Anju S. Pillai[1,*],
T. B. Isha[2]

J. Electrical Systems 10-1 (2014): 36-47

Regular paper

Dynamic Frequency Scaling Based Energy Consumption Reduction for Power-aware Embedded Systems - A Simulation and Experimental Approach

Journal of Electrical Systems

Energy conservation is a critical issue in battery operated systems. Energy consumption reduction of portable embedded systems is essential for increasing the battery life, enabling better system usefulness. One of the promising techniques of energy consumption reduction is Dynamic Voltage and Frequency (DVFS). In this paper, a Dynamic Frequency Scaling (DFS) method is used to reduce the energy consumed by an Atmel ATmega 16 microcontroller while executing a robotic application. A noticeable amount of energy saving was observed both by simulation and experimental validation when switching frequency was varied from rated frequency to below rated frequencies, without violating any of the temporal requirements.

Keywords: Energy consumption reduction; embedded processors; real time systems; dynamic voltage and frequency scaling; power saving.

1. Introduction

Today, there is a dramatic increase of life style of mankind due to the wide spread use of embedded systems. These types of systems are designed to perform specific tasks, rather than a general purpose system performing multiple jobs. Some of the embedded systems are of real time nature, which has the additional requirement of meeting the real time performance constraints. Meeting the real time constraints is essential for delivering the functional and temporal requirements apart from providing safety and reliability for such systems. These types of systems have limited hardware and software to simplify the cost and increase the performance. Therefore, the functioning of such real time embedded systems is to be given at most care and attention so that the system delivers the correct output at the right time, without violating any of the temporal constraints.

The usefulness of any portable embedded system lies on factors like: the capability of delivering its functionalities meeting the temporal constraints and the system reliability to provide enough battery backup. Since most of the systems are battery operated portable devices, one of the crucial factors which determine its effective usage is the battery life. When compared to the advancements in the device technology, the growth in battery technology is less and there exists a battery gap. The battery gap is ever rising compared to the abrupt increase in energy requirements of embedded systems. This advocates the use of resources in a more effective way, to increase the lifetime of the system by reducing the energy consumption [1]. Also, as the scale of integration improves, smaller and faster transistors are fused into a single chip as per performance demands with increased operating frequency and processing capacity per chip. It also accelerates the power dissipation of the chip. Another important observation made by Intel co-founder Gordon E. Moore in 1965 is that: the transistor density doubles every 18 months [2]. Moore’s law predicts that this tendency will continue in the future also. All these factors lead to the significance of an energy constrained system.

Energy consumption reduction of an embedded processor can be attempted in many different ways viz. hardware design approaches, software approaches and power aware approaches. Among these three broad techniques, significant energy savings can be accomplished by software approaches; where in power aware algorithms and protocols are implemented to reduce the energy consumption of the embedded processor. There is a good deal of algorithms available in the literature which attempt to reduce the energy consumption of embedded processors without modifying the underlying architecture and the circuit. Some of the major techniques include Real Time Dynamic Voltage Scaling (RT-DVS), feedback priority based scheduling with DVS, inter task and intra task DVS technique, preemption reduction by DVS scheme, Preemption Threshold Scheduling (PTS) and processor slow down (DVS) with PTS and many more.

In the proposed work, energy consumption reduction by DFS is addressed. In DFS, the processor is made to switch between appropriate frequencies within the operating frequency range, so as to minimize the energy consumption

Manuscript received December 06, 2013; revised December 31, 2013; accepted January 28, 2014.

[1,*] Corresponding author: Anju S. Pillai, Department of Electrical and Electronics Engineering, Amrita Vishwa Vidyapeetham, Amritanagar Post, Coimbatore, Tamilnadu, India. E-mail: s_anju@cb.amrita.edu
[2] Professor, Department of Electrical and Electronics Engineering, Amrita Vishwa Vidyapeetham, Amritanagar Post, Coimbatore, Tamilnadu, India. E-mail: tb_isha@cb.amrita.edu

without violating any functional and temporal constraints of the system. In most of the works, the performance is studied by simulation only. In the current work, an experimental validation of DFS is carried out where the embedded processor is made to operate at different frequencies to reduce energy consumption with no task missing its respective deadlines.

The rest of the paper is organized as follows: Section 2 describes the related work done and section 3 details the system model. Section 4 depicts the methodology followed by experimental details in section 5. Section 6 briefs the experimental results and finally, section 7 concludes the work.

2. Related Work

The continuous growth in semiconductor technology has caused a commendable increase in performance. The main techniques used for these improvements are mainly by increasing the clock rates and exploiting the instruction level and thread level parallelism and by providing redundant functional units to cater to multi-core and multi-processor systems. These performance increases are earned with some direct and indirect costs like: increased packing cost of the chip and cooling cost of the system, environmental concerns and many more. One of the notable negative impact is the rise in power dissipation. There are two types of power dissipation occurring in any CMOS circuit - static and dynamic. Static power dissipation is related to the logical states of the circuit and leakage current is the main source of this dissipation. On the contrary, dynamic dissipation is due to the switching activities of the circuits. Higher the frequency of switching activities in the circuit, larger the power dissipation. This is mainly due to the charging and discharging of capacitances in the circuit [3].

Most of the low power design techniques attempt to reduce the power dissipation by reducing the switching voltages, and frequencies, capacitances and leakage and static currents of CMOS chips. Power optimization can be done at device level, circuit level, logic/gate level, architectural level and system and software level. In the present work, power reduction by software approach is attempted. DVFS is one of the best techniques to reduce switching power and leakage power. There are a class of novel algorithms proposed by Padmanabhan Pillai et.al [4] for Real Time Dynamic Voltage Scaling (RT-DVS) for low power embedded systems namely: Cycle conserving RT-DVS and Look-ahead RT-DVS. In both the cases, energy reduction was accomplished by dynamically switching the operating points of the processor. Woonseok Kim [5] suggested DVS for Fixed Priority System (FPS) by using work demand analysis and by efficient slack determination of the embedded processor. Simultaneous execution of inter task and intra task DVS technique w as proposed by Jaewon, but with increased computational overhead [6].

Energy consumption reduction was also attained by reducing the unwanted context switches of the processor. Even though context switches are desirable for the implementation of a real time system to achieve better flexibility meeting the temporal constraints, if not controlled, may result in increased delay, memory usage, cache related preemption delay, difficulty in manipulating the task queues and extra energy wastage [7], [8]. So, energy consumption of an embedded processor can be minimized by controlling context switches. Context switch reduction by PTS was proposed by Wang and Saksena [9] for FPS and later modified by Jejurikar et. al [10] to Dynamic Priority System (DPS). Two techniques for preemption reduction viz. accelerated completion based technique and delayed preemption based control technique for preemption reduction were proposed by W. Kim and J.Kim [5]. In [11], a method by identifying the maximum number of preemption occurring at run time in FPS is suggested by reassigning task attributes. Other remarkable techniques include: Deterministic Stretch-to-Fit (DSF) technique by Bhatti et.al [12], DVS for battery powered real-time systems proposed by Ranvijay [13], Inter and intra application dynamism by Kodem [14], Job phasing aware preemption deferral initiated by Marinho [15], and task partitioning to reduce temperature, thereby reducing the power dissipation introduced by Wang [16].

In two of the papers presented earlier [17], [18] energy consumption reduction by modified Look Ahead RT-DVS algorithm was performed where the tasks are switched between possible appropriate operating points consisting of operating voltages and switching frequencies to save energy consumption of the processor. Also, by controlling context switches by improved PTS, a substantial amount of energy savings was noticed [19]. The current work is an experimental validation of energy reduction of an embedded processor by DFS technique.

3. System Description

This section details the system model. The system under consideration is a uni processor system where different tasks are scheduled based on priority assignment. Priority assignment is by Fixed Priority where every task holds a priority based on Rate Monotonic (RM) policy. The priority assignment rule is: *shorter the time period, higher the priority*. The assigned priority remains constant throughout the task execution period. Each invocation of the same task will hold the same priority. Every task is assumed to hold deadline (D_i) equal to its time period (T_i) which is one of the assumptions needed to schedule tasks by RM policy ($D_i = T_i$). Tasks are scheduled only when feasibility is ensured by the utilization upper bound condition as given below:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \le n\left(2^{1/n} - 1\right) \tag{1}$$

where: U is the total utilization of the task set, C_i the worst case execution time and T_i the corresponding time period of task τ_i. In case, utilization test fails, Response Time Analysis (RTA) is performed. The schedulability check is done for the hyper period (H) which is the Least Common Multiple (LCM) of the task time periods.

3.1 Processor Model

The processor used for both simulation analysis and hardware implementation is Atmel ATmega 16 microcontroller. The input voltage of the microcontroller is within the range of 6.5 to 8.4V with 11.059MHz crystal oscillator. For energy conservation, the oscillator frequency is varied from 11.059MHz to discrete values of 5.53, 2.77 and 1MHz.

3.2 Energy Model

The power consumed by the tasks while running on a processor is computed using:

$$P = C_{eff} * v^2 * f \tag{2}$$

where: P is the dynamic power consumed by the processor, C_{eff} is the effective load capacitance, v is the supply voltage and f the operating frequency of the processor [20]. Since, energy is the rate of power flow; the total energy consumed by the processor for the hyper period is calculated as the sum of the individual task instance energy consumption as given below:

$$E = \sum_{i=1}^{n} \sum_{j=1}^{H/T_i} P_i^j * C_i^j \tag{3}$$

where: P_i^j is the power consumed by the jth instance of task τ_i and C_i^j is the execution time of jth instance of task τ_i at a particular frequency.

3.3 Application Details

The application chosen is an automatic path tracking in a known environment using SPARK IV robot. The objective is, the robot needs to start from an assigned initial point to a predetermined destination point. On its way, it is to avoid any obstacles and find its way to the destination and reach there on or before 55sec. The operating voltage and frequency of the ATmega microcontroller is fixed initially and the time required by the SPARK IV robot to reach the destination and complete one revolution was measured using a stop watch. The robot is then made to continue the travel until the battery of the robotic system was drained. The application being real time in nature has to meet both functional and temporal requirements. These requirements are:

Functional requirement:

- Reach the destination avoiding obstacles on the way

Temporal requirement:

- Complete one revolution within 55sec. (Deadline)

The deadline constraint of the SPARK IV robot is fixed based on the computation time of the robot as follows:

- Measure the time taken by robot to complete one revolution when operated at rated frequency and rated supply voltage (Computation time).
- Change the clock frequency of the controller to the next feasible value and measure the time taken to complete one revolution.
- Find the additional time required by the robot to travel when the clock frequency is lowered
- Similarly, the time difference for other clock switching is also measured and the average time for a clock switch is computed
- With the computation time add the average time for a clock switching * no. of clock switches; to find the deadline

Real time tasks to implement the application

- Motor task – To run continuously to explore the path
- Obstacle avoidance task – On identification of an obstacle, find an alternate path to avoid a collision
- Buzzer task – On completion of each revolution blow the buzzer

4. Methodology

The energy consumption reduction of the embedded processor is achieved by DFS technique. In this technique, the tasks are made to run at lower frequency values ranging from minimum (f_{min}) to maximum (f_{max}) frequency of the processor. The tasks are executed at lower frequencies to conserve energy without compromising the temporal as well as functional constraints of the system. The DFS technique is explained in the following section.

4.1 Dynamic Frequency Scaling

In the case of any processor, changing the operating frequency of the processor will change the time required to run a task on the processor. Thus, changing the operating frequency will make the task to run either fast or slow at high or low frequency respectively. The operating frequency of the processor is reduced to conserve energy, without missing any of the task deadlines.

The execution time of a task is computed as the number of processor clock cycles required to run the task code. These processor cycles are computed when the processor is operated at the rated frequency. And the execution time of the code thus computed from the processor cycles are assumed as the Worst Case Execution Time (WCET). But, at run time, tasks run for Actual Case Execution Time (ACET), which is either less or equal to the WCET. These unused task execution times will normally idle the processor. In DFS technique, instead of idling the processor, the unused slack times are used to slow down the task instances which have enough time left to meet the deadline.

If C_{i_w} is the WCET of task τ_i when operated at rated frequency f_rated and if τ_i takes only C_{i_a} time at run time which is the ACET ; where $C_{i\_a} \leq C_{i\_w.}$ In such a case, the processor idle time (T_idle) is given by:

$$T\_idle = C_{i\_w} - C_{i\_a} \tag{4}$$

This processor idle time can be used by the just completed task before the slack $\tau_{i\ -\ 1}$ to slow down its execution time to conserve energy. If the WCET of $\tau_{i\ -\ 1}$ is $C_{i\ -\ 1\_w}$, which now can be modified as $c_{i\ -\ 1\ \ new}$:

$$C_{i-1\_new} = C_{i-1\_w} + T\_idle \tag{5}$$

where: C_{i-1_w} is the WCET of a task τ_{i-1} when operated at rated frequency $f_{\_rated}$. As the execution time of a task is inversely proportional to the operating frequency of the processor, it can be written as:

$$C_{i-1\_w} \propto 1/f\_rated\_ \tag{6}$$

and

$$C_{i-1\_new} \propto 1/f\_new \tag{7}$$

Taking the ratio of equation 6 and 7

$$C_{i-1\_w} / C_{i-1\_new} = f\_new/f\_rated \tag{8}$$

From the above relation, the new frequency at which the processor is to be operated now to run the task $\tau_{i\ -\ 1}$, to have an increased execution time at low frequency can be computed as:

$$f_{\_new} = \frac{C_{i-1\_w}}{C_{i-1\_new}} * f_{\_rated} \tag{9}$$

Thus, to reduce the energy consumption of the embedded processor, the processor slack times are effectively utilized to execute the possible task instances at lower frequency by ensuring that:

$$f_{min} < f_{\_new} < f_{max} \tag{10}$$

In order to conserve energy, the tasks are required to run at lower frequencies which means some unused processor time or slack time is available. i.e. energy conservation is possible only with the available processor slack times. This indirectly puts a limit on the amount of energy that could be conserved. So, when the work load on the processor is low, the amount of energy saving will be high when compared to a heavily utilized processor.

4.2 Feasibility Test

The DFS technique is applied only to a task set which is schedulable by RM policy. So the analysis starts with the feasibility test. The sufficient condition for schedulability is utilization upper bound test as mentioned in equation 1. If the task set utilization is less than or equal to the upper bound criteria, then it is guaranteed that the given task set is schedulable. If this condition is not satisfied then RTA is done to check the schedulability. In RTA, worst case response time of each task is computed analytically and is compared with the individual task deadlines. If the response time is within the deadline, then the schedulability is assured. The equation for RTA is given by [21]:

$$R_i^{n+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil * C_j \tag{11}$$

where: R_i represents the response time of task τ_i and C_j and T_j is the execution time and time period of the high priority task τ_j respectively.

4.3 Simulation Analysis

To check the validity of DFS technique, a simulation analysis was carried out using MATLAB. The performance of DFS technique was tested with synthetic tasks. The synthetic tasks are generated using UUnifast algorithm [22]. A small group of tasks with varying utilizations from 0.03 to 0.3 with 4 tasks of 25 numbers are generated using UUnifast algorithm for testing and validating. For the simulation analysis, once the tasks with (C_i, T_i) are given, the actual execution time for each task is computed by using the frequency factor. Frequency factor is the ratio of set frequency (f_set) to the rated frequency $(f_{\_rated})$. For energy conservation, tasks are to be run at frequencies which are less than the rated frequency. $(f_{\_set} < f_{\_rated})$. The frequency factor determines the fraction of reduction from WCET. Once the fractional reduction C_{i_fract} is computed, the ACET is computed as:

$$C_{i\_fract} = C_{i\_w} * \frac{f_{\_set}}{f_{\_rated}} \tag{12}$$

$$C_{i\_a} = C_{i\_w} - C_{i\_fract} \tag{13}$$

Using the above equation, the actual execution time for each task is computed. For a task instance whose frequency cannot be reduced below the rated frequency due to the infeasibility; the frequency factor is kept as 1. For such tasks the actual execution time is calculated as a fixed percentage of WCET. In this analysis for tasks with frequency factor 1, the actual case execution is taken as 85% of WCET. The given tasks are assigned priority by RM policy and scheduled with the computed ACET: $C_{i\_}a$ and T_i. From the schedule, the processor slack times are determined and the task which had just completed before the slack time was also identified. On finding the just completed task instance and knowing its respective deadline, the reduced frequency at which the task instance could be run without missing the deadline to save energy consumption was calculated. That task instance was then run at the reduced frequency and the checking is then continued to identify the further slack times and so on until every slack time is discovered. The flow chart of the DFS technique is explained in Fig.1.

Considering a sample task set for DFS technique as in Table I. The schedule obtained when the processor is operated at rated frequency is shown in Fig. 2 and with DFS is shown in Fig. 3.

Table I: Example task set for DFS

Tasks	C_i	T_i
Task 1	1	5
Task 2	1	6
Task 3	3	10
Task 4	2	15

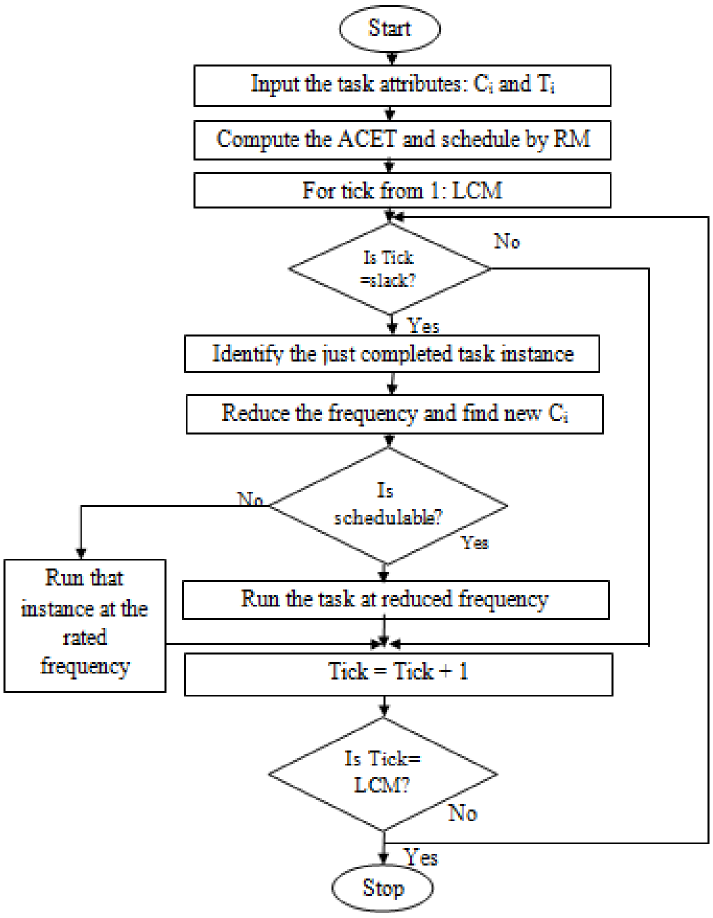
Start
Input the task attributes: C_i and T_i
Compute the ACET and schedule by RM
For tick from 1: LCM
Is Tick =slack?
No
Yes
Identify the just completed task instance
Reduce the frequency and find new C_i
Is schedulable?
No
Yes
Run that instance at the rated frequency
Run the task at reduced frequency
Tick = Tick + 1
Is Tick= LCM?
No
Yes
Stop

Fig. 1 Flow chart describing DFS algorithm

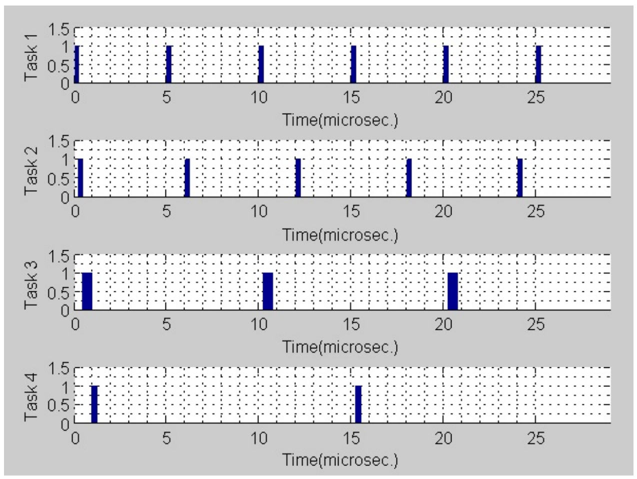

Fig. 2 Schedule with processor at rated frequency

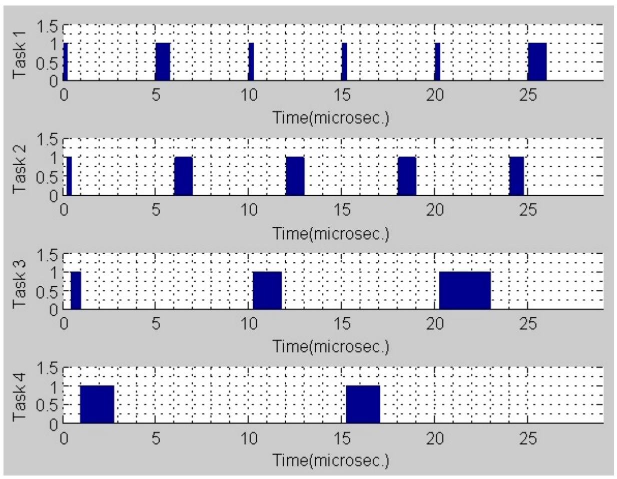

Fig.3 Schedule with DFS technique

When the tasks are scheduled using DFS technique, the different instances were run at different frequencies to conserve energy without losing the schedulability. The variable frequency values at which the instances were run is shown in Table II.

Table II: Task instance level frequencies (MHz.)

Instances	1	2	3	4	5	6
Task 1	11.059	2.77	11.059	11.059	11.059	1
Task 2	11.053	1	1	1	2.77	0
Task 3	11.059	5.53	1	0	0	0
Task 4	1	1	0	0	0	0

4.4 Robotic Application using Atmel ATmega Microcontroller

A simulation study of SPARK IV robotic application using DFS algorithm was carried out initially. The ATmega 16 microcontroller was used for implementation. For performing the simulation, the task computation time C_i for the three real time tasks: motor task, obstacle avoidance task and buzzer task were computed analytically and the time period T_i was fixed accordingly. The Table III shows the task attributes for the chosen hardware application.

Table III: Task attributes for the robotic application

Tasks	Ci(µsec.)	Ti(µsec.)
Buzzer task	1.5	3
Obstacle avoidance task	4.1667	10
Motor task	116.043	300

The corresponding execution trace is shown in Fig. 4 and Fig. 5 at rated frequency and with DFS technique.

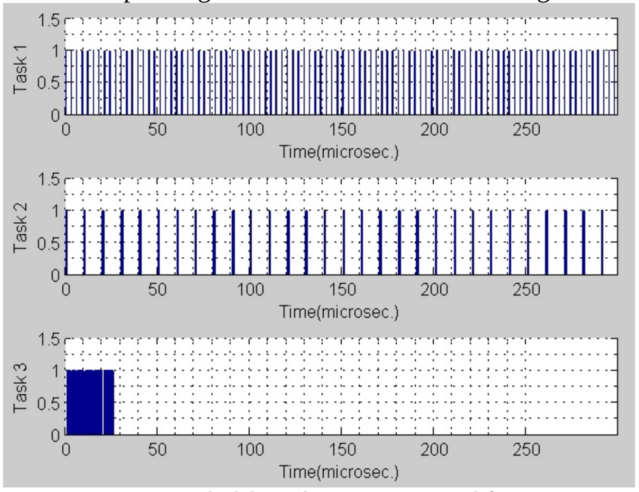

Fig. 4 Schedule with processor at rated frequency

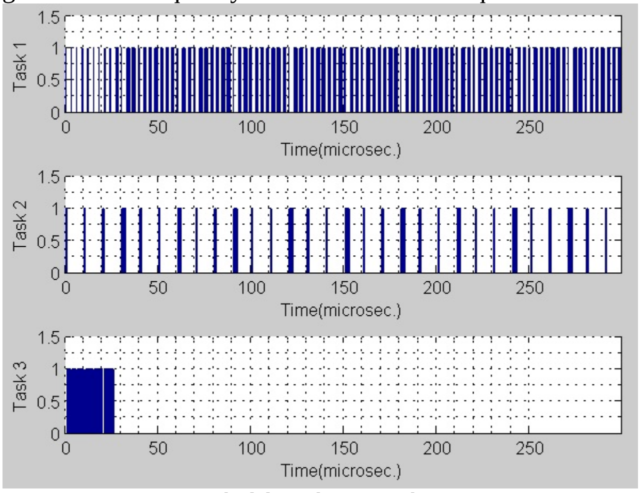

Fig. 5 Schedule with DFS technique

4.5 Simulation Results

The results obtained from MATLAB simulation is shown in Fig. 6, which describes the variation in energy consumed by the microcontroller to run the synthetic tasks when operated at different frequencies. From the graph, it is clear that as the processor operating frequency is reduced, the energy consumption also reduces. Energy consumption is minimum when the processor is operated at the lowest frequency 1MHz. But, when operated at 1MHz, some of the task sets were not schedulable, because of the inability of the tasks to meet its deadline. These tasks energy consumption is represented as negative values in the graph for identification. Therefore, it can be concluded that DFS is optimal for reducing energy consumption of the processor, while ensuring the real time constraints. Fig. 7 shows a comparison between the average energy consumed by the microcontroller when operated at rated frequency and with DFS technique.

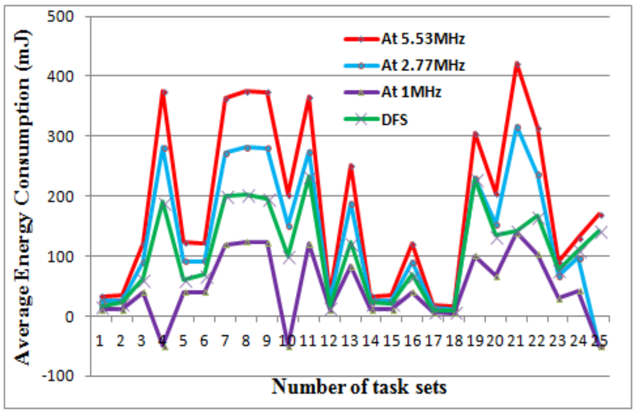

Fig. 6 Average energy consumption Vs No. of task sets

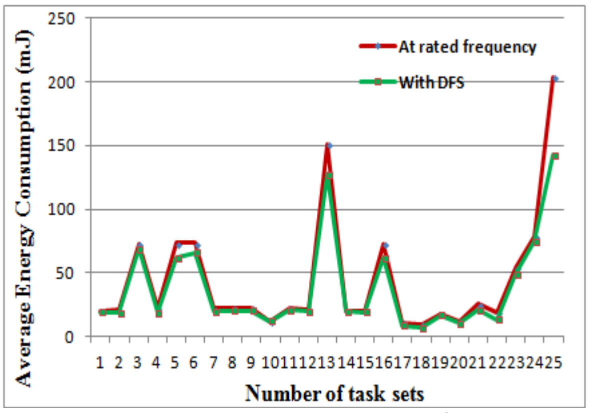

Fig. 7 Energy consumption variation with DFS

4.6 Experimental Validation

A SPARK IV robot was selected for the present application. SPARK IV is a universal robotic research platform based on AVR microcontrollers. SPARK IV supports ATMEL ATmega 8-bit microcontroller - ATmega16.

Sensors:

- Three white line sensors
- Forward looking Sharp GP2D120C infrared range sensors
- Two position encoders one on each wheel
- One directional light intensity sensor
- Battery voltage sensing
- Gyroscope and accelerometer

Power:

- Two Lithium ion batteries of 8.4V and current rating of 660mAH
- Input voltage ranges 6.5 to 8.4V

Locomotion:

- Two DC geared motors and caster wheel at front as support
- Top Speed: 20 cm/second
- Wheel Diameter: 40mm
- Position encoder: 30 pulses per revolutions

The block diagram of SPARK IV robot is shown in Fig. 8

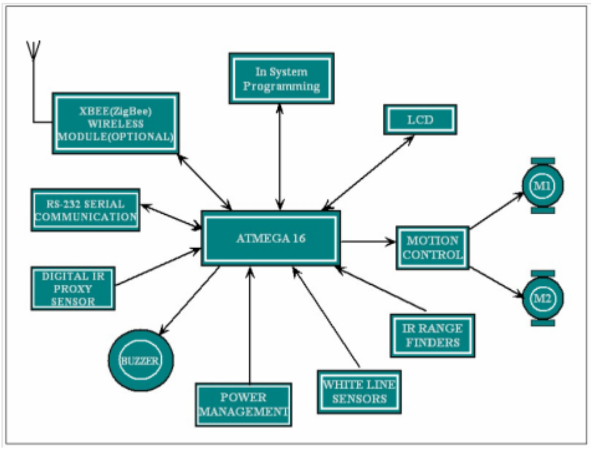

Fig. 8 Block diagram of SPARK IV Robot

4.7 Software Details

The ATmega microcontroller is programmed using Embedded C code using ICC AVR Integrated Development Environment (IDE). The hex code is generated from the source code using ICC AVR. The generated hex code is downloaded into the ATmega microcontroller through In System Programmer (ISP) using AVR DUDE GUI. While running the above robotic application, the energy consumed by the SPARK IV robot to perform the desired functionality meeting the temporal requirements was computed.

- Without including energy reduction techniques
- Including energy reduction techniques-DFS

For reducing the energy consumption of the ATmega microcontroller, the controller is switched between different frequency levels. Energy can be saved when processor is operated at lower frequency ensuring temporal requirements. For the analysis and implementation an Atmel ATmega 16 microcontroller is used. The crystal frequency of ATmega controller is 11.0592MHz and can have clock frequency variations between 0 to 16MHz. For the current application, the operating frequency of the ATmega controller is varied for the following discrete frequency values: 11.059MHz, 5.529MHz, 2.765MHz and 1MHz. The different frequency values are selected by configuring the Oscillator Calibration Register (OSCCAL) of ATmega microcontroller. When OSCCAL is zero, the lowest available frequency (1MHz) is chosen and writing $FF gives the highest available frequency (11.059MHz). Writing nonzero values to this register will increase the frequency of the internal oscillator. The oscillator is intended for calibration in powers of 2 and tuning to other values is not guaranteed. This fixes the selection of frequency switches as the rated available value of 11.059MHz, its half 5.529MHz, its half 2.765 and 1MHz.

5. Experimental Setup

The experimental set up is as shown in Fig. 9. After the successful compilation, the code was downloaded to the ATmega microcontroller. The SPARK IV robot was initially charged up to the full voltage and the initial voltage level is maintained constant for all iterations. In this experiment the initial voltage value was set at 8.26V. The robot was initially placed at the start point and then powered on. The time required by SPARK IV to reach the destination was measured using a stop watch and noted as shown in Fig. 10.

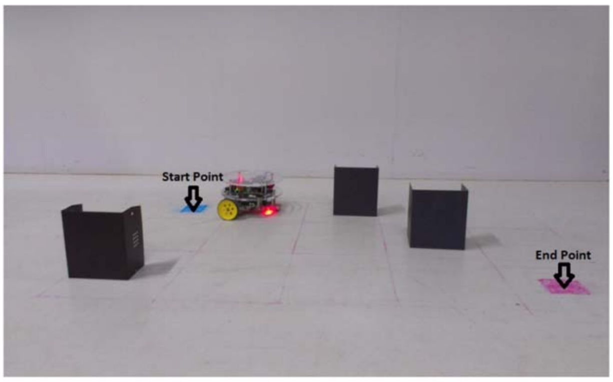

Fig. 9 Experimental setup

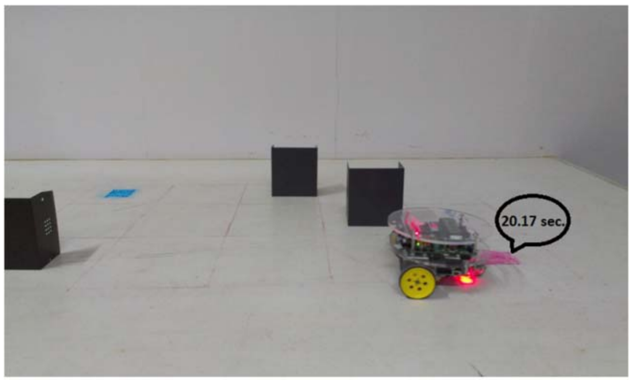

Fig. 10 Time taken to reach the destination

Also the, time taken to complete one revolution, i.e., the time taken to travel from start point to destination point and back to the start point was also measured. Then, the SPARK IV robot was run continuously until, the battery was drained. The battery drain warning beep sound indicates this state.

6. Experimental Results

The experiment was conducted by changing the crystal oscillator frequency. The different iterations were run at 11.059MHz, 5.529MHz, 2.765MHz and 1MHz. The different parameters measured are tabulated. Table IV shows the results obtained when Atmel ATmega 16 microcontroller operated at 11.059MHz. Table V to Table VIII shows the respective parameters when ATmega microcontroller operated at 5.529MHz, 2.765MHz and 1MHz and with DFS respectively.

Table IV: Atmel ATmega 16 operated at 11.059MHz.

A	B	C	D	E	F
	8.26V	53.24sec	49.37min	51	7.11V
2	8.26V	53.61sec	49.50min	51	6.77V
3	8.26V	53.33sec	49.27min	51	6.78V
Avg.	8.26V	53.39sec	49.38min	51	6.89V

Table V: Atmel ATmega 16 operated at 5.53MHz.

A	B	C	D	E	F
1	8.26V	54.17sec	48.16min.	48	6.72V
2	8.26V	54.12sec	48.23min.	48	6.77V
3	8.26V	54.31sec	48.01min.	47	6.75V
Avg.	8.26V	54.2sec.	48.13min.	48	6.75V

A-No. of iterations, B-Voltage across the battery when fully charged, C-Time required by the robot to complete one revolution, D-Time taken to travel until the battery is drained, E-No. of revolutions completed, F-Voltage across the battery when warning beep sound is heard

Table VI: Atmel ATmega 16 operated at 2.77MHz.

A	B	C	D	E	F
1	8.26V	54.68sec.	48.07min.	46	7.02V
2	8.26V	54.98sec.	47.23min.	46	6.78V
3	8.26V	54.78sec.	47.41min.	46	6.74V
Avg.	8.26V	54.81sec.	47.57min.	46	6.85V

Table VII: Atmel ATmega 16 operated at 1MHz.

A	B	C	D	E	F
1	8.26V	55.04sec	47.02min.	45	7.03V
2	8.26V	55.21sec	47.12min.	45	6.74V
Avg.	8.26V	55.13sec	47.07min.	45	6.89V

A-No. of iterations, B-Voltage across the battery when fully charged, C-Time required by the robot to complete one revolution, D-Time taken to travel until the battery is drained, E-No. of revolutions completed, F-Voltage across the battery when warning beep sound is heard

Table VIII: Atmel ATmega 16 operated with variable frequency

A	B	C	D	E	F
1	8.26V	55.52sec.	46.59min.	46	6.74V
2	8.26V	55.45sec.	47.27min.	48	6.76V
3	8.26V	55.55sec.	47.32min.	46	6.82V
Average	8.26V	55.507sec.	47.06min.	47	6.77V

A-No. of iterations, B-Voltage across the battery when fully charged, C-Time required by the robot to complete one revolution, D-Time taken to travel until the battery is drained, E-No. of revolutions completed, F-Voltage across the battery when warning beep sound is heard

Thus, the SPARK IV robot was run to meet the system requirements varying the clock frequencies. The data obtained are consolidated in Table IX. The functional requirements of the application was met when the ATmega was operated at the four discrete frequencies: 11.059MHz., 5.529MHz., 2.765Mhz., 1MHz and with DFS. But, the temporal requirement of completing one revolution within 55sec. was violated when the controller was operated at 1MHz. This means that for implementing the above application, operating the controller at 1MHz. static frequency is not feasible. Also, it is verified by the conduction of above experiment that energy consumption reduction can be effectively reduced using DFS technique rather than static frequency scaling satisfying the temporal requirements of the application.

Table IX: Consolidated results

	11.059 MHz	5.53 MHz	2.77 MHz	1 MHz	DFS MHz
Time to complete one revolution (sec.)	53.39	54.2	54.81	55.13	55.507
Time taken to travel until the battery is drained (min.)	49.38	48.13	47.57	-	47.06
No. of revolutions	51	48	46	-	47
Energy consumed by the robot to travel one revolution (J)	291.06	295.48	298.8	-	297.17
Energy consumed to continue the rotation until battery is drained (kJ)	16.152	15.74	15.56	-	15.393

The savings in energy consumption are tabulated in X.

Table X: Comparison of savings in energy consumption by simulation and hardware

Frequency (MHz.)	By Simulation Energy Saving (%)	By Hardware Energy Saving (%)
5.53	11.11	2.55
2.77	17.89	3.66
1	Not schedulable	Not schedulable
DFS	10.17	4.699

Though small, a saving is noticed for this simple application. In this experiment only the clock frequency of the microcontroller is changed keeping the supply voltage constant. That is one of the reason for reduced energy saving. Better savings can be obtained when the processor operating voltage and clock frequencies are changed dynamically at run time, because of the quadratic dependency of voltage on power consumption (As in equation 2). Also, the energy saving will be worthwhile when the application is run over a period of time, where a noticeable saving can be expected. Another reason is in this particular work, only a single processor is considered for a simple embedded real time application implementation; but for complex networked embedded systems with processors of the order of thousands and more are present, the small energy saving of a single processor when accumulated over the number of processors will definitely be significant and remarkable. Similarly support for more frequency and voltage scaling will also be beneficial for more energy saving.

Fig. 11 shows a comparison of the simulation and hardware results. The simulation and experimental results obtained are found not to be the same. This might be due to the non-inclusion of the physical constraints like: friction, stray and inertia losses of the motor. Also, numerous peripherals viz. LEDs, LCD etc. in the experimental setup are not considered while doing the simulation study. In the graph, the energy consumption of the embedded processor is shown negative at 1MHz, to represent the non-feasibility to run the application. The experimental results ensure that energy consumption reduction of an embedded processor can be effectively achieved by Dynamic Frequency Scaling technique.

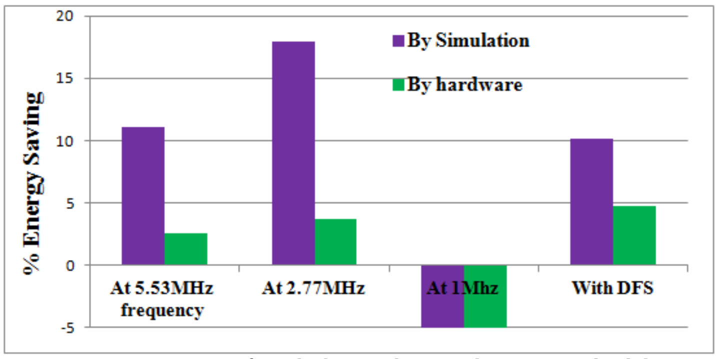

Fig. 11 Comparison of results by simulation and experimental validation

7. Conclusion

In this paper, a technique for energy consumption reduction for an embedded processor is carried out using Dynamic Frequency Scaling (DFS) technique. Both the simulation analysis and experimental validation is carried out to evaluate the performance. DFS technique is implemented using Atmel ATmega microcontroller in SPARK IV robot. An application of automatic path tracking in a known environment was chosen and the micro controller was operated at different static frequencies and also the operating frequency of the controller was changed dynamically at run time to conserve energy. Dynamic frequency scaling was found to be more effective than static frequency scaling, satisfying the temporal requirement of application. For the chosen application, an average of 4.699% and 10.17% of energy saving obtained with dynamic frequency changes ensuring the temporal requirement of the system by hardware and simulation analysis respectively. This shows that a considerable energy saving is possible in applications with numerous tasks.

References

[1] S. I. Park, The design of power aware embedded systems, *PhD Thesis*, University of California, Los Angeleles.
[2] G. E. Moore, Moor's law, Available at http://www.mooreslaw.org/.
[3] G. Yeap, *Practical Low Power Digital Vlsi Design*, Springer London, Limited, 1998.
[4] P. Pillai, K. G. Shin, Real-time dynamic voltage scaling for low-power embedded operating systems, *Symposium on Operating Systems Principles*, 2001, pp. 89–102.
[5] W. Kim, J. Kim, S. L. Min, Preemption-aware dynamic voltage scaling in hard real-time systems, *Proceedings of the 2004 international symposium on Low power electronics and design*, ISLPED '04, ACM, New York, NY, USA, 2004, pp. 393–398. doi:10.1145/1013235.1013328. URL http://doi.acm.org/10.1145/1013235.1013328
[6] J. Seo, T. Kim, N. Dutt, Optimal integration of inter-task and intra-task dynamic voltage scaling techniques for hard real-time applications, *IEEE/ACM International Conference on Computer-Aided Design*, 2005. ICCAD-2005., Nov., pp. 450–455. doi:10.1109/ICCAD.2005.1560110.
[7] H. Ramaprasad, F. Mueller, Tightening the bounds on feasible preemption points, in: RTSS '06: *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, IEEE Computer Society, Washington, DC, USA, 2006, pp. 212–224. doi:http://dx.doi.org/10.1109/RTSS.2006.49.
[8] A. Burns, K. Tindell, A. Wellings, Effective analysis for engineering real-time fixed priority schedulers, *IEEE Transactions on Software Engineering* 21 (5) (1995) 475–480. doi:http://doi.ieeecomputersociety.org/10.1109/32.387477.
[9] Y. Wang, M. Saksena, Scheduling fixed-priority tasks with preemption threshold, *Sixth International Conference on Real-Time Computing Systems and Applications*, 1999. RTCSA '99., 1999, pp. 328–335. doi:10.1109/RTCSA.1999.811269.
[10] R. Jejurikar, R. K. Gupta, Integrating processor slowdown and preemption threshold scheduling for energy efficiency in real time embedded systems (2004).
[11] R. Dobrin, G. Fohler, Reducing the number of preemptions in fixed priority scheduling, *Proceedings 16th Euromicro Conference on Real-Time Systems*, ECRTS 2004., 2004, pp. 144 – 152. doi:10.1109/EMRTS.2004.1311016.
[12] M. Bhatti, C. Belleudy, M. Auguin, An inter-task real time dvfs scheme for multiprocessor embedded systems, *Conference on Design and Architectures for Signal and Image Processing* (DASIP), 2010, pp.136 –143. doi:10.1109/DASIP.2010.5706257.
[13] Ranvijay, R. Yadav, S. Agrawal, Efficient energy constrained scheduling approach for dynamic real time system, *1st International Conference on Parallel Distributed and Grid Computing* (PDGC), 2010, pp.284–289. doi:10.1109/PDGC.2010.5679912.
[14] M. Koedam, S. Stuijk, H. Corporaal, Exploiting inter and intra applica- tion dynamism to save energy, in: DSD'11, 2011, pp. 708–715.
[15] J. Marinho, S. Petters, Job phasing aware preemption deferral, *9th International Conference on Embedded and Ubiquitous Computing (EUC)*, 2011 IFIP, pp. 128–135. doi:10.1109/EUC.2011.46.
[16] Z. Wang, S. Ranka, P. Mishra, Temperature-aware task partitioning for real-time scheduling in embedded systems, *VLSI Design*, 2012, pp.161–166.
[17] A. L. Mohan, A. S. Pillai, Dynamic voltage scaling for power consumption reduction in real-time mixed task model, *Int. conference on Computer Science Engineering and Applications*, July 2011.
[18] A. L. Mohan, A. S. Pillai, Dynamic voltage scaling with reduced frequency switching and preemptions, *Int. conference on Computer Science and Information Technology*, 2011.
[19] A. Paul, A. S. Pillai, Reducing the number of context switches in real time systems, *Int. conference on Process Automation, Control and Computing* (PACC), 2011,pp.1–6. doi:10.1109/PACC.2011.5979044.
[20] Principles of CMOS VLSI Design, Person Education, India,
[21] M. Joseph, P. Pandya, Finding response times in a real- time system, *The Computer Journal* 29, 5, 1986, pp.390–395.
[22] E. Bini, G. Buttazzo, Biasing effects in schedulability measures, *Proceedings of 16th Euromicro Conference on Real-Time Systems*, 2004. ECRTS 2004, pp.196–203. doi:10.1109/EMRTS.2004.1311021.