A Quantitative Study and Estimation Models for Extensible Instructions in Embedded Processors

Newton Cheung†, Sri Parameswaran†§, Jörg Henkel‡
†School of Computer Science and Engineering, University of New South Wales, Australia
§National Information and Communications Technology Australia (NICTA), Australia
‡Department of Computer Science, University of Karlsruhe, Germany
{ncheung, sridevan}@cse.unsw.edu.au, henkel@informatik.uni-karlsruhe.de *

ABSTRACT

Designing extensible instructions is a computationally complex task, due to the large design space each instruction is exposed to. One method of speeding up the design cycle is to characterize instructions and estimate their peculiarities during a design exploration. In this paper, we study and derive three estimation models for extensible instructions: area overhead, latency, and power consumption under a wide range of customization parameters. System decomposition and regression analysis are used as the underlying methods to characterize and analyze extensible instructions. We verify our estimation models using automatically and manually generated extensible instructions, plus extensible instructions used in large real-world applications. The mean absolute error of our estimation models are as small as: 3.4% (6.7% max.) for area overhead, 5.9% (9.4% max.) for latency, and 4.2% (7.2% max.) for power consumption, compared to estimation through the time consuming synthesis and simulation steps using commercial tools. Our estimation models achieve an average speedup of three orders of magnitude over the commercial tools and thus enable us to conduct a fast and extensive design space exploration that would otherwise not be possible. The estimation models are integrated into our extensible processor tool suite.

1. INTRODUCTION

The demand for increasing product functionality combined with tight design constraints, and at the same time decreasing product life cycles, provides a constant push towards application specific designs, and increasing software content in embedded System-On-Chips (SoCs). Therefore, customizability and programmability are two important necessities in embedded SoCs' design paradigms, and remain one of the main reasons for the preferred use of application specific instruction-set processors over application specific integrated circuits and off-the-shelf processors.

Extensible processors represent the state-of-the-art in application specific instruction-set processors. They typically consist of a base processor core containing a base instruction set, plus the capability to extend this instruction set through new extensible instructions. Extensible instructions are customized to replace computationally intensive code segments (groups of primitive instructions) in the application, satisfying performance and design constraints.

Extensible instructions can be customized in numerous ways: by selecting and parameterizing components like arithmetic operators etc. Designers can judiciously select from the available components and parameterize them for specific functionality. Parallelism techniques can be deployed to achieve a further speedup. There are three known techniques: a) very long instruction words (VLIW); b) vectorization; and c) hard-wired operation. Each of them with varying tradeoffs in performance, power etc. [1, 2, 3]. Plus, these techniques can be used in conjunction with one another. Designers can also schedule the extensible instruction to run in multi-cycles. Thus the design space of extensible instructions is almost unfeasibly complex.

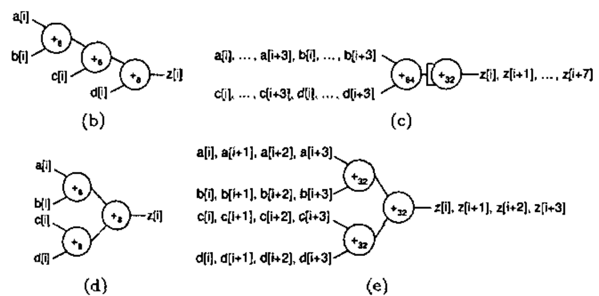


Figure 1: Example: four varieties to design an instruction which replaces a code segment

Fig. 1 shows four instructions (sequences) that can be designed in order to replace a single code segment in the original software-based application. A code segment with four vectors a, b, c and d, are summed up to generate a vector, z, in a loop with a loop iteration count of 1000. If an extensible instruction is to replace the code segment, then a summation in series, or a summation in parallel, using 8-bit adders can be defined. These are shown in Fig. 1b and 1d respectively. Designers can also group four sets of 8-bit data together and perform a 32-bit summation in parallel, which is shown in Fig. 1e. Note that, this implementation loops only 250 times. In Fig. 1c, an instruction using a 64-bit adder and a 32-bit adder can also be implemented, requiring just 125 loop iterations. On top of that, each of these designs, while functionally equivalent, will have differing characteristic in power, performance etc. To verify the area overhead, latency, and power consumption of each instruction is a time consuming task and is not a tractable method for exploring the instruction design space. *For a good design it is crucial to explore as many of the design points as possible. This requires fast and accurate estimation techniques.*

In this paper, we present estimation models of extensible instructions for area overhead, latency and power consumption

*National ICT Australia is funded by the Australian Government's Department of Communications, Information Technology and the Arts and the Australian Research Council through Backing Australia's Ability and the ICT Centre of Excellence program

0-7803-8702-3/04/$20.00 ©2004 IEEE.

using *system decomposition* [4] and *regression analysis* [5]. As we will see, we achieve high accuracy, which enables us to control our previously presented techniques for semi-automatic instruction selection for extensible processors.

The rest of this paper is structured as follows. The next section describes the related work and emphasizes the contribution of our work in that context. Section 3 gives the background and theory. The derivation of the estimation models are described in Section 4 whereas Section 5 discusses the experiments and results. Finally, Section 6 concludes the paper.

2. RELATED WORK AND CONTRIBUTION

An overview of extensible processor platforms, its benefits and challenges are described in [6, 7]. In addition, commercial and research tool suites have shown that performance and power benefits can be orders of magnitude more efficient compared to general purpose processors when deployed in the same embedded systems [8, 9, 10, 11, 12]. Methodologies for designing application specific instruction-set processors are presented in [13, 14, 15].

Recent research in extensible processors has mainly been revolving around automatic instruction generation/selection. In [16, 17], the authors describe methods to generate custom instructions from operation patterns. The work in [15] proposes an instruction encoding scheme for complex instructions under an area constraint. Methods to identify custom instructions from applications are described in [18]. In [19], the authors search for regular templates in a dataflow graph and generate complex instructions by grouping the primitive ones. An automatic system to generate extensible instructions is described in [20].

Estimation methods evaluate different aspects of extensible processors such as area overhead, latency, power consumption, and the performance gains. In [21], a method to estimate speed, area, and power consumption of software intellectual property at an architectural level is presented. The work in [22] describes an energy estimation for VLIW processors based on instruction clustering. In [23], the authors describe a hybrid methodology for estimating energy consumption of extensible processors. However, the proposed energy model does not include the schedule of operations or instructions. A technique to rapidly estimate the performance of the application in a newly designed extensible processor is described in [24]. In addition, the work in [25] proposes a rapid estimation scheme for area and power utilizing parameterized components in high level synthesis. While several methods do exist in the high level synthesis literature for estimation of speed, area and power, they often relate to estimation methods for a fully customized circuit, whereas an extensible instruction is a partially customized circuit and is surrounded by built-in control logic of the extensible processor, storage areas and busses connecting the instruction to the storage area.

The novel contributions of our work are:

1. to derive accurate and fast estimation models (area overhead, latency, and power consumption) of extensible instructions;
2. to simplify the process of modeling extensible instructions by using system decomposition and regression analysis;
3. to take into account both parallelism techniques and schedules alternatives for instruction models.

The estimation models have been successfully embedded into our extensible processor tool suite and provide fast feedback in the instruction generation/selection phase.

3. BACKGROUND AND THEORY

As a platform we use the Xtensa processor from Tensilica Inc. [12]. It consists of a RISC base core with approx. 80 base instructions, plus the capacity to define specific functionality through extensible instructions, using Tensilica Instruction Extension (TIE), which coexist with the base instructions. An extensible instruction (in Xtensa) decomposes mainly into five parts: the decoder, which uses data from the instruction decoding stage and assigns internal signals for the execution stage; hardware to clock gate the instruction, such that the instruction can be turned on and off as needed; control logic (also known as top-logic), to schedule the operations within the instruction; customized registers, to store any additional variables (there are two types of the customized register: register file and instruction register); and combinational operations, such as arithmetic and logic operations.

We use methods described in *system decomposition theory* to decompose the embedded processor hardware into independent subsystems which can be analyzed separately [4]. System decomposition theory originated from the ontological model of an information system decomposition. The following basic definitions and theorems are obtained, with a detailed description given in [4].

Basic definitions of the system decomposition theory:

- A system, σ, comprises a set of parameters.
- A parameter, c, is a discrete variable from an ordered and finite set.
- A parameter space, S, is a multidimensional discrete space, where each dimension corresponds to a parameter and each point corresponds to an extensible instruction.
- A function, f, over a parameter space, S, is a function that corresponds to a model of the extensible instruction .
- A system $\acute{\sigma}$ is a subsystem of σ, and σ is a supersystem of $\acute{\sigma}$ if and only if the composition of $\acute{\sigma}$ is a subset of the composition of σ.
- A decomposition of a system σ is a set of subsystems $D(\sigma) = \{\sigma_i\}_{i\in I}$, such that each parameter in the system is included in at least one of the subsystems.

Theorem 3.1: Let $D(\sigma) = \{\sigma_i\}_{i\in I}$ be a decomposition σ. The parameter space of the decomposition is:

$$S(D) \equiv S(D(\sigma)) = \otimes_{i\in I} S(\sigma_i) \tag{1}$$

A parameter space of S(D) will be called a parameter space of the decomposition.

Let c_i be a parameter in an ordered and finite set S_i. C is an n tuple of parameters, $C = \{c_1, c_2, ..., c_n\}$. Let C_1 be an x tuple of parameters $\{c_1, c_2, ..., c_x\}$ and C_2 be an $n - x$ tuple of parameters $\{c_{x+1}, c_{x+2}, ..., c_n\}$ such that n tuple C can be expressed as $\{C_1, C_2\}$. In addition, a function, $f(C_1, C_2)$, is independent of C_2 if $f(C_1, C_2) = f(C_1)$ which can be completely represented by parameters in C_1.

There are three requirements to ensure a valid system decomposition: i) the system must have a well-defined structure; ii) the system must only be represented by a known set of the parameters; iii) a change in a parameter, that belongs to a subsystem, must result in a change on the function of the subsystem. The reasons that system decomposition theory is applicable to model extensible instructions are as follows: i) extensible instructions are well structured into five architectural parts (as described above); and, ii) we represent extensible instructions using a set of customization parameters. These parameters represent a wide range of instruction customizations, which model differing components, dissimilar parallelism techniques, and diverse scheduling.

Regression analysis is an analysis method that expresses a model as a function of parameters. In extensible instructions, each of the area overhead, latency, and power consumption is a model that is expressed as a function of customization parameters. For example, a model of a system, $M(\sigma)$, is expressed as a linear function of c_1, c_2, ..., c_n, where each c_i is a parameter of the system and can be represented as follows:

$$M(\sigma) = m_0 + m_1c_1 + m_2c_2 + ... + m_nc_n \tag{2}$$

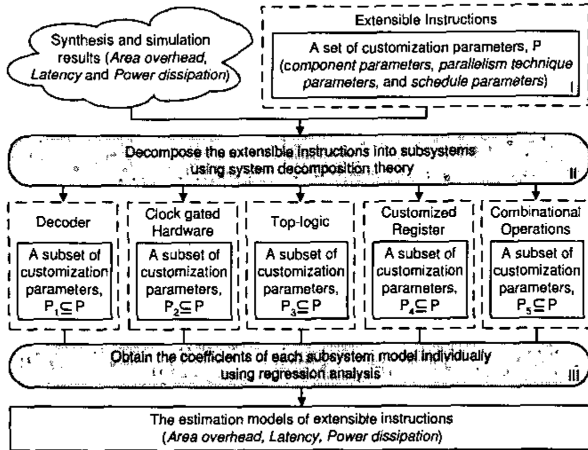


Figure 2: An overview for characterizing and estimating the models of the extensible instructions

where m_0, m_1, ..., m_n are coefficients of the parameters. The function can take other forms of expression, such as quadratic or polynomial etc. The coefficients of the parameters, and the relationship of the model (i.e. linear, quadratic, polynomial etc) can be determined (if such a relationship exists) by commercial tools, when a sample dataset and parameters are given.

4. EXTENSIBLE INSTRUCTIONS MODEL

In this section, we first present an overview of the derivation methodology. We then describe how extensible instructions are represented by customization parameters, and how the extensible instructions are divided into subsystems with a subset of customization parameters using *system decomposition.* Finally, we explain how the estimation models are derived using regression analysis in terms of customization parameters.

4.1 Overview

An overview of the method to derive estimation models is shown in Fig. 2. Extensible instructions with a set of customization parameters and synthesis & simulation results (including the results for each subsystem, such as the decoder, top logic, etc) are inputs. The outputs are the estimation models of the extensible instructions. An extensible instruction represented by a large set of customization parameters is complex and therefore hard to analyze. Hence, we apply the system decomposition theory to decompose an instruction into its independent structural subsystems: decoder, clock gating hardware, top-logic, customized register, and combinational operations. Each such subsystem is represented by a subset of customization parameters. A customization parameter belongs to a subsystem if and only if a change in the customization parameter would affect synthesis and simulation results of the subsystem. In addition, one and the same customization parameter can be contained in multiple subsystems. We then use regression analysis in each subsystem to determine: i) the relationship between synthesis & simulation results and the subset of the customization parameters; and ii) the coefficients of the customization parameters in the estimation models. In addition, the decomposition of subsystems is refined until the subsystem's estimation model is satisfactory. The estimation models for the subsystems are then combined to model extensible instructions for the purpose of estimating its characteristics. This procedure is applied separately for area overhead, latency, and power consumption.

4.2 Customization Parameters

Customization parameters are properties of the instruction that designers can customize when designing extensible instructions. They can be divided into three categories: a) component parameters, b) parallelism technique parameters, and c) schedule parameters.

Component parameters characterize primitive operators of an instructions. They can be classified based on their structural similarity, such as: i) adder and subtractor $(+/-)$; ii) multiplier $(*)$; iii) conditional operators and multiplexers $(<, >, ?:)$; iv) bitwise and reduction logic $(\&, |, ...)$; v) shifter $(<<, >>)$; vi) built-in adder and subtractor from the library (*LIB_add*) (we used these custom built components to show the versatility); vii) built-in multiplier (*LIB_mul*); viii) built-in selector (*LIB_csa*); ix) built-in *mac* (*LIB_mac*); x) register file; and xi) instruction register. The bitwidths of all primitive operators can be altered, too.

Parallelism parameters characterize various levels of parallelism during instruction execution. As for parallelism techniques, there are three: *VLIW*, allows a single instruction to execute multiple independent operators in parallel; *vectorization*, increases throughput by operating multiple data elements at a time; and *hard-wired operation*, takes a set of single instructions with constants and composes them into one new-custom complex instruction. The parallelism technique parameters include: i) the width of the instruction using different parallelism techniques, which models the additional hardware and wider busses for paralleling the instructions; ii) the connectivity of the components (register file, instruction register, operations etc), which represents the components that commonly shared; iii) the number of operations in series; iv) the number of operations in parallel; and the number of operations in the instruction.

Schedule parameters represent the scheduling for instruction execution such as multi-cycling etc.. The schedule parameters are i) the number of clock cycles it requires to execute an instruction; ii) the max. number of instructions that may reside in the processor; and iii) the max. number of registers that may be used by an instruction. Table 1 shows the notations and the descriptions for customization parameters[1]. Hence, notations from table 1 are used to refer to customization parameters.

4.3 Characterization for various Constraints

Area Overhead Characterization

Unless the subsystems share common hardware, the area overhead of extensible instructions can be defined as the summation of the individual subsystem's area overhead.

The *decoder*, the *clock gating hardware*, and the *top-logic* are built-ins and are actually shared amongst extensible instructions which has to be taken in cosideration. The customization parameters for these subsystems are: i) Con_{oper}; ii) Con_{regf}; iii) Con_{ireg}; iv) Num_{inst}; and v) Num_{minst}.

Also, a customized register can be shared amongst the extensible instructions in the processor. The area overhead of the *customized register* is based on the size and the width of the registers. Therefore, the customization parameters are: i) Num_{regf}; and ii) Num_{ireg}.

The *combinational operations'* area overhead is not shared with other instructions and is dependent only upon the operations within the instruction. The customization parameters for combinational operations are: i) $Num_{add/sub}$; ii) Num_{mul}; iii) Num_{cond}; iv) Num_{logic}; v) Num_{shift}; vi) Num_{LIB_add}; vii) Num_{LIB_mul}; viii) Num_{LIB_csa}; and ix) Num_{LIB_mac}.

[1]To handle scalability and to limit the design space, we only consider the following bitwidths: 8/16/32/64/128 for the operators with suffix i; and 32/64/128 for register files with suffix j in the table

	Customization parameters	Descriptions
Components	Num_{add/sub_i}	Number of i-bit addition/subtraction operators
	Num_{mul_i}	Number of i-bit multiplication operators
	Num_{cond}	Number of condition operator and multiplexors
	Num_{logic}	Number of bitwise and reduction logics
	Num_{shift_i}	Number of i-bit shifters
	$Num_{LIB_add_i}$	Number of i-bit built-in adders
	$Num_{LIB_csa_i}$	Number of i-bit built-in selectors
	$Num_{LIB_mul_i}$	Number of i-bit built-in multipliers
	$Num_{LIB_mac_i}$	Number of i-bit built-in macs
	Num_{regf_j}	Number of j-bit width register files
	Num_{ireg}	Number of instruction registers
Parallelism tech.	Wid_{vliw}	Width of the VLIW instructions
	Wid_{vector}	Width of the vectorization instructions
	Wid_{hwired}	Width of the hard-wired instructions
	Con_{regf_j}	Connectivity of j-bit register files
	Con_{ireg}	Connectivity of instruction registers
	Con_{oper_i}	Connectivity of operations
	Num_{oper_i}	Number of i-bit operations in total
	Num_{ser}	Number of operations in serial
	Num_{para}	Number of operations in parallel
Sche.	Num_{mcyc}	Number of cycles scheduled
	Num_{minst}	Number of instructions included
	Use_{reg_j}	Usage of the j-bit register files

Table 1: Customization parameters of extensible instructions

Latency Characterization

The latency of extensible instructions can be defined as the maximum delay of each subsystem in the critical path when that specific extensible instruction is executed. Major part of the critical path is contributed to by combinational operations. Other subsystems either have very little effect on the latency, or do not lie on the critical path.

The customization parameters for latency of the *decoder*, *clock gated*, *top-logic*, and *customized register* are similar to the area overhead characterization. The reason is that these subsystems mainly revolve around the connectivity between each other (i.e. fan-ins / fan-outs) while the internal latency is relatively constant within these subsystems.

In the *combinational operations*, the latency depends not only on structural components, but also on parallelism technique parameters and schedule parameters. Component parameters represent latency of independent operators. Parallelism technique parameters describe latency of internal connectivity, the number of stages in the instruction, and the level of parallelism; and schedule parameters represent the multi-cycles instructions.

Power Consumption Characterization

The characterization of power consumption is similar to the constraints described above.

The customization parameters of *decoder* and *top-logic* relate to the connectivity between the subsystems, and therefore are dependent upon: i) Con_{oper}; ii) Con_{regf}; iii) Con_{oper}; iv) Num_{minst}; and v) Num_{mcyc}.

For *clock gating hardware*, the customization parameters include the connectivity and complexity of operations and the scheduling: i) Num_{oper}; ii) Num_{regf}; iii) Num_{inst}; iv) Num_{mcyc}; v) Num_{ser}; and vi) Num_{para}. The last two parameters specify the power consumption of the clock tree in the extensible instruction.

For *customized register*, the power consumption refers to the number of customized registers that the instruction used. The customization parameters are i) Num_{regf}; ii) Num_{ireg}; iii) Use_{reg}.

For *computational operations*, the power consumption characterization is further categorized into number of stages in the instruction, and the level of parallelism in the stage. The reason for capturing power consumption when operations execute in parallel and when multi-cycle instructions are present, is that stalling increases energy dissipation significantly.

4.4 Estimating Design Constraints of Extensible Instructions

Area Overhead Estimation

As discussed previously, the area of extensible instructions, $A(inst)$, can be defined by using system decomposition (eqn. 1):

$$A(inst) = \otimes_{i \in \{dec,clk,top,reg,opea\}} A(i) \tag{3}$$

or as:

$$A(inst) = \sum_{i \in \{dec,clk,top,reg,opea\}} A(i) \tag{4}$$

where $A(i)$ is the area overhead estimation of all affected subsystems. Applying regression analysis on each subsystem and its customization parameter subset, the area overhead estimation of subsystems is derived and are described as follows:

The decoder has five customization parameters (according to Table 1) Using regression analysis, the relationship of the estimation model is seen to be linear and the area overhead estimation, $A(dec)$, is hence defined as:

$$\begin{aligned} A(dec) = \Sigma_{i \in 32,64,128} A_{regf_i} Con_{regf_i} + A_{ireg} Con_{ireg} + \\ \Sigma_{i \in 8,16,32,64,128} A_{oper_i} Con_{oper_i} + A_{mcyc} Num_{mcyc} + \\ A_{minst} Num_{minst} \end{aligned} \tag{5}$$

where A_{regf_i}, A_{ireg}, A_{oper_i}, A_{mcyc}, and A_{minst} are the respective coefficients.

For clock gating hardware, the area overhead estimation $A(clk)$ can be defined as:

$$\begin{aligned} A(clk) = \Sigma_{i \in 32,64,128} A_{regf_i} Con_{regf_i} + A_{ireg} Con_{ireg} + \\ A_{minst} Num_{minst} \end{aligned} \tag{6}$$

$A(top)$ is the area overhead estimation of a top-logic and is defined as:

$$\begin{aligned} A(top) = \Sigma_{i \in 8,16,32,64,128} A_{oper_i} Con_{oper_i} + \\ \Sigma_{i \in 32,64,128} A_{regf_i} Con_{regf_i} + A_{ireg} Con_{ireg} + \\ A_{mcyc} Num_{mcyc} + A_{minst} Num_{minst} \end{aligned} \tag{7}$$

The area overhead estimation of customized register, $A(reg)$, is defined as:

$$A(reg) = \Sigma_{i \in 32,64,128} A_{regf_i} Num_{regf_i} + A_{ireg} Num_{ireg} \tag{8}$$

$A(opea)$ is the area overhead estimation of combinational operations and is defined as:

$$\begin{aligned} A(opea) = \Sigma_{i \in 8,16,32,64,128} \{ A_{add/sub_i} Num_{add/sub_i} + \\ A_{LIB_mul_i} Num_{LIB_mul_i} + A_{LIB_mac_i} Num_{LIB_mac_i} + \\ A_{LIB_add_i} Num_{LIB_add_i} + A_{LIB_csa_i} Num_{LIB_csa_i} + \\ A_{mul_i} Num_{mul_i} + A_{shift_i} Num_{shift_i} \} + \\ A_{cond} Num_{cond} + A_{logic} Num_{logic} \end{aligned} \tag{9}$$

Latency Estimation

As described in Section 4.3, the latency of extensible instructions is the maximum delay of each subsystem in the critical path of the extensible instruction. Therefore, the latency estimation, $T(inst)$, is defined as:

$$T(inst) = max_{i \in \{dec,clk,top,reg,opea\}} T(i) \tag{10}$$

where $T(dec)$ is the latency estimation of the decoder which is defined as follow:

$$\begin{aligned} T(dec) = \Sigma_{i \in 32,64,128} T_{regf_i} Con_{regf_i} + T_{ireg} Con_{ireg} + \\ \Sigma_{i \in 8,16,32,64,128} T_{oper_i} Con_{oper_i} + T_{mcyc} Num_{mcyc} + \\ T_{minst} Num_{minst} \end{aligned} \tag{11}$$

	Coefficients	Descriptions	Area (gates)	Latency (ps)	Power (μW)
Decoder	$A_{regf}, T_{regf}, P_{regf}$	register file connection	10	28	18
	$A_{ireg}, T_{ireg}, P_{ireg}$	instruction register connection	12	30	22
	$A_{oper}, T_{oper}, P_{oper}$	operator connection	45	451	91
	$A_{mcyc}, T_{mcyc}, P_{mcyc}$	multiple clock cycle connection	15	47	24
	$A_{minst}, T_{minst}, P_{minst}$	multiple instructions connection	20	60	26
Clock	$A_{regf}, T_{regf}, P_{regf}$	register file connection	12	51	64
	$A_{ireg}, T_{ireg}, P_{ireg}$	instruction register connection	15	79	67
	$A_{mcyc}, T_{mcyc}, P_{mcyc}$	multiple clock cycle connection	52	341	431
	$A_{minst}, T_{minst}, P_{minst}$	multiple instructions connection	49	292	181
Toplogic	$A_{regf}, T_{regf}, P_{regf}$	register file connection	492	435	211
	$A_{ireg}, T_{ireg}, P_{ireg}$	instruction register connection	2047	542	481
	$A_{oper}, T_{oper}, P_{oper}$	operator connection	512	970	786
	$A_{mcyc}, T_{mcyc}, P_{mcyc}$	multiple clock cycle connection	489	285	162
	$A_{minst}, T_{minst}, P_{minst}$	multiple instructions connection	3593	312	252
Reg	$A_{regf_i}, T_{regf_i}, P_{regf_i}$	i-bit width register file	16018	5482	612
	$A_{ireg}, T_{ireg}, P_{ireg}$	instruction register	5735	6885	893
	$A_{minst}, T_{minst}, P_{minst}$	number of instructions uses register	539	454	341
Combinational Operations	$A_{add/sub}, T_{add/sub}, P_{add/sub}$	addition/subtraction	13591	4125	617
	$A_{mult}, T_{mult}, P_{mult}$	multiplier	48095	7480	23401
	$A_{logic}, T_{logic}, P_{logic}$	logic operator	226	2745	12
	$A_{shift}, T_{shift}, P_{shift}$	shifter	22301	2918	3417
	$A_{cond}, T_{cond}, P_{cond}$	condition operator	283	1282	85
	$A_{LIB_add}, T_{LIB_add}, P_{LIB_add}$	build-in adder	12878	3581	589
	$A_{LIB_csa}, T_{LIB_csa}, P_{LIB_csa}$	build-in selector	28138	2472	1250
	$A_{LIB_mac}, T_{LIB_mac}, P_{LIB_mac}$	build-in mac	55110	6485	24921
	$A_{LIB_mul}, T_{LIB_mul}, P_{LIB_mul}$	build-in multipler	48344	6172	22911
	$A_{reg}, T_{reg}, P_{reg}$	register connection	387	870	481
	$A_{ireg}, T_{ireg}, P_{ireg}$	instruction register connection	456	490	721
	$A_{mcyc}, T_{mcyc}, P_{mcyc}$	multiple clock cycle connection	102	1810	651
	$A_{minst}, T_{minst}, P_{minst}$	multiple instructions	1148	490	891
	$A_{VLIW}, T_{VLIW}, P_{VLIW}$	VLIW technique	–	1910	1767
	$A_{vector}, T_{vector}, P_{vector}$	vectorization technique	–	810	2313
	$A_{hwired}, T_{hwired}, P_{hwired}$	hard-wired operation technique	–	1125	429
	$A_{ser}, T_{ser}, P_{ser}$	number of operation in serial	–	648	872
	$A_{para}, T_{para}, P_{para}$	number of operation in parallel	–	238	657

Table 2: The coefficients of the extensible instructions for the purpose of calibrating through regression

the latency estimation of the clock gated, $T(clk)$, is shown:

$$T(clk) = \Sigma_{i\in 32,64,128} T_{regf_i} Con_{regf_i} + T_{ireg} Con_{ireg} + T_{mcyc} Num_{mcyc} + T_{minst} Num_{minst} \quad (12)$$

$T(top)$ is the latency estimation of the top logic and is shown:

$$T(top) = \Sigma_{i\in 32,64,128} T_{regf_i} Con_{regf_i} + T_{ireg} Con_{ireg} + \Sigma_{i\in 8,16,32,64,128} T_{oper_i} Num_{oper_i} + T_{mcyc} Num_{mcyc} + T_{minst} Num_{minst} \quad (13)$$

$T(reg)$ is the latency estimation of the customized register and is shown:

$$T(reg) = \Sigma_{i\in 32,64,128} \{ \frac{1}{Num_{regf} + Num_{ireg}} \times T_{regf_i} Num_{regf_i} + T_{ireg} Num_{ireg} \} \quad (14)$$

$T(opea)$ is the latency estimation of the combinational operations and is shown:

$$T(opea) = \frac{Num_{ser}}{Num_{mcyc} \times Num_{oper}} \times \Sigma_{i\in 8,16,32,64,128} \{ T_{add/sub_i} Num_{add/sub_i} + T_{LIB_add_i} Num_{LIB_add_i} + T_{LIB_csa_i} Num_{LIB_csa_i} + T_{LIB_mul_i} Num_{LIB_mul_i} + T_{LIB_mac_i} Num_{LIB_mac_i} + T_{mul_i} Num_{mul_i} + T_{shift_i} Num_{shift_i} \} + T_{cond} Num_{cond} + T_{logic} Num_{logic} + T_{VLIW} Wid_{VLIW} + T_{vector} Wid_{vector} + T_{hwired} Wid_{hwired} + T_{ser} Num_{ser} + T_{para} Num_{para} + T_{mcyc} Num_{mcyc} + T_{minst} Num_{minst} \quad (15)$$

Power Consumption Estimation

Similar to, previous argumentations, the power consumption can be modeled as:

$$P(inst) = \sum_{i\in\{dec,clk,top,reg,opea\}} P(i) \quad (16)$$

where

$$P(dec) = \Sigma_{i\in 32,64,128} P_{regf_i} Con_{regf_i} + P_{ireg} Con_{ireg} + \Sigma_{i\in 8,16,32,64,128} P_{oper_i} Con_{oper_i} + P_{mcyc} Num_{mcyc} + P_{minst} Num_{minst} \quad (17)$$

$$P(clk) = \Sigma_{i\in 32,64,128} P_{regf_i} Con_{regf_i} + P_{ireg} Con_{ireg} + P_{mcyc} Num_{mcyc} + P_{minst} Num_{minst} \quad (18)$$

$$P(top) = \Sigma_{i\in 32,64,128} P_{regf_i} Con_{regf_i} + P_{ireg} Con_{ireg} + \Sigma_{i\in 8,16,32,64,128} P_{oper_i} Con_{oper_i} + P_{mcyc} Num_{mcyc} + P_{minst} Num_{minst} \quad (19)$$

$$P(reg) = \Sigma_{i\in 32,64,128} \{ \frac{Use_{reg_i}}{Num_{regf_i} + Num_{ireg}} \times (P_{regf_i} Num_{regf_i} + P_{ireg} Num_{ireg} + P_{minst} Num_{minst}) \} \quad (20)$$

$$P(opea) = \Sigma_{i\in 8,16,32,64,128} \{ P_{add/sub_i} Num_{add/sub_i} + P_{LIB_mul_i} Num_{LIB_mul_i} + P_{LIB_mac_i} Num_{LIB_mac_i} + P_{LIB_add_i} Num_{LIB_add_i} + P_{LIB_csa_i} Num_{LIB_csa_i} + P_{mul_i} Num_{mul_i} + P_{shift_i} Num_{shift_i} \} + P_{cond} Num_{cond} + P_{logic} Num_{logic} + P_{VLIW} Wid_{VLIW} + P_{vector} Wid_{vector} + P_{hwired} Wid_{hwired} + P_{ser} Num_{ser} + P_{para} Num_{para} + P_{mcyc} Num_{mcyc} + P_{minst} Num_{minst} \quad (21)$$

5. EXPERIMENTAL RESULTS

The purpose of the experiments is to evaluate the constraint estimation models against "measured" values i.e. compared to the case where the instructions have actually been synthesized and power has been estimated at gate-level.

For evaluation purposes, we used the T1040.0 version of the Xtensa processor (0.18μ technology) from Tensilica Inc. [12],

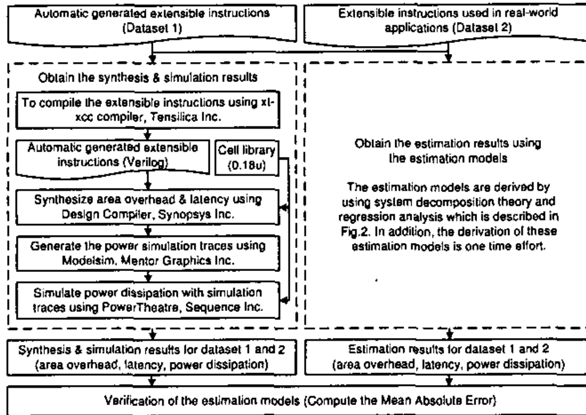


Figure 3: Experimental methodology

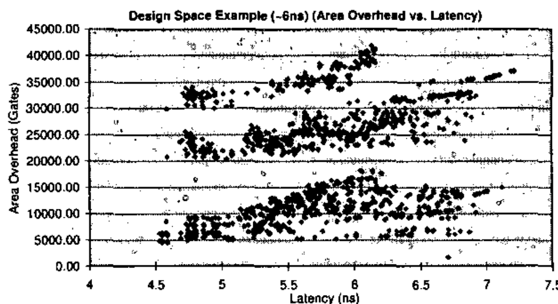


Figure 4: Design space example (around 6ns)

with a clock speed of 166.7MHz. All experiments were conducted on a Sun UltraSPARC III running at 900MHz (dual) with 4Gb of RAM. Although these models are based on the Xtensa processor and 0.18μ technology[2], the underlying method for deriving models is general and can be applied to any extensible processor platform of similar capability (ability to design custom instructions).

5.1 Experiments Conducted

We conducted two series of experiments: i) determining the coefficients and deriving the estimation models (this is a one-time effort); ii) verifying the estimation models. Fig. 3 shows the verification methodology. In the first experiment, an amount of about 5,000 extensible instructions[3] (TIE) were automatically generated (dataset 1) with a wide range of customization parameters. Fig. 4 shows the possible extensible instructions' design points around the latency of 6ns. These instructions are first compiled and Verilog implementations are generated. We then synthesized instructions to obtain area overhead and latency using *Design Compiler* from Synopsys, Inc. [26] with cell library (0.18μ). The power consumption figures are obtained using *PowerTheatre* from Sequence Design, Inc. [27] with simulation data (testbenches) generated using *Modelsim* from Mentor Graphics, Inc. [28]. After that, we used dataset 1 to determine the coefficients of the estimation models using *S-Plus* from Insightful, Inc. [5]. Note that, *determining the coefficients of estimation models is a one time effort.* Table 2 shows all obtained coefficients of the

[2]We conducted preliminary experiments on 0.13μ technology, and results show that the methodology is valid. Due to page limit, those results are not shown here

[3]The number 5,000 results from an analysis of the design space. We made sure that all areas of the design space were reasonably well covered.

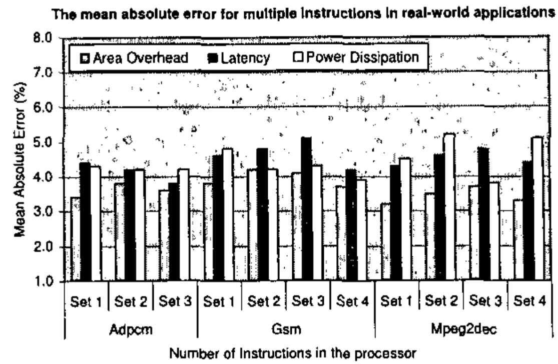


Figure 5: The accuracy of the estimation models for multiple instructions (Sets of instructions: set 1 contains single instruction; set 2 contains group of two instructions, etc)

extensible instructions[4]. We computed the mean absolute error between the estimation models and the synthesis results in all automatically generated instructions[5]. In addition, we further discuss the error rates for: VLIW instructions; vectorization instructions; hard-wired operation instructions; multi-cycle instructions; and when multiple instructions are part of the processor.

The second set of experiment used 11 extensible instructions from three real-world applications: adpcm, gsm, and mpeg2 [29]. We first obtained synthesis and simulation results using commercial tools (as shown in Fig. 3). We then computed the estimation results using our estimation models. These 11 instructions were grouped into sets of instructions to evaluate the estimation models, for the case when multiple instructions are present/selected in the processor. The accuracy of the estimation models for individual extensible instructions and multiple instructions are then verified.

5.2 Evaluation Results

In our first experiment, we examine the accuracy of the estimation models under differing customizations: VLIW; vectorization; hard-wired operation; multi-cycle; and sequences (multiple) of extensible instructions. Table 3 shows the mean absolute error for the area overhead, latency, and power consumption of the estimation models in these categories. The mean absolute error (area overhead, latency, and power consumption) in hard-wired operation is lower than those instructions using VLIW and vectorization techniques. The reason is that the estimation for VLIW and vectorization techniques instructions depends on a larger number of customization parameters, and hence, higher error rates are observed. In terms of schedules, the mean absolute error is relatively close to the average mean error. The mean absolute error of the estimation models for all automatically generated instructions is only 2.5% for area overhead, 4.7% for latency, and 3.1% for power consumption.

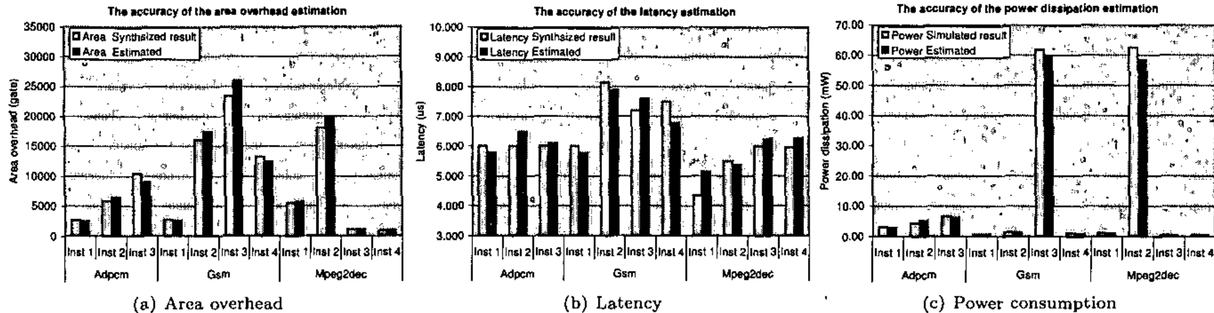
Fig. 5 shows the mean absolute error of the estimation models for sequences (multiple) of instructions in the real-world applications. The mean absolute error for previously unseen multiple instructions range between 3-6% for the three estima-

[4]This table shows only the 32-bit coefficients due to the page limit

[5]The reasoning for automatically generating instructions is as follows: if we would only generate those (few) extensible instructions that are actually useful for a certain application, then we would most likely not cover all cases to evaluate our estimation techniques. Even though these instruction are automatically generated, they are indeed useful for the estimation evaluation even though they might not speedup the application at all; we, however, have also generated and evaluated extensible instructions for real-world applications such that those applications profit from the extensible instructions

Instructions Categories	Area Overhead Error			Latency Error			Power consumption Error		
	Mean Abs.	Maximum	Minimum	Mean Abs.	Maximum	Minimum	Mean Abs.	Maximum	Minimum
VLIW	3.5	6.7	0.3	4.2	6.7	1.4	3.8	6.9	0.6
Vectorization	2.8	7.5	1.4	3.5	7.0	0.3	2.5	6.8	1.7
Hard-wired	2.6	4.4	0.2	2.5	5.4	0.3	2.0	5.1	0.8
Multi-cycle	4.2	7.6	0.0	4.1	8.4	0.2	4.5	7.9	0.3
Multiple Inst	3.2	6.7	0.4	2.5	6.8	1.7	2.8	6.2	0.8
Overall	2.5	6.2	1.0	4.7	9.1	1.1	3.1	6.1	0.7

Table 3: The mean absolute error of the estimation models in different types of extensible instructions



(a) Area overhead (b) Latency (c) Power consumption

Figure 6: The accuracy of the estimation models in real-world applications

tion models. In addition, Fig. 6 summarizes the accuracy of estimation models for extensible instructions with unseen individual real-world application's extensible instructions (dataset 2). The maximum estimation error is 6.7%, 9.4%, and 7.2% while the mean absolute error is only 3.4%, 5.9%, and 4.2% for area overhead, latency and power consumption respectively. The estimation errors are all far below the estimation errors of the commercial estimation tools (typically around 20% absolute error at gate-level) we verified our models against. As such, we can conclude that our models are accurate enough for the purpose of high-level design space exploration for extensible instructions.

Our estimation models are by far faster than a complete synthesis and simulation process with commercial tools. The average time taken by *Design Compiler* and *PowerTheatre* to determine the customization of an extensible instruction can be up to several hours, while our estimation models only require a few seconds in the longest case. This is another prerequisite for extensive design space exploration.

6. CONCLUSIONS

We have presented fast and accurate techniques for estimating area overhead, latency and power consumption of extensible instructions. As opposed to similar work of others we also include models for instruction parallelism, and multicycling, as we found that this is crucial for accurate and reliable estimation. Using our techniques that have been calibrated through regression models and compared against commercial synthesis/estimation tools, we are able to explore the large design space of extensible processors. The estimation techniques have been integrated into our extensible processor tool suite. The results are as follows: the mean absolute error for a set of instructions used in real-world applications is 3.4% (6.7% max.) for area overhead, 5.9% (9.4% max.) for latency, and 4.2% (7.2% max.) for power consumption. The estimation models execute in a few seconds for an instruction, while synthesis and subsequent estimation would take hours. We currently extend our estimation techniques to estimate more complex extensible instructions as they will be available soon in commercial extensible processor tool suites.

7. REFERENCES

[1] S. Aditya, B. R. Rau, and V. Kathail, "Automatic architectural synthesis of vliw and epic processors," in *ISSS*, 1999.

[2] A. Peymandoust, L. Pozzi, P. Ienne, and G. Micheli, "Automatic instruction-set extension and utilization for embedded processors," in *ASAP*, 2003.

[3] H. Zima and B. Chapman, "Supercompilers for parallel and vector computers," in *Addison-Wesley (ACM)*, 1990.

[4] Y. Wand and R. Weber, "An ontological model of an information system," in *IEEE Tran. of Software Engineering*, 1990.

[5] "Splus." Insightful, Inc. (http://www.insightful.com).

[6] J. Henkel, "Closing the soc design gap," in *IEEE Computer Magazine, vol.36, Iss.9, pp119-121.*, 2003.

[7] K. Keutzer, S. Malik, and A. R. Newton, "From asic to asip: The next design discontinuity," in *ICCD*, 2002.

[8] "Arctangent processor." ARC, Inc. (http://www.arc.com).

[9] "Asip-meister." (http://www.eda-meister.org/asip-meister/).

[10] "Jazz dsp." Improv Systems, Inc. (http://www.improvsys.com).

[11] "Lisatek." CoWare, Inc. (http://www.coware.com).

[12] "Xtensa processor." Tensilica, Inc. (http://www.tensilica.com).

[13] H. Choi, J.-S. Kim, C. W. Yoon, *et al.*, "Synthesis of application specific instructions for embedded dsp software," in *IEEE Trans. Computers*, 1999.

[14] V. Kathail, S. Aditya, R. Schreiber, B. R. Rau, D. C. Cron-quist, and M. Sivaraman, "Pico: Automatically designing custom computers," in *Computer*, 2002.

[15] J. Lee, K. Choi, and N. Dutt, "Efficient instruction encoding for automatic instruction set design of configurable asips," in *ICCAD*, 2002.

[16] K. Atasu, L. Pozzi, and P. Lenne, "Automatic application-specific instruction-set extensions under microarchitectural constraints," in *DAC*, 2003.

[17] F. Sun, A. Raghunathan, S. Ravi, and N. K. Jha, "A scalable application specific processor synthesis methodology," in *ICCAD*, 2003.

[18] N. Clark, W. Tang, and S. Mahlke, "Automatically generating custom instruction set extensions," in *WASP*, 2002.

[19] P. Brisk, A. Kaplan, R. Kastner, and M. Sarrafzadeh, "Instruction generation and regularity extraction for reconfigurable processors," in *CASES*, 2002.

[20] D. Goodwin and D. Petkov, "Automatically generating custom instruction set extensions," in *CASES*, 2003.

[21] J. Sanghavi and A. Wang, "Estimation of speed, area, and power of parameterizable soft ip," in *DAC*, 2001.

[22] A. Bona, M. Sami, D. Sciuto, C. Silvano, V. Zaccaria, and R. Zafalon, "Energy estimation and optimization of embedded vliw processors based on instruction clustering," in *DAC*, 2002.

[23] Y. Fei, S. Ravi, A. Raghunathan, and N. Jha, "Energy estimation for extensible processors," in *DATE*, 2003.

[24] N. Cheung, J. Henkel, and S. Parameswaran, "Rapid configuration & instruction selection for an asip: A case study," in *DATE*, 2003.

[25] P. Jha and N. Dutt, "Rapid estimation for parameterized components in high-level synthesis," in *IEEE Tran. on VLSI*, 1993.

[26] "Design compiler." Synopsys, Inc. (http://www.synopsys.com).

[27] "Powertheater." Sequence, Inc. (http://www.sequencedesign.com).

[28] "Modelsim." Model, Inc. (http://www.model.com).

[29] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems," in *Int. Symp. on Microarchitecture*, 1997.