

An Evolutionary Power Management Algorithm for SoC Based EHW Systems

Lirong Tian

Aeronautic Computing Technology Research Institute
Xi'an China
tianlrcn@yahoo.com.cn

Tughrul Arslan

School of Engineering and Electronics
Edinburgh University, UK
Tughrul.Arslan@ee.ed.ac.uk

Abstract

This paper proposes a dynamic power management algorithm based on an evolutionary algorithm for minimisation of power on a system on chip (SoC) evolvable hardware (EHW) systems. The algorithm is specially designed to operate within an embedded microcontroller processor such as the ARM. The paper describes the algorithm, its implementation on an ARM based embedded processor and provides results demonstrating power consumption of the algorithm itself using different ARM instruction sets.

1. Introduction

The advance in Silicon technology has made it possible to integrate a variety of systems within a single chip environment. This implies that current EHW systems could have full systems functionality including multiple evolutionary algorithms each or all focusing on different processor fabrics within the chip. A crucial part of such EHW systems would be the operating system and power management system which ensure that operations within the system are executed such that the overall power consumption of the system on chip (SoC) EHW system is minimised. This is crucial feature as such SoC devices find use in power constrained environments such as satellites, autonomous vehicles, etc. Besides the limitation on power consumption, these systems usually have real-time requirement, which means every task should be scheduled within a specific time or deadline.

A real-time scheduler using evolutionary algorithm is presented in this paper. The algorithm is specially designed to operate within an SoC environment such as the one shown in figure 1. The architecture contains much of the basic functionality needed for a processor-based system, which include:

- AMBA advanced high-performance bus (AHB) and advanced peripheral bus (APB);

- 8K*32 internal memory;
- an external memory controller;
- two timers;
- a UART;
- an interrupt controller;
- a system controller;
- a system watchdog;
- general purpose I/O (GPIO);
- ten expansion ports (three AHB masters, three AHB slaves, four APB slaves).

The ARM architecture not only provides convenient interface for SoC hardware integration, it also provides flexibility for software designer to achieve the most reasonable solution for different application. Two instruction sets are provided in ARM architecture, these are ARM and Thumb instructions. Trade off can be made among the code density, performance and energy efficiency according to application specifications.

The paper describes the algorithm, its implementation on an ARM based embedded processor, and provides results demonstrating power consumption of the algorithm itself using different instruction set. The results can be used to tailor the algorithm for low power consumption.

2 Problem Description

Unlike a desktop system, an SoC systems are more likely to deal with periodic real-time tasks. For such a system, deadlines must be met, so a more accurate schedule is needed for the system. On the other hand as the tasks are periodic, it is possible to have an accurate schedule for these tasks in terms of energy saving. In a multi-tasking real-time system, different tasks usually use different devices, and to save energy, unused devices can be put into sleep

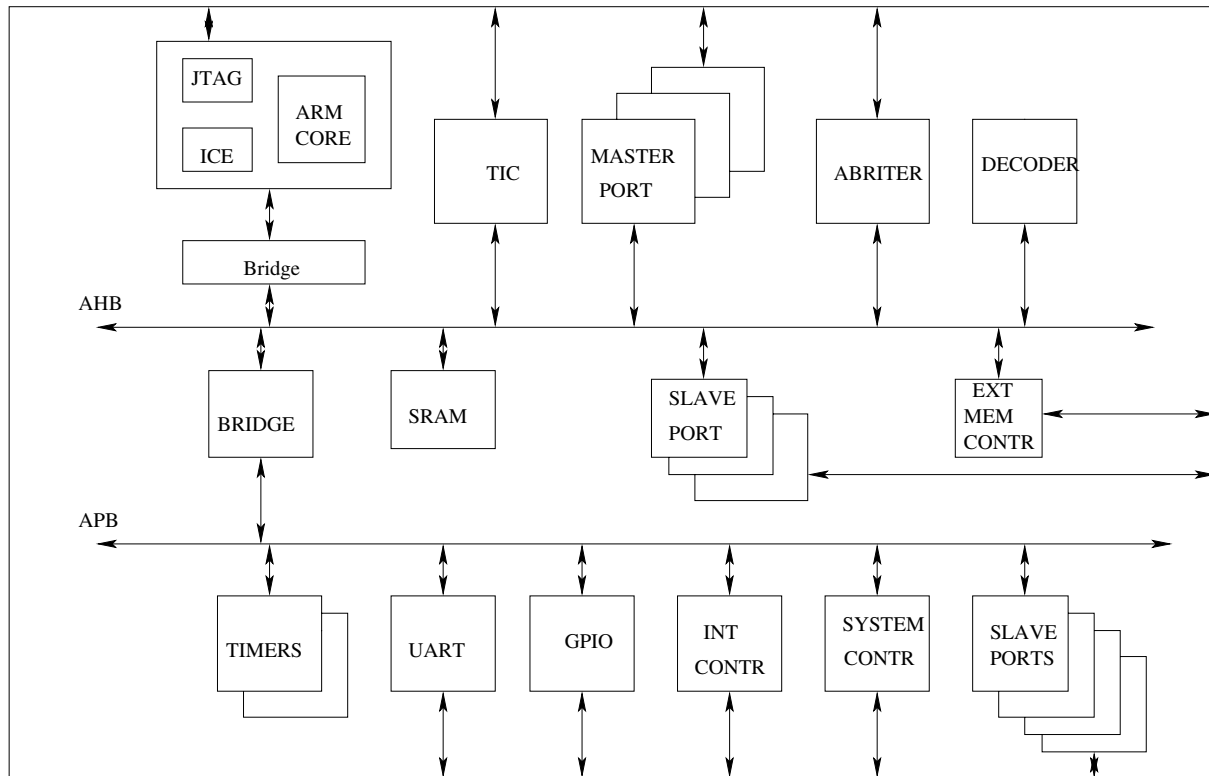


Figure 1. Architecture of the SoC platform

mode. However frequently switching a device between the active and sleep states also results in energy consumption. For further energy saving, a schedule for these tasks must reduce the transitions as much as possible. The scheduler we present here takes as input a predetermined set of tasks and a device usage list for each task. The output is a task execution sequence such that the power consumption of the devices is near minimized. We define a set $T = \{t_1, t_2, \dots, t_n\}$ of n periodic tasks. Each task t_i in T , has the following parameters, which are also shown in table 1:

- periodic time p_i ;
- execution time e_i ;
- device set K_i (which consists of all the I/O devices used by t_i).

Because the tasks are periodic, it is possible to find a global period H such that in every H interval, all of the tasks are repeated. If we get the optimal schedule for these tasks during the first H interval, the same schedule can be applied to other H intervals. Here we define the global period H as the smallest common multiple of the periods of all tasks in T . Associated with each task set T , there is a job set $J = \{j_1, j_2, \dots, j_m\}$ (where $m = \sum_{i=1}^n H/p_i$) consisting of all the executions of each task t_i in T during the first global

Table 1. Task Parameter Needed for the Schedule.

Task name	Execution time	Period	Device
t_1	e_1	p_1	K_1
t_2	e_2	p_2	K_2
\dots	\dots	\dots	\dots
t_n	e_n	p_n	K_n

period H . We assume that each task t_i is submitted once every p_i interval, and that it must be executed before the next p_i interval. The deadline for each job is taken as the sum of its submit time and the period p_i . If we define the delay as the difference between the job start time and its submit time, then to satisfy the deadline, the minimum and maximum delay should follow equation (1) and (2) respectively.

$$Mindelay_i = 0 \quad (1)$$

$$Maxdelay_i = deadline_i - submit_i - e_i \quad (2)$$

Table 2. Jobs to be Scheduled in the First Global Period.

Job	Submit time	Deadline	Min delay	Max delay
$j1_1$	0	p_1	0	$p_1 - \epsilon_1$
$j1_2$	p_1	$p_1 * 2$	0	$p_1 - \epsilon_1$
...
$j1_{l1}$	$p_1 * (l1-1)$	$p_1 * l1$	0	$p_1 - \epsilon_1$
$j2_1$	0	p_2	0	$p_2 - \epsilon_2$
$j2_2$	p_2	$p_2 * 2$	0	$p_2 - \epsilon_2$
...
$j2_{l2}$	$p_2 * (l2-1)$	$p_2 * l2$	0	$p_2 - \epsilon_2$
...
jn_1	0	p_n	0	$p_n - \epsilon_n$
jn_2	p_n	$p_n * 2$	0	$p_n - \epsilon_n$
...
jn_{ln}	$p_n * (ln-1)$	$p_n * ln$	0	$p_n - \epsilon_n$

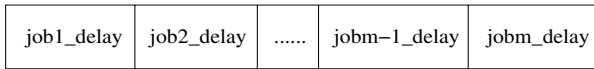


Figure 2. Chromosome Structure

Supposing that all tasks submit their first job at time 0, we can get the submit time, deadline, Mindelay and Maxdelay for each job in $J = \{j_1, j_2, \dots, j_m\}$, as shown in table 2.

3 Genetic Algorithm Implementation

3.1 The Chromosome

For our specific scheduling problem, the start sequence for multiple jobs must be generated simultaneously, so the multi-parameter coding method was adopted. In order to construct a multi-parameter coding, we simply concatenate many individual parameters. Each individual parameter is defined as a sub-chromosome, representing the delay between start time and submit time for each job. The chromosome structure is shown in figure 2:

All of the sub-chromosomes have the same length, and the length is decided by the maximum delay of all jobs. Suppose:

$$M = \text{Max}\{Maxdelay_1, Maxdelay_2, \dots, Maxdelay_m\}$$

We can find the sub-chromosome length SL using:

$$SL = \lceil \log_2(M + 1) \rceil \quad (3)$$

So, the chromosome length is:

$$L = SL * m \quad (4)$$

To satisfy the deadline requirement, from table 2 we can see that the delay time must be within [Mindelay, Maxdelay], but the value for the sub-chromosome can be decoded as any value within $[0, 2^{SL}]$ (where SL is the length of binary string for the sub-chromosome), so we need to map the decoded unsigned integer linearly from $[0, 2^{SL}]$ to the range of delay times within [Mindelay, Maxdelay]. The mapping equation is:

$$delay_i = (Maxdelay_i - Mindelay_i + 1) * x_i / (2^{SL} - 1) \quad (5)$$

Where x_i is the value of a sub-chromosome, $x_i \in [0, 2^{SL}]$.

3.2 Fitness Function and the Reduction of Device Power Consumption

To achieve the minimal power consumption, the schedule should (i) minimize the time for which the devices are powered up, (ii) maximize the time for which the devices are shutdown, (iii) minimize the number of device transitions. The schedule based on the following criterion. First, a device is put into a working state only when it is being used by the executing job which guarantee the minimized power up time and the maximized sleep time for a device. Second, device transitions between the working and the sleeping states are reduced as much as possible. This will reduce transition energy, because the frequent transitions between the different power states of a device is very energy consuming. From this point of view, if two or more jobs which are using the same device can be executed successively, it will reduce the transitions of the device and extend the sleep time for other devices. To find a schedule with many appearances of such a condition is our objective. In order to evaluate a schedule, we define the number of transitions avoided during one global period as its score. see equation (6) and (7).

If the deadline is not satisfied:

$$Score = 0; \quad (6)$$

If the deadline is satisfied:

$$Score = T_{\max} - T_s \quad (7)$$

Where T_{\max} is defined as the maximum number of transitions without reasonable schedule during one global period time H, and T_s represent the transitions needed by the evaluated schedule during the same period.

In order to simplify the representation, we assume that the power consumption for shut down and wake up is the same for all devices, and so do the times for shut down and

wake up. Here we define the power consumed by transaction for both wake up and shut down as P_t , and the time for transaction as T_t . The fitness is defined as the power saved by avoided transitions through reasonable schedule. It is calculated through the following equation:

$$Fitness = 2 * P_t * T_t * Score \quad (8)$$

3.3 GA Operation

Our GA operates through the following stages: initialization of the population, generation of new populations, and fitness evaluation. An initial population of random binary strings are produced by the system. Each member of the population is a potential scheme for the schedule. Each scheme is evaluated for its fitness according to the fitness function. The system keeps track of the current highest fitness value so far. The highest fitness value of each new generation is compared with this value and larger, the current highest fitness is replaced by the new value, otherwise the value is kept the same. To produce a new generation, we use Roulette Wheel selection to select the parents, then apply crossover and mutation operators to make out the new generation. The same process is repeated until the termination condition is satisfied. That is the algorithms converges.

The Generations needed to get the high fitness schedule is mainly dependent on the population sizes of GA. While for the fixed population size, the fitness value is dependent on the generations. The results are presented in figure 3 and figure 4 respectively.

Figure 3 shows the population size against the number of generation of the GA to obtain high fitness. It could be deduced from the figure that as the size of the population increases, the number of generation is reduced to obtain a same performance. However with population size 20 and above, the generations is a constant to reach the same performance. For this reason a population size of 20 was selected to provide a trade-off between memory size and performance. Figure 4 demonstrates a typical run for the GA. A solution is found in this case after 45 generations. The parameters used in our GA are:

- Population Size=20;
- Max Generation=50;
- Crossover Probability=0.6;
- Mutation Probability=0.01.

4. Power Analysis of the Scheduler for Different Instruction Set

In order to study the problem of energy consumption in the SoC systems, the variation in energy consumption due to

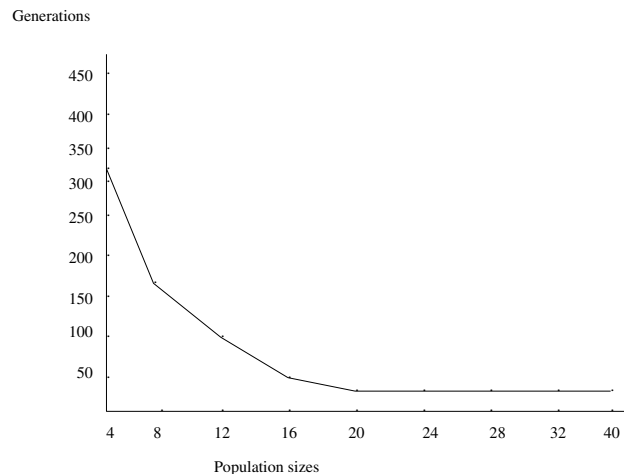


Figure 3. Graph to show population size against the number of generation of the GA

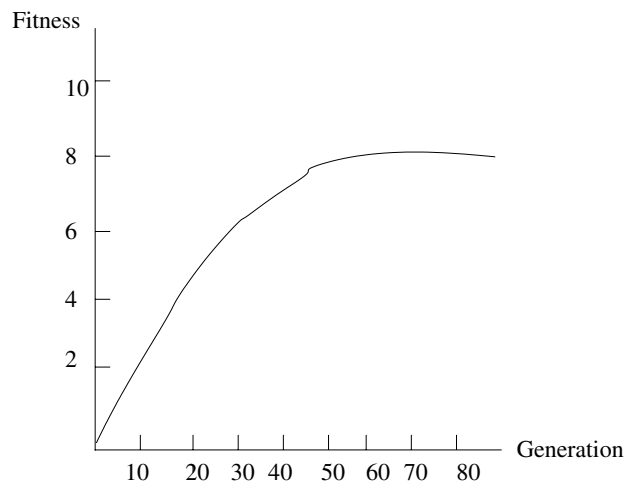


Figure 4. Graph to show generation against the fitness of the GA

variations in software is important. Since it is the software which drives the hardware in most systems. Decisions taken during software design has significant impact on the energy consumption of the processor.

Software impacts the system power consumption at various level. The structure of system software, the application source code and the compilation of the code into machine-level instruction, etc. all of these affect the power consumption of software. To optimize the power consumed by software, it is necessary to analyse the power consumption of the software first. In the following section we will analyse the power consumption of each subroutine for the power management algorithm described above, when using ARM and Thumb instruction set. The analysis is based on the data measured from the hardware platform. Because of the application-independent feature of the genetic algorithm, the result should be valuable to other designers who use the genetic algorithm in their systems.

4.1 ARM and Thumb Instruction Set

ARM processors are typical of RISC processors. Besides the 32-bit ARM instruction set, The ARM architecture defines a 16-bit instruction set called Thumb instruction set. The solution is in effect to reapply the rules of RISC and create a "really reduced" instruction set drawn from the original ARM set. The Thumb instruction set is a 16-bit subset of critical instruction in the 32-bit ARM instruction sets [9]. Each Thumb instruction has an exact equivalent ARM instruction. A small amount of logic added to the processor macrocell decompresses these 16-bit Thumb instruction back to their 32-bit ARM equivalents in real time. The reconstituted 32-bit ARM instruction is then executed as normal by the core.

Some comparisons have been made on the code density and performance between ARM and Thumb instruction set [9]. The results show that Thumb instruction set typically provides 30% improvement in code density over native ARM code. However, more 16-bit instructions than 32-bit instructions are required to execute the same compiled program, and this results in about 15% performance loss. The comparison of power consumption on these 2 instruction sets has not been mentioned so far because of the complexity of power consumption model. We'll compare the core power consumption of our power management algorithm when using different instruction set in the following sections.

4.2 Power Measurement Platform

There are several software power analysis techniques. Some of these are based on circuit-level or architectural-level simulation [11, 12], but these techniques are inaccur-

ate to estimate the power cost for a given piece of application software. In some cases these even cannot be applied due to lack of circuit-level or architecture-level information of the embedded processors [1]. As an alternative method, instruction-level analysis technique has been developed [2, 10, 14], and applied to some general-purpose commercial microprocessors [3, 4] and target DSP processor [1]. Recently a cycle-accurate energy measurement has been studied in [5, 13, 15]. In this project, the power consumption is measured directly on the hardware platform. This seems to be the most accurate and simplest way to evaluate energy consumption when the prototype is available.

The average power P consumed by a processor while running a certain program is given by $P=I*V_{dd}$, while I is the average current and V_{dd} is the supply voltage. The energy consumed by a program E is given by $E=P*T=I*V_{dd}*T$, where T is the execution time of the program. The V_{dd} is fixed to a specific application, and the current I and execution time T should be measured.

The measurement setup is shown in figure 5. The Multi-ICE is used to download programs from PC to the prototype. An Wavetek Meterman 5XL ammeter is used to measure the current consumed by the ARM7TDMI core. The reason for measuring the core current instead of the board current is that the board current depends on the configuration of each individual board. It is meaningless when the hardware configuration is changed. While the core current for a specific program is constant. It can be used by other designers as references in other systems. The time needed by each subroutine can be recorded by the timer which is implemented in the prototype. A timer start and stop instruction are added to the beginning and end of each subroutine respectively. The difference of the timer value at the 2 points is taken as the execution time of the subroutine.

The power consumption is proportional to the product of current and time. From the measured current and time, the core energy comparison can be made for each individual subroutine when using ARM and Thumb instruction set.

4.3 Experimental Results

Each subroutine is compiled into ARM instruction and Thumb instruction respectively, the code is downloaded to the prototype to execute. The current and execution time are measured. The results are shown in table 3. The table lists the core current, execution time and code size for each subroutine using ARM and Thumb instructions. It also lists the power ratio and code size ratio between 2 different instructions. The unit for current is mA, and execution time is presented as the counts of core system clock. The ratio for power is the ratio between the ARM instruction set and Thumb instruction set, while the ratio for code size is the ratio between the Thumb instruction set and ARM instruction

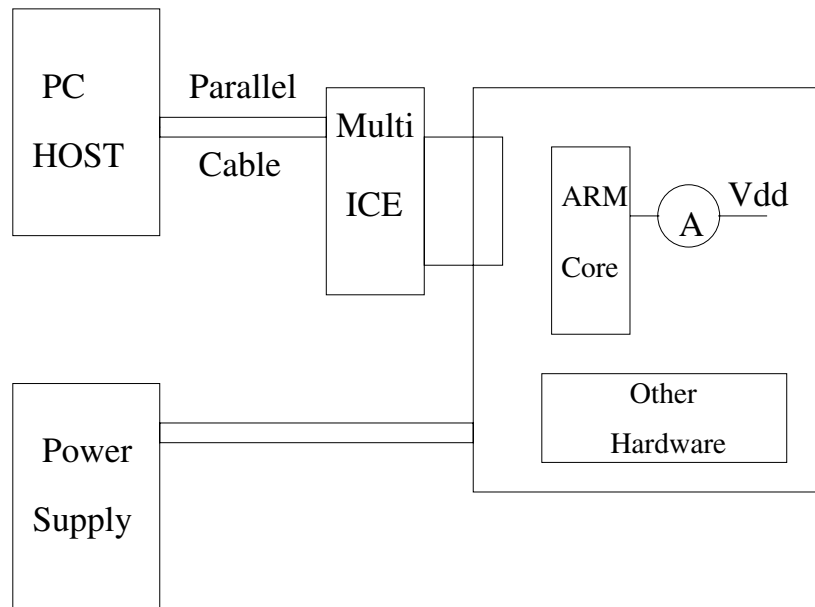


Figure 5. The Current Measurement Setup

Table 3. Currents and Execution Times for GA subroutines Using ARM and Thumb.

Name	ARM			Thumb			Power Ratio (%)	Code Size Ratio (%)
	Current (mA)	Time (CLK)	Code Size (Byte)	Current (mA)	Time (CLK)	Code Size (Byte)		
App_malloc	3.3	146	1835	3.1	253	1563	61	85
Advance_random	3.8	8051	3763	3.6	10751	3335	79	88
Objfun	3.9	38910	11739	3.6	46619	10023	90	85
Statistic	4.0	14875	4259	3.7	19134	4187	84	98
Initdata	4.7	1789	20	3.5	2555	12	94	60
App_init	3.4	403	488	3.4	559	324	72	66
Preselect	4.3	6334	3691	4.0	7826	3567	87	96
Initmalloc	3.6	14008	1987	3.1	21996	1667	74	83
Warmup_random	4.0	44572	4431	3.6	59904	4063	83	91
Randomperc	3.8	8281	4631	3.6	12159	4195	72	90
Randomize	4.0	43020	4519	3.6	57608	4219	83	93
Flip	3.7	8332	4759	3.6	11061	4556	77	95
Rnd	3.8	8375	6143	3.2	11064	5415	90	88
Initpop	3.7	948736	22508	3.6	1355264	15815	72	70
Mutation	3.9	427520	4859	3.6	627712	4591	74	94
Crossover	3.8	8704	6403	3.6	12544	6011	73	93
Select	3.8	17018	8219	3.6	24752	7871	72	95
Initialize	4.0	1001472	7439	3.5	1268480	6583	90	88
Generation	3.8	1014272	16343	3.6	1281792	15207	91	80

set.

5. Discussion and Conclusions

In this paper, we described an implementation of a genetic algorithm based real-time power management scheduler. The scheduler saves more energy for the SoC system, because it not only reduces the power by putting the unused device into sleep state, but also reduces the transition power by reducing the state transitions.

We also analyse the power consumption of the scheduler itself. From the experimental result above, It can be found that the ARM instruction set consumes less core power than Thumb instruction set. There are two reasons for this. The first reason is that the decomposition process of the Thumb instruction consumes extra power in the core. The second is because Thumb instruction is a subset of the ARM instruction set, where sometimes more than 2 thumb instructions are needed for one ARM instruction function. In such a case the one ARM instruction in fact is a package instruction for these Thumb instructions. Using the package instruction always leads to a reduction in energy. Because the average current for the packed instruction is only slightly more than the unpacked instructions, the reduction of execution cycle is more significant than the slight current increase which leads to a large overall energy reduction [1].

Though the core power consumption using ARM instruction set is less than Thumb, in the view of the whole system, the use of ARM instruction set is not always the energy efficient solution. Because the complete pipeline for an instruction include instruction/data fetch, decode and execution 3 stages, core consumption only considers the decode and execution stages. The trade-off can be made among different stages to get the optimized scheme. For example, we can use ARM instruction set when it is much more energy efficient than the Thumb. Otherwise we can choose Thumb instead of ARM. Because the Thumb code size is smaller than ARM, it will consume less power in code fetch stage. If we can get more code size benefit from the Thumb instruction set, which means the code size ratio is less than the core power ratio, then Thumb instruction set should be adopted. Otherwise the ARM instruction set should be adopted. From table 3, we can identify which instruction set should be applied to the individual subroutines with our GA scheduler. For example, routines such as Mutation and Crossover in table 4 are associated with more power reduction than memory reduction. For this reason the use of ARM instruction will provide more overall power reduction. The opposite is the case for the routines in table 5.

The results given here are based on our scheduler algorithms. Because most part of the genetic algorithm is application-independent except the objective function, the

Table 4. Subroutines using ARM instruction Set.

Name	Power Ratio	Code Size Ratio
App_malloc	61	85
Advance_random	79	88
Statistic	84	98
Preselect	87	96
Initmalloc	74	83
Warmup_random	83	91
Randomperc	72	90
Randomize	83	93
Flip	77	95
Mutation	74	94
Crossover	73	93
Select	72	95

Table 5. Subroutines using Thumb instruction Set.

Name	Power Ratio	Code Size Ratio
Objfun	90	85
Initdata	94	60
App_init	72	66
Rnd	90	88
Initpop	72	70
Initialize	90	88
Generation	91	80

results can be applied to other genetic algorithm applications.

References

- [1] Mike Tien-Chien Lee, Vivek Tiwari, Sharad Malik, and Masahiro Fujita: Power analysis and minimization techniques for embedded DSP software. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol 5, No.1, March 1997.
- [2] Kaushik Roy, Rabindra Roy, and Tan-Li Chou: Design of low Power digital system. *Designing Low Power Digital Systems, Emerging Technologies (1996)*, 1996 Pages: 137-204.
- [3] V. Tiwari, S. Malik, and A. Walfe: Power Analysis of Embedded Software: A First Step Towards Software Power Minimization. *VLSI Design J*, 1996.
- [4] V. Tiwari, S. Malik, and T-C Lee: Power Analysis of a 32-bit Embedded microcontroller. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, Volume:2 Issue:4, Dec 1994 Pages: 437-445.
- [5] Naehyuck Chang, Kwanha Kim, Hyung Gyu Lee: cycle-accurate energy measurement and characterization with a case study of the ARM7TDMI. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, Volume:10 Issue:2, Apr 2002 Pages: 146-154.
- [6] Sandy Irani, Sandeep Shukla, and Rajesh Gupta: Competitive analysis of dynamic power management strategies for systems with multiple power saving states. *Design Automation and Test in Europe Conference and Exhibition, 2002. Proceedings*, pp. 117-123, 2002.
- [7] Jinwoo Suh, Dong-In Kang, and S. P. Crago: Dynamic power management of Multiprocessor systems. *Proceedings International Parallel and Distributed Processing Symposium, IPDPS* pp. 97-104, 2002.
- [8] J. Lorch and A. Smith: Software strategies for portable computer energy management. *IEEE Personal Communication Mag*, vol 3, No. 5 pp. 60-73 June 1998.
- [9] Liam Goudge and Dave Jaggar: Low power RISC Trims system space requirement. *Advanced RISC Machines Inc.*
- [10] David Sarta, Dario Trifone, and Giuseppe Ascia: A Data Dependent Approach to Instruction Level Power Estimation. *IEEE Alessandro Volta Memorial Workshop on Low-Power Design*, 1999, pp. 182-190.
- [11] R. Yu Chen, M. J. Irwin and R. S. Bajwa: An Architectural level power estimator. *ISCAW*, June 1998.
- [12] R. Yu Chen, M. J. Irwin and R. S. Bajwa: validation of an Architectural Level Power Analysis Technique. *Design Automation Conference* June 1998, pp. 242-245.
- [13] Tajana Simunic, Luca Benini and Giovanni De Micheli: Cycle-accurate simulation of energy consumption in embedded systems. *Design Automation Conference* June 1999, pp. 867-872.
- [14] Peggy Laramine: Instruction Level Power Analysis and Low Power Design Methodology of a Microprocessor. Master Thesis, U. C. Berkeley.
- [15] Naehyuck Chang and Kwan-Ho Kim: Real-Time Per-cycle Energy Consumption Measurement of Digital System. *IEE Electronics Letters*, 2000.
- [16] Huzefa Mehta, Robert Michael Owens and Mary Jane Irwin: Techniques for Low Energy Software. *Low Power Electronics and Design, 1997 International Symposium* pp. 72-75.
- [17] S. Osborne, A. T. Erdogan, T. Arslan, D. Robinson: A Bus Encoding Architecture for Low Power Implementation of an AMBA based SoC Platform. *IEE Proceedings-Computers and Digital Techniques (Special Issue on Low Power System on chip)*, Volume:149, Issue:4, July 2002, pp.152-156. *Low Power Electronics and Design, 1997 International Symposium* pp. 72-75.