International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 5, No.6 (Nov-2016)
http://dx.doi.org/10.12785/ijcds/050606

Adaptive Fuzzy Hardware Scheduler for Real Time Operating System

D. G. Harkut[1] and M. S. Ali[2]

[1] *Department of Computer Science & Engineering, Prof Ram Meghe College of Engineering & Management, Badnera-Amravati (M.S.) India.*
[2] *Prof Ram Meghe College of Engineering & Management, Badnera-Amravati (M.S.) India.*

Received 21 Dec.2015, Revised 16 Jan. 2016, Accepted 19 Feb. 2016, Published 1 Nov. 2016

Abstract: In embedded system, a real-time operating system (RTOs) is often used to structure the application code and ensure that the deadlines are met by reacting on events in the environment by executing the functions within precise time. Most embedded systems are bound to real-time constraints with determinism and latency as a critical metrics. Generally RTOs are implemented in software, which in turns increases computational overheads, jitter and memory footprint which can be reduced even if not remove completely by utilizing latest FPGA technology, which enables the implementation of a full featured and flexible hardware based RTOs. Scheduling algorithms play an important role in the design of real-time systems. This paper proposes the novel FIS based adaptive hardware task scheduler for multiprocessor systems that minimizes the processor time for scheduling activity which uses fuzzy logic to model the uncertainty at first stage along with adaptive framework that uses feedback which allows processors share of task running on multiprocessor to be controlled dynamically at runtime. This Fuzzy logic based adaptive hardware scheduler breakthroughs the limit of the number of total task and thus improves efficiency of the entire real-time system. The increased computation overheads resulted from proposed model can be compensated by exploiting the parallelism of the hardware as being migrated to FPGA.

Keywords: Task Scheduling, Scheduling Algorithms, Fuzzy Inference System, Hardware Scheduler, Real-time Operating System, Determinism, Jitter, Reconfigurable Computing, FPGA, Priority Queue.

1. INTRODUCTION

Today's consumer market is driven by technology innovations. Many technologies that were not available a few years ago are quickly being adopted into common use. Equipment for these services requires microprocessors inside and can be regarded as embedded system. Embedded devices are often designed to serve their unique purpose and are included in a variety of products within different technical areas such as industrial automation, consumer electronics, automotive industry and communications and multimedia systems. Embedded systems find application in almost all the product ranging from train and airplanes to microwave ovens and washing machines. As semiconductor prices drop and their performance improves, there is a rapid increase in the complexity of embedded applications. The increased complexity of embedded applications and the intensified market pressure to rapidly develop cheaper product have caused the industry to streamline software development. Use of embedded operating system or Real Time Operating System (RTOS) is one technique used to reduce development time of such system as it has effects on hardware abstraction, multitasking, code size, learning curve and the initial investment. Unfortunately, operating systems do introduce several forms of overheads.

FPGAs have been the reconfigurable computing mainstream in recent time. Gate-level reconfigurability supports of FPGA results in reducing the development time to market and cost as compared to ASIC's which can be exploited to harness the benefit of developing the full featured and flexible hardware based RTOs.

Real time systems are embedded systems in which the correctness of application implementations is not only dependent upon the logical accuracy of its computations, but its ability to meet its timing constraints as well [1]. Thus the design of the RTOses have dual goal of minimizing the overheads and maximizing the determinism.

E-mail address: dg.harkut@gmail.com, softalis@hotmail.com

This paper is organized as follows. Section 2 is an overview of the Hardware/Software co-design approaches. Section 3 describes related work of other research projects, proposed model is discussed in section 4 and section 5 covers summary and conclusion from mainly previous work and related work.

2. HARDWARE SOFTWARE CO-DESIGN ARCHITECTURE

RTOs are often used in embedded systems to structure the application code to ensure that deadlines are met. The notions of best-effort and real-time processing have fractured into a spectrum of processing classes with different timeliness requirements including desktop multimedia, soft real-time, firm real-time, adaptive soft real-time and traditional hard real-time [2-4]. Many Real-Time systems are hard and missing deadline is catastrophic where as in soft real-time system, occasional violation of deadline may not result in useless execution of the application but decreases utilization [5].

Traditionally RTOS's are implemented in software, but major drawbacks of standard software based RTOS's is that they suffer from computational overheads, indeterminism, jitter and often a large memory footprint. RTOS computational overheads is caused mainly by tick interrupt management, which get even worse with more task and high tick frequencies, but also task scheduling , resource allocation and de-allocation, deadlock detection and various other OS/API functions take execution time from the task running on the CPU.

Embedded system always consists of software and hardware components and can no longer depend in independent hardware or software solutions to real time problem due to cost, efficiency, flexibility, upgradability, scalability and development time.

Task implemented as software programs running on microprocessor have the properties of high flexibility but poor performance. On the other hand, task implemented as hardware modules placed in Hardware have the characteristics of high performance along with low flexibility and high cost. The FPGA technology, which can be programmed virtually an *n* number of times (depends upon the technology), which paved the way for enhanced flexibility and made it possible to implement established software algorithms in hardware i.e. real-time kernel activity like scheduling, inter-process communications, interrupt management, resource management, synchronization and time management controls. Algorithm implemented in hardware has unique characteristics of high level parallelism and improved determinism that consequently decreases system overhead, improve predictability and increases response time.

As a tradeoffs, reconfigurable and hardware/software co-design approaches that offer real time capabilities while maintaining flexibility to support increasing complex systems become more feasible solution to allow software tasks running on a microprocessor along with hardware task running in an FPGA device (Figure 1). This hardware/software co-design approach reach a level of maturity that are allowing system designers to perform operating systems core and housekeeping functionality such as time management and task scheduling in hardware harness the advantages of higher level program development while achieving the performance potential offered by executions of these functions in parallel hardware circuits.

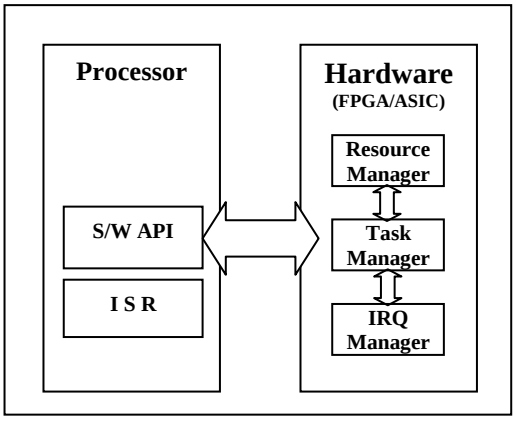

Figure 1 – Hw/Sw System Architecture

3. RELATED WORK

The main source of indeterminism in real time systems are varying instruction cycle time caused by pipeline, caches, varying execution time of RTOs kernel functions, external asynchronous interrupts etc. By migrating real time kernel from software to hardware it is possible to remove jitter, lessen CPU overhead and improve the indeterminism due to cache and pipeline problems. Various models and systems have been proposed [6] to overcome this problem and some of them were discussed in remaining section.

Lennart Lindh *et al.* [7] proposed a system FASTCHART, an RISC based uniprocessor system which puts ID of tasks into various queues. It consists of hardware based RT kernel capable of handling 64 tasks with 8 different priorities.

POLIS - proposed by F. Balarin, G. Berry, F. Boussinot *et al.* [8], is an HW SW Co-Design Finite State Machine (CSFM) synthesis model, which supports globally asynchronous and locally synchronous computation. Implementation is splits between Software

and ASICs and Co-simulation is provided using Ptolemy environment [9]. Complexity of processors makes static estimation is difficult and does no support for large design as it generates customized C-code for selected processors only.

Lennart Lindh *et al.* [10] also proposed FASTHARD which supports features like rendezvous, external interrupts, periodic start and termination of task without CPU interference. However system is limited in supports for customization and scalability. It is extension to earlier work FASTCHART, based on general purpose processors. Paper does not provided any benchmarks or test results.

The COSYMA system proposed by [11] uses simulated annealing for partitioning which can be fine or coarse grained, to speedup software executions to meet timing constraints. It does not support burst-mode communication. List and path based techniques are used to estimate execution time of hardware.

J. Adomat *et al.* [12] come up with RTU (Real Time Unit), a multi-processor system which uses single interrupt input of each CPU to control and context switching. Lindh *et al.* [13] also proposes extensible multiprocessor system - SARA, which can be used together with RTU to remove the all scheduling and tick processing overheads.

STRON system, based on µTRON project proposed by T. Nakano *et al.* [14] come up with hardware kernel which implements system calls and functionality results in increasing speedup and reducing jitter. This hardware kernel is supported by small micro kernel has been implemented to take care of the features not implemented in hardware. This system has tick frequency limitations and does not have hardware support to prevent unbounded priority inversion.

In order to minimize hardware cost while maintaining timing constraints, R. Gupta *et al.* developed VULCAN [15] Hardware/Software partitioning tool, which uses heuristic graph partitioning algorithm that runs in polynomial time. The original description was in Hardware-C [16], which is mapped to fine grained Control-Data Flow Graph.

Hardware software co-design framework for embedded system- CHINOOK, proposed by P. Chou *et al.* [17,18] is an automated interface synthesis which supports mapping of an embedded system model to one or more processor and peripherals. Though more emphasis is put on distributed architecture which ensuring timing constraints but system is inflexible and more complex.

A heterogeneous hardware/software DSP system CoWare in [19] proposed by H. De. Man *et al.*, is basis of commercial CoWare N2C [20]. This system supports the re-use and encapsulation of hardware and software by a clear separation between functional and communication behavior of a system components. Though this system allows co-specification using VHDL, DFL, Sliage & C languages, but imposes increased demands on generation of exhaustive library elements.

Bjorn B. Brandenburg *et al.* [21] discuss a soft real-time extension of the Linux kernel, the LITMUS^{RT} project with focus on multiprocessor real-time scheduling and synchronization. It supports the sporadic task model with both partitioned and global scheduling [22]. The primary goal is to provide a useful experimental platform for applied real-time systems research but LITMUS^{RT} failed to establish as stable interfaces.

F-Timer framework suggested by A. Parisoto *et al.* [23] is FPGA based task scheduler capable of managing 32 tasks with 64 different priorities which is targeted at general purpose processor. System does not have any hardware support for task synchronization and resource handling. Paper does not discussed about scheduling algorithm employed.

Spring kernel is basically designed for large and complex multiprocessor based RTOS proposed by J. Stankovic *et al.* [24,25] takes a radically different approach to task scheduling which is based on dynamic and speculative planning implemented through heuristic algorithm and tree search. Fine granularity of task deadlines is possible at the cost of large amount of pre-calculation overheads which affects the performance.

Hardware scheduling accelerator which can be configured for several different algorithms is proposed by J. Hildebrandt *et al* in [26,27]. This hardware implementation of dynamic scheduling coprocessor also supports advanced Enhanced Least Laxity First (ELLF) algorithm. This system could not address trashing of task but increases the overall determinism at the cost of higher complex logic.

δ-Framework- a hardware/software co-design RTOs framework proposed by V. Mooney *et al.* in [28], supports 30 different processors. The system is cost effective as far as overall speedup and hardware area (number of gates) is concerned. This framework generates all HDL code which can be implemented in FPGA. More work on SOC was conducted [29] to integrate priority inheritance and deadlock avoidance mechanism.

Configurable hardware scheduler with improved response time, interrupt latencies, CPU utilization has been design and developed by V. Mooney *et al.* [30],

which also supports high tick frequency. This model supports three different algorithms which can be change at run time dynamically and interrupt controller in scheduler supports 8 external interrupts each can be configured for dispatching a specific task.

Issues of extension to OS and flexibility arises out of moving entire OS to hardware can be overcome in model propose by Z. M. Wirthlin *et al.* in [31]. The nano-processor provides upgradability, flexibility and also enhancing the execution time by moving selected inefficient OS services in hardware to save on power consumption to a great extent as shown in [32].

Paul Kohot *et al.* in [33], developed Real-Time Manager (RTM) which leverages the potential of hardware parallelism, In this system, routine housekeeping tasks are implemented in hardware and thus free the processor for critical functions which boosts the overall performance. RTM supports static priority scheduling and handles task, time and event management. The author claims RTM decreases RTOS overheads by 90% decreases response latency by 81%.

Problem arises out of low tick granularity can which cause jitter and result in deadline misses is overcome by M.Vetromille *et al.* [34] in their proposed system HaRTS. The HaRTS supports high tick frequency and thus reduce jitter without lower CPU available time for task to process. Though it is more complex to implements but it requires less chip area and uses less power than additional processor.

The Hardware RTOS implemented for accelerating eCos, HW-eCos is interfaced to an ARM processor requires fewer gates to implement and provides better speedup. Communication speed between RTOS and hardware overshadowed the speed gain by hardware scheduler is overcome by S. Chandra *et al.* in [35] by intelligent design. Paper does not discuss the number of tasks and resources supported by this system.

SRTOS proposed by Z. Murtaza, S. Khan *et al.* [36] aims at real-time DSP application which is targeted on AVZ21 DSP processor. Though this paper doesn't provide any experimental test result but system supports additional instruction for fast resource allocation and context switching.

M. Song *et al.* [37] come up with H-Kernel, an outcome of through use of FPGA and thoughtful HW/SW co-design for specific application. Though system become more complex and bulky as number of task increases but increase in performance in the tune of 50-60%, is achievable with the system with small numbers of task.

Sebastien Pillement *et al.* [38] proposed DART – an FPGA based reconfigurable architecture which deals concurrently with high-performance, flexibility and low-energy constraints. Flexibility of FPGAs is achieved at a very high silicon cost interconnecting huge amount of processing primitives. These interconnection and configuration overheads result in energy waste. DART was designed as a platform-based architecture which define cluster level interface to implement user dedicated logic which allows for the integration of application-specific operators which efficiently support bit-level parallelism. The main concern of this class of architectures is high reconfiguration overhead.

ARPA-MT multi-threading processor with five stage pipeline system is proposed by A. S. R. Oliveira *et al.*[39]. This system supports heterogeneous task and context switches without hampering the processor performance.

Latency introduced due to PLB bus interface in the system can be removed by better and more direct connections between CPU and coprocessor as proposed by Luis Almeida *et al.* in [40,41] OReK_CoP i.e. Hardware implementation of OReK Real-Time Kernel. All kernel functions execute in absolute time and almost in parallel, without interfering CPU which improves determinism and improve resource utilization.

Xaingrong Zhou, Peter Petrov *et al.* [42] presented model by converging compiler, micro-architecture and OS kernel to reduce the context switching cost and improve overall responsiveness which the main source of performance degradation in most of the HW SW based solutions. In this proposed model context switching may be deferred until next switch point to limit the number of context registers required to hold state. Though this arrangement results in more deadline miss which can be avoided by more complex and good RTOS kernel design.

ARTESSO architecture as proposed by N. Maruyama *et al.* in [43], ported RTOS, checksum calculation, memory copying and TCP header rearrangement to hardware. It uses novel virtual queue instead of FIFO based queues used in RTU and STRON, which are logic expensive. The author claims that this system is 6-9 times faster than STRON and 7 times more energy efficient than its software counterpart.

Numbers of research projects have approached the task of designing OS for FPGA based reconfigurable computers (RC). By providing native kernel support for FPGA hardware Hayden Kwok-Hay *et al.* [44-46] proposed BORPH, an operating system designed for FPGA-based RC. BORPH offers a homogeneous UNIX interface for both software and hardware processes. Hardware processes inherit the same level of service from the kernel.

Static scheduling of DAGs (Direct Acyclic Graph) on multi-reconfigurable-unit system under strict real-time constraints and from a parallel processing perspective is proposed by Ikbel Belaid *et al.* [47]. Clustering the task, mapping the task in these clusters and placing these clusters on reconfigurable devices, dynamic partial reconfiguration and efficient placement are achieved. However, this approach face difficulty in dealing with nondeterministic systems with run-time characteristics that are not well known before the DAG running and this approach will work only for small DAGs.

HartOS- Hardware implemented Real-Time Operating System is proposed by Lange A.B. *et al.* [48,49] is designed to be very flexible and support most of the features normally found in a standard software RTOS directly in hardware without sacrificing flexibility. The HartOS's ability to run kernel at a higher clock frequency than the microprocessor, enables more tasks to be processed serially at the same tick frequency and thus speed up the part of the API functions executed in the kernel. Comparative study of various methodologies/models reviewed in the literature is given in the Table 1[50].

Table 1 - Comparative study of various methodologies/models

Methodology/ Model	Architecture Used & Claims by Authors
FASTCHART (1991) *Hybrid* [7]	RISC based processor with Load Store architecture. Migrated full kernel to Hardware to improve determinism and remove jitter.
POLIS (1991) *Hybrid* [8]	Co-design Finite State Machine (CFSM) design. Flexibility to evaluate HW/SW partitioning, architecture & scheduler through mixed implementation of SW & ASICs.
FASTHARD (1992) *Hybrid* [10]	Memory mapped design (address/data bus). HW based RT Kernel to support external interrupts & rendezvous.
RTU (1994) *H/W based* [12]	Memory mapped design (VME bus). Supports multiple task, binary semaphores, event flags, watchdogs with minimum overheads and improved predictability.
Silicon TRON (1995) *Hybrid* [14]	Memory mapped design (address/data bus). Improve determinism and supports task mgt., flags, semaphores, timers & external interrupt.
VULCAN (1995) *Hybrid* [15]	CDFG based fine grained mapping design. Hardware/software partitioning results in reducing the overall cost.
CHINOOK (1996) *Hybrid* [17]	Distributed Architecture. Supports mapping of processor & peripherals with strict timing constraints with automated interface synthesis.
COWARE (1996) *Hybrid* [19]	Memory mapped design (address/data bus). Supports re-use, encapsulation of HW & SW by separation of functional behavior to supports heterogeneous HW/SW DSP systems.
COSYMA (1997) *Hybrid* [11]	Memory mapped design (address/data bus). Uses novel list & path-based scheduling to estimate HW execution time & speedup SW executions to meet timing constraints.
F-Timer (1997) *Hybrid* [23]	Memory mapped design (address/data bus). Supports external interrupts by reducing overall RTOs overheads with improved determinism.
Spring Coproc (1999) *Hybrid* [25]	Memory mapped design (address/data bus). Supports fine granularity of task deadlines & multiprocessors with guaranteed scheduling without blocking resources.
ELLF Sched. Coproc. (2000) *Hybrid* [26]	Memory mapped design (address/data bus). Supports ELLF algorithm with dynamic priority calculation by exploring parallelism in HW.
The δ-Framework (2002) *Hybrid* [28]	Memory mapped design (address/data bus). Uses less nos. of gates for equivalent HW area targeted for HW/SW co-design.
Mooney (2003) *Hybrid* [29]	Memory mapped and instruction set acceleration based design. Configurable scheduler which supports Priority based, Rate monotonic & EDF algorithms & high tick rate.
Nano-processor (2003) *Hybrid* [31]	Memory mapped design (address/data bus). Provides flexibility of choosing services to perform in HW with faster execution with compatibility with range of hardware.
RT Task Manager (2003) *Hybrid* [33]	Memory mapped design (address/data bus). Supports static priority & handles task, time & event mgt. with same tree by migrating routine task to HW.
HaRTS (2006) *Hybrid* [34]	OPB Bus Scheme based design. Requires less power, less chip area and supports high tick frequency and granularity with lowering jitters.
LITMUSRT (2006) *S/W based* [21]	Push/Pull approach. Effective testbed to evaluate diff RT Scheduler & also supports G-EDF based scheduling with private queue for each processor.
HW- eCos (2006) *Hybrid* [35]	Memory mapped design (address/data bus). Removes context switching overheads through interrupt line to CPU, reduce code size and thus improve performance.
Silicon RTOS (2006) *Hybrid* [36]	Memory mapped design (address/data bus). Supports external interrupt management & uses priority based scheduling to make RT DSP applications efficient.
H-Kernel (2007) *Hybrid* [37]	Memory mapped design (address/data bus). Supports priority based task, interrupt, event & time mgt through H-kernel and performance through thoughtful HW/SW co-design.
OReK_CoP (2009) *Hybrid* [41]	PLB bus interface with stack based priority ceiling design. Ported OReK kernel to HW to improve performance & supports asynchronous interrupt handling which improve determinism
Xiangrong et al (2010) *S/W based* [42]	Micro-architecture & OS kernel. Uses micro-architecture to lower context switching and improve responsiveness.
ARTESSO (2010) *Hybrid* [43]	TCP/IP protocol. Improve throughput by moving TCP Header calculations to HW & supports priority based FCFS scheduler by using novel virtual queue structure.
BORPH (2011) *S/W based* [45]	OS uses Virtual file system. Reduces context switching drastically by exploiting the benefits of parallelism and FPGA reconfigurability.
ARPA-MT (2011) *Hybrid* [38]	Stack based priority ceiling design. Specialized, Predictable and customized Processor design which supports heterogeneous task & schedules using RM or EDF protocol.
HartOS (2012) *Hybrid* [48]	FSL-AXI stream interface. Interrupt handled as task & mutex are protected by stack based priority ceiling which reduces jitters and memory footprints.

Scheduling algorithm plays as important role in the design of real-time systems which involves allocation of resources and time to jobs in such way that certain performance requirements are met.

Most of the model discussed and reviewed are mainly focused on to improve the performance by migrating some of the house keeping routine jobs from software to hardware with a aim to leverage the potential of parallel processing of hardware which can further be improved to a greater extent if more realistic scheduling algorithm is devise and migrate it on hardware to assist processor and RTOs so as to increase the overall performance without increasing memory footprint and power consumptions.

4. HARDWARE TASK SCHEDULER

Mostly researchers dealing with real-time system scheduling, assumes scheduling constraints to be precise. But in practical reality, the values of these parameters are vague in most of the cases. To overcome these limitation of vagueness of jobs scheduling parameters [51], Fuzzy logic play important role in generating most optimal scheduling which enhance the utilization of the resources and thus increases the overall schedulability of the system by treating these vague scheduling parameters are treated as fuzzy variables. In this research paper, a two phase adaptive scheduling algorithm is developed and migrated on FPGA to harness the potential of parallel processing which will compensate added computational cost for executing of complex fuzzy algorithms.

4.1 Architecture

We proposed Fuzzy Inference System (FIS) based adaptive hardware task scheduler framework which is discussed in subsequent paragraph basically consists of:

1. Global Fuzzy scheduler – Long term scheduler. (FIS 1)
2. Local Adaptive scheduler – Short term scheduler. (FIS II)

Both of these scheduler work in cascade and are migrated on hardware which will work in synchronous with processor and RTOs to fulfill the overall systems objectives as illustrated in figure 2.

To build a fuzzy system, inputs and output(s) to it must be first selected and partitioned into appropriate conceptual categories which actually represent a fuzzy set on a given input or output domain. Parameters which affects the schedulers performance are selected as input to the Fuzzy Inference System (FIS) [52,53], which consist of five stages:

1. Fuzzifying inputs
2. Applying fuzzy operators
3. Applying implication methods
4. Aggregating outputs
5. De-fuzzifying outputs

Here Madani's Fuzzy inference method of TSK or simply Sugeno method of fuzzy inference may be used [54-56].

Block diagrams of FIS I and FIS II along with the parameters selected as Input and Output are along with surface viewer are shown in figure 3.

Input to FIS I are - Job Exterior Priority (JEP), Job Processing Priority (JPT) & Job Waiting Time (JWT) which generates Job Processing Priority (JPP).

Input to FIS II are – Job Processing Priority (JPP) generated by FIS I and Job Worst Case Execution Time (JWCET) which generates Job Final Priority (JFP).

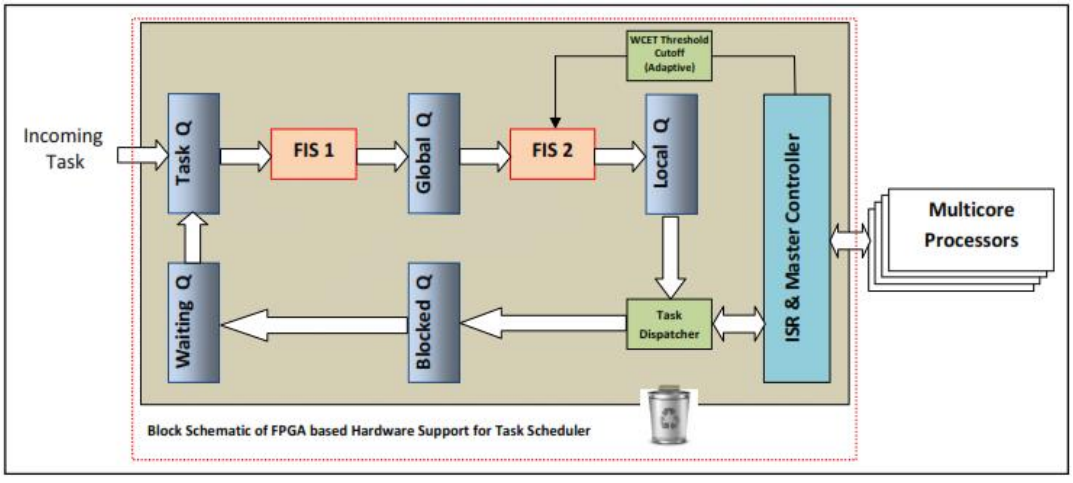

Figure 2. Proposed FIS based Adaptive Hardware Task Scheduler

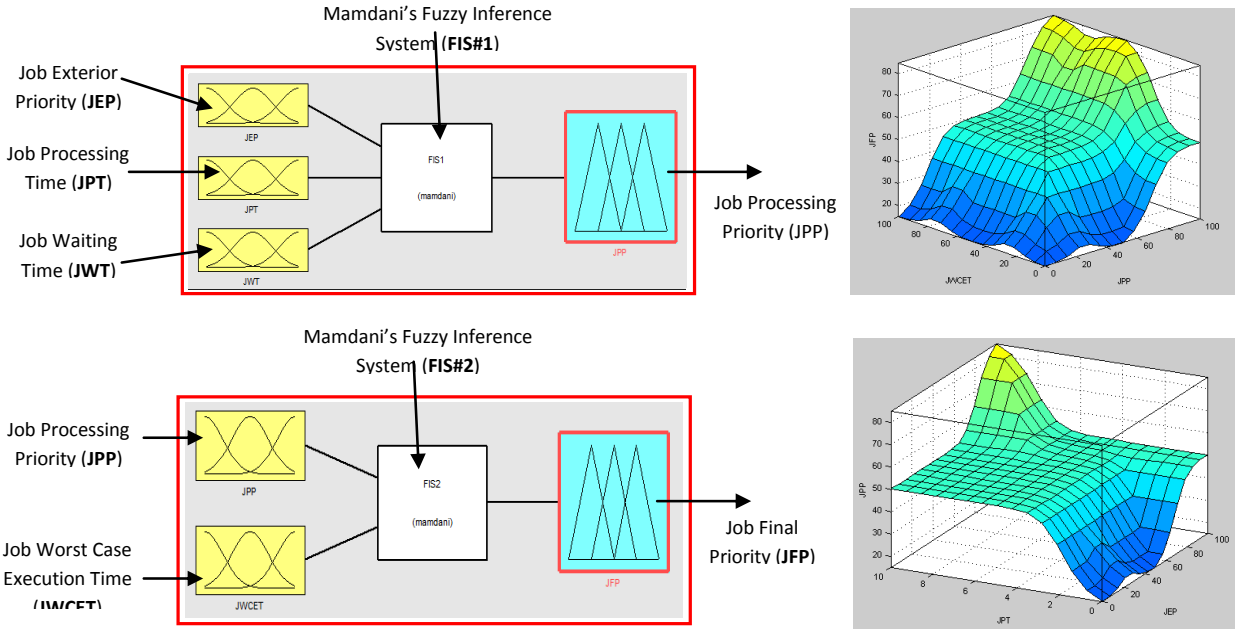

Figure 3. Fuzzy Inference System I & II block diagram

Output of the FIS I is single value which is treated as Job Processing Priority (JPP) and maintained in global queue in sorted order. This newly calculated JPP along with task's worst-case execution time (WCET)[57], feed to FIS II, a second stage scheduler.

The working of proposed novel Two phase Fuzzy Inference System based hardware task scheduler which uses fuzzy logic to model is depicted as –

An arrival of new task in system initiates the application. These new task are stored in Arrival Queue in First-in-First-out manner (FIFO) waiting to be get processed by the Fuzzy Inference System (Phase I). Task entering the systems are tagged with some basic parameters which play important role in scheduling these task. These jobs are stored in sorted order as per newly calculate Job Processing Priority (JPP). Task queued in Global queue are feed to Fuzzy Inference System (Phase II). Local Queue holds the task in sorted order as per the Job Final Priority (JFP) calculated by FIS 2. Master controller keeps track of actual execution time (AET) of each task being processed and if the difference between Worst Case Execution Time (WCET) and AET for a task in beyond certain threshold value i.e. δ (t), then is it notified back to FIS II which will update the value of WECT by AET and consider this new updated value of WCET during next scheduling cycle. Task blocks on shared resources are stored in Block Queue where semaphore is used to resolve the deadlock and task are moved from block queue to Waiting Queue if the task is yet to be complete. These tasks are then added back to Arrival Queue along with newly entered task in FIFO order.

4.1.1 Adaptive Fuzzy Scheduling

Under traditional task model like periodic, sporadic etc., the schedulability of system is based on each task's worst-case execution time (WCET), which defined the maximum amount of time each of its jobs can execute. The disadvantage of using WCETs is that system may be deemed un-schedulable even if they would function correctly most of the time when deployed. This drawback can be overcome by making our scheduler adaptive to the runtime varying conditions, to allocate per-task processors time share, instead of always using constant share allocation based on constant WCET and readjusting the priority of task. When there is variation in the WCET and the actual execution time of a particular job beyond some predetermined threshold value, adaptive task schedulers is invoked with actual execution time and reschedule the task and refresh and reorder the tasks in local queue accordingly. This results into adjusting the per task processor time share based on the runtime conditions which will effectively increases the overall schedulability and processor utilization. Overall quality-of-service (QoS) can be improved by ignoring the transient overload conditions. Dispatcher will dispatch the task from local queue to processors bank to get serve.

Further resource synchronization is used to optimize scheduling of the tasks blocked on shared resource which are parked on blocked or waiting queue. Task blocks on shared resources are stored in Block Queue are moved

Overall Process Flow –

DO

1. Incoming Task arrives and stores at Task Arrival Queue
2. Task came with following tags:
   - Job External Priority (JEP)
   - Job Processing Time (JPT)
   - Job Waiting Time (JWT)
   - Job Worst Case Execution Time (JWCET)
3. These tasks are fed to Fuzzy Inference System (FIS I) and Generates Task Processing Priority (JPP).
4. Tasks are stored in Global Queue as per newly calculated Processing Priority (JPP).
5. These tasks are having following tags:
   - Job Worst Case Execution Time (JWCET)
   - Job Processing Priority (JPP)
6. These tasks are now fed to ***Adaptive*** Fuzzy Inference System (FIS II) and Generates Jobs Final Priority (JFP).
7. Tasks are stored in Local Queue (Hardware based Priority Queue) as per newly calculated Jobs Final Priority (JFP).
8. Task Dispatcher will perform following tasks:
   - Dispatches the ready task from Local Queue to Master Controller
   - Collects the Finished or Blocked task from Master controller
   - Forward the Blocked task to Block Queue
   - Flush-out the finished task from System
9. Task Dispatcher will fetch the ready to run scheduled tasks from Local Queue and dispatch it to ISR and Master controller.
10. Master Controller will assign the task to pool of processors (multi-core) by maintaining the proper load balancing.
11. Task dispatcher collects and dispatches the blocked tasks (blocked on shared resources) from master controller and stored them on Blocked Queue.
12. Task dispatcher also flush-out the finished tasks
13. Task moves from Blocked Queue to the Waiting Queue, once shared resources are free after resolving the aging issue by resource synchronizer.
14. Task from Waiting Queue and Incoming task from outer environment are replenished in Task Arrival Queue in FIFO order.

LOOP

Feedback process – (Adaptive Scheduling)

DO

1. FIS II calculate the Job Final Priority (JFP) based on two tags: Worst Case Execution Time (WCET) and Job Processing Priority (JPP).
2. Master Controller keeps track of each task Actual Execution Time (AET) during execution.
3. If the difference between WCET & AET exceeds certain predefined τ, WCET of that job is updated by AET.
   i.e. (WCET –AET) >= τ, WCET← AET
4. FIS II will use this modified value of WCET in next turn of this task for generating Job Final Priority (JFP) which is more rational.

LOOP

from block queue to Waiting Queue if the task is yet to be complete. These tasks are then added back to Arrival Queue along with newly entered task in FIFO order. Resource synchronization module which implements priority queue with aging to avoid the task starvation and thus improve chance of fair treatments to all the tasks in the queue is used to remove the deadlocks on resources among task from block task queue which will increase the overall performance of the RTOs. Processors share allocations are adjusted using feedback and resource synchronization techniques [58].

Fine grained time management and frequent sorting and re-arrangements of tasks in Local Queue and Waiting Queue increases the CPU overhead and thus affects the processor utilization which can be overcome by implementing these queues as hardware priority queue fig.4.

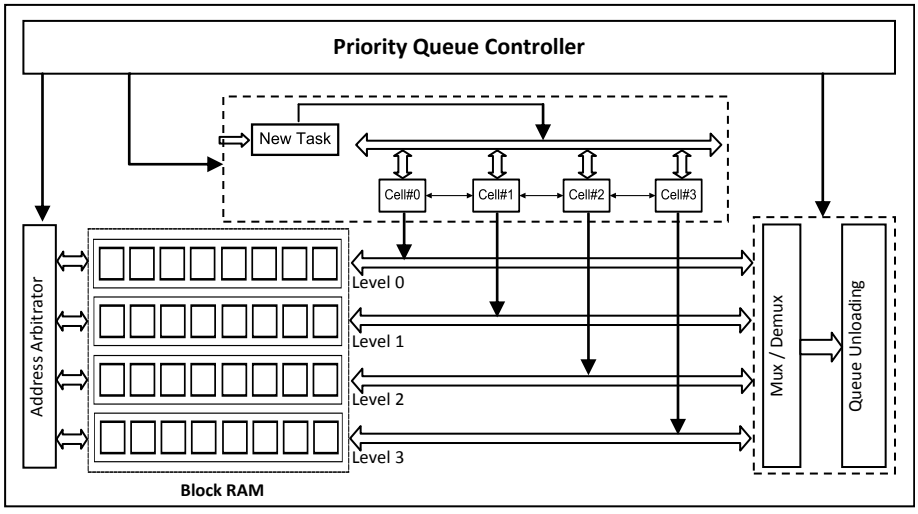

Figure 4. Hardware Priority Queue architecture

Queue Loading process – (Hardware Priority Queue)
DO
1. New task arrived
2. Calculate the path from vacant leaf node. This path includes all the ancestors from leaf node to the heap's root. (Heap property always ensures that the elements are sorted)
3. Address arbitrator generates the addresses of all the nodes from this identified path.
4. Nodes from this path from stored at Block RAM are mapped and loaded to enqueue cells in queue loader.
5. Newly arrived tasks priority is broadcasted to all the enqueue cells simultaneously in shift register.
6. Comparator in enqueue cells compares, swaps and latch the newly task at appropriate enqueue cell.
7. Updated Data from enqueue cells is loaded back to the Block RAM and thus the elements in heaps are sorted with accommodating the newly arrived task

LOOP

Queue Un-loading process – (Hardware Priority Queue)
DO
1. Remove the highest priority task stored at root node and replace it with last leaf to balance the heap.
2. In order to ensure the heap's basic property, rearranged the elements
3. Compare and swap the root node with immediate child node so that heap is sorted.
4. Move to next child node level till all the nodes are exhausted.

LOOP

Resource Synchronization Process –
DO
1. Task Arrived in Task Blocked Queue on share resources.
2. Calculate the Task Unload Priority to move task from Task Blocked Queue to Task Waiting Queue.
   i.e. UP = JPP * AP where UP = Task Unload Priority, JPP = Task Processing Priority and AP = Task Aging Priority (if AP > φ (t))
3. Migrate the task having highest UP from Task Blocked Queue to Task Waiting Queue.
4. At each migration from Task Blocked Queue to Task Waiting Queue, increase the AP of all remaining task in Task Blocked Queue by 1 i.e. AP = AP + 1
5. Move the Task from Task Waiting Queue to Task Arrival Queue in FIFO order.

LOOP

4.1.2 Queue Loading process

Queue loading is accomplished by inserting the newly arrived task at the bottom of binary heap. Process of repeatedly comparing and swapping with adjacent parent node is performed until the priority of newly arrived task is less than its parents. Shift register mechanism shown in figure 4 inserts the newly arrived task in constant time. The heap property ensures that elements are sorted in order.

4.1.3 Queue Un-loading process

Remove the root task from the queue and reconstruction of the heap constituted the queue un-loading operation. Root element is removed by replacing it with the last element in the queue to keep the heap balanced. Process of repeatedly comparing and swapping with smallest of the child node is perform until the priority root node is less than its child. Highest priority value is obtained in constant time and as priority queue is managed in hardware, the processor is not required to wait for the operation to complete.

4.1.4 Resource Synchronization Process

Task which are blocked on shared recourses are park on blocked queue which is implemented as hardware priority queue. To avoid the task starvation and fair share of CPU time, Priority queue with aging technique is used. Task upload priority is calculated, which will used to decide which task next to be moved from blocked queue to waiting queue.

It is observe that, generally to ensure tasks must meet its deadline, the scheduler's WCET are often overestimated. This causes system to be under-utilise and wastes CPU resources. Here we have examined how the scheduler overheads and its variation can be reduced by migrating the scheduling functionality to hardware logic. Further by accommodating the varying WCET on runtime, in scheduling, there is a twofold increase in the idle time of CPU which can be utilised effectively and thus results in increase in overall performance, enhance system predictability and timing resolution.

An analytical result comparison of three different cases namely:

1. RTOS with Software Scheduler
2. RTOS with Hardware Scheduler &
3. RTOS with Adaptive Hardware Scheduler is depicted in figure 5.

CONCLUSION

The conclusion from a comprehensive literature review of the publication throughout the last three decades, is that the major drawback from software based RTO's can be removed by implementing the entire/ partial kernel of a real-time operating system in hardware. All past attempts to design a hardware RTOS

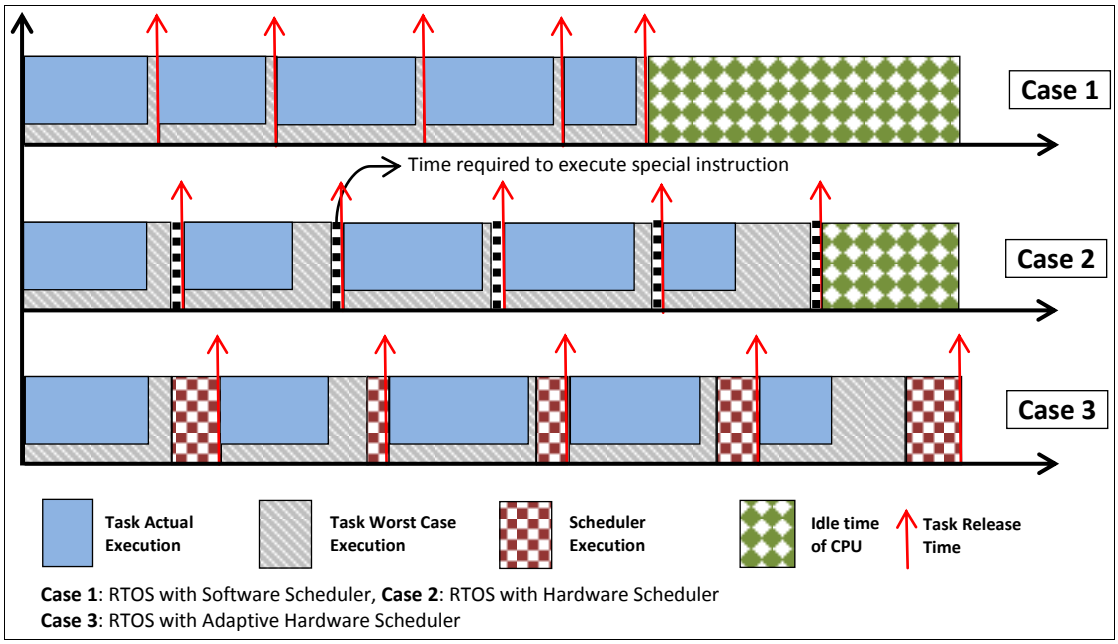

Figure 5. Scheduler Execution Time Variations

kernel has had limitations either in form of lacking key RTOS features/resources, being inflexible in terms of configurability or perhaps suffering from poor performance. By addressing the set of desired features, performance goals, clever design and utilization of the latest FPGA technologies, the implementation of a full featured and flexible hardware based RTOs is be possible which could address the shortcomings found in the literature.

A hardware Intellectual Property (IP) can be used for implementing routine frequently used housekeeping activities like scheduling, inter-process communication and time management control from the software OS-kernel to hardware unit. This result in significantly reducing the overhead by migrating kernel services to hardware which will improve the response time by increasing the CPU utilization. A hardware kernel executes in parallel to the CPU, minimizes the processor time for scheduling activity and thus relieves pressure from the CPU which gets almost full execution time for the application tasks. There is less software code in memory since the functionality is implemented in hardware instead [23].

A software OS will generate a clock tick interrupt to the CPU when either it is executed or the lists of tasks (queues) are worked at or new periodic delay times are calculated for the tasks. With the hardware kernel in the system, it checks all queues concurrently and only generates an interrupt to the CPU when there is to be a task switch [59,60]. Another advantage of having the kernel in hardware is the possibility to use complex scheduling algorithms, unlimited of different queue types without any performance loss.

When real-time kernels are implemented in software, one of the disadvantages is that the execution time for the service calls will have a minimum and a maximum time [61,62]. The time gap can be big and the worst-case time is one of the factors that will decide the utilization factor of the system. The scheduling time varies with the number of tasks and scheduling algorithm and must be bounded by a pessimistic worst case execution time, which decrease the determinism.

We have proposed two phase FIS based hardware task scheduler which uses fuzzy logic to model the uncertainty at first stage along with adaptive framework that uses feedback in second stage. Scheduling based on static WCET will results in lower utilization of processors, which can be overcome by adaptive feedback mechanism which will update the WCET parameter of the task with AET, if the difference between the WCET & AET is exceeding the pre define threshold value τ, which allows processors share of task running on multiprocessor to be controlled dynamically at runtime and thus increases the overall processor utilization and thus the schedulability. Further, Starvation of low priority task problem is overcome by Resource synchronization module which in turns avoids the aging of task. Because of high granularity, frequent sorting and updation of the tasks in queue increases the overhead which can be reduced to greater extent by using Hardware Priority Queue [63] to store the task which increase the sorting speed and thus lessen the burden of CPU. This increases

the overall utilization of CPU and increases the schedulability of the tasks.

Our future work is to map this proposed model on MicroBlaze soft processor core as MicroBlaze FPGA designs are readily available and can be implemented with little effort. The FreeRTOS port in MicroBlaze is being targeted to be modified and run tasks concurrently on multiple processors as FreeRTOS provides simple, easy to use and highly portable kernel. The aim to produce a version of FreeRTOS that supports multi-core hardware and efficient hardware based task scheduler.

REFERENCES

[1] D. Stewart, "Introduction to Real Time", Embedded systems programming, CMP Media, November 2001.

[2] Z. Deng, J.W. Liu and S. Sun, "Dynamic scheduling of hard real-time application in open system environment", Tech. Rep., University of Illinois at Urbana-Champaign 1996.

[3] G. Buttazzo and J. A. Stankovie, "RED: robust earliest deadline scheduling", in Proceeding of 3rd International Workshop Responsive Computing Systems, Lincoln, NH, pp. 100-111, 1993.

[4] S. M. Petters, "Bounding the execution time of real-time task on modern processors", in Proceeding of 7th International Conference Real-Time Computing Systems and Applications, Cheju Island, pp. 498-502, 2000.

[5] J. Zhu, T.G. Lewis, W. Jackson and R.L. Wilson, "Scheduling in hard real-time applications", IEEE software, Volume 12, pp. 54-63, 1995.

[6] D. G. Harkut & M.S.Ali, "Hardware Support for Real Time Operating System: A Review", in Proceedings of IEEE International Conference on Engineering and Technology (ICETECH'15), 2015.

[7] L. Lindh, F. Stanischewski, "FASTCHART - Performance, Benefits and Disadvantages of the Architecture", in Proceeding of 5th Euromicro Workshop on Real-Time Systems, 1993.

[8] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, K. Suzuki and B. Tabbara. "Hardware-Software Co-Design of Embedded Systems: The POLIS Approach", Kluwer Academic Publishers, 1997.

[9] J.T. Buck, S. Ha, E.A. Lee and D.G. Messerschmitt, "Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems", International Journal of Computer Simulation, special issue on "Simulation Software Development", pp.155-182, April 1994.

[10] L. Lindh, "FASTHARD - a fast time deterministic hardware based real-time kernel", in Proceedings of Real-Time Systems, 4th Euromicro workshop, pp. 21-25, June 1992.

[11] R Ernst, J. Henkel, Th. Benner, W. Ye, U. Holtmann, D Herrman and M. Trawny, "The COSYMA environment for hardware software co-synthesis of small embedded systems", IEEE Micro, pp.159-166, 1996.

[12] J. Adomat, J. Furunas, L. Lindh, and J. Starner, "Real-time kernel in hardware RTU: a step towards deterministic and high-performance real-time systems", in Proceedings of the 8th Euromicro Workshop on Real-Time Systems, L'Aquila, pp. 164-168, Jun. 1996.

[13] L. Lindh, T. Klevin, L. L. T. Klevin, and J. Furunäs, "Scalable architecture for real-time applications sara", in CAD & CG'99, pp. 208-211, 1999.

[14] T. Nakano, A. Utama, M. Itabashi, A. Shiomi and M. Imai, "Hardware implementation of a real-time operating system", in proceeding of IEEE International Symposium of 12th TRON project, Tokoy, Japan, pp. 34-42, Nov. 1995.

[15] R. Gupta. "Co-Synthesis of Hardware and Software for Digital Embedded Systems", the Springer International Series in Engineering and Computer Science, Volume 329, 1995.

[16] D.C. Ku and G. DeMicheli, "HardwareC - a language for hardware design Ver 2.0" CSL Technical Report CSL-TR-90-419, Stanford, April 1990.

[17] P. Chou, R. Ortega and G. Borriello, "The Chinook Hardware Software Co-Synthesis System", in Proceedings of the International Symphosium on System Synthesis, pp. 22-27, Sept. 1995.

[18] P. Chou, E. Walkup and G. Borriello. "Scheduling for Reactive Real-Time Systems". IEEE Micro archive Journal, IEEE Computer Society Press Los Alamitos, CA, USA. Volume 14, Issue 4, pp. 37-47, August 1994.

[19] H. De Man, D. Verkest, K. Van Rompary and I. Bolsens, "Coware - A Design Environment for Heterogeneous Hardware Software Systems", Design Automation of Embedded Systems, pp.357-386, Oct. 1996.

[20] S. Michael, "CoWare revs tool for SoC platform design", Electronic Engineering Times, pp. 54-58, August 2000.

[21] B. Brandenburg, A. Block, J. Calandrino, U. Devi, H. Leontyev and J. Anderson, "LITMUSRT: A Status Report", in Proceedings of the 9th Real-Time Linux Workshop, pp. 107-123, November 2007.

[22] B. Brandenburg , R. Spliet, M. Vanga and S. Dziadek, "Fast on Average, Predictable in the Worst Case: Exploring Real-Time Futexes in LITMUSRT", in Proceedings of the 35th IEEE Real-Time Systems Symposium, Rome, Italy, pp. 96-105, Dec.2014.

[23] Parisoto, J. Souza, A., L. Carro, M. Pontremoli, C. Pereira, and A. Suzim, "F-timer: Dedicated FPGS to real-time systems design support", in proceeding of 9th Euromicro Workshop on RTS, Toledo, Spain, pp. 35-40, Jun.1997.

[24] J. Stankovic and K. Ramamritham, "The spring kernel: a new paradigm for real-time systems", Software, IEEE, Volume 8, Issue 3, pp. 62-72, May 1991.

[25] J. Stankovic, W. Burleson, J. Ko, D. Niehaus, K. Ramamritham, G. Wallace and C. Weems, "The spring scheduling coprocessor: a scheduling accelerator", in IEEE Transactions on Very Large Scale Integration Systems, Volume 7, pp. 38-47, Mar. 1999.

[26] J. Hildebrandt, F. Golatowski, and D. Timmermann, "Scheduling coprocessor for enhanced least-laxity-first scheduling in hard real-time systems", in Proceedings of the 11th Euromicro Conference on Real-Time Systems, pp. 208-215, 1999.

[27] J. Hildebrandt and D. Timmermann, "An FPGA based scheduling coprocessor for dynamic priority scheduling in hard Real-Time systems", in Proceeding of 10th International Conference On Field Programmable Logic & Applications, Villach, Austria, pp. 777-780, 2000.

[28] V. Mooney, J. Lee, and K. Ryu, "A Framework for Automatic Generation of Configuration Files for a Custom Hardware/Software RTOS", in Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'02), pp. 31-37, June 2002.

[29] V. Mooney and J. Lee, "Hardware/Software Partitioning of Operating Systems: Focus on Deadlock Detection and Avoidance", in IEEE Proceeding, Computer and Digital Techniques, UK, pp. 167-182, July 2005.

[30] V. Mooney III, P. Kuacharoen and M. A. Shalan, "A configurable hardware scheduler for real-time systems", in Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms, CSREA Press , pp. 96-101, 2003.

[31] M. Wirthlin, B. Hutchings, and K. Gilson, "The Nano Processor: a Low Resource Reconfigurable Processor", in IEEE Workshop on FPGAs for Custom Computing Machines, Napa, CA, pp.23-30, April 2003.

[32] R. Dick, G. Lakshminarayana, A. Raghunathan, and N. Jha, "Power Analysis of Embedded Operating Systems", in proceedings of the 37th Design Automation Conference, Los Angeles, CA, pp. 312-315, June 2000.

[33] P. Kohout, B. Ganesh, and B. Jacob, "Hardware support for real-time operating systems", in Proceeding of First IEEE/ACM/IFIP International Conference on Hardware/Software Co-design and System Synthesis (CODES+ISSS 2003), Newport Beach CA, pp. 45-51, Oct. 2003.

[34] M. Vetromille, L. Ost, C. Marcon, C. Reif, and F. Hessel, "RTOS scheduler implementation in hardware and software for real time applications", in 17th IEEE International Workshop on Rapid System Prototyping, pp. 163-168, Jun. 2006.

[35] S. Chandra, F. Regazzoni, and M. Lajolo, "Hardware/software partitioning of operating systems: a behavioral synthesis approach", in GLSVLSI '06 Proceedings of the 16th ACM Great Lakes symposium on VLSI, (NY, USA), pp. 324-329, ACM, 2006.

[36] Z. Murtaza, S. Khan, A. Rafique, K. Bajwa, and U. Zaman, "Silicon real time operating system for embedded DSPs", in ICET' 06: Proceedings of International Conference on Emerging Technologies, (Peshwar), IEEE, pp. 188-191, Nov. 2006.

[37] M. Song, S. H. Hong, and Y. Chung, "Reducing the overhead of real-time operating system through reconfigurable hardware", in proceedings of 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools, pp. 311-316, Aug. 2007.

[38] Sebastien Pillement, Olivier Sentieys and Raphael David "DART: A Functional-Level Reconfigurable Architecture for High Energy Efficiency", EURASIP Journal on Embedded Systems, Volume 2008, Article ID 562326, Hindawi Publishing Corporation, 2008.

[39] A. S. R. Oliveira, L. Almeida, and A. B. Ferrari, "The ARPA-MT embedded SMT processor and its RTOS hardware accelerator", Industrial Electronics, IEEE Transactions on, Volume 58, No. 3, pp. 890-904, March 2011

[40] L. Almeida, A. S. R. Oliveira and A. B. Ferrari, "A specialized and predictable processor for real-time systems", in Workshop on Application Specific Processors, pp. 32-38, Nov. 2009.

[41] L. Almeida, N. Silva, A. Oliveira and R. Santos, "The OReK real-time micro kernel for FPGA-based systems-on-chip", in proceedings of 6th Workshop on Embedded Systems for Real-time Multimedia, (ESTImedia 2008), IEEE Xplore, Atlanta Georgia, pp. 75-80, Oct. 2008.

[42] Xiangrong Zhou, Peter Petrov "Rapid and low-cost context-switch through embedded processor customization for real-time and control applications" DAC San Francisco, CA, pp. 352-357, July 2006.

[43] N. Maruyama, T. Ishihara, and H. Yasuura, "An RTOS in hardware for energy efficient software-based TCP/IP processing", in IEEE Symposium on Application Specific Processors, pp. 58-63, June 2010.

[44] H.K. Hay So, X. Changqing, W. Mei, W. Nan and Z. Chunyuan, "Extending BORPH for shared memory reconfigurable computers Field Programmable Logic and Applications (FPL)" in 22nd International Conference on IEEE Improving Usability of FPGA-Based Reconfigurable Computers Through Operating System Support, Oslo. pp. 563-566, Aug. 2012.

[45] H.K. Hay So and R. W. Broderson, "BORPH: An Operating System for FPGA-Based Reconfigurable Computers" DAC University of California, Berkeley, Technical Report No. UCB/EECS, pp. 92-96, July 2007.

[46] H.K.Hay So, and R. W. Brodersen, "A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH", ACM Transactions on Embedded Computing Systems (TECS) TECS Homepage archive Volume 7, Issue 2, Article No. 14 ACM New York, USA, February 2008.

[47] Ikbel Belaid, Fabrice Muller and Maher Benjemaa "Static Scheduling of Periodic Hardware Tasks with Precedence and Deadline Constraints on Reconfigurable Hardware Devices", International Journal of Reconfigurable Computing, Volume 2011, Article ID 591983, Hindawi Publishing Corporation, 2011

[48] A. B. Lange, K.H. Andersen, U.P. Schultz and A. S. Sørensen, "HartOS - A hardware implemented RTOS for hard real-time applications", in Proceedings of the 11th IFAC/IEEE International Conference on Programmable Devices and Embedded Systems, Brno, Czech Republic, 2012.

[49] A. B. Lange, "Hardware RTOS for FPGA based embedded systems", Master's thesis, University of Southern Denmark. http://www.hartos.dk/publications/thesis/hartos.pdf accessed on Nov.2015.

[50] D. G. Harkut & M.S.Ali, "Hardware Support for Adaptive Task Scheduler in RTOS", Intelligent Systems Technologies & Applications, Volume 384, Springer, UK, pp. 227-245, 2015.

[51] M.M.M. Fahmy, "A fuzzy algorithm for scheduling non-periodic jobs on soft real-time single processor system", Ain Shams Engineering Journal, Elsevier B.V., doi:10.1016/j.asej.2010.09.004, pp 31-38, 2010

[52] M. Sabeghi, M. Naghibzadeh and T. Taghavi, "Scheduling Non-Preemptive Periodic Task in Soft Real-time Systems using fuzzy Inference", 9th IEEE International Symposium on Object and component-oriented Real-Time distributed Computing(ISORC), April 2006.

[53] H. Mahdi, M. F. Sied and L. Caro, "Soft real-time fuzzy task scheduling for multiprocessor systems", International journal of intelligent technology Vol. 2 No. 4, pp. 211-216, 2007. 98 E. H. Mamdami and S. Assilian , "An experiment in linguistic synthesis with a fuzzy logic controller ", in International Journal of Man-Machine Studies, Vol.7,No.1, pp. 1-13, 1975.

[54] E. H. Mamdami and S. Assilian , "An experiment in linguistic synthesis with a fuzzy logic controller ", in International Journal of Man-Machine Studies, Vol.7,No.1, pp. 1-13, 1975.

[55] M. Sugeno, "Industrial applications of fuzzy control", Elsevier Science Inc., New York, NY, 1985.

[56] Elragal, Hassan M. "Takagi-Sugeno Fuzzy System Accuracy Improvement with A Two Stage Tuning." Int. J. Com. Dig. Sys 4.4 (2015).

[57] Sharma, Mridula, Haytham Elmiligi, and Fayez Gebali. "Performance Evaluation of Real-Time Systems." Int. J. Com. Dig. Sys 4.1 (2015).

[58] J.S.R. Jang, "ANFIS: Adaptive-Network-based Fuzzy Inference Systems", IEEE Transactions on Systems, Man, and Cybernetics, Vol. 23, No. 3, pp. 665-685,1993.

[59] L. Lindh, J. Stärner and J. Furunäs, "From Single to Multiprocessor Real-Time Kernels in Hardware", in IEEE Real Time Technology and Applications Symposium. Chicago, May 1995.

[60] L. Lindh, "Utilization of Hardware Parallelism in Realizing Real Time Kernels", Doctoral Thesis, TRITA – TDE 1994:1, ISSN 0280-4506, ISRN KTH/TDE/FR-94/1-SE, Department of Electronics, Royal Institute of technology, Stockholm, Sweden, 1994, accessed on Nov.2015.

[61] Lindh, L. "Utilization of Hardware Parallelism in Realizing Real Time Kernels", Doctoral Thesis, TRITA – TDE 1994:1, ISSN 0280-4506, ISRN KTH/TDE/FR-94/1-SE, Department of Electronics, Royal Institute of technology, Stockholm, Sweden, 1994, accessed on Aug.2015.

[62] Karloff, Anthony C., and Esam Abdel-Raheem. "Performance Analysis of a Flexible, Optimized and Fully Configurable FPGA Architecture for Two-Channel Filter Banks." Int. J. Com. Dig. Sys 2.2 (2013): 53-62.

[63] A. Rehman and M. Shakir, "Comparative Analysis of Scheduling Algorithms in IEEE 802.16 WiMAX", International Journal of Computing and Digital Systems. Vol. 3, No. 2, pp 161-172, 2014.

Dinesh G Harkut received B.E.(Computer Science & Engineering) & M.E. (Computer Science & Engineering) from SGB Amravati University in 1991 and 1998 respectively. He completed his masters in Business Management and obtained his Ph.D. from SGB Amravati University in Business Management in 2013 while serving as a full-time faculty in the Dept. of Computer Science & Engineering at Prof Ram Meghe College of Engineering & Management, Badnera – Amravati. His research interests are Embedded Systems and RTOS.

M. S. Ali is a Professor and Principal of Prof Ram Meghe College of Engineering & Management, Badnera – Amravati. He obtained his B.E.(Electronics & Power) and M.Tech.(Power Electronics) from Nagpur University and I.I.T. Powai, Mumbai in 1981 & 1984 respectively He obtained his Ph.D. from SGB Amravati University in 2006. He has been on the SGB University's various body like Board of Studies, Faculty of Engineering & Technology and Academic Council since last fifteen years. He is Hon'ble Chancellors nominee on the senate of RTM Nagpur University. His research interests are Operating Systems, Artificial Intelligence and Java Technologies.