Effective Scheduling Algorithm and Scheduler Implementation for use with Time-Triggered Co-operative Architecture

S. Kuankid[1], A. Aurasopon[1], W. Sa-Ngiamvibool[1]
[1]*Department of Engineering, Mahasarakham University, Kantharawichai district, Mahasarakham 44150, Thailand*
sanykuo@hotmail.com

***Abstract*—Time-triggered Co-operative (TTC) scheduler provides a simple and reliable operating environment that matches precisely the needs for use in safety-related applications. However, in the design and implementation phases, such scheduler may be suffered from several failure modes that are the fragility of scheduler and the impact of task jitter. The main contribution of this paper is to address the need for an effective scheduling algorithm and scheduler implementation to assure that the TTC can be executed appropriately without an impact on the timing behavior in the system. The paper has developed an algorithm called "TTSA-MTI" to automate the process of scheduling TTC-MTI scheduler which is designed based on the employment of "Sandwich Delay" mechanisms and technique of "Multiple Timer Interrupt". The results show that the proposed algorithm can help in a significant reduction of computation time and achieve better performances in task release jitter as opposed to a related scheduler. In conclusions, the paper indicates that TTSA-MTI algorithm is an effective task scheduling for use with a range of TTC architecture.**

***Index Terms*—Time-triggered co-operative scheduler, offline scheduler, sandwich delay mechanism, time-triggered architecture.**

I. INTRODUCTION

When developing software architecture that applied in many safety-related and safety-critical embedded systems, it is generally recognized that the use of "time-triggered" (TT) architecture offers significant advantages over other alternative approaches [1], [2]. The TT architecture is referred as "offline scheduler" or "static scheduler" architecture, it is statically computed before the system begins to execute [1]–[4]. This means that such architecture prepares a complete planning sequence of all task set before executing the system. As a result, the TT system can help to produce highly predictable patterns of behaviour and suit for many safety-related applications such as in automotive systems, biomedical equipment, and control applications [5], [6].

In the TT designs, task scheduling can be characterized into two different types of scheduling algoritms: time-triggered co-operative (TTC) and time-triggered pre-emptive (TTP) scheduler. The co-operative scheduler also known as "non-preemptive", only one task is executed at any point in time, this task runs to completion, and then return to the schedule [7]. As for pre-emptive scheduler, the task with the highest priority that is ready to execute is always allowed to control the CPU [3], [4]. However, many researchers stated that TTC scheduler provided a simple and reliable operating environment, with appropriate designs and implementations, this scheduler can be applied in the systems which have hard real-time characteristics [8]–[10].

When designing TTC architecture, the most cases are employed a typical "TTC-Dispatch" scheduler because this scheduler is simple and very useful while using in a resource-constrained system. Despite many advantages, there are failure modes that can greatly impair system performances. These are the fragility of scheduler, the problems of task jitter, and task overruns [5], [11]. To avoid these problems, there is other design option by developing the scheduler implementation based on the employment of "Sandwich Delay" (SD) mechanisms and technique of "Multiple Timer Interrupt" (MTI) called "TTC-MTI" scheduler. Reference [12] indicated that TTC-MTI scheduler achieved better performance in timing behaviour as opposed to TTC-Dispatch. However, as for task scheduling in TTC-MTI scheduler, there has been a few studies explore the algorithm to automate the process of such scheduler.

According to the aim of this paper, it is attended to address the need of an effective scheduling algorithm and scheduler implementation for use with a range of TTC architecture to assure that the developers can select an appropriate scheduling method and the scheduler implementation without having an impact on the timing behaviour in the system. This paper is organized as follows: Section II gives related work of scheduling in real-time system. Section III presents the operation of TTC scheduler. Section IV modifies the technique of TTSA-MTI scheduler. Section V evaluates the proposed algorithm. Finally, Section VI concludes the use of proposed technique.

II. RELATED WORK

There has been a considerable amount of previous work

Manuscript received October 11, 2013; accepted February 8, 2014.
The work presented in this paper is supported by the Ministry of Science and Technology, Royal Thai Government, Thailand.

on the scheduling algorithms and scheduler implementations in time-triggered architecture.

A. Scheduling Algorithms in Real-Time Embedded Systems

In literatures, Cottet stated that scheduling algorithm assigns tasks to processor and provides an ordered list of tasks [3]. Therefore, scheduling algorithm is the key component to basically determine the way in which task will be executed by the scheduler. In task scheduling, while the order of task execution is determined based on its priority, the tasks are assigned priorities either "Fixed priority" or "Dynamic priority" [2], [4], [13].

In a fixed priority assignment, the priorities of the tasks are assigned before the scheduler begins to execute and those priorities remain unchanged during the life-time of the system. The most popular fixed priority scheduler is the Rate Monotonic and Deadline Monotonic algorithm [14], [15]. As for the dynamic priority, the priority assigned to a given task can be changed at run time. It is basically based on a dynamic parameter that may changes during the system evolution [3]. Examples of common dynamic priority algorithm are Earliest-Deadline-First and Least-Laxity-First [4], [16].

In cases of task scheduling in time-triggered resource-constrained embedded system, one of the simplest scheduler is a cyclic executive, it generates a complete sequence of periodic task with highly-predictable schedule [9]. The TTC design is also a simple cyclic-executive scheduler that designs complete co-operative tasks on a processor according to an offline scheduler [7]. In addition, there are other design options that involve task pre-emption. For examples: Time-Triggered Rate Monotonic (TTRM) and Time-Triggered Hybrid (TTH) scheduler [7].

B. Scheduler Implementations in TT Architecture

With regard to implement the TT scheduler in small embedded devices, many researchers agreed with building software without real-time operating system (RTOS) because such system requires a large amount of computation and memory resources which is not readily available in resource-constrained environment [7], [17]. Therefore, despite a full RTOS, some forms of simple schedule is generally be implemented.

In literatures, the most cases of TTC architecture were implemented by "TTC-Dispatch" scheduler. For example, Pont employed TTC-Dispatch scheduler by using simple embedded operation system which integrated of operating system becomes a part of the application itself [7], [18]. This work demonstrated that such scheduler used a few hundred line of high-level programming, very low processor load, easy to port the system onto a new device environment, and very useful while using in a resource-constrained system. Besides, there have been significant studies to improve the capability of such scheduler. As Phatrapornnant and Pont has presented, they maintained their low-jitter characteristics with dynamic voltage scaling (DVS) technique to reduce system power consumption in TTC scheduler [5]. In similar case, Hughes employed a Task Gardian (TG) mechanism to reduce the impact of task overruns in TTC system [19].

III. TTC SCHEDULER OPERATION

The operation of a traditional TTC scheduler is considered to analyse cause of the problems while employing such system.

A. The Typical TTC-Dispatch Scheduler

The normal operation of TTC-Dispatch scheduler is briefly described in this part. An example of TTC-Dispatch design can be shown in Fig. 1. In this figure, there are two significant cycles for scheduling which are major cycle and minor cycle. The major cycle is the periodic time which tasks are executes in form of cycle. Such cycle is divided into a number of small cycles called "minor cycle" or "tick" interval.

The flowchart of a TTC-Dispatch scheduler can be illustrated in Fig. 2. This scheduler is usually implemented using an on-chip timer, which will be set to generate interrupts on a periodic basic or tick interval [11]. At the beginning, the parameters of the timing scheduler are set, followed by the initialization of all tasks, and adding those tasks in the task queue data structure. During the operation, Dispatch (function) is invoked from within an endless loop in the foreground processing. The Dispatch examines each task in list and executes any tasks which are due to run in this tick interval. After completed their execution, the scheduler then calls Sleep (function) to place the processor into an idle mode to reduce power of system. When the tick (timer ISR) occurs, the Update (function) is invoked and the scheduler then back to foreground again and the cycle thereby continues.

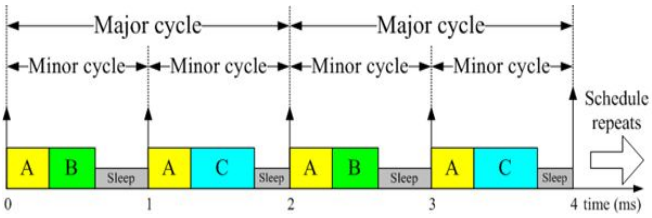

Fig. 1. The operation of a typical TTC-Dispatch scheduler.

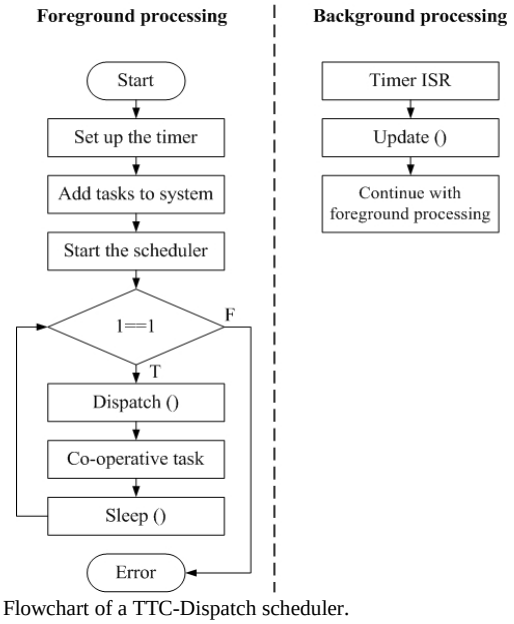

Fig. 2. Flowchart of a TTC-Dispatch scheduler.

B. Time-Triggered Scheduling Algorithm in TTC Scheduler

As for the TT scheduling algorithm, there has been designed the algorithm to automate the process of configuring TTC scheduler called "TTSA" algorithm [11]. This algorithm is designed to test the schedulability of all tasks in the system and identified a scheduler to assure that all task constraints are met. It can help developer to reduce time at the design process. However, in situations that the system has large-scale of task set, this may take a long computation time for testing schedulability due to the algorithm will proceed iteratively until the search identifies the first workable solution.

C. Causes of the Problems

Despite many advantages, if inappropriate designs, the TTC-Dispatch scheduler may have suffered from failure modes that can greatly impair system performances, can identify as follows:

1. The scheduler is very fragile in the during overload situations, since a task exceeding its predict execution time, this can generate a domino effect on the subsequent tasks [20]. More specifically, a task overrun that does not return causes the system to hang indefinitely [19].
2. The scheduler may be suffered from the variation in time called "release jitter", the causes of jitter can identify several possible which are variation time by interrupt timing behaviour, scheduler overhead, and task placement [5], [12].

Both problems can also be addressed by using various techniques such as DVS and TG mechanism [5], [19]. However, these techniques are very complicated due to the developers necessary to integrate complex techniques altogether and a need to construct with high resource requirements. In practical terms, this may be unsuitable applying with the realistic cases in resource-constrained system.

IV. MODIFIED TECHNIQUE OF TTC SCHEDULER

The software tool which can help to address the need for effective TTC scheduler is described in this section.

A. Modified Implementation Based On the Employment of "Sandwich Delay" and "Multiple Timer Interrupts"

Time-Triggered Co-operative Multiple Timer Interrupts (TTC-MTIs) scheduler is designed to address the impact of task placement and the problem of jitter at the starting time of the tasks. All tasks which employ such technique can be executed separately without an impact on the task placement of the preceding tasks.

To clarify the TTC-MTI scheduler, the technique has modified implementation by using two interrupts for schedule which are Tick interrupt and Task interrupt [12]. Tick interrupt is used to generate the periodic tick interval whereas Task interrupt is used to notify for execution task within the period of Tick interrupt. Figure 3 illustrates an example of the process of multiple timer interrupts technique with employments SD mechanism to fixed period of each task which runs in tick interval. In this figure, to reduce jitter of Task B, the required release time prior to Task B is equal to the worst-case execution time (WCET) of Task A plus scheduler overhead. This technique can save the power using by enter to an idle mode after completion of each task.

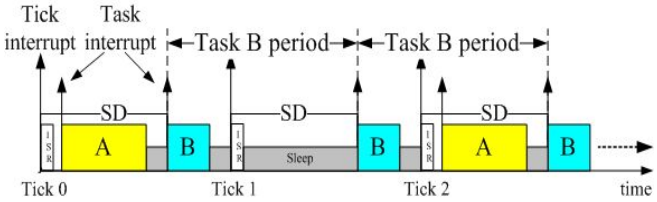

Fig. 3. The operation of TTC scheduler based on MTI technique which employs SD mechanism.

B. TTSA-MTI Scheduling Algorithm

The proposed algorithm which can help to schedule the process of TTC-MTI implementation called "time-triggered scheduling algorithm with multiple timer interrupts" (TTSA-MTI). Figure 4 shows flowchart for testing the schedulability of such algorithm.

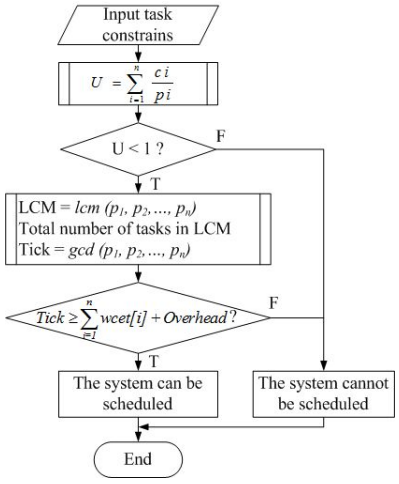

Fig. 4. Flowchart for testing the schedulability of the TTSA-MTI algorithm.

The TTSA-MTI algorithm can be described as follows:

1. Input a list of task specifications and constraints to the algorithm, each task t_i comprises of the following parameters by equation as

$$t_i = \{p_i, c_i, d_i, o_i\}, \tag{1}$$

where 'p_i' is the task period, 'c_i' is the worst-case execution time, 'd_i' is the task's relative deadline, and 'o_i' is the task offset time.

2. The algorithm calculates and tests the CPU utilization of the task set, denoted as '*U*'. The processor utilization factor of a set of *n* periodic task can be given by equation as

$$U = \sum_{i=1}^{n} \frac{ci}{pi}. \tag{2}$$

3. In the utilization's testing, if the utilization is greater than processor capacity, this means that the system cannot be scheduled at all. In contrast, if the utilization is not greater than the processor capacity ($U < 1$), the algorithm will be calculated the length of major cycle (*LCM*), the total number of tasks in major cycle (LCM_{total}), and the

most suitable tick interval (*Tick*) following by equations as:

$$LMC = lcm\{p_1, p_2, ..., p_n\}, \quad (3)$$

$$LCM_{total} = \sum_{i=1}^{n} \frac{LCM}{pi}, \quad (4)$$

$$Tick = gcd\{p_1, p_2, ...p_n\}, \quad (5)$$

where *'lcm'* is the least common multiple and *'gcd'* is the greatest common divisor.

4. The algorithm will test the schedulability by (6), if the tick interval (*Tick*) is greater or equal to the summation of WCET of all tasks and scheduler overhead. This means that all tasks can be scheduled. On the other hand, the system cannot be scheduled

$$Tick \geq \sum_{i=1}^{n} wcet[i] + Overhead. \quad (6)$$

As regards to scheduler overheads, there are many factors which can be lead to a different level of the overhead, such as the scheduler technique, the number of tasks in the system, and the speed of CPU used to implement the system [11]. However, the scheduler overhead of this algorithm comprises of the tick interrupt overhead ($Overhead_{tick}$) and the task interrupt overhead ($Overhead_{task}$). These overheads arise from the time spent in handling the interrupt service routing (ISR) and while updating and checking the schedule of each task following by equation as

$$Overhead = Overhead_{tick} + \sum_{i=1}^{n} Overhead_{task}[i]. \quad (7)$$

C. TTSA-MTI Scheduling Implementation

In the scheduler implementation, the scheduler is calculated the length of major cycle, the total number of tasks in the LCM, and the most suitable tick interval in order to configure the timer of the tick interrupt and prepare to arrange tasks in the LCM. In addition, the scheduler is calculated the required release time of all tasks to avoid the problem of task release jitter by placing SD mechanism around tasks which execute prior to other tasks in the same tick interval. The require release time of each task is equal to the sum of WCET of the preceding tasks plus the scheduler overhead following by equation as

$$t_r(k) = \sum_{i=1}^{k-1} (wcet[i] + Overhead[i]), \quad (8)$$

where t_r is required release time of each task in tick interval.

V. EVALUATING THE TTSA-MTI ALGORITHM

To evaluate the performance of TTSA-MTI scheduler, it presents algorithm and system performance, and also the jitter results by comparing with a traditional scheduler.

A. Algorithm Performance

The algorithm performance was focused to measure the computation time of schedulability test. It then compared between the traditional TTSA and the proposed algorithm.

1) Hardware Platform and Software Development Tools

In this experiment, the target platform is a small microcontroller LPC2129. The LPC2129 is based on a 32 bit ARM7 microcontroller, which is used an oscillator frequency of 12 MHz and a CPU frequency of 60 MHz. The CPU consists of two 32 bit timers with 4 multiple channels in each timer. Accordingly, this CPU has enough timers to implement the TTSA-MTI scheduler. As for the software development, this paper used development tools from Keil products [21]. The tool chain was used RealView MDK version 4.12.

2) Task Specifications

To explore the performance of this algorithm, 100 tasks were randomly generated. By assuming all tasks are period, the deadline is equal to its period, and the random task set can be scheduled at all. In addition, the worst case execution time of all tasks is generated according to the following inequalities:

$$0 < WCET(i) < 1000\ us, \quad (9)$$

$$WCET(i) < P(i) < 10000\ us. \quad (10)$$

3) Results

In the experiment, the computation time of TTSA and TTSA-MTI algorithm can be shown in Fig. 5. The result shows that scheduling time of both algorithms is increased following the number of tasks in the system. Furthermore, TTSA-MTI scheduler is significant reduction in computation time when compare with a traditional approach.

B. System Performance

For meaningful testing the system performance, the experiments were tested and compared the performance of traditional TTC-Dispatch and proposed TTC-MTI scheduler.

1) Measurement approach and methodology

In order to investigate the system performance, measurement were taken for the CPU test and memory requirements. The test was run system approximately 25 seconds, and then measured the scheduler time.

2) System performance results

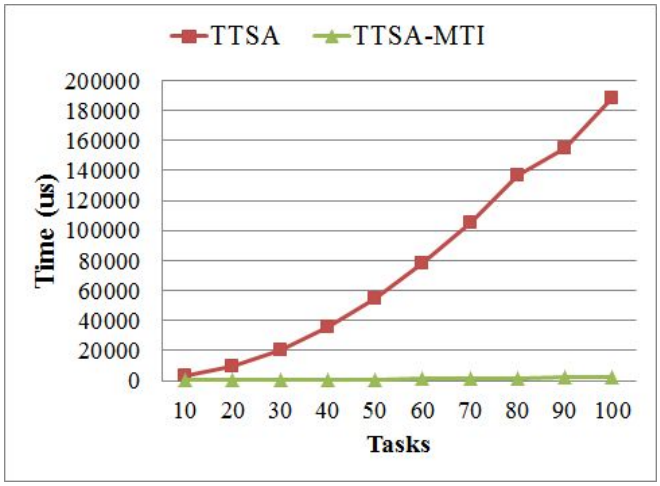

Fig. 5. Computation time of the schedulability analysis between TTSA and TTSA-MTI algorithm.

Table I shows the CPU overhead and memory requirements of TTC-Dispatch and proposed scheduler. The results show that the scheduler overheads of both schedulers were slightly different. However, the memory requirements of proposed scheduler was required more code memory than TTC-Dispatch approximately 2.3 Kbytes.

TABLE I. COMPARISON OF THE SYSTEM PERFORMANCE IN THE TTC-DISPATCH AND TTC-MTI SCHEDULERS.

Scheduler	TTC-Dispatch	TTC-MTI
Scheduler time (s)	7.45	7.40
Total time (s)	25	25
Scheduler overhead (%)	29.80	29.60
ROM (bytes)	11,912	14,272
RAM (bytes)	3,732	3,340

C. Jitter Test

1) Measurement Approach and Methodology

In an ideal timing of the system, when task is ready to execute, it should be started at the same place in each period. If the task execution deviates from its ideal release time, then this time variation is described as release jitter. Fig. 6 shows multiple instances of a periodic task that refers to release jitter. From the figure, task is character by its starting time '*s*', release time '*r*' , and release jitter '*x*'.

Generally, release jitter can be expressed as relative release jitter and as absolute release jitter. Relative release jitter is the maximum deviation of the start time of two consecutive instances [22]. The relative release jitter can be expressed as

$$RRj_i = max_k \left| (s_{i,k} - r_{i,k}) - (s_{i,k-1} - r_{i,k-1}) \right|. \quad (11)$$

As per absolute release jitter, this is the maximum deviation of the start time among all instances [22]. The absolute release jitter can be expressed as

$$ARj_i = max_k (s_{i,k} - r_{i,k}) - min_k (s_{i,k} - r_{i,k}). \quad (12)$$

Another significant measurement is the average jitter, this is represented by the standard deviation in the measure of average periods [12]. In this experiment, relative release jitter and absolute release jitters were used to assess the jitter level and analyse the system reliability of TTC schedulers.

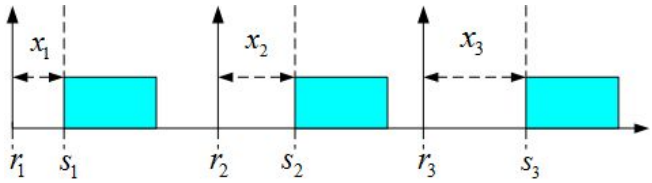

Fig. 6. Release jitter in periodic task.

2) Task Specification

To allow exploring the impact of task release jitter, 200 set of tasks were randomly generated. Figure 7 shows an example of the task set for experiment. Each task set consisted of three tasks, Task A which has the highest priority was scheduled to run every two ticks whereas task B and task C were run every tick interval. Note that task C is assigned as the lowest priority.

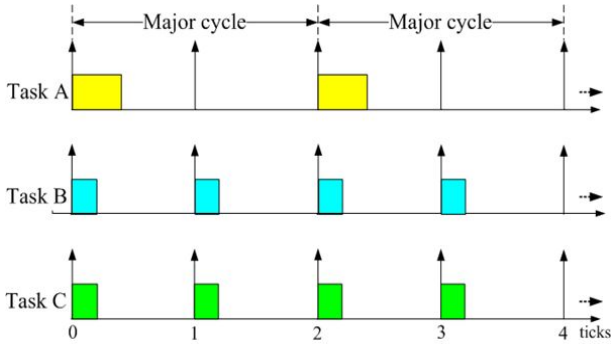

Fig. 7. Graphical representation of the task set in jitter test.

3) Jitter Results

TABLE II. TASK RELEASE JITTER FROM THE TTC-DISPATCH AND TTC-MTI SCHEDULERS.

Scheduler	TTC-Dispatch			TTC-MTI		
Tasks	Task A	Task B	Task C	Task A	Task B	Task C
Min Period (us)	1,999	934	930	1,999	989	970
Max period (us)	2,001	1,057	1,095	2,001	1005	1,015
Average Period (us)	2,000	995.5	1012.5	2,000	997	992.5
Absolute jitter (us)	2	123	165	2	16	42

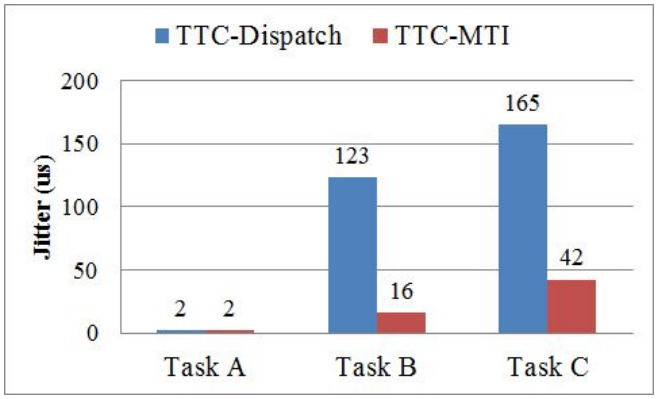

Fig. 8. Comparison of release jitter values in TTC-dispatch and TTC-MTI scheduler.

To explore the jitter level of both algorithms, the measurement between the start time of two consecutive instances of the dummy task were measurement by using LabView 11.0 software and NI USB-6008 hardware. In each experiment, 10,000 consecutive pulse widths were measured to give the results presented in this paper. Table II and corresponding Fig. 8 show the result of release jitter from both schedulers.

VI. CONCLUSIONS

This paper is attended to overcome the problems of the fragility and the impacts of task release jitter in TTC scheduler by employing the scheduler implementation called "TTC-MTI" scheduler. To assure that the developers can select an effective scheduling algorithm, this paper proposed algorithm called "TTSA-MTI" to automate the process of scheduling in such scheduler. The results show that the TTSA-MTI algorithm can help in a significant reduction of computation time and available to overcome the problem of

task release jitter as opposed to a traditional scheduler. Overall, this work concludes that the use of TTSA-MTI algorithm is an effective task scheduling for use with a range of TTC-MTI scheduler. This is also a practical way of implementing TTC architecture with resource-constrained embedded system.

REFERENCES

[1] A. Albert, "Comparison of event-triggered and time-triggered concepts with regard to distributed control systems", in *Proc. of Embedded World*, Nurnberg, Germany, 2004.

[2] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic, 1997.

[3] F. Cottet, J. Delacroix, C. Kaiser, Z. Mammeri, *Scheduling in real-time systems*. John Wiley & Sons Ltd, 2002. [Online]. Available: http://dx.doi.org/10.1002/0470856343

[4] J. W. S. Liu, *Real-time system*. Prentice Hall, 2000.

[5] T. Phatrapornnant, M. J. Pont, "Reducing jitter in embedded systems employing a time-triggered software architecture and dynamic voltage scaling", *IEEE Trans. Computers*, pp. 113–124, 2006. [Online]. Available: http://dx.doi.org/10.1109/TC.2006.29

[6] D. Ayavoo, *et al.*, "Two novel shared-clock scheduling algorithms for use with 'Controller Area Network' and related protocols", *Microprocessors and Microsystems*, vol. 31, pp. 326–334, 2007. [Online]. Available: http://dx.doi.org/10.1016/j.micpro.2006.11.002

[7] M. J. Pont, *Patterns For Time-triggered Embedded Systems: Building Reliable Applications with the 8051 Family of Microcontrollers*, ACM Press Books, 2001.

[8] C. D. Locke, "Software architecture for hard real-time applications: cyclic executives vs. fixed priority executives", *Real-Time Syst.*, vol. 4, pp. 37–53, 1992. [Online]. Available: http://dx.doi.org/10.1007/BF00365463

[9] T. P. Baker, A. Shaw, "The cyclic executive model and Ada", in *Proc. Real-Time Systems Symposium*, 1988, pp. 120–129. [Online]. Available: http://dx.doi.org/10.1109/REAL.1988.51108

[10] T. Nghiem, *et al.*, "Time-triggered implementations of dynamic controllers", *ACM Trans. Embed. Comput. Syst.*, vol. 11, pp. 1–24, 2012. [Online]. Available: http://dx.doi.org/10.1145/2331147.2331168

[11] A. K. Gendy, M. J. Pont, "Automatically configuring time-triggered schedulers for use with resource-constrained, single-processor embedded systems", *IEEE Trans. Industrial Informatics*, vol. 4, no. 1, pp. 37–46, 2008. [Online]. Available: http://dx.doi.org/10.1109/TII.2008.916053

[12] M. Nahas, "Employing two 'sandwich delay' mechanisms to enhance predictability of embedded systems which use time-triggered co-operative architectures", *Journal of Software Engineering and Applications*, vol. 4, pp. 417–425, 2011. [Online]. Available: http://dx.doi.org/10.4236/jsea.2011.47048

[13] M. Park, Y. Cho, "Feasibility analysis of hard real-time periodic tasks", *Journal of Systems and Software*, vol. 73, pp. 89–100, 2004. [Online]. Available: http://dx.doi.org/10.1016/S0164-1212(03)00236-X

[14] C. L. Liu, J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment", *J. ACM*, vol. 20, pp. 46–61, 1973. [Online]. Available: http://dx.doi.org/10.1145/321738.321743

[15] J. Y. T. Leung, J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks", *Performance Evaluation*, vol. 2, pp. 237–250, 1982. [Online]. Available: http://dx.doi.org/10.1016/0166-5316(82)90024-4

[16] A. M. K. Cheng, *Real- time systems, scheduling, analysis, and verifications*. Willey-Interscience, 2002.

[17] Q. Li, C. Yao, *Real Time Concepts for Embedded Systems*. USA: CMP Books, 2003.

[18] M. J. Pont, *Embedded C*. Pearson Education, 2002.

[19] Z. M. Hughes, M. J. Pont, "Reducing the impact of task overruns in resource-constrained embedded systems in which a time-triggered software architecture is employed", in *Trans. Institute of Measurement and Control*, vol. 30, pp. 427–450, 2008. [Online]. Available: http://dx.doi.org/10.1177/0142331207086183

[20] G. Buttazzo, "Rate monotonic vs. EDF: judgment day", *Real-Time Systems*, vol. 29, pp. 5–26, 2005. [Online]. Available: http://dx.doi.org/10.1023/B:TIME.0000048932.30002.d9

[21] *Embedded Development Tools*, Keil, 2012. [Online]. Available: www.keil.com

[22] G. C. Buttazzo, *Hard real time computing systems predictable scheduling algorithms and applications*. Springer, 2011. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-0676-1