Volume 2, No. 3, May-June 2011

ISSN No. 0976-5697

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Implementing ARM7TDMI-S through EXPRESSION ADL

Veena Ramnani
Department of Computer Science
Mohanlal Sukhadia University Udaipur,
Rajasthan, India
ramnaniv@yahoo.com

Dr. Manoj Kumar Jain*
Department of Computer Science
Mohanlal Sukhadia University Udaipur,
Rajasthan, India
manoj@cse.iitd.ernet.in

Abstract: The embedded systems designs critically require a fast and automated architecture exploration methodology. Furthermore, shrinking time-to-market has created an urgent need to design hardware and software in parallel. Architecture Description Languages (ADLs) are effectively used for designing Application Specific Instruction set Processors (ASIP).This paper focuses on EXPRESSION ADL.
EXPRESSION utilizes MIPS 4K like processor called **acesMIPS** and supports a mixed behavioral/structural representation of the architecture.It can capture a processor- memory architecture description and generate a compiler and simulator automatically from this description the purpose of this paper is to simulate ARM7TDMI-S through EXPRESSION and in the process analyze the framework for it structural and instruction set capabilities. It was found that EXPRESSION has several shortcomings and is not able to simulate ARM7TDMI-S.

Keywords: Retarget able Compiler, ASIP, Design Space Exploration, ADL, EXPRESSION tool-kit.

I. INTRODUCTION

ASIP design (Jain, M.K. et al 2005, Jain, M.K. et al 2007) allow a wide range of memory organizations and hierarchies to be explored and customized for the specific embedded application. The ASIP designer is faced with the task of rapidly exploring and evaluating different architectural and memory configurations. Furthermore, shrinking time-to-market has created an urgent need to automatically generate compiler/simulator tool-kit. Retargetable compilers are a promising approach for automatic compiler generation. A compiler is said to be 'retargetable' if it can be used to generate code for different processor architectures by reusing significant compiler source code. This has resulted in a paradigm shift towards a language-based design methodology using Architecture Description Language (ADL) for embedded System-on-Chip (SOC) optimization, exploration of architecture /compiler co-designs and automatic compiler/simulator generation.

An ADL (Mishra, P. et al 2005) for design space exploration should capture both structural and behavioral aspects of the system. In a mixed-level ADL the system designer is able to make structural changes like varying the number of functional units, register files, data paths etc and behavioral changes by varying the Instruction Set by adding/deleting operations, changing data types etc. EXPRESSION follows a mixed-level approach to enable changes to structure or Instruction Set (IS) or the memory subsystem.

II. RELATED WORK

Architecture description languages (ADL) (Mishra, P. and Dutt, N., 2005) are used to perform early exploration, synthesis, test generation, and validation of processor-based designs. ADLs are used to specify programmable architectures. The specification can be used for generation of a software toolkit including the compiler, assembler, simulator and debugger. The application programs are compiled and simulated, and the feedback is used to modify the ADL specification with the goal of finding the best possible architecture for the given set of applications. The ADL specification can also be used for generating hardware prototypes under design Constraints such as area, power and clock speed. Several researches have shown the usefulness of ADL-driven generation of functional test programs and test interfaces. The specification can also be used to generate device drivers for real-time operating systems.

Retargetable compilers MSSQ, RECORD, CHESS, EXPRESS, SPAM, LISATek employ ADLs to achieve retargetability. Traditionally, the ADLs are classified as behavioral, structural or mixed level, depending on whether they describe the Instruction Set or structure of the processor or both.

MIMOLA (Leuper, R. and Marwedel, P.1998) primarily captures the structure of the processor. The MSSQ and RECORD compilers use MIMOLA for retargetability. MIMOLA does not support cycle-accurate simulator and does not fully cover the memory hierarchy. The RECORD (Leuper, R. and Marwedel, P.1997) compiler does not include standard optimization techniques and also its scope is limited to DSP architectures. MSSQ's (Leupers, R. et al 1994) architecture scope is limited to micro programmable controllers and it lacks explicit description of processor pipelines which may result in poor code quality. MIMOLA descriptions are generally very low-level and laborious to write.

nML and ISDL are examples of behavioral ADLs. nML is the basis of retargetability offered by the CHESS/CHECKERS environment .In nML (Mishra, P. and Dutt, N., 2005), the processor's instruction set is described as an attributed grammar with the derivations reflecting the set of legal instructions. However, nML does not support multi-cycle or multi-word instructions. ISDL (Hadjiyiannis, G. et al 1997) does not support parallelism, which makes code generation for complex architecture like DSPs and VLIW machines difficult.

FLEXWARE, MDes, LISATek and EXPRESSION have a mixed-level structural/behavioral representation (Halambi, A., et al 2001). FLEXWARE is not able to handle the resource conflict between parallel/pipelined instructions and also explicit specification of memory subsystem is not possible. MDes is used in the Trimaran framework .Trimaran provides parameterized ILP architecture called HPL-PD. It has fixed instruction set, but numerous parameters that can be adapted (e.g. registers,FUs,latencies,etc.). There is a limit, however, to the extent the machine can be modified, since the machine must remain in the HPL-PD space. MDes permits description of traditional memory hierarchy (register files, caches, etc) and it cannot handle on-chip DRAM, frame buffers, partitioned memory address spaces,etc.

LISATek (Hohenauer, M. et al 2004) processor employs Language for Instruction Set Architecture (LISA) that describes the behavior, the structure and the I/O interface. LISATek offers high flexibility at the expense of additional manual compiler description effort. Retargetability in EXPRESS is achieved by EXPRESSION ADL which is capable of describing behavior and structure of the processor. It integrates the structural and instruction-set description and provides support for novel memory subsystems. SPAM uses OLIVE as its machine description language. OLIVE only supports description of Instruction Set Architecture (ISA). It does not capture the pipeline timing and irregular data paths. The comparison of the above ADLs has been given in (Jain, M.K. and Ramnani, V., 2007). It is observed in the above ADLs that there is a trade off between retargetability effort and design space explored . We have chosen EXPRESSION as the base of our research because it has the mixed-level approach, the retargetability efforts are less as it has a GUI and it covers a considerable design space.

III. EXPRESSION

The EXPRESSION ADL (Grun, P. et al 1998, Halambi, A. et al 1999) follows a mixed-level approach (behavioral and structural) to facilitate automatic software toolkit generation, validation, HDL generation, and design space exploration for a wide range of programmable embedded systems. The ADL captures the structure, behavior, and mapping (between structure and behavior) of the architecture. The ADL captures all the architectural components and their connectivity as a net list. It considers four types of components: units (e.g., ALUs), storages (e.g., register files), ports, and connections (e.g., buses). It captures two types edges in the net list: pipeline edges and data transfer edges. The pipeline edges specify instruction transfer between units via pipeline latches, whereas data transfer edges specify data transfer between components, typically between units and storages or between two storages. It has the ability to capture novel memory subsystem.

The behavior is organized into operation groups, with each group containing a set of operations having some common characteristics. Each operation is then described in terms of its opcode, operands, and behavior. The mapping functions map components in the structure to operations in the behavior. It defines, for each functional unit, the set of operations supported by that unit (and vice versa).

The EXPRESSION tool-kit comes with a GUI (V-SAT) (Khare, A. et al 1999) front-end to schematically enter the architecture connectivity and instruction set description. The GUI back end converts the schematic description and instruction set description into EXPRESSION ADL format.

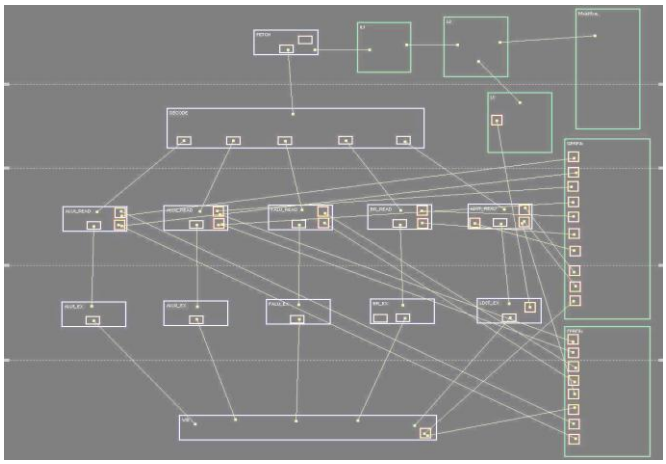

Figure 1 : Base Architecture

A snap shot of the base architecture acesMIPS (Biswas, P. et al 2003) is shown in Figure 1. It has five pipeline stages:

Fetch, Decode, Operand Read, Execute and Writeback. The Operand Read and Execute stages have five parallel pipeline paths: ALU1, ALU2, Floating-Point, Branch, and Load Store. It has two register files: integer and float. It has two level of cache hierarchy with common L2 for both data and instruction. It also uses SRAM as a scratch-pad memory.

IV. EXPERIMENTATION WITH EXPRESSION FRAMEWORK

We present in this section, some of the exploration directions, which are deemed to be important by a system designer. The benchmarks that were used for testing the EXPRESSION framework comprise the following:

a. Livermore Loops. (Benchmarks/LLs)
b. Multimedia kernels. (Benchmarks/MMs)

We present here the result of running the Livermore Loops and Multimedia kernels on the acesMIPS architecture and performing the architecture explorations discussed in the next sub sections.

A. Adding a Parallel Pipeline path

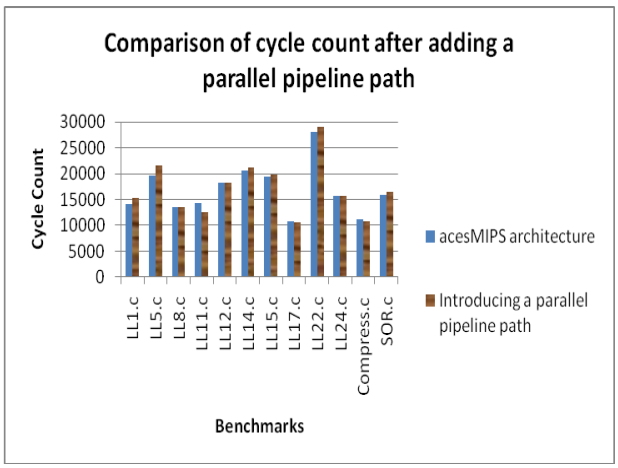

Figure 2: Change in performance after adding a parallel pipeline path

Our first experiment demonstrates the impact of adding a parallel pipeline path comprising of the ALU3_READ and ALU3_EXECUTE units.

```
(OpReadUnit ALU3_READ
(CAPACITY 1)
(INSTR_IN 1)
(INSTR_OUT 1)
(TIMING (all 1))
(OPCODES ALU_Unit_ops)
(LATCHES (OUT Alu3ReadExLatch))
(LATCHES (IN DecAlu3ReadLatch))
(PORTS Alu3ReadPort1 Alu3ReadPort2)
 )
(OperationLatch Alu3ReadExLatch
 )
(UnitPort Alu3ReadPort1("_READ_")
(ARGUMENT _SOURCE_1_)
(CAPACITY 1)  )
(UnitPort Alu3ReadPort2("_READ_")
(ARGUMENT _SOURCE_2_)
(CAPACITY 1)  )
(ExecuteUnit ALU3_EX
(CAPACITY 1)
(INSTR_IN 1)
(INSTR_OUT 1)
(TIMING (all 1))
(OPCODES ALU_Unit_ops)
(ARGUMENT _UNIT_)  (LATCHES (OUT
Alu3ExWbLatch))
(LATCHES (IN Alu3ReadExLatch))
 )
(OperationLatch Alu3ExWbLatch
  )
)
```

It can be observed in Figure 2 that the cycle count in some of the benchmarks increases after a parallel path is introduced .Adding a parallel path should increase the performance of the benchmarks i.e. the cycle count should decrease or should remain the same. On the contrary the cycle count has increased which is due to poor scheduling in EXPRESSION.

B. Introducing a Pipeline Stage

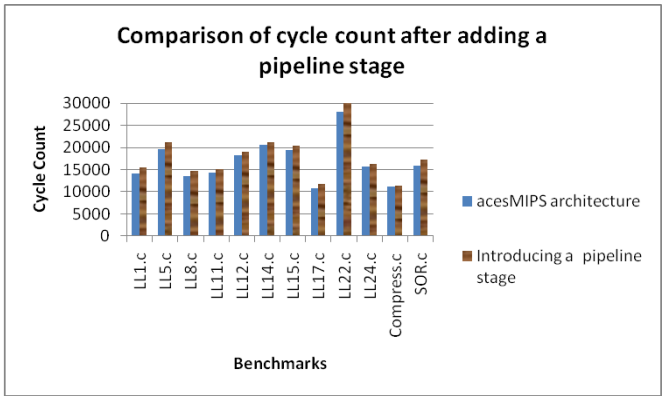

Figure 3: Change in Performance after Introducing a new Pipeline Stage

In Figure 3 we have studied the impact of adding a new pipeline stage (Tomiyama, H. et al 1999) consisting of "mult" operation.

```
(SimpleStageUnit Alu2_S2
(CAPACITY 1)
(INSTR_IN 1)
(INSTR_OUT 1)
(TIMING (mult 1))
(OPCODES MultGroup)
(ARGUMENT _UNIT_)  (LATCHES (OUT
Alu2_S2WbLatch))
(LATCHES (IN Alu2ExWbLatch))
 )
(OperationLatch Alu2_S2WbLatch
 )
)
```

The benchmarks were analyzed for parallelization using affine partitioning (Lim, A.2001). It is found that the benchmarks which can be parallelized, have not benefited from the new pipelined stage as is expected.

C. Adding a MAC Operation

The MAC (multiply and accumulate) operation is a combination of three simple generic machine operations – IMUL, MFLO and IADD, described in the EXPRESSION generic machine model in (Biswas, P. 2003).

```
(
( GENERIC
(
(IMUL DST[1] = REG(1) SRC[1] = REG(2) SRC[2] =
REG(3))
(MFLO DST[1] = REG(4) SRC[1] = REG(2))
(IADD DST[1] = REG(5) SRC[1] = REG(6) SRC[2] =
REG(4))
)
)
( TARGET
(
(mac DST[1] = REG(5) SRC[1] = REG(2) SRC[2] =
REG(3) SRC[3] = REG(4))
)
)
)
```

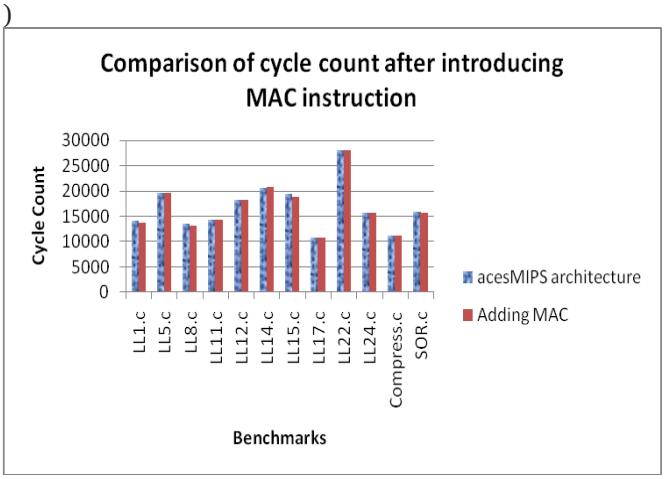

Figure 4: Change in Performance after adding a MAC operation

The rule for mac above should be specified before the rules of mult and addu.

In Figure 4, we analyze the performance of the benchmarks by introducing the MAC operation. Benchmarks which have several addition and multiplication operations benefit from this instruction which combines the two operations into one instruction, while most of the benchmarks remain unaffected. If we analyze the

benchmarks closely, it is found that the performance of the MAC operation is arbitrary. Some benchmarks that have scope of use MAC operation do not show any improvement in performance and vice versa.

V. INSTRUCTION SET EXPLORATION

The instruction set has been compared with that of ARM7TDMI(ARM Limited ,2004) . The Instruction Set description can be addressed under the following heads:

a. Setting VAR_GROUPS
b. Setting OP_GROUPS
c. Setting OPERAND_MAPPING
d. Setting TREE_MAPPING
e. Setting Instruction Description

The GUI permits adding/deleting instructions in the instruction set(OP_GROUPS).It is possible to introduce complex instructions, which are a combination of other instructions ,in the OP_GROUP. In (Jain, M.K. and Ramnani, V., 2009) it has been shown that on introducing the MAC instruction (multiply and accumulate) operation, a combination of three simple generic machine operations – IMUL, MFLO and IADD, the performance of the MAC operation is arbitrary. Benchmarks which have several addition and multiplication operations benefit from this instruction which combines the two operations into one instruction, while most of the benchmarks remain unaffected. If we analyze the benchmarks closely, it is found that some benchmarks that have scope of use MAC operation do not show any improvement in performance and vice versa.

EXPRESSION supports integer, float and double data types .The target registers are classified into var_groups or register classes based on their data types and mappings with the var_groups in generic register files. The data types have been further categorized into var_groups depending on the register classes. 33 VAR_GROUPS have been specified in EXPRESSION .The utility of these var_groups is questionable .It is observed that 'int_any' , 'int_cc' , 'int_hilo' , 'int_normal' refer to the same register class GPRFile[1-28]. Also, out of the 33 VAR_GROUPS only a few of them have been used to describe the source and destination operands .The var_groups that have been utilized are:

Integer - int_any, int_hilo, int_all, int_immediate, int_mem
Double - double1_normal, double_all
Float - float_normal, float_all

It can be said that the var_groups have been defined according to their usage, otherwise the var_groups are basically the data types i.e. integer, float and double. Existence of the VAR_GROUPS, in turn ,has generated the need for OPERAND_MAPPING .

EXPRESSION does not provide flags like Carry, Zero, Overflow, Negative etc. It has a single flag register 'CC' which used to store the output for all types of logical tests. This prevents conditional execution of the instructions .Due to the absence of these flags , the framework has separate instructions for all the logical tests like in ALU_OP there are 'sgt' , 'sge' , ' slt' ,'sle' , 'seq' and 'sne'. There are similar instructions in FALU_OP for float and double data.
The framework cannot handle function / procedure, since no program status register for storing the program status on function/procedure call has been defined. Interrupt and exception handling have also not been addressed in the framework.

A lot of heterogeneity is observed in the in the instruction set. For instance 'and' and 'andi' are separate instructions for register and immediate operands. Similarly, there are 'or' and 'ori' and 'xor' and 'xori' . But, 'mult', 'div', 'addu' and 'subu' are single instructions for register and immediate 2nd operand. There is no instruction for the logical test "not equal to "in FALU_OP for float and double operands, but instructions for all other tests have been defined:

For Double: c_le_d , c_lt_d , c_ge_d , c_gt_d , c_eq_d
For Float: c_le_s , c_lt_s , c_ge_s , c_gt_s , c_eq_s

The instruction for the same has been defined in ALU_OP.

The instruction set does not contain any instruction for bit-wise manipulation of data. Bit-wise operations are one's complement, bitwise AND, masking , manipulation of half-word , etc.

VI. IMPLEMENTING ARM7TDMI-S WITH EXPRESSION TOOL-KIT

ARM7TDMI-S(Technical Publications,1998, ARM Limited ,2005) is a high-performance 32-bit RISC Microcontroller with Thumb extensions .It supports 512KB on-chip Flash ROM with In-System Programming (ISP) and In-Application Programming (IAP), 32KB RAM and Vectored Interrupt Controller,

In order to simulate ARM7TDMI-S, the following modifications were made in the basic acesMIPS architecture:-

a. ALU2 and FALU and their data paths were removed.
b. The op-group for FALU is removed.
c. The register file FPRfile is removed and then the data types in VAR-GROUPS linked to FPRfile are removed. Consequently, the instruction description also needs to be changed.
d. Change the size of GPRfile to 16.
e. Change the size of Main Memory to 32KB , delete the IL1 and L2 memory modules.

The Livermore Loops (Benchmarks/LLs) available with the EXPRESSION tool-kit were first simulated on ARM7TDMI-S (Chipset LPC 2148) (Koninklijke Philips Electronics ,2005)and then on the EXPRESSION tool-kit with the above modifications made on acesMIPS architecture. A difference in the cycle count is observed as has been shown in Chart 1.

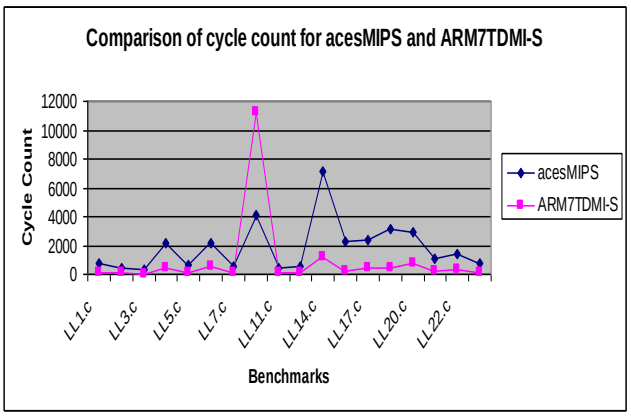

Chart 1: Cycle count for benchmarks implemented on acesMIPS and ARM7TDMI-S

VII. CONCLUSIONS

The EXPRESSION tool-kit is capable of automatic compiler / simulator generation. The GUI in the framework facilitates exploring various processor architectures. However, some of the modifications do not give expected result. The scope of this tool-kit is analyzed by varying various parameters through the GUI and executing the benchmarks. The results of variations are shown in Figure 5. It is observed that introduction of a parallel path increases the cycle count of some benchmarks. Similar decrease in performance is observed on adding a new pipeline stage and introducing MAC instruction. The EXPRESSION tool-kit needs improvement on the instruction scheduling part, specially the parallelization and pipelining of the data path.

In this paper, we have simulated ARM7TDMI-S on the EXPRESSION tool-kit to see whether they give the same cycle count on the benchmarks. It is observed that the results not comparable. In future research, validation of these results i.e. cycle count of these benchmarks will be carried out using other simulators.

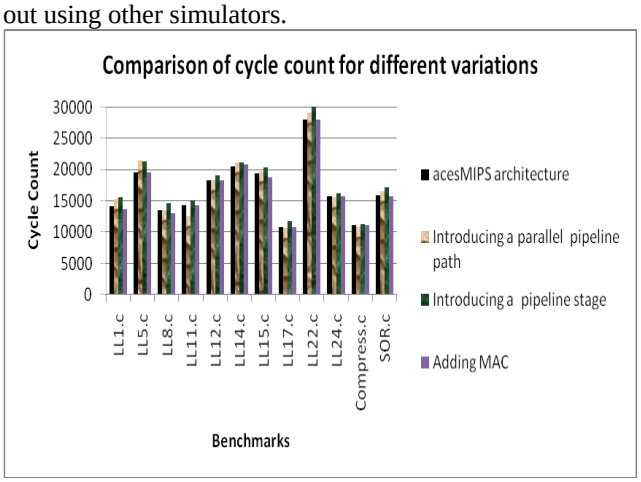

Figure 5 : Cycle count variations after different variations in acesMIPS architectute

The design space needs to be enhanced, so that it can generate code for various VLIW processors. The review made in this paper will help develop a framework which requires less retargetability efforts and capable to cater to a wide range of processors.

VIII. REFERENCES

[1] Jain , M.K., Kumar, A., Balakrishnan ,M., and Gangwar, A.(2005) Customizing Embedded Processors for Specific Applications , In proceedings of Recent Trends in Practice and Theory of Information Technology, Proc. of NRB Seminar, Cochin, pp. 261-284

[2] Jain, M.K. and Ramnani, V.,(2007) Challenges in Retargetable Compiler Technology in ASIP Design, Indian Engineering Congress

[3] Mishra , P. and Dutt, N.,(2005) Architecture Description Languages for Embedded systems, In IEE Proceedings on Computer and Digital. Techniques, Vol. 152, No. 3

[4] Grun, P., Halambi, A., Khare, A., Ganesh , V. And Dutt, N.(1998) EXPRESSION : An ADL for System Level Design Exploration .Dept. of Information and Computer Science. University of California , Irvine. Technical Report #98-29

[5] Halambi, A., Grun, P.,Ganesh, V. , Khare, A. , Dutt , N. and Nicolau , A. , (1999) EXPRESSION: A Language for Architecture Exploration through Compiler/Simulator Retargetability. In Proc. DATE

[6] Khare, A., Savoiu, N., Halambi, A., Grun, P., Dutt, N. and Nicolau, A. (1999) V-SAT: A visual specification and analysis tool for system-on-chip exploration. In Proc. EUROMICRO.

[7] Biswas, P., Pasricha, S., Mishra, P., Shrivastava, A., Dutt, N. and Nicolau, A. (2003) EXPRESSION User Manual, Version 1.0

[8] Tomiyama, H., Halambi, A., Grun, P., Dutt, N. and Nicolau, A. ,(1999) Modeling and Verification of Processor Pipelines in SOC Design Exploration , In Proc. of 4th International High Level Design Validation and Test Workshop (HLDVT'99), pp. 10--16

[9] Lim, A.(2001) Improving Parallelism and Data Locality with affine Partitioning ,Ph.D. Thesis, Standford University

[10] Biswas, P., Pasricha, S., Mishra, P., Shrivastava, A., Dutt, N. and Nicolau, A.(2003) EXPRESSION User Manual, Version 1.0,

[11] Jain, M.K. and Ramnani, V., (2009) Reviewing the EXPRESSION Framework ,Multidisciplinary Conference on emerging Issues and Global Scenario , Udaipur , ISSN 0975-3613

[12] Technical Publications (1998) ARM7TDMI Microprocessor Core ,LSI Logic Corporation , Document DB14-000058-02, First Edition

[13] ARM Limited (2004) ARM Technical Reference Manual, ARM Limited , ARM DDI 0210C

[14] ARM Limited (2005) ARM Architecture Reference Manual, ARM Limited , ARM DDI 0100I

[15] Koninklijke Philips Electronics (2005)LPC214X User Manual, Philips Semiconductors ,Volume 1

[16] Leupers, R. and Marwedel , P.(1998) Retargetable code generation based on structural processor descriptions . Design Automation for embedded Systems, 3(1) : 1-36

[17] Leupers, R. , Schenk , W. and Marwedel , P.(1994) Micro code Generation for Flexible Parallel Target Architectures , PACT

[18] Leupers, R. and Marwedel, P.(1997)Retargetable Compilers for Embedded DSPs , In: Proceedings 7th European Multimedia , Microprocessor Systems and Electronic Commerce Conference(EMMSEC) , Florence/Italy.

[19] Hadjiyiannis, G. , Hanono , S. and Devadas , S. (1997) ISDL: An Instruction Set Language for Retargetability , In: Proceeding of Design Automation Conference Anaheim ,CA

[20] Halambi, A., Shrivastava, A., Dutt, N. and Nicolau, A. (2001) A Customizable Compiler Framework for Embedded Systems, In: SCOPES

[21] Hohenauer , M. , Scharwaechter , H., Karuri , K. , Wahlen , O. , Kogel , T. , Leupers,R., Ascheid , G. , Meyr , H. And Braun, G. (2004) Compiler-in-loop Architecture Exploration for Efficient application specific Embedded Processor Design, In: Design & Electronik , Munich , Germany.