ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 3, May 2013

# Task Allocation and Memory Partitioning for MPSoC in Embedded System

Anuradha B[1], Hemalatha M[2], Vivekanandan C[3]
Associate Professor[1], PG Scholar[2], Profeesor[3]
SNS College of Engineering, Coimbatore, Tamilnadu, India

***Abstract*— Today complex embedded system uses multiprocessor system-on-chip (MPSoC). Multiprocessor system on-chip is an integrated circuit containing multiple instruction set processors on a single chip that implements most of the functionality of a complex embedded system. MPSOC provides the portability to meet the performance requirements of multimedia applications. Scratch pad memory has been replaced by the cache nowadays due to its inherent advantages in terms of area, energy and timing predictability. The two major issues in the embedded systems are the task scheduling and the partitioning of available Scratch pad memory (SPM) memory. These steps are decoupled in the traditional design space exploration process. In this paper an integrated approach is going to follow along with the dynamic updation of the data in the SPM to improve the performance by reducing the execution time of the embedded applications.***

*Keywords*— **Embedded System, MPSOC, Scratch Pad Memory, Task Scheduling**.

## I. INTRODUCTION

Miniaturization of electronic components has led to the introduction of complex electronic systems which are integrated onto a single chip, so-called systems-on-a-chip (SoCs). At the same time, performance requirements have been increasing. The resulting performance requirements can no longer be met by single processor systems. If achievable at all, this performance would come with enormous power consumption and serious cooling problems, due to the necessary very high clock speed. Currently, the only option for achieving the necessary performance is to use several processors with moderate clock speeds. Therefore, many of the currently available SoCs contain several processors, making them multiprocessor systems on a chip (MPSoCs). Embedded systems are designed to do some specific task, rather than be a general-purpose computer for multiple tasks. Some also have real time performance constraints that must be met, for reasons such as safety and usability; others may have low or no performance requirements, allowing the system hardware to be simplified to reduce cost. These systems are not always separate devices. Many embedded systems consist of small, computerized parts within a larger device that serves a more general purpose. Embeded systems become increasingly difficult; the increase in memory access speed has failed to keep up with the increase in processor speed. This makes the memory access latency a major issue in scheduling embedded applications on embedded systems. A MPSoC is an attractive solution to the increasing complexity and size of embedded applications. Multiprocessor systems-on-chips (MPSoCs) are, first of all, systems-on-chips. They implement complete applications on a single chip. MPSoCs are systems-on-chips that include one or more programmable processors. Systems-on-chips are generally adapted to the application to meet performance, power, and cost goals. Although modern VLSI fabrication technology provides us with very large chips, applications keep getting larger. Multiprocessor systems on chips try to balance specialization and programmability. Programmable processors allow the SoC to be programmed after fabrication; MPSoCs are often referred to as platforms because they allow for many implementations of a given type of system. Programmability offers many advantages: the same chip can be used in several products, reducing product cost; design tasks can be compartmentalized; and the platform chip may have a longer shelf life than a highly specialized SoC. Execution time predictability is a critical issue for real-time embedded applications; this means that data caches are not suitable since it is hard to model the exact behaviour and to predict the execution time of programs. To alleviate such problems, many modern MPSoC systems use software-controlled memories known as scratchpad memories, which allow execution times to be predicted with accuracy. Many studies have recently proposed Scratch Pad Memory (SPM) which is software controlled memory. SPM consists of only data array and memory decoder logic where as cache consists of data array, memory decoder logic along with tag memory which makes it both larger in size and power consumption than SPM, hence SPM is efficient than cache. SPM require an explicit support such as strong programming or some compiler directed tool for their proper utilization, a strong programming logic can make efficient use of this memory. SPM consume 40% less energy and 34% less space than cache. Execution time can be accurately predicted in SPM. Programmer can determine what part of code or data should be placed into the

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 3, May 2013

SPM, it uses special high speed memory circuit to hold small item of data for rapid retrieval. SPM simplifies caching logic and helps system work without main memory contention especially in cache of MPSOC. Many multiprocessor system-on-chip models use a memory hierarchy with slow off-chip memory and fast on-chip scratchpad memories. Such hierarchy means that proper allocation of variables to the on-chip memory is an essential part in reducing the off-chip accesses [1]. The computation time of a program on a processor depends on how much SPM is allocated to that processor. There are a large number of complex embedded applications consisting of multiple concurrent real-time tasks [2]. The tasks can be scheduled on different processors. The computation time of each task depends on the amount of SPM allocated to the processor executing this task. The problem of task scheduling and memory allocation on MPSoCs is an NPcomplete problem [3]. Traditionally, these two steps are performed separately where tasks are usually scheduled first and the SPM budget is then partitioned among the processors. Such a decoupled technique may prevent better schedules in terms of minimizing the computation time of the whole application. The integration of those two steps is critical to improve the performance. Task mapping/scheduling and SPM configuration are dependent on each other. The remainder of this paper is organized as follows. Section 2 consists of Problem formulation. Section 3 presents the motivation. Section 4 explains the related work. Section 5 presents the Task Scheduling and Memory Partitioning. Section 6 explains the Pipeline Scheduling.

## II.PROBLEM FORMULATION

***Architectural Model.*** An embedded single-chip multiprocessor architecture is shown in Figure 1. The architecture contains multiple processor cores on chip. The cores can be homogeneous or heterogeneous. The processor cores communicate with the shared off-chip memory via a bus. The architecture uses scratchpad memory (SPM), which is fast SRAM managed by software (application and/or compiler). In the single-chip multiprocessor setting, each processor core can access its private SPM as well as SPMs of other processors (remote SPMs). Such a setup is called virtually shared scratchpad memory or VS-SPM. A processor core has dedicated access to its private SPM with minimum latency usually a single cycle. Access to a remote SPM also incurs low latency (e.g., 4–16 cycles) due to the fast on-chip communication link among the processor cores. However, off-chip memory access has very high latency (e.g., 100 cycles) due to the performance gap between processor and DRAM technology. Access conflicts between multiple processors may also arise in the bus, adding non-trivial variable delays to the off-chip memory accesses. For simplicity, assume that the latency incurred by every off-chip memory access is a constant. To avoid coherency issues, a memory location can be mapped to at most one SPM. The Cell processor is an example of a real system with similar architecture even though its recommended programming model is somewhat different.

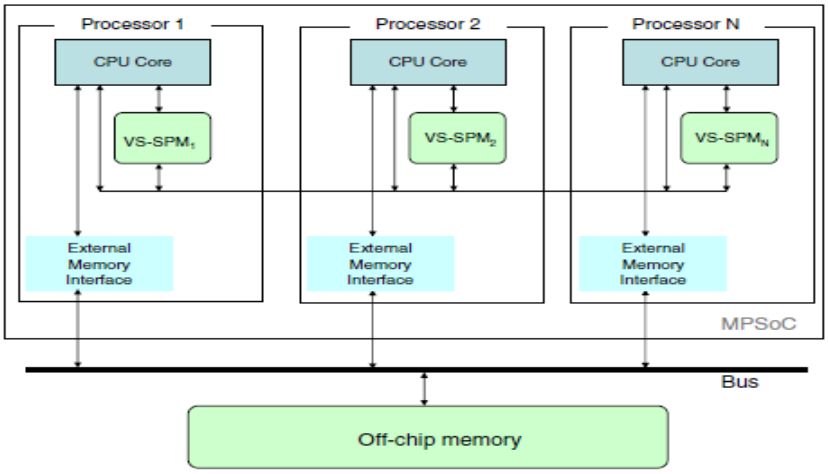


**Fig 1: Embedded Multiprocessor With Scratchpad Memory**

***Task Graph.*** By assuming that the embedded application is specified as a task graph. The task graph is a directed acyclic graph that represents the key computation blocks (tasks) of an application as nodes and the communication between these tasks as edges. A task can be mapped to any of the processing cores. Therefore, associated with each task T are the execution times corresponding to running the task T on each of the processing cores. In case of homogeneous cores, there is only one execution time associated with each task. The execution time of a task T on

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 3, May 2013

a processor P depends on the placement of its data variables in the SPM. Therefore, The execution time is estimated assuming that all the data variables are accessed from the off-chip memory. An edge from task T to T' in the task graph represents data transfer between these tasks. Therefore, associated with each edge is the amount of data transferred along that edge.

***Pipelined Scheduling.*** Most streaming applications, such as multimedia and digital signal processing (DSP) applications, are repetitive in nature. For these applications, the execution of the task graph is evoked repeatedly for a stream of input data. Hence, these applications are amenable to pipelined implementation for greater throughput. The pipelined implementation benefits from allowing multiple processors execute multiple iterations of the task graph at the same time. Even though pipelined scheduling is more challenging, so consider it in the problem formulation. Note that the objective for sequential implementation is to minimize the execution time of a single iteration of the task graph. In contrast, the objective of pipelined implementation is to minimize the initiation interval (II), which is the time difference between the start of two consecutive iterations of the task graph and the throughput for a streaming application.

***Problem Statement.*** Given a task graph, the architectural model and a bound on the total available on-chip SRAM. The objective is to minimize the execution time in cycles of the embedded application on the MPSoC architectural model. This problem can be decomposed into three sub-problems.

- Mapping/Scheduling of the tasks to the processors as well as communication among the tasks.
- SPM Partitioning: Finding the optimal size for each SPM.
- Data allocation: Allocating data variables corresponding to the tasks to the SPMs.

A processor cannot start executing task T until all the necessary data communication is performed. The weight of edge is the communication cost. Each task can be mapped to any of the available processors. Since the processors in the architectural model can be heterogeneous, the execution time of each task depends on the processor to which this task is mapped as well as the SPM memory allocated to that processor. Generally speaking, a larger SPM results in less computation time since off-chip access is more expensive in terms of the clock cycles compared to fast on-chip SPM. A large portion of the execution cycles of a task goes to accessing the data variables. Accessing a data variable from a SPM is usually in the order of 100 times faster than accessing it from the off-chip memory. Since the available SPM memory is usually limited due to the MPSoC's design constraints, a good utilization of SPM can be critical in narrowing the gap with the processor's speed.

## III.MOTIVATING EXAMPLE

Most works so far have treated task scheduling and memory partitioning as two decoupled steps that are performed by first scheduling the tasks onto processors and then partitioning the available memory among processors. Unfortunately, task scheduling and memory partitioning are interdependent and they should be integrated in one step in order to get high quality schedules. Unlike current approaches that treat task scheduling and memory partitioning as two separate problems, by solving these two problems in an integrated fashion. This paper deals with developing an effective heuristic for the task scheduling memory partitioning problem for a multiprocessor system on chip where a single application is using the MPSoC at a time. These two steps are performed in an integrated fashion where the private on-chip memory budget allocated to a processor is decided as tasks are mapped to this processor. The computation time of a task depends on the processor to which it is mapped, as well as on the SPM memory available for that task. Therefore, task scheduling should take into consideration the varying computation time of a task based on the processor and the SPM budget. Existing design-space exploration strategies will first map and schedule the tasks and communication onto the two processors without considering the allocation of variables to the SPM. However, fixing the task schedule a-priori without considering the effect of data allocation on the execution time may miss the global optima. For example, task *T*2 has the farest execution time without data allocation and hence it is mapped onto a separate processor. However, its execution time may reduce considerably after data allocation and hence it may not be optimal to allocate this task on a separate processor. Exploring the design space of task scheduling, SPM partitioning and data allocation together could potentially reach a design point that is not possible through decoupled scheduling and SPM allocation.

## IV.RELATED WORK

In this section, a brief review of previous research in the area of scheduling task graphs on multiprocessors and data location for scratchpad memory (SPM). The problem of scheduling a task graph on multiple homogenous processors in order to minimize execution time (or energy) has been studied extensively. In its general form, this problem is NP-complete and hence a number of heuristics have been proposed for a comprehensive comparison of

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 3, May 2013

these heuristics. These works mostly consider non-pipelined scheduling. Benini et al. propose a hybrid constraint programming and integer programming based approach for finding the optimal pipelined schedule. The related problem of mapping and scheduling tasks to a set of heterogeneous processing elements has been studied in the context of hardware/software co-design [4]. Technically, this partitioning and scheduling problem for co-design is quite similar to the multiprocessor scheduling problem. Niemann and Marwedel [5] propose an ILP-based solution for the co-design problem. Recently, various research groups have proposed pipelined scheduling solutions for this problem as well, especially in the context of streaming applications. Chatha and Vemuri [6] propose a branch and bound solution whereas Kuang et al. [7] propose an ILP-based solution. However, the objective of both these works is to minimize the total component cost and hence the number of pipeline stages.

Panda et al. have developed a comprehensive allocation strategy for scratchpad memory [8] to improve the program performance. Avissar et al. [9] propose a 0-1 ILP solution to optimally allocate global and stack variables. Their more recent work extends this approach to heap memory. Sjodin et al. [10] also propose a 0-1 ILP solution for scratchpad allocation. The main goal is to reduce the code size through allocation. Similarly, Steinke et al. [11] formulate an ILP-based allocation strategy to reduce the overall energy consumption. Angiolini et al. [12] propose an alternative approach where the SPM is optimally partitioned into multiple banks in order to improve energy efficiency through a dynamic programming solution. An excellent reference to memory system design in the context of chip multiprocessors is [13]. [14] Proposes an optimal memory allocation technique based on ILP for application-specific SoCs.The closest work is the flexible SPM design for MPSoCs investigated in [15]. However, they focus on data parallelism as opposed to task parallelism in the context of homogeneous multiprocessors.

## V. TASK SCHEDULING AND MEMORY PARTITIONING

A good heuristic for task scheduling and memory partitioning should take into consideration the varying execution time of a task throughout the process of building the schedule. This varying execution time is the result of the SPM budget assignment to processors that is dynamically calculated during the execution of the algorithm. Using profiling of the tasks in the embedded application, Min, Avg, and Max values are calculated for each task on each of the available heterogeneous processors. The elasticity of a task is the extent to which this task can benefit from a larger SPM. Although it can be defined in different ways, the elasticity is defined as the extent to which the computation cost of a task on Pi may decrease as the SPM budget of Pi is increased from the current budget to size where size is the maximum amount of SPM budget available in the model. Equation (1) defines elasticity of task Ti where Cur is the computation time of the task under the current memory budget. The elasticity of a task Ti is a measure of the room for computation time reduction of Ti with more SPM budget

elasticity (Ti) = (Curi – Mini) / Mini ----(1)

A bigger value of elasticity means that the computation time of Ti is more amenable for reduction with the increase in the

SPM allocated to that task. Note that the value of elasticity (Ti) changes through the course of the heuristic since the current computation time of Ti, Curi, may change as the SPM budget distribution changes. Although elasticity can be easily defined in some other forms, elasticity is defined as a number between 0 and 1 for uniformity. A definition of (Curi −Mini) for elasticity moved the heuristic one step closer to the greedy heuristic. The value between 0 and 1 for elasticity is not the main guide for memory allocation for a certain processor. The heuristic starts with profiling the application to extract important information. Using the profiling data, the embedded application will be divided into tasks with a necessary data communication between two tasks imposing a certain kind of dependence. Based on the extracted tasks and the communication between them, the task dependence graph is created. In this graph, each task is represented by a vertex and each communication cost by a weighted directed edge. For each available task *Ti* and processor *Pj* , calculate the number of variables, the size of the variables, and *Minij* , *Avgij* , and *Maxij* values. All these values are computed through profiling. Then, the *ASAP* values for all tasks are calculated based on the *Avg* values that are assuming the SPM budget is equally divided among the available processors. Tasks will be sorted in an increasing order of the *ASAP* values in a list *L*1. For each task, following the *ASAP* sort, evaluate the best processor to assign this task so that the overall computation time is minimally increased. The minimum start time of a task Ti on processor Pj , Start−time(Ti, Pj ), is equal to the maximum of the end time of processor Pj, End−time(Pj ), and the maximum end time of all communication time. Two dependent tasks mapped to the same its parent tasks, MaxTjO∈ Parent(Ti)(Tj ), plus the corresponding processor will have zero communication cost. In general, task Ti will be scheduled on the processor Pj corresponding to the minimum additional overhead time in the schedule. The PEC(Pk) value is a guide to the scheduler of how much this overhead time may decrease with additional SPM memory transfers in future steps if Ti

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 3, May 2013

is mapped to Pk. PEC is an estimate of how much the end time of processor Pk will be if more SPM budget is assigned to it. The PEC of a processor is closely related to the elasticity of the tasks scheduled on that processor. The PEC value provides the flexible essence of the heuristic as at each step the heuristic looks beyond the current SPM budgets distribution in its task mapping decision to an estimate of future distributions in future steps. By assigning task Ti to either of two different processors will have the same increase value in the overall schedule time, by scheduling Ti on the processor with the higher elasticity under the current SPM budget. The elasticity of a processor is the average value of the elasticity of the tasks scheduled on this processor.

## VI. PIPELINE SCHEDULING

An embedded application is usually executed many times for a stream of input data on an MPSoC. Such multiple executions make embedded applications amenable to pipelined implementation. Pipeline scheduling benefits from allowing tasks from different embedded application instances to be scheduled at each stage of the pipeline. But this schedule does not decrease the computation time of one instance of embedded application, but rather it decreases the time between the start times of two consecutive iterations of the task graph. The main aim is to decrease the pipeline stage time interval, after filling up the pipeline an instance execution of the application is performed each pipeline stage. The maximum number of stages is equal to the number of processors in the MPSoC system.

## VII.EXPERIMENTAL RESULTS

Three approaches have been implemented to solve the task scheduling and memory allocation problem on MPSoC systems, such as:

1) Decoupled task scheduling and memory partitioning assuming equally partitioned SPM among all available processors TSMP−EQUAL;

2) Decoupled task scheduling and memory partitioning with SPM partitioned among different processors with different ratio, TSMP−ANY;

3) Integrated task scheduling and memory partitioning heuristic described in Section V, TSMP−INTEG.

By using the following real-life programs from the Media bench and MiBench as test benchmarks. The key computation blocks of each application are identified through profiling. Each computation block will be used as a task (vertex) in the TDG. Dependences between different blocks will be represented as edges between tasks in the TDG. Finding the exact values for communication cost is difficult. The application is divided into tasks or basic blocks. Then by using the control/data flow information to find the dependences among tasks and to come up with an estimate of the communication cost. By using the Simple scalar architectural simulation to profile the used benchmarks. Simple scalar can simulate the execution of an application on complex multiprocessor system on chip architectures with different memory hierarchies. Through the simple scalar profiler, extract the profiling data such as variable sizes and access frequencies and execution time of each task. The MPSoC architecture used is similar to the one in Fig. 1. The profiling is intended to: 1) divide each application into computation blocks referred to as tasks; 2) find the computation times (Min, Avg, Max) for each task on each available processor in processor cycles; 3) find the number of variables; 4) the number of times each variable is used, freq; and 5) the size in bytes for each variable in the current application.

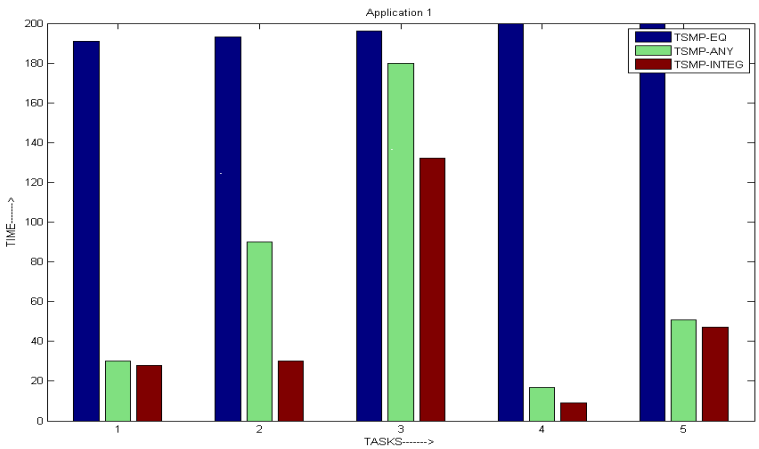


**Fig 2: Result for benchmark**

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 3, May 2013

Figure 2 show the results achieved by the heuristic when considering a system with four processors and an SPM budget The results in the figure are the normalized execution cycles with respect to TSMP−EQ, TSMP−ANY, and TSMP−INTEG.

## VII.CONCLUSION

In this work, an effective heuristic that integrates task scheduling and memory partitioning of embedded applications on multiprocessor systems-on-chip with scratchpad memory is implemented to further reduce the execution time. Compared to the widely-used decoupled approach, this integrated approach significantly will improve the results since the appropriate partitioning of SPM spaces among different processors depends on the tasks scheduled on each of those processors and vice versa.

## REFERENCES

[1] P. Panda, N. Dutt, and A. Nicolau, Memory Issues in Embedded Systems on-Chip: Optimization and Exploration. Dordrecht, The Netherlands: Kluwer, 1999.

[2] Z. Ma, C. Wong, S. Himpe, E. Delfosse, F. Catthoor, J. Vounckx, and G. Deconinck, "Task concurrency analysis and exploration of visual texture decoder on a heterogeneous platform," in Proc. IEEE Workshop Signal Process. Syst., Aug. 2003, pp. 245–250.

[3] Y.-K. Kwok and I. Ahmad, "Benchmarking and comparison of the task graph scheduling algorithms," J. Parallel Distributed Computing, vol.59, no. 3, pp. 381–422, Dec. 1999.

[4] G. De Micheli, R. Ernst, and W. Wolf. Readings in Hardware/Software Co-Design. Morgan Kaufmann, 2002.

[5] R. Niemann and P. Marwedel. Hardware/software partitioning using integer programming. In Design, Automation and Test in Europe (DATE), 1996.

[6] K. S. Chatha and R. Vemuri. Hardware-software partitioning and pipelined scheduling of transformative applications.IEEE Transactions on VLSI, 10(3), 2002.

[7] S-R. Kuang, C-Y. Chen, and R-Z. Liao. Partitioning and pipelined scheduling of embedded system using integer linear programming. In International Conference on Parallel and Distributed Systems (ICPADS), 2005.

[8] P. R. Panda, N. D. Dutt, and A. Nicolau. Memory Issues in Embedded Systems-On-Chip: Optimizations and Exploration. Kluwer Academic Publishers, 1999.

[9] O. Avissar, R. Barua, and D. Stewart. An optimal memory allocation scheme for scratch-pad-based embedded systems. ACM Transactions on Embedded Computing Systems, 1(1), 2002.

[10] J. Sjodin and C. von Platen. Storage allocation for embedded processors. In International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES), 2001.

[11] S. Steinke, L. Wehmeyer, B-S. Lee and P. Marwedel. Assigning program and data objects to scratchpad for energy reduction. In Design, Automation and Test in Europe (DATE), 2002.

[12] F. Angiolini, L. Benini, and A. Caprara. Polymonial-time algorithm for on-chip scratchpad memory partitioning. In International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES), 2003.

[13] M. Kandemir and N. Dutt. Memory systems and compiler support for mpsoc architectures. In A. Jerraya and W. Wolf, editors, Multiprocessor Systems-on-Chips. Morgan Kaufmann, 2005.

[14] S. Meftali, F. Gharsalli, F. Rousseau, and A. A. Jerraya. An optimal memory allocation for application-specific multiprocessor system-on-chip. In International Symposium on Systems Synthesis (ISSS), 2001.

[15] M. Kandemir, J. Ramanujam, and A. Choudhury. Exploiting shared scratch pad memory space in embedded multiprocessor systems. In Design Automation Conference (DAC), 2002.

## AUTHOR BIOGRAPHY

**Hemalatha.M** received BE degree in computer science and Engineering from Bharathiyar University, Tamil Nadu, and India. She is currently pursuing her M.E in Computer Science and Engineering from SNS College of Engineering, Coimbatore. Her research interest includes Embedded System, Computer Architecture, and Operating System.

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 3, May 2013

**B.Anuradha** obtained her bachelor's degree in Computer Hardware and Software Engineering from Avinashilingam University and Masters Degree in Embedded systems from Anna University and currently pursuing her Ph.D from Anna University of Technology, Coimbatore. She has more than 10 years of teaching experience and currently, she is working as Associate Professor in Department of Computer Science and Engineering, in SNS College of Engineering, Coimbatore, Tamil Nadu. Her areas of interest include Embedded System, Operating Systems and Computer Architecture. She has published 11 papers in reputed international, national level conferences and International journals.

**Dr. C. Vivekanandan** obtained his bachelor's degree in Electrical and Electronics Engineering and Masters Degree in Applied Electronics, both from Bharathiar University in 1986 and 1988 respectively. He obtained his doctoral degree in Electrical Engineering from Anna University, Chennai for his research contributions in Sliding Mode Control. He is having more than two decades of academic service and presently working as Professor and Vice Principal, at SNS College of Engineering, Coimbatore, India. His areas of interest include Design of Digital Controllers, Microprocessor Based System Design and Algorithm Developing.