Zbornik radova 56. Konferencije za ETRAN, Zlatibor, 11-14. juna 2012.
Proc. 56th ETRAN Conference, Zlatibor, June 11-14, 2012

ENERGY EFFICIENCY OF HARD REAL-TIME SYSTEMS BASED ON STANDBY SPARING

Borisav Jovanović, Milunka Damnjanović, *University of Niš, Faculty of Electronic Engineering*
{borisav.jovanovic, milunka.damnjanovic}@elfak.ni.ac.rs

Abstract – *The proposed low-power technique is used in hard real-time systems. It is based on Standby sparing technique. Besides, to validate technique, we have developed the multiprocessor system, consisting of two identical microcontroller cores. The system, maintains both the fault tolerance and low-power operation.*

1. INTRODUCTION

Energy management and power dissipation of complex System-on-chips (SoC) are key parts of design specifications. In the past, several power management techniques have been proposed. Among these, dynamic voltage and frequency scaling (DVFS), and power gating, became very popular and nowadays are widely applied in many electronic circuits and systems.

The DVFS technique reduces the supply voltage and leads to a quadratic reduction in dynamic energy consumption [1]. Because it is very efficient technique, the DVFS is used for power reduction of many commercial microprocessors. The other technique - the power gating, shuts down the power supply of currently inactive blocks and subsystems, reducing the leakage power in an efficient way [2].

In real-time systems, the techniques like time redundancy [3] and hardware redundancy [4] are commonly used to achieve the fault tolerance. The advantage of time redundancy is that it requires less hardware as compared to hardware redundancy. However, the time redundancy may not be able to accomplish the required consistency of hard real-time systems that are used in safety-critical applications. To achieve high reliability of these systems, the use of hardware redundancy is a obligation [4]. However, the utilization of low power techniques, and especially, low voltage operation, has a negative influence on system's reliability and fault tolerance. It was found that utilization of DVFS exponentially increases the existence of transient faults during microprocessor’s operation [5].

The proposed low-power technique is intended to be used in hard real-time systems. It represents the modification of Standby sparing technique [6], one of the techniques which are based on hardware redundancy. The low-power fault-tolerant methodology is described in the next section. For the utilization in hard real-time applications, we have developed the multiprocessor system, consisting of two identical 8052 microcontroller cores. The system, which is described in section three, maintains both the fault tolerance and low-power operation.

2. THE STANDBY SPARING METHODOLOGY

The low-power technique dedicated to real-time applications is explained on the example of group of tasks, labeled with T_1, T_2, and T_3 (as shown in Fig.1). Each task T_i (i=1,2,3) is characterized by worst case time interval $WCET_i$. Each task T_i is executed by microcontroller core, called the primary core. The task is executed within the time interval named with actual execution time - AET_i, which is less or equal than $WCET_i$. All the tasks should be completed before the specified deadline time D (Fig.1).

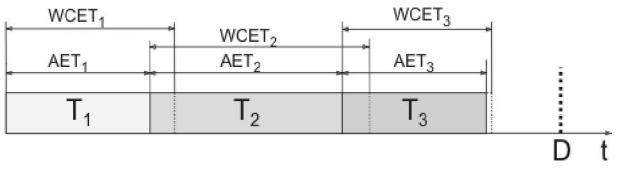

Fig.1. *The schedule of tasks*

The microcontroller core executes the tasks and can operate at the frequency maximum f_{max}, obtained at voltage level V_{max}. Beside V_{max}, several other lower supply voltages and operating frequency values are at the disposal.

The reduction of power consumption is one of the design priorities and microcontroller should spend the minimum energy for a task execution. By using the DVFS, the power consumption of primary core is efficiently reduced. The supply voltage is decreased to minimum value $V_{reduced}$ which is two times less than V_{max}. Also, the primary core executes the instructions at frequency $f_{reduced}$ which is operating frequency maximum at $V_{reduced}$. The frequency value f_{max} is two times greater than $f_{reduced}$.

In real-time systems the safe operation is required and microcontroller system should operate properly in the event of the transient fault. The fault tolerance is provided by exploiting the Standby sparing technique, which is described as follows.

In Standby sparing technique [6], the hardware redundancy relies on the utilization of one additional microcontroller core – called spare core, which is identical to the primary core. The spare core can execute the same sequence of tasks as primary core, and operates in standby mode most of the time. If a transient fault is detected, the primary core switches on the spare unit, which after, executes the back-up task.

To meet the deadline time, the spare core is often turned on before the primary unit finishes its operation and possible faults are checked. Therefore, the back-up task, executed by spare unit, is started before the primary task has been completed. This is the main characteristic of Standby sparing technique. The fault tolerance is provided by using the standby time of the spare unit. If primary core finishes the task without an error, spare cancels further operation, because it not needed anymore, and returns immediately to standby mode. If primary core fails executing the task, the back-up task is completed by spare unit.

The power consumption of primary microcontroller is reduced by decreasing the supply voltage to $V_{reduced}$. The other core, the spare unit, resides often in standby mode and utilizes only power gating for energy efficiency. Because the number of faults exponentially increases with decreasing of supply voltage level, only high supply voltage level of spare unit conserves the fault tolerance [5]. Consequently, the spare unit operates at maximum of the both frequency and supply voltage levels - f_{max} and V_{max}. These are two times greater than the levels in the primary unit.

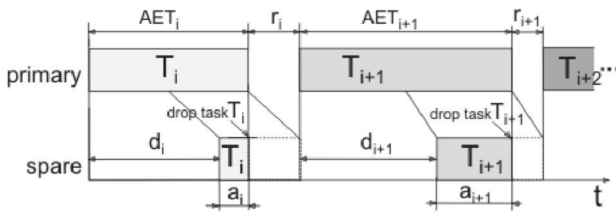

Fig.2. *The task execution performed by primary and spare cores, proposed by standby sparing technique*

The Fig.2 shows the execution of tasks T_i and T_{i+1}, performed by primary and spare units. The primary unit finishes the task within the time interval AET_i. The spare unit, working at f_{max}, starts the execution after delay d_i and finishes after the time interval $AET_i/2$. During the delay time d_i the spare unit is in standby, and thus, saves the energy.

At the moment when primary core finishes the task T_i, the operation of task T_i is checked. The error detection is usually assumed to be part of the software and error detection overhead is considered as a part of the execution time [7]. If error is not found, the task T_i finishes successfully. The execution of spare task is cancelled, and spare immediately goes from active to standby mode. If primary core fails, the next task T_{i+1} can be started after the spare unit completes the backup task T_i.

The power consumption of spare unit is approximately eight times greater than in the primary unit (voltage and frequency levels are two times greater). To reduce the total power dissipation, the time interval during which the spare stays in Active mode should be minimized. To reduce the time interval, the delay time d_i should be increased. However, delay time d_i cannot be increased arbitrarily, because total delay times of all tasks is limited by the deadline time D.

The inter-task time r_i is the delay between two consecutive tasks T_i and T_{i+1}, and can be determined by Eq.1:

$$r_i = d_i - AET_i / 2 \tag{1}$$

where delay d_i has range:

$$d_i \in [\frac{AET_i}{2}, AET_i] \tag{2}$$

If primary unit completes the task, the energy consumption of spare depends on the duration of spare active time interval a_i, given by Eq.3.

$$a_i = \frac{AET_i}{2} - r_i = AET_i - d_i \tag{3}$$

The previous equation can be modified as follows:

$$a_i + r_i = \frac{AET_i}{2} \tag{4}$$

The slack time SLT represents the maximum value of sum of all inter-task times, and it is equal to the difference of deadline time D and the sum of all actual execution times (Eq.5).

$$SLT = (\sum_{i=1}^{N} r_i)_{\max} = D - \sum_{i=1}^{N} ATE_i \tag{5}$$

The spare unit active time A, which contributes the most of total power consumption is:

$$A = \sum_{i=1}^{N} a_i = \sum_{i=1}^{N} \frac{AET_i}{2} - \sum_{i=1}^{N} r_i \tag{6}$$

The spare active time minimum is:

$$\begin{aligned} &(A)_{\min} = \frac{3}{2}\sum_{i=1}^{N} AET_i - D, \quad \sum_{i=1}^{N} AET_i > \frac{2}{3}D \\ &A = 0, \quad \sum_{i=1}^{N} AET_i < \frac{2}{3}D \end{aligned} \tag{7}$$

It can be concluded that if the sum of all actual execution times is less or equal than two thirds of deadline time D, there is no need for spare unit to operate. The spare active time is then equal to zero. Moreover, it can be found from Eq.4 that the sum of the inter-task time and spare active time is equal to the value $AET_i/2$. To reduce the spare power for particular task T_i, the value a_i should be reduced and r_i increased. To meet the deadline time D, there is a need to have enough time to finish all the tasks, but, slack time SLT limits the increase of r_i.

In the hard real time systems the deadline time is close to the sum of actual execution times. The power consumption of spare unit can be large and unacceptable, even greater of the consumption of primary unit. To reduce the power consumption of spare unit, the tradeoff between the fault tolerance and spare core power consumption is made, which is described as follows.

The operation is slightly modified compared to previous description. Consider the execution of two successive tasks T_i and T_{i+1} (Fig.3). Now, there is not any inter-task period between T_i and T_{i+1}, neither for primary nor for the spare units. Immediately after a unit finishes task T_i, it begins the execution of the next task T_{i+1}.

Similar to previous description, the primary unit, operates at $f_{reduced}$, and finishes the task within the time interval AET_i. The spare unit, working at f_{max}, starts the execution after delay d_i and finishes after the time interval equal to $AET_i/2$.

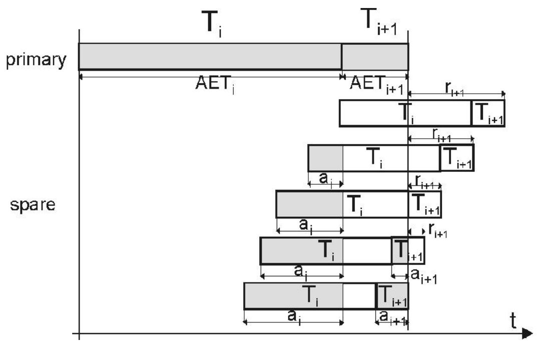

Fig.3. *The operation which doesn't utilize inter-task time periods; case 1: $AET_i/2 > AET_{i+1}$*

Whenever the task T_i is finished successfully, the back-up task T_i is cancelled. The T_{i+1} is started immediately on the primary unit. At the moment when primary core finishes the T_{i+1}, the operation of T_{i+1} is checked. If T_{i+1} is finished successfully, the back-up task T_{i+1} is stopped.

If a fault occurs on the primary unit during T_i, the execution of back-up task T_i is continued. The execution of task T_{i+1} on primary unit should be cancelled, because it would make calculations without the valid outcome from previous task T_i. When the spare unit finishes back-up task T_i, it starts T_{i+1}. The T_{i+1} is executed at V_{max}. In this situation the fault tolerance of task T_{i+1} is lost. Therefore, the power consumption of spare unit is reduced in exchange for the slightly losing the fault tolerance of task T_{i+1}.

Consider first the relation between periods AET_i. and AET_{i+1}, which is given by Eq.8. The relation is illustrated in Fig.3.

$$\frac{AET_i}{2} > AET_{i+1} \tag{8}$$

In the case when spare unit executes only the part of task T_i, the sum of inter-task times and spare active times of tasks T_i and T_{i+1} is equal to:

$$\sum_{j=i,i+1}(r_j + a_j) = (AET_i - AET_{i+1})/2, \quad r_{i+1} \in [AET_{i+1}/2, (AET_i - AET_{i+1})/2] \tag{9}$$

When the spare unit executes the parts of both tasks, the sum of inter-task times and spare active times of tasks T_i and T_{i+1} is given by Eq.10:

$$\sum_{j=i,i+1}(r_j + a_j) = AET_i/2 - r_{i+1}, \quad r_{i+1} \in [0, AET_{i+1}/2] \tag{10}$$

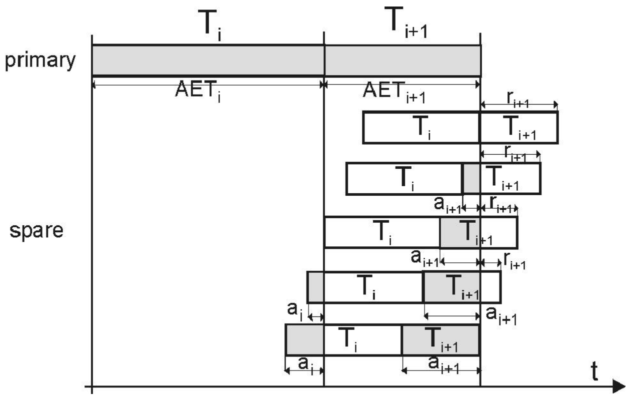

Fig.4. *The operation which doesn't utilize inter-task time periods; case2: $AET_i/2 < AET_{i+1} < AET_i$*

The situation when $AET_i/2 < AET_{i+1} < AET_i$ is shown in Fig.4.

$$\frac{AET_i}{2} < AET_{i+1} < AET_i \tag{11}$$

If spare unit executes only task T_{i+1}, the sum of inter-task times and spare active times of tasks T_i and T_{i+1}, is equal to:

$$\sum_{j=i,i+1}(r_j + a_j) = AET_{i+1}/2, \quad r_{i+1} \in [(AET_i - AET_{i+1})/2, AET_{i+1}/2] \tag{12}$$

When the spare unit executes the parts of both tasks, the sum is given by Eq.13:

$$\sum_{j=i,i+1}(r_j + a_j) = AET_i/2 - r_{i+1}, \quad r_{i+1} \in [0, (AET_i - AET_{i+1})/2] \tag{13}$$

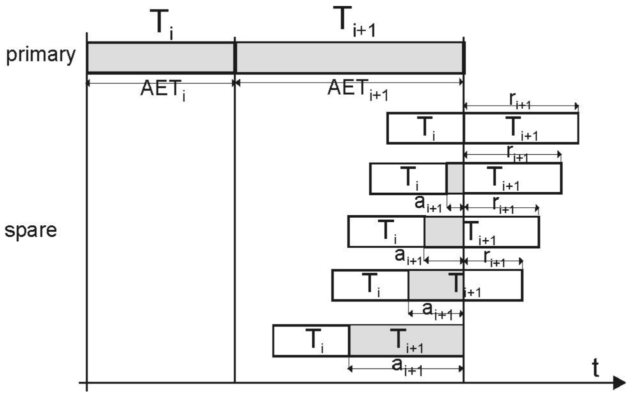

Fig.5. *The operation which doesn't utilize inter-task time periods; case 3: $AET_{i+1} > AET_i$*

The third situation, given in Fig.5, is expressed by condition $AET_{i+1} > AET_i$. The sum of inter-task times and spare active times of tasks T_i and T_{i+1}, is provided by Eq.15.

$$AET_i < AET_{i+1} \tag{14}$$

$$\sum_{j=i,i+1}(r_j + a_j) = AET_{i+1}/2, \qquad r_{i+1} \in [0, AET_{i+1}/2] \qquad (15)$$

The described operation decreases the sum of inter-task times and spare active times for tasks T_i and T_{i+1}. In the first case, which is shown in Fig.3, the total sum of inter-task times and spare active times of all tasks is reduced by value AET_{i+1}. It other two cases (Figures 4 and 5), the reduction is equal to $AET_i/2$. The power consumption of spare unit is reduced when spare active time interval a_i is decreased.

The described operation is repeated until the sum of all inter-task times and spare active times becomes equal or less than slack time SLT. Further reduction would not yield the spare unit energy efficiency.

3. RESULTS

To validate proposed technique, an implementation of 8052 microcontroller core was used. The microcontroller was implemented in the Laboratory for Electronic Design Automation and represents one of the blocks of Integrated Power Meter SoC designed in the same laboratory [8].

The structure of proposed microcontroller block consists of core, memory blocks, and peripheral units. The peripherals are comprised of: three digital input/output parallel ports, liquid crystal display driver control circuit and several communication modules - two asynchronous universal receiver/transmitter blocks and one serial interface. Also, three timer/counter circuits are present.

The memory organization is similar to that of the industry standard 8052. Three main memory areas associated with the proposed microcontroller are: program memory (on-chip 8kB SRAM block), external data memory XRAM (physically consisting of on-chip 2kB SRAM block and I/O RAM block made of standard cells), and internal data memory IRAM (comprising of 256 Internal Dual port RAM and Special Function Register block).

To reduce the leakage power, the design was divided into three power domain areas and MTCMOS switches [2] were inserted into each power domain. MCU core, the peripherals and memories were embedded in distinct power domains – called CORE, PER and MEM respectively.

The microcontroller is provided with a few low-power operation modes so that user can choose the most appropriate for particular application. These modes switch off the power supply in the inactive microcontroller parts, significantly reducing the leakage current. The Power management unit is the part which is responsible for changing the power saving modes. Beside Active operating mode, microcontroller offers Standby mode. In Standby the core is switched off from the power supply, and the peripheral units and memories are kept powered.

The microcontroller was implemented using 90nm technology library and Synopsys EDA tool suite [9].

The novel design of microcontroller system was created which incorporates two identical 8052 cores, called the primary and spare cores. The primary and spare cores were implemented in different power domains areas with separate power supply lines. Since the spare core executes the same program code as primary unit, it shares with primary core RAM memory blocks, which are used for program and external data storing. The RAM memories represent the largest microcontroller blocks and their area is three times larger than the area of single microcontroller core [8]. Therefore, from area-efficiency point of view, the additional microcontroller core does not increase significantly the total chip occupied area.

The power consumption of primary core is reduced by decreasing the supply voltage to $V_{reduced}$=0.6V and frequency $f_{reduced}$ =60MHz. Besides, the spare core utilizes power gating for energy reduction. The spare core operates at nominal supply voltage V_{max}=1.1V and executes the instructions at the maximum clock frequency f_{max}=120MHz.

The power management unit was modified to support the operation of spare core. Also, the primary core was changed to enable toggling on and off the spare core.

The power estimation for the implemented microcontroller system is based on the information about static and dynamic power consumption of digital standard cells provided by the Synopsys 90nm technology files. These files were used during the synthesis process and after, during the placement and routing phases. The power consumption was obtained after layout implementation, and logical verification of the final layout netlist. The switching activity of the nets was recorded during logical verification for dynamic power estimation.

The Fig. 6 illustrates an example of schedule consisting of four tasks which is used for estimation of power consumption.

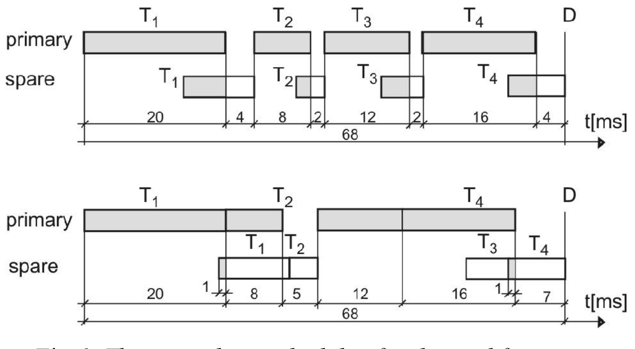

Fig.6. *The example: a schedule of tasks used for power estimation*

By using the Eq.4, the sum of the all inter-task periods and spare active times is calculated for task schedule which doesn't use the power reduction methodology. The sum is given by Eq.16.

$$\sum_{j=1}^{4}(r_j + a_j) = \frac{1}{2}\sum_{j=1}^{4} AET_j = \frac{20+8+12+16}{2} = 28 \qquad (16)$$

During schedule optimization process, the pairs of tasks T1, T2, and T3, T4 are concatenated. The sum of inter-task periods and spare active times is equal to the new value, given by Eq.17. The equations 9 and 15 are used in the calculation.

$$\sum_{j=1}^{4}(r_j' + a_j') = \frac{AET_1 - AET_2}{2} + \frac{AET_4}{2} = 14 \qquad (17)$$

The slack time SLT is equal to:

$$\sum_{j=1}^{4} r_j = D - \sum_{j=1}^{4} AET_j = 68 - 56 = 12 \quad (18)$$

The spare unit active time A, which contributes the most of energy consumption, is given by Eq. 19 and 20, for not-optimized and optimized design respectively.

$$\sum_{j=1}^{4} a_j = 16 \quad (19)$$

$$\sum_{j=1}^{4} a_j' = 2 \quad (20)$$

The relevant power consumption values are given in the Table 1.

Table 1. *The power optimization results*

before optimization				
Task	AET_i [ms]	a_i [ms]	Energy of primary core [μJ]	Energy of spare core [μJ]
T_1	20	6	26.58	58.8
T_2	8	2	10.632	19.6
T_3	12	4	15.948	39.2
T_4	16	4	21.264	39.2
Sum	56	16	74.424	156.8
after optimization				
Task	AET_i [ms]	a_i [ms]	Energy of primary core [μJ]	Energy of spare core [μJ]
T_1	20	1	26.58	9.8
T_2	8	0	10.632	0
T_3	12	0	15.948	0
T_4	16	1	21.264	9.8
Sum	56	2	74.424	19.6

In the example (Fig.6), before the tasks were concatenated, the energy consumption of microcontroller system was 231.224μJ. When proposed techniques were applied, the total energy was 94.024μJ; in other words, a 59.3% energy reduction compared to the non-optimized implementation. The energy of spare unit was reduced from 156.8 μJ to 19.6 μJ.

4. CONCLUSION

In this paper, the modification of standby sparing technique is proposed, in which the tradeoff between the fault tolerance and energy consumption is made. The technique relies on hardware redundancy and utilizes low power techniques: dynamic voltage scaling and power gating.

To validate the technique, the hard real time system was created, which incorporates two identical microcontroller cores. The first microprocessor core is called the primary unit and operates at lower voltage, reducing the power consumption. The other microcontroller core, called spare unit, exploits power gating to conserve the power and ensures the fault tolerant operation. The spare unit operates at higher voltage, but is most of the time inactive, reducing the total energy consumption.

The system is implemented in Synopsys 90nm technology. The main objective, which was to realize power efficient fault tolerant design, was fully reached.

ACKNOWLEDGEMENT

Results presented in this paper are part of achievements obtained within the project TR 32004 funded by the Serbian Ministry of Science and Technology Development.

REFERENCES

[1] M. Keating, D. Flynn, R. Aitken, A. Gibbons, K. Shi, Low Power Methodology Manual, Springer, 2007.

[2] P. Bipul, A. Agarwal, K. Roy "Low-Power Design Techniques for Scaled Technologies," *Integration, The VLSI Journal*, Vol. 39, Issue 2 (2006), pp. 64–89

[3] H. Kopetz, Real-time systems: Design principles for distributed applications, Kluwer Academic Publishers, 2002

[4] S. Poledna, Fault tolerant real-time systems: The problem of replica determinism, Kluwer Academic Publishers, 1996

[5] A. Ejlali, B. M. Al-Hashimi M. T. Schmitz P. Rosinger, S. G. Miremadi, "Combined Time and Information Redundancy for SEU -Tolerance in Energy Efficient Real-Time Systems," *IEEE Trans. VLSI Systems*, Vol.14 no.4 pp.323-335, April 2006.

[6] D.K. Pradhan, Fault tolerant computer system design, Prentice Hall, 1996

[7] P. Eles, V. Izosimov, P. Pop, Z. Peng "Synthesis of FaultTolerant Real-Time Systems" in Proc. Design, Automation and Test in Europe (DATE 2008), pp. 1117-1122, 2008

[8] B. Jovanović, M. Zwolinski, M. Damnjanović, "Low power digital design in Integrated Power Meter IC," *in Proc. of the Small Systems Simulation Symposium* 2010, Niš, Serbia

[9] Synopsys 90nm Generic Library for Teaching IC Design, http://www.synopsys.com, accessed April 2010