Vol.2Issue IV, April 2014
ISSN: 2321-9653

INTERNATIONAL JOURNAL FOR RESEARCHIN APPLIED SCIENCE AND ENGINEERING TECHNOLOGY (IJRASET)

A Novel Approach ForHost-Compiled Processor Model With Cache And PipeliningIntegration

N.Vinothini[1] T.Uma Maheswari[2]K.Kannan[3] M.Sankar [4]

[1]*PG Scholar*
M.E – Embedded Systems
Department of Electrical and Electronics Engineering,
R.V.S College of Engineering and Technology, Dindigul.

[2, 3, 4]*Assistant Professor*
Department of Electrical and Electronics Engineering,
R.V.S College of Engineering and Technology,
Dindigul.

Abstract – In the increasing model of the embedded system software solutions host compiled as well as the Real Time Operating System made the dramatic added applications for the mobile applications all over the world. However, designerspay the price for higher performance with a loss in timing accuracy.In this work, we introduce a novel predictive OS model toprovide fast software simulation with accurate scheduling of periodicreal-time tasks. The OS model predicts the next preemptionpoint by monitoring system state, and automatically and optimallyadjusting the granularity of back-annotated delays. We evaluatedour simulator on a range of periodic task sets. Our observationsshow that we can achieve the same 99% accuracy as a simulationat 1 s granularity with an average 230x speedup.

Keywords: Real time operating system (RTOS), Host-compiled processor, cache model and pipelining.

I. INTRODUCTION

In recent years, the complexity of embedded systems has increased dramatically. The challenge is to design such complex systems with constrains on design goals, especially real-time performance, at reduced development time and cost. Since software provides a high degree of flexibility and easy code reuse, the trend over the last years has been to shift more and more functionality into software. Hence, effective evaluation of such complex, software-intensive systems in early stages of the design process is essential. Many studies have focused on methods to provide fast and accurate simulation by abstracting the software execution environment.

For example, virtual platforms provide a high level functional prototype of a target architecture, which allows designers to debug and simulate their software along with the rest of the system before the actual hardware is available. Virtual platforms execute the binary code of the software on the target architecture prototype at close to real-time speeds. However, such approaches only provide fast functional simulation, with limited or no timing information. Recently, host-compiled approaches have been developed to provide fast simulation coupled with accurate timing execution. In such approaches, the software is natively compiled and executed on a host machine while an abstract model of the target architecture manages the execution order of user application tasks. For timing accuracy, the application code is instrumented with back-annotated execution delays. In host compiled approaches, higher speed is achieved by coarse grained simulation of the system, which inherently comes at a loss in timing accuracy. In other words, there is a fundamental tradeoff between simulation speed and timing accuracy. In this paper, we aim to eliminate this tradeoff in host compiled software simulation. We present a

platform modeling approach for fully accurate yet fast simulation of real-time applications. At its core, this is enabled by a novel RTOS model, which is capable of permanently monitoring system state to automatically adjust simulated timing granularities and eliminate task schedulingerrors while maintaining fast simulation speed. In such an approach, designers need not be concerned with manually selecting a proper granularity for optimizing the speed and accuracy tradeoff. Instead, the plat form simulator automatically, continuously and dynamically adjusts to changing system conditions in order to achieve an optimal simulation. The remainder of this paper is organized as follows: in the following subsections, we review related work and present an overview of our simulator. Then, we discuss the details of our approach in Section II. Results of our experiments are summarized in Section III. Finally, we conclude this paper with a summary and outlook on future work in Section IV.

A. RELATED WORK

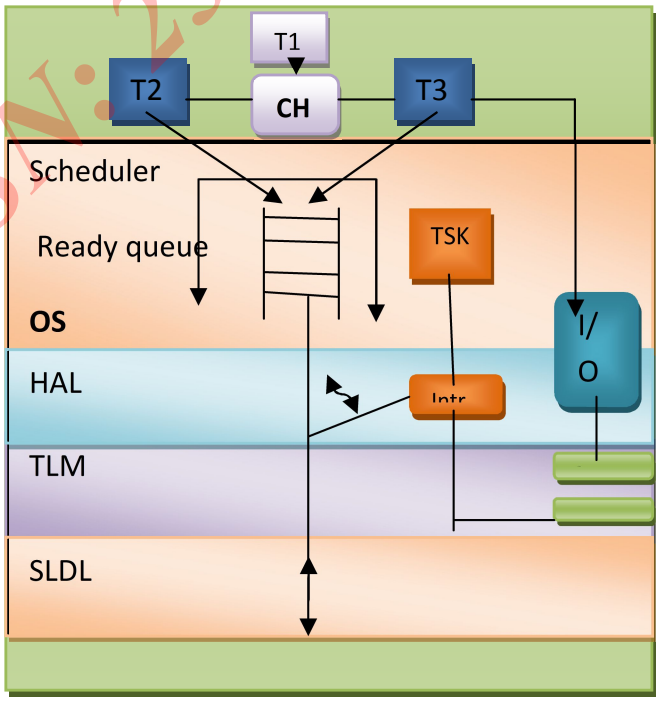
Recently, so-called host-compiled or source-level simulation approaches have received widespread attention as a solution for rapid evaluation of software at early design stages. Such approaches provide high performance by abstracting the simulation platform [1], [2], [3], [4]. The high-level source code of applications is back-annotated with timing estimates, which are typically obtained by compiling to an intermediate representation [5], [6]. Application execution is managed by an abstract model of the software execution environment, which is usually developed on top of standard system-level design languages (SLDLs) (e.g. System C [7] or Spec C [8]).Some of the earliest host-compiled approaches were centered on models of the OS itself [9], [10], [11].

Later, these approaches were extended into complete processor models that include timing-accurate descriptions of interrupt chains and TLM-based bus interfaces [12], [13]. Such processor model shave been shown to simulate at speeds beyond 500 MIPS with more than 95% timing accuracy. Several researchers have focused on improving the accuracy of high-level simulators while maintaining similar performance. Krause et al. [14] present combined ISS and abstract RTOS model co-simulation.

This approach replaces an actual RTOS binary code with an abstract model running outside the ISS and performs cycle-accurate thread switches. Khaligh et al. [15] present an adaptive TLM simulation kernel, which changes the level of accuracy during simulation to the level expected by designers. Schirner et al. [16] introduce a granularity-independent approach for accurate simulation of interrupts on host-compiled processor models by applying optimistic prediction and correction. In all cases, however, fundamental static speed and accuracy tradeoffs remain. By contrast, we adjust granularities automatically, optimally and dynamically to achieve fast and accurate simulation.

B. Host-Compiled Software Simulator

We have developed a high-level, host-compiled software simulator, details of which can be found in [17]. Figure 1 show the structure of our simulator, which is designed in a layered-based fashion.



A standard SLDL kernel provides abasic platform for running simulations on a host machine. In combination with the underlying SLDL, a TLM layer interfaces the software simulator with standard TLM back planes that provide a fast system-wide co-simulation platform. A hardware abstraction layer (HAL) includes necessary I/O drivers and implements an

abstract interrupt handling mechanism. When an interrupt is captured by the TLM layer, the HAL suspends application execution and lets the interrupt handler trigger the registered.

Interrupt task in the OS. On top of the HAL, an OS layer replicates a typical OS architecture to manage the execution order of a multi-tasking application. The OS model there by schedules, queues, dispatches and executes application and interrupt tasks according to a chosen scheduling policy. At the highest level, the application layer consists of concurrent and sequential high-level SLDL processes, which communicate with each other using abstract SLDL channels.Kernel in order to control the state of the system.

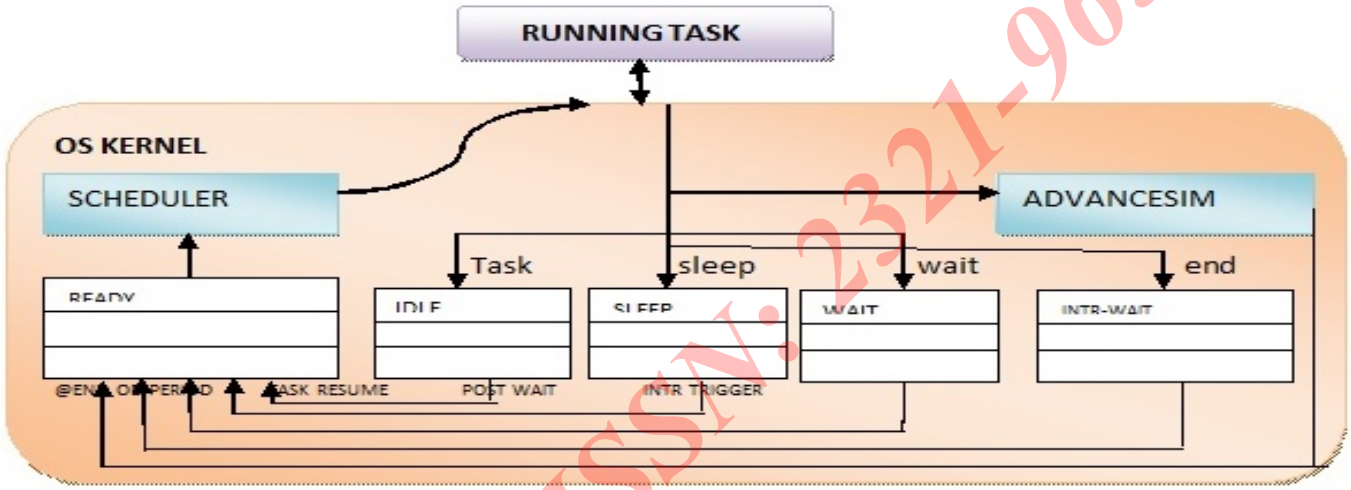


Fig.2.RTOS Model

The user application isintegrated into the simulator and accesses services of the OS model via a canonical OS API. At the core of the simulation engine is the OS model, which dynamically schedules concurrent application tasks to emulate their sequential execution in software. The structure of our OS kernel is shown in Figure 2. The key component of the kernel is a task scheduler, which is invoked by the OS API methods whenever a context switch is possible or required. It decides on the next task to execute and preempts the currently running task

If needed in the OS model, each task can be in five states, and tasks move to different states by calling API methods of the OS model maintains tasks in five internal queues: a Ready queue holds tasks that are ready to execute and is sorted based on a user-defined scheduling policy. An Idle queue holds periodic tasks that have called the kernel's Task End Cycle method at the end of their iteration. The Idle queue is ordered based on the release time of each task's next iteration. Idle tasks are retrieved from the head of queue and placed in the Ready queue by the OS kernel at the start time of their next period. Tasks waiting for an event are suspended and transferred to a Wait queue upon calling a Pre Wait method. Respectively, a blocked task will be placed back in the Ready queue when a Post Wait method is called to release it. To distinguish tasks that are waiting for an external event, an lntr Wait queue holds interrupt tasks until the interrupt handler in the HAL calls the lntr Trigger()method to move them to the Ready queue. Finally, a Sleep queue holds tasks that have been suspended until they are resumed again.

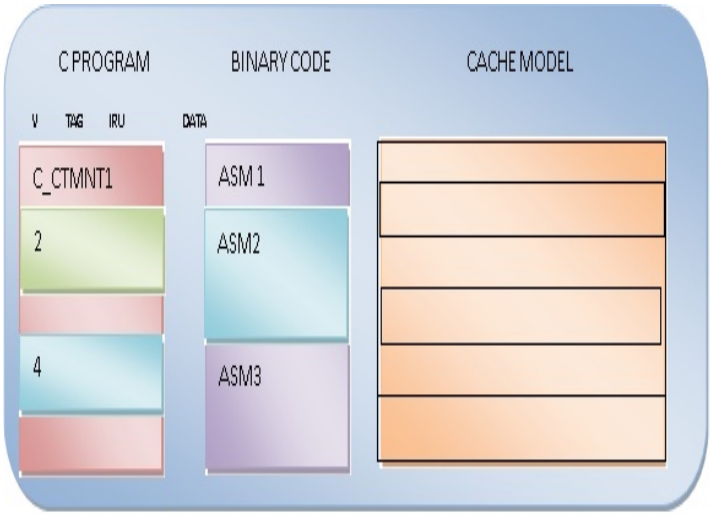


Fig.3. Cache memory

In addition to basic OS services, the OS kernel simulates task execution delays using underlying SLDL primitives whenever the running task calls a Time Wait method. Basic execution delays of the task code are back-annotated from estimations or measurements once atcompile time. In traditional models, the granularity of delays is defined by the application code. The scheduler is only called after advancing the simulation time to allow for preemption of the current task by any higher priority task that became available in the meantime. As such, errors in the preemption model are a direct function of the back-annotated application-level timing model. Large granularities result in fast simulation, but may lead to preemption points being shifted by a large delay. On the other hand, accurate simulations require a fine granularity at slow simulation speeds. By contrast, we propose an approach that automatically adjusts timing granularities to the level needed. The OS model has complete knowledge of the system state at any given time. As such, we can develop a kernel that utilizes this knowledge to automatically control simulation timing such that an error-free scheduling mechanism is provided at the fastest possible speed.

PROPOSED WORK

THE CACHE MODEL:

The cache model, as it can be seen on the right side of Figure 3, contains data space that is used for the administration of the cache. In this space, the valid bit, the cache tag and the least recently used (lru) information (containing the replacementstrategy) for each cache set during the run-time is saved. The number of cache tags and the according amount of valid bits that are needed depends on the associatively of the cache (e.g. fora two-way set associative cache, two sets of tags and valid bits areneeded).

Cache analysis blocks

In the middle of Figure 3, the C source code which is corresponding to a basic block is divided in several smaller blocks, the so-called cache analysis blocks. These blocks are needed for the considerationof the effects of instruction caches. Each one of these blocks contains that part of a basic block that fits into a single cache line. As every machine language instruction in such a cache analysis block has the same tag and the same cache index, the addresses of the instructions can be used to determine how a basic block has to be divided into cache analysis blocks.

This is because each address consists of tag information and cache index. The cache index information is used to determine at which cache position the instruction with this address is cached. The tag information is used to determine which address was cached, as there can be multiple addresses with the same cache index. Therefore, a changed cache tag can be easily determined during thetraversal of the binary code with respect to the cache parameters. The block offset information is not needed for the cache simulation, as no real caching of data takes place. After the tag has been changed or at the end of a basic block, a function call that handles the simulated cache and the calculation of the additional cycles of cache misses is added to this block. More details about this function are described in the next section.

Cycle calculation code as previously mentioned, each cache analysis block is characterized by a combination of tag and cache set index information. At the end of each basic block, a call to a function is included. During run-time, this function should determine whether the different cache analysis blocks

which the basic block consists of are in the simulated cache or not. This way, cache misses are detected. The function is shown in Figure 4. It has the tag and the range of cache set indices (iStart to iEnd) as parameters.

```
-------------------------------------------------------
intcycleCalculationICache( tag, iStart, iEnd )
{
for index = iStart to iEnd
if tag is found in index and valid bit is set then
{ // cache hit
renewlru information
return 0
}
else
{ // cache miss
uselru information to determine tag to overwrite
write new tag
set valid bit of written tag
renewlru information
return additional cycles needed for cache miss
}
end for
}
-------------------------------------------------------
```

Fig.4.Function for cache cycle correction

To find out if there is a cache hit or a cache miss, the functionchecks whether the tag of each cache analysis block can be foundin the specified set and whether the valid bit for the found tag is set.If the tag can be found and the valid bit is set, the block is alreadycached (cache hit) and no additional cycles are needed. Only thelru information has to be renewed.In all other cases, the lru information has to be used to determinewhich tag has to be overwritten. After that, the new tag has to bewritten instead of the found old one, and the valid bit for this taghas to be set. The lru information has to be renewed as well. In alast step, the additional cycles are returned and added to the cyclecorrection counter.

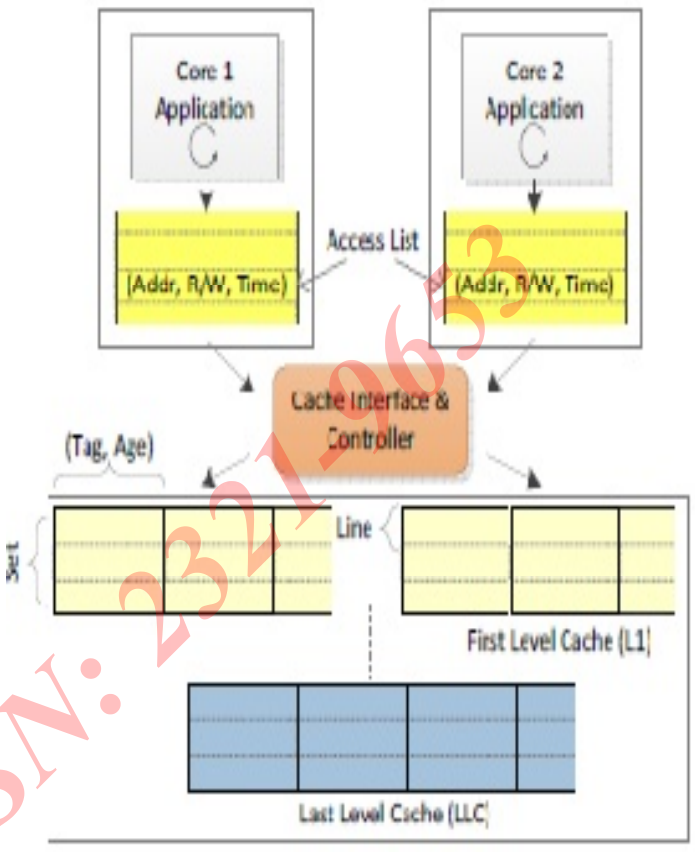


Fig.5.High-level cache hierarchy model

For accurate performance evaluation, we need to considerperformance penalties due to cache misses and update staticback-annotated delays during simulation. For this purpose, wedeveloped a high-level model of a cache channel that emulatesthe system memory behavior by updating its internal stateson every memory access. Note that we only need to modelhit/miss behavior of the cache, i.e. we are not concerned aboutthe data that is stored in the cache. Instead, the simulationhost takes care of maintaining coherent data values. In ourcache model, each cache line is composed out of an addresstag, an age counter to implement a replacement policy, and a coherency flag to store the current state of each linecompared to other cores' caches (Figure 5). Associated with each core, anaccess list stores locally ordered memory references reportedby the application running on that core. Each location in thislist contains a memory address, access mode i.e.

Read orWrite), and an access time. Using this information, a cachecontroller is able to manage the cache state updating processand to report back total miss cycles. Accordingly, the simulator

Pipelining Model:
At the heart of the ARM7 CPU is the instruction pipeline. The pipeline is used to process instructions taken from the program store. On the ARM 7 a three-stage pipeline is used.A three-stage pipeline is the simplest form of pipeline and does not suffer from the kind of hazards such as read-before-write seen in pipelines with more stages. The pipeline has hardware independent stages that execute one instruction while decoding a second and fetching a third. Fig 6 shows that the 3 stage pipelining. The pipeline speeds up the throughput of CPU instructions so effectively that most ARM instructions can be executed in a single cycle. The pipeline works most efficiently on linear code.

Example:
The instruction:

0x4000 LDR PC, [PC,#4]

will load the contents of the address PC+4 into the PC. As the PC is running eight bytes ahead then the contents of address 0x400C will be loaded into the PC and not 0x4004 as you might expect on first inspection.

Implementation of Proposed Work

Micro C OS-II kernel (RTOS) IAR Embedded Workbench is the world-leading C/C++ compiler and debugger tool suite for applications based on 8-, 16-, and 32-bit microcontrollers. It supports more devices in more processor architectures than any other tool on the market. Outstanding speed optimizations enable IAR Embedded Workbench to generate faster code than ever before. Read more about the advantages of IAR Embedded Workbench.

IAR Embedded Workbench Components

IAR Embedded Workbench incorporates a compiler, an assembler, a linker and a debugger into one integrated development environment (IDE). This gives you an uninterrupted workflow and one single toolbox in which all components integrate seamlessly. IAR Embedded Workbench is advanced and powerful, yet easy to use thanks to its smart functionality and user-friendly interface. Read more about the advantages of the world-leading tool suite.

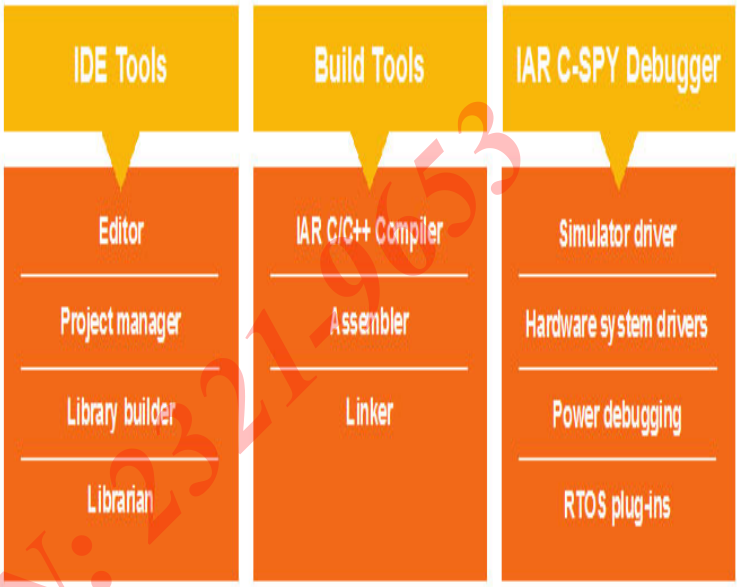


Fig.7. IAR Embedded Workbench Components

MicroC/OS-II:

MicroC/OS-II (commonly termed as µC/OS-II or uC/OS-II), is the acronym for Micro-Controller Operating Systems Version 2. It is a priority-based pre-emptive real-time multitasking operating system kernel for microprocessors, written mainly in theC programming language. It is intended for use in embedded systems. Its features are:

- It is a very small real-time kernel.
- Memory footprint is about 20KB for a fullyfunctional kernel.
- Source code is written mostly in ANSI C.
- Highly portable, ROMable, very scalable,preemptive real-time, deterministic, multitaskingkernel.
- It can manage up to 64 tasks (56 user tasksavailable).
- It has connectivity with µC/GUI and µC/FS(GUI and File Systems for µC/OS II).
- It is ported to more than 100 microprocessorsand microcontrollers.

- It is simple to use and simple to implement butveryeffective compared to theprice/performance ratio.

Task Set	S1	S2	S3
No of Tasks	2	3	4
CPU utilization	19%	25%	31%
Avg Err (1us)	0.12%	0.19%	0.26%
Avg Err (10us)	0.14%	0.20%	0.28%
Avg Err (100us)	0.16%	0.21%	0.29%
Avg Err (1000us)	1.58%	2.33%	3.47%
Avg Err (P-RTOS)	0.11%	0.17%	0.22%

Table.1.Performance Comparison Table.

MicroC/OS-II is the second generation of a kernel originally published (with source code) in a two-part 1992 article inEmbedded Systems Programming magazine and the book µC/OS The Real-Time Kernel by Jean J. Labrosse (ISBN 0-87930-444-8). The author intended at first to simply describe the internals of a portable operating system he had developed for his own use, but later developed the OS as a commercial product.µC/OS-II is currently maintained by Micrium Inc. and can be licensed per product or per product line. Use of the operating system is free for educational non-commercial use. Additionally, Micrium provides other middleware software products such as µC/CAN, µC/FL, µC/FS, µC/GUI, µC/Modbus, µC/TCP-IP, µC/USB and a large assortment of µC/TCP-IP applications such as client software for DHCP, POP3, SNTP, FTP, TFTP, DNS, SMTP, and TTCP. Server software includes HTTP, FTP, and TFTP. PPP is also available.

EXPERIMENTAL RESULTS:

We simulated a set of randomly generated artificial periodic tasks and compared the simulation performance of ours model to a conventional model under different timinggranularities. Accuracy is analyzed by comparing results to the execution of tasks on a reference ISS [15] modeling a single core MIPS Malta platform running a Linux 2.6 kernel configured with preemption and high resolution timers.

The experimental setup consists of randomly generated periodic tasks with uniformly distributed periods over and task weights over for small (S), for large (L), and for medium (M) tasks. The priority of tasks is assigned inversely to their periods following a rate monotonic scheduling scheme. The execution delay of tasks is modeled by a delay loop of no-operation (NOP) instructions. We ran each task set for 10 s of simulated time. At a nominal rate of 100 MIPS simulated by the reference ISS, this corresponds to 1000 million NOP instructions. Task sets have been generated to cover various task weight range sunder different total system utilizations. We analyzed the accuracy of our approach by comparing the response times of periodic tasks in the reference ISS with our host-compiled simulator. Delays were back-annotated into host-

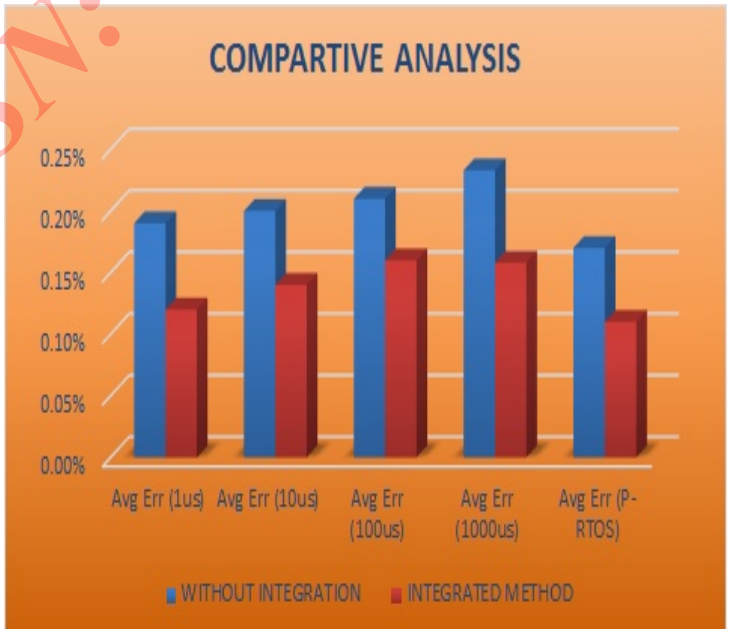


Fig.8.Error Rate Performance Comparison chart

Compiled models directly from measurements taken when running on the ISS. Model error was measured as the average absolute difference in individual task response times overall

tasks and task iterations.Table 1 shows that the comparative analysis for error rate. Table summarizes the task set properties and compares the accuracy and performance of our predictive RTOS (P-RTOS) model with that of a conventional model at four different back-annotation granularities.

We can observe that the highest possible accuracy is achieved using the P-RTOS model. This is equivalent to a conventional model at 1 s granularity, which loses a large amount of accuracy at coarser granularities. Table 1 shows that the comparative analysis for error rate. Note that although we would expect to see zero errors on the predictive model, our previous experience has shown [14] that remaining errors are caused by OS context switch overheads and non-ideal behavior of a real Linux system not included in our RTOS model. In terms of simulation performance, an average simulation speed of 67 GIPS is achieved on the P-RTOS model. This is 233 times faster than the original OS model at a granularity of 1 s and similar to the original model at 1 ms granularity.In the conventional OS model, designers are responsible for choosing the timing granularity to achieve acceptable accuracy and performance. However, selecting the proper granularity is not straightforward.For example, using the granularities of 1 s and 10 s, the same accuracy is provided while the former simulates 10 times faster than the latter. In addition, the lack of a reference platform for many applications makes it impossible to find a reliable granularity.

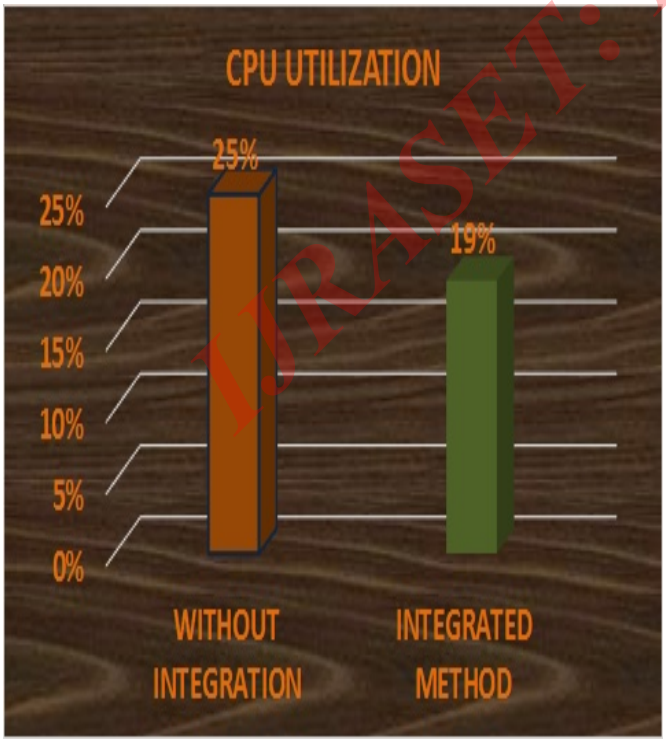


Fig.9. CPU Loading Time Performance Comparison chart.

The table plots the tradeoff between average accuracy and simulation speed over all task sets. Fig 7 shows that the overview of the hardware implementation. As can be seen decreasing the timing granularity results in higher accuracy but comes at a loss in simulation performance. By contrast, our predictive model provides both fast and accurate results regardless of the timing granularity. In order to evaluate our approach under realistic conditions with HW/SW interactions, we also simulated a task set composed out of a subset of applications from the automotive category of the MI Bench suite.

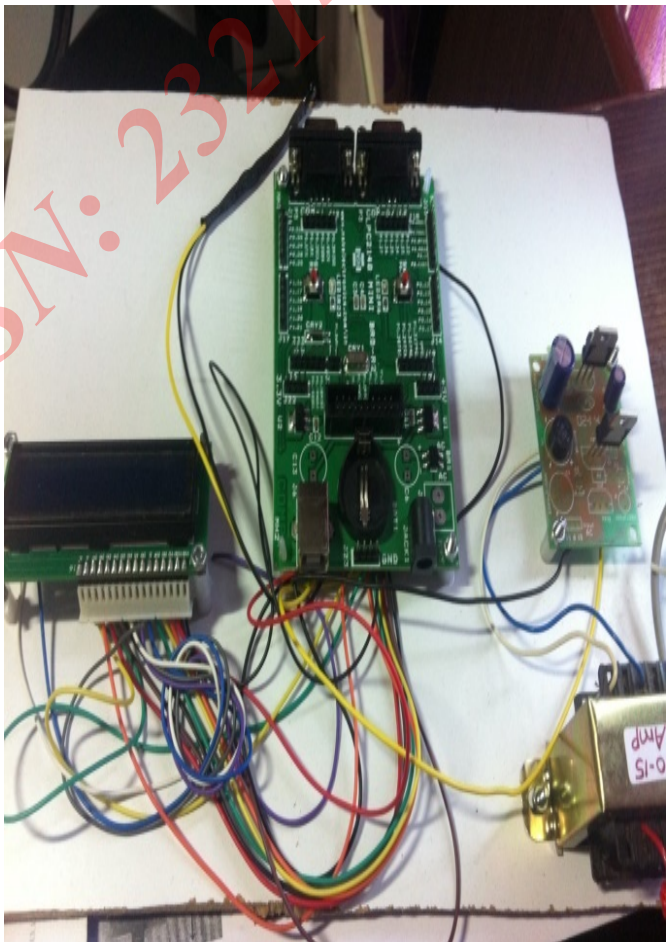


Fig.10.Hardware Setup.

Benchmarks were converted to execute periodically and concurrently based on rate monotonic scheduling policy, where task Susan (edge) was modified to interact with an FPGA bystreaming its outputs over the system bus. The resulting task set was simulated for 500 s.

CONCLUSION

We presented a predictive RTOS model designed for host-compiled software simulation of real-time periodic task sets. The simulator automatically adjusts the granularity of back-annotated delays by predicting the next task preemption point. Our model combines very fast simulation speed with error-free scheduling, which makes host-compiled simulators suitable for rapid, early evaluation of the real-time performance of periodic task systems within a HW/SW co-simulation context. In this work, we focused on solutions for avoiding errors in the preemption model. We have started to integrate this approach into our full host-compiled processor model and system simulator, which support sporadic tasks and inter- and intra-processor task communications. In the future, we plan to include models of cache, pipeline, and other dynamic effects that influence preemption costs, task execution times and hence, accuracy of overall real-time scheduling behavior.

REFERENCES

[1] ParisaRazaghi and Andreas Gerstlauer,"Predictive OS Modeling for Host-Compiled Simulation of Periodic Real-Time Task Sets," IEEE Embedded Systems Letters, Vol. 4, No. 1, March 2012.

[2] J. Ceng, W. Sheng, J. Castrillon, A. Stulova, R. Leupers, G. Ascheid, and H.Meyer, "A high-level virtual platform for early MPSoC software development," , Sep. 2009.

[3] K. Lin, C. Lo, and R. Tsay, "Source-level timing annotation for fast and accurate TLM computation model generation," *ASP-DAC*, Jan. 2010.

[4] T. Meyerowitz, A. Sangiovanni-Vincentelli, M. Sauermann, and D. Langen, "Source-level timing annotation and simulation for a heterogeneous multiprocessor," *DATE*, Mar. 2008.

[5] H. Posadas, J. A. Adamez, E. Villar, F. Blasco, and F. Escuder, "RTOS modeling in SystemC for real-time embedded SW simulation: A POSIX model," *DAES*, vol. 10, no. 4, Dec. 2005.

[6] J. C. Prevotet, A. Benkhelifa, B. Granado, E. Huck, B. Miramond, F. Verdier, D. Chillet, and S. Pillement, "A framework for the exploration of RTOS dedicated to the management of hardware reconfigurable resources," *Reconfigurable Computing and FPGAs*, 2008.

[7] A. Bouchhima, I. Bacivarov, W. Yousseff, M. Bonaciu, and A. Jerraya, "Using abstract CPU subsystem simulation model for high level HW/SW architecture exploration," *ASPDAC*, Jan. 2005.

[8] G. Schirner, A.Gerstlauer, and R. Dömer, "Fast and accurate processor models for efficient MPSoC design," *TODAES*, vol. 15, no. 2, Feb. 2010.

[9] M. Krause, D. Englert, O. Bringmann, and W. Rosenstiel, "Combination of instruction set simulation and abstract RTOS model execution for fast and accurate target software evaluation," *CODES+ISSS*, Oct. 2008.[10] R. S. Khaligh and M. Radetzki, "Modeling constructs and kernel for parallel simulation of accuracy adaptive TLMs," *DATE*, Mar. 2010.

[11] G. Schirner and R. Dömer, "Introducing preemptive scheduling in abstract RTOS models using result oriented modeling," *DATE*, Mar. 2008.

[12] F. Singhoff, J. Legrand, L. Nana, and L. Marce, "Cheddar: A flexible real time scheduling framework," *ACM SIGAda Ada Letters*, vol. 24, no. 4, pp. 1–8.

[13] RTSIM: Real-Time Simulator [Online]. Available: http://rtsim.sssup.it

[14] P. Razaghi and A. Gerstlauer, "Host-compiled multicore RTOS simulator for embedded real-time software development," *DATE*, Mar. 2011.

[15] OVP: Open Virtual Platform [Online]. Available: http://www.ovpworld.org

[16] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," presented at the WWC Dec. 2001.

[17] P. Razaghi and A. Gerstlauer, "Automatic timing granularity adjustmentfor host-compiled software simulation," *ASPDAC*, Jan. 2012.

AUTHORS PROFILE

Vinothini Narayanan[1] received the Bachelor of Engineering in Electrical and Electronics Engineering from P.S.N.A. College of Engineering & Technology., Dindigul, and Anna University Chennai in 2010.Currently doing Master of Engineering in Embedded Systems in RVS CET, Dindigul, and Anna University Chennai and worked as a Lecturer in K.Ramakrishnan College of Engineering in Trichy for the period of two years. And have published research papers in various International Journal and International Conference. Her research area includes control systems, Embedded Automotive Networking with CAN, power electronics, embedded systems.

Uma Maheswari T[2] received the Bachelor of Engineering in Electrical and Electronics Engineering from P.S.N.A. College of Engineering & Technology., Dindigul in 2004 and received Master of Engineering in Power Electronics & Drives from P.S.N.A. CET, Dindigul in 2008. And currently working as an Assistant Professor in RVS CET, Dindigul for past five years. And have published research papers in various International Journal and International Conference.Her research area includes power electronics, and power system, power quality and High voltage engineering.

.
Kannan K[3]received the Bachelor of Engineering in Electrical and Electronics Engineering from RVS College of Engineering & Technology., Dindigul in 2006 and received Master of Engineering in Power Electronics & Drives from S.V.C.E Chennai in 2009. And currently working as an Assistant Professor in RVS CET, Dindigul for past five years. And have published research papers in various International Journal and International Conference.His research area includes power electronics, power system, and power quality.

Sankar M[4] received the Bachelor of Engineering in Electrical and Electronics Engineering from PSNA CET,Madurai Kamaraj University, Dindigul in 2001, received Master of Engineering in Applied Electronics from RVSCET,Anna University in 2006 received Master of technology in Computer and Information Technology from MS University in 2012. Currently pursuing Ph.D., in Information and CommunicationEngineering in Anna University. And currently working as Head of department in RVS CET, Dindigul. And having experience of 12 years. And have published research papers in various International Journal and International Conference.His research area includes power electronics, power system, and power quality.

****