pp. 78.1, IEEE Computer Society Press, 2003.

[3] I. Fedotova and E. Siemens, "Usage of high-precision timers in the wind turbines control systems," Supercomputers Jornal, Moscow, Publishing House SCR-Media LTD, Number 16, winter 2013.

[4] R. Lehrbaum, "Using Linux in Embedded and Real Time Systems," Linux Journal, vol. 2000 (75), Specialized Systems Consultants, Inc., 2000.

[5] B. Japenga, "Why Use Linux for Real-Time Embedded Systems" White paper. [Online]. Available: http://www.microtoolsinc.com/Articles/Why%20Use%20Embedded%20Linux%20for%20Real%20Time%20Embedded%20Systems%20Rev%20A.pdf

[6] A. A. Fröhlich, G. Gracioli, and J. F. Santos, "Periodic timers revisited: The real-time embedded system perspective," Computers \& Electrical Engineering, vol. 37, no. 3, 365-375, May 2011.

[7] K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating system," Proc. of the 8st ACM Symposium on Operating Systems Principles, New York, USA, pp. 89-102, 2001, doi: 10.1145/502034.502044

[8] A. Pásztor and D. Veitch, "PC based precision timing without GPS," The 2002 ACM SIGMETRICS international conference on Measurement and modeling of systems, Marina Del Rey California, USA, vol. 30, no. 1, pp. 1-10, June, 2002, doi: 10.1145/511334.511336.

[9] A. Aust, J. Brocke, F. Glaeser, R. Koehler, S. Kubsch, and E. Siemens, "Method for processing time values in a computer or programmable machine," US Patent 8, 185, 770, 2012.

[10] I. Fedotova, E. Siemens, H. Hu. "A high-precision Time Handling Library," Proc. of the 9th International Conference on Networking and Services, (ICNS 2013), Lisbon, pp. 193-199, March 2013.

[11] AM335x ARM Cortex-A8 Microprocessors, Technical Refernce Manual p. 5.2.5. [Online]. Available: https://s3-us-west-1.amazonaws.com/123d-circuits-datasheets/uploads%2F1378501288286-gibpl1belakmx6r-2561e976ef65a4ecf67b3a3ba2590088_%2FAM335x_ARM_Cortex-A8%28spruh73h%29.pdf