Scheduling of Dependent Real-Time Tasks Using Fuzzy Logic Technique

Medhat Awadalla[1], Afaq Ahmad[1], Samir Al-Busaidi[1]

[1]Department of Electrical and Computer Engineering, Sultan Qaboos University
PO Box 33, Zip Code 123, Oman
medhatha@squ.edu.om

Abstract. Many scheduling algorithms have been studied to guarantee the time constraints of real-time processes. Scheduling decision of these algorithms is usually based on parameters which are assumed to be crisp. However, in many circumstances the values of these parameters are vague. Moreover, reducing energy consumption is a critical issue in the design of battery-powered real time systems to prolong battery life. With dynamic voltage scaling (DVS) processors, energy consumption can be reduced efficiently by making appropriate decisions on the processor speed/voltage during the scheduling of real time tasks. Therefore, a fuzzy logic approach is proposed to reduce energy consumption by determining the appropriate supply-voltage/speed of the processor provided that timing constraints are guaranteed. Intensive simulated experiments and qualitative comparisons with the most related literature have been conducted in the context of dependent real-time tasks. Experimental results have shown that the proposed fuzzy scheduler saves more energy and creates feasible schedules for real time tasks. It also considers tasks priorities which cause higher system utilization and lower deadline miss time.
Keywords: dynamic voltage scaling (DVS) processors, fuzzy logic approach, real-time tasks.

1. INTRODUCTION

Many applications namely avionics, traffic control, automated factory, and military systems require real time communication and computation. In real-time systems, all tasks have specific parameters such as deadline, priority, etc. Many real-time systems are hard and missing deadline is catastrophic, Chen and Kuo (2007). A schedule which is executing all real-time tasks within their deadlines and all the other constraints are met, is called a feasible schedule. Tasks are classified as periodic and non-periodic (Behera et al., 2012). The execution requests of a periodic task repeatedly occur at regular intervals. On the contrary, execution requests of a non-periodic task are unpredictable. In all cases, time has an essential role and having the right answer too late is as bad as not having it at all. In the literature, these systems have been defined as: "systems in which the correctness of the system depends not only on the logical results of computation, but also the time at which the results are produced". Such systems must react to the requests within a fixed amount of time which is called deadline. Scheduling algorithms of these systems may be considered one of the key components of a real-time system, which can either enable the system to thrive or bring it to its knees. Strict timing requirements must often be met within highly dynamic environments which do not lend themselves well to static scheduling algorithms. The level of uncertainty in dynamic, real-time environments is such as to require significant flexibility and adaptivity from real systems. Fuzzy logic contributes in this issue in the form of approximate reasoning, where it provides decision-support and expert systems with powerful reasoning capabilities bounded by a minimum number of rules. Theoretically, fuzzy logic is a method for representing analog processes, or natural phenomena that are difficult to model mathematically on a digital computer. Therefore, Fuzzy systems fit as scheduling algorithm building into the real-time system flexibility and adaptation to the uncertainty inherent in real-time environments and offers a means to improve several important characteristics of real-time systems (Venkatachalam and Franz (2005). Since most of real time systems (devices) are battery powered. As the applications on these devices are being complicated, the energy consumption is also effectively increasing. So, minimizing energy consumption is a critical issue in the design of these systems, and techniques that reduce energy consumption have been studied at different levels in details Liu (2007).

One promising mechanism that provides the best of both low-power and high-performance processors in the same system is DVS. Dynamic voltage scaling (DVS) is a technique that varies the supply voltage and clock frequency (speed) based on the computation load to provide desired performance with the minimal amount of energy consumption in ubiquitous embedded systems. DVS relies on special hardware, in particular, a programmable DC switching voltage regulator, a programmable clock generator, and a high-performance processor with wide operating ranges, to provide this best-of-both-worlds capability. In order to meet peak computational loads, the processor is operated at its normal voltage and frequency (which is also its maximum frequency). When the load is lower, the operating frequency is reduced to meet the computational requirements. In CMOS technology, used in virtually all microprocessors today, the maximum operating frequency increases (within certain limits) with increased operating voltage, so when the processor is run slower, a reduced operating voltage suffices (Khaniet al., 2012). A second important characteristic is that the energy consumed by the processor per clock cycle scales quadratically with the operating voltage ($E \alpha V^2$), so even a small change in voltage, V, can have a significant impact on energy consumption, E. By dynamically scaling both voltage and frequency of the processor based on computation load, DVS can provide the performance to meet peak computational demands, while on average, providing the reduced power consumption (including energy per unit computation) benefits typically available on low performance processors.

In this paper, our goal is to develop a fuzzy logic approach to reduce energy consumption by determining the appropriate supply-voltage/speed of the processor provided that timing constraints are guaranteed. Fuzzy logic approach is proposed because in a dynamic hard real-time system, not all the characteristics of tasks (e.g., precedence constraints, resource requirements, etc.) are known a priori. For example, the arrival time for the next task is unknown for aperiodic tasks. To be more precise, there is an inherit uncertainty in hard real-time environment which will worsen scheduling problems (e.g. arbitrary arrival time, uncertain computation time and deadline). Characteristics of a task that may be uncertain include expected next arrival time, criticality, or importance of the task, system load and/or predicted load of individual processors, and run time, or more specifically average vs. worst-case run time. Therefore, our goal is to develop an approach for hard real-time scheduling that can be applied to a dynamic environment involving a certain degree of uncertainty. In this paper, we focus on a hard real time system on a preemptable uniprocessor system with a set of dependent tasks. These tasks will be characterized by worst-case computation time, blocking time ,task deadline and arriving time.

The rest of the paper is organized as follows: section 2 outlines the related work to the theme of this paper. Section 3 demonstrates the multi-speed algorithm. Section 4 gives an overview about fuzzy inference system. Section 5 shows the proposed fuzzy system. Section 6 presents experiments and discussions. Section 7 concludes the paper.

2. RELATED WORK

Many researchers have tried to implement fuzzy logic to schedule the processes. There are four main approaches reported in the literature for the fuzzy scheduling problems; fuzzifying directly the classical dispatching rules, fuzzy ranking, fuzzy dominance relation methods, and solving mathematical models to determine the optimal schedules by heuristic approximation methods (Pandey and Sharma, 2011). Round robin scheduling using neuro fuzzy approach and Soft real-time fuzzy task scheduling for multiprocessor systems have been developed (Hamzeh et al., 2007) Fuzzy Better Job First (FBJF) scheduling algorithm logically integrates parameters and uses fuzzy ranking approach to determine the next most worthy job to be executed has been proposed (Pandey and Sharma, 2011). A fuzzy scheduling approach to arrange real-time periodic and non-periodic tasks with reference to optimal utilization of distributed processors has been proposed (Alam et al., 2011). In their paper, an attempt is made to apply fuzzy logic in the design and implementation of a modified scheduling algorithm to overcome the shortcoming of well-known scheduling algorithms. Furthermore, many dynamic and static scheduling algorithms based on fuzzy logic approach have been proposed and applied on uniprocessor systems. Also multiprocessor and distributed systems have been considered (Kadhim and Al-Aubidy 2010).

Regarding the energy efficient scheduling, Jejurikar and Gupta (2006) are considered the pioneers in that field where they expected the DVS technique, then they have proposed an optimal static (offline) scheduling algorithm by considering a set of aperiodic jobs on an ideal processor. However, the problem of DVS with dependent tasks because of shared resources has been first addressed, Pillai and Shin (2007). Jejurikar and Gupta have proposed two algorithms for scheduling fixed priority, Rate Monotonic (RM) scheduler, tasks using priority ceiling protocol (PCP) described in (Shin and Kim, 2004) as resource access protocol. They have computed static slowdown factors which guarantee that all tasks will meet their deadlines taking into account the blocking time caused by the task synchronization to access shared resources. In their first algorithm, critical section maximum speed (CSMS), they have let the critical sections (sections deal with shared resources) to be executed at maximum processor speed and they have computed slowdown factors for executing non critical sections. The second

algorithm, constant static slowdown (CSS), computes a uniform slowdown factor for all tasks and for all sections (critical and non-critical) saving speed switches occurred in the first algorithm (CSMS).

Zhang and Chanson (2004) have then extended the algorithms CSMS and CSS to handle dynamic priority. The dynamic priority ceiling protocol is an extension of original priority ceiling protocol to deal with dynamic priority tasks (EDF scheduling). Jejurikar and Gupta (2004) have also proposed a generic algorithm that works with both EDF and RM schedulers, and they have introduced the concept of frequency inheritance in their algorithm. They have worked on the same problem (scheduling of dependent tasks) and proposed three algorithms for energy efficient scheduling of dependent tasks with shared resources over EDF scheduler, where they have used stack resource policy (SRP) as resource access protocol. The SRP can handle static and dynamic priority tasks (EDF and RM schedulers), reduces context switches over PCPs, and it is easy to implement. The first algorithm is the same as CSS for EDF scheduler because they have derived the static slowdown factor directly from the EDF schedulability test with blocking time in (Lee, 2007):

$$\forall i, 1 \le i \le n, \quad \frac{B_i}{D_i} + \sum_{k=1}^{i} \frac{C_k}{D_k} \le 1 \tag{1}$$

where C is the computation time (worst case execution time WCET), D is the task relative deadline, n is the number of tasks, and B is the blocking time that can be defined as the maximum time through which a high priority task can be blocked by a low priority task due to exclusive access to shared resource (index i refers to the blocked high priority task). The second algorithm is the dual speed (DS) algorithm. The main concept of this algorithm is using two speeds (L, H) and switching between them. Initially the algorithm operates with the low speed L, and switches to the high speed H as soon as a blocking occurs. The last algorithm is the dual speed dynamic (online) reclaiming algorithm which dynamically collects the residue time from early completed jobs and redistributes it to the other pending jobs to further reduce the processor speed and achieve more energy saving. An improvement multi-speed (IMS) algorithm that further reduces the energy dissipation by considering only remaining blocking time to compute a lower speed has been developed (Awadalla, 2009).

3. SYSTEM MODEL

3.1 Task Model

In this paper, for simplicity, real-time periodic tasks are considered. Each task τ is characterized by the following parameters:

- The release time (r): the time when the task first released.
- The period (T): the constant interval between jobs.
- The relative deadline (D): the maximum acceptable delay for task processing.
- The computation time (C): the worst case execution time (WCET) of any job.
- The blocking time (B): the maximum time a task can be blocked by another lower priority task.

In this paper we consider well formed tasks that satisfy the condition: $0 \le C \le D \le T$.

A 3-tuple $\tau = \{C, D, T\}$ represents each task, the relative deadline is assumed to be the same as the period in all illustrative examples.

3.2 Processor Model

The tasks are scheduled on a single DVS processor that supports variable frequency (speed) and voltage levels continuously, i.e. DVS processors can operate at any speed/voltage in its range (ideal). Of course, practical DVS processors supports discrete speed/voltage levels (non ideal). So, the desired speed/voltage of the ideal DVS processor is rounded to the nearest higher speed/voltage level the practical DVS processor supports. The time (energy) required to change the processor speed is very small compared to that required to complete a task. It is assumed that the voltage change overhead, similar to the context switch overhead, is incorporated in the task computation time. In this paper, it is assumed that the processor's maximum speed is 1 and all other speeds are normalized with respect to the maximum speed.

3.3 Power Model

The power consumption has two essential components: dynamic and static power. The dynamic power consumption, which is the main component, has a quadratic dependency on supply voltage (Liu, 2007) and can be represented as:

$$P_{dynamic} = C_{ef} \cdot Vdd^2 \cdot F \tag{2}$$

Where C_{ef} is the switched capacitance, V_{dd} is the supply voltage, and F is the processor clock frequency (sometimes referred as speed S) which can be expressed in terms of supply voltage V_{dd} and threshold voltage V_t as following:

$$F = k \,.\, (V_{dd} - V_t)^2 / V_{dd} \tag{3}$$

The static power consumption is primarily occurred due to leakage current (I_{leak}), and the static (leakage) power (P_{leak}) can be expressed as:

$$P_{leak} = I_{leak} \;.\; V_{dd} \tag{4}$$

Static energy dissipation is a result of leakage current due to the finite-resistance of the off transistors between power and ground that exist whenever power is applied to a CMOS circuit. The magnitude of the leakage current is highly dependent on the applied and threshold voltages characteristics. As integrated circuit technology scales to ever smaller dimensions, supply voltage levels are likewise scaled. To improve circuit speed, the threshold voltages are also decreased. This decrease in threshold voltage results in an exponential increase in the sub-threshold leakage current (Venkatachalam and Franz 2005).

When the processor is idle, a major portion of the power consumption comes from the leakage. Currently leakage power is rapidly becoming the dominant source of power consumption in circuits and persists whether a computer is active or idle, and much work has been done to address this problem.

So in this paper, lowering supply voltage is one of the most effective ways to reduce both dynamic and leakage power consumption. As a result, it reduces energy consumption where the energy consumption is the power dissipated over time:

$$\text{Energy} = \int \text{Power}\, dt \tag{5}$$

However, DVS aims at reducing energy consumption by reducing the supply-voltage/speed of the processor provided that timing constraints are guaranteed. In other words, DVS makes use of the fact that there is no benefit of finishing a real time job earlier than its deadline.

DVS processors have two types: ideal and non-ideal. An ideal processor can operate at any speed in the range between its minimum available speed and maximum available speed. A non-ideal processor has only discrete speeds with negligible or non-negligible speed transition overheads. Another classification defines four different types of DVS systems: ideal, feasible, practical, and multiple (Liu, 2007).

4. MULTI-SPEED ALGORITHM

Multi-speed (MS) algorithm proposed by Lee (2007) is a blocking aware scheduling algorithm with non-preemptible critical sections using SRP as resource access protocol.

The MS algorithm can be considered as an extension of dual speed (DS) algorithm, where the difference between the two algorithms is that MS algorithm uses many speeds (low speed SL and multiple high speeds S_m where $1 \le m \le n$) instead of two speeds (low speed L and one high speed H) in DS algorithm. Like DS algorithm, MS algorithm initially starts with the low speed S_L then it switches to one of high speeds as soon as a blocking occurs. The high speed to which the MS algorithm switches is determined according to the blocking task, i.e. each blocking task τ_m has its own high speed S_m, and according to the blocking task, the algorithm switches to the convenient high speed. The low speed SL, which is exactly the same as low speed L in DS algorithm, is the optimal lowest speed with which all tasks can be scheduled without missing any deadline, and it is derived from the plain EDF schedulability test without shared resources:

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \le S_L \tag{6}$$

The high speed S_m for a blocking task τ_m is derived as in DS algorithm from the EDF schedulability test with shared resources and SRP protocol:

$$\forall k, 1 \le k < m, \sum_{i=1}^{k} \left(\frac{C_i}{D_i} \right) + \frac{B_m}{D_k} \le S_m \tag{7}$$

Where the blocking time B_m here is the maximum time (length of critical section of τ_m) through which a low priority task τ_m can block another high priority task due to exclusive access to shared resource and unlike mentioned before, index m refers to the blocking task (low priority task).

The above mentioned speeds S_L and S_m have to satisfy the condition:

$$S_L \leq S_m \leq 1 \tag{8}$$

MS algorithm ends the high speed interval when the deadline of the blocking task is reached or the processor becomes idle. In some real time tasks, MS improves the energy consumption however in others tasks, the timing constraints are not guaranteed.

5. FUZZY INFERENCE SYSTEMS

A fuzzy inference system (FIS) tries to derive answers from a knowledge base by using a fuzzy inference engine. The inference engine which is considered to be the brain of the expert systems provides the methodologies for reasoning around the information in the knowledgebase and formulating the results. Fuzzy logic is an extension of Boolean logic dealing with the concept of partial truth that denotes the extent to which a proposition is true. Whereas classical logic holds that everything can be expressed in binary terms (0 or 1, black or white, yes or no), fuzzy logic replaces Boolean truth values with the degree of truth. Degree of truth is often employed to capture the imprecise modes of reasoning that play an essential role in the human ability to make decisions in an environment of uncertainty and imprecision. The membership function of a fuzzy set corresponds to the indicator function of the classical sets. It can be expressed in the form of a curve that defines how each point in the input space is mapped to a membership value or a degree of truth between 0 and 1. The most common shape of a membership function is triangular, although trapezoidal and bell curves are also used. The input space is sometimes referred to as the universe of discourse (Pandey and Sharma, 2011). Fuzzy Inference Systems are conceptually very simple. An FIS consists of an input stage, a processing stage, and an output stage. The input stage maps the inputs, such as deadline, execution time, and so on, to the appropriate membership functions and truth values. The processing stage invokes each appropriate rule and generates a result for each. It then combines the results of the rules. Finally, the output stage converts the combined result back into a specific output value. As discussed earlier, the processing stage, which is called the inference engine, is based on a collection of logic rules in the form of IF-THEN statements, where the IF part is called the "antecedent" and the THEN part is called the "consequent". Typical fuzzy inference subsystems have dozens of rules. These rules are stored in a knowledgebase. An example of fuzzy IF THEN rules is: IF deadline is early then priority is high, in which deadline and priority are linguistics variables and early and high are linguistics terms. The five steps toward a fuzzy inference are as follows:

- fuzzifying inputs
- applying fuzzy operators
- applying implication methods
- aggregating outputs
- defuzzifying results

Below is a quick review of these steps. However, a detailed study is not in the scope of this paper. Fuzzifying the inputs is the act of determining the degree to which they belong to each of the appropriate fuzzy sets via membership functions. Once the inputs have been fuzzified, the degree to which each part of the antecedent has been satisfied for each rule is known. If the antecedent of a given rule has more than one part, the fuzzy operator is applied to obtain one value that represents the result of the antecedent for that rule. The implication function then modifies that output fuzzy set to the degree specified by the antecedent. Since decisions are based on the testing of all of the rules in the Fuzzy Inference Subsystem (FIS), the results from each rule must be combined in order to make the final decision. Aggregation is the process by which the fuzzy sets that represent the outputs of each rule are processes into a single fuzzy set. The input for the defuzzification process is the aggregated output fuzzy set and the output is then a single crisp value. This procedure can be summarized as follows: mapping input characteristics to input membership functions, input membership function to rules, rules to a set of output characteristics, output characteristics to output membership functions, and the output membership function to a single crisp valued output. There are two common inference methods (Pandey and Sharma 2011). The first one is called Mamdani's fuzzy inference method proposed by Sheo et al., (2012) and the second one is Takagi-Sugeno-Kang, or simply Sugeno, method of fuzzy inference introduced in Kadhim Al-Aubidy (2010). These two methods are the same in many respects, such as the procedure of fuzzifying the inputs and fuzzy operators. The main difference between Mamdani and Sugeno is that the Sugeno's output membership functions are either linear or constant but Mamdani's inference expects the output membership functions to be fuzzy sets.

6. THE PROPOSED MODEL

In the proposed model, the input stage consists of four input variables i.e. worst execution time, deadline, blocking time and arriving time as shown in Fig. 1. Worst execution time is the actual amount of time a task requires on CPU to get executed, blocking time is how much time a task can wait before getting a chance to get executed, Deadline represents the final time limit before that a task has to get terminated whereas the arriving time is the time at which the job is ready to be assigned to the processor. The combination of four input parameters decides the job priority and the appropriate processor speed to execute it.

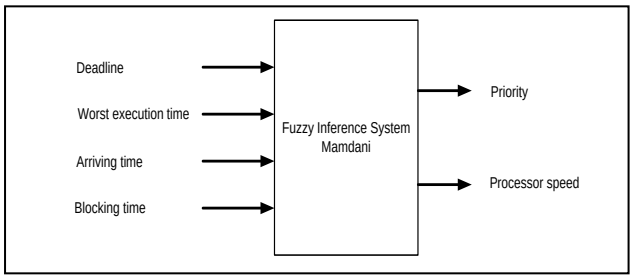

Fig.1. Inference system block diagram

Membership functions describe the degree to which each input parameter represents its association. Linguistic variables are assigned to each input parameter, to represent this association. Worst execution time is categorized as Low, Medium and High. Similarly blocking time is defined in the same way. However, Deadline and arriving time are defined as early, medium and late. The output parameters, job priority and processor speed are defined as low, medium and high as depicted in Fig. 2. Due to the overlapping among the membership functions, some values of inputs or outputs will be classified for more than one membership function.

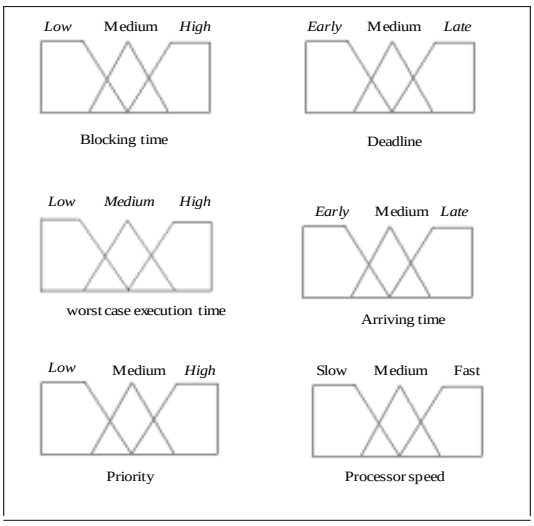

Fig. 2: Membership functions of the system

Fuzzy rules try to combine these parameters as they are connected in real worlds. Based on the different situations, more than one fuzzy rule can be fired at the same time and the role of fuzzy inference system is to provide the proper consequent.

In fuzzy inference systems, the number of rules has a direct effect on its time complexity. Therefore, having fewer rules may result in a better system performance.

In our proposed approach, a newly arrived task will be added to the input of job queue. This queue has the remaining tasks from last cycle that has not yet been assigned. The following algorithm will be executed:

Loop

1. For each ready task, feed the deadline, execution time, blocking time and arriving time into the inference engine. Consider the output of inference module as priority of the task and the processor speed.
2. Execute the task with highest priority with the decided speed unless it is blocked by lower priority job until a scheduling event occurs (a running task finishes, a new task arrives).
3. Update the system states.

End Loop

We chose to treat deadline time as the most important principles behind choosing a task for scheduling because the major purpose of hard real-time scheduling is to meet the deadline. After this, worst execution time and then earliest arriving time. However if the lowest priority job is only available job, it will be assigned to the processor till another higher priority job arrives. Since the paper has another objective which is to reduce the power consumption, after deciding which job will be assigned to the processor, the system will decide at which speed the processor should perform. The processor speed is mainly affected by the blocking time of the lower priority tasks.

7. EXPERIMENTS AND DISCUSSION

To illustrate the fuzzy approach and the contribution of this paper. Examples implemented in (Awadalla, 2009) are repeated for the sake of qualitative comparison and other tasks have been performed to address the generalization of applying fuzzy logic as a scheduler approach and its capability to minimize the energy consumption of powered real time systems. The first hard real time system with three tasks is considered as following:

$$\tau1=\{1 ,4 ,4 \}, \tau2=\{1.5 ,12 ,12 \}, \tau3=\{3 ,24 ,24 \}$$

The arrival times and critical sections of the three tasks within the least common multiple (LCM) of periods are shown in figure 3(a).

According to the developed inference system, $\tau1$ has the highest priority, followed by $\tau2$ and lastly $\tau3$. The resultant low speed SL is equal 0.5, which is the same as calculated based on equation 6 that represents the processor utilization factor $U=\sum C/T$. There are two blocking tasks in this example: $\tau2$ that can block higher priority task $\tau1$ for maximum time B2=1.5 and $\tau3$ that can block higher priority tasks $\tau1$ and $\tau2$ for maximum time B3=3. So, there will be two high speeds (S2, S3) according to these two blocking tasks.

Based on the memberships in table 1, the worst case execution time of $\tau1$ could be considered as low or medium, arriving time is early, the blocking time is low and the priority is high. According to the rules in table 2, rule 1 will be fired and the crisp output which represents the low speed is the center of area under the curve, so SL is 0.5. By the same procedure for $\tau2$ and $\tau3$, both S2 and S3 are 0.6 and 0.92 respectively. The two speeds (S2, S3) also satisfy the condition (7), where $0.5 \leq S2 \leq 1$ and $0.5 \leq S3 \leq 1$.

The rectangles represent the processing of tasks (jobs) by CPU where the vertical dimension represents the processor speed, and the horizontal dimension represents the execution time elapsed for processing tasks according to their WCETs and the processor speed.

It is clearly noted that the area of the rectangles of the jobs of the same task is the same due to that a task always takes the same number of execution cycles which equals to processor speed multiplied by elapsed time.

Back to Figure 3(a), it is assumed that $\tau3$ is released before $\tau1$ with enough time (ε) to lock the shared resource. When task $\tau1$ is released, it will be blocked by the lower priority task $\tau3$ due to exclusive access to shared resource (according to SRP, a task may be blocked when it is released, and as soon as it starts, it cannot be blocked). So, the processor will start executing $\tau3$ with high speed S3. At time t=4, when the second job of task $\tau1$ is released, it is also blocked by the lower priority task $\tau2$ released before it with enough time (ε) to lock the shared resource.

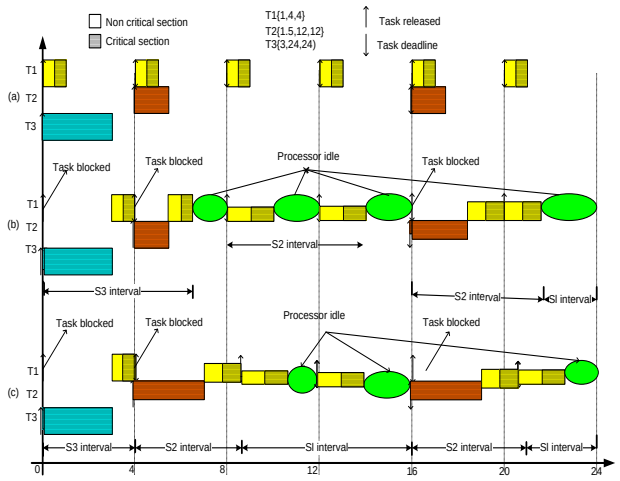

Fig. 3. (a) Task set description: arrival times, computation times, and critical sections.
(b) MS algorithm. (c) Fuzzy logic.

MS algorithm which operates with high speed S3 switches to the maximum of the two high speeds (S3,S2) which is S3, and MS algorithm ends this high speed interval and switches to the low speed SL at time t=6.5 when the processor becomes idle.

At time t=16, when the fifth job of task τ1 is released, it will be blocked by the second job of task τ2, and MS algorithm switches to the high speed S2 and ends this high speed interval when the processor becomes idle.

The same scenario is happened in the case of the proposed fuzzy logic approach, as shown in Figure 3(c), it starts with high speed S3, however it switches to S2 as long as τ2 showed up. Fuzzy logic ends this high speed interval S2 and switches to the low speed SL at time t=9 when τ1 is released.. The processor idle time is reduced from 33% in MS to 23% in fuzzy logic approach which reflects the improvements achieved in the system performance in addition to there is no deadline miss, furthermore the interval of S3 is reduced which in turn reduce the power consumption.

This relative performance improvement in terms of reducing static power consumption is calculated based on percentage ratio of the portions of idle times in both MS and fuzzy logic approach. Where the total execution time, 24 time units, is equal in both techniques MS and fuzzy logic, and idle time in MS approach was 7.92 and 5.52 for fuzzy logic approach.

Another hard real time system with three tasks is addressed:

$$\tau1 = \{2, 5, 5\}, \tau2 = \{2.5, 10, 10\}, \tau3 = \{4, 40, 40\}$$

Again the arrival times and critical sections of the three tasks within the least common multiple (LCM) of periods are shown in Figure 4 (a). In this example, τ1 has the highest priority, τ2 is middle and then τ3 is the lowest one. The resultant low speed SL is equal 0.7, which is less than the low speed calculated based on equation 6. There are two blocking tasks in this example: τ2 that can block higher priority task τ1 for maximum time B2=2, and τ3 that can block higher priority tasks τ1 and τ2 for maximum time B3=3. So, there will be two high speeds (S2, S3). S2= 0.8, and S3=1. The two speeds (S2, S3) also satisfy the condition (7), where $0.7 \leqslant S2 \leqslant 1$ and $0.7 \leqslant S3 \leqslant 1$.

Figure 4(b) shows MS algorithm which ends the high speed interval when processor becomes idle (at times t=10.75, t=19.75, t=29.75, and t=39.75) or the deadline of blocking task is reached, while Figure 4(c) shows the fuzzy logic approach which ends the high speed interval when the blocked task deadline is reached (at time t=6), the processor becomes idle, or a lower or equal priority is selected to run (at times t=10, t=19.125, t=29.125, and t=39.125).

Even though the high speeds and low speed are close to each other in this example, there is again an improvement in the system performance based on fuzzy logic approach compared with MS where the processor idle time is reduced from 10% to 5.8 %.

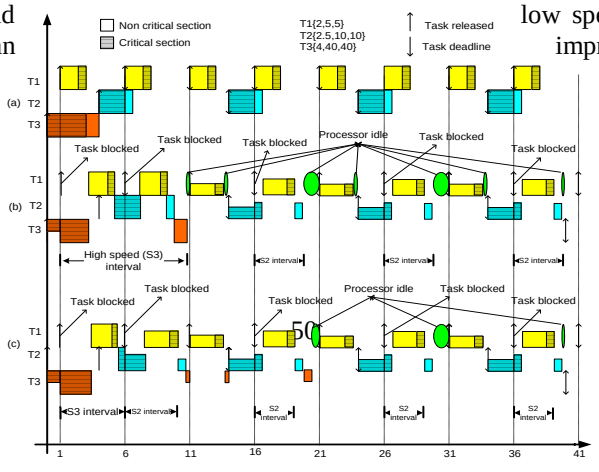

Fig. 4. (a) Task set description: arrival times, computation times, and critical sections.
(b) MS algorithm. (c) Fuzzy logic.

Another more example, hard real time system with the following three tasks is implemented:

$\tau1=\{1, 4, 4\}$, $\tau2=\{2, 8, 8\}$, $\tau3=\{3, 10, 10\}$

Again the arrival times and critical sections of the three tasks within the least common multiple (LCM) of periods are shown in Figure 5 (a). Again $\tau1$ has the highest priority, $\tau2$ is middle and then $\tau3$ is the lowest one. The resultant low speed SL is equal 0.8, and s2= 0.825 and s3=0.875 There is again an improvement in the system performance based on fuzzy logic compared with MS algorithm where the processor idle time is reduced from 4.28% to 2.6%.

For the sake of comparison, the most related approaches, IMS approach developed by us (Awadalla, 2009) and both MS and CSS (Liu, 2007), to the theme of this paper are addressed. Power consumption is computed based on the intervals of idle time, the more idle time, the more power consumption. It can be approximately calculated as a function of the processor speed multiplied with the execution time elapsed for processing tasks. As shown in fig. 3, The power consumption in MS= 0.92x6+0.5x2+0.6x6+0.5x2 + 0.6 x 6 +0.6 x 2 =15.2, while in case of fuzzy logic approach, the power consumption = 0.92x4+0.6x5+0.5x7+0.6x5+0.6x3=13.48. Based on the archived results in (Awadalla, 2009), the power consumption in case of IMS and CSS are 14.2 and 16.7 respectively. Table 1 summarizes the achieved average power consumption based on idle time intervals, while table 2 summarizes and total power consumption of being either executing the tasks or idle.

Table 1: Idle time based average power consumption in the proposed approach, MS, IMS and CSS.

Approach / Task	Task1	Task2	Task3
Fuzzy logic	23%	5.8%	2.6%
IMS	28.6%	7.2%	3.8%
MS	33%	10%	4.28%
CSS	37%	12%	6.3%

Table 2: Total power consumption in the proposed approach, MS, IMS and CSS.

Approach / Task	Task1	Task2	Task3
Fuzzy logic	13.48	34.8	32.73
IMS	14.2	35.2	32.97
MS	15.2	35.6	33.1
CSS	16.7	36.4	33.64

Referring to Figures (3-5), reducing the time during which the processor is idle comes from lowering the processor speed for longer time intervals. This, in turn, reduces the energy consumption dramatically due to quadratic dependency between power and processor speed. To verify that, a comparison study has been performed by computing the energy consumed in CSS, MS, and IMS using the simplified power model where the blocking time B changes from 0 to the highest amount at which the task set is schedulable (when S_m=1). Of course, the high speed S_m changes from S_m=S_L (when B=0) to S_m=1, while the low speed S_L does not change. As it is clear from Figure 6, fuzzy logic approach is the most energy efficient algorithm especially with high blocking times, where the difference between the low and high speeds (S_m, S_L) increases significantly.

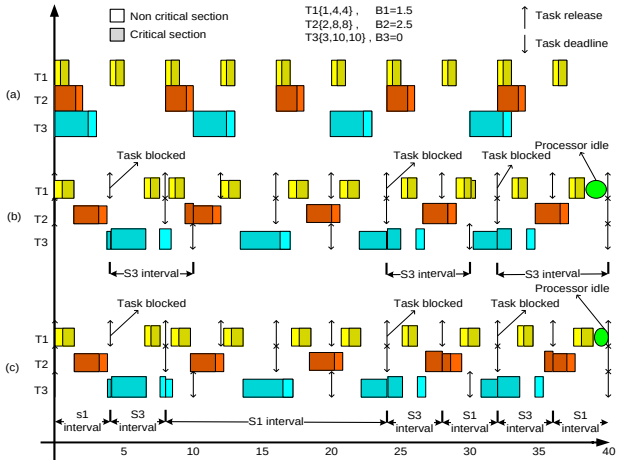

Fig. 5. (a) Task set description: arrival times, computation times, and critical sections. (b) MS algorithm. (c) Fuzzy logic.

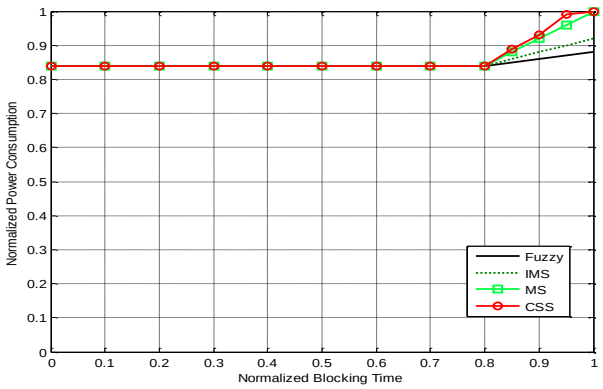

Fig. 6: Power Consumption Versus Blocking Time Changes In Example1

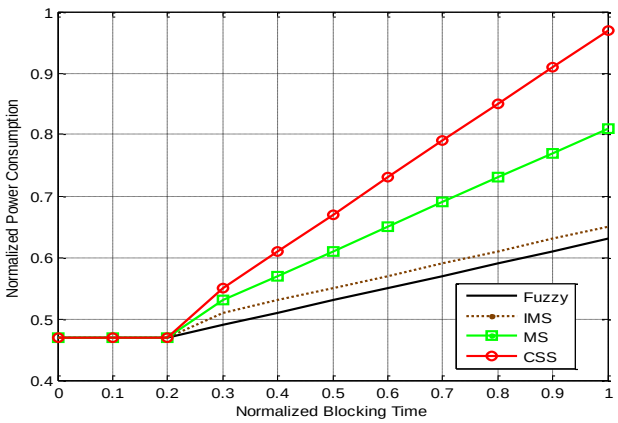

Fig. 7: Power Consumption Versus Blocking Time Changes in Example2

The comparison is repeated for the second and third examples, it is noticed that, as shown in Figure 7 and figure 8, fuzzy logic approach exhibits a slight improvement over IMS with the highest blocking time due to the small difference between high and low speeds (S_m, S_L).

As a result, when the blocking time is low (the high speed is almost the same as the low speed), the three algorithms exhibit the same performance. When the blocking time increases (the difference between the high and low speeds also increases), fuzzy logic approach behaves better than the other two algorithms (CSS and MS) especially when this difference is significant.

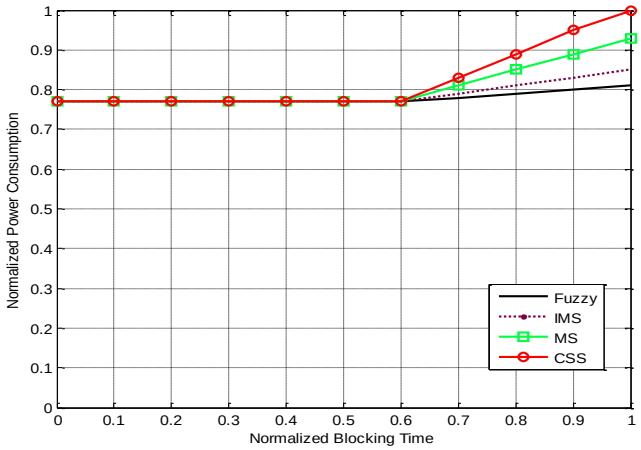

Fig. 8: Power Consumption Versus Blocking Time Changes in Example3

8. CONCLUSION

The paper has addressed the problem of real time scheduling of dependent tasks due to exclusive access shared resources taking into account the reducing of energy consumption as a main goal. The paper has proposed fuzzy logic approach to perform multi-speed (MS) scheduling algorithm, where the proposed algorithm has shown more energy saving than traditional MS algorithm and other related approaches in terms of both static and dynamic power consumption. In this paper, the fuzzy logic approach is developed to reduce energy consumption by providing the appropriate supply-voltage/speed of the processor provided that timing constraints are guaranteed.

REFERENCES

Alam B., Doja M.N., Biswas R., and Alam M. (2011). Fuzzy Priority CPU Scheduling Algorithm. IJCSI International Journal of Computer Science Issues8(6), No 1, 1694-0814.

Awadalla, M.H. (2009). Energy Efficient Real Time Scheduling of Dependent Tasks. International Journal of Computers, Systems and Signals, 10(2).

Behera H. S., Pattanayak R. and Mallick P. (2012). An Improved Fuzzy-Based CPU Scheduling (IFCS) Algorithm for Real Time Systems. International Journal of Soft Computing and Engineering (IJSCE), 2(1), 2231-2307.

Chen J. and Kuo C. (2007). Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms. 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp. 28-38, 2007.

D. Shin and J. Kim. "Dynamic voltage scaling of periodic and aperiodic tasks in priority-driven systems". ASPDAC, pp. 635–658, 2004.

Hamzeh M., Fakhraie S. M., and Lucas C. (2007). Soft Real-Time Fuzzy Task Scheduling for Multiprocessor Systems. International journal of intelligent technology, 2(4).

Jejurikar R. and Gupta R. (2006). Energy aware task scheduling with task synchronization for embedded real time systems. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, 25(6), pp. 1024 – 1037.

Kadhim S.J. and Al-Aubidy K.M. (2010). Design and Evaluation of a Fuzzy-Based CPU Scheduling Algorithm. V. V Das et al. (Eds.): BAIP 2010, CCIS 70, pp. 45 – 52, 2010. Springer-Verlag Berlin Heidelberg.

Khani Z., Singh R. and Alam J. (2012). Tasks allocation using fuzzy inference in parallel and distributed system. Journal of Information and Operations Management. E-ISSN: 0976-7762, 3(2).

Lee J., Koh K., and Lee C. (2007). Multi-speed DVS algorithms for periodic tasks with non-preemptible sections. 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp. 459-468.

Liu. W. (2007). Techniques for leakage power reduction in nanoscale circuits: A survey. IMM-Technical Report-2007-04, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, 2007.

Pandey D., and Sharma M. K. (2011). Fuzzy Better Job First Scheduling Algorithm. International Journal of Computer Applications (0975 – 8887) 33(9).

Pillai P. and Shin K. G. (2007). Dynamic DVFS scheduling. J. Henkel and S. Parameswaran (eds.), Designing Embedded Processors – A Low Power Perspective, pp.243–258.

Sheo D., Goel P., and Kaur K. (2012). A Fuzzy Approach Scheduling on More Than One Processor System in Real Time Environment. International Journal of Scientific Research Engineering & Technology (IJSRET) 1(5), pp 289-293.

Venkatachalam V., and Franz M. (2005). Power reduction techniques for microprocessor systems. ACM Computing Surveys (CSUR), 37(3), pp: 195-237.

Zhang F. and Chanson S. T. (2004). Blocking-aware processor voltage scheduling for real-time tasks. ACM Transactions in Embedded Computing Systems, pp. 307–335.