ISSN 1512-1232

REVIEW OF PERFORMANCE ANALYSIS METHODS FOR REAL-TIME EMBEDDED SYSTEMS DESIGN

BAKARE K. AYENI

Nigerian College of Aviation Technology, Zaria
Aeronautical Telecommunication Engineering School,
P.M.B. 1031, Zaria, Kaduna State, Nigeria.
bakarre@yahoo.com

JUNAIDU B. SAHALU

Department of Mathematics
Ahmadu Bello University, Zaria, Kaduna State, Nigeria.
abuyusra@gmail.com

Abstract

Embedded systems interact with their physical environment in which they are connected through sensors and actuators. Consequently they must execute at a pace determined by their environment.

Performance analysis of real-time embedded system plays an important role in the design process of complex embedded systems for analyzing essential performance characteristics of system design at an early phase. This gives the choice of important design decisions before much time and resources are invested in detailed implementation. The classification criterion for performance analysis methods is given by the set of analyzable performance metrics like timing aspects, memory requirement, resource utilization or power consumption.

This paper reviews and compares various performance analysis methods. It also identifies the limitations of these methods. There are many performance analysis methods; notable among them are simulation and formal analysis methods, real-time calculus, holistic scheduling analysis, compositional method, timed automata based performance analysis and stochastic analysis method.

KEYWORDS: *Embedded systems; sensors and actuators; performance metrics; best-case and worst-case latencies; end-to-end delays.*

1 Introduction

The embedded system designers face the problem of evaluating many candidate hardware-software architectures with respect to various performance indexes in the early design cycle. These indexes may include system's throughput, response times, end-to-end delays, resource utilization, memory requirements, etc. In most cases, building a prototype for each design alternative to directly measure these performance characteristics is infeasible because of high implementation costs and stringent time-to-market constraints. On the other hand, due to the increasing complexity of modern embedded systems, back-of-the-envelope estimations cannot be used without taking the risk of being totally incorrect. As a result, the only option left for the designers is to carry out the performance analysis based on some kind of a performance model of the system.

Modular Performance analysis plays an important role in the design process of complex embedded systems for analyzing essential performance characteristics of system design at an early phase. It gives the choice of important design decisions before much time and resources are invested in detailed implementation.

Embedded systems are typically reactive systems that are in continuous interaction with their physical environment to which they are connected through sensors and actuators. Consequently they must execute at a pace determined by their environment.

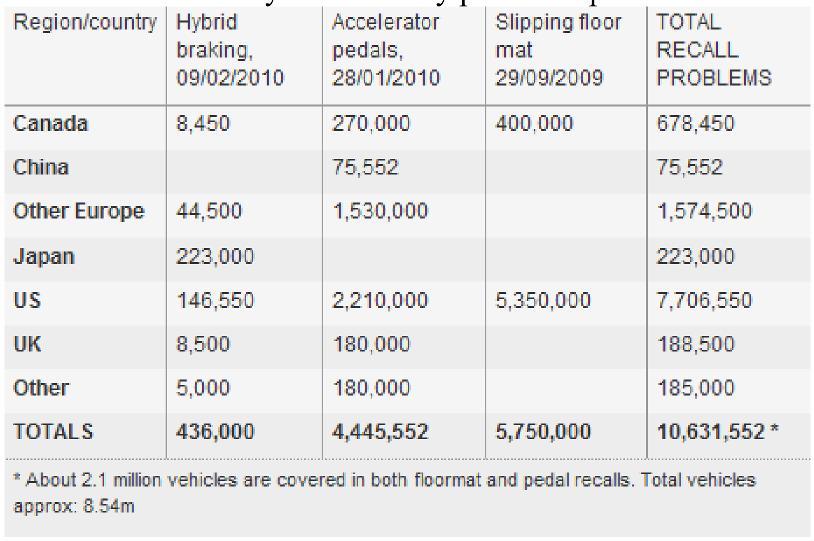
Generally, embedded systems must be highly efficient in terms of power consumption, size and cost, for example these are desirable in space exploration. In addition, they usually have to be highly dependable, as a malfunction or breakdown of the device they control is not acceptable.

For example, three separate but related recalls of automobiles by Toyota Motor Corporation occurred at the end of 2009 and start of 2010. Toyota initiated the recalls, the first two with the assistance of the U.S. National Highway Traffic Safety Administration (NHTSA), after several vehicles experienced unintended acceleration. The first recall, on November 2, 2009, was to correct a possible incursion of an incorrect or out-of-place front driver's side floor mat into the foot pedal which can cause pedal entrapment. The second recall, on January 21, 2010, was begun after some crashes were shown not to have been caused by floor mat incursion. This latter defect was identified as a possible mechanical sticking of the accelerator pedal causing unintended acceleration, referred to as Sticking Accelerator Pedal by Toyota. The original action was initiated by Toyota in their Defect Information Report, dated October 5, 2009, amended January 27, 2010 [35]. Following the floor mat and accelerator pedal recalls, Toyota also issued a separate recall for hybrid anti-lock brake software in February 2010 [36].

As of January 28, 2010, Toyota had announced recalls of approximately 5.2 million vehicles for the pedal entrapment/floor mat problem, and an additional 2.3 million vehicles for the accelerator pedal problem. Approximately 1.7 million vehicles are subject to both. Certain related Lexus and Pontiac models were also affected. The next day, Toyota widened the recall to include 1.8 million vehicles in Europe and 75,000 in China. By then, the worldwide total number of cars recalled by Toyota stood at 9 million. Sales of multiple recalled models were suspended for several weeks as a result of the accelerator pedal recall, with the vehicles awaiting replacement parts. As of January 2010, 21 deaths were alleged due the pedal problem since 2000, but following the January 28 recall, additional NHTSA complaints brought the alleged total to 37 [39].

The recall came at a difficult time for Toyota, as it was struggling to emerge from the recession and had already suffered from a resultant decrease in sales and the low exchange rate from yen to US dollar. On the day that the recall was announced in the US, it was announced that 750 jobs will be cut at Toyota's British plant at Burnaston, near Derby. It was estimated that each Toyota dealership in the US could lose between $1.75 million to $2 million a month in revenue, a total loss of $2.47 billion across the country from the entire incident. Additionally, Toyota Motors as a whole announced that it could face losses totaling as much as $2 billion from lost output and sales worldwide. Between 25 January and 29 January 2010 Toyota shares fell in value by 15% [11]. The summary of Toyota recall and problem is shown in table 1.

Table 1: Toyota recalls by place and problem



Region/country	Hybrid braking, 09/02/2010	Accelerator pedals, 28/01/2010	Slipping floor mat 29/09/2009	TOTAL RECALL PROBLEMS
Canada	8,450	270,000	400,000	678,450
China		75,552		75,552
Other Europe	44,500	1,530,000		1,574,500
Japan	223,000			223,000
US	146,550	2,210,000	5,350,000	7,706,550
UK	8,500	180,000		188,500
Other	5,000	180,000		185,000
TOTALS	**436,000**	**4,445,552**	**5,750,000**	**10,631,552 ***

* About 2.1 million vehicles are covered in both floormat and pedal recalls. Total vehicles approx: 8.54m

Nowadays automotive industry generally employs what is called engine management systems with complex processors. The application embeds microprocessor that is fast enough to provide a real-time operation. This was firstly used in Formula 1 racing cars, but it didn't take long until regular cars used them as well. (www.vehicle-lab.net/ecu.html). This embedded system, Electronic Control Unit (ECUs) use a microprocessor which can process the inputs from the various sensors in real time (see fig.1). ECUs contain both the hardware and software (firmware). The hardware consists of electronic components on a printed circuit board (PCB), ceramic substrate or a thin laminate substrate. The main component on this circuit board is a microcontroller chip (CPU). The software is stored in the microcontroller or other chips on the PCB, typically use EPROMs or flash memory so the CPU can be re-programmed by uploading updated code or replacing chips. This is referred to as an (electronic) Engine Management System (EMS).

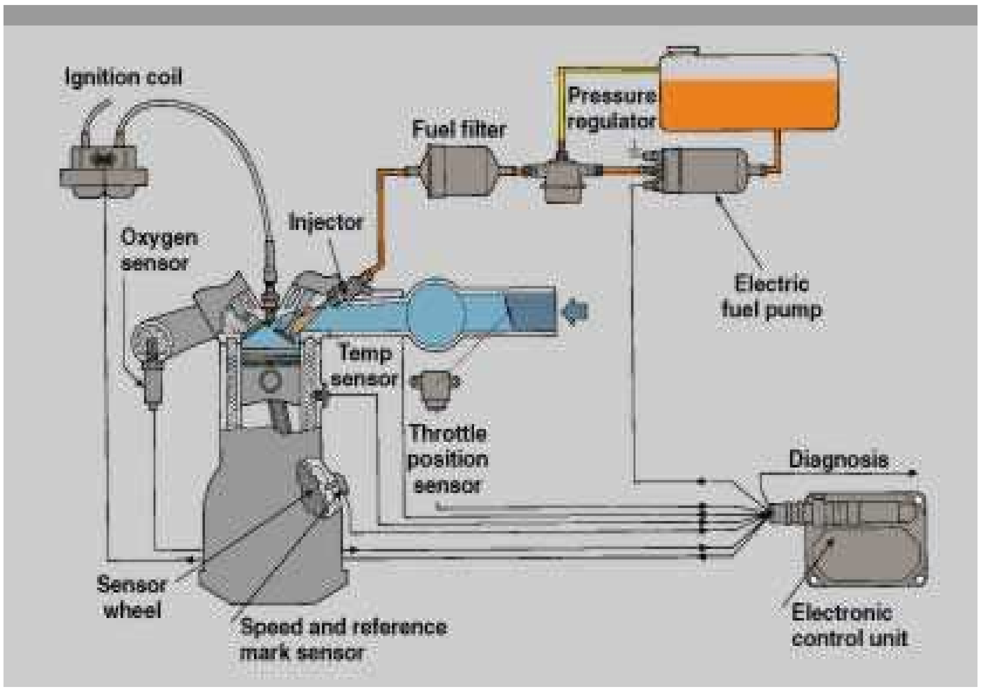


Fig. 1 : Engine Management System

Despite the use of EMS, product recalls happen all the time in the auto industry and elsewhere. They are almost always embarrassing because they reveal short-comings in research and development that have failed to be ironed out before the product has reached full-scale production [11].

Errors can occur at various stages of design process, errors can occur at analysis, conceptual design, programming, design test, system test and operation. But the error may not be detected until at the final system test or in an actual operation as in the case of Toyota cars recalls.

Without overestimating, we can fairly conclude that there is a strong need for an integration of performance analysis of system into design process at an early stage, for an economical point of view. This paper presents various performance analysis methods and compares these methods.

The rest of the paper is organised as follows: Performance analysis methods is discussed in section 2. Section 2.1 focuses on simulation and analysis method, while section 2.1 is on real-time calculus. Section 2.3 is on holistic scheduling analysis. Sections 2.4, 2.5, 2.6 discussed compositional approach, timed automata based performance analysis and stochastic analysis method. Comparison of performance analysis method and future work are discussed in sections 3 and 4 respectively. Section 5 concludes the work.

ISSN 1512-1232

2 Performance Analysis Methods

There are many approaches to performance analysis methods; notable among them are simulation and formal analysis methods, real-time calculus, holistic scheduling analysis and compositional method. There are also Timed Automata based performance analysis and stochastic analysis method. In the next subsections, we discussed these methods and previous work on them. We also identified limitation of each method.

2.1 Simulation and Formal Analysis Methods

Simulation is defined as the imitation of some real thing, state of affairs, or process. The act of simulating something generally entails representing certain key characteristics or behaviours of a selected physical or abstract system.

Simulation is used in many contexts, including the modelling of natural systems or human systems in order to gain insight into their functioning. Other contexts include simulation of technology for performance optimization, safety engineering, testing, training and education. Simulation can be used to show the eventual real effects of alternative conditions and courses of action.

Simulation provides a tool to estimate the performance of a system model. Based on the simulation, monitoring of system properties such as data throughput or resource usage can help to evaluate different designs [2].

The trend for simulation based performance analysis goes to full system simulation. Although simulation is accurate with completely specified models, it is not guaranteed to cover any corner case. Simulation based methods do not reveal worst-case bounds on essential system properties like end-to-end delay of events, throughput, and memory requirement. In addition, they often suffer from long running times which depend on the accuracy that is aimed for and from high set-up effort for each new architecture and mapping to be analyzed.

The use of SystemC, the Open SystemC Initiative (OSCI), is a widespread platform for system level modelling and simulation, is very common. Many simulation methods

are trace-based, that is, the system designer provides traces of input stimuli that drive the simulation of the modeled system.

SystemC is a system description language that can be used to model the behaviour level of systems [12]. It consists of a set of library routines and macros implemented in C++, which makes it possible to simulate hardware building blocks and concurrent processes written in standard C++. SystemC supports the communication of C++ objects in a simulated real-time environment. SystemC is both a description language and a simulation kernel. The code written will compile together with the library's simulation kernel to give an executable that behaves like the described model when it is run. It supports the simulation of a system at various levels of abstraction from cycle-accurate up to the behavioural level.

The main advantage of simulation is the large modelling scope. In contrast to formal analysis methods, basically every system can be modeled, as many dynamic and complex interactions can be taken into account for the simulation. In most of the cases the functioning and performance of a system can be verified with the same simulation environment, simulation traces and benchmarks.

Another advantage of simulation based methods is that they are usually reusable over different abstraction levels, as the simulation models can be refined. In other words, the level of abstraction for the simulation can be adapted to the required degree of accuracy.

However, hardware-software co-simulations are often computationally complex and have long running times. Therefore, performance estimation quickly becomes a bottleneck in the design process, especially if it is used to drive a design space exploration.

For example, [14] shows that in order to find simulation patterns that lead to corner cases is an exiting challenge for apparently trivial distributed embedded systems. Fig.2 depicts the system architecture. In application A1, a task P1 of the CPU reads periodically data bursts from the sensor and stores the data in the memory. A second task P2 reads the data from the memory, processes it and transfers it to an output device via the shared bus. The task P2 has a best case execution time

(BCET) and a worst case execution time (WCET). Suppose that the CPU implements static fixed priority scheduling and that P1 has higher priority than P2. In the second application A2, a task P4 running on the input interface periodically sends data packets to the Digital Signal Processor (DSP) over the shared bus. Task P5 on the DSP stores the data packets into the buffer. A second task P6 periodically removes data packets from the buffer, e.g. for playback. Suppose that the bus uses a first come first serve scheme for arbitration. As the two data streams of A1 and A2 interfere on the shared bus, there will be a jitter in the packet stream received by the DSP that may lead to an underflow or overflow of the buffer.

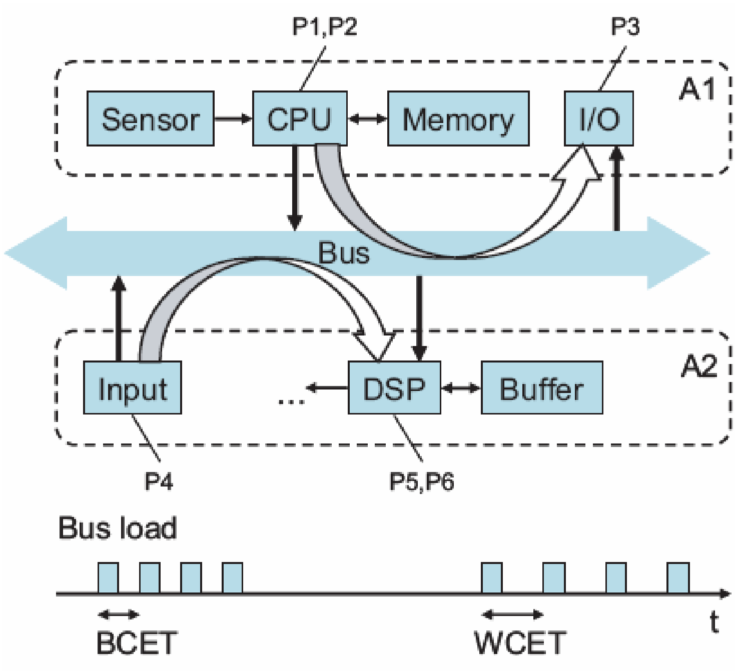


Fig. 2 : Interference of two data streams on a shared communication resource

The interesting property of this system is that the DSP experiences the worst case input jitter when P2 executes continuously with its BCET. The reason is that in this case the distance between the packets of A1 on the bus is shortest and thus the transient bus load is highest. In other words, the worst case execution of A2 coincides with the best case execution of A1.

The designer must perceive this system particularity in order to provide a simulation trace that reaches the corner case. In case of larger and more realistic systems, several computation and communication resources will be shared simultaneously, there may be different scheduling policies for the various resources and data/control dependencies will play a role.

It can be deduced that in Henia et. al., work, the corner cases will be extremely difficult to find, hence it cannot satisfied all [18] criteria for performance analysis. Therefore, simulation based methods are not suited to determining hard performance bounds of a general distributed embedded system. But nevertheless, simulative approaches can be useful to estimate the average system performance.

Formal methods for performance evaluation are emerging which enable the analysis of the whole systems using holistic and compositional approaches. In particular, the system can be analyzed using models of the individual components that can be later composed to capture the complete system. Formal analysis based methods lack the ability to incorporate complex interactions and state-dependent behavior. Analysis results are therefore often pessimistic, but this pessimism does not threaten their correctness.

In a related work of [1], Analytical performance models for Digital Signal Processor (DSP) systems and embedded processors, the computation, communication, and memory resources of a processor are all described using simple algebraic equations that do not take into account the dynamics of the applications such as variations in resource loads and shared resources. These methods are therefore lacking in accuracy and the analysis results typically show large deviations from the properties of the final system implementation.

Actually, there is no sharp division into simulation-based approaches and formal methods for system-level analysis [30]. In the area of embedded real-time systems design, one of the major differentiation criteria between these two classes of methods is the quality of results that are obtained with the respective methods.

To overcome the limitations of these two methods, [19] combined simulation and analysis method by a hybrid trace-based simulation methodology which though shorten simulation run-times, but the problem of insufficient corner case coverage still remains. To overcome this problem [3] proposed a method which combined SystemC simulation and formal analysis method based on Real-Time Calculus (RTC). Their work guarantee to deliver worst-case (and best-case) results for various system properties and also exhibit fast analysis run-times. Since an implementation of an embedded real-time system must meet a number of performance requirements related for example to end-to-end delays, buffer requirements, or throughput. When these quantities is measured on the final system implementation, the major variations is normally observed over time, as for example end-to-end delays may vary largely due to different input data or interference between concurrent system activities. However, there typically exists a worst-case and a best-case result for every quantity, such that we know for example that every observed end-to-end delay is larger or equal the best-case delay d_{BC} and smaller or equal the worst-case delay d_{WC} [38].

2.2 Real-Time Calculus

Real-Time Calculus (RTC), a system-level performance analysis method for embedded systems is a tool to characterize workload and processing capabilities respectively [30]. R*eal-time calculus* represents the resources and their processing or communication capabilities in a compatible manner and therefore, allows for a modular hierarchical scheduling and arbitration for distributed embedded systems [25].

In addition, the Real-Time Calculus allows computing various performance indexes of the system, such as upper bounds on the delay and backlog experienced by the events while being processed in the system [38].

The Real-Time Calculus provides powerful abstractions of the event and resource streams and uses these abstractions to mathematically model the behavior of an elementary performance component. This basic model can then be used for a component-wise evaluation of a whole scheduling network.

Fig.3 depicts Real Time Calculus abstract processing component that models the processing of an event stream by an application process. In particular, an incoming event stream represented as a pair of arrival curves α^l and α^u, flows into a FIFO buffer in front of the processing component. The component is triggered by these events and will process them in a greedy manner while being restricted by the availability of resources, which are represented by a pair of service curves β^l and β^u . On its output, the component generates an outgoing stream of processed events, represented by a pair of arrival curves $\alpha^{l'}$ and α^u. Resources left over by the component are made available again on the resource output and are represented by a pair of service curves $\beta^{l'}$ and $\beta^{u'}$.

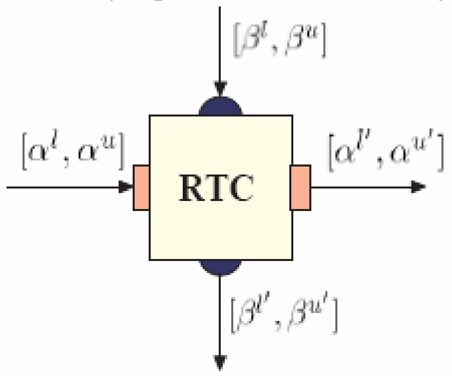


Fig. 3 : Real Time Calculus processing component

The processing components can be freely combined to form performance models of distributed embedded systems. For instance in order to model the sequential processing of an event stream by two tasks, it is sufficient to connect two processing components in series so that the outgoing event stream of the first one is the ingoing event stream of the second one.

Characterization of Event and Resource streams

Timing properties of event and resource streams are captured using arrival and service curves.

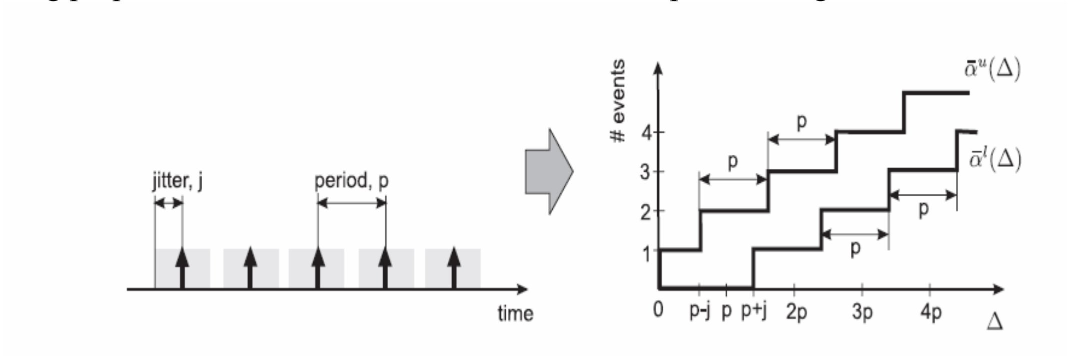


Fig. 4 : Modelling periodic event streams with jitter using arrival curves.

An event stream is abstracted by a pair of arrival curves, α^{-u} (Δ) and α^{-1} (Δ), which give respectively upper and lower bounds on the number of events seen in the event stream within any time interval of length Δ.

A resource stream is modelled by a pair of service curves, β^{-u} (Δ) and β^{-1} (Δ), which give respectively upper and lower bounds on the resource amount (e.g. number of processor cycles) offered within any time interval of length Δ.

The arrival and service curves can accurately describe streams with arbitrary complex timing behavior. On the other hand, a single pair of upper and lower curves can capture an entire class of streams with similar timing properties. For example, many standard event models (e.g. sporadic or periodic, periodic with jitter, periodic with bursts as discussed above) can be represented by the arrival curves [7]. Fig.4 illustrates this fact by showing how the arrival curves model a class of periodic event streams with jitter.

Evaluation of Event and Resource streams

The evaluation can be accomplished component wise, by propagating the event and resource streams through the network. Doing this requires a model describing how the timing properties of the event and resource streams get changed as a result of passing through the performance components.

The model assumes that the events belonging to the same stream are processed in their arrival order and that they are stored in a FIFO buffer while waiting to be served. Wandeler [38] introduced a compact representation for a special class of variability characterization curves, together with other methods to efficiently compute various Real-Time Calculus curve operations on these compact variability characterization curves, in order to efficiently conduct system level performance analysis and interface-based design within the Modular Performance Analysis (MPA) framework.

The Real-Time Calculus (RTC) Toolbox is now a toolbox for MATLAB that enables MPA framework based performance analysis and interface based design of embedded real-time systems within MATLAB.

In a related work, [31] proposed framework for analysis of system properties which has a concept of lower and upper arrival and service curves that capture the best- and worst-case behaviour of the workload. In addition, they enhanced the analytical framework of Naedele et. al., [26] with mechanisms to determine properties of the output event streams.

These developments pave the way to a modular approach to the performance analysis [32]. The limitation of Thiele et. al. is that it can model only tasks that consume and produce only one event per activation. Maxiaguine et. al., [24] addressed this limitation, for computing the output event streams it becomes necessary to convert the arrival curves expressed in terms of event-based units into equivalents expressed in resource-based units and backwards. Since this conversion is performed by scaling the curves with a constant factor corresponding to *worst-case execution demand* (WCED) for processing of one event, the execution time variability is not accounted for, resulting in overly pessimistic analytic bounds for workloads with large variations in execution demand of tasks, therefore execution demand curves is used to model more than one event per activation.

[33] relied on Real-Time Calculus to estimate various performance metrics, such as required buffer sizes and packet delays, resulting from implementing different scheduling policies on resource types (*processing elements* PEs) of a network processor. Their limitation is that their work could not account for the buffer constraints especially those related to the playout buffers and the variability of the task I/O rates.

Chakraborty et. al., [8] proposed a new task model for streaming applications combining the concept of arrival curves with the recurring real-time task model (RRT), which may help to reduce the size of task graphs of the RRT model while modelling complex event streams.

Wandeler employed RTC in conjunction with other method to analyze complex distributed embedded real-time systems with a modular and extensible framework for system level performance analysis but his work could not address the challenge of timing correlations in complex embedded systems, problem of cyclic dependencies and fixed-point calculation within performance models and the problem with dynamic systems with feedback and state-dependant behaviour.

The limitations of RTC is that it has high level of abstraction and time-interval domain.

This shows that in MPA of real-time embedded systems, one particular method is not enough, a hybrid approach is the best.

2.3 Holistic Scheduling Analysis

In the real-time systems domain many results are available on schedulability analysis and worst-case response time analysis of individual tasks on single processor systems with various scheduling policies. Examples are analysis methods for fixed-priority, rate-monotonic, deadline monotonic or earliest deadline first scheduling [21]. Several proposals have been made to extend the concepts of classical scheduling theory to distributed systems. Such extensions must in particular consider the delays caused by the use of possibly shared communication resources that can typically not be neglected. The analytic integration of processor and communication infrastructure scheduling is often referred to as holistic scheduling analysis. But rather than denoting a specific performance analysis method, holistic scheduling analysis comprises a collection of techniques for scheduling analysis of distributed embedded systems. In comparison to other performance analysis methods, the MPA framework models the service offered to an event stream explicitly, using the concept of resource streams. This approach has a number of advantages: First, it allows to model arbitrary complex resource availability patterns which may be experienced by individual event streams (or tasks) as a result of applying a certain scheduling or arbitration policy. Second, it supports the modularity of the performance analysis. Third, using the concept of resource streams it is easier to model hierarchical scheduling schemes and various resource reservation mechanisms. A variety of scheduling and arbitration policies can be modelled by a proper calculation (or definition) of the service curves within a scheduling network. For example, a fixed priority scheduling can be modelled by directly connecting the output resource stream (i.e. the remaining service) of a higher priority component to the resource input of the next (in terms of priority) component. For example, in the scheduling network in Fig. 5, tasks T3 and T5 are scheduled on the CPU resource using the fixed priority scheme. T3 has the highest priority. To model proportional share schemes and their derivatives, we need to introduce into the scheduling network the corresponding scheduling modules that distribute the resource streams according to specified shares, and after that collect the remaining service. Fig.5 depicts an example of such an arrangement for tasks T2 and T4.

The concept of scheduling network can be used to model the interactions between the requested and the offered service. In a scheduling network, the requested and the offered service are modelled by event and resource streams.

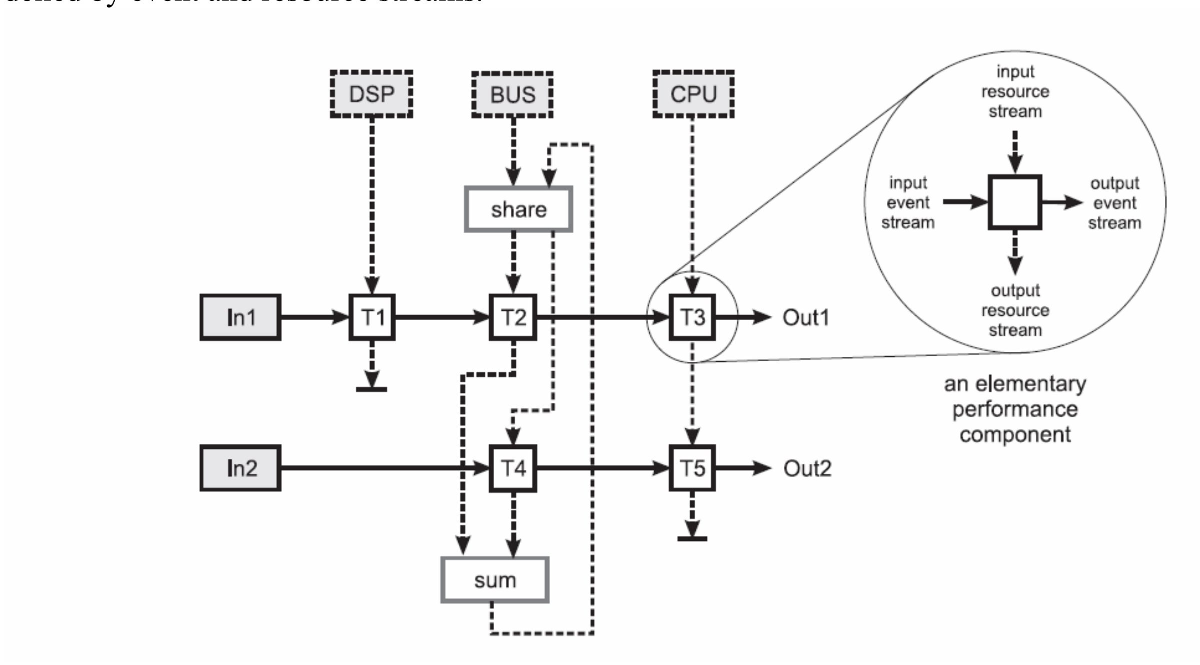


Fig. 5 : A scheduling network modelling the application-to-architecture mapping.

These streams flow through the network nodes, called performance components that model the interactions between the streams. Fig.5 shows an example of scheduling network corresponding to the application to architecture mapping. Solid and dashed arrows correspond to the event and resource streams, respectively.

An elementary performance component receives one event and one resource stream as its input (see Fig.5). The input event stream abstracts arrivals of a certain request type, while the input resource stream models availability of a given resource for processing of this request type. Abstractly seen, the input event stream triggers the performance component, which in response proceeds by consuming resources provided by the input resource stream. This represents execution of a task on a PE.

An elementary performance component typically also produces one event and one resource stream as its output. An event within the output event stream signifies a completed processing of a corresponding input event. The output resource stream represents the remaining service, that is, the service which has not been consumed by the performance component. This remaining service can then be used to process another event stream, i.e. it may serve as an input to another performance component. Likewise, the output event stream may represent requests for another resource, that is, it also may serve as an input to a different performance component. In this way, a scheduling network representing a performance model of the entire system (with a multitude of event streams and processing resources) can be constructed out of multiple independent performance components.

Besides the elementary performance components, a scheduling network may contain other types of nodes:

Resource modules model processing capabilities of PEs within the architecture. A resource module produces a stream corresponding to the unloaded resource that it models. In Fig.5, resource modules are marked with dashed boxes. They represent the bus, DSP and CPU resources.

Input modules inject into the scheduling network event streams generated by the system's environment. In Fig. 5, these are In1 and In2 modules.

Scheduling modules distribute resource streams between different performance components in accordance with a given resource management policy. A scheduling module receives and produces only resource streams (originated by the same resource). Using scheduling modules we can model different scheduling and arbitration policies deployed on the PEs of the architecture. In Fig.5, for example, we have share and sum scheduling modules.

Hierarchical modules are complex performance components containing subnetworks of other components.

Calculating upper bounds on delay and backlog

In the performance analysis, in addition to the structural performance view of the system provided by the scheduling network, there is a need also to characterize the behavior of the event and resource streams, and of the associated performance components. That is, we need to characterize timing properties of the streams and determine how these properties change when the streams pass through the performance components in the scheduling network as well as the backlog.

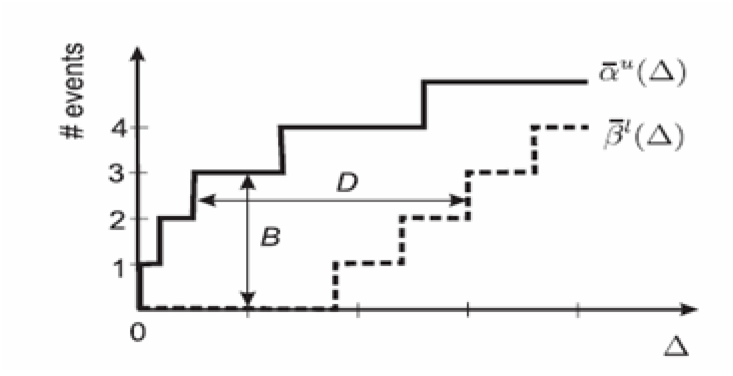


Fig. 6: Computing upper bounds on the delay, *D,* and the backlog, *B*.

Given an upper arrival curve α^u and a lower service curve β^1 at the input of an elementary performance component, (fig.6) we can compute upper bounds on the delay and on the backlog experienced by the event stream as a result of passing through this component [20].

The performance analysis of a distributed embedded system is done by combining the analysis of the single processing components of a performance model.

In a similar way, we can find upper bounds on the total delay and on the total backlog which an event stream may suffer as a result of passing through a chain of performance components. How this can be done is described in [20]. This allows estimating such performance indexes of an embedded system as the worst-case end-to-end delay and memory requirements. In the collection of holistic scheduling analysis techniques, every technique is tailored towards a particular combination of input event model, resource sharing policy and communication arbitration. While this permits detailed analysis of the temporal behaviour of a specific distributed system, it has the drawback that a new analysis method must be developed for every new input event model, communication protocol, resource sharing policy and combinations thereof. This circumstance not only restricts the applicability of holistic scheduling analysis, but the consequently large heterogeneous collection of different techniques also makes it difficult to use holistic scheduling analysis in practice [38].

In a related work of the area of classical real-time scheduling theory, [21] worked on scheduling but the result lead to poor processor utilization, and consequently to system designs with unreasonably high cost, or power consumption.

[23] proposed the multiframe task model that extends the classical periodic task model of Liu by permitting periodic tasks whose WCETs may vary from one instance to another. In the multiframe task model, the WCETs of consecutive task instances are determined following a fixed cyclic pattern. The model was further extended in [5] which allowed not only to determine the WCET of a task instance, but also the time separation between two task instances following a cyclic pattern.

[4] presents a recurring real-time task model (RRT) - a further generalization of the multiframe models. In the RRT model, a task is modelled by a set of subtasks arranged in a directed acyclic graph representing the conditional, non-deterministic behavior of the task. Each subtask is

characterized by its WCET, a relative deadline and a minimum triggering separation from its direct predecessors. The whole task graph is triggered sporadically with a specified minimum time separation between the triggering of the last subtask in the graph and the triggering of the next task instance. Another workload model, also using conditional directed acyclic graphs to model tasks, is reported by [27]. Instead of associating a deadline to each subtask in a task graph, the model associated a single deadline with the whole graph. Furthermore, it exposed the parallelism within a task for mapping on a multiprocessor architecture. In comparison to classical task models, the RRT model offers a great flexibility in modelling variability of the execution demand and irregular inter-arrival times. This flexibility is, however, limited to recurring patterns. The limitation of Baruah and Pop et. al. is that if workload bursts (characterized by periods with dense arrivals of tasks or increased execution demand or both) occur relatively seldom, then avoiding overly pessimistic results under the RRT model necessitates to consider very large task graphs, leading to inefficiency of the analysis. In other words, designers have to trade off the accuracy of the analysis for the analysis time, which for the RRT model increases exponentially with the problem size [6].

[37] addressed systems with fixed priority scheduling policy deployed on processor nodes communicating via a bus using a time division multiple access (TDMA) protocol. The methods can be very effective in modelling complex timing relations (e.g. phasing) between the tasks. However, they are often attributed to a lack of scalability and modularity [16].

2.4 Compositional Approach

In compositional approach, every single processor or communication link of a distributed system is analyzed locally. To interconnect the various components, the method relies on a set of standard event arrival patterns. Based on the arrival patterns of the incoming event streams and on the scheduling policy of the component, the appropriate classical analysis technique is chosen individually for every single processor or communication link to compute the worst-case and best-case response time of every event stream at the component as well as to compute the arrival patterns of the outgoing event streams that will trigger succeeding components. The local analysis results are then combined to obtain global end-to-end delays and buffer requirements.

The approach is however only feasible if the arrival patterns of the incoming event streams at a component fit the basic models for which results on computing bounds on the response times are available. While using compositional methods, three main problems may arise. Firstly, the architecture of such systems which is highly heterogeneous, the different architectural components are designed assuming different input event models and use different arbitration and resource sharing strategies. This makes any kind of compositional performance analysis difficult. Secondly, applications very often rely on a high degree of concurrency. Therefore, there are multiple control threads, which additionally complicate timing analysis. And thirdly, we can not expect that an embedded system only needs to process periodic events where to each event a fixed number of bytes is associated. If for example the event stream represents a sampled voice signal, then after several coding, processing and communication steps, the amount of data per event as well as the timing may have changed substantially. In addition, stream based systems often also have to process other event streams that are sporadic or bursty, e.g. they have to react to external events or deal with best-effort traffic for coding, transcription or encryption. There are only a few approaches available that can handle such complex interactions [18].

In a related work, Henia et. al. proposed a compositional performance analysis methodology with the main goal to directly exploit the successful results of classical scheduling theory, in particular for sharing a single processor or a single communication link.

In their work, they defined two types of interfaces that may be placed between components. The first one is Event Model Interfaces (EMIF) which performs a type conversion between certain arrival patterns, that is, they change the mathematical representation of event streams. The second is Event Adaption Functions (EAF) which must be used whenever there exists on EMIF. In this case, the hardware (HW) implementation of the analyzed system must be changed in order to make the system analyzable, for example by adding play-out buffers between components.

The work of Henia et. al. has the following limitation; the compositional approach is bound to a limited set of classical arrival patterns that is often not sufficient to represent event streams with complex timing behaviors. As a result, they must be represented in one of the supported arrival patterns, usually with loss in accuracy. Also the arrival patterns often need to be adapted between components, either again with loss in accuracy (EMIF), or even with enforcing a change in the system HW implementation (EAF). Finally the approach is not compositional in terms of the resources, as their service is not modelled explicitly.

To overcome these limitations, [17] extended the compositional performance analysis framework presented by Henia et. al., they introduced the concept of intra-stream contexts that specify a cyclic pattern of different events that arrive on an event stream. The timing properties of the event stream are thereby decoupled from the cyclic event pattern, and are specified using a set of classical arrival patterns. On such an event stream with intra-stream context, the WCET of every event, when triggering a computation resource, is then determined from its event type.

2.5 Timed Automata based performance analysis

Timed automata is a popular formalism for the specification of real- time systems. It can be used in combination with a logic language to verify system properties by model checking [28].

The work of [9] showed that the schedulability analysis of an event driven system can be represented as a reachability problem for timed automata and thus can be tackled with model checking. The timed automata based schedulability analysis is implemented in the TIMES tool [34]. TIMES permits users to analyze systems that are described as a set of tasks which are triggered either periodically or by external event streams modelled through appropriate timed automata. However, the TIMES tool is limited to the schedulability analysis of single processors. Thus, it is not suited for performance analysis of distributed systems.

In a related work, [15] presented an approach to performance analysis of distributed embedded systems based on the model checking of timed automata networks. They modelled the environment and the resources of a system as timed automata. The various components are then composed into a network of timed automata that models a distributed embedded system. The performance properties of the system are verified through exhaustive model checking. They used UPPAAL for the modelling and verification of timed automata networks.

The UPPAAL (University of Uppsala, Sweden) tool environment allows users to validate and verify real-time systems modelled as networks of timed automata [13].

Their approach models the environment and the hardware resources. The timed automata models of the single system components are aggregated into a timed automata network that represents a distributed embedded system.

In modelling the environment, Fig.7 shows a timed automaton that models a periodic event stream with period *P*. After an undefined initial offset the automaton generates events at intervals of exactly *P* time units. The generation of an event is modelled by the increment of the global variable *req*. Fig.7 shows a timed automaton that models a periodic event stream with jitter $J \leq P$.

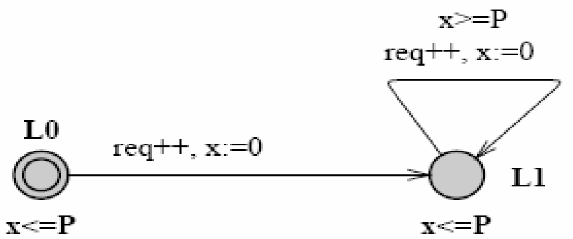


Fig. 7 : Timed automata model for a periodic event stream

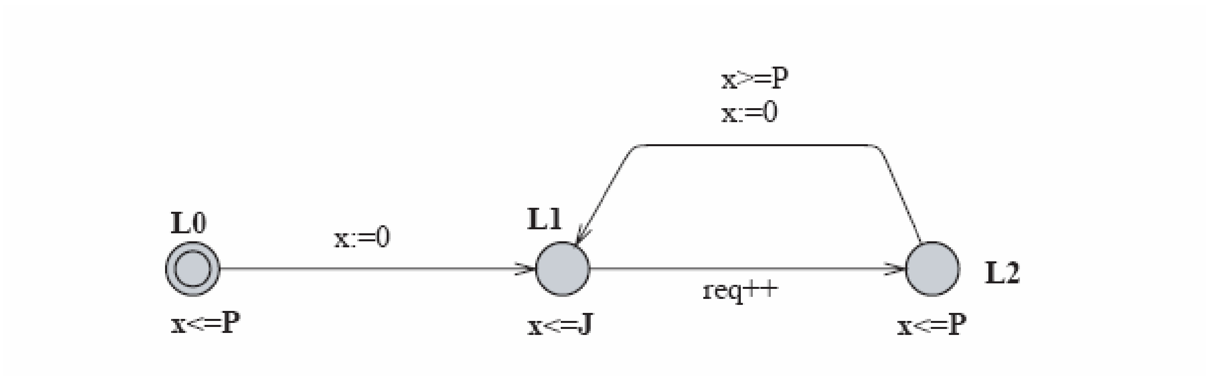


Fig. 8: Timed automata model for a periodic event stream with jitter

From the above, new event stream models can be designed easily. Basically any deterministic event stream can be modelled.

In modelling the hardware resources, each processing component is modelled as a separate timed automaton. A processing component is either idle or busy computing some function. In the same way, each communication link is modelled as a timed automaton. Each link is either idle or transporting some data. For shared resources the adopted scheduling policy determines the structure of the model. For example Fig.9 shows a timed automaton that models a hardware resource with two tasks implementing preemptive fixed priority (FP) scheduling. The resource can either be idle or process T1 or process T2. The location *pre T1* models the fact that T1 can pre-empt T2. The synchronization models a so-called urgent edge and makes sure that the corresponding edge is taken as soon as it is enabled, see [13] for details.

In performance analysis, timed automata models of the single system components are aggregated into a timed automata network that represents a distributed embedded system. The single components interact via global variables and channels. For example, suppose that the timed automaton of an input event generator increments a global variable *req* to model the request of a task activation on a certain resource.

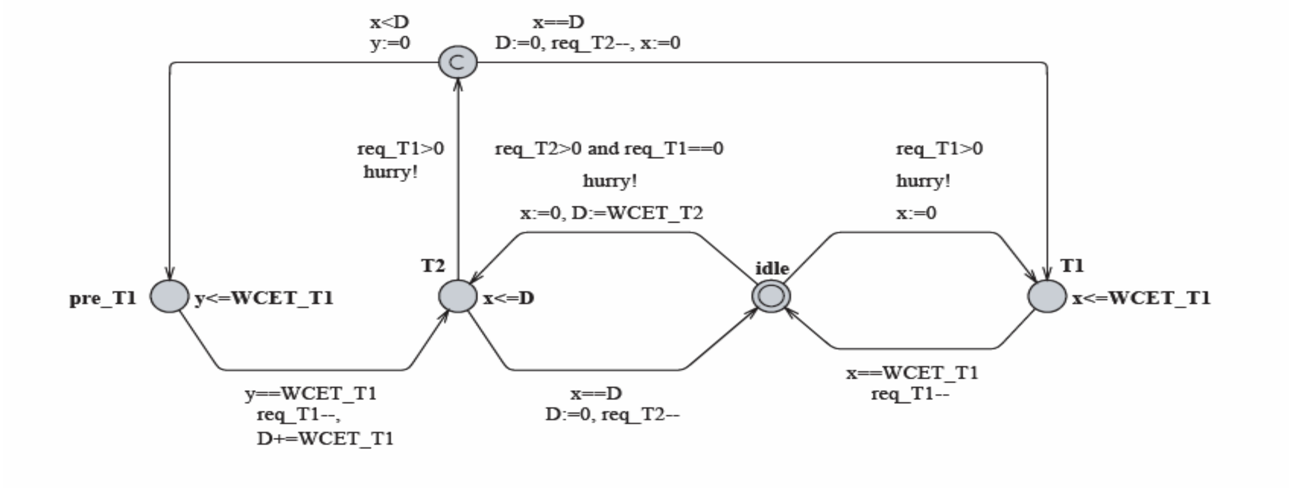


Fig. 9 : Timed automata model for a preemptive FP resource with two tasks

The timed automaton that models the corresponding resource is sensitive to increments of the variable *req* and immediately starts the execution of the corresponding task if no higher priority task has to be executed. The completion of the task execution is modelled by the decrement of the variable *req*. Let's suppose that the corresponding output event triggers a second task. This can be modelled by incrementing a second global variable *req2* simultaneously with the decrement of *req*. Again, another automaton will be sensitive to the increments of *req2*, start the corresponding task and so on. In this way, the propagation of events through the distributed system can be easily modelled.

The performance attributes of a distributed embedded system are derived by verifying properties of the corresponding timed automata network. For instance, to ensure that the maximum backlog of a certain task does not exceed a given value *b*, it is sufficient to verify the following property by model checking: $AG\ (req \leq\ b)$ where 'AG' stands for 'always generally' (= invariantly) and *req* is the global variable that counts the activation requests of the corresponding task. Also it is possible to derive the *exact* maximum backlog by finding the smallest *b* that satisfies the above property which can be done by using a binary search strategy.

The verification of end-to-end delays is a little more involved as it requires to adapt the timed automata models of the corresponding input event generators.

Fig.10 shows the variant of a periodic event stream generator that permits to verify end-to-end latencies.

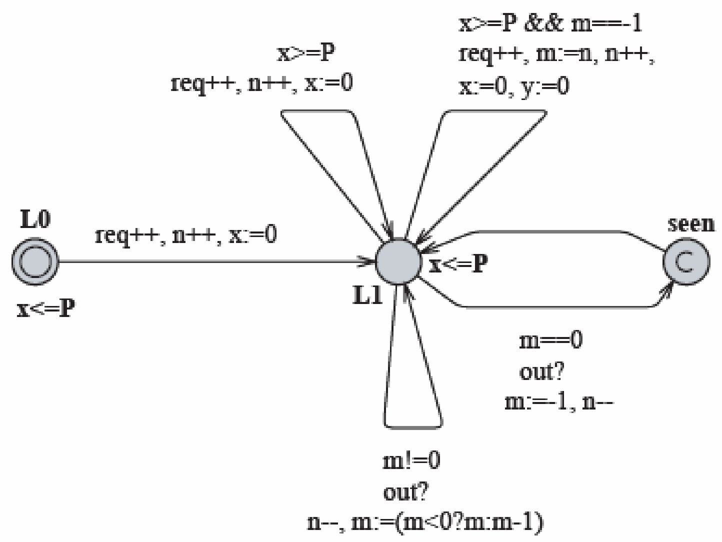


Fig. 10: Timed automata model for a periodic input generator that measures the end-to-end delay

The automaton is synchronized with the system output over the global channel *out* and can keep track of the amount of time that passes between the generation of an event and its output from the system. Basically, the automaton can generate input events in the same way as the automaton of Fig.7 (left upper transition), but it can also arbitrarily choose to measure the end- to-end delay of an event (right upper transition). The variable *n* (initially 0) keeps track of the number of events that have been fed into the system and for which no response (a synchronization over the channel *out*) has been received yet. The clock *y* measures the response time and *m* (initially -1) equals the number of responses that must be discarded before the one used for the measurement is seen. At most one measurement can be in progress and $m = -1$ if no measurement is in progress.

Similarly measuring automaton variants are available also for other event streams. To ensure that the worst-case end-to-end delay of an event does not exceed a given value *d* it is sufficient to verify the following property by model checking: $AG\ (IG.seen \rightarrow IG.y < d)$

where we assume that 'IG' is the name of the measuring automaton. Again, the *exact* worst-case end-to-end delay can be determined by finding the smallest *d* that satisfies the property.

The timed automata method for performance analysis based on model checking permits to derive not only hard but also exact bounds for performance properties of a distributed system but with space problem. The modelling of a distributed embedded system as a network of timed automata can easily lead to a state space explosion turning the analysis effort to be prohibitive [10].

2.6 Stochastic analysis method

Stochastic analysis method is rarely used for performance analysis because of its tighter analytic bounds. [22] uses stochastic characterization for inter-arrival times, execution demands and deadlines, and relies on queuing theoretic methods for performance evaluation. The stochastic workload models can result in tighter analytic bounds and hence in more economical designs, but at the expense of some (usually controlled) fraction of missed deadlines. Because of this, their application area is limited to soft real-time.

As discussed above each of these approaches has its short coming hence, in practice, a hybrid approach is adopted.

3 Comparison of Performance Analysis Methods

Most of the comparison and classification criteria for performance analysis methods are not directly quantifiable, but play an important role in the distinction of performance analysis approaches.

The classification criterion for performance analysis methods is given by the set of analyzable performance metrics. A performance analysis approach may support the analysis of system characteristics like timing aspects, memory requirement, resource utilization or power consumption. The analysis of timing aspects includes the determination of best-case and worst-case latencies and

end-to-end delays. The analysis of the memory requirement is often related to the determination of worst-case buffer fill levels.

Modelling scope:

A fundamental comparison criterion for performance analysis methods is the modelling scope. By the modelling scope of a certain approach we mean the set of distributed embedded systems that can be represented and analyzed using the modelling power of the method. For instance, the capability to model several particular system characteristics, such as hierarchical scheduling, blocking times, multiple task activation etc., differentiates the modelling scopes of the various performance analysis methods.

Correctness and accuracy:

A worst-case analysis is said to be correct if the result is a hard upper bound for the real worst-case performance of the considered system. In other words, there are no reachable system states which would allow the calculated bound to be violated.

The accuracy of a performance analysis is usually not quantifiable because the exact worst-case performance of the considered system is unknown. However, a performance analysis method is more accurate than another for a certain system if it provides a tighter upper (lower) bound for the worst-case (best-case) performance.

Modularity:

Performance analysis methods can be classified into modular and holistic approaches. The modular approaches analyze the performance of single components of the system and propagate the results in order to determine the performance of the entire system. In contrast, the holistic approaches consider the system as a whole. Modular performance analysis methods are typically less complex and easier to reuse than holistic ones.

Modelling effort and tool support:

An important criterion for the comparison of performance analysis methods is the effort that it costs the designer to create system models. The modelling effort can be largely alleviated by appropriate software tools.

Analysis effort:

This criterion considers the computational effort that is necessary to obtain performance analysis results. For instance one could compare the running times of the tools that implement the different performance analysis approaches.

Scalability:

Another relevant comparison criterion for performance analysis approaches is scalability. This point is pertinent to several of the previous criteria: the modelling and analysis efforts as well as the accuracy of the results may be greatly influenced by the dimension of the analyzed system.

End-user complexity and learning curve:

ISSN 1512-1232

Other aspects that can be considered for the comparison of performance analysis approaches are the complexity experienced by the end-user that applies a certain method or tool, as well as the progression of its learning curve. In particular, these points are largely influenced by the amount of background knowledge that a user must acquire about a certain performance analysis approach in order to be able to apply it. The multitude of heterogeneities among the performance analysis approaches and the different levels of abstraction in the modelling of particular system attributes make this task very complex [29]. As a result, the comparison of the various modelling scopes is restricted to a number of key attributes and the result is shown in Table 2.

The comparison is based on the modelling capabilities of concrete implementations of the various performance analysis approaches. Also it is necessary to point out that a 'low/poor' in a cell of the table does not mean that the modelling of the corresponding system characteristic is conceptually impossible for the corresponding performance analysis approach. Only that, it denotes that no significant research has so far been conducted to integrate this particular aspect.

Table 2 : Comparison of performance analysis methods

	Simulation and Formal Analysis	Real-Time Calculus	Holistic Scheduling Analysis	Compositional Method	Timed Automata Based Performance Analysis
Resource Utilization	5	5	3	4	5
Memory (Buffer Spaces)	4	4	2	3	3
Throughput	3	4	4	3	4
End-to-end delays	4	2	2	2	4
Modularity	2	4	3	4	4
Response Time	3	4	3	3	4
Reusability	4	3	2	3	3

Interpretation

Very Good / Very High	-	5
Good / High	-	4
Average / Medium	-	3
Poor / Low	-	2
Very Poor / Very Low	-	1

4 Future Work

Future extensions of this work is possible, for instance there is a need to identify required minimal set of states to be covered to prevent state space explosion in timed automata method. Further research work is also required on a worst-case execution time (WCET) of a program during design and verification of real-time embedded systems.

5 Conclusion

Failure of many real-time embedded systems may endanger human life or may cause substantial loss in economic values. Embedded system designer will find it difficult to build a prototype for each design alternative to directly measure performance characteristics because of high implementation costs, stringent time-to-market constraints or the risk of being totally incorrect.

Performance analysis plays an important role in the design process of complex embedded systems for analyzing essential performance characteristics of system design at an early phase. It gives designer the choice of important design decisions before much time and resources are invested in detailed implementation of the system.

But each of the performance analysis technique has one or more limitations which can be partitioned into two sub-problems; program path analysis and micro-architecture modelling. These limitations are categorised as worst-case execution time (WCET) of a program when designing and verifying real-time embedded systems. The WCET depends both on the program flow, such as loop iterations and function calls, and on hardware factors, such as caches and pipelines. WCET estimates should be both safe, no underestimation allowed and tight as little overestimation as possible. This shows that in performance analysis of real-time embedded systems design, implementation of one particular method may not be enough, a hybrid approach may be the best option.

6 Acknowledgement

This work would not have been possible without the advice and support of staff of my department, Mathematics department, Ahmadu Bello Univerity. Zaria, Nigeria.

7 References

1. Agarwal A., Performance Tradeoffs in Multithreaded Processors.*IEEE Transactio on Parallel and Distributed Systems*, 3(5): (September 1992) 525–539.
2. Alexander Viehl, Timo Schonwald, Oliver Bringmann and Wolfgang Rosenstie. Formal Performance Analysis and Simulation of UML/SysML Models for ESL Design (http://citeseerx.ist.psu.edu) 2009.
3. Benini L., D. Bertozzi, D. Bruni, N. Drago, F. Fummi, and M. Poncino. SystemC cosimulation and emulation of multiprocessor SoC designs. *IEEE Computer*, 36(4): (2003) 53–59.
4. Baruah S. K. A general model for recurring real-time tasks. In *Proceedings of the*
    a. *IEEE Real-Time Systems Symposium (RTSS)*, (1998) pages 114–122.
5. Baruah, S. K., D. Chen, S. Gorinsky, and A. K. Mok. Generalized multiframe tasks. *Real-Time Systems*, 17(1): (1999) 5–22.
6. Baruah S. K.. Dynamic and static-priority scheduling of recurring real time tasks. *Real-Time Systems*, 24(1): (2003) 93–128.
7. Chakraborty S., S. K¨unzli, and L. Thiele. A general framework for analysing system
    a. properties in platform-based embedded system designs. In *Design, Automation and Test in Europe (DATE)*, Munich, Germany, IEEE Press, (2003) pages 190–195.
8. Chakraborty S. and L. Thiele. A new task model for streaming applications and its schedulability analysis. In *Design, Automation and Test in Europe (DATE)*, (2005) pages 486–491.
9. Christer Norstrom, Anders Wall, and Wang Yi. Timed automata as task models forevent-driven systems. In *RTCSA '99: Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications*, Washington, DC, USA, IEEE Computer Society, (1999) page 182.
10. Ericsson C., A. Wall, and W. Yi. Timed Automata as Task Models for Event-Driven Systems. In *RTCSA '99: Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications*, Washington, DC, USA, (1999) page 182.
11. Guardian.co.uk. (May 7, 2008) "Toyota shares fall for 6th day as recall woes deepen", London http://www.guardian.co.uk/business/feedarticle/8921329. Retrieved 2010-04 -30.
12. Grotker, T., S. Liao, G. Martin, and S. Swan. *System Design with SystemC.* Kluwer Academic Publishers, Boston, MA, USA, May 2002.
13. Gerd Behrmann, Alexandre David, and Kim G. Larsen. A tutorial on UPPAAL. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time*

ISSN 1512-1232

Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004, number 3185 in LNCS, Springer-Verlag, (September 2004) pages 200-236.

14. Henia R., A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level Performance Analysis – the symta/s approach. *IEE Proceedings – Computers and Digital Techniques*, 152(2): (2005) 148–166.
15. Hendriks Martijn and Verhoef Marcel. Timed automata based analysis of embedded System architectures. Technical Report ICIS-R06003, ICIS, Radboud University, Nijmegen, The Netherlands, (2006).
16. Jersak M., R. Henia, and R. Ernst. Context-aware performance analysis for efficient embedded systems design. In *Proc. 7th Design, Automation and Test in Europe (DATE)*, (2004) 1046 - 1051.
17. Jersak M., Richter K., and Ernst R.. Performance analysis for complex embedded applications. *International Journal of Embedded Systems, Special Issue on Codesign for SoC (*2004).
18. Lothar Thiele and Ernesto Wandeler. Performance Analysis of Distributed Embedded Systems Networked Embedded Systems Handbook, CRC Press/Taylor & Francis 2009.
19. Lahiri K., A. Raghunathan, and S. Dey. System level performance analysis for designing on-chip communication architectures. *IEEE Transactions on Computer Aided-Design of Integrated Circuits and Systems*, 20(6): (2001) 768–783.
20. Le Boudec J.-Y. and Thiran P. Network calculus: a theory of deterministic queuing Systems for the Internet. Springer-Verlag New York, Inc., (2001).
21. Liu L. C. and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1): (1973) 46–61.
22. Lehoczky J. P. Real-time queueing theory. In *Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS)*, Washington, DC, USA, IEEE Computer Society (1996) page 186.
23. Mok A. K. and D. Chen. A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering*, 23(10): (1997) 635–645.
24. Maxiaguine A., S. K¨unzli, and L. Thiele. Workload characterization model for tasks with variable execution demand. In *Design, Automation and Test in Europe (DATE)*, Paris, France, IEEE Computer Society (Feb.2004) pages 1040–1045.
25. Nikolay Stoimenov, Lothar Thiele ETH Zurich. Determining a Schedulability Region using Modular Performance Analysis and Real-Time Interfaces in WSN. (2007)
26. Naedele M., L. Thiele, and M. Eisenring. Characterising variable task releases and processor capacities. In *Proceedings of the 14th IFAC World Congress,* Beijing, July (1999).
27. Pop P., P. Eles, and Z. Peng. Performance estimation for embedded systems with data and control dependencies. In *Proceedings of the 8th International Workshop on Hardware/Software Co-Design (CODES)*, (2000) pages 62–66.
28. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2): (1994) 183-235.
29. Simon Perathoner. Evaluation and Comparison of Performance Analysis Methods for Distributed Embedded Systems Master's thesis submitted to the Swiss Federal Institute of Technology Zurich (2006).
30. Simon Kunzli, Francesco Poletti, Luca Benini, Lothar Thiele. Combining Simulation and Formal Methods for System-Level Performance Analysis In Proceedings of DATE. 2006, 236-241.
31. Thiele L., S. Chakraborty, M. Gries, A. Maxiaguine, and J. Greutert. Embedded software in network processors - models and algorithms. In *Proceedings of the 1st International Workshop on Embedded Software (EMSOFT)* London, UK, Springer-Verlag (2001) pages 416–434.
32. Thiele L., E.Wandeler, and S. Chakraborty. A stream-oriented component model for performance analysis of multiprocessor DSPs. *IEEE Signal Processing Magazine, special Issue on Hardware/Software Co-design for DSP*, 22(3): (2005) 38 - 46.
33. Thiele L., S. Chakraborty, M. Gries, and S. K¨unzli. A framework for evaluating design tradeoffs in packet processing architectures. In *Proceedings of the 39th Design Automation Conference (DAC)*, New Orleans LA, USA, ACM Press (June 2002) pages 880–885.

ISSN 1512-1232

34. Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Times - A tool for modelling and implementation of embedded systems. In *TACAS '02: Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, London, UK,. Springer-Verlag (2002) pages 460-464
35. Toyota USA Newsroom http://pressroom.toyota.com/pr/tms/toyota/toyota-consumer-safety-advisory-102572. Retrieved April 2010.
36. Takahashi and Yoshio http://online.wsj.com/article/BT-CO-20100209702754 html?mod=WSJ_World_MIDDLEHeadlinesAsia *Wall Street Journal.* Retrieved April 4, 2010.
37. Tindell K. and Clark J. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing & Microprogramming*, 40(2-3): (1994) 117–134.
38. Wandeler Ernesto, Modular Performance Analysis and Interface-Based Design for Embedded Real-Time Systems. A dissertation submitted to the Swiss Federal Institute of Technology Zurich.Embedded.com – Under Hood:Robot Guitar embeds autotuning (2006).
39. www.en.wikipedia.org/wiki/Embedded_system. Retrieved *2010-02-29.*

Article received: 2012-08-09