A COMMUNICATION SCHEDULING STRATEGY FOR SAFETY-CRITICAL EMBEDDED SYSTEMS

B. Abdul Rahim*

Dr. K. Soundara Rajan**

ABSTRACT

Scheduling in an embedded system has been researched for long in the recent past. Number of scheduling algorithms proposed and implemented for general architectures and recently in particular applications for the embedded systems. As the time triggered paradigm is taking control in safety critical applications, progressively change in scheduling strategies has came up, in this move we propose a scheduling strategy that will be optimal for implementation on distributed system. Pre-emptive algorithms have been in existence basically for event-triggered architectures. Considering a non pre-emptive execution environment in which the activation of processes and communications is triggered at certain point of time, for which we generate a schedule table. In order to run a predictable real-time application, the overhead of the kernel and a worst case delay has to be determined which can be guaranteed under any condition. Such a scheduling policy will be well suited for safety-critical application class. For evaluation purpose, we used conditional process graphs to represent the scheduled processes and the communication protocol used is Flexray as it provides flexibility and time-triggered assignments.

Keywords: *Reliability, Safety-Critical Systems, Embedded systems, Time-triggered systems.*

*Professor, Department of Electronics and communication Engineering, Annamacharya Institute of Technology and Sciences, New Boyanapalli, Rajampet, Andhra Pradesh, India.

**Professor, Department of Electronics and communication Engineering, Jawaharlal Nehru Technological University Anantapur College of Engineering, Anantapur, Andhra Pradesh, India.

1. INTRODUCTION

An embedded system in the recent past replacing most of the common discretely assembled system into an efficiently organized compact system of heterogeneous nature. Embedded system can be viewed as a set of interacting processes mapped on an architecture consisting of several programmable processors and ASICs interconnected by a communication channel following a particular protocol. Process interaction is not only in terms of dataflow but also to represent control flow, because some processes can be activated depending on conditions computed from previously executed processes. Scheduling for performance estimation and synthesis of embedded systems has been intensively researched in the recent past [1]. As known, simple and good scheduling strategies have been designed and developed for event triggered systems. Here we are concentrating on scheduling in time triggered systems consisting of multiple processors and ASICs, which are interconnected by a communication channel. We consider a non pre-emptive execution environment communication as well as the activation of processes is triggered at certain points in time, and a schedule table is developed, derived by a worst case delay which is guaranteed under any condition. Such a scheduling strategy is well suited to a large class of safety critical fault tolerant applications [2].

The system architecture is built on a communication model which is based on the Flexray protocol. The Flexray protocol is well suited for safety critical distributed real-time control systems and represents one of the emerging standards for several application areas like, for example, automotive electronics [3]. The scheduling strategy is based on a communication model and execution environment. The requirements of the communication protocol such as overheads due to communications and the execution environment are considered during the scheduling process. Our scheduling algorithm performs an optimization of parameters defining the communication protocol which is essential for the reduction of the execution delay.

The next section of the paper describes the basic differences and advantages of event triggered and time triggered systems. The conditional process graph and system architecture used to describe the scheduling of processes are presented in the section 3 & 4 respectively. The problem formulation and scheduling strategy are discussed in the preceding sections. The conclusions and references of the implementation are presented in final sections.

2. EVENT VS TIME-TRIGGERED SYSTEMS

Real-time systems complexity grows not only with the physical system being controlled, but also with the tightness of the predictability and dependability requirements. Also, different architectures must be considered depending on the type of real-time control system and its predictability requirements. Event-triggered approach is dictated by the external environment. In the event-triggered real-time systems, all communication and processing activities are initiated whenever a significant change of state, i.e., an event other than regular event of a clock tick, is noted. The signaling of significant events is realized by the well-known interrupt mechanism. The event-triggered based systems require a scheduling strategy to achieve the appropriate software task that services the event [4].

In an event-triggered system, state changes are observed through specific sensors that generate task activation that request modification of associated objects. But, being in an event-driven environment, this task can start execution only if the scheduler grants processing time: consequently, only after task finishes its execution, the internal object is correctly updated and reflects the last change from the associated state. Therefore, a temporal uncertainty is introduced by this approach, because it is impossible to know exactly the time points when this task starts and finishes, mainly for two reasons: execution of a task can be delayed or interrupted by tasks with higher priority, critical sections or uninterruptible code segments. Varying durations of these delays cause temporal uncertainty and the limited precision and granularity of any reference clock on the time measurement as well as on the time depending scheduling decisions which also induces temporal uncertainty. These temporal uncertainties affect system's predictability and determinism, because it is impossible to guarantee, that system responds to relevant environmental changes in a timely manner, even if the real-time scheduling problem might be solved [5].

Another issue that affects system's predictability and determinism refers to the manner in which systems detect sensor's faults, especially when referring to timing faults. In an event-triggered architecture these timing sensors faults cause faulty tasks activations to update the object in accordance to the value from corresponding state. Consequently, it is impossible to guarantee timely response in an event-triggered approach in the presence of sensor (timing) faults [6]. Besides the above shortcomings, there are also some advantages for using event triggered architectures such as design effort for complex systems is lower than for the timed-triggered architecture approach and exhibits better resource utilization.

In the time-triggered systems, all communication and processing activities are initiated at predetermined points in time. There is only one interrupt and that is the periodic clock interrupt, which partitions the continuum time into sequences of equally spaced granules [4]. If the main advantage of event-driven approach is flexibility and better resource utilization, the main advantage of time-driven approach is predictability.

In the timed-triggered architecture, task activation for updating object based on relevant state changes of corresponding states is in the sphere of computer system and takes place at predetermined points of time. Consequently, timed triggered architecture is much more suited to reliable representation of states than event-triggered approach. A timed-triggered activated task has a fixed activation period and can be scheduled off line without the knowledge of future requests: therefore, while for timed triggered architecture a fixed activation period for tasks could be defined, for event triggered architecture at least a minimum inter arrival period for activated task should be assessed. A main shortcoming for timed-triggered approach is that, in order to obtain a predictable real-time application, the worst case scenario must be known a-priori in order to be considered into the development process.

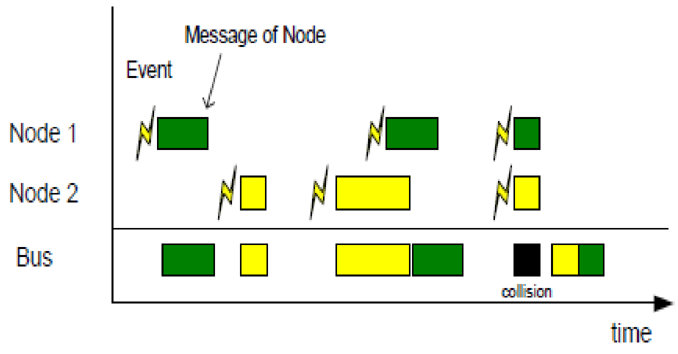

Figure 1. Event-triggering and Time-triggering illustration.

3. CONDITIONAL PROCESS GRAPH

Many real-time applications are periodic and running at multiple rates. To describe each application or process we use task graph model [7]. The application is partitioned into task graph, which is a directed, acyclic, polar task graph Γ(V,Es,Ec) with conditional edges. Each node $P_i \in V$ is represented as a process. Such a process can be an ordinary process or a communication process which capture the message passing activity. Es is a set of simple edges and Ec is a set of the conditional edges. Es ∩ Ec = Ø and Es □ Ec = E, where E is the set consisting of all edges. An edge $e_{ij} \in E$ from Pi to Pj indicates that the output of Pi is the input of Pj. The graph is polar meaning there are two nodes, called Source and Sink, which conventionally represent the first and last process. These are dummy processes nodes introduced, so that all other nodes in the graph are successors of the source and predecessors of the sink respectively as shown in figure 2.

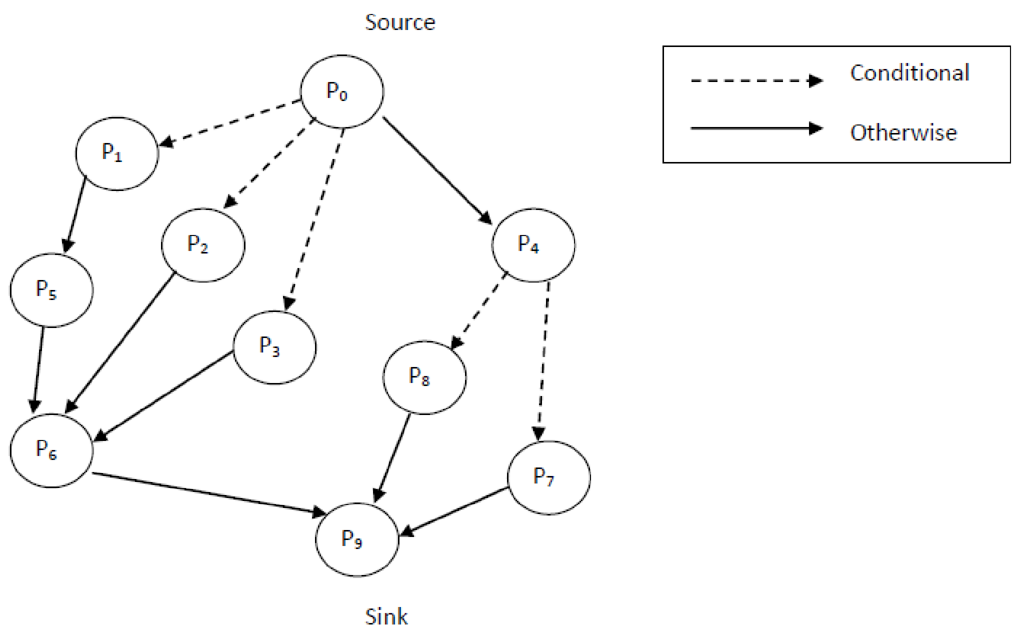

Figure 2 Conditional Process Graph

The mapping of processes is given by a function F: V→PE, where PE={pe1,pe2, …, peN} is the set of processing elements. For any process Pi, F(Pi) is the processing element to which Pi is assigned for execution. Each process Pi, assigned to processor is characterized by the worst case execution time (WCET). For illustration P0 and P9 are the source and sink nodes respectively. The nodes denoted P1 to P8 are ordinary processes. The transmission of a message on a conditional edge will take place only if the associated condition is satisfied. A process can be activated only after all its inputs are available and issues its outputs only when it is terminated. Once activated, the process cannot be preempted by any other processes. After the termination of a process which produces a condition, the value of the condition is broadcasted from the corresponding processing element to all other processing elements. This broadcast is scheduled as soon as possible on the communication channel, and considered together with the scheduling of the messages.

4. SYSTEM ARCHITECTURE

The hardware architecture consist nodes/processing element connected by a broadcast communication channel as shown in figure 3. Every node consists of a Flexray communication controller, a CPU, a host interface controller. The Flexray was designed for distributed real-time application that requires predictability and reliability (e.g, drive-by-wire, etc.). it integrates all the services necessary for fault-tolerant safety-critical real-time systems. The bus is a broadcast communication channel, so the message sent by a node is received by all the other nodes connected to it. The Flexray is a hybrid type of protocol composed of static and dynamic segments, which are arranged to form a bus cycle that is repeated periodically. The static segment is similar to TTP and employs a generalized time-division multiple-access (GTDMA) scheme. The dynamic segment is similar to Byteflight and uses a flexible TDMA (FTDMA) bus access scheme [8]. But basically it is TDMA scheme where each node can transmit only during a predetermined time interval called slot. In such a slot, a node can send several messages packaged in a frame and the sequence of slots allotted to all nodes in the architecture is called a TDMA round. Several TDMA rounds can be combined together in a cycle that is repeated periodically. The sequence and lengths of the slots are the same for all the rounds. However, the contents of the frame and the lengths may differ.

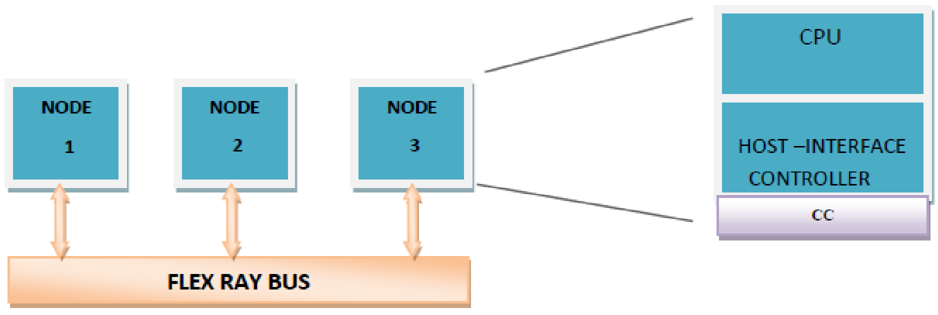

Figure 3: The Hardware Architecture

The software architecture runs on a CPU in each node has a real-time kernel as its main component. Each kernel has a schedule table that consists of all the information that is needed to take decisions on activation of processes and transmission of messages, based on the values of the condition. In order to run a predictable hard real-time application, the worst case execution time and overhead of the kernel has to be determined.

5. PROBLEM FORMULATION

For Time-Triggered system, all the activity is derived from the progression of time which means that there are no other interrupts except for the timer interrupt. As the application is for safety-critical, which has several operating modes like polling of the I/O or diagnostics etc., and each mode is modeled by a conditional process graph. Each conditional process graph in the application has its own independent period. Typically, a global deadline on the delay of each conditional process graph is imposed and not individual deadlines on the processes.

Here we are to develop a schedulability for a system modeled as a set of processes by a conditional process graph.

The worst case execution delay of a process is estimated taking into account the administrative overhead of message passing and kernel decision making for process activation and is given as

$$T_{Pi} = (C_{pi} + \alpha + \beta) .(1+ k)$$

Where α is the overhead for communication from Pi to the processes on the same node, and β is the overhead for communication between the processes of different nodes. C_{pi} is the WCET of the code of process Pi and k is the kernel overhead.

The α consists of summation of number of processes sent by Pi on the same node for execution where as the β is the summation of number of processes sent by Pi to the other nodes. From this the delay on the system execution time for each operating mode can be determined so that this delay is as small as possible.

6. THE SCHEDULING STRATEGY

The scheduling algorithm should perform allocation of the tasks on the node and scheduling of the tasks on the processing elements simultaneously such that the algorithm can take advantage of the resource sharing. This is different from the algorithms what proposed in [9][10], where it is assumed that the allocation of the tasks on the processing elements is fixed. Here largely it is the optimization process analyzing the best schedule process and allocate to the particular processing element. First of all we have to define the message schedule, for this worst case execution time has to be computed as according to the formula given in the above section.

```
Message_schedule
    Slot=slot of the node sending the message
    Round=current_time/round_length
    If current_time-round*round_length>slots available in round then
            Round=next_round
    End if
    While message not fits in the slot of the round then
            Insert(allow slot length to fit, recommended slot length)
            Round=next_round
    Else
    Place schedule table with (message, round, slot)
```

end

The major concern is the scheduling of the messages on the Flexray bus, considering a given order of slots in the TDMA round and particularly for the given slot lengths. This computes the schedule and corresponding tables based on the given slot order and slot lengths. For optimization concern, ordering of the slot and slot lengths have to be determined so that execution delay is as small as possible.

Algorithm*:*

```
Start with first slot of the TDMA round;
Find the node which is allotted a slot producing smallest delay;
    Selected node is given first slot;
For next task allocate by checking the smallest delay as above
Continue
Form a best schedule for all the tasks;
```

With the processes considered for conditional process graph and for experimental purpose we considered two nodes connected with a Flexray communication protocol. The schedules obtained are based on the best processing element gets the slot in the TDMA round. Also we did not consider the dynamic segment of the frame which was kept vacant as we are concentrating on the static scheduling only. The scheduling results are shown in figure below for the example processes. The node 1 has been allotted ST slot 2 and node 2 transmits through ST slots 1 and 3. For each of these slots, the controller host interface reserves a buffer that can be written by the CPU and read by the communication controller. These buffers are read by the communication controller at the beginning of each slot, in order to prepare the transmission of frames. When the time comes for an ST message to be transmitted, the CPU will place that message in its associated ST buffer of the controller host interface. As specified by the schedule table that it will be sent in that slot.

Round 1				Round 2				Round 3				Round 4				Round 5			
ST segment			DYN segment	ST segment			DYN segment	ST segment			DYN segment	ST segment			DYN segment	ST segment			DYN segment
	P_0			P_1	P_4	P_2		P_3	P_5	P_8		P_7	P_6				P_9		

1 2 3 1 2 3 1 2 3 1 2 3 1 2 3

Slots **Figure 4: Flex ray communication cycle**

At the beginning of each communication cycle, the CC of a node resets the slot and at the beginning of each communication slot, the controller checks if there are messages ready for transmission and pack them into frames. Messages selected and packed into ST frames will be transmitted during the bus cycle/round that is about to start according to the schedule table. For example in the above figure, P1 and P2 are placed into the associated ST buffers in the controller host interface in order to be transmitted in the second round.

7. CONCLUSION

An approach to process scheduling for safety-critical distributed embedded systems is presented with a simple illustration here. The quality of the scheduling is improved by considering the overheads of real-time kernel and the communication protocol. The scheduling algorithm can be used for performance estimation and system synthesis. To further improve the quality of scheduling, better heuristic can be used for selecting the slots order and lengths.

8. REFERENCES

1. Balarin. F, Lavagno. L, Murthy. P, Sangiovanni-Vincentelli. A, "*Scheduling for Embedded Real-Time Systems",* IEEE Design & Test of Computers, Jan-Mar, 1998.
2. Kopetz. H, "*Real-Time systems – Design Principles for Distributed Embedded Applications",* Kluwer Academic Publishers, 1997.
3. Abdul Rahim. B, Soundara Rajan. K, "*Fault-Tolerance in Real-Time Systems through Time-Triggered Approach",* CiiT Intl. Journal of Digital Signal processing, Vol.3, No.3, April, 2011, PP-115-120.
4. Kopetz. H, "*Event triggered versus time triggered real-time systems",* Technical report 8/91, Insitut fur Technische Informatik TU Vienna, Austria, 1991.
5. Kopetz. H, Ki. K, "*Temporal Uncertainties in interaction among real-time objects",* Proceedings of 9th Symposium on Reliable Distributed Systems, PP. 165-174, 1990.
6. Poledna. S, "*Tolerating Sensor Timing Faults in Highly Responsive Hard Real-Time systems",* IEEE Transactions on Computers, Vol.44, No.2, 1995.
7. Wayne Wolf and Jorgen Staunstrup, "*Hardware/software co-design-principles and practice",* Kluwer Academic Publishers, 1997.
8. Abdul Rahim. B, Soundara Rajan. K, "*Comparision of CAN and Flexray protocols for Real-Time Systems",* Proceedings of International conference on Recent Advancements in Computing Technologies (ICRACT'11), Chennai, Mar, 2011.

9. Petru Eles et.al, *"Scheduling of conditional process graphs for the synthesis of embedded systems",* Proceedings of DATE, PP-23-26, 1998.
10. Paul. Pop, *"Scheduling and communication synthesis for distributed real-time systems",* Ph.D thesis, Linkopings University, 2000.