FAULT INDULGENT IN EMBEDDED MEMORY USING WCET Real-Time Embedded System

D.DIVYA[1], A.KARTHIKEYAN[2] and G.PANNEERSELVAM[3]
PG Scholar, Embedded System Technologies of VelTech MultiTech DR.Rangarajan DR.Sakunthala Engg. College,
[2&3] Assistant Professor, EEE Department, VelTech MultiTech DR.Rangarajan DR.Sakunthala Engg. College, Chennai.
E-mail: ddivya18@gmail.com[1]

Abstract— **The growing density of integration and the increasing percentage of system-on-chip memory occupied by embedded programs have led to an increase in the expected amount of power consumption. In order to reduce the integrity and iterations of the embedded programs the WCET has been implemented. By monitoring the Worst Case Execution time we can reduce the clock cycles required by each instruction of the program, which analogously increases the memory consumption based on both Ram and Rom memory in the embedded system and also power consumption criteria. In this paper, a compiler level optimization, namely WCET-aware rescheduling register allocation, is proposed to achieve WCET minimization for real-time embedded systems. The novelty of the proposed approach is that the effects of register allocation, instruction scheduling, and cluster assignment on the quality of generated code are taken into account for WCET minimization. Three compilation processes are integrated into a single phase balanced result obtained with 6kbytes of ROM reduction from 8kbytes.**

Keywords— embedded; memory; worst case execution time (WCET); fault tolerance; Instruction-level Parallelism (ILP)

I. Introduction

Embedded systems play a vital role in many areas of human life. Cell phones, PDAs, home appliances such as washing machine and satellites are only few examples of devices with a processor embedded in them. A large group of these systems are portable battery powered devices that have a limited source of energy. This makes the energy consumption a prominent characteristic. Since software is responsible for a large portion of the system energy consumption, an accurate energy model [7] is necessary for the system energy optimization.

Instruction-level Parallelism (ILP) is a critical technique used in computer architecture for processor [8] and compiler design. ILP can perk up the program execution performance by causing individual machine operations to execute in parallel. There are two modes of operation, sequential mode and parallelism mode. Macros and Pointers are two important functions which is been used to estimate the memory size of RAM and ROM. Macros is used for such as addresses and array where the pointers are using without addresses that is addresses are hold in this process. This process can be useful in avoiding critical state of affairs in worst case of execution.

To reduce WCET, a program is the maximal time execution can ever exhibit. Since WCET is one of the most important attributes of real-time systems, compiler level optimization of a program *P* should be conducted along the Worst-Case Execution Path [1]. In addition, considering the characteristics of the clustered VLIW architecture, it is important to take into account the phase ordering problem of register allocation, scheduling and cluster assignment for WCET reduction so that a program's WCET can be minimized.

II. Average Case Execution Time

Current register allocators are mostly Average-Case Execution Time (ACET) oriented. These traditional allocators may have negative effect on the code quality for real time systems considering real-time constraints. Nowadays real-time constraint must be safely met to ensure the correctness of real time properties so Worst-Case Execution Time (WCET) has fetching a best suit in today's scenario. For embedded systems with clustered VLIW architecture [11][15], instructions can be executed in parallel and in different clusters. An efficient schedule considering instruction-level parallelism (ILP) is critical to the system performance of such VLIW style machines. Register allocation, a key activity during the compilation process, aims at multiplexing a large number of target program variables onto a small number of physical registers. In particular for embedded systems with limited number of registers, an efficient allocation solution contributes significantly to the system performance.

III. VLIW Archicture

The traditional register allocation phase for clustered VLIW systems is implemented independently from two other key phases: cluster assignment and instruction scheduling. There is a phase ordering problem among these phases: Independently performing register allocation, scheduling and cluster assignment [5] could have negative effect on the other phases. This phase ordering problem of register allocator and scheduler could also have negative effect [13-14] on the code quality, resulting in an even worse WCET [9]. Therefore, conventionally register allocation technique is not suitable for real-time embedded systems with clustered VLIW architecture. This paper proposes a novel register allocation

ISBN No.978-1-4799-3834-6/14/$31.00©2014 IEEE

technique to minimize WCET for real-time embedded systems with clustered VLIW architecture is been implemented.

IV. Instruction Level Parallelism

An instruction-level parallelism (ILP) [5] can highly enhance a program's performance. However, this property requires more functional units and registers. This increase in resources directly affects the cycle time of processor since the register file access time can become the bottleneck of delays. One solution of this problem is to cluster functional [9] units and registers into groups, where functional units in the same cluster have direct access to the local registers within the cluster. Any access to the other clusters needs the assistance of global communication bus. In this way, the register access time delay derived from VLIW design can be effectively masked. There are a set of functional units in each cluster, which can access a set of local registers. Each cluster is connected by the communication bus. A *Move* operation [2] can be inserted in the schedules so that data can be transferred among the clusters.

V. Worst Case Execution Time

Worst-case execution time (WCET) is one of the most important metric in real-time embedded system design. The creativity of the proposed approach is that the singular property of instruction scheduling, register allocation, and cluster assignment on the quality of spawned code are taken into elucidation for WCET minimization.

A. Emerges of WCET

The Worst-Case Execution Time (WCET) [1-4] of a computational task takes a maximum time of length the task could take to execute on a specific hardware platform. Worst case execution time is derived by schemes either by measurement or by analysis. Both the methods are contradiction to each other and have downside model in their own way. Worst case execution time is characteristically used in reliable real-time systems, where indulgencing the worst case timing behavior of software is essential for reliability or correct functional behavior. While WCET is potentially applicable to many real-time systems, in practice an assurance of WCET is mainly used by real-time systems that are related to high reliability or safety.

Worst case execution is mainly classified into Measurement-Based Worst-Case Execution Time (WCET) Analysis; Static Program Analysis based Worst Case Execution Time. For safety critical systems both the approaches are useful.

It requires functional and temporal correctness [13][4]. In this real-time system is broken into number of tasks and tasks are scheduled according to their temporal parameters, it attends circumventing catastrophe. Static analysis tool works at a high-level [14] to conclude the structure of a program's task, working on either a part of source code or by disassembling binary executable code. They also work at low-level [14], using timing information real hardware but that the task will execute on, with all its specific features.

By mingling those two different levels, the tool efforts to gives an upper bound on the required time to execute a given task on a specified hardware platform. In this paper Static Program Analysis based Worst Case Execution Time is taken into consideration for Multi-core hardware [4][14] and safety execution in RTOS.

B. Programming analysis in WCET

The programming approach of WCET is of two types if basic blocks are known: Timing schema and Integer Linear Programming (ILP). Timing schema takes longer time in execution path in control flow analysis. Integer Linear Programming is modern approach it overcomes all disadvantage of timing schema.

The technique engross two distinct activities for analyzing its worst-case execution time

1) Directed graph of basic blocks is been decomposed from the process. The basic blocks symbolize straight-line code.
2) Machine code is used analogous to basic blocks in above technique.

The engrossing depends on basic blocks, once its known total directed graph can be malformed. This paper C program is used which is converted to machine code has basic blocks.

VI. Design Methodology

This paper deals with the instruction level of energy estimation model. The energy estimation can be done from RAM and ROM. The electronic applications which is used here is load. Lamp is taken as the load. Two DC motors are used to run the module, one is stepper motor. Stepper is connected to the integrated circuits and DC motor is connected to the relay. In this paper the main goal is to design and get an accurate result for yet energy estimation to ensure the memory capacity.

A. WCET-aware Rescheduling Register Allocation Technique

The compiler approach is to trim down WCET for clustered VLIW architectures. The WCET of a program *P* is the maximal time *P*'s execution can ever reveal. It is equal to the length of the longest execution path from the start node to the end node in *P*'s control flow graph (CFG) [1][2]. Where Control flow graph (CFG) in Fig.1 precedes an input to this technique. In view of the fact that WCET is one of the most significant attributes of real-time systems, compiler level optimization of a program *P* should be conducted along the Worst-Case Execution Path (WCEP) as shown in Fig.1, which is distinct as the longest execution path of *P*. With every optimization process reduces the current WCEP by compiler optimization, the new WCEP is generated which may be changed. Therefore, WCET optimization should be attentive of the change of WCEP to ensure the effectiveness of WCET reduction. In addition, considering the characteristics of the clustered VLIW architecture [4], it is important to take into account the phase ordering problem of register allocation, scheduling and cluster assignment for WCET reduction so that a program's WCET can be minimized. The suitable CFG node along the current WCEP is static WCEP analysis is conducted through the CFG of a given program *P*, where an aggressive initial schedule and an initial cluster assignment solution have been generated without considering register pressure this analysis should carefully selected for better register allocation, where the re-schedulability of each node is taken into account which is very thorny, costly and other instruction would be inserted in interior.

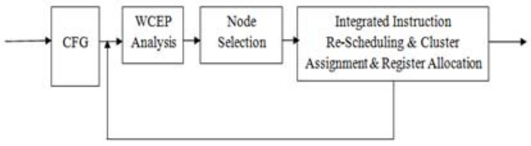

CFG
WCEP Analysis
Node Selection
Integrated Instruction Re-Scheduling & Cluster Assignment & Register Allocation

Fig. 1. WCET-aware Rescheduling Register Allocation Technique

Otherwise, *Load* and *Store* operations can be inserted. Then the WCEP information is updated and next CFG node is selected for processing. This iteration continues until all CFG nodes are processed. Motivational [1] is best example for demonstrating WRRA which uses *Moves, Load* and *Store* operations.

TABLE I. THE WCET UNDER DIFFERENT APPROACHES

Approach	WCET under the Approach	Reduction Ratio
Traditional	55	-
WRRA	37	33%

Traditional approaches in Table I were used to reduce worst case execution earlier coloring graph [10], block based re-scheduling [12], local and global variable allocation [15] and other WRRA approach WCET is measured in schedule length along WCEP. In traditional register allocation a *Load* takes four unit times and a *Store* takes two unit times [1] and other operations are assumed to take one unit of time. Traditional approach have maximum number of worst case under execution which cannot be reduced, the proposed WCET-aware rescheduling register allocation (WRRA) approach achieves the WCET reduction effectively by comparing two results of traditional register allocation, WRRA can lead to a saving of WCET by 33% effectively.

B. *Proposed system implementation using WCET*

By implementing WCET in embedded programming criteria register handling can be widely reduced so that garbage handling in the memory unit is getting easier, so that we can avoid failure occurrence in the embedded Memory (registers) WORST-CASE Execution Time (WCET) in Fig.2, is an important real-time constraint in real-time embedded systems, which must be safely met to ensure the correctness of real-time properties instructions can be executed in parallel and in different clusters [9]. An efficient schedule considering instruction-level parallelism (ILP) is decisive to the system performance of such VLIW [4][8] style machines. Register allocation, a key activity during the compilation process, aims at multiplexing a large number of target program variables onto a small number of physical registers.

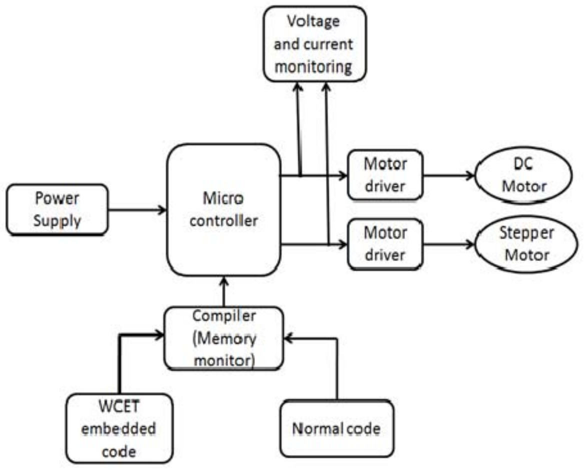


Fig. 2. WCET implementation system

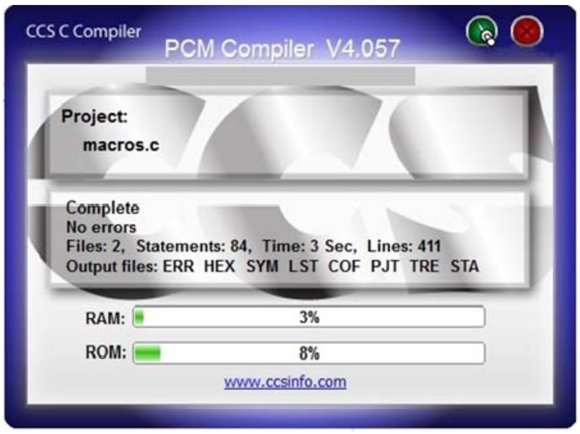
In proposed embedded memory is executed in two different codes to be precise with normal code and embedded code by instruction level parallelism is compiled and compared to monitor the memory used in two distinct approaches using microcontroller in Worst case execution time- aware rescheduling allocation technique to achieve better performance in critical and hazard situations.

VII. Simulation

In this paper mainly execution timing difference and memory allocation in ROM is been imitated by simulation using two different kind of programming in C. The three compilation processes are integrated into a single phase to obtain a balanced result. The segment shows the simulation results achieved by macros and pointer in critical circumstances

A. *Upshot using MACROS Programming*

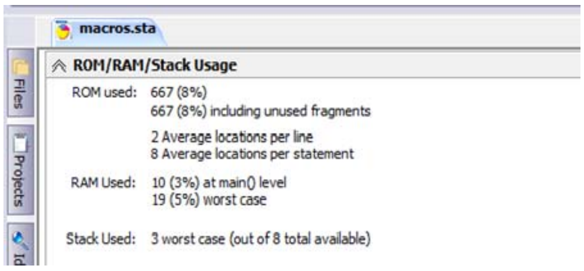
Macro is a facility provided by the C preprocessor, by which a token can be replaced by the user-defined sequence of characters. Every time when a macros function is called it access ROM memory for execution which intent increases garbage value in system automatically slow down the system.



CCS C Compiler
PCM Compiler V4.057
Project:
macros.c
Complete
No errors
Files: 2, Statements: 84, Time: 3 Sec, Lines: 411
Output files: ERR HEX SYM LST COF PJT TRE STA
RAM: 3%
ROM: 8%
www.ccsinfo.com

Fig. 3. Macros simulation for worst case

By this approach of programming 3% of RAM memory is used and 8% of ROM is used as shown in Fig.3. Remembrance used in ROM should be reduced for efficient embedded system memory allocation. So, this fault is tolerated in forthcoming method.



macros.sta
ROM/RAM/Stack Usage
ROM used: 667 (8%)
667 (8%) including unused fragments
2 Average locations per line
8 Average locations per statement
RAM Used: 10 (3%) at main() level
19 (5%) worst case
Stack Used: 3 worst case (out of 8 total available)

Fig. 4. ROM/RAM/Stack usage for Macros

In ROM memory allocation 667bytes are been used including unused fragment segment in system as the program content are not stored in same segment location in sequential order. It occupies 8% of total ROM memory.

The memory stored according Fig.4. 2bytes are used at an average of bytes locations per line. 8bytes are used at an average of bytes locations per statement.

In RAM memory 10bytes are used for executing main function which is 3% of total ram memory. And 19bytes i.e. 5% of worst case are decipher.

Stack usage is determined by the function usage of while (1) loop. In this Program only one loop is used. There are three processes worst cases are used out of 8 totals available

1) *MACROS Parameters Analysis*

Table II gives an idea about memory allocations for each and every function used in this programming style.

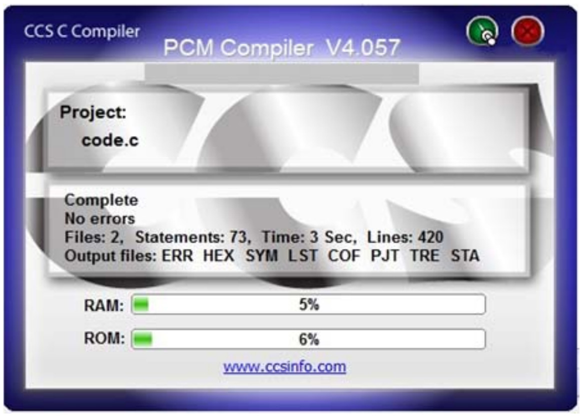
First column in Table II gives an idea about ROM memory, the most important allocation are delay_ms() function is 21bytes, main() function is 311bytes, printf_d() function is 105bytes, and const63 function defined at 63th line is 27bytes.

TABLE II. FUNCTION BREAKDOWN ANALYSIS FOR MACROS

ROM	%	RAM	Volume	Diff	Functions
21	3	1			@delay_ms1
47	7	3	236	2.3	lcd_cmd
21	3	0	106	1.3	lcd_init
47	7	3	231	2.3	lcd_data
311	47	3	1686	4.4	MAIN
12	2	0			@const61
17	3	0			@const62
27	4	0			@const63
21	3	3			@DIV88
105	16	6			@PRINTF_D
17	3	0			@const70
17	3	0			@const71

Second column represents percentage of RAM usage, the most important allocation are delay_ms() function is 3%, main() function is 47%, printf_d() function is 16%, and const63 function defined at 63th line is 16% of total RAM memory. Third column in Table II gives an idea about RAM memory, the most important allocation are delay_ms() function is 1byte, main() function is 3bytes, printf_d() function is 6bytes and some doesn’t use RAM memory. Fourth column is volume of bytes combining both memories. Fifth column is the difference between RAM and ROM memory. Last column is list of all the functions used in program from top to bottom.

B. *Upshots using POINTER Programming*

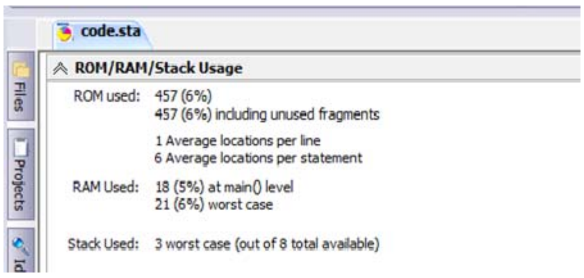


CCS C Compiler
PCM Compiler V4.057
Project:
code.c
Complete
No errors
Files: 2, Statements: 73, Time: 3 Sec, Lines: 420
Output files: ERR HEX SYM LST COF PJT TRE STA
RAM: 5%
ROM: 6%
www.ccsinfo.com

Fig. 5. Pointer simulation for worst case

A *Pointer* is a variable that holds the address of a variable or a function. A Pointer is a powerful feature that adds enormous power and flexibility to C. So this type of programming uses RAM memory rather than ROM memory for execution, since content in RAM memory gets reset after every start-up the garbage value accumulation is reduced in worst case.

In this approach of programming 5% of RAM memory is used and 6% of ROM is used as shown in Fig.5. By this technique ROM memory is reduced this makes an embedded system memory more efficient than preceding technique. This technique is my WCET [5] designed for simulation.



code.sta

ROM/RAM/Stack Usage

ROM used: 457 (6%)
457 (6%) including unused fragments
1 Average locations per line
6 Average locations per statement

RAM Used: 18 (5%) at main() level
21 (6%) worst case

Stack Used: 3 worst case (out of 8 total available)

Fig. 6. ROM/RAM/Stack usage for Pointers

In ROM memory allocation 457bytes are been used including unused fragment segment in system because the program content are not stored in same segment location in sequential order. It occupies 6% of total ROM memory.

The memory stored according Fig.6. 1byte is used at an average of bytes locations per line. 6bytes are used at an average of bytes locations per statement.

In RAM memory 18bytes are used for executing main function which is 5% of total ram memory. And 21bytes i.e. 6% of worst case are decipher.

Stack usage is determined by the function usage of while (1) loop. In this Program only one loop is used. There are three processes worst cases are used out of 8 totals available in stack.

1) *POINTER Parameters Analysis*

TABLE III. FUNCTION BREAKDOWN ANALYSIS FOR POINTERS

ROM	**%**	**RAM**	**Volume**	**Diff**	**Functions**
21	5	1			@delay_ms1
47	10	3	236	2.3	lcd_cmd
21	5	0	106	1.3	lcd_init
47	10	3	231	2.3	lcd_data
244	53	11	1539	5.5	MAIN
12	3	0			@const66
19	4	0			@const67
12	3	0			@const68
11	2	0			@const69
19	4	0			@const70

Table III gives an idea about memory allocations for each and every function used in this programming style. The details of columns are same as mentioned Table II.

C. *Comparing Two Programming Techniques*

From Tables II & III parameters analyzation is made for every functions used in two different programming techniques and demonstrated in Table IV. Finally it is verified that pointer based program is more efficient in worst case execution time than macros programming. Foremost parameter is compared and recognized in Table IV.

TABLE IV. COMPARISON BETWEEN MACROS AND POINTERS PARAMETERS

	ROM	%	RAM	Volume	Diff	Functions
POINTER	21	5	1			@delay_ms1
	47	10	3	236	2.3	lcd_cmd
	21	5	0	106	1.3	lcd_init
	47	10	3	231	2.3	lcd_data
	244	53	11	1535	5.5	MAIN
MACROS	21	3	1			@delay_ms1
	47	7	3	236	2.3	lcd_cmd
	21	3	0	106	1.3	lcd_init
	47	7	3	231	2.3	lcd_data
	311	47	3	1686	4.4	MAIN

Table IV be confirmation for the variation between two technique namely macros and pointer up to main function since these are always fasten with programs, other functions also endorses the difference which is based on application.

First column in Table IV gives an idea about ROM memory, main function uses 311bytes in macros programming and 244bytes in pointer programming. This shows comprehensible difference in ROM memory. Second column represents percentage of RAM usage, main function uses 47% in macros programming and 53% in pointer programming of total RAM memory. Third column in Table IV gives an idea about RAM memory, the main function uses 3bytes in macros programming and 11bytes in pointer programming. This shows comprehensible difference in RAM memory and some function doesn't use RAM memory. Fourth column is volume of bytes combining both memories. The main function uses 1686bytes in macros programming and 1539bytes in pointer programming including all segment allocations. This shows comprehensible difference in memory. Fifth column is the difference between RAM and ROM memory. The Diff is more macros i.e. 5.5 and less in pointer 4.4 used by main function. Last column is list of all the functions used in program from top to bottom.

D. *Circuit designed for Worst Case Execution in Proteus VSM*

The designed is such a way that it should have some parallelism in execution and characterizes worst case in complication. This circuit shown in Fig.7 has three processes namely a Lamp, a DC motor and Stepper motor. All three instruction-level parallelism processes are connected to PICC 16F877A microcontroller and LCD display board. The lamp and DC motor is connected via a Relay to PortB RB0 pin of microcontroller. Stepper motor is connected to PortB RB1-RB4 pins of microcontroller. LCD display uses PortD for register banks RD0-RD3 pins. The relay runs lamp and dc motor with period time difference such that only one runs at a time. Parallelism is shown when DC motor runs simultaneous Stepper motor also runs. This is hindrance application and the faults are likelihood to occur as well as ruin the application. This worst case execution time (WCET) can trim down by indulgent programming for proficient embedded memory.

LCD display in Fig.7, shows the usage of RAM and ROM memory for the worst case execution time.

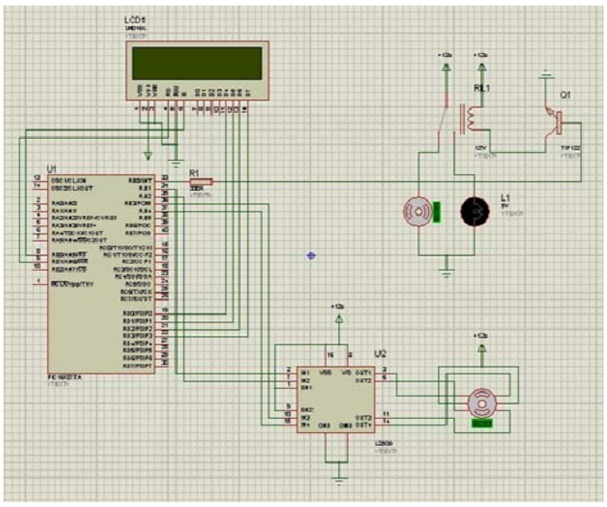


Fig. 7. Designed circuit

VIII. Conclusion

This paper proposes a compiler level optimization technique, namely WCET-aware re-scheduling register allocation for WCET reduction on real-time embedded systems with instruction level parallelism. A processor that executes every instruction in a non-pipelined scalar architecture may employ processor resources inefficiently, potentially leading to deprived performance. The performance can be enhanced by executing different sub-steps of sequential instructions simultaneously (i.e. *pipelining*), or even executing multiple instructions exclusively concurrently as in superscalar architectures. Auxiliary enhancement can be achieved by executing instructions in an order different from categorize they materialize in the program; this is called out-of-order execution. The prospect work of this project is to implementation in hardware by using scheduling in RTOS and LIN protocol to obtain the best result.

Acknowledgment

Dr.E.N.Ganesh at veltech is gratitude for supporting this work. The authors acknowledge the Department of Science and Technology; India for sponsoring FIST project (SR/FST/College-189/2013).

References

[1] H. Falk. "WCET-aware register allocation based on graph coloring," in *DAC '09: Proceedings of the 46th annual design automation conference,* 2009, pp. 726–731.

[2] P. Lokuciejewski, H. Falk, and P. Marwedel, "WCET-driven cachebased procedure positioning optimizations," in *ECRTS '08: Proceedings of euromicro technical committee on real-time systems,* 2008 pp.321–330.

[3] Vadim Smolyakov, Student Member, IEEE, Glenn Gulak, Senior Member, IEEE, Timothy Gallagher, *Member, IEEE*, and Curtis Ling, *Senior Member,* "Fault-Tolerant Embedded-Memory Strategy for Baseband Signal Processing Systems" IEEE Transcations (VLSI), Vol. 21, No. 7, July 2013

[4] T. Liu, M. Li, and C. J. Xue, "Minimizing WCET for real-time embedded systems via static instruction cache locking," in *RTAS '09: The fifteenth IEEE real time and embedded technology and applications symposium,* 2009, pp. 35–44.

[5] D. Sciuto, C. Silvano and V.Zaccaria(2012), "An Instruction-Level Energy Model for Embedded VLIW Architectures", IEEE Transcations

[6] Y. Zorian, "Embedded-memory test and repair: Infrastructure IP for SoC

yield," in *Proc. Int. Test Conf.*, 2002, pp. 340–348.

[7] I. Puaut and C. Pais, "Scratchpad memories vs locked caches in hard real-time systems: a quantitative comparison," in *DATE '07: Proceedings of design, automation and test in Europe,* 2007, pp. 1484–1489.

[8] K. Kailars, A. Agrawala, and K. Ebcioglu, "Cars: a new code generation framwork for clustered ILP processors," in *HPCA '01: Prodeedings of the 7th international symposium on high-performance computer architecture,* 2001, pp. 133–143.

[9] J. Sanchez and A. Conzalezor, "Instruction scheduling for clustered VLIW architectures," in *Proceedings of 13th international symposium on system synthesis,* 2000, pp. 41–46.

[10] M. D. Smith, N. Ramsey, and G. Holloway, "A generalized algorithm for graph-coloring register allocation," in *PLDI '04: Proceedings of the ACM SIGPLAN 2004 conference on programming language design and implementation,* 2004, pp. 277–288.

[11] M. Poletto and V. Sarkar, "Linear scan register allocation," in *ACM transactions on programming languages and systems,* 1999, pp. 895–913.

[12] S. Hack, D. Grund, and G. Goos, "Register allocation for programs in SSA form," in *CC '06: International conference on compiler construction,* 2006, Vol. 3923, pp. 247–262.

[13] V. S. Lapinskii and M. F. Jacome, "Cluster assignment for highperformance embedded VLIW processors," in *ACM transactions on design automation of electronic systems,* 2002, pp. 430–454.

[14] R. Leupers, "Instruction scheduling for clustered VLIW dsps," in PACT '00: Proceedings of the international conference on parallel architecture and compilation techniques, 2000, pp. 291–300.

[15] D. W. Goodwin and K. D. Wilken, "Optimal and near-optimal global register allocation using 0–1 interger programming," in *Software: practice and experience,* 1996, pp. 929–965.

ISBN No.978-1-4799-3834-6/14/$31.00©2014 IEEE