Edith Cowan University Research Online

Theses: Doctorates and Masters

Theses

1-1-2005

3D-SoftChip: A novel 3D vertically integrated adaptive computing system

Chul Kim Edith Cowan University

Follow this and additional works at: https://ro.ecu.edu.au/theses

Part of the Engineering Commons

Recommended Citation

Kim, C. (2005). *3D-SoftChip: A novel 3D vertically integrated adaptive computing system*. https://ro.ecu.edu.au/theses/656

This Thesis is posted at Research Online. https://ro.ecu.edu.au/theses/656

Edith Cowan University

Copyright Warning

You may print or download ONE copy of this document for the purpose of your own research or study.

The University does not authorize you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site.

You are reminded of the following:

- Copyright owners are entitled to take legal action against persons who infringe their copyright.
- A reproduction of material that is protected by copyright may be a copyright infringement. Where the reproduction of such material is done without attribution of authorship, with false attribution of authorship or the authorship is treated in a derogatory manner, this may be a breach of the author's moral rights contained in Part IX of the Copyright Act 1968 (Cth).
- Courts have the power to impose a wide range of civil and criminal sanctions for infringement of copyright, infringement of moral rights and other offences under the Copyright Act 1968 (Cth).
 Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

3D-SoftChip:

A Novel 3D Vertically Integrated Adaptive Computing System

A Dissertation

Presented to the School of Engineering and Mathematics

Edith Cowan University

Western Australia

In partial fulfillment of the requirements for the degree of Master of Engineering Science

by Cbul KIM

Supervisor: *Dr. Alexander Rassau* Submission Date: *June 2005*



Dedication

To my fiancé Sang-Mi Hyun,

my father Nam-Gil Kim,

my mother Sung-Sun Park,

my brother-in-law Sun-Shin Lee,

and my sisters Hee-Joung, Su-Joung, Youn-Joung Kim.

© 2005 Chul KIM All Rights Reserved

-1-

Table of Contents

LICE OF THERE	
USE OF THESIS	
DECLARATION	
ACKNOWLEDGMENTS	
ABSTRACT	의 사람이는 것 같이 가장을 알았는 것을 가지 않는다
PUBLICATIONS	na ng manana na manana na kanana na kana Manana kanana na kana Manana kanana na kanana na kanana na kanana na kanana kanana na kanana na kanana na kanana na kanana na kanana n
LIST OF FIGURES	- 11 - 11 - 11 - 11 - 11 - 11 - 11 - 1
LIST OF TABLES	
·	
1. INTRODUCTION	
1 1 2D MEDDICALLY VINTECT AND STOTEMS OVERVIEW	n an an Anna a Anna an Anna an
1.1 5D YERIICALLI INTEGRATED 5151 ENS OVERVIEW.	10 10
1.2 ADAPTIVE COMPUTING STSTEMS OVERVIEW	
1.2.1 Adaptive Computing Systems	10
1.2.1.1 The Iveea for Adaptive Computing Systems	10
1.2.1.2 The Concept of Adaptive Comparing Systems	19
1.2.2 Classification of Adaptive Computing Systems	
1.2.2.1 Previous works	
1.2.2.2 MorphoSys Vs 3D-SoftChip	
1.5 MOTIVATION OF THESIS	la se en la susta a la sere en la susta a la sere en la
1.4 SCOPE OF THESIS	
1.4.1 Scope of Each Chapters	
1.5 CONCLUSIONS	
2. SYSTEM ARCHITECTURE OF 3D-SOFTCHIP	
A 1 Community of the second and the second sec	n en en en star her star fan de skriet fan de skriet fan en en star fan en en en skriet fan en en en en en en 🦡
2.1 CORE TECHNOLOGY FOR 3D-SOFTCHIP	21
2.2 OVERALL ARCHITECTURE OF 3D-SOFTCHIP	
2.3 FEATURES OF 3D-SOFTCHIP	
2.4 SYSTEM COMPONENTS	
2.4.1 Configurable Array Processor (CAP) Chip	
2.4.1.1 Helerogeneous Types of PEs	
2.4.2 Intelligent Configurable Switch (ICS) Culp	
2.4.2.1 Switch Block	
2.4.2.2 ICS_RISC	
2.4.2.5 Data Frame Buffer	n de la companya de l El companya de la comp
2.4.2.4 Program Memory	
2.4.2.5 Data Memory	
2.4.2.5 DMA Controller	
2.4.2.7 3D Interconnection Technology	
2.5 DESIGN GUIDELINES	
2.6 DESIGN METHODOLOGY	36
2.6.1 Suggested HW/SW Co-design and Verification	Methodology
2.7 CONCLUSIONS	37
3. ARCHITECTURE OF CAP CHIP	30
3.1 OVERALL ARCHITECTURE OF CAP CHIP	an a
3.2 TWO TYPES OF PROCESSING ELEMENT (PE)S	

2 A 1 Gameland BE (C BE)	
3.2.1 Standard-PE (S-PE) 3.2.2 Processing Accelerator-PE (PA-PE)	
3.3 PE FUNCTIONS	*****
3.3.1 Standard-PE Functions	a an
3.3.2 Processing Accelerator-PE Functions	
3.3.3 PE Instruction Formats and Operation M	lodes
3.4 EMBEDDED LOCAL SRAM	
3.5 CONFIGURABLE NATURE OF ARITIMETIC PRIM	ITTIVES
3.5.1 Scalable Parallel Multiplier Cell	
3.6 QUAD-PE	
3.7 UNITCAP CHIP AKCHITECTURE	
J.O CUNCLUSIONS	
4. ARCHITECTURE OF ICS CHIP	***************
4.1 SWITCH BLOCK	
4.2 ICS RISC	n na sense se s
4.2.1 Features of ICS_RISC	n en
4.2.2 System Components of ICS_RISC	
4.2.3 Types of Instruction Set	
4.2.4 ICS_RISC Instruction Set Architecture-	Version1.0
4.3 HIGH BANDWIDTH DATA INTERFACE UNIT	a da anti-arresta da anti-arresta da anti-arresta da anti-arresta da anti-arresta da anti-arresta da anti-arres A anti-arresta da anti-arresta da anti-arresta da anti-arresta da anti-arresta da anti-arresta da anti-arresta d
4.4 CONCLUSIONS	en and state of the second state of the
5. ARCHITECTURE OF UNITCHIP	
5.1 UNITCHIP ARCHITECTURE	TTD
5.2 PIPELINED OPERATION MECHANISM OF UNITE 5.3 Apra Ferimations and Constraints	ann an
5.5 AREA ESTIMATIONS AND CONSTRAINTOURING	a service and a service of the servi
6. INTERCONNECTION NETWORK	
6.1 HIERARCHICAL INTERCONNECTION ARCHITEC	TURE
6.1.1 PE and Switch Block Array Interconnect	lion
6.1.1.1 Programmable Nature of PE Array Interc	connection
6.1.2 Indium Bump Interconnection	
6.2 CONCLUSIONS	
7. HIGH-LEVEL MODELING OF 3D-SOFTCHIP	USING SYSTEMC
7.1 SYSTEMC OVERVIEW	an a
7.1.1 CAD Environment for System Comments of a Sugara Length Monte the optimized and the second statements of the second statement of the second state	
7.2 DISTEM-LEVEL MODELING OF DD-BOFTCHIP. 7.2 I Stondard-DF	
7.2.1 Standard-1 Processing Accelerator-PE	an an Agus an ann an Angalach an Airteachailte an A
7.2.3 ICS RISC	
7.2.4 UnitChip	an a
7.3 CONCLUSIONS	and a start of the s
A LODI TO MICH MADDING FOR IN COPPOSIT	De la construcción de la const
8. APPLICATION MAPPING FOR 3D-SOFTCHIL	
8. APPLICATION MAPPING FOR 3D-SOFTCHIN 8.1 Full Search Block Matching Algorithm	P 1 (FBMA)
8. APPLICATION MAPPING FOR 3D-SOFTCHI 8.1 Full Search Block Matching Algorithm 8.2 FBMA MAPPING METHOD FOR 3D-SOFTCHIP	P 1 (FBMA)
8. APPLICATION MAPPING FOR 3D-SOFTCHI 8.1 Full Search Block Matching Algorithm 8.2 FBMA MAPPING METHOD FOR 3D-SOFTCHIP 8.3 PERFORMANCE ANALYSIS	P 1 (FBMA)
8. APPLICATION MAPPING FOR 3D-SOFTCHI 8.1 Full Search Block Matching Algorithm 8.2 FBMA MAPPING METHOD FOR 3D-SOFTCHIP 8.3 PERFORMANCE ANALYSIS 8.4 CONCLUSIONS	P 1 (FBMA)
8. APPLICATION MAPPING FOR 3D-SOFTCHI 8.1 Full Search Block Matching Algorithm 8.2 FBMA MAPPING METHOD FOR 3D-SOFTCHIP 8.3 PERFORMANCE ANALYSIS 8.4 CONCLUSIONS	P ((FBMA)
 APPLICATION MAPPING FOR 3D-SOFTCHIN 8.1 FULL SEARCH BLOCK MATCHING ALGORITHM 8.2 FBMA MAPPING METHOD FOR 3D-SOFTCHIP 8.3 PERFORMANCE ANALYSIS 8.4 CONCLUSIONS 9. CONCLUSIONS 	P 1 (FBMA)
 APPLICATION MAPPING FOR 3D-SOFTCHIN 8.1 FULL SEARCH BLOCK MATCHING ALGORITHM 8.2 FBMA MAPPING METHOD FOR 3D-SOFTCHIP 8.3 PERFORMANCE ANALYSIS 8.4 CONCLUSIONS CONCLUSIONS CONCLUSIONS 	P

-3-

Declaration

I certify that this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any institution of higher education; and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in the text.

Signature

1

Date 12/08/05

-6-

Acknowledgements

I would like to express my gratitude to the following people, who helped me to stand this position.

Prof. Kamran Eshraghian as my principle supervisor who initiated the research program and gave me the opportunity to commence my master course at Edith Cowan University providing financial support and great inspiration towards my research. Unfortunately, he left the university towards the end of my research however he left significant impression of his great leadership that I want to follow.

Prof. Mike Myung-Ok Lee, who inspired me to study overseas and gave me an opportunity and warmth supervision during my course. I have learned strong propulsion and passion through his supervision.

Prof. Byung-Lok Cho, he used to be my supervisor during my undergraduate study. I have started with his great supervision and learned the life and belief as an electronic engineer. I will not forget his guidance that has changed my whole life.

Dr. Alexander Rassau, my principle supervisor, I am really a lucky fellow to meet him as a principle supervisor. Sometimes, he becomes my ear, mouth, hands and legs. I can not forget his infinite interest and supervision capacity for me. I could not finish my course without his great supervision which will never be forgotten and it is very much appreciated. Thanks Dr. Alexander Rassau.

My family is the most precious in my life. They motivated and encouraged me unfailingly so my deepest gratitude goes to my family and I dedicate my dissertation to my family; my father, my mother, my brother-in-law, my sisters and my future new family; my father-in-law, mother-in-law and my new brother-in-law.

Lastly, my fiancé Sang-Mi, she pushing me to study hard, but ironically, she gave me so many interruptions as well. But I even love these interruptions. I will promise that I will be a good husband and I will love you forever.

ABSTRACT

At present, as we enter the nano and giga-scaled integrated-circuit era, there are many system design challenges which must be overcome to resolve problems in current systems. The incredibly increased nonrecurring engineering (NRE) cost, abruptly shortened Timeto-Market (TTA) period and ever widening design productive gaps are good examples illustrating the problems in current systems. To cope with these problems, the concept of an Adaptive Computing System is becoming a critical technology for next generation computing systems. The other big problem is an explosion in the interconnection wire requirements in standard planar technology resulting from the very high data-bandwidth requirements demanded for real-time communications and multimedia signal processing. The concept of 3D-vertical integration of 2D planar chips becomes an attractive solution to combat the ever increasing interconnect wire requirements. As a result, this research proposes the concept of a novel 3D integrated adaptive computing system, which we term 3D-ACSoC. The architecture and advanced system design methodology of the proposed 3D-SoftChip as a forthcoming giga-scaled integrated circuit computing system has been introduced, along with high-level system modeling and functional verification in the early design stage using SystemC.

A major challenge in this research is to explore the proposed 3D-SoftChip platform to investigate the effectiveness of the first novel 3D vertically integrated Adaptive Computing System-on-Chip (ACSoC) as a next generation computing system. The suggested 3D-SoftChip has been modeled at a system level using SystemC and the functional verification of the modeled system has been firmly verified. The hand-crafted assembler code for implementation of the MPEG4 motion estimation algorithm has been applied with more than 3.8 times performance improvement over conventional systems. It can be clearly demonstrated that it is a highly suitable architecture for next generation computing systems. Finally, further work to realize the full implementation of the novel concept of a 3D-ACSoC has been suggested.

Publications

The following is a list of papers published during the course of this research.

International Journals

Chul Kim, Alex Rassau, Stefan Lachowicz, Mike Myung-ok Lee and Kamran Eshraghian (2005) – "3D-SoftChip: A Novel Architecture for Next Generation: Adaptive Computing Systems", to be published in EURASIP Journal on Applied Signal Processing.

Chul Kim, Alex Rassau, Stefan Lachowicz, Mike Myung-ok Lee and Kamran Eshraghian (2005) – "3D-SoftChip: A System-level Verification and Characterization of a Novel 3D Vertically Integrated Adaptive Computing System-on-Chip", Submitted in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

International Conferences

Chul Kim, Mike Myung-ok Lee, Kamran Eshraghian and Byung Lok Cho (2004) – "SoC-B Design and Testing Technique of IS-95C CDMA Transmitter for Measurement of Electronic Field Intensity using FPGA and ASIC", 2rd IEEE International Workshop on Electronic Design, Test and Applications(DELTA2004), Perth, Australia, pp.251-254.

Chul Kim, Mike Myung-ok Lee, Kamran Eshraghian and Byung Lok Cho (2005) – "A Highly Accurate Electric Field Intensity Measurement System for IMT2000 and CDMA Network", Accepted on Advanced Industrial Conference on Telecommunication (AICT2005), Lisbon, Portugal. Chul Kim, Alex Rassau, Mike Myung-ok Lee and Kamran Eshraghian (2005) – "3D-SoftChip: A Novel 3D Vertically Integrated Adaptive Computing System", 13th ACM International Symposium on Field-Programmable Gate Array (FPGA2005), Monterey, California, U.S.A., Poster Section, pp.270

Chul Kim, Alex Rassau, Mike Myung-ok Lee and Kamran Eshraghian (2005) – "3D-SoftChip: A Novel 3D Vertically Integrated Adaptive Computing System", submitted in IFIP International Conference on Very Large Scale Integration (IFIP VLSI-SOC 2005).

Chul Kim, Alex Rassau, Mike Myung-ok Lee and Kamran Eshraghian(2005) – "Highlevel System Modeling and Functional Verification of a 3D-SoftChip Adaptive Computing System using SystemC", submitted in IEEE International Conference on Computer Design (ICCD 2005)

List of Figures

FIGURE 1.1: 3D-SOFTCHIP PHYSICAL ARCHITECTURE	15
FIGURE 1.2: 3D-SOFTCHIP : A NOVEL 3D VERTICALLY INTEGRATED ADAPTIVE COMPUTING SYSTEM-ON-CHIP	16
FIGURE 1.3: COMPUTING SYSTEMS	18
FIGURE 1.4: AN EXAMPLE OF "DO-IT-ALL" DEVICE	20
Figure 2.1: Core Technology for 3D-SoftChip	28
FIGURE 2.2: OVERALL ARCHITECTURE OF 3D-SOFTCHIP	28
FIGURE 2.3: COMPUTATION ALGORITHM : 3 TYPES OF SIMD COMPUTATION MODELS (A) MASSIVELY	
PARALLEL SIMD COMPUTATIONAL MODEL, (B) MULTITHREADED SIMD COMPUTATIONAL MODEL,	
(C) PIPELINED SIMD COMPUTATIONAL MODEL	31
FIGURE 2.4: WORD-LENGTH CONFIGURATION ALGORITHM (A) 8BIT CONFIGURATION, (B) 16BIT	
CONFIGURATION, (C) 32BIT CONFIGURATION	32
FIGURE 2.5: SUGGESTED HW/SW CO-DESIGN AND VERIFICATION METHODOLOGY	38
FIGURE 3.1: TYPES OF PES (A) HOMOGENEOUS TYPE, (B) HETEROGENEOUS TYPE, (C) HETEROGENEOUS TYPE	
WITH DEDICATED FUNCTIONS FOR SPECIAL PURPOSE	40
FIGURE 3.2: TWO TYPES OF PE (A) STANDARD-PE, (B) PROCESSING ACCELERATOR-PE	41
FIGURE 3.3: PE INSTRUCTION FORMATS (A) STANDARD-PE INSTRUCTION FORMAT, (B) PROCESSING-	
Accelerator-PE Instruction Format	43
FIGURE 3.4: PE ARRAY OPERATION MODES (A) HORIZONTAL MODE, (B) VERTICAL MODE, (C) CIRCULAR	
Mode	44
FIGURE 3.5: A GENERIC 1× 1-BIT MULTIPLIER CELL FOR N=1	45
FIGURE 3.5: 8 × 8 MULTIPLIER USING 4-BIT GENERIC CELLS	46
FIGURE 3.7: QUAD-PE	47
FIGURE 3.8: UNITCAP CHIP ARCHITECTURE	48
FIGURE 4.1: ARCHITECTURE OF SWITCH BLOCK : A 6-SIDED SWITCH BLOCK, 7-SIDED SWITCH BLOCK AND 8-	
SIDED SWITCH BLOCK	50
FIGURE 4.2: ARCHITECTURE OF ICS_RISC 32-BIT DEDICATED CONTROL PROCESSOR	51
FIGURE 4.3: A DETAILED ARCHITECTURE OF ICS_RISC	52
FIGURE 4.4: DMA CONTROLLER ARCHITECTURE AND INSTRUCTIONS FOR DMA CONTROLLER	55
FIGURE 5.1: OVERALL ARCHITECTURE OF UNITCHIP	58
FIGURE 6.1: THREE HIERARCHICAL INTERCONNECTION NETWORKS: (A) PE ARRAY INTERCONNECTION	
NETWORK : 2D-MESH INTERCONNECTION FOR LOCAL INTERCONNECTION; (B) SWITCH BLOCK ARRAY	
INTERCONNECTION NETWORK: 2D-MESH INERCONNECTION FOR LONG INTERCONNECTION, (C) INDIUM	
BUMP INTERCONNECTION: SINGLE INDIUM BUMP AFTER REFLOW	63
FIGURE 6.2: QUAD-PE AND PROGRAMMABLE INTERCONNECT ARCHITECTURE	63

FIGURE 6.3: 3D FLIP-CHIP WAFER BONDING TECHNOLOGY USING INDIUM BUMP INTERCONNECTION
ARRAYS
FIGURE 7.1: SYSTEM DESIGN METHODOLOGY: (A) CONVENTIONAL DESIGN METHODOLOGY, (B) SYSTEMC
DESIGN METHODOLOGY
FIGURE 7.2: THE CAD ENVIRONMENT FOR SYSTEMC: VISUAL C++ VERSION 6.0, GTKWAVE WAVEFORM
VIEWER
FIGURE 7.3: HIGH-LEVEL MODELING OF S-PES: (A) S-PE BLOCK DIAGRAM, (B) FILE STRUCTURE OF S-PE
FIGURE 7.4: THE OUTPUT WAVEFORM OF S-PE
FIGURE 7.5: HIGH-LEVEL MODELING OF PA-PES: (A) PA-PE BLOCK DIAGRAM, (B) FILE STRUCTURE OF PA-
PE70
FIGURE 7.6: THE OUTPUT WAVEFORM OF PA-PE
FIGURE 7.7: HIGH-LEVED MODELING OF ICS_RISC: (A) ICS_RISC BLOCK DIAGRAM, (B) FILE STRUCTURE
OF ICS_RISC
FIGURE 7.8: THE PSUEDO CODE FOR ICS_RISC
FIGURE 7.9: THE OUTPUT WAVEFORM OF ICS_RISC73
FIGURE 7.10: THE INSTRUCTION INDEX
FIGURE 7.11: HIGH-LEVEL MODELING OF UNITCHIP: (A) UNITCHIP BLOCK DIAGRAM, (B) FILE STRUCTURE
OF UNITCHIP
FIGURE 7.12: THE OUTPUT WAVEFORM OF UNITCHIP
FIGURE 8.1: BLOCK MATCHING MOTION ESTIMATION
FIGURE 8.2: MAPPING METHOD FOR FULL SEARCH BLOCK MATCHING AND DATA FLOW
FIGURE 8.3: PERFORMANCE COMPARISON FOR MOTION ESTIMATION
FIGURE 9.1: THE PARAMTERIZED MEMORY MODELING EXAMPLE USING SYSTEMC

List of Tables

TABLE 1.1: 3D FABRICATION TECHNOLOGIES	17
TABLE 1.2: RECONFIGURABLE COMPUTING VS ADAPTIVE COMPUTING	20
TABLE 1.3: RECONFIGURABLE AND ADAPTIVE COMPUTING SYSTEMS	23
TABLE 1.4: COMPARISON OF MORPHOSYS WITH 3D-SOFTCHIP	24
TABLE 3.1: CHARACTERISTICS OF EACH PE TYPES	40
TABLE 3.2: CHARACTERISTICS OF THE THE TWO TYPES OF PE	41
TABLE 3.3: STANDARD-PE FUNCTIONS	42
TABLE 3.4: PROCESSING ACCELERATOR-PE FUNCTIONS	43
TABLE 4.1: TYPES OF INSTRUCTION SET	52
TABLE 4.2: INSTRUCTION SET SUMMARY (ICS_RISC ISA VERSION 1.0)	53
TABLE 5.1: PIPELINED UNITCHIP OPERATION MECHANISM	58
TABLE 5.2: AREA ESTIMATION AND CONSTRAINT OF UNITCHIP (TARGET TECHNOLOGY: 0.13UM PROCESS)	60
TABLE 6.1: INTER-PE BUS (IPB) INTERCONNECTION CONNECTIVITY	64

Chapter 1 Introduction

System design is becoming increasingly challenging as the complexity of integrated circuits and the time-to-market pressures relentlessly increase. Adaptive computing is a critical technology to develop for future computing systems in order to resolve most of the problems that system designers are now faced with due in no small part to its potential for wide applicability. Up until now, however, this concept has not been fully realized because of the many constraints such as chip real-estate limitations and the software complexity. Advancements of semiconductor processing technology and software technology, however, adaptive computing has more recently started to receive considerable research attention [2, 3, 7] and this concept is now starting to move and expand into the realm of adaptive computing. Software defined virtual hardware [9] and "Do-it-all" devices [12] are good examples that demonstrate this development direction for computing systems.

Another growing problem in advanced computation systems, particularly for real-time communication or video processing applications, is the data bandwidth necessary to satisfy the processing requirements. A novel 3D integration system such as 3D SoC [24], 3D-SoftChip [14,15] which is able to satisfy the severe demand of more computation throughput by effectively manipulating the functionality of hardware primitives through

vertical integration of two 2D chips is another concept proposed for next generation computing systems. This research explores the proposed 3D-SoftChip platform to investigate the effectiveness of the first novel 3D vertically integrated Adaptive Computing System-on-Chip (3D-ACSoC) as a next generation computing system. This thesis outlines research into the system level design and functional verification of 3D-SoftChip in the initial stage of development of the novel 3D vertically integrated ACSoC.



Figure 1.1: 3D-SoftChip Physical Architecture

Figure 1.1 illustrates the physical architecture of the 3D-SoftChip comprising the vertical integration of two 2D chips. The upper chip is the Intelligent Configurable Switch (ICS). The lower chip is the Configurable Array Processor (CAP). Interconnection between the two 2D chips is achieved via Indium bump interconnections. As the starting point for our 3D mapping, the 2-D plane architecture of the 3D-SoftChip is also illustrated in Figure 1.2 in order to demonstrate the principle.



Figure 1.2: 3D-SoftChip: A novel 3D vertically integrated adaptive computing system-on-chip

1.1 3D Vertically Integrated Systems Overview

During the past few years, there has been significant research demand for 3D vertically integrated systems due to the ever growing wiring requirements, which are fast becoming the major bottleneck for future gigascale integrated systems [23,24]. In Very Deep Submicron silicon geometry, standard planar technology has many drawbacks such as performance, reliability etc. caused by limitations in the wiring. Moreover, the data bandwidth requirements for the next generation computing systems are becoming ever larger. To overcome these problems, the concept of 3D-SoC, 3D-SoftChip has been developed, which exploits the vertical integration of two or more 2D planar chips to effectively manipulate computation throughput. Previous work has shown that the 3D integration of systems can significantly reduce interconnection requirements [25]. As described by Joyner, et al [25], 3D system integration offers a 3.9 times increase in wire-limited clock frequency, an 84% decrease in wire-limited area or a 25% decrease in the

number of metal levels required per stratum. There are three feasible 3D integration methods; a stacking of packages, a stacking of ICs and Vertical System Integration as was introduced by IMEC [23]. There are four main enabling technologies for the fabrication of 3D-Integrated Circuits, Bean Recrystallization, Silicon Epitaxial Growth, Solid Phase Crystallization and Processed Wafer Bonding [26]. Table 1.1 shows the main characteristics of each of these 3D fabrication technologies. In this research, however, the focus is on the use of processed wafer bonding technology using an indium bump interconnection array (IBIA). The reason why wafer bonding technology is adopted for this work is because the process has particular benefits for applications where each chip carries out independent processing. The characteristic of the 3D-SoftChip are that each of the two planar chips should be effectively manipulated to maximize computation throughput with parallelism. Also indium has good adhesion, a low contact resistance and can be readily utilized to achieve an interconnect array with a pitch as low as 10μ m. The development of the 3D integrated systems will allow improvements that should be seen in the packaging cost, the performance, the reliability and a reduction in the size of the chips.

3D Fabrication Technologies	Characteristics		
Beam Recrystallization	Deposit poly-silicon and fabricate Thin-film Transistors (TFTs). High performance of TFT's High temperature of melting poly-silicon(Not practical Fab.Tech.) Suffers from low carrier mobility		
Silicon Epitaxial (SE) Growth	Epitaxially grow a single crystal Si High temperature causes degradation in quality of devices Process not yet manufacturable		
Solid Phase Crystallization	Low temperature alternative to SE Flexibility of creating multiple layers Compatible with current processing environments Useful for stacked SRAM and EEPROM cells		
Processed Wafer Bonding	Bond two fully processed wafers together Similar electrical properties on all devices Independent of temperature since all chips are fabricated then bonded / Good for applications where chips do independent processing Lack of precision(alignment) restricts inter-chip communication to global metal line		

Table 1.1: 3D Fabrication Technologies

1.2 Adaptive Computing Systems Overview

There are three types of computing system currently in existence; a general-purpose computing system, a reconfigurable/adaptive computing system and an application specific computing system. The general-purpose computing system is based on using a general-purpose processor for broad applications. Discrete application specific ICs are used for application specific computing systems for dedicated and limited applications. These computing systems have certain drawbacks such as low performance in the case of the general-purpose computing system. The reconfigurable/adaptive computing system, however, allows for an optimum trade-off between flexibility and performance. Because of this fact, reconfigurable/adaptive computing systems are attracting attention as a new alternative for the next generation of computing systems. Figure 1.3 illustrates how the reconfigurable/adaptive computing system provides an optimum trade off between flexibility and performance.



Figure 1.3: Computing systems

1.2.1 Adaptive Computing Systems

1.2.1.1 The Need for Adaptive Computing Systems

The nonrecurring engineering (NRE) costs associated with the design and testing of complex chips are one of the great threatening factors in current system design approaches. According to the International Technology Roadmap for Semiconductors (ITRS), the manufacturing engineering costs of complex chips have reached almost one million dollars. The associated design NRE costs almost reached tens of millions of dollars in year 2003 [21]. Moreover, product life cycles are getting ever shorter due to rapid changes in technology and as a result the time-to-market (TTM) period is keenly shortened. On the other hand, design and verification cycle times are getting longer into the months or even years. As a consequence of these issues, a reconfigurable/adaptive computing system that could be metamorphosed across multiple standards and applications becomes very attractive for the next generation of computing systems.

1.2.1.2 The Concept of Adaptive Computing Systems

A reconfigurable system is one that has reconfigurable hardware resources that can be adapted to the application currently under execution providing the possibility to customize across multiple standards and applications. In most of the previous research the concepts of reconfigurable and adaptive computing have been described interchangeably. In this document, however, these two concepts will be more specifically described and differentiated. Adaptive computing will be treated as a more extended and advanced concept of reconfigurable computing systems, which means it includes more advanced software technology to effectively manipulate the mapping and scheduling of context memory over a wide range of applications along with more advanced reconfigurable hardware resources to support fast and seamless execution across these applications. Table 1.2 shows the differentiations between reconfigurable computing and adaptive computing. The benefits of adaptive computing are silicon reuse, bug-fixing postshipping, updating and fixing in market allowing for standards evolution, faster TTM and lower costs. The reconfiguration capacity allows for significant reuse of silicon. If bugs are found post-shipping or standards evolve, the adaptive computing system is easy to fix and update simply by changing the contexts in the reconfigurable hardware resource. The forthcoming impact from the deployment of adaptive computing is "Do-it-all" devices. A small handheld PDA size device can assume the functionality of about 10 standard devices simply depending on the context programs included such as a cellular phone, a GPS receiver, an MP3 player, an e-book reader, a digital camera, a portable television, a satellite radio, a held-held gaming platform etc. Figure 1.4 shows the futuristic concept of "Do-it-all" devices.



Figure 1.4: An Example of "Do-it-all" Device (*Source: www.chosun.com)

	Reconfigurable Computing	Adaptive Computing	
Hardware Resources	Linear array of homogeneous elements (Logic gates, look-up tables)	Heterogeneous algorithmic elements (Complete function units such as ALU, Multiplier)	
Configuration	Static, Dynamic configuration Slow reconfiguration time	Dynamic, partial run-time reconfiguration.	
Mapping methods	Manual routing , conventional ASIC Design tools (HDL)	High-level language (SystemC,C)	
Characteristics	Large silicon area, Low speed (high capacitance), high power consumption, high cost	Smaller silicon size, high speed, high performance, low power consumption, low cost	

Table 1.2: Re	configurable	Computing	Vs Adaptive	Computing.
---------------	--------------	-----------	-------------	------------

1.2.2. Classification of Adaptive Computing Systems

Adaptive computing systems are mainly classified in terms of granularity, programmability, reconfigurability, computational methods, hardware mapping methods and target applications. The granularity is the basic data size of the reconfigurable hardware resources. In fine grained systems, the primitive reconfigurable hardware resources are typically logic gates, flip-flops and look-up tables and operate using bit-level computations. Field Programmable Gate Array (FPGA) and Complex Programmable Logic Gates (CPLD) are good examples of fine grained systems. In contrast, the coarse grained systems have complete function units such as ALU, multiplier and dedicated functional units and operate using word-level computations. The grained systems and the coarse grained systems creates a mixed grained system.

The programmability relates to the capacity of the configuration. Singleprogrammability allows only one customization, while multiple-programmability allows for customization on-the-fly. The reconfigurability is executed by changing the context memory. Static (interrupted execution) and Dynamic (in parallel execution) are two categories of reconfigurability. Common computational methods used in the adaptive computing systems are Single-Instruction stream Multiple-Data stream (SIMD)/Multiple-Instruction stream Multiple-Data stream (MIMD) and Very-Long Instruction Word (VLIW). The hardware mapping methods vary depending on developed systems from manual routing to high-level language compilation. Most of the target applications for adaptive computing are in the areas of wired and wireless communications and multimedia digital signal processing

1.2.2.1 Previous Works

The research and commercial development of reconfigurable/adaptive computing systems has been going vigorously since the early 1990's. According to the classification of adaptive computing described above, the nature of this research is classified in the Table 1.3[3, 22] and it shows the best-known existing coarse-grain reconfigurable

systems, the fine-grain reconfigurable systems have been excluded because these are different category from our research.

The Matrix [1], REMARC [5] and MorphoSys [3] belong to the category of meshbased reconfigurable systems which is a combination of an array of word-level processing elements with a control processor, such as a multi-granular array of Basic Functional Units (BFUs) in the case of the MATRIX, an 8 by 8 array of 16-bit nanoprocessor with MIPS-II RISC processor in the REMARC, or an 8 by 8 array of reconfigurable cells with MIPS-like processors in the MorphoSys. These are dynamic reconfigurable cells with MIPS-like processors in the MorphoSys. These are dynamic reconfiguration, mesh based hierarchical interconnection fabric architectures. Their application is restricted only to DSP type tasks and they have certain disadvantages in term of the power consumption because of frequent data movement between the control processor and processing elements. As well as, need to access external memory resources.

Another category is a linear array-based reconfigurable system such as, RaPiD [6] or PipeRench [2]. These are linear arrays of processing elements with row-wide interconnection fabrics. Each combination of the processing element array and the rowwide interconnection can make a pipeline stage. The target application of these systems is pipelining regular computation-intensive applications. The other categories such as crossbar-based [35, 36] and reconfigurable processors [37] have been excluded in this table.

The Trisend A7 [10] is considerably similar to other mesh-based reconfigurable systems, the difference is in the granularity of the processing elements. The A7 has a fine-grain reconfigurable fabric in comparison with the word-level processing elements in the mesh-based reconfigurable system.

The MRC6011 [11], Adapt 2400 [16], DFA1000 [9], PC102 [17] are up-to-date commercially developed adaptive computing systems, which have mostly heterogeneous arrays of reconfigurable hardware except the DFA1000 and dynamic configurability. The main target application is computation-intensive multimedia DSP and communication signal processing. These have more advanced adaptive computing characteristics compared with the systems introduced earlier.

As indicated, the early research and development was into single linear array type reconfigurable systems with single and static configuration [8,1,6,5,4,2] but this has evolved to large adaptive SoCs with heterogeneous types of reconfigurable hardware resources and multiple and dynamic configurability. The MRC6011, Adapt2400, DFA100, PC102 and 3D-SoftChip are good examples to show the current research and commercial development directions. The ultimate goal for the adaptive computing system is currently the "Do-it-all" device as explained before.

System	Granukarity	Programmebility	Reconfiguration	Computation Method	Mapping	Target Application
PADDI [8]	Coarse(16bit)	Multiple	Static	VLIW, SIMD	Routing	DSP applications
MATRIX [1]	Coarse(8bit)	Multiple	Dynamic	MIMD	Multi-length	General Purpose
RaPiD [6]	Coarse(16bit)	Single	Mostly static	Lincar array	Channel routing	Systolic arrays, Data-intensive
REMARC [5]	Coarse(16bit)	Multiple	Static	SIMD	N/A	Data-parallel application
RAW [4]	Mixed	Single	Static	MIMD	Switch box routing	General purpose
PipeRench [2]	Mixed(128bit)	Multiple	Dynamic	Pipe-lincd	Scheduling	Data-parallel, DSP applications
MorphoSys [3]	Coarse(16bit)	Multiple	Dynamie	SIMD	Assembler, Manual P&R	Data-parallel, Image applications
Triscend A7 [10]	Mixed	Multiple	Dynamic	N/A	Co-compilation (Assembler, C, Verilog, VHDL)	General Purpose Embedded System
Motorola MRC6011 [11]	Coarse(16bit)	Multiple	Dynamie	SIMD	C-Compilation	Computation Intensive applications
QuickSilver Adapt2400 [16]	Coarse(8,16,24, 32bit)	Multiple	Dynamic	Heterogeneous Nodes array	SilverC	Comm., Multimedia DSP
Elixent DFA1000 [9]	Coarse (4bit)	Multiple	Dynamic	Linear D-Fabric Array	Verilog, VHDL, Handle-C, Matlab	Multimedia applications
picoChip PC102 [17]	Coarse(16bit)	Multiple	Dynamic	3way-LIW	Assembler	Wireless Communications
3D-SoftChip	Coarse(4bit)	Multiple	Dynamie	Various types of computation models	C-compilation (Assembler, C)	Comm. Multimedia Signal Processing

Table1.3: Reconfigurable and Adaptive Computing Systems

÷,

1.2.2.2 MorphoSys Vs 3D-SoftChip

One of the most successful reconfigurable systems to date is the MorphoSys system, so it is meaningful to make a comparison of the proposed 3D-SoftChip architecture to this. Table 1.4 shows the comparison between the MorphoSys and the 3D-SoftChip. It can be seen that the 3D-SoftChip is more appropriate to the most up-to-date adaptive computing system.

	MorphoSys	3D-SoftChip	
Integrated Model	System-on-Chip except main memory	Vertically Integrated complete System-on-Chip with abundant memory capacity	
Memory Interface	Employs a two-set data buffer that enable overlap of computation with data transfers.	Using Indium bump technology, vertical data communication. Variable memory word-length for adaptive computing	
Reconfiguration	Multiple contexts on-chip (32 planes) with dynamic and single- cycle.	Multiple context on-chip with dynamic and single-cycle	
Controller	On-chip general-purpose processor.	Every unit 3D-SoftChip has an ICS_RISC which role of control processor.	
Examples of Application	MPEG-2 Video Compression, Encoder Automatic Target Recognition Data Encryption	Real time communication and multimedia signal processing	
Characters	SIMD nature. Fixed Word length Comprehensive tool sets.(mView, mLoad, mSched, mcc, MuLate, MorphoSim)	Various types of Computational models (SISD, SIMD, MISD, MIMD) And 3 types of SIMD Computation models(massively parallel, multithreaded, pipelined) Configurable word length and variable memory word length for Adaptive Computing. 3D Vertically Integrated System – High speed data interface. Optimum System Architecture for Comm. and Multimedia Signal Processing	

Table1.4: Comparison of MorphoSys with 3D-SoftChip

1.3 Motivation of Thesis

As the microelectronics industry enters the nano and giga-scaled integrated circuit era, many problems, as described before, have been to occur. To cope with these problems, especially the system-on-chip complexity and interconnection crisis, innovative new computing systems with novel interconnection methods will be required. A very promising candidate to overcome these problems is the concept of a 3D vertically integrated adaptive computing system-on-chip (3D-ACSoC). This concept may well be a critical technology for the next generation of computing systems because of its wide applicability/adaptability and because of the significant benefits gained from 3D systems such as reduction in interconnect delays and densities, and reduction in chip areas due to the possibility for more efficient layouts etc.

Conventional SoC design methodologies include many error-prone and tedious iteration processes which can result in a lack of system reliability and extend the design time. Moreover, the portion taken up by verification processes in the total design time is exponentially increasing. By adopting the suggested SoC design methodology using SystemC, the design time can be significantly reduced and more reliable systems can be realised. To satisfy these needs, the concept of the 3D-ACSoC and advanced HW/SW co-design and verification methodology has been suggested.

1.4 Scope of Thesis

In this thesis, the novel 3D-SoftChip architecture for real-time communication and multimedia signal processing is introduced, and its high-level system modelling and functional verification using SystemC is described. The 3D-SoftChip has been fully modelled using SystemC at high-level and implementation of the MPEG4 full search block matching motion estimation algorithm has been mapped to the modelled 3D-SoftChip. Finally, the performance analysis is detailed in the last chapter. The thesis is composed of nine chapters including this one. The following is the scope to be covered by each of the following chapters.

1.4.1. Scope of Each Chapters

Chapter 2 is an introduction to the overall 3D-SoftChip architecture. The novel architecture and several salient features for next generation computing system will be introduced along with the suggested HW/SW co-design and verification methodology. The detailed architecture of the CAP chip will be described in Chapter 3. Heterogeneous types of Processing Elements architecture and functions will be presented. Chapter 4 covers the ICS chip, its components and the ICS_RISC instruction Set Architecture with instruction set summary. Chapter 5 presents the architecture of the UnitChip, its pipeline operation mechanism and area constraint. A three hierarchical interconnection architecture and the configurable nature of the inter-PE bus will be introduced in Chapter 6. Chapter 7 presents the high-level modelling of 3D-SoftChip using SystemC. The simulation result of each component of the 3D-SoftChip is also provided to show the verification of the functionality of the each component. Chapter 8 introduces application mapping for high-level modelled 3D-SoftChip with the MPEG4 full search block matching algorithm. The performance analysis will be performed in comparison with conventional systems. Finally, the last Chapter outlines the contributions of this thesis and suggested future work.

1.5 Conclusions

In this chapter, the motivation for the emergence of the novel 3D vertically integrated system-on-chip and the benefits which can be acquired through its use have been described. This concept will be a promising candidate for the next generation of computing system.

Chapter 2 System Architecture of 3D-SoftChip

In this chapter, the core technology for the 3D-SoftChip along with its detailed and overall architecture will be described. Finally a design guideline and the suggested design methodology will be introduced.

2.1 Core Technology for 3D-SoftChip

The core technology for the 3D-SoftChip can be mainly classified into 3 fields of technology is follows, a Very Deep Submicron (VDSM) silicon process technology, a 3D Interconnection technology and an advanced software technology. The target silicon process technology for the 3D-SoftChip is less than 0.13um to maximize the effect of large scale integration in order to fit as much as possible into the Processing Element (PE)s. This large scale integration into the PEs can be leveraged to amplify the computation capacity of the 3D-SoftChip because of the SIMD computation nature of the 3D-SoftChip. The 3D Interconnection technology using the Indium Bump Interconnection Array (IBIA) is another state-of the art technology for the 3D-SoftChip. The IBIA can cope with the severe demand for data bandwidth from real time communication and multimedia signal processing applications. The last technology is the advanced software technology, which is able to effectively execute context mapping and scheduling within the context memory in the 3D-SoftChip.



Figure 2.1: Core technology for the 3D-SoftChip

2.2 Overall Architecture of 3D-SoftChip

Figure 2.2 shows the overall architecture of the 3D-SoftChip.



Figure 2.2: Overall architecture of the 3D-SoftChip

As can be seen, it is comprised of 4 UnitChips. By including four separate unit chips in the architecture, sufficient flexibility is provided to allow multiple optimized task threads to be processed simultaneously. Given the primary target applications of communication and multimedia processing four UnitChips should be sufficient for all such requirements. Each UnitChip has a PE array, a dedicated control processor and a high bandwidth data interface unit. According to a given application program, the PE array processes a large amount of data in parallel, the ICS controls the overall system and directs the PE array execution and data and address transfers within the system.

2.3 Features of 3D-SoftChip

The 3D-SoftChip has 4 distinctive features: Various types of computation modes, adaptive Word-length configuration [14], optimized system architecture for real-time communication and multimedia signal processing and dynamic reconfigurability for adaptive computing.

Computation Algorithm : Various Computation Models

As described above, one 32-bit RISC controller can supply control, data and instruction addresses to 16 sets of PEs through the completely freely controllable switch block so various computation models can be achieved such as SISD. SIMD, MISD, MIMD as required. Enough flexibility is thus achieved for an adaptive computing (AC) system. Especially, in the SIMD computation model, 3 types of different SIMD computational model can be realized, massively parallel, multithreaded and pipelined SIMD computational models [13]. In the massively parallel SIMD computation model, each UnitChip operates with the same global program memory. Every computation is processed in parallel, maximizing computational throughput. In the Multithreaded SIMD computation model, the executed program instructions in each UnitChip can be different from the others, so multithreaded programs can be executed. The final one is the pipelined SIMD computation model. In this case each UnitChip executes a different pipelined stage. These three computational models are illustrated in Figure 2.3.



(a) Massively Parallel SIMD Computational Model



(b) Multithreaded SIMD Computational Model



(c) Pipelined SIMD Computation Model Figure 2.3: Computation Algorithm: 3 types of SIMD Computation Models

Word-length Configuration

This is a key characteristic in order to classify the 3D-SoftChip as an adaptive computing system. Each PE's basic processing word-length is 4-bit. This can, however, be configured up to 32-bit according to the application in the program memory. Figure 2.4 illustrates the proposed word-length configuration algorithm. When 2 PEs configure together, an 8-bit word-length system is created. If 4 PEs configure together this extends to 16-bit. And finally when 8 PEs configure together a full 32-bit word length is achieved. This flexibility is possible due to the configurable nature of the arithmetic primitives in the PEs [18], (see chapter3.5) and the completely freely controllable switch block architecture in the ICS chip



(a) 8-bit Word-length Configuration



(b) 16-bit Word-length Configuration



(c) 32-bit Word-length Configuration

Figure 2.4: Word-length Configuration Algorithm

Optimized System architecture for Communication and Multimedia Signal Processing

There are many similarities between communications and multimedia signal processing, such as data parallelism, low precision data and high computation rates. The different characteristics of communication signal processing are basically more data reorganization such as matrix transposition and potentially higher bit level computation. To fulfill these signal processing demands, each UnitChip contains two types of PE. One is a standard-PE for generic ALU functions, which is optimized for bit-level computation. The other is a processing accelerator-PE for Digital Signal Processing (DSP). In addition, special addressing modes to leverage the localized memory along with 16 sets of Loop buffers to generate iterative

address in the ICS_RISC add to the specialized characteristics for optimized communication and multimedia signal processing.

Dynamic Reconfigurability for Adaptive Computing

Every PE contains a small quantity of local embedded SRAM memory and additionally the ICS chip has an abundant memory capacity directly addressable from the PEs. With multiple sets of program memory and the abundant memory capacity, it is possible to switch programs easily and seamlessly, even at run-time.

2.4 System Components

As introduced above, the 3D-SoftChip consists of a linear array of heterogeneous PEs with an associated array of Indium bump 3D Interconnects, dedicated Switch Blocks, the ICS_RISC and a high bandwidth data interface unit.

2.4.1. Configurable Array Processor (CAP) Chip

2.4.1.1 Heterogeneous Types of PEs

The CAP chip comprises a linear array of two types of PE, a Standard-PE and a Processing Accelerator-PE. The advantages of heterogeneous PEs with dedicated functions for special purpose DSP are more suitability for specific applications with only a medium flexibility trade-off compared with homogeneous type PEs. In this case, two Standard-PE and two Processing Accelerator-PEs form one Quad-PE. These will be in detail in a later section.

2.4.2. Intelligent Configurable Switch (ICS) Chip

2.4.2.1 Switch Block

Each group of 4 PEs (Called Quad-PEs) are controlled by one Switch Block through the IBIA. This transfers data from/to each PE and also provides instruction data for the
PEs. It can completely freely configure each PE group, and makes it possible to achieve efficient variable word-length configuration.

2.4.2.2 ICS_RISC

A 32-bit dedicated RISC processor is used to control each set of 4 Quad-PEs (called UnitCAP). It controls the execution of the PE array and provides control and address signals to the Switch Block and the high bandwidth data interface unit in the UnitChip.

2.4.2.3 Data Frame Buffer

Two sets of Data Frame Buffers are included to support the transfer of large volumes of data from/to data/program memory and the ISC.

2.4.2.4 Program Memory

This is separated into two areas. One is a program memory for the ICS_RISC and the other is the program memory for the PE array. This memory supports adaptively configured word-lengths to increase the computation efficiency dependent on the application. Additionally, multiple sets of program memory are included to allow dynamic program switching.

2.4.2.5 Data Memory

Abundant memory capacity is one of the characteristic of the 3D-Softchip with each PE containing its own embedded local memory along with a high bandwidth connection to the memory store on the ICS.

2.4.2.6 DMA Controller

A dedicated controller is included to facilitate the transfer of large volumes of data from/to program memory, data memory and the ICS. This provides a high efficiency data interface between any of these units.

2.4.2.7 3D Interconnection Technology

The CAP chip carries out all data manipulation operations in the system. There is rarely the need for data transfer within the CAP beyond basic nearest neighbor interconnects, except for computation with word-lengths configured to > 4-bit. All the manipulated data is, therefore, transferred through the Indium Bump Interconnection Array (IBIA) and processed by the ICS allowing for very high speed computation because the IBIA provides very high bandwidth and very low inductance/capacitance [15].

2.5 Design Guidelines

The design guidelines and constraints to satisfy the design goals are as follows.

- The 3D-SoftChip is the first novel 3D vertically integrated Adaptive Computing System-on-Chip (3D-ACSoC)
- Using Indium bump technology, data can be manipulated at very high speed with wide bandwidth.
- The variable memory word-length and configurable word-length are unique features for an adaptive computing system.
- Various computation models (SISD, SIMD, MISD, MIMD) are possible for adaptability/flexibility in accordance with the current application and 3 types of SIMD computational models (massively parallel, multithreaded, pipelined) allow for maximized computational throughput.
- The heterogeneous types of PE architectures are optimized for communication and multimedia signal processing.
- Dynamic run-time reconfigurability for adaptive computing. (Multiple sets of program memory and abundant memory capacity)
- The area constraint of PE should be minimized as much as possible (less than 60um × 60um in 0.13um technology) for a 4-bit word size.

2.6 Design Methodology

2.6.1. Suggested HW/SW Co-design and Verification Methodology

HW/SW co-design is a development methodology that supports the concurrent and cooperative development of hardware and software (co-specification, co-development, coverification). It helps to evaluate the effect of design decisions and to explore the design space at an early stage to obtain the optimal architecture. As a result of this, design cost and design cycle time can be reduced and more reliable system can be realised because of the verification at the high-level of the system. Figure 2.5 shows a suggested HW/SW codesign methodology for the 3D-SoftChip. Once the system specification is firmly decided, HW/SW partitioning is executed to determine which functions should be implemented in hardware and which in software. The HW can then be modeled using SystemC [19] and SW modeled in C. After that, a co-simulation and verification process is implemented to verify the 3D-SoftChip operation and performance and to decide on an optimal HW/SW architecture.

More specifically, the SW is modelled using a modified GNU C Compiler and Assembler. After the compiler and assembler for ICS_RISC has been finalised, a program for the implementation of the MPEG4 motion estimation algorithm will be developed and compiled using it. After that, object code can be produced, which can be directly used as the input stimulus for an instruction set simulator and system level simulation. The HW/SW verification process can be achieved through the comparison between the results from instruction level simulation and system level simulation. From this point on, the rest of the procedure can be processed using any conventional HW design methodology, such as full and semi-custom design. N.B. SystemC is a system design language which supports concurrent HW/SW co-design methodologies and offers a simulation kernel that supports hardware modeling concepts at the system level, behavioral level and register transfer level [20].

2.7 Conclusions

The core technology, overall and detailed architecture of the 3D-SoftChip has been presented. The four kinds of salient features, as described Section 2.3 can differentiate the 3D-SoftChip from conventional reconfigurable/adaptive computing systems. The design time and reliability of the system will be significantly improved by adopting the suggested HW/SW co-design and verification methodology using SystemC.



Figure 2.5: Suggested HW/SW Co-design and Verification Methodology

Chapter 3 Architecture of CAP Chip

In this chapter, the overall architecture of the Configurable Array Processor (CAP) chip will be described along with the PE architecture for communication and multimedia signal processing. The integration of 4 heterogeneous PEs forms one Quad-PE, and four Quad-PE make up the UnitCAP chip.

3.1 Overall Architecture of CAP Chip

The basic architecture of the CAP chip is a linear array of heterogeneous PEs. Figure 3.1 shows three possible architecture choices for the PEs. The architecture in Figure 3.1(b) is suggested as the most feasible architecture for the PE in the 3D-SoftChip because it has the optimum trade-off between application specific performance and flexibility. Examples of type A can be seen in [1,2,3], type B in [16] and type C in [17]. The CAP chip has the basic role of the processing engine for the 3D-SoftChip. It manipulates large amounts of data at a high computational rate using any of the three different SIMD computation models previously described.



Figure 3.1: Types of PEs (a) homogeneous type, (b) heterogeneous type, (c) heterogeneous type with dedicated functions for special purpose.

	PE Architecture	Flexibility	Performance
Туре А	Homogeneous type PEs with Embedded memory, ALU, MAC, Address decoder etc.	Suitable for general purpose High flexibility	Relative low performance for specific applications.
	Each PEs are optimized for special functions		
<u>Type B</u>	De B Example : Suitable for • PE1: Multiple MAC, ALU array Media • PE2: Bit-oriented operations Media • PE3: General purpose RISC or Control Logic. • PE4: Memory PE4: Memory	Suitable for specific applications. Medium flexibility	Relative medium performance for specific applications
	Combination of the Type B arch. with dedicated functions for special purpose		
Type C	Example : • PE1: Multiple MAC, ALU array • PE2: Memory and Control • PE3,PE4: A Co-processor optimized for dedicated signal processing functions (FEC, Preamble detect etc)	Suitable for dedicated applications. Low flexibility	Relative high performance for specific applications

Table 3.1: Characteristics of each PE types

3.2 Two Types of Processing Element (PE)s

Figure 3.2 illustrates the two type of PE architecture chosen to optimize communication and multimedia signal processing type applications. Table 3.2 shows the characteristics of the two type of PE.



(a) Standard-PE

(b) Processing Accelerator-PE

Figure 3.2: Two Types of PE

Table 3.2: Cl	haracteristics of	of the	two	type of I	PE
---------------	-------------------	--------	-----	-----------	----

	Standard-PE	Processing Accelator-PE
Components	Standard ALU(Mul, Add, Sub, Comparator) MUX A,B, Registers, Embedded SRAM	Multiplier, modified Adder, Subtractor, 8-bit Barrel Shifter, Registers, Embedded SRAM
Purpose	Bit-wise manipulation, Standard ALU functions	Dedicated for MAC, MAS functions (for DSP application)
Characteristics	Standard ALU functions Comparison operation.	Single clock cycle MAC, MAS absolute value computation operations 8bit barrel shifter. (Logical, Arithmetical Shift)

3.2.1. Standard-PE (S-PE)

The S-PE is for standard ALU functions and is also optimized for bit-level operation for communication signal processing. It comprise 4 sets of 19-bit registers for S-PE instruction decoding, two multiplexers to select input operands from the data bus, adjacent PEs or internal registers, a standard ALU with bit-serial multiplier, adder, subtractor and comparator, embedded local SRAM and 4 sets of Registers. The arithmetic primitives are scalable so as to make it possible to reconfigure the word-length for specific tasks. The scalable architecture arithmetic primitive architecture is presented in [18].

3.2.2. Processing Accelerator-PE (PA-PE)

The PA-PE is dedicated specifically for Digital Signal Processing (DSP) operations. It consists of 4 sets of 19-bit registers for PA-PE instruction decoding, two multiplexers to select input operands from the data bus, adjacent PEs or internal registers, a signed 4-bit scalable parallel/parallel multiplier, and accumulator/subtractor modified to enable Multiple-and-Accumulate (MAC), Multiple-and-Subtract (MAS) operations within one clock cycle, an 8-bit configurable barrel shifter, embedded local SRAM and 4 sets of Registers. Two shifters in the Quad-PE can also be configured to produce a 16-bit barrel shifter. Its distinctive features are the single clock cycle MAC, MAS operations and parallel/parallel multiplier to accelerate DSP applications. Moreover it can execute single clock cycle absolute value computation.

3.3 PE Functions

PE functions are mainly divided into S-PE or PA-PE functions.

3.3.1. Standard-PE Functions

Table 3.3 shows the functions of S-PE. It is useful for bit-wise manipulation and generic ALU functions.

Function	Mnemonics
A and B	AND
A or B	OR
not A	NOT
A xor B	XOR
A + B	ADD
A – B	SUB
A×B	SPMUL
A comp B	COMP

Table 3.3: Standard-PE Functions

3.3.2. Processing Accelerator-PE Functions

Table 3.4 describes the PA-PE functions. It is specialized for DSP such as MAC, MAS, logical Shift, Arithmetic Shift, Rotate function, absolute value computation.

Function	Mnemonics
A×B	PAMUL
$A \times B + out(t)$	MAC
$A \times B - out(t)$	MAS
Logical Shift Left	LSL
Logical Shift Right	LSR
Arithmetic Shift Right	ASR
Rotate	ROR
A (Absolute value)	ABS

Table 3.4: Processing Accelerator-PE Functions

3.3.3. PE Instruction Formats and Operation Modes

The PE instruction format consists of a 19-bit instruction word. The most significant 2-bits, 18 and 17 in the instruction word (WS_en/RS_en, WR_en/RR_en) are used for the Read/Write enable bit of the embedded SRAM and registers. Bits 16 to 10 are used for SRAM and register selection (addressing). Bit 9 is used for data output register enable signal and bits 8 to 6 are used to specify the PE operation. Finally, bits 5 to 0 are used to control the input multiplexers for input operand selection. This format is illustrated in Figure 3.3 below.

18	17	16	15 12	11 10	9	8 6	5 3	2 0
WS_en/	WR_en/	SRAM	SRAM Selection	Register	Dout	SPEOR	MUXB	MUXA
RS_en	RR_en	en		Selection	RCtl			
			(a) Standard-PE	Instruction fo	rmat			
18	17	16	15 12	11 10	9	8 6	5 3	2 0
18 WS_en/	17 WR_en/	16 SRAM	15 12 SRAM Selection	11 10 Register	9 Dout	86 PA-PE	5 3 MUXB	2 0 MUXA
18 WS_en/ RS_en	17 WR_en/ RR_en	16 SRAM en	15 12 SRAM Selection	11 10 Register Selection	9 Dout RCtl	8 6 PA-PE OP	53 MUXB	2 0 MUXA
18 WS_en/ RS_en	17 WR_en/ RR_en	16 SRAM en (b	15 12 SRAM Selection	11 10 Register Selection	9 Dout RCtl	8 6 PA-PE OP	53 MUXB	2 0 MUXA



Figure 3.4: PE Array Operation Modes

Figure 3.4 illustrates 3 types of PE operation modes that can be realized on the PE array; Horizontal mode, Vertical mode and Circular mode. In the horizontal and vertical mode, the each rows or columns of the PEs can connected together respectively. These operation modes optimized for the SIMD computational method. Lastly, in the circular mode, the PEs in the one Quad-PE connects together and each Quad-PE can work separatively. These allow for even greater flexibility and help to maximize computational throughput according to the target application.

3.4 Embedded Local SRAM

Each PE has a small quantity of local embedded SRAM. As the effective memory bandwidth is increased dramatically by as much as the number of the PEs, which will result in an increase in effective processing speed in many applications. Bus traffic can also be reduced because many data transmission operations can be contained within a PE. Consequently, a lowering of power dissipation will also be achieved. Effectively this can act as cache, which can be continuously refreshed

3.5 Configurable Nature of Arithmetic Primitives

As described in the Chapter 2.3, one of the distinguished features as an adaptive computing system is the word-length configuration. The basic word-length of each PE is 4-bit. It can be configured 8, 16, 32-bit according to the target application. The

configurable nature of the arithmetic primitives in the PE allows this configuration [18]. The most complex component in the PE is multiplier so the example of configurable arithmetic primitives, the configurable parallel multiplier will be introduced

3.5.1. Scalable Parallel Multiplier Cell

Figure 3.5 shows a generic 1×1-bit multiplier cell. It includes a full adder, an AND gate and three multiplexers to select the input operand through the control signals CTRLH and CTRLL. In this figure, A represents the multiplicand and B is the multiplier. SIN is the SUM signal from the adjacent cell above, Cour is the propagated carry output, CIN is the carry input from the adjacent multiplier cell. Mour represent the multiplication result. The 2×2-bit multiplier can be implemented using the generic 1-bit cell and moreover an 8×8-bit multiplier can be realised by arranging the basic 4×4 primitive in a 2×2 array as shown in figure 3.6 [18]. Because of this configurable characteristic, the word-length can be extended up to 32-bit in the 3D-SoftChip



Figure 3.5: A generic 1×1-bit Multiplier Cell for n=1

-45-



Figure 3.6: 8×8 multiplier using 4-bit Generic Cells

3.6 Quad-PE

As previously described one Quad-PE consists of two pairs of PEs (two S-PE and two PA-PE). The Quad-PE is controlled and configured by the Switch Block according to the control and address data from the ICS_RISC transmitted through the IBIA. Figure 3.7 shows the architecture of a single Quad-PE.



Figure 3.7: Quad-PE

3.7 UnitCAP Chip Architecture

The CAP chip consists of 4 sets of UnitCAP. Each UnitCAP has an array of 16 heterogeneous S-PEs and PA-PEs. Figure 3.8 shows the UnitCAP chip architecture. The configurable interconnectivity is realised through the input multiplexer in each PE. The detailed description of interconnection between the PEs will be described in Chapter 6.

3.8 Conclusions

The heterogeneous types of PE architecture for communication and multimedia signal processing have been described. The adoption of the PE architecture can accelerate the

performance where intensive bit-level computation and digital signal processing is required and achieve more flexibility compare with homogeneous types of PE array. The suggested PE architecture has been fully modelled and its functionality verified using SystemC at high-level. The details regarding the system level modelling of the PE will be introduced in Chapter 7.



Figure 3.8: UnitCAP Chip Architecture

Chapter 4 Architecture of ICS Chip

The ICS chip comprises the Switch Blocks, ICS RISC, program memory, data memory, data frame buffers and DMA controller. The ICS chip is a control processor which controls the CAP chip via the IBIA as well as the overall system. The ICS_RISC provides control and address signals and data to the system as a whole. The switch blocks configure each PE based on the current program instruction. The high bandwidth data Interface Unit enables efficient transmission of data and instructions within the system. In this chapter, the detailed architecture of the ICS chip is described.

4.1 Switch Block

The Switch Block provides data from/to each PE and also provides instruction data to each PE. Three types of Switch Block, 6-sided, 7-sided and 8-sided provide optimized interconnection within the ICS chip. Figure 4.1 shows the Switch Block architecture which connects between the PEs and other Switch Blocks. The architecture of the Switch Block is similar to conventional Switch Blocks in Field Programmable Gate Arrays (FPGA) [32]. The lines in the figure represent switches to connect data/instruction data within the PEs, Switch Blocks and the ICS chip. A pass transistor design is used to optimize performance and minimise area, allowing a completely free configuration for each PE.

4.2 ICS_RISC

The ICS_RISC is a 32-bit dedicated RISC control processor. The ICS_RISC controls the execution of the PE array and provides control and address signals to program/data memory, the data frame buffers and the DMA controller. It has a 3 stage pipelined architecture that is Fetch (F), Decode (D) and Execute (E). To cope with the iterative nature of DSP arithmetic, it has 16 sets of loop buffers so as to provide direct instruction to instruction decoding instead of fetching from program memory in each case. This significantly reduces bus utilization allowing for improved performance and lower power dissipation. Moreover 32 general purpose registers and specialized addressing modes are provided for optimized communication and multimedia signal processing. For detailed architecture descriptions refer to Appendix B.







Figure 4.2: Architecture of ICS_RISC 32-bit dedicated Control Processor

4.2.1 Features of ICS_RISC

The ICS_RISC has a simple and efficient architecture. It has a harvard architecture and simple 3 stage pipelined architecture. Memory access during the execution stage is carried out using load/store instructions only and all operations, except load/store, PE and DMA operations, are register-to-register within the ICS_RISC. This provides improvements in the performance and power dissipation.

4.2.2 System Components of ICS_RISC

The ICS_RISC consists of a 32×32 -bit general purpose register, a program counter which is the 32th general purpose register, a 16×32 -bit loop buffer to generate instruction addresses for iterative sets of instructions, a status register (N:Negative/Less than, Z:Zero, C:Carry/Borrow, V:Overflow), an instruction register for instruction decoding, ALU, shifter, multiplier and 32-bit data input/output registers [30,31]. Figure 4.3 shows the detailed architecture of the ICS_RISC.



Figure 4.3: A detailed architecture of the ICS_RISC

4.2.3. Types of Instruction Set

Table 4.1 describes the instruction set and instruction processing components of the 3D-SoftChip. All control instructions are executed in the ICS chip, while computation instructions, such as arithmetic and logical operations for PEs are executed in the CAP chip using various computation methods (SISD, SIMD, MISD, MIMD). The detailed instruction set is described in Appendix A.

Function	Processing Component
Move	ICS
Arithmetic (S-PE, PA-PE)	CAP
Logical (S-PE)	CAP
Arithmetic	ICS
Logical	ICS
Branch	ICS
Load	ICS
Store	ICS
Addressing Mode/Loop Buffer Addressing	ICS
PE Control	ICS

Table 4.1:	Types	of Inst	ruction	Set
------------	-------	---------	---------	-----

PE Configuration	ICS
Program/Data Load (ICS,PE Program/ Data for PE)	ICS
DMA Control	ICS

4.2.4. ICS_RISC Instruction Set Architecture - Version 1.0

Table 4.2 shows the instruction set architecture (ISA) for ICS_RISC. This is the first version of the ISA, more efficient and dedicated instructions can be added is needed. It has 50 instructions, largely divided into arithmetic and logic, branch, data transfer, bit and bit-test, PE control, DMA control and lastly a loop buffer instruction.

Mnemonic	Operation Operand		Flags
ARITHMETI	C AND LOGIC INSTRUCTIONS		
ADD	Add Two Registers	Rd, Rs1, Rs2	N,Z,C,V
ADDI	Add Register and Constant	Rd, Rs1, #I	N,Z,C,V
SUB	Subtract Two Registers	Rd, Rs1, Rs2	N,Z,C,V
SUBI	Subtract Register and Constant	Rd, Rs1, #I	N,Z,C,V
MUL	Multiply Two Registers	Rd, Rs1, Rs2	N,Z,C,V
MULI	Multiply Register and Constant	Rd, Rs1, #I	N,Z,C,V
AND	Logical AND Registers	Rd, Rs1, Rs2	N,Z,C,V
ANDI	Logical AND Register and Constant	Rd, Rs1, #I	N,Z,C,V
OR	Logical OR Registers	Rd, Rs1, Rs2	N,Z,C,V
ORI	Logical OR Register and Constant	Rd, Rs1, #I	N,Z,C,V
XOR	Logical XOR Registers	Rd, Rs1, Rs2	N,Z,C,V
XORI	Logical XOR Register and Constant	Rd, Rs1, #I	N,Z,C,V
NOT	Logical NOT Registers	Rd, Rs1, Rs2	N,Z,C,V
NOTI	Logical NOT Register and Constant	Rd, Rs1, #I	N,Z,C,V
BRANCH IN	STRUCTIONS		
BREQ	Branch if Equal (Z=1)	PC, Offset	None
BRNE	Branch if NOT Equal (Z=0)	PC, Offset	None
JMP	Unconditional Branch (PC=PC+Offset)	PC, Offset	None
CMP	Compare Registers	Rs1, Rs2	N,Z,C,V
CMPI	Compare Register and Constant	Rd, #I	N,Z,C,V
DATA TRA	NSFER INSTRUCTIONS		
MOVA	Move between Registers (Rd=Rs1)	Rd, Rs1	None
MOVAI	Move between Reg. & Const. (Rd=Const)	Rd, #I	None
MOVB	Move between Registers (Rd=Rs2)	Rd, Rs2	None
MOVBI	Move between Reg & Const. (Rd=Const)	Rd, #I	None
MSR	Move Register to Status Register(SR=Rs1)	SR, Rs1	None

Table4.2: Instruction Set Summary (ICS_RISC ISA Version1)

MSRI	Move Imm value to Status Register(SR=#I)	SR, #I	None
MRS	Move Status Register to Register(Rs1=SR)	Rs1, SR	None
LD	Load indirect with Register (Rd=Mem[Rb])	Rd, Rb	None
ST	Store indirect with Register (Mem[Rb]=Rd)	Rd, Rb	None
BIT AND BI	T-TEST INSTRUCTIONS		
LSL	Logical Shift Left	Rd, Rs1	N,Z,C,V
LSR	Logical Shift Right	Rd, Rs1	N,Z,C,V
ÁSR	Arithmetic Shift Right	Rd, Rs1	N,Z,C,V
ROT	Rotate	Rd, Rs1	N,Z,C,V
PE CONTRO	OL INSTRUCTIONS		
PECON4	PE Word-Length Configuration (4-bit)	None	None
PECON8	PE Word-Length Configuration (8-bit)	None	None
PECON16	PE Word-Length Configuration(16-bit)	None	None
PECON32	PE Word-Length Configuration (32-bit)	None	None
PESEL	Select certain PE (PEO~PE15)	None	None
PEMODH	PE Operation mode (Horizontal mode)	None	None
PEMODV	PE Operation mode (Vertical mode)	None	None
PEMODC	PE Operation mode (Circular mode)	None	None
PEEXEH	Execute specific program to each PEs in the	None	None
DEEVEN	same Horizontal line	Neza	News
PEEXEV	same Vertical line	INORE	INONE
PEEXEC	Execute specific program to each PEs in the	None	None
			<u> </u>
DMA INSTR	RUCTIONS		
LDPEPRG	Load Program Data from Program memory to Instruction decoder in PE.	addrMem	None
LDDFB	Load large amount of processing data for PEs	addrMem,	None
	from Data Memory to Data Frame Buffer	addrDFB	
LDPEDATA	Load large amount of processing data for PEs	addrDFB,	None
	from DFB to Embedded SRAM in PE	addrSRAM	
WBREG	Write back processed data in Embedded	addrSRAM	None
	SRAM to the registers in the ICS_RISC		
WBDFB	Write back processed data in Embedded	addrSRAM,	None
	SRAM to DFB	addrDFB	
STDFB	Load large amount of processed data in PEs	addrDFB,	None
	from Data Frame Buffer to Data Memory	addrMem,	
LOOP BUFE	FER INSTRUCTION		
LBEN	Generate an Iterative Set of Instruction	PC	None
	Addresses		
	(16 sets of Loop Buffer)		

4.3 High Bandwidth Data Interface Unit

The high bandwidth data interface unit allows the efficient transfer of data within the 3D-SoftChip. Two sets of data frame buffer and the DMA controller make it easy to transfer large amounts of data. Multiple sets of program memory support run-time program switching and, because of this dynamic reconfigurable feature, adaptive computing is possible. The data memory has a variable word width so it can easily be combined to build wider/deeper memories and thus increase flexibility for different application programs. The DMA instructions and data flow for the DMA controller can be seen in Figure 4.4. A detailed description of the operations of the DMA instructions can be seen in Appendix A.



Figure 4.4: DMA Controller Architecture and Instructions for DMA Controller

4.4 Conclusions

The ICS chip architecture has been described in this chapter. The system components in the ICS Chip allow it to efficiently supply data and instructions to the PEs through the IBIA. The PE array can be freely configured due to the highly controllable characteristic of the switch block. This allows more than sufficient adaptability/flexibility for adaptive computing systems. Moreover, the DMA controller enables transfer of the bulk data fast and effectively through the 3D-SoftChip.

Chapter 5 Architecture of UnitChip

The 3D-SoftChip consists of 4 sets of UnitChip. Each UnitChip has one UnitCAP and one UnitICS. As described in the chapter 3, the UnitCAP comprises 16 sets of heterogeneous arrays of S-PEs and PA-PEs and the UnitICS consists of a switch block, a 32-bit dedicated RISC control processor, a high bandwidth data interface unit, 2 sets of data frame buffers and program/data memory for both the ICS and the PE array. In this chapter, the UnitChip architecture and its pipeline operation mechanism which can maximize the computational throughputs [3], are described.

5.1 UnitChip Architecture

As mention above, the UnitChip is a combination of the UnitCAP and the UnitICS chip and four UnitChips form the complete 3D-SoftChip. Figure 5.1 illustrates the overall architecture of the UnitChip. The control, data and instructions transfer through the IBIA to the UnitCAP, and the processed data from the UnitCAP can be rapidly transferred back to the ICS_RISC to be manipulated and stored in the data memory.



Figure 5.1: Overall Architecture of the UnitChip.

5.2 Pipelined Operation Mechanism of UnitChip

	Stage (1)	Stage (2)	Stage (3)	Stage (4)	Stage (5)	Stage (6)
ICS_RISC Instructions	LDPEPRG PESEL, PEMODH,V,C	LDDFB	LDDFB PESEL PEMODH,V,C	LDPEPRG PECON4,8,16,32 PEEXEH,V,C	PECON4,8,16,32 PEEXEH,V,C WBREG,WBDFB	PECON4,8,16,32 PEEXEH,V,C WBMEM
PEs Op.				Execute (1-1)	Execute (1-2)	Execute (2-1)
PROGRAM for PEs (in Local memory)	Load PRGM for PEs (1)		PRGM for PEs(1-1)	Load PRGM for PEs (2)	PRGM for PEs(2-1)	PRGM for PEs(2-2)
Data Frame Buffer 0		Load Data for PEs (1)	Data for PEs (1-1)	Data for PEs (1-2)	Write back Execution (1-1) results	
Data Frame Buffer 1			Load Data for PEs (2)		Data for PEs(2-1)	Data for PEs(2-2)
Memory						Write back Execution(1-1) results

Table5.1: Pip	elined UnitChip	Operation	Mechanism
---------------	-----------------	-----------	-----------

* Dark-sided boxes: DMA Control Instructions

Table 5.1 illustrates the pipelined operation mechanism of UnitChip to improve its performance. The detailed explanation is as follows.

- STEP 1 LOAD PROGRAM FOR PEs: The first operation is to load 16 instruction words for PEs from program memory to the instruction decoder in the each PE, the row and column decoder in the UnitCAP can specify a certain PE to load the programs, depending on the desired computational mode (e.g, SIMD, MIMD).
- STEP 2 LOAD PROCESSING DATA FOR PEs (1): Load large amount of processing data for PEs from data memory to data frame buffer. The start address of memory and an amount of data to transfer can be indicated by the DMA instructions.
- STEP 3 LOAD PROCESSING DATA FOR PEs (2): Load the processing data for PEs from data frame buffer to embedded SRAM in each PEs. The row and column decoder in the UnitCAP can specify a certain PE to load the processing data.
- STEP 4 EXECUTE PEs: Execute the PE array
- STEP 5 RELOAD PROGRAM FOR PEs (1): Reload 16 instruction words from program memory to the instruction decoder in each PE, the row and column decoder in the UnitCAP can again specify a certain PE to load the programs to.
- STEP 6 RELOAD PROCESSING DATA FOR PEs (2): Reload the processing data for PEs from data frame buffer to each PEs, the row and column decoder in the UnitCAP can specify a certain PE to load the data into.
- STEP 7 WRITE BACK PROCESSED DATA TO DFB: Write back processed data from embedded SRAM in each PEs to data frame buffer
- STEP 8 TRANSFER PROCESSED DATA TO Memory: Transfer large amount of processed data from data frame buffer to memory

5.3 Area Estimations and Constraints

Table 5.2 shows the feasible estimated area of 3D-SoftChip components. The performance of the integrated circuits largely depends on integration density. The tight area constraints can be achieved through more integration density, which means it can maximize benefit from large scaled integration. The area constraints should be tight in order to achieve the best performance.

Component	Estimated Area
S-PE	60 um × 60 um
PA-PE	60 um × 60 um
IBIA	15 um × 15 um
One Quad-PE	130 um × 130 um
UnitCAP	500 um × 500 um
CAP(4×4 UnitCAP)	1100 um × 1100 um
CAP(16×16 UnitCAP)	2200 um × 2200 um
ICS_RISC	300 um × 300 um

Table 5.2: Area Estimation and Constraint of UnitChip (Target Technology: 0.13 um Process)

5.4 Conclusions

As explained above, by using the pipeline operation mechanism that is a 6-stages pipelined architecture, the performance of the UnitChip can be 6 times more improved. This pipelined operation is another distinguished character to accelerate the computational throughput as the computation is executed simultaneously as much as the pipelined stages.

Chapter 6 Interconnection Network

In this chapter, the three hierarchical interconnection architectures: Inter-PE bus, Switch Block Array interconnection and IBIA, will be introduced along with the configurable nature of the Inter-PE bus using the input operand multiplexer in each of the PEs.

6.1 Hierarchical Interconnection Architecture

The interconnection network of the 3D-SoftChip can be broken into three hierarchical levels. The Inter-PE bus between PEs in the CAP chip is the first level. This local interconnection network has a 2D-mesh architecture providing nearest-neighbor interconnection between the PEs. The second level of the interconnection network is the switch block array interconnection. This supports longer interconnections on the ICS chip but also has a basic 2D-mesh architecture. The last hierarchical level of interconnection is the IBIA. With progression of technology to ever decreasing semiconductor geometry scales, the prediction of interconnection delay and the portion of interconnection delay in the total system delay are crucial factors. It is also a major factor in the limitation of overall system performance. To overcome these problems, 3D interconnection technology using Indium bump becomes very attractive because it supports a very high bandwidth coupled with a very low inductance/capacitance (and thus low power dissipation) and can be readily utilized to achieve an interconnect array with a pitch as

low as 10μ m. The development of 3D integrated systems will allow improvements in packaging costs, performance, reliability and a reduction in the size of the chips [15]. However, any other equivalent 3D interconnection technology could also be applied to realize this interconnection level within the 3D-SoftChip architecture. Figure 6.1 shows the three hierarchical interconnection networks.



(a) PE Array Interconnection Network: 2D-mesh interconnection for local interconnection



(b) Switch Block Array Interconnection Network: 2D-mesh interconnection for long interconnection

-62-



(c) Indium Bump Interconnection: Single indium bump after reflow Figure 6.1: Three hierarchical Interconnection Networks

6.1.1. PE and Switch Block Array Interconnection

6.1.1.1 Programmable Nature of PE Array Interconnection





IPB Signal Name	Source(Output)	Destination(Input)	
IPB 1	SPE1(dOutadjPE)	PAPE1(dLeft)	
IPB2	SPE1(dOutadjPE)	PAPE2(dUp)	
IPB3	PAPE1(dOutadjPE)	SPE1(dRight)	
IPB4	PAPE1(dOutadjPE)	SPE2(dUp)	
IPB5	PAPE1(dOutadjPE)	Next Quad-PE(SPE1(dLeft))	
IPB6	PAPE2(dOutadjPE)	SPE1(dDown)	
IPB7	PAPE2(dOutadjPE)	SPE2(dLeft)	
IPB8	PAPE2(dOutadjPE)	Downside Quad-PE(SPE1(dUp))	
IPB9	SPE2(dOutadjPE)	PAPE1(dDown)	
IPB10	SPE2(dOutadjPE)	PAPE2(dRight)	
IPB11	SPE2(dOutadjPE)	Next Quad-PE(PAPE2(dLeft))	
IPB12	SPE2(dOutadjPE)	Downside Quad-PE(PAPE1(dUp))	

Table6.1: Inter-PE Bus (IPB) interconnection connectivity

Figure 6.2 shows the Quad-PE architecture and Inter-PE interconnection architecture [3]. Because of the input multiplexer in each PE, the connectivity can be readily configured. The input multiplexer can choose certain input operands from among the 6 different inputs; data input, data from left side, right side, upward side and down side PE (dIn, dLeft, dRight, dUp, dDown) and each PE's output (dOutadjPE) becomes input operand to the neighbour PEs. Table 6.1 describes the connectivity within one Quad-PE and indicates that it can be configured by the PE programming according to the target application.

6.1.2. Indium Bump Interconnection

Indium is an excellent material to use as an interconnect material due to its excellent adhesion to most metals, including aluminum, which is the metallization for the pads used in most VLSI technologies. Indium has a low melting point, which implies a low work hardening coefficient, allowing for direct bonding on processed VLSI wafers. Additionally, it provides excellent mechanical as well as electrical connectivity (contact resistance < 1 m Ω per bump). Reflow techniques can be used for flexibility and to increase the bump height to width ratio as needed. Such techniques can also be used to incorporate self-alignment features to the bonding process. Figure 6.3 illustrates 3D filpchip wafer bonding technology using indium bump interconnection arrays.



Figure 6.3: 3D Flip-Chip wafer bonding technology using Indium Bump Interconnection Arrays

6.2 Conclusions

The three hierarchical interconnection network architectures have been described. With the exception of the 3D interconnection there are similar to conventional interconnection architectures in reconfigurable systems. The Inter-PE bus provides configurable connectivity with 2D mesh architecture and the switch block interconnection offers longer interconnection in the ICS chip. Lastly, the IBIA presents vertical interconnection between the two separated chips providing a high bandwidth, high speed, low power memory bus, reducing and eliminating the needs for external memory resources.

Chapter 7 High-level modeling of 3D-SoftChip using SystemC

In this chapter, the high-level modelling of 3D-SoftChip using SystemC will be introduced. Firstly, an overview of SystemC, Computer Aided Design (CAD) environment for SystemC will be briefly described, followed by a presentation of the high-level simulation output waveforms for each of the 3D-SoftChip components and analysis of these. Finally, some conclusions are provided.

7.1 SystemC Overview

SystemC is a C++ class library and design methodology which can effectively design a software algorithm, hardware architecture, interface with SoC and system level designs. System-level modelling, quick simulation to validate and optimize design and HW architecture and various software algorithms explorations can all be achieved using conventional C++ development environments. The current system design methodology is for the system engineer to write high-level language (C, C++, Matlab etc.) programs to verify the concepts and algorithms at system-level. After the concepts and algorithms are validated, the high-level modelled designs are manually converted to the Hardware Description Languages (VHDL, Verilog-HDL) in order to implement the hardware. But this approach gives rise to a number of problems, such as errors arising from the manual conversion from C to HDL, a disconnection between the system level model and HDL model and conversion limitation as design sizes is get ever bigger and more complex. As a result of this, new C language based system design languages are starting to emerge as a new design methodology. Figure 7.1 shows the conventional system design in contrast to a SystemC based design methodology.



Figure 7.1: System Design Methodology: (a) Conventional Design Methodology, (b) SystmC Design Methodology (*Source: www.systemc.org)

The system design methodology using SystemC has many advantages over the conventional system design methodology including increased more productivity and reliability from the progressive refinement process and the use of a single language. In the design methodology using SystemC, the time consuming manual conversion process is no longer necessary because the high-level modelled code becomes a more reliable and high performance hardware model while hardware concepts and timing constructs can be added through the progressive refinement process. More productivity can be achieved by using a single design language, the high-level modelled SystemC code can result in smaller code that is easer to write as well as relatively faster simulation time, moreover

the testbench code for functional verification at high-level can be reused at any level or design stage[19,20].

7.1.1. CAD Environment for SystemC

As described above, SystemC is a C++ class library, which means any conventional C++ compiler can be a CAD development environment for SystemC. Any Unix, linux or PC based C++ compiler can be used, however, in this research, the PC based CAD environment (Microsoft Visual C++ Version 6.0) has used to compile the high-level modelled SystemC code because of its easy accessibility. Once SystemC code is compiled, the results are stored in various types of file. The most common file type for the results is a Value Change Dump (VCD) type and the GTKWave waveform viewer is used to validate VCD type of results. The figure below shows the Visual C++ and GTKWave waveform viewer.



Figure 7.2: The CAD Environment for SystemC; Visual C++ Version 6.0, GTKWave Waveform Viewer.

7.2 System-level Modeling of 3D-SoftChip

In this section, the high-level modelled single Standard-PE, Processing-Accelerator-PE, ICS_RISC and UnitChip will be introduced with output simulation waveform. The functionality of these components has been fully verified. For a more detailed description of the system-level modelling of 3D-SoftChip see Appendix B.

7.2.1. Standard-PE

The detailed architecture of the S-PE was introduced in Chapter 3. Based on the architecture, it has been high-level modelled using SystemC. Figure 7.3 shows the block diagram and SystemC file structure of the S-PE.



Figure 7.4 shows the output waveform of the S-PE execution results after ALU instructions between data from internal registers and embedded SRAM. The input signals (dIn, dLeft, dRight, dUp, dDown) have been selected by the input multiplexer. The ALU output signals can be seen in the dOut, and dOutadjPE signals. The functionality of the S-PE was confirmed by checking the output result.
VCD loaded successfully. (37) facilities found, (4985) regions found.			1	Zoom		Page F	etch Disc	Shift		Fram: 0 s To 4995	ec 5 ns	- M	4995 n Current T 160 ns	Time Is Time
Signals	Waves	17		17			100			21				225.1
SystemC clock = SystemC.reset =(SystemC.instICS[18:0] =t SystemC.dln[3:0] =t SystemC.dln[3:0] =t SystemC.dln[3:0] =t SystemC.dln[3:0] =t	300000150+ 30 \$5 30 \$4 30 \$3 50 \$3	150+160+160+	[92+ \$2+]	\$2+ \$2+ \$1+ \$	UTUF 0+ \$1+ \$	UTUTI 0+1\$1+1\$04	JT_JT_JT \$1+ \$0+ \$	5+ \$4+ \$	5+104+10	5+154+153	5+154+15	3+1\$2+165	5+1\$4+1	50+]si
SystemC.dDown[3:0] =1 SystemC.aluOut[3:0] =1 SystemC.sramData[3:0] =1 SystemC.doutBus[3:0] =1 SystemC.dOut[3:0] =1 SystemC.dOut[3:0] =1	<u>so</u> <u>so</u> <u>sx</u> <u>so</u>	\$1 \$3 \$2	SB SS S1 S1	5 53 52 58 53 52 5	1 (\$ 1 (\$ B	7 \$A 7 \$A	\$6 \$6	95 \$1 \$1 \$1	\$7 \$7 \$7	5A \$A \$A	\$6 \$6 \$6	\$3 \$3 \$1 \$1 \$1	\$5 \$3 \$3 \$3 \$3	19;

Figure 7.4: The Output Waveform of S-PE

7.2.2. Processing Accelerator-PE

The PA-PE architecture has been described in Chapter 3. The high-level modelling was executed from this description. Figure 7.5 shows the PA-PE block diagram and file structure for the SystemC modelling.





[36] facilities found. [5085] regions found.	_			Page Fetch	Disc Shift	From 0 sec To 4995 ns	Maximum Time 4995 ns Current Time 93 ns
Signals	Waves		104		201 4-	200	
Time	b	57 nz	134 05		201 ns	266 DS	335 1
SystemC.clock =			uuuu	0000	uuu		านนนนนน
SystemC.reset =	Tan Jan Jan Ja	an lon lon lon lon l	an lan lan lan l	an lan lan lan l	an lan lan la	the last last last last	las las las las las las
SystemC.insticS[180]=:	<u>\$0+1\$0+1\$0+1\$</u>	s0+120+120+120+120+120+1	20+120+120+120+1	\$0+130+150+192+1	\$2+192+192+193	1+120+121+120+121+120	+151+180+155+134+155+134+1
SystemC.din[3:0] =:	50 151						
SystemC.dLeft[3:0]=:	\$0 1\$1						
SystemC.dRight[3:0]=!	\$0 191						
SystemC.dUp[3:0] =:	50 151 19	12					
SystemC.dDown[3:0]=1	<u>s0 [s1</u>	The loss for the I		1.0 11.0			
SystemC, muxADut[3:0] =:	\$0 \$1	192 191 192 181	50 191	192 191			
SystemC.muxBOut[3:0] =!	\$0 \$1	\$2 \$1 \$2 \$1	\$0 [91	192 191			
SystemC s_aluOutI30i=!	50 51	153 154 190 192 1	60 68 51	92	\$1	1\$3 \$0	182 81
all	1					181 183 180	1 1 93
SystemC.sramData[3:0] =:					\$1 \$2		151 93
SystemC.sramData[3:0] =: SystemC.doutBus[3:0] =:	SX				the second se		
SystemC.sramData[3:0] =: SystemC.doutBus[3:0] =: SystemC.dOut[3:0] =:	\$0				\$1 \$2		\$1 \$3

Figure 7.6: The Output Waveform of PA-PE

The figure above shows the output waveform of the high-level modelled PA-PE. The selected input operand through the input multiplexer executes the ALU instruction (MAC, MAS, Shift, etc) and the results are then stored to the embedded SRAM. The output signal shows the operation executed as required.

7.2.3. ICS_RISC

The ICS_RISC and instruction set architecture was introduced in Chapter 4. The ICS_RISC can largely be classified into control and datapath units. The 32×32 -bit general purpose register, a program counter, a 16×32 -bit loop buffer, a status register, an instruction register, ALU, shifter, multiplier and 32-bit data input/output registers form the datapath architecture. The fetch, decoding and execution unit make up the control unit. Additionally, a bus control unit is used to control the 32-bit operand A, operand B, data write bus, input bus and output bus to avoid data collision. Figure 7.7 shows the top block diagram of the ICS_RISC and its SystemC file structures.



Figure 7.7: High-level modeling of ICS_RISC: (a) ICS_RISC block diagram, (b) file structure of ICS_RISC

The output waveform shows the results after execution of simple loop and ALU instructions. Figure 7.8 shows the pseudo code for the instructions. The circle in figure 7.9 which is written as a loop instruction indicates the internal general purpose register address. It increases as programmed and the other circle presents the output result of the ALU operations.

MOV	PO.	#0. 115	imple Loop Inst			
MOV	D1	#0, //-	simple coop mar			
MOV	P2	#1,				
MOV	R3	#2.				
MOV	R4	#4.				
MOV	RS	#5:				
MOV	R6.	#6				
MOV	RZ.	#7:				
MOV	R8.	RO:				
MOV	R9.	R1:				
MOV	R10,	R2:				
MOV	R11,	R3;				
MOV	R12,	R4;				
MOV	R13,	R5;				
MOV	R14,	R6;				
MOV	R15,	R7; //E	End of Loop Inst	-		
AND	R16,	R8, R9;	//ALU Inst			
OR	R17,	R10, R11;				/
XOR	R18,	R12, R13;				/
ADD	R19,	R14, R15;			/	-
	000	DIA DIE.	1/End		/	

Figure 7.8: The Psuedo Code for ICS_RISC



Figure 7.9: The Output Waveform of ICS_RISC

• "C:\SystemC\3D-SoftChip\TEST_ICS\ICS_RISC-in pro	gress/Debug/ICS_RIS 💶 🗙
ALUIS	
ALUR	
Press any key to continue	
4	* 4

Figure 7.10: The Instruction Index

Figure 7.10 shows the instruction index during ICS_RISC instruction execution for debugging purposes. The instruction index was perfectly matched with the instruction of the pseudo code.

7.2.4. UnitChip

The composition of the UnitCAP and UnitICS becomes the UnitChip. It can be largely divided into 4 kinds of sub-SystemC files, that is ICS_RISC, Memory, DMA and UnitCAP. As described in Chapter 5, the architecture and the pipelined operation mechanism can be identified in the high-level system simulation results.



Figure 7.11: High-level modeling of UnitChip: (a) UnitChip block diagram, (b) file structure of UnitChip

Figure 7.11 illustrates the UnitChip block diagram and SystemC file structure of the UnitChip. Each sub-SystemC block's functionality has been described before, the UnitChip is a simple combination (port-mapping) of these sub-blocks at the top module. The simple ALU instruction has been mapped in this high-level modelled UnitChip. The simulation result shows its functionality. In figure 7.12, the upper side circle indicates the ICS_RISC operation introduced before, and lower circle shows the PEs operations, which is the execution of simple ALU functions for the PEs with parallelism. The signal named as a PE1.dOut means the output signal from PE1. The functionality can be verified by checking these signals (from PE1~PE16) and is as expected.

/CD loaded successfully. 49] facilities found 5182] regions found.					Page	Fetch Di	sc Shift		From To: 4	D sec 995 ns	=	Maximum 4995 r Current	Time ns Time
Signals	Wayas		iii.										
Time		102 ns		136	DS		170 ns			204 1	15		2
SystemC clock								-					
SystemC.reset							1				_		
SystemC (Data(31.0)	1\$0006+ 1\$0007+	\$0408+ 1	0409+ \$0	40A+ 5040B	1+ \$040C+ \$	040D+ \$040	E+ \$040E	+ \$045	0+ 180471	+ \$0492	2+ \$04D	3+ [\$04F4-	+ 71000
SystemC.opAldx[4:0]	1\$05 1\$05	\$07 1	09 601	0 1901	\$	02	\$03		ISON	\$0B	SOD.	SOE	1
SystemC opBidx[4:0]	\$03		801	0					1908	SOA	SBC	SOD	
SystemC.rdAOEn			CS_RIS	SC Ope	rations	1-							1
SystemC.rdBOEn	-					-1	6				-		1
SystemC.wbldx[4:0]	1805 1906	\$07 18	08 \$0	a Isoa	\$0B \$	oc isop	302	SUF	1010	\$11	\$12	\$13	\$14
SystemC aluOut[31:0]	\$0000000	1	1	Discost 12		1			-	1\$047	+ 1\$049	2 1004D3-	+ 1304E
SystemC (Addr[31:0]	\$0+ 150000+ 1900	100+ 1\$000	1+ 1\$0000+	1\$0000+ 1\$	0000+ 190000	+ 1\$0000+1	\$0000+ I\$	0000+ 1\$	0000+ 12	00004 19	0000+15	0000+ 150	1000+
SystemC PE1_dout[3:0]	30						_	187	159	Ter	1\$3	150	
SystemC PE2_dOut[30]	50							182	Isa	ISA	1SE	Iso	
SystemC.PE3_dOut[3.0]	\$0			DE On	oration			Ten	The	Int	ISC	190	
Systemic PE4_dout[30]	10		_	PE OP	erations	1		158	180	195	159	190	
SystemC PE5_dOut[3:0]	50						-	151	197	ISE	1\$7	136	_
Systemic.PEb_dOut[3:0]	50							Int	[\$1	ISC.	ISB	185	
Systemic PE7_dout[3:0]								184	Ten	19.4	150	140	
Systemic PEB_dout[3:0]	10							147	197		ler	121	_
Systemic PE9_dout[3:0]	50						_	19/	Inc	162	155	154	_
ystemc PEIU_dout[30]	50						-	lette	189	154	lén	Tre	_
ystemo.PETT_dout[3:0]	20							170	lep	104	100	192	
rstemC/PE12_dout[3.0]	20							127	ler	100	lan	199	_
value PE14 dOut301	10						_	60	len	150	ToB	191	_
visionC PE15 dOut201	30		_				-	ISE .	les	len	Dal	ler	
olateuro Le 12 gondarol	30							102	Lea	190	112	196	

Figure 7.12: The Output Waveform of UnitChip

7.3 Conclusions

In this chapter, the overview of SystemC and its CAD tool develop environment has been introduced. The high-level system modelling and functional verification of the 3D-SoftChip using SystemC has been described and some simulation results provided. The waveforms show the correct functionality for each of the sub-blocks and for the top module of the UnitChip.

Chapter 8 Application Mapping for 3D-SoftChip

The MPEG4 Full Search Block Matching Motion Estimation Algorithm (FBMA) has been applied to the high-level system modeled 3D-SoftChip to verify its functionality and demonstrate its architectural superiority. The hand-crafted assembler code for implementation of the algorithm becomes the input stimulus of the system-level modeled 3D-SoftChip. The performance will be analyzed in compraison with a conventional DSP processor, Application Specific ICs (ASICs) and MorphoSys.

8.1 Full Search Block Matching Algorithm (FBMA)

Motion estimation (ME) is introduced to exploit the temporal redundancy of video sequences and is an indispensable part of video compression standards such as the ISO/IEC, MPEG-1, MPEG-2, MPEG-4 and the CCITT, H.261/ITU-T, H.263 etc. Since ME is computationally the most demanding portion of the video encoder, it can take up to 80% of total computation time and it can be a major limiting factor for the performance. Among the many different ME algorithms, FBMA is one of the most widely used in hardware, despite its high computational cost because it has the optimal performance and lowest control overhead. The block matching motion estimation algorithm compares a specific sized block of pixels in the current frame with a range of equally sized pixel blocks in the previous frame to find the best match (minimum difference) between two of the blocks. The position of the best matched block can then be encoded as a motion

- STEP 1 LOAD REF. BLOCK DATA INTO PE ARRAY SRAM: The first operation is to load reference block data (I_k(m,n)) into embedded SRAM in each PE in the array.
- STEP 2 EACH PE MOVES THIS DATA TO INTERNAL REGISTER: Each PE moves the reference data from the embedded SRAM into an internal register so it is available to be used for calculation of SAD values for the entire search window.
- STEP 3 LOAD FIRST SEARCH POSITION BLOCK DATA INTO PE ARRAY SRAM: The block data for the first search position $(I_{k-1}(m+dx, n+dy))$ is then loaded into the embedded SRAM in each PE in the array ready for calculation of the SAD value between the reference block and this first search position.
- STEP 4 EACH PE EXECUTES SUBTRACTION AND ABSOLUTE VALUE COMPUTATION: In this step, each PE carries out a subtraction operation between the reference block data and the current search position in SRAM, the absolute value of this resulting difference is stored as the absolute difference value for that block position.
- STEP 5 PARTIAL SUMMATION (1): In this step every odd columned PE performs a partial sum operation of its absolute difference value with the value from the PE to its immediate right in the array, the result is stored as a double-word value across both PEs.
- STEP 6 PARTIAL SUMMATION (2): In this step the two partial sums computed in the previous step are summed in the same way, every odd columned PE pair sums its result with the result from the PE pair to its right, this result is stored as a quad-word value across all four PEs in each row.
- STEP 7 PARTIAL SUMMATION (3): In this step the column wise operation carried out in step 5 is repeated row wise to accumulate another set of partial sums,

in this case, however, the second row of PEs accumulated its result with the result from the row above, while the third row of PEs accumulates its result with the result from the row below.

- STEP 8 PARTIAL SUMMATION (4): In this final partial sum accumulation, the second row of PEs sums its result with the result from the third row, producing the total SAD value for that search position.
- STEP 9 WRITE BACK RESULT DATA TO THE ICS_RISC: Finally the resultant SAD value calculated in STEP 8 is written back to the internal register in the ICS_RISC for comparison with the previous minimum and updating of the motion vector if applicable.
- STEP 10 REPEAT STEPS 4 TO 9: The next search position data block can be loaded into the SRAM in the PE array while the SAD calculation is being carried out for the current search position so once the result had been written back the calculation of the SAD for the next search position can be begun immediately.

8.3 Performance Analysis

Figure 8.3 shows the performance comparison of the 3D-SoftChip with a DSP processor, several ASICs and MorphoSys for matching on 8×8 reference block against its search area of 8 pixels displacement. There are 81 candidate blocks (27 iterations) in each search area [33]. In the 3D-SoftChip, as described above, the number of processing cycles for one candidate block is just 7 clock cycles (each UnitChip computes one quarter block, so with 4 UnitChips one complete block is computed every 7 cycles), so the total number of processing cycles for the 3D-SoftChip becomes 567 (81 iterations of 7 cycles each).

The number of clock cycles required is very close to that reported for MorphoSys, with just 4 UnitChips, this, however, can readily be improved simply by increasing the number of UnitChips on a scaled up 3D-SoftChip. A 4×4 UnitChip array, for example, would have an effective throughput of one block every 142 cycles. In addition to this, considering the characteristics of the 3D system, there are other significant advantages.

Data dependency is largely eliminated so there after the initial set-up there is a 100% PE utilisation. The reference and candidate block data can be moved into the embedded SRAM in the PE concurrently with array execution, so the PEs can operate continuously. Also low power consumption can be achieved through a minimisation of the number of data accesses, because most of data manipulation can be executed within the PE array. Most importantly, however, because all memory is directly accessible within the 3D-SoftChip via the IBIA there are effectively zero external data reads and thus power consumption will be greatly improved over all the other approaches.



Figure 8.3: Performance comparison for Motion Estimation

When comparing with the performance of the DSP processor and dedicated ASICs, the performance of the suggested 4×4 UnitChip 3D-SoftChip has remarkable advances with a theoretical capability of more than 3.8 times the performance. Given its wide applicability/adaptability to any number of other applications, the performance achieved compared to these dedicated processors is a potentially enormous advancement. This clearly demonstrates the architectural superiority of the suggested novel 3D-SoftChip.

8.4 Conclusions

In this chapter, the mapping of the implemented MPEG4 full search block matching algorithm has been applied to the system-level modelled 3D-SoftChip in order to demonstrate its architectural superiority. According to the described results, the proposed 3D-SoftChip architecture has the potential for a more than 3.8 times performance improvement over conventional systems. The suggested 3D-ACSoC is clearly a highly suitable system for the coming giga-scaled integrated computing age.

Chapter 9 Conclusions

In this chapter, the contribution of this thesis will be summarised and future research work will be suggested.

9.1 Contributions

In this thesis, a novel 3D vertically integrated adaptive system-on-chip architecture as a next generation computing system along with its functional verification and the mapping of an MPEG4 motion estimation algorithm has been presented. The suggested architecture has a number of advantages compared with conventional current generation reconfigurable/adaptive computing systems, such as wide applicability, various and powerful computation methods, adaptive word-length configuration and benefits from the architecture including 3D interconnect performance, reliability and a reduction in the size of the chips (and thus the cost), as described before. As outlined in chapter 5.3, the size of total chip as described is relatively small at around 1.1 mm² for an array of 2×2 UnitChips or 2.2 mm² for an array of 4×4 UnitChips. This is based on a 4-bit word-length for the PEs so there is also ready potential to extend to a wider word-length (8-bit word-length) and more integration of the PEs to maximise the computational throughput and benefits from large integration. Moreover, the ICS_RISC can also be readily extended on the upper chip layer by adopting advanced computation algorithms and dedicated instructions for specific applications to allow more efficient controllability and

performance over the current relatively simple ICS_RISC design. As minimum feature sizes continue to decrease in more advanced chip fabrication processes the inherent scalability of the UnitChip design means that the array size can simply be increased to within the constraints of the maximum die size to realise ever more power adaptive computing systems.

The performance of the execution of the MPEG4 full search block matching motion estimation algorithm has been shown to be more 3.8 times improved over current generation processors. Due to these significant performance, power and cost advantages it can be shown that the suggested 3D-ACSoC is one of the most suitable architectures for the next generation of computing system.

Moreover, the suggested advanced HW/SW co-design and verification methodology can accelerate the reliability and significantly reduce the design time, especially the time and effort required for verification. This thesis indicates a highly promising research direction for future adaptive computing systems and an advanced and efficient HW/SW development methodology for ever more complicated SoCs.

9.2 Future Work

As introduced in the suggested design methodology, the high-level modelling and functional verification has been carried out, the next task is the architectural explorations to obtain an optimized HW specification. The method to explore various architecture options is through parameterized memory, data frame buffer and DMA controller modelling using SystemC, followed by simulation with various HW configurations so as to find the best HW specification. The use of the parameterized modelling method makes the architecture exploration considerably easier, the parameter values can simply be changed in the SystemC code. Figure 9.1 shows the SystemC modelling of the parameterized memory. Once the optimum HW specification is decided, the rest of the procedure can be executed with any conventional hardware design method, such as full and semi-custom design and the SW design should be concurrently performed so that the novel concept of an adaptive system-on-chip computing system can be realised.

```
//Paramteterized RAM
#ifdef RAMT_H
#define RAMT_H
#include "systemc.h"
template <class T, int size = 100>
SC_MODULE(ram) {
        sc_in<bool> clock;
       sc_in<bool> nRW; //Read/Write
sc_in<int> addr; //Address
sc_inout<T> data; //Parameterized Word-length
void ram_proc();
SC_HAS_PROCESS(ram);
ram(sc_module_name name_, bool debug_ = false);
       sc_module(name_), debug(debug_)
1
SC_THREAD(ram_proc);
sensitive << clock.pos();
buffer = new T[size];
if (debug) [
       cout << "Running constructor of" << name() << endl;
       cout << "Number of location is" << size << endl;
3
private :
        T* buffer;
       const bool debug;
Ŀ
template <class T, int_size>
void ram<T, size>::ram_proc()
while(true) {
       wait();
       if (nRW) {
                data = buffer[addr];
        }else {
                buffer[addr] = data;
        }
#endif
```



Bibliography

[1] E. Mirsky and A. DeHon, "MATRIX : A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources," Proc. IEEE Symp. FPGAs for Custom Computing Machines, pp.157-166, April 1996

[2] S.C. Goldstine, H. Schmit, B.Mihai, S. Cadambi, M.Matt, R.R.Taylor, "PipeRench : A Reconfigurable Architecture and Compiler," IEEE Computer, pp.70-77, April 2000

[3] S. Hartej, L. Ming-hua, L. Guangming, J.K. Fadi, B. Nadar, M.C.F Eliseu, "MorphoSys: An Integrated reconfigurable system for data-parallel and computation-intensive applications". IEEE transactions on computers, Vol49, No.5, pp.465-481, May 2000

[4] E. Waingold et al. "Bring it all to software : RAW Machines", Computer, Vol30, Issue9, pp.86-93, Sept.1997

[5] T. Miyamori , K. Olukotun, "REMARC : Reconfigurable Multimedia Array Coprocessor", Proc. ACM/SIGDA FPGA98, Monterey, Feb.1998

[6] C. Ebeling et al., "Architecture design of reconfigurable pipelined datapaths", Advanced Research in VLSI, 1999. Proceedings. 20th Anniversary Conference on, pp.23-40, 21-24. Mar.1999

[7] Goldstein, S.C. et al, "PipeRench: a reconfigurable architecture and compiler", Computer, Vol.33, Issue4, pp.70-77, April 2000

[8] D.Chen, J. Rabaey, "PADDI: Programmable arithmetic devices for digital signal processing", VLSI Signal Processing IV, IEEE Press, 1990

[9] Elixent Ltd, "The Reconfigurable Algorithm Processor", http://www.elixent.com/products/white_papers.htm

[10] Triscend Corp., "Triscend A7S Configurable System-on-Chip Platform", http://www.triscend.com/

[11] Motorola Inc., "MRC6011: Reconfigurable compute Fabric(RCF) Device", http://www.motorola.com/semiconductors/

[12] Nick Tredennick, Brion Shimamoto, "Special Report: Do-it-all devices", IEEE Spectrum, pp.37-40, Dec. 2003,

[13] L. Guangming, "Modeling, Implementation and Scalability of the MorphoSys Dynamically Reconfigurable Computing Architecture," PhD thesis, Univ. of California, Irvine, 2000

[14] S. Eshraghian, S. Lachowicz, K. Eshraghian, "3-D Vertically Integrated Configurable Soft-Chip with Terabit Computational Bandwidth for Image and Data Processing", Proc. MIXDES'2003, Lodz, Poland, June 26-28, 2003

[15] A. Rassau, G. Alagoda, A. Ehrdardt, S. Lachowicz, K. Eshraghian, "Design Methodology for a 3D-SoftChip Video Processing Architecture", 6th World Multiconference on Systemics, Cyvernetics and Informatics (SCI2002), Orlando, Florida, U.S.A, pp.324-329, July 14-17, 2002

[16] QuickSilver Technology Inc., "Adapt2400 ACM architecture Overview", http://www.quicksilvertech.com/pdfs/Adapt2400_Whitepaper_0404.pdf

[17] picoChip Design Limited, "PC102 Product Brief", http://www.picochip.com

[18] S. Eshraghian, "Implementation of Arithmetic Primitives Using Truly Deep Submicron Technology (TDST)", Ms thesis, Edith Cowan University, 2004

[19] Open SystemC Initiative, "The Functional Specification for SystemC 2.0", http://www.systemc.org/

[20] Open SystemC Initiative, "SystemC 2.0.1 Language Reference Manual Rev 1.0", http://www.systemc.org/

[21] International Technology Roadmap for Semiconductors (ITRS), "International Technology Roadmap for Semiconductors 2003 Edition", http://public.itrs.net/

[22] AMDREL Consortium, "Existing Functional Level Reconfigurable Implementation Platforms", http://vlsi.ee.duth.gr/amdrel/deliverables.html

[23] IZM, "3D System Integration", http://www.pb.izm.fhg.de/izm/015_Programms/010_R/

[24] Joyner J.W., Zarkesh-Ha P.J, Meindl J.D, "Global Interconnect Design in a Three-Dimensional System-on-a-Chip", IEEE Transactions on VLSI systems, Vol 12, Issue4, pp.367-372, April 2004

[25] J.W. Joyner, et al., "Impact of three-dimensional architectures on interconnects in gigascale integration", IEEE Trans. VLSI Syst. Vol. 9, pp.922-928, Dec. 2001

[26] Kaustv. Banerjee, et al, "3-D ICs: A Novel Chip Design for Improving Deep-Submicrometer Interconnection", Proceedings IEEE Special Issues on Interconnections, Vol. 89, No 5, pp602-633, May 2001

[27] Texas Instruments, "TMS320C6000 Assembly Benchmarks", http://www.ti.com/sc/docs/products/dsp/c6000/benchmarks/67x.htm

[28] K M Yang, M-T Sun and L.Wu, "A Family of VLSI Design for Motion Compensation Block Matching Algorithm", IEEE Trans. on Circuits and Systems, Vol. 36, No 10, pp1317-25, October, 1989

[29] C. Hsieh and T.Kin, "VLSI Architecture for Block-Matching Motion Estimation Algorithm", IEEE Trans. on Circuits and Systems for Video Technology, Vol.2, pp169-175, June 1992

[30] Yeong-don Bae, "Basic Microprocessor Design", http://www.donny.co.kr

[31] Yap Zi He, "Building A RISC Microcontroller in an FPGA, "http://www.opencores.org/projects/riscmcu/

[32] Michael Shyu, Guang-Ming Wu, Yu-Dong Chang and Yao-Wen Chang, "Generic Universal Switch Blocks", IEEE Trans. on Computer, Vol. 49, Issue 4, pp348-359, April 2000

[33] Hartej Singh, "Reconfigurable Architectures for Multimedia and Data-Parallel Application Domains", PhD Thesis, University of California Irvine, 2000

[34] R.Gao, D.Xu and J.P.Bently, "Reconfigurable Hardware Implementation of an Improved Parallel Architecture for MPEG-4 Motion Estimation in Mobile Applications", IEEE Trans. on Consumer Electronics, Vol. 49, No4, pp1383-1390, November 2003

[35] J.Rabay et al, "Reconfigurable Computing: The Solution to Low Power Programmable DSP", Proc. ICASSP 97, Munich, Germany, April 1997

[36] Stylianos Perissakis et al, "Embedded DRAM for a Reconfigurable Array", Proc. of VLS199, June 1999

[37] Chameleon Systems Inc. "CS2000TM Reconfigurable Communications Processor", CS2000TM Advanced Product Information, http://www.chameleonsystems.com

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix A = IC8_RISC ISA Version 1.0

ICS_RISC Instruction Set Architecture Version 1.0

1

	21	20	70	78	77	76	75	24	73 77	וי	20 10	18 17	16	15 14 13	- 12 - 11	10 9	876	5 4	3 2 1 0
		50	24	-0				++ T			1	10 11	10						· · · · · · · · · · · · · · · · · · ·
Immedi.(Iword)	0	0	0	0	0	0	0		Opcode	:		Rđ		·	Ir	nmedia	te(4,8,16-bit)	
Immedi.(2word)	0	0	0	0	0	0	I		Opcode	:		Rđ				U	nused		
							_				In	nmediate ((32-bit)	÷.,		• •			
Register	0	0	0	0	0	I	0		Opcode	•		Rd		Rs2			Rsl	l t	Jnused
LB Addressing	0	0	0	0	0	ļ	1		Opcode	e :		Rd		Rs2			Rsl	RWEn	Unused
Shift / Rotate	0	0	1	0	1	0	0		Shift	x		Rd		ShiftA	mt		Rsl	<u> </u>	Jnused
Load	0	0	1	0	1	0	1	0	x			Rd					Rb	ι ι	Jnused
Store	0	0	I	0	1	1	0	1	x			Rd		x			Rb	ι	Jnused
Branch	0	0	1	0	I	1	1		Cond	x	· ·				Offset				
PE Control	0	1	0	1	0	0	0		PE Op	x	Opmode	Config		PE Sel			Unuse	d	
DMA Control	1		DM/	\	DFB	A	mount	of D:	ata to	Star	rt address of	DFB	SRAM	Start addre	ess of	Mem	Start addi	ess of Pr	ogram/Data
			Ор		Sđ		Tra	nsfer			(Sou/Dst)	I	Reg.Sci	SRAM/ICS R	leg(S/D)	Set	Mer	nory (Sou	/Dst.)
Multiply	0	1	1	1	1	1	1		x			Rd		Rs2			Rsl	ι	Jnused
Dedicated						• .]	Not yet de	ecided						
Instructions				. ••					n de la composición de										

A DISSERTATION FOR THE DEGREE OF MASTER OF ENGINEERING SCIENCE

A Novel 3D Vertically Integrated Adaptive Computing System Appendix A = ICS_RISC ISA Version 4.0

	Opc	odes		Mnemonics	Description (Immediate)	Description (Register)
0	0	0	0	MOVA	Rd = Immediate	Rd = Rs1
0	0	0	1	MOVB	Rd = Immediate	Rd = Rs2
0	0	1	0	AND	Rd = Rd & Immediate	Rd = Rs1 & Rs2
0	0	1	1	OR	Rd = Rd Immediate	Rd = Rs1 Rs2
:0	1	0	0	XOR	Rd = Rd ^ Immediate	Rd = Rsi ^ Rs2
0	1.	ò	1	NOT	Rd = ~ Immediate	Rd = ~RsI
· 0 ·	1	1	.0	ADD	Rd = RsI + Immediate	Rd = Rs1 + Rs2
0	1	Γ.	1	SUB	Rd = Rs1 – Immediate	Rd = Rs1 - Rs2
1	0	0	0	СМР	Compare Rs1 and Immediate	Compare Rs1 and Rs2
- 1	0	0	1	MSR	Status Register = Immediate	Status Register = Rs1
1	0	1	0	MRS	N/A	Rs1 = Status Register

	Shift		Mnemonics	Description
0	0	0	LSL	Shift Left
0	0	1	LSR	Shift Right
0	L	0	ASR	Arithmetic Shift Right
I	0	0	ROT	Rotate

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix A = ICS_RISC ISA Version 1.0

	Cond		Mnemonics	Description
0	0	0	EQ	Equal
0	0	1	NE	Not Equal
0	1	0	AL	Always (Unconditional)

PE (Operat	ions	Mnemonics	Description	
0	0	0	PECONF	Configuration of each PEs (4,8,16,32 bits)	
0	0	1	PESEL	To select certain PE (PE0 ~ PE15)	
0	1	0	PEMODE	To select PE operation modes (Horizontal/Vertical/Circular modes)	· · · · · ·
0	1	1	PEVEXE	To execute specific program to each PEs in the same vertical line	
· 1	0	0	PEHEXE	To execute specific program to each PEs in the same horizontal line	• •
1	0	1	PECEXE	To execute specific program to each PEs in the same circular line	• •

	DMA		Mnemonics	Description
O	peratio	ns		
0	0	0	LDPEPRG	Load maximum16 program data from Program memory to Embedded SRAM in PEs
0	0	1	LDDFB	Load large amount of processing data for PEs from Memory to Data Frame Buffer
0	1	0	LDPEDATA	Load large amount of processing data for PEs from DFB to Embedded SRAM in PE
0	1	1	WBREG	Write back processed data in Embedded SRAM to the Registers in the ICS_RISC
1	0	0	WBDFB	Write back processed data in Embedded SRAM to DFB
1	0	1	WBMEM	Write back processed data in DFB to Data Memory

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix A = ICS_RISC IS V Version 1.0 Instruction descriptions 1.1 Immediate addressing : Short immediate values : 4,8,16 bit (1 instruction word), Long immediate value : 32 bits (2 instruction words) Rd = Rd op Immediate (4,8,16,32 bit) 2 1 16 15 14 13 12 3 23 21 20 19 18 17 11 26 25 24 22 31 30 29 28 27 Rd 4, 8, 16 bit Constant 0 1 Instruction word 0 0 0 0 0 Opcode 0 9 8 7 6 5 4 3 2 1 25 24 23 22 21 20 19 18 17 16 15 14 13 12 -11 10 0 30 29 28 27 26 31 Rd Unused Opcode 2 Instruction words 0 0 0 0 0 0 . 32 bit Constants Description: The processed data from PEs and immediate data from regFile or data memory can be manipulated in the ICS_RISC so it can process 4,8,16,32bit data. **Register addressing :** Rd = Rs1 op Rs2Description: Rs1 and Rs2 indicates the address of internal regFile(32 sets of 32bit data(32 × 32bit)). The opcode identifies the operations and the manipulated data between Rs1 and Rs2 is stored in the register which indicated by Rd. LB Addressing: Rd = Rs1 op Rs2Description: When the LB Addressing becomes active, the sources of addresses become a Loop Buffer. It has 16 depths of looping capacity. Shift / Rotate : Rd = Rs1 Shift by Amount Description: According to the shiftCtl and shiftAmt, the shifter can shift the input operands. Load : Rd = Mem [Rb]

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix A = 4CS_RISC ISA Version 1.0

Description: Rd in the regFile can load the data from data memory address which indicates by Rb.

Store :

Mem [Rb] = Rd

Description: Data in the regFile can store to the data memory address which indicates by Rb

Branch :

If (Cond) PC = PC + Offset

Description: According to the Cond signals, the Program Counter value can increase as much as offset value:

Multiply :

Rd = Rs1 * Rs2

Description: The operands can multiplied and stored in the Rd.

PE Control :

PECONF, PESEL, PEMODE (Horizontal/Vertical/Circular modes), PEEXE

DMA Control :

LDPEPRG: Load maximum16 program data from Program memory to Instruction Decoder in PEs LDDFB: Load large amount of processing data for PEs from Data Memory to Data Frame Buffer LDPEDATA: Load large amount of processing data for PEs from DFB to Embedded SRAM in PE WBREG: Write back processed data in Embedded SRAM in PE to the registers in the ICS_RISC WBDFB: Write back processed data in Embedded SRAM in PE to DFB

WBMEM: Write back processed data in DiFB to Data Memory

Dedicated Instructions

Not yet decided

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix A = ICS/RISC ISA Version 1.0



94

3D-SoftChip A Novel 3D Vertically Integrated Adaptive Computing System Appendix B-High-level Modeling of 3D-SoftChip Using SystemC

High-level Modeling of 3D-SoftChip using SystemC

1 Configurable Array Processor (CAP) Chip

1.1 Processing Element : Standard-PE



Figure 1.1: Standard-PE architecture

1.2 System Components

- MUX A, MUX B: input operand selection
- Instruction Decoder: 4 sets of ID, each ID have 4 sets of 19-bit registers for S-PE instruction decoding
- ALU: 4-bit ALU with bit-serial multiplier, adder, subtractor, comparator
- Registers : 4 sets of registers
- DourReg : data out register to send data for adjacent PEs(Up/ Down/Left/ Right)
- Embedded SRAM : embedded SRAM (word-length: 4-bit, address : 0~15)

3D-SoftChip A Novel 3D Vertically Integrated Adaptive Computing System Appendix B-High-level Modeling of 3D-SoftChip Using SystemC

1.3 S-PE functions

Table 1.1: S-I	PE functions
Function	Mnemonics
A and B	AND
A or B	OR
not A	NOT
A xor B	XOR
A + B	ADD
A – B	SUB
A × B	SPMUL
A comp B	COMP

1.4 S-PE Instruction Format

18	17	16	15	12	11	10	9	8	6	5	3	2	0
WS_en/	WR_en/	SRAM		SRAM Selection	Re	egister	DoutR	SPE	_OP	MU	хв	MU	KA
RS_en	RR_en	en			Sel	lection	Cil						



1.5 S-PE Block Diagram (In/Output Pin Description)



Figure 1.3: S-PE block diagram (Input/Output Pin Description)

A Novel 3D Vertically Integrated Adaptive Computing System Appendix B-High-level Modeling of 3D-SoftChip Using SystemC

1.6 Data-path Architecture of S-PE



Figure 1.4: Data-path architecture of S-PE

1.7 S-PE Operation Flow



Figure 1.5: S-PE operation flow

A Novel 3D Vertically Integrated Adaptive Computing System Appendix B-High-level Modeling of 3D-SoftChip Using SystemC

1.8 SystemC File Structure



Figure 1.6: SystemC file structure of S-PE

1.9 SystemC Codes for S-PE

See Appendix C

1.10 Output Waveform

VCD loaded successfully. (37) facilities found. Toom Page Fetch Disc Maximum Time 4995 ns Signals To 4995 ns To 4995 ns Current Time 180 ns Signals Signals To 4995 ns Current Time 180 ns Signals Signals Waves 67 ns 134 ns 201 ns 268 ns 335 SystemC.clock = SystemC.dnst(S[180] =: SystemC.dlat[30] =: 50 185 50 185 50 185 50 185 SystemC.dlat[30] =: SystemC.dlat[30] =: SystemC.dlu0(30] =: SystemC.doubus[30]	T GTKWave - C:\SystemC\3D File Edit Search Time	-SoftChi Markets	ip\SPE\s	pe_wave.vo	đ						-		-		1	
Signals Waves 268 ns 335 SystemC.clock = SystemC.inst(CS)18:01 == SystemC.dln[3:01 == SystemC.dl	VCD loaded successfully. (37) facilities found. (4985) regions found.				1	Zoom	< ▶ <- URDO] →	Page Fe	tch Disc	Shift		From: 0 s To: 4995	ec i ns	- Ma	4995 ns urrent Tin 160 ns	me ne
Time 67 ns 134 ns 201 ns 268 ns 335 SystemC.clock = SystemC.inst(CS[180] =1 SystemC.dln[30] =1 Sy	Signals	Waves									8904444 (1111) (177-1-1-					
SystemC.clock = SystemC.instiCS[18:0] =: SystemC.instiCS[18:0] =: S00000[\$0+ \$0+ \$0+ \$2+ \$2+ \$2+ \$2+ \$1+ \$0+ \$1+ \$0+ \$1+ \$0+ \$1+ \$0+ \$5+ \$4+ \$5	Time	-		67 ns		13	l ns		201	ns		26	58 ns			335
SystemC. InstiCS [18:0] =: \$00000 [\$0+ \$0+ \$0+ \$2+ \$2+ \$2+ \$2+ \$2+ \$1+ \$0+ \$1+ \$0+ \$1+ \$0+ \$5+ \$4+ \$5+ \$5+ \$4+ \$	SystemC.clock = SystemC.reset =I		บาบ		uur	nnn	บบ	un		JU	uu		uu	uuu		
SystemC. dln[3:0] =: 50 55 SystemC. dLaf[3:0] =: 50 54 SystemC. dLip[3:0] =: 50 153 SystemC. dLip[3:0] =: 50 153 SystemC. dLip[3:0] =: 50 151 SystemC. dlup[3:0] =: 50 151 SystemC. aluOut[3:0] =: 50 151 SystemC. aluOut[3:0] =: 50 151 SystemC. doutBu[3:0] =: 50 151 153 SystemC. dOutadjPE[3:0] =: 50 151 153 153 SystemC. dOutadjPE[3:0] =: 50 151 153 153 151 153 SystemC. dOutadjPE[3:0] =: 50 151 153 153 154 153	SystemC.instICS[18:0] =!	\$0000	00 \$0+1	\$0+ \$0+ \$0)+ \$2+ \$2+	\$2+ \$2+ \$1+ \$	0+ \$1+ \$	0+\$1+\$0+	\$1+ \$0+ \$	5+ \$4+ \$	5+ \$4+ \$	5+ \$4+ \$5	5+ \$4+ 63	+ \$2+ \$5	+ \$4+ \$1	1+ \$[
SystemC.dLeft[30] =: \$0 \$4 SystemC.dDig10] =: \$0 \$3 SystemC.dDip[30] =: \$0 \$2 SystemC.dlown[30] =: \$0 \$1 \$2 \$5 \$1 \$5 \$2 SystemC.aluOut[30] =: \$0 \$1 \$2 \$1 \$2 \$2 \$2 SystemC.sramData[3.0] =: \$0 \$1 \$2 \$2 \$2 \$2 \$2 SystemC.doutBu[30] =: \$0 \$1 \$2 \$2 \$2 \$2 \$2 SystemC.doutBu[30] =: \$2 \$2 \$2 \$2 \$2 \$2 \$2 SystemC.doutBu[30] =: \$2 </td <td>SystemC.dln[3:0]=1</td> <td>\$0</td> <td>\$5</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>_</td> <td></td> <td></td> <td></td> <td>0000</td> <td></td> <td></td> <td></td>	SystemC.dln[3:0]=1	\$0	\$5						_				0000			
SystemC. dRight[3:0] =: 50 §3 SystemC. dUp[3:0] =: 50 §2 SystemC. aluOut[3:0] =: 50 §1 §2 SystemC. aluOut[3:0] =: 50 §1 §2 §3 SystemC. aluOut[3:0] =: 50 §1 §2 §3 §5 §1 §2 SystemC. aluOut[3:0] =: 50 §1 §2 §1 §2 §3 §5 §2 SystemC. doutBis[3:0] =: 50 §1 §2 §2 §3 §3 §5 §3 §3 SystemC. doutBis[3:0] =: 50 §1 §3 §2 §B §1 §7 §4 §6 §1 §3 SystemC. dOutadjPE[3:0] =: 50 §1 §3 §2 §B §1 §7 §4 §6 §1 §3 SystemC. dOutadjPE[3:0] =: 50 §1 §3 §2 §8 §1 §7 §4 §6 §1 §3 §5 §3 §3 §4 §6 §6 §1 §3 §3 §5 §3 §5 §5	SystemC.dLeft[3:0]=!	\$0	\$4												_	
SystemC. dUp[3:0] =: \$0 \$2 SystemC. dDuwn[3:0] =: \$0 \$1 SystemC. aluOut[3:0] =: \$0 \$1 \$2 \$1 SystemC. sramData[3:0] =: \$0 \$1 \$2 \$1 \$2 \$2 SystemC. dout[3:0] =: \$2 \$1 \$2	SystemC.dRight[3:0] =:	\$0	\$3													_
SystemC. dDown[3:0]=: 50 [\$1 SystemC. aluOut[3:0]=: 50 [\$1 [\$2 [\$1 [\$7 [\$4 [\$5 [\$3 [\$5 [\$5 [\$3 [\$5 [\$5 [\$3 [\$5 [\$5 [\$3 [\$5 [\$5 [\$3 [\$5 [\$5 [\$3 [\$5 [\$5 [\$3 [\$5 [\$5 [\$3 [\$5 [\$5 [\$3 [\$5 [\$5 [\$3 [\$5 [\$5 [\$3 [\$5 [\$5 [\$3 [\$5 [\$3 [\$5 [\$3 [\$5 [\$3 [\$5 [\$3 [\$5 [\$3 [\$5 [\$3 [\$5 [\$3 [\$5 [\$3 [\$5 [\$3 [\$5 [\$3 [\$5 [\$3 [\$5 [\$3 [\$5 [\$3 [\$5 [\$3 [\$5 [\$3 [\$5 [\$3 [\$3 [\$5 [\$3 [\$3 [\$5 [\$3 [\$3 [\$3 [\$5 [\$3 [\$3 [\$3 [\$5 [\$3 [\$3 [\$3 [\$3 [\$3 [\$3 [\$3 [\$3 [\$3 [\$3 [\$3 [\$3 [\$3 [\$3<	SystemC.dUp[3:0] =:	\$0	\$2													_
SystemC aluOut[3:0]=! \$0 \$1 \$3 \$2 \$1 \$7 \$4 \$5 \$3 \$5 \$5 SystemC sramData[3:0]=! \$2 \$1 \$2 \$2 \$2 \$2 \$2 \$3 \$5 \$5 SystemC doutBus[3:0]=! \$2 \$1 \$2 \$2 \$2 \$2 \$3 \$5 \$1 \$3 SystemC doutBus[3:0]=! \$2 \$1 \$2 \$2 \$2 \$2 \$3 \$5 \$1 \$3 SystemC doutGut[3:0]=! \$0 \$1 \$53 \$2 \$2 \$3 \$5 \$1 \$3 SystemC doutadyPE[3:0]=! \$0 \$1 \$53 \$2 \$2 \$3 \$5 \$1 \$3 \$0 \$1 \$1 \$3 \$2 \$2 \$3 \$5 \$1 \$3 \$1 \$2 \$2 \$2 \$3 \$2 \$5 \$1 \$3 \$2 \$5 \$1 \$3 \$2 \$5 \$1 \$3 \$5 \$5 \$1 \$3 \$5 \$5	SystemC.dDown[3:0] =:	\$0	\$1												-	-
SystemC. sramData[3.0] =! Image: System C. doutBus[3:0] =! Image:	SystemC.aluOut[3:0] =1	\$0		\$1 \$3 \$2	\$B \$5		1 19	7 \$A	1\$6	185				\$3	\$5	\$:
SystemC. doutBus[3:0] =! SystemC. dout[3:0] =!	SystemC.sramData[3:0] =!					3	1 5	7 SA	55	\$1	\$7	SA	\$6	\$3	\$3	
SystemC.dOut[3:0]=! 50 IS1 IS3 IS2 ISB IS1 IS7 ISA IS6 IS1 IS3 SystemC.dOutadjPE[3:0]=! 50 ISA ISA IS6 IS1 IS3 IS6 IS1 IS3 IS1	SystemC.doutBus[3:0] =1	<u>\$X</u>			\$1	\$3 \$2 \$B	_			\$1	\$7	I\$A	\$6	\$1	\$3	
SystemC.dOutadjPE[3:0]=! <u>\$0</u> [\$A [\$6	SystemC.dOut[3:0]=!	\$0			\$1	\$3 \$2 \$	B			\$1	\$7	SA	\$6	\$1	\$3	_
	SystemC.dOutadjPE[3:0] =:	\$0										ISA	\$6			_
		1														
		-														E

Figure 1.7: Top level simulation result of S-PE

1.11 Processing Element : Processing Accelerator-PE



Figure 1.8: Processing Accelerator-PE architecture

- 1.12 System Components
 - . MUX A, MUX B : input operand selection
 - . Instruction Decoder : 4 sets of ID, each ID has 4 sets of 19-bit registers for PA-PE instruction decoding
 - Multiplier : a signed 4-bit scalable parallel/parallel multiplier ٠
 - . Accumulator/Subtractor : to enable MAC, MAS operations within one clock cycle.
 - ٠ 8-bit Barrel shifter
 - Registers : 4 sets of registers. ٠
 - Embedded SRAM : embedded SRAM (word-length : 4-bit, address :0~15)

1.13 PA-PE Functions

Function	Mnemonics
A × B	PAMUL
$A \times B + out(t)$	MAC

Table	1.2.PA-	PE fi	inctions

A Novel 3D Vertically Integrated Adaptive Computing System Appendix B-High-level Modeling of 3D-SoftChip Using SystemC

MAS
LSL
LSR
ASR
ROR
ABS

1.14 PA-PE Instruction Format

18	17	16	15	12	11	10	9	8	6	5	3	2	0
WS_en/	WR_en/	SRAM	SRAM	Selection	Re	egister	DoutR	PA-F	E_OP	MU	ХВ	MU	KA
RS_en	RR_en	en			Sel	lection	Ctl						
			Figure	1.9: PA-PE in	structi	on form	nat						

1.15 PA-PE Block Diagram (In/Output Pin Description)



Figure 1.10: PA-PE block diagram (Input/Output Pin Description)

3D-SoftChip A Novel 3D Vertically Integrated Adaptive Computing System

Appendix B-High-level Modeling of 3D-SoftChip Using SystemC



Figure 1.11: Data-path architecture of PA-PE

1.17 PA-PE Operation Flow



Figure 1.12: PA-PE operation flow

3D-SoftChip A Novel 3D Vertically Integrated Adaptive Computing System Appendix B-High-level Modeling of 3D-SoftChip Using SystemC

1.18 SystemC File Structure



Figure 1.13: SystemC file structure of PA-PE

1.19 SystemC Codes for PA-PE

See Appendix C.

1.20 Output Waveform



Figure 1.14: Top level simulation of PA-PE

A Novel 3D Vertically Integrated Adaptive Computing System Appendix B-High-level Modeling of 3D-SoftChip Using SystemC

2 ICS(Intelligent Configurable Switch) Chip

2.1 ICS_RISC (32-bit Dedicated RISC Control Processor)



Figure 2.1: Overall architecture of ICS_RISC

2.2 ICS_RISC-Detailed Architecture



Figure 2.2: Detailed ICS_RISC Architecture

A Novel 3D Vertically Integrated Adaptive Computing System Appendix B-High-level Modeling of 3D-SoftChip Using SystemC

- 2.3 Special Features (ICS_RISC)
 - Harvard architecture, 3 Stage Pipelined architecture(Fetch, Decode, Execute)
 - Memory access, during the execution stage, is done by load/store instructions only
 - All operations except load/store, PE and DMA operations, are register-to-register within the ICS RISC
 - Single-cycle instruction execution
- 2.4 System Components (ICS_RISC)
 - Program Counter : 32th GPR is a program counter
 - Loop Buffer : 16 × 32-bit buffer to generate instruction address for iterative characteristic instructions
 - Register file(General Purpose Register): 32 × 32-bit general purpose register
 - Status Register : 4 kinds of flags (N: Negative / Less Than, Z: Zero, C: Carry / Borrow, V: Overflow)
 - Instruction Register: Instruction decoder for ALU and Control Unit
 - ALU & Control Unit: It is consist of ALU, Shifter, Multiplier
 - I/O Unit: 32-bit Data input/output register (dInReg, dOutReg)
 - ICS RISC Functions
 - See Appendix A

2.5

2.6 ICS_RISC Block Diagram (Input/Output Pin Description)



Figure 2.3: ICS_RISC Block Diagra.n (Input/Output Pin Description)

A Novel 3D Vertically Integrated Adaptive Computing System Appendix B-High-level Modeling of 3D-SoftChip Using SystemC

2.7 UnitICS Block Diagram (Input/Output Pin Description)



Figure 2.4: UnitICS Block Diagram (Input/Output Pin Description)

2.8 Three-stage Pipeline Architecture (ICS_RISC)

FETCH	DECODE	EXECUTE		
	FETCH	DECODE	EXECUTE]
		FETCH	DECODE	EXECUTE

Figure 2.5: Three-stage Pipeline Architecture (ICS_RISC)

2.9 Register Architecture (ICS_RISC)

	0
RO	
R1	
R2	
R3	
R4	
R5	
R6	
R25	
R26	
R27	
R28	
R29	
R30	
R31(PC)	



Figure 2.6: Register Architecture (ICS_RISC)

3D-SoftChip A Novel 3D Vertically Integrated Adaptive Computing System Appendix B-High-level Modeling of 3D-SoftChip Using SystemC





Figure 2.7: Data-path architecture of ICS_RISC

2.11 SystemC File Structure



Figure 2.8: SystemC file structure of ICS_RISC

2.12 SystemC Modeling of Data-path Architecture of ICS_RISC

- System Components: (1) Program Counter, (2) Status Register, (3) Loop Buffer, (4) General Purpose Register, (5) ALU, (6) Barrel Shifter, (7) Multiplier, (8) Data Input Register, (9) Data Output Register
- Program Counter (PC)

A Novel 3D Vertically Integrated Adaptive Computing System Appendix B-High-level Modeling of 3D-SoftChip Using SystemC

VCD loaded successfully. [11] facilities found. [699] regions found.	WAINETS.				etch Disc Shift	From Disec To: 1995 ns	Maximum Time 1995 ns Current Time
Signals Time	Waves 1	N	111700 ps		223400 ps	335100	232900 ps
SystemC.clock = SystemC.reset =(wwww	www	nnnnn	wwww	www
SystemC.iAregCtl =(SystemC.dAregCtl =I SystemC.aluOutl31:01=		110					
SystemC.iAddrTmp[31:0] = SystemC.incrOut[31:0] =	1 2 3	0 1 2 3	4 5 6 7 6 7 8 9			8 9 + + 12 + + + + 14	
SystemC iAddrOut = SystemC iAddrln[31:0] = SystemC iAddrl[31:0] =	120	1 2 3	4 5 6 7				
SystemC dAddr[31:0]=	0	16 10 14 10	10 17 10 110	110		12 17 17 17 17 14 140	



Status Register

°CD loaded successfully. 3] facilities found. 134] regions found.				Page Fetch Disc Shit	From: 0 se To: 495 n	Maximum Time c 495 ns s Current Time 94 ns
Signals	Waves	Contraction of the second			and the second s	244-0-
Time		56 ns	and a second	112 ns	168 ns	224 ns
SystemC.clock SystemC.reset SystemC.srOEn						
SystemC.wbSel						
SystemC.wbSel SystemC.wbData[3:0] SystemC.condFlag[3:0] SystemC zFlag	20000 20000	×0101 ×1100				



Loop Buffer (LB)
A Novel 3D Vertically Integrated Adaptive Computing System Appendix B-High-level Modeling of 3D-SoftChip Using SystemC



Figure 2.11: Output waveform of Loop Buffer

General Purpose Register (GPR)



Figure 2.12: Output waveform of Register File

• ALU

	Opcodes Mnemonics			Mnemonics	Description (Immediate)	Description (Register)
0	0	0	0	MOVA	Rd = Immediate	Rd = Rs1
0	0	0	1	MOVB	Rd = Immediate	Rd = Rs2
0	0	1	0	AND	Rd = Rd & Immediate	Rd = Rs1 & Rs2
0	0	1	1	OR	Rd = Rd Immediate	Rd = Rs1 Rs2
0	1	0	0	XOR	Rd = Rd ^ Immediate	$Rd = Rs1 \wedge Rs2$

Table2.1: ALU Functions

A Novel 3D Vertically Integrated Adaptive Computing System Appendix B-High-level Modeling of 3D-SoftChip Using SystemC

0	1	0	1	NOT	Rd = ~ Immediate	Rd = -Rs1
0	1	1	0	ADD	Rd = Rs1 + Immediate	Rd = Rs1 + Rs2
0	1	1	1	SUB	Rd = Rs1 – Immediate	Rd = Rs1 - Rs2
1	0	0	0	СМР	Compare Rs1 and Immediate	Compare Rs1 and Rs2
1	0	0	1	MSR	Status Register = Immediate	Status Register = Rs1
1	0	1	0	MRS	N/A	Rs1 = Status Register



Figure 2.13: Output waveform of ALU

32bit Barrel Shifter

Table2.2: Shifter Functions

	Shift		Mnemonics	Description
0	0	0	LSL	Shift Left
0	0	1	LSR	Shift Right
0	1	0	ASR	Arithmetic Shift Right
0	1	1	ROT	Rotate

A Novel 3D Vertically Integrated Adaptive Computing System Appendix B-High-level Modeling of 3D-SoftChip Using SystemC

VCD loaded successfully [5] facilities found. [52] regions found.	4	Zoom Page Fetcl	Disc Shift Fr	om: 0 sec 195 ns	Maximum Time 195 ns Current Time 18 ns
Signals V Time SystemC.cLK SystemC.shift(131:0) 3 SystemC.shiftCt[[2:0] 3 SystemC.shiftAmt[4:0] 3 SystemC.shiftOut[31:0] 3	Vaves 10 na 20000000+120000000000 200000 2000000122000000 20000000+120000000000	15 ns	20 ns 2001 200000000000	25 ns	11101

Figure 2.14: Output waveform of 32-bit Barrel Shifter

32 × 32 Signed Multiplier

VCD loaded successfully. [5] facilities found. [492] regions found.				Page Fetch	Disc Shift	From: 0 sec To: 995 ns	Maximum Time 995 ns Current Time 55 ns
Signals	Waves					4	
Time		34 ns		51 ns	Б	6 ns	85 ns
SystemC.clock SystemC reset							
SystemC.mulAln[31:0]	0		15	111	22	5	111
SystemC.mulBin[31:0]	0		10	2	7	10	2
SystemC.mulOut[31:0]	0		50	22	154	50	22

Figure 2.15: Output waveform of Signed 32 × 32 Multiplier

- Data Input Register
- Data Output Register

A Novel 3D Vertically Integrated Adaptive Computing System Appendix B-High-level Modeling of 3D-SoftChip Using SystemC

ded successfully lities found. regions found			20	om elee je Pag e UHDO - HI -	Fetch Disc	Shift	From 0 sec To: 99995 ns	Maximum Tin 99995 ns Current Tim 312 ns
W	aves	8	1 10	165/10		249. 110	33	2
SystemC.clock = 7	JUL	mm	mm	mm	nnnn	www	www	mm
mC imm[31:0] =: s emC.cmpFlag =: stemC.srOEn =:	00000000							
temC.srWbEn = mC.aluCti[3:0] =:	0		\$7	[\$4	190	\$7	54	Iso
emC.shiftOEn =	000		12011	2010	1%000	12011	2010	1%000
C.opAldx[4:0]=1 5	00	SOF	\$15	1914	SOF	\$15	\$14	SOF
C.opBidx(4:0) =! s lemC.rdAOEn =	00	1803		815	1903		1915	1803
nC.vbldx(4.0) =:	00		1615	1900	İŝOF	\$15 	1500	léof
emC:dOutCt) =(0000000	15000000A	1900000014	\$0000001E	Isoboooba	\$00000014	ISDODCODIE	1\$0000000A

Figure 2.16: Output waveform of top module in data-path architecture

- 2.13 Control Architecture of ICS_RISC
 - Fetch Unit : Fetch the instructions
 - Decoder Unit



Figure 2.17: Instruction Decoding (1)

A Novel 3D Vertically Integrated Adaptive Computing System Appendix B-High-level Modeling of 3D-SoftChip Using SystemC



Figure 2.18: Instruction Decoding (2)

Instruction ID	Instruction[31:25]	Description		
INST_ALUIS	000/0000	ALU Immediate (1 Inst. Word)		
INST_ALUIL	000/0001	ALU Immediate (2 Inst. Word)		
INST_ALUR	000/0010	ALU Register		
INST_ALULB	000/0011	ALU Loop Buffer Addressing		
INST_SHRO	001/0100	Shift / Rotate		
INST_LOAD	001/0101	Load		
INST_STORE	001/0110	Store		
INST_BRANCH	001/0111	Branch		
INST_PECON	010/1000	PE Control		
INST_DMA	1xx/xxxx	DMA Control		
INST_MUL	011/1111	Multiply		

Table 2.3: Instruction ID for Instruction Decoding

A Novel 3D Vertically Integrated Adaptive Computing System Appendix B-High-level Modeling of 3D-SoftChip Using SystemC

/CD loaded successfully. 33] facilities found. 71674 regions found.					1.152		1) 14- 100 ->	Pag	e Fet	th Di	sc Si	1111 		From To:	0 sec 99995 n	IS	-	Maximu 9999 Curren 495	im Time 15 ns 11 Time 3 ns
Signals	Waves								1050	-			100	10					
Time	1 DS	-		48	tell ns				4320	ns			50	40 85				5130	D.S
SystemC clock	1000			цц		цu		цц			UU.	цц			ЦЦ	uц	uц	цυ.	
SystemC reset	e.leoso	leaso	Jenio	lerer		C. 18.20	Paleona	lease	lenor	10017	11000	Jento	large	Terac	lean	lenic	1000	C. Lénot	leain
Systemo ansi[51 0]	3+19010	+12030	+12010	HOLLE	+1931	1230	0+12040	+19035	HOUDE	+ISDID	10030	+12010	+19FFF	+19510	+1230E	2+12040	+1503	5+19001	H1901D
SystemC refill	-			-			-			-	-		_						
SystemC instid[3:0]	50	1\$1	1\$0	199	Ise	-	\$2	1\$1	150	-	\$1	Iso	1\$9	158		\$2	191	Isa	-
SystemC.cond(2.0)	2010				-	_													
SystemC.opcode[3:0]	\$5 SE	Isc.	\$8	SF	SE.	\$5	\$3	SA	\$5	SE.	SC	\$8	SF	SE	\$5	\$3	15A	\$5	SE
SystemC shift[2:0]	2+ 2111	1%110	1 12100	1 1/11		1%01	0 12001	2/101	2010	2111	2110	2100	2111		8010	12001	1/10	1 2010	1 1:111
SystemC.rs1ldx[4:0]	\$+ \$02	\$1E	\$15	S1F	\$13	\$04	\$09	\$15	\$18	1502	SIE	\$15	SIF	\$13	\$04	\$09	\$15	\$18	\$02
SystemC rs2ldx[4:0]	\$+ \$0A	\$17	\$1E	\$1F	\$19	SOD	304	\$06	S1E	SUA	\$17	\$1E	\$1F	\$19	\$0D	\$04	\$06	\$1E	SOA
SystemC.rdidx[4 0]	\$+\$1F	\$00	\$08	\$1F	\$06	\$18	\$00	\$1À	\$17	\$1F	\$00	\$08	\$1F	\$06	\$18	\$00	1\$14	\$17	\$1F
SystemC imm[31:0]	\$+\$0.00	+ \$+ \$	+ \$+ \$	+ \$000	OFFFF			1	+ \$+ \$	+ \$000	+++++	+ \$+ \$	+ \$000	OFFFF			1	+ 5+ 3	+ \$000
SystemC PEOp[20]	2001				3	*+ *01	0. 2001							2	+ 3010	2 2001			
SystemC PEOpmode[1:0]	200	_				211	200								2:11	200			
SystemC PEConfig[1 0]	2:00				1	*+ 200							_	12	+ 200	_			
SystemC.PESel[3:0]	\$1				1	\$6 \$3	\$1			_				5	6 \$3	\$1			
SystemC DMAOp[2:0]	2101			12	+ 1210:	1						-	2	+ 2101	-				
SystemC.DFBSet					-									_					
SystemC.dataAmt[3.0]	\$3			S	F \$3	_	_	_	_				5	F 83	-				
SystemC_startAddrDFB[5:0]	\$23		-	\$	+ 923								15	+ \$23					
SystemC, SRAMRegSel	1			_	_	-		_	_		_	_		-					
SystemC startAddrSRAMReg[4:0]	\$19	-		S	+ \$19		_			_			\$	+ \$19	_				

Figure 2.19: Output waveform of Instruction Decoding

Execute Unit

Instruction	ALU Op.	ALU Out	Shifter Out	Mul. Out	Operand A	Operand B
ALU Immediate (1 Inst. Word)	Op Code	Enable	Disable	Disable	Rd	Immediate (4,8,16bit)
ALU Immediate (2 Inst. Word)	Op Code	Enable	Disable	Disable	Rd	Immediate (32bit)
ALU Register	Op Code	Enable	Disable	Disable	Rs1	Rs2
ALU LB Addr.	Op Code	Enable	Disable	Disable	Rs1	Rs2
Shift / Rotate	Don't Care	Disable	Enable	Disable	Rb (Rs1)	ShiftAmt
Load	MOV	Disable	Disable	Disable	Rb (Rs1)	Don't Care
Store	MOV	Disable	Disable	Disable	Rb (Rs1)	Rd
Branch	ADD	Enable	Disable	Disable	PC	Immediate
PE Control	Don't Care	Disable	Disable	Disable	Don't Care	Don't Care
DMA Control	Don't Care	Disable	Disable	Disable	Don't Care	Don't Care
Multiply	Don't Care	Disable	Disable	Enable	Rs1	Rs2

Table 2.4: Control Signal according to the Instruction

A Novel 3D Vertically Integrated Adaptive Computing System Appendix B-High-level Modeling of 3D-SoftChip Using SystemC

Search Tir	ne Markers Vie	w Help		_						
ns found, Insidewidd eted.			Ĵ		Page 4-	etch Disc	Shift	From: 0 sec To 9995 ns	– Maxie 9 – Cun 10	num Tim 195 ns ent Time 161 ns
	Waves									
		924 ns		1056	ns	1188 :	ns	1320 ns		1
stemC.clock	100000	ากกับกกก	บบบบบ	บบบบบบบบบบบบบบบบบบบบบบบบบบบบบบบบบบบบบบบ	บบบบบบบบ	กากกาก	ากกกกก	mmmm	www	m
C.instid[3:0]	50 51	\$2	\$4	\$0	\$1 \$2	\$4	\$0	\$1 \$2	54	\$0
nC.cond[2:0]	2000									
opcode[3:0]	\$6	\$2	190	56	1\$2	190	\$6	1\$2	180	196
mC.shift[2.0]	2000		2:001	1×000		1×001	1%000		1×001	200
C. rs1ldx[4.0]	900									-
Grs2ldx[4:0]	501									
Cimm[31:0]	\$000000000									_
C imm[310]	\$000000000									
nC cmnElag										
temC srOEn	11									-
mC stWhEn	10 mm						-			_
C.aluCti(3.0)	\$6	192	150	\$6	\$2	50	196	92	150	1\$6
amC.aluOEn										1
shiftCti[2:0]	2000		2001	2000		2001	2000		8001	200
mC.shiftOEn	110									
mC.mulOEn	1									100
opAldx[4:0]	1502	\$00		\$02	5	30	\$02	600		1\$
opBidx[4:0]	\$03	\$01	\$03		\$	01 903		1901	\$03	
mC.rdAOEn										
mC.rdBOEn	1									
C.wbldx(4:0)	902									
stemC.wbEn										
nc.immOEn										
mc taregoti										
nc.dAregCtl	1									
aemo dinoti				-						
hill Acut (4 Ct			1.	10		1.	10		1.	10
Amterul a	4		11	10		11	10		11	10
and sufficiely										

Figure 2.20: Output waveform of Instruction Execution

Pipeline Register



Figure 2.21: Conventional Pipeline Register Architecture



Figure2.22: Modified Pipeline Register Architecture (High-Speed)

A Novel 3D Vertically Integrated Adaptive Computing System Appendix B-High-level Modeling of 3D-SoftChip Using SystemC

Pipeline Control (reset, flush and refill)



Figure 2.23: Branch Instruction Execution

- 2.14 Top-level Simulation Result of ICS_RISC
 - Simple Program for Verification

0000/0000	//MOV	R0,	#0	//Simple Loop Program
0001/0001	//MOV	R1,	#1	
0002/0002	//MOV	R2,	#2	
0003/0003	//MOV	R3,	#3	
0004/0004	//MOV	R4,	#4	
0005/0005	//MOV	R5,	#5	
0006/0006	//MOV	R6,	#6	
0007/0007	//MOV	R7,	#7	
0408/0000	//MOV	R8,	RO	
0409/0020	//MOV	R9,	R1	
040A/0040	//MOV	R10,	R2	
040B/0060	//MOV	R11,	R3	
040C/0080	//MOV	R12,	R4	
040D/00A0	//MOV	R13,	R5	
040E/00C0	//MOV	R14,	R6	
040F/00E0	//MOV	R15,	R7	//End
0450/4280	//AND	R16,	R8 & R9	//Simple ALU Program
0471/52C0	//OR	R17,	R10 R11	
0492/6340	//XOR	R18,	R12 ^ R13	
04D3/6B80	//ADD	R19,	R14 + R15	

3D-SoftChip A Novel 3D Vertically Integrated Adaptive Computing System Appendix B-High-level Modeling of 3D-SoftChip Using SystemC

04F4/6B80

) //SUB R20,

R20, R14 – R15

//End

(3650) regions found. Dragging 2 traces. Drop completed.	4	Zoom Pags Fetch	Disc Shift From	0 sec Maximum Time 4995 ns Current Time
Signals	Waves			138 ns
Time	62 ns	124 ns	186 ns	248 ns
SystemC, clock =1 SystemC, raset =1 SystemC, IData[31:0] =1 SystemC, apAldx[4:0] =1 SystemC, opBldx[4:0] =1 SystemC, rdAOEn =	<u>\$+ \$0+ \$0+ \$0+ \$0+ \$0</u> <u>\$00 \$01 \$02 \$03 \$0</u> <u>\$03</u>	+ \$0+ \$0+ \$0+ \$0+ \$0+ \$0+ \$0+ 4 \$05 \$06 \$07 \$08 \$00 \$01 \$0; \$00	190+ 190+ 190+ 190+ 190+ 190 190+ 190+ 190+ 190 1903 ISOA ISOB 1908 ISOB	+ \$Q+ \$Q+ \$Q+ \$Q+ \$Q+ \$Q+ \$Q + \$QE \$QQ \$Q1 \$Q2 \$Q3 \$Q D \$QE \$QQ \$Q3 C \$QD \$QQ \$Q3
SystemC.rdBOEn=" SystemC.wbldx[4:0]=: SystemC.aluOut[31:0]=: SystemC.iAddr[31:0]=:	\$00 s01 s02 s03 s0 \$00000000 s0+ s0+ s0+ s0+ s0+ s0+	4 \$05 \$05 \$07 \$08 \$09 \$0A \$0B \$00 \$0+ \$0+ \$0+ \$0+ \$0+ \$0+ \$0+ \$0+	: \$0D \$0E \$0F \$10 \$11 \$1 \$0+ \$0 \$0+ \$0+ \$0+ \$0+ \$0+ \$0+	2 \$13 \$14 \$00 \$01 \$02 \$03 \$0 + \$0+ \$0+ \$00000000 \$0+ \$0+ \$0+ \$0+ \$0+ \$0+ \$0+

Figure2.24: Top level Simulation Result of ICS_RISC



Figure2.25: Instruction Index

A Novel 3D Vertically Integrated Adaptive Computing System Appendix B-High-level Modeling of 3D-SoftChip Using SystemC

3 UnitChip

3.1 SystemC File Structure



Figure 3.1: SystemC file structure of UnitChip

3.2 Top-level Simulation Result of UnitChip

gnale				1		P	Ange Fe	tch Dis				From To) sec 195 ns	-	4995 Current 144	Time ns
	Waves															
me		102 ns			136	ns			170 ns				204 :	15		
SystemC.clock								_	-5	2	L				201	
SystemC.reset									1				-			-
SystemC.iData[31:0]	1\$0006+ 1\$00	07+ \$0408+	\$0409+	1\$040A-	+ 1\$040B	+ 150400	+ 190400	+ 19045	E+ 1504	OF+	\$0450-	+ \$0471	+ 1\$0497	2+ [\$04D	3+ 1504F	4+ 1700
SystemC.opAldx[4:0]	1\$05 1\$06	1\$07	1\$08	\$00	1901		1902	_	1903	-		AD\$	1\$08	1\$0D	ISOE	-
SystemC.opBldx[4:0]	\$03		ICC	1\$00	~			_		_		\$08	1\$0A	1\$00	1\$0D	_
SystemU, rdAUEn			ICS	RISC	Ope	ratio	ns-	-								
Systeme.rdbuch	Tent lens	len7	leng	leng	leni	lenn	leor	Ison	leng		enr	lein	lett	1015	1017	1014
Systemu wordx[4:0]	1505 1506	1907	1908	1509	1996	1908	Iane	1900	SS UE	-	SUE	1910	1911	1. 10049	24 1040	24 1004
SvetamC (Addr31:0)	S0+ 150000+	Isnnn+ Isn	nna+ Isni	ana+ Isr	ionna Ter		nnnn+ Is	noon+ B	+0000+	Isonr	n+ Isi	funce la	14047	minne le	COOR4 I	100004
SystemC PE1_dOut[3.0]	sh	1990001 190	4441 1445	1001 140	10001-100	10004 145	99991-19	00001 1		10000	\$7	159	000011	153	Iso	COUSE +
SystemC PE2 dOut[3:0]	\$0	-		_							1.7	Iso	ISA	SE	1\$0	-
SystemC.PE3 dOut/3:01	\$0									1				ISC.	100	
SystemC PE4_dOut[3:0]	\$0			P	E Ope	eratic	ons_	_	-	1	\$B	190	\$5	\$9	\$0	
avatemC.PE5 dOut[3:0]	\$0							~		100	\$1	\$7	\$E	\$7	136	
SystemC.PE5_dOut(3.0)	\$0									-		\$1	SC.	SB	182	
SystemC.PE7_dOut[3:0]	\$0			_		-					\$4		\$9	[\$0	1	
SystemC.PE8_dOut(3:0)	\$0										\$5	\$7			\$1	
systemC.PE9_dOut[3.0]	\$0							-	1		\$7			\$5	51	
stemC.PE10_dOut[3:0]	\$0								- 1	-		\$9	\$4		ES	
stemC.PE11_dOut[3:0]	\$0		_							1	\$B	ISC	\$1	\$0	115	-
stemC.PE12_dOut[3:0]	\$0									3	\$7	\$B	\$7	\$1	1\$8	-
stemC.PE13_dOut[3:0]	\$0								_	1	\$9	\$5	1\$0	\$8	\$7	_
stemC,PE14_dOut[3:0]	50						_		_	-	.c	1\$0	1	_	196	-
ystemC PE15_dOut[3:0]	50			_	-					-	SF	ISA	180	1\$9	ISE	

Figure3.2: Top-level Simulation Result of UnitChip

3D-SoftChip A Novel 3D Vertically Integrated Adaptive Computing System Appendix B-High-leyel Modeling of 3D-SoftChip Using SystemC

References for the ICS_RISC

[1] Yeong-don Bae, "Basic Microprocessor Design", http://www.donny.co.kr

[2] Yap Zi He, "Building A RISC Microcontroller in an FPGA",

http://www.opencores.org/projects/risemcu

A Novel 3D Vertically Integrated Adaptive Computing System Appendix C-SystemC Codes

SystemC Codes

1 Standard-PE.

1*

- * iReg: Instruction Reg for Standard-PE(header file for iReg)
- * Copyright(c) 2005 by Chul KIM, All right reserved
- * Author: Chul KIM(ckim@student.ecu.edu.au)

* File name: iReg.h

* Revision history: Version1

* Date: 17/1/2005

*/

#include "systemc.h"

SC_MODULE(iReg) {

sc_in<sc_uint<19>>
sc_out<sc_uint<3>>
sc_out<sc_uint<3>>
sc_out<sc_uint<3>>
sc_out<sc_uint<3>>
sc_out
sc_out
sc_out<cc_uint<2>>
sc_out
sc_out<sc_uint<4>>
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_out
sc_ou

muxACtl; muxBCtl; sopSel; doutRCtl; regSel; sramSel; sramEn; rwRegEn; rwSEn;

inst:

void do_iReg();

SC_CTOR(iReg) { SC_METHOD(do_iReg); sensitive << inst;

#ifdef SIM

muxACtl.initialize(0); muxBCtl.initialize(0); sopSel.initialize(0); doutRCtl=0; regSel.initialize(0); sramSel.initialize(0); sramEn=0; rwRegEn=0; rwSEn=0;

#endif

1;

14

- * iReg: Instruction Reg for Standard-PE(source file for iReg)
- * Copyright(c) 2005 by Chul KIM, All right reserved
- * Author: Chul KIM(ckim@student.ecu.edu.au)
- * File name: iReg.cpp
- * Revision history : Version1
- * Date: 17/1/2005

*1

#include "iReg.h"

void iReg::do_iReg() {
 sc_uint<19> tmp_inst;
 tmp_inst = inst.read();

rwSEn = tmp_inst[18]; rwRegEn = tmp_inst[17]; //instruction input //muxA Ctl //muxB Ctl //S-PE operation sel //data-out reg ctl //internal reg sel //SRAM sel //SRAM enable signal //internal reg read/write signal //SRAM read/write enable signal

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes

```
sramEn
                                 = tmp_inst[16];
           sramSel
                                 = tmp_inst.range(15,12);
           regSel
                                 = tmp_inst.range(11,10);
                                 = tmp_inst[9];
           doutRCtl
           sopSel
                                 = tmp_inst.range(8,6);
           muxBCtl
                                 = tmp_inst.range(5,3);
                                 = tmp_inst.range(2,0);
           muxACtl
1
1*
 * Mux: Mux for Standard-PE(header file for Mux)
 * Copyright(c) 2005 by Chul KIM, All right reserved
 * Author: Chul KIM(ckim@student.ecu.edu.au)
 * File name: mux.h
 * Revision history: Version1
 * Date: 17/1/2005
 */
#include "systemc.h"
SC_MODULE(mux) {
                                             muxCtl;
                                                                  //mux ctl input
           sc_in<sc_uint<3>>
           sc_in<sc_uint<4>>
                                             dIn;
                                                                   //input data
                                                                  //data from internal Reg
           sc in<sc uint<4>>
                                            dReg;
                                                                  //data from adjacent PE(from left PE)
           sc_in<sc_uint<4>>
                                            dLeft;
           sc_in<sc_uint<4>>
                                            dRight;
                                                                  //data from adjacent PE(from right PE)
           sc_in<sc_uint<4>>
                                            dUp;
                                                                  //data from adjacent PE(from upside PE);
           sc in<sc uint<4>>
                                            dDown;
                                                                  //data from adjacent PE(from downside PE);
           sc_out<sc_uint<4>>
                                            muxOut;
           sc_out<bool>
                                            dReg;
                                                                  //data request for internal register
           void do_mux();
           SC_CTOR(mux) {
                      SC_METHOD(do_mux);
                      sensitive << muxCtl << dIn << dReg << dLeft << dRight << dUp << dDown;
#ifdef SIM
                      muxOut.initialize(0);
                      dReq=0;
#endif
1;
1+
 * Mux: Mux for Standard-PE(source file for Mux)
 * Copyright(c) 2005 by Chul KIM, All right reserved
 * Author: Chul KIM(ckim@student.ecu.edu.au)
 * File name: mux.cpp
 * Revision history: Version1
 * Date: 17/1/2005
 */
#include "mux.h"
void mux::do_mux() {
           switch (muxCtl.read()) {
                                                        dReq=0;
                                                                   break;
                      case 0: muxOut = dIn;
                      case 1: muxOut = dReg;
                                                        dReq=1;
                                                                   break;
                                                        dReq=0;
                                                                   break:
                       case 2: muxOut = dLeft;
                       case 3: muxOut = dRight;
                                                        dReq=0;
                                                                   break;
                                                        dReq=0;
                       case 4: muxOut = dUp;
                                                                   break;
                       case 5: muxOut = dDown;
                                                        dReq=0;
                                                                   break;
                                                                   break;
                       default:
            3
}
```

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes

```
/*
 * SPE: Standard-PE for CAP(Configurable Array Processor)(header file for SPE)
 * Copyright(c) 2005 by Chul KIM, All right reserved
 * Author: Chul KIM(ckim@student.ccu.edu.au)
 * File name: spe.h
 * Revision history: Version1
 * Date: 17/1/2005
 */
#include "systemc.h"
#include "iReg.h"
#include "mux.h"
SC_MODULE(spe) {
           sc_in<boob
                                            clock;
           sc_in<bool>
                                            reset:
           sc_in<sc_uint<19>>
                                            instICS;
                                                      //instruction input from ICS
                                            dIn, dLeft, dRight, dUp, dDown;
           sc_in<sc_uint<4>>
                                                                                        //data inputs
           sc_out<sc_uint<4> >
                                            dOut:
                                                                 //data output
           sc_out<sc_uint<4>>
                                            dOutadjPE;
                                                                  //data output for adjacent PEs
           //temp signal for Instruction
           sc_signal<sc_uint<19>>
                                            s_inst;
           //temp signals from iReg(Instruction Decoder)
           sc_signal<sc_uint<3>>s_muxACtl;
           sc_signal<sc_uint<3>> s_muxBCtl;
           sc_signal<sc_uint<3>> s_sopSel;
           sc_signal<bool>
                                   s_doutRCtl;
           sc_signal<sc_uint<2>> s_regSel;
           sc_signal<sc_uint<4>> s_sramSel;
           sc_signal<bool>
                                s_sramEn;
           sc_signal<bool>
                                 s_rwRegEn;
                                 s_rwSEn;
           sc_signal<bool>
           //temp signals for mux in/output and ALU inputs
           sc_signal<sc_uint<4>>s_dIn, s_dLeft, s_dRight, s_dUp, s_dDown;
           sc_signal<sc_uint<4>>dRegOutA;//reg out for muxA input
           sc_signal<sc_uint<4>>dRegOutB;//reg out for muxB input
           sc_signal<sc_uint<4>>muxAOut;
           sc_signal<sc_uint<4>>muxBOut;
           sc_signal<bool>
                                 dRcqA, dReqB;
                                                       //data request for register
           //temp signal for ALU output
           sc_signal<sc_uint<4>>aluOut;
           //temp signal for internal register
           sc_signal<sc_uint<4> >regIn;
           sc_signal<sc_uint<4>>tmp1, tmp2, tmp3, tmp4;
           sc_signal<sc_uint<4>>regOut;
           //temp signals for SRAM
           sc_signal_rv<4>
                                sramData;
           sc iv<4>
                                 ramData[16];
           //temp signal for output data bus
//
           sc_signal<sc_uint<4>>doutBus;
           sc_signal<sc_lv<4>> doutBus;
           void do_latch();
           void do _alu();
           void do_reg();
           void do_sram();
           void do_doutReg();
           iReg*
                      iReg1;
           iReg*
                      iReg2;
           iRcg*
                      lReg3;
           iReg*
                      iReg4;
```

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes

mux•	muxA;
mux*	muxB;

SC_CTOR(spe) {

iReg1=new iReg("iReg"); IRcg1->inst(s_inst); iRcg1->muxBCtl(s_muxBCtl); iReg1->doutRCtl(s_doutRCtl); iReg1->sramSel(s_sramSel); iReg1->rwRegEn(s_rwRegEn); iReg2=new iReg("iReg"); iReg2->inst(s_inst); iReg2->muxBCtl(s_muxBCtl); iReg2->doutRCtl(s_doutRCtl); iReg2->sramScl(s_sramScl); iReg2->rwRegEn(s_rwRegEn); iReg3=new iReg("iReg"); iReg3->inst(s_inst); iReg3->muxBCtl(s_muxBCtl); iReg3->doutRCtl(s_doutRCtl); iReg3->sramSel(s_sramSel); iReg3->rwRegEn(s_rwRegEn); iRcg4=new iRcg("iRcg"); iReg4->Inst(s_inst); iReg4->muxBCtl(s_muxBCtl); iReg4->doutRCtl(s_doutRCtl); iReg4->sramSel(s_sramSel); iReg4->rwRcgEn(s_rwRegEn);

muxA=ncw mux("mux"); muxA->nuxCtl(s_muxACtl); muxA->dReg(dRegOutA); muxA->dRight(s_dRight); muxA->dDown(s_dDown); muxA->dReq(dReqA); muxB=new mux("mux"); muxB->muxCtl(s_muxBCtl); muxB->dReg(dRegOutB); muxB->dRight(s_dRight); muxB->dRown(s_dDown); muxB->dReq(dReqB);

SC_METHOD(do_latch); sensitive_pos << clock; sensitive_neg << reset;

SC_METHOD(do_alu); sensitive << clock << muxAOut << muxBOut << s_sopSel << s_rwRegEn;

SC_METHOD(do_reg); sensitive << clock << s_regSel << s_rwRegEn << regIn << dReqA << dReqB;

SC_METHOD(do_sram); sensitive << clock << s_rwSEn << s_sramEn << s_sramSel << regin << sramData;

SC_METHOD(do_doutReg); sensitive << clock << s_doutRC(l << doutBus;

#ifdef SIM

doutBus.initialize(0); dOut.initialize(0); dOutadjPE.lnitialize(0); for (int i=0; i<16; i++) ramData[l]="XXXX";

#endif

ł

};

iReg1->muxACtl(s_muxACtl); iReg1->sopSel(s_sopSel); iReg1->regSel(s_regSel); iReg1->regSel(s_regSel); iReg1->rwSEn(s_rwSEn);

iReg2->muxACtl(s_muxACtl); iReg2->sopSel(s_sopSel); iReg2->regSel(s_regSel); iReg2->sramEn(s_sramEn); iReg2->rwSEn(s_rwSEn);

IReg3->muxACtl(s_muxACtl); IReg3->sopSel(s_sopSel); IReg3->regSel(s_rcgSel); iReg3->sramEn(s_sramEn); iReg3->rwSEn(s_rwSEn);

iReg4->muxACtl(s_muxACtl); iReg4->sopSel(s_sopSel); iReg4->regSel(s_regSel); iReg4->sramEn(s_sramEn); iReg4->rwSEn(s_rwSEn);

muxA->dIn(s_dIn); muxA->dLeft(s_dLeft); muxA->dUp(s_dUp); muxA->muxOut(muxAOut);

muxB->dIn(s_dIn); muxB->dLeft(s_dLeft); muxB->dUp(s_dUp); muxB->muxOut(muxBOut);

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes

```
/+
 * SPE: Standard-PE for CAP(Configurable Array Processor)(source file for SPE)
 * Copyright(c) 2005 by Chul KIM, All right reserved

    Author: Chul KIM(ckim@student.ecu.edu.au)

 • File name: spe.cpp
 * Revision history: Version1
 * Date: 17/1/2005
 •/
#include "spe.h"
// Latch
vold spe::do_latch() {
            If (reset) {
                       s_inst.write(0);
                                               s_dIn.write(0);
                                                                       s_dLeft.write(0);
                       s dRight.write(0);
                                               s_dUp.write(0);
                                                                      s_dDown.write(0);
           | clse |
                       s_inst.write(instICS.read());
                       s_dIn.write(dIn.rcad());
                                                          //input data bus
                       s_dLeft.write(dLeft.read());
                       s_dRight.write(dRlght.read());
                       s_dUp.write(dUp.read());
                       s_dDown.write(dDown.rcad());
           }
ł
// ALU
#define comp(a,b) (((a)>(b))?1: (((a)==(b))?0: -1))
                                                          //comparator
void spe::do_alu() {
#ifdef SIM
           unsigned short result=0;
#else
           unsigned short result;
#endif
           unsigned short src1=muxAOut.read();
           unsigned short src2=muxBOut.read();
           switch(s_sopSel.read()) {
                       case 0: result = src1 & src2;
                                                                      break;
                                                                                  //and
                       case 1: result = src1 | src2;
                                                                      break;
                                                                                  llor -
                       case 2: result = -srcl;
                                                                      break;
                                                                                  //not
                       case 3: result = src1 ^ src2;
                                                                      break;
                                                                                  lixor
                                                                                  //add
                       case 4: result = src1 + src2;
                                                                      break;
                       case 5: result = src1 - src2;
                                                                      break;
                                                                                  //sub
                       case 6: result = src1 * src2;
                                                                      break;
                                                                               //spmul
                       case 7: result = comp(src1,src2);
                                                                      break;
                                                                                  //comp
                       default:
                                                                      break;
           aluOut.write(result);
           regIn.write(result);
}
// Internal Register
void spe::do_reg() {
           if (s_rwRegEn) {
                                               //read operation
                       switch (s_regSel.read()) {
                                   case 0: regOut.write(tmp1);
                                                                                  break;
                                   case 1: regOut_write(tmp2);
                                                                                  break;
                                   case 2:
                                               regOut.write(tmp3);
                                                                                  break:
                                   case 3:
                                               regOut.write(tmp4);
                                                                                  break:
                                   default:
                                                                                  break;
                       if (dRegA) {
                                                                      //output control
                                   dRegOutA = regOut;
                       } else {
                                   doutBus = sc_lv<4> (regOut);
                                   dOut = sc_uint<4> (doutBus);
                       ł
```

3D-SoftChip A Novel 3D Vertically Integrated Adaptive Computing System Appendix C-SystemC Codes

		if (dRcqB) { } else { }	{ dRegOutB = regOut; doutBus = sc_lv<4> (re dOut = sc_uint<4> (dou	gOul); htBus);	
	} else (switch (s_re	egScl.read()) { case 0: tmp1 = regIn; case 1: tmp2 = regIn; case 2: tmp3 = regIn; case 3: tmp4 = regIn; default:	//write operation	break; break; break; break; break;
}	}	}			
// SRAM void spe::d	o_sram() { if (s_sram}	En) { if (s_rwSE1 } else { }	b) { sramData.write(ramDa doutBus = sramData; dOut = sc_uint<4> (sra dOut = sc_uint<4> (do sramData= sc_lv<4> (do ramData[s_sramSel.re	//read operation ata[s_sramSel.read()]); amData); autBus); // dOut has a //write operation regIn); ead()] = sramData;	dummy value(#F)
}	} else { }	sramData :	= "ZZZZ";		
// Data out void speitd	put register lo_doutReg if (s_doutF }	i) { RCti) { dOutadjPE	E = sc_uint<4> (doutBu	s);	

2 Proceeding Accelerator-PE.

```
/*
 * iReg: Instruction Reg for Processing Accelerator-PE(header file for iReg)

    Copyright(c) 2005 by Chul KIM, All right reserved

 * Author: Chul KIM(ckim@student.ecu.edu.au)
 • File name: (Reg.h
 * Revision history: Version1
 * Date: 29/1/2005
 4;
#include "systeme.h"
SC_MODULE(iReg) (
           sc_in<sc_uint<19>>
                                                                               //instruction input
                                                        inst;
                                                        muxACtl:
                                                                               //muxA Cil
           sc_out<sc_uint<3> >
           sc_out<sc_uint<3>>
                                                         muxBCil;
                                                                               //muxB Ctl
                                                        sopSel:
                                                                               //S-PE operation sel
           sc_out<sc_uint<3>>
           sc_out<bool>
                                                         doutRCil;
                                                                               //data-out reg ctl
                                                                               //internal reg sel
                                                        regSel;
           sc_out<sc_uint<2>>
                                                                               //SRAM sel
                                                        sramSel;
           sc_out<sc_uint<4> >
           sc_out<bool>
                                                        sramEn;
                                                                               //SRAM enable signal
           sc_out<bool>
                                                         rwRegEn;
                                                                               //internal reg read/write signal
                                                                               //SRAM read/write enable signal
           sc_out<bool>
                                                         rwSEn;
           void do_iRcg();
           SC_CTOR(iReg) [
                       SC_METHOD(do_iReg);
                       sensitive << inst;
#ifdef SIM
                       muxACtl.initialize(0);
                       muxBCti.initialize(0);
                       sopSel.initialize(0);
                       doutRC11=0;
                       regSel.initialize(0);
                       sramSel.initialize(0);
                       sramEn=0;
                       rwRegEn=0;
                       rwSEn=0;
#endif
           }
};
 * iReg: Instruction Reg for Processing Accelerator-PE(source file for iReg)
 * Copyright(c) 2005 by Chul KIM, All right reserved
 * Author: Chul KIM(cklm@student.ecu.edu.au)
 * File name: iReg.cpp
 * Revision history: Version1
 * Date: 29/1/2005
 +7
#include "iReg.h"
void iReg::do_iReg() {
            sc_uint<19>
                                  tmp_inst;
           imp_inst = inst.read();
            rwSEn
                                  = tmp_inst[18];
            rwRegEn
                                  = tmp_inst[17];
            sramĒn
                                  = tmp_inst[16];
            sramSel
                                  = tmp_inst.range(15,12);
                                  = tmp_inst.range(11,10);
            regSel
            doutRCtl
                                  = tmp_inst[9];
                                  = tmp_inst.range(8,6);
            sopSel
            muxBCtl
                                  = tmp_inst.range(5,3);
                                  = tmp_inst.range(2,0);
            muxACtl
}
```

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes



- * ALU: ALU for Processing Accelerator-PE(header file for ALU)
- * Copyright(c) 2005 by Chul KIM, All right reserved
- * Author: Chul KIM(ckim@student.ccu.cdu.au)
- * File name: alu.h
- Revision history: Version1
- + Date: 29/1/2005
- */

#include "systemc.h"

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes

```
SC_MODULE(alu) {
           sc_in<sc_uint<4>>
                                  aloAIn;
           sc_in<sc_uint<4>>
                                  aluBln;
           sc_in<sc_uint<3>>
                                  aluCtl:
           sc_out<sc_ulnt<4> >
                                  aluOut;
           sc_out<sc_uint<4> >
                                  regIn;
           sc_out<sc_uint<4> >
                                  regTmp;
                                                          //temp reg for MAC,MAS & output for test
           void do_alu();
           SC_CTOR(alu) {
                       SC_METHOD(do_alu);
                       sensitive << aluAIn << aluBIn << aluCil;
#ifdef SIM
                       aluOut.initialize(0);
                       regIn.initialize(0);
                       regTmp.initialize(0);
#endlf
           ł
};
 * ALU: ALU for Processing Accelerator-PE(source file for ALU)

    Copyright(c) 2005 by Chul KIM, All right reserved

 * Author: Chul KIM(ckim@student.ccu.cdu.au)
 * File name: alu.cpp
 * Revision history: Version1
 * Date: 29/1/2005
 */
#include "alu.h"
//#define MAC(A,B,P) (((A)*(B))+(P))
                                                          //mac
//#define MAS(A,B,P) (((A)*(B))-(P))
                                                          //mas
//#define
           //asr-when the data-type is signed, it should be modified
                      ((((A \& 0 x 0 f) \& 0 x 1)?(((A \& 0 x 0 f) >> 0 x 1)[0 x 8):(A \& 0 x 0 f) >> 0 x 1) \& 0 x 0 f) //rotate
#define
           ROR(A)
#define
           ABS(A)
                       (((A&0x0f)<0x0?(-1*(A&0x0f)):(A&0x0f))&0x0f) //abs
void alu::do_alu() {
           sc_uint<4> result,src1,src2,tmp,mulTmp;
                                                                     //temp signals
           src1 = aluAIn.read();
           src2 = aluBIn.read();
           switch (aluCtl.read()) {
                       case 0: result = src1*src2;
                                                                                 break:
                                                                                             //pamul
                       case 1: mulTmp = src1*src2;
                                  result = mulTmp + sc_uint<4> (regTmp);
                                                                                 break; //mac
                       case 2: mulTmp = src1*src2;
                                  result = mulTmp - sc_uint<4> (regTmp);
                                                                                 break;
                                                                                             //mas
                       case 3: result = src1<<1;
                                                                                 break;
                                                                                          //Isl
                       case 4: result = src1>>1;
                                                                                 break;
                                                                                         //lsr
                       //when the data-type is signed, it should be modified(asr)
                       case 5: result = src1>>1;
                                                                                 hreak; //asr(divider/2)
                       case 6: result = ROR(src1);
                                                                                 break; //ror
                       //when the data-type is signed, it can be applied(abs)
                                                                                 break; //abs
                       case 7: result = ABS(src1);
                       default:
                                                                                 break;
            aluOut.write(result);
            regIn.write(result);
            tmp = aluOut;
            regTmp.write(tmp); //defined in the header file, signal for Test
           //sc_out<sc_uint4> > regTmp;
```

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes

```
/*

    PAPE: Processing Accelerator-PE for CAP(header file for PAPE)

 * Copyright(c) 2005 by Chui KIM, All right reserved
 * Author: Chul KIM(ckim@student.ccu.edu.au)
 * File name: pape.h
 * Revision history: Version1
 * Date: 29/1/2005
 •7
#include ''systeme.h''
#include "iReg.h"
#include "nux.h"
#include "alu.h"
SC_MODULE(pape) {
                                            clock:
           sc_in<bool>
           sc_in<bool>
                                            reset;
                                                       //instruction input from ICS
                                             instICS;
           sc_in<sc_uint<19>>
                                             dIn, dLeft, dRight, dUp, dDown;
                                                                                         //data inputs
           sc_in<sc_uint<4>>
                                                                   //data output
                                             dOut;
           sc_out<sc_uint<4>>
                                             dOutadjPE;
                                                                   //data output for adjacent PEs
           sc_out<sc_uint<4>>
           //temp signal for instruction
           sc_signal<sc_uint<19>>
                                            s_inst;
           //temp signals from iReg(Instruction Decoder)
           sc_signal<sc_uint<3>>s_muxACtl;
           sc_signal<sc_uint<3>> s_muxBCtl;
           sc_signal<sc_uint<3>> s_sopSel;
           sc_signal<bool>
                                 s_doutRCtl;
           sc_signal<sc_uint<2>> s_regSel;
           sc_signal<sc_uint<4> > _s_sramSel;
           sc signal<bool>
                                   s_sramEn;
                                   s_rwRegEn;
           sc_signal<bool>
           sc signal<bool>
                                   s_rwSEa;
           //temp signals for mux in/output and ALU inputs
           sc_signal<sc_uint<4>>s_dIn, s_dLeft, s_dRight, s_dUp, s_dDown;
           sc_signal<sc_uint<4>>dRegOutA;//reg out for muxA input
           sc_signal<sc_uint<4> >dRegOutB;//reg out for muxB input
           sc_signal<sc_uint<4>>muxAOut;
           sc_signal<sc_uint<4>>muxBOut;
                                 dReqA, dReqB;
                                                        //data request for register
           sc_signal<bool>
           //temp signal for ALU output
           sc_signal<sc_uint<4>>s_aluOut;
           sc_signal<sc_uint<4>>s_regTmp;
           //temp signal for internal register
           sc_signal<sc_uint<4>>s_rcgIn;
           sc_signal<sc_uint<4>>tmp1, tmp2, tmp3, tmp4;
           sc_signal<sc_uint<4>>regOut;
           //temp signals for SRAM
                                  sramData;
            sc_signal_rv<4>
                                  ramData[16];
            sc_lv<4>
           //temp signal for output data bus
            sc_signal<sc_uint<4>>doutBus;
 //
            sc_signal<sc_lv<4>> doutBus;
            void do_latch();
            void do_reg();
            void do_sram();
            void do_doutReg();
            iReg*
                       iReg1;
            iReg*
                       iReg2;
            iReg*
                       Reg3;
```

3D-SoftChip A Novel 3D Vertically Integrated Adaptive Computing System <u>Appendix C-SystemC Codes</u>

	iReg*	iReg4;	
	mux*	muxA;	
	mux*	muxB;	
	alu*	aluPAPE;	
	SC_CTOP	R(pape) {	
		Regi=new ineg("ineg");	(Dept.) - mont Cit/e mont Cit) -
		(Reg1->inst(s_inst);	Regi->muxACH(s_muxACH);
		(Reg1->nuxDCh(s_muxDCh);	(Reg1->sopSet(s_sopSet);
		iDog1 > gramSal/a symmSally	iReg1+>reg3ci(s_reg3ci);
		iDeg1->ris DegEn(c_staticSet);	iDeg1 >sranch(s_sranch);
		iDon2-now iDon("iDon"))	INCH1->1 #3EII(5_1 #3EII);
		iReg2-new integ integ);	Beg2.sminACtl/s_munACtl)
		Ree2->mat(a_tist);	iReg2->muxACh(s_muxACh);
		Rev2->doutRCl(s_doutRCl);	IReg2->soport(a_soport);
		Reg2.>sramSel(s_sramSel)	iReg2->reg3ci(3_reg3ci);
		Reg2->rwRegEn(s_srunger);	iReg2->sinilin(s_sinilin);
		Regarnew (Reg("iReg"))	Integration and the provide a second
		iRegi-sinst/s inst):	iRea3.smuxACtl(s_muxACtl);
		Reg3.smuxBCtl(s_muxBCtl)	iReg3-stonSol/s tonSol):
		iReg3-sdoutRCtl(s_doutRCtl);	iPen3.>reaSel(< reaSel);
		iReg3.5sramSel(s sramSel)	iRega-scamEn(s cramEn)
		iReg3.>rwRegFn(s_rwRegEn);	iReo3.sruSEnis ruSEn).
		iRend=new iReo("iRen").	mego->1 ************************************
		iReod.sinstis inchi	iReg4.smuxACtl(s_muxACtl);
		iReed.smuxBCil(s_muxBCil)*	iReg4->mds/red(s_mds/setu);
		iReg4->doutRCtl(s_doutRCtl);	iReg4->regSel(s_regSel);
		iReg4->sramSel(s_sramSel):	iReg4->sramEn(s sramEn);
		iReg4->rwRegEn(s_rwRegEn);	iReg4->rwSEn(s_rwSEn):
		muxA=new mux("mua");	
		<pre>muxA->muxCtl(s_roaxACtl);</pre>	nmxA->dIn(s_dIn);
		muxA->dReg(dP.gOutA);	muxA->dLeft(s_dLeft);
		muxA->dRight(s_dRight);	muxA->dUp(s_dUp);
		muxA->dDown(s_dDown);	muxA->muxOut(muxAOut);
		muxA->dReq(dReqA);	
		muxB=new mux("mux"');	
		muxB->muxCtl(s_nmxBCtl);	muxB->dIn(s_dIn);
		muxB->dReg(dRegOutB);	muxB->dLeft(s_dLeft);
		muxB->dRight(s_dRight);	muxB->dUp(s_dUp);
		muxB->dDown(s_dDown);	muxB->muxOut(muxBOut);
		muxB->dReq(dRcqB);	
		aluPAPE=new alu("alu"");	
		aluPAPE->aluAln(muxAOut);	aluPAPE->aluBIn(muxBOut);
		aluPAPE->aluCtl(s_sopSel);	aluPAPE->aluOut(s_aluOut);
		aluPAPE->regIn(s_regIn);	aluPAPE->regTmp(s_regTmp);
		an Mertional States	
		SC_METHOD(do_latch);	
		sensitive_pos << clock;	
		sensitive_neg << reset;	
		SC METHOD/do reals	
		sensitive << clock << s reaSel << :	s ruBenFn << s ronin << dBook << dBooR:
		primitie a circu a p_1 cBoci aa	T and the state of
		SC_METHOD(do_sram);	
		sensitive << clock << s_rwSEn <<	s_sramEn << s_sramSel << s_regin << sramData;
		SC_METHOD(do_doutReg);	
		sensitive << clock << s_doutRCtl ·	<< doutBus;
#ifdef STM			
440CI 3111		dout Rus initializate):	
		dOut.initialize(0)+	
		doutadiPE initialize(0).	
		for (int i=0: i<16' i++) ramDatatil	="****
#endif		··· (m· ·-·, ···) (*··) (amaata[1]	
	}		
k	-		

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes

/* * PAPE: Processing Accelerator-PE for CAP(source file for PAPE) * Copyright(c) 2005 by Chul KIM, All right reserved * Author: Chul KIM(ckim@student.ecu.edu.au) • File name: pape.cpp * Revision history: Version1 * Date: 29/1/2005 47 #include "pape.h" // Latch void pape::do_latch() { if (reset) { s_inst.write(0); s_dIn.write(0); s_dLeft.write(0); s dRight.write(0); s_dUp.write(0); s dDown.write(0);] else { s_inst.write(instICS.read()); s_din.write(din.rcad()); //input data bus s_dLeft.write(dLeft.read()); s_dRight.write(dRight.read()); s_dUp.write(dUp.read()); s dDown.write(dDown.read()); ١ } // Internal Register vold pape::do_reg() { if (s_rwRegEn) { //read operation switch (s_regSel.read()) { case 0: regOut.write(tmp1); break; case 1: regOut.write(tmp2); break; case 2: regOut.write(tmp3); break: case 3; regOut.write(tmp4); break; default: break: if (dRegA) { //output control dRegOutA = regOut; } else { doutBus = sc_lv<4> (regOut); dOut = sc_uint<4> (doutBus); ş, If (dReaB) (dRegOutB = regOut; } else { doutBus = sc_lv<4> (regOut); dOut = sc_uint<4> (doutBus); ł } else if (s_rwRegEn==0) { //write operation switch (s_regSel.read()) { case 0: tmp1 = s_regln; break: case 1: tmp2 = s_regIn; break; case 2: tmp3 = s_regIn; break: case 3: tmp4 = s_regin; break; default: break: } ł ł // SRAM void pape::do_sram() { if (s_sramEn) { if (s_rwSEn) { //read operation sramData.write(ramData[s_sramStl.read()]); doutBus = sramData; dOut = sc_uint<4> (sramData); 11 dOut = sc_uint<4> (doutBus); //-- dOut has a dummy value(#F) } else (//write operation sramData= sc_lv<4> (s_regIn);

3D-SoftChip A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes

ramData[s_sramSel.read()] = sramData;

} else {
 sramData = "ZZZZ";
 }
}
// Data output register
void pape::do_doutReg() {
 if (s_doutRCt)) {
 dOutadjPE = sc_uint<4> (doutBus);
 }
}

3 ICS_RISC

3.1 Datapath Architecture

/•

* PC: Program Counter * for ICS(Intelligent Configurable Switch)RISC Core(header file for pc) * Copyright(c) 2005 by Chul KIM, All right reserved * Author: Chul KIM(ckim@student.ecu.edu.au) • File name: pc.h * Revision history: Version1 * Date: 23/3/2005 */ #include "systeme.h" SC_MODULE(pc) { sc_in<bool> clock; sc_in<bool> resct; iAregCtl; //Select Signal between aluOut/incrOut sc_in<bool> sc_in<bool> dAregCtl; aluOut; sc_in<sc_uint<32>> sc_out<sc_uint<32>> iAddr; //Instruction Address sc_out<sc_uint<32> > //Data Address dAddr; void do_pc(); void do_autoIncr(); iAddrTmp; sc_uint<32> sc_uint<32> incrOut; bool IAddrOut: sc_uint<32> iAddrln; SC_CTOR(pc) { SC_METHOD(do_pc); sensitive << clock.pos() << reset << iArcgCtl << dArcgCtl << aluOut; SC_METHOD(do_autoIncr); sensitive << clock.pos() << reset; #ifdef SIM LAddr.initialize(0); dAddr.initialize(0); #endif } ł÷ * PC: Program Counter * for ICS(Intelligent Configurable Switch)RISC Core(source file for pc) * Copyright(c) 2005 by Chul KIM, All right reserved * Author: Chul KIM(ckim@student.ecu.edu.au)

A DISSERTATION FOT THE DEGREE OF MASTER OF ENGINERING SCIENCE

3D-SoftChip A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes

* File name: pc.cpp

- * Revision history: Version1
- * Date: 23/3/2005
- +/

#include "pc.h"

```
void pc::do_pc() {
           bool
                       iAddrOutTmp:
           if (reset) {
                       iAddr = 0;
                       dAddr = 0:
           | else {
                       iAddrTmp = iAddrIn;
                      incrOut = iAddrTmp + 2;
           if (iAregCti) {
                       iAddr = aluOut;
                       iAddrOutTmp = 0;
           | else {
                       iAddr = incrOut:
                      iAddrOutTmp = 1;
           iAddrOut = iAddrOutTmp;
           If (dAregCtl) {
                       dAddr = aluOut;
           1
ł
void pc::do_autoIncr() {
           if (reset) {
                       iAddrIn = 0;
           } else if (iAddrOut) {
                      if (clock.posedge()) {
                                  iAddrIn++;
                       ł
           ł
ł
```

74 * SR: Status Register * for ICS(Intelligent Contigurable Switch)RISC Core(header file for sr) * Copyright(c) 2005 by Chul KIM, All right reserved * Author: Chul KIM(ckim@student.ecu.edu.au) * Flie name: sr.h * Revision history: Version1 • Date: 3/2/2005 */ #include "systemc.h" SC_MODULE(sr) { sc_in<bool> clock: sc_in<bool> reset; sc_in<sc_uint<4>> condFlag; wbData; sc_in<sc_uint<4>> sc_in<bool> wbSel; sc_in<bool> srOEn; sc_in<bool> srWbEn; sc_out<bool> zFlag; sc_out<sc_uint<4>> rdData; void do_sr(); sc_uint<4> srData;

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes

```
SC_CTOR(sr) {
                       SC_METHOD(do_sr);
                       sensitive << clock << reset << condFlag << wbData
                         << wbSel << srOEn << srWbEn;
#ifdef SIM
                       zFlag.initialize(0);
                       rdData.inltialize(0);
#endif
           1
ł:
/*
* SR: Status Register
 * for ICS(Intelligent Configurable Switch)RISC Core(source file for sr)
 * Copyright(c) 2005 by Chul KIM, All right reserved
 * Author: Chut KIM(ckim@student.ecu.edu.au)
* File name: sr.cpp
 • Revision history: Version1
 * Date: 3/2/2005
 +/
#include "sr.h"
void sr::do_sr() {
           if (reset) {
                       srData = 0;
           } else if (srWbEn) {
                       if (wbSel) {
                                  srData = condFlag;
                       } else {
                                  srData = wbData;
                       }
           ł
           zFlag = srData[2];
           if (srOEn) {
                       rdData = srData;
           ł
           rdData = srOEn ? sc_lv<4> (srData) : "ZZZZ";
II
ł
/*
 * LF: Loop Buffer
 * for ICS(Intelligent Configurable Switch)RISC Core(header file for If)
 * Copyright(c) 2005 by Chui KIM, All right reserved
 * Author: Chul KIM(ckim@student.ecu.cdu.au)
 • File name: If.b
 * Revision history: Version1
 * Date: 14/3/2005
 47
#include "systeme.h"
SC_MODULE(II) {
           sc_in<bool>
                                  clock;
           sc_in<bool>
                                  reset:
           sc_in<bool>
                                  lbEn;
                                              //Loop Buffer Enable
                                  IbRWEn; //Loop Buffer Read/Write Enable
           sc in<bool>
           sc_in<sc_uint<32> > iAddrIn;
                                             //iAddr Input for LF
           sc_out<sc_uint<32>> iAddrOut; //iAddr
sc_out<sc_uint<4>> incr;
                                                         Output for LF
11
```

void do_lf();

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes



* RegFile: 32 x 32 Register file

- * for ICS(Intelligent Configurable Switch)RISC Core(header file for regFile)
- * Copyright(c) 2005 by Chul KIM, All right reserved
- * Author: Chul KIM(ckim@student.ecu.edu.au)
- * File name: regFile.h
- Revision history: Version1
- * Date: 3/2/2005
- */

#include "systeme.h"

SC_MODULE(regFile) {

sc_in<booi> clock; sc_in<sc_uint<5> > rdAIdx; sc_in<sc_uint<5>> rdBldx; sc_in<bool> rdAOEn; sc_in<bool> rdBOEn; sc_ln<sc_uint<5>> wbldx; sc_in<sc_uint<32> > wbData; sc_in<bool> wbEn: sc_in<sc_uint<32>> pc;

//read index A //read index B //read A output enable //read B output enable //writeback index //writeback data //writeback enable

3D-SoftChip A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes

	sc_out <sc_ui< td=""><td>int<32> > rdAData;</td><td>//read data A</td><td></td><td></td></sc_ui<>	int<32> > rdAData;	//read data A		
	sc_out <sc_u< td=""><td>int<32> > rdBData;</td><td>//read data B</td><td></td><td></td></sc_u<>	int<32> > rdBData;	//read data B		
			and one and one one on	w7 and and	
	sc_signal <sc< td=""><td>_uun<52> >gpr0,gpr1 mr10 anr11 anr12 anr</td><td>,gpr2,gpr3,gpr4,gpr3,gpr0,gp 13.onr14.onr15.onr16.onr17.s</td><td>nr18.epr19.gor20.</td><td></td></sc<>	_uun<52> >gpr0,gpr1 mr10 anr11 anr12 anr	,gpr2,gpr3,gpr4,gpr3,gpr0,gp 13.onr14.onr15.onr16.onr17.s	nr18.epr19.gor20.	
	8	pr 10,8pr 11,8pr 12,8pr mr 21.epr 22.epr 23.epr	24.gor25.gor26.gor27.gor28.	gor29.gor30.gor31;	
	,	h		01	
	sc_signal <sc< td=""><td>_uint<32> ></td><td>rdADataTmp, rdBDataTmp</td><td>1</td><td></td></sc<>	_uint<32> >	rdADataTmp, rdBDataTmp	1	
	-				
	vold do_regi	file();			
	60 0TOD				
	SU_CIUR	C METHOD/do rod	File).		
		sensitive << clock.pos()	< rdA idx << rdBidx << rd	AOEn << rdBOEn << wbIdx	
			<< wbData << wbEn << p	c;	
#ifdef SIM	1		-		
	ſ	rdAData.initialize(0);			
	1	dBData.initlalize(0);			
#endif		1			
1.	1	1			
1i					
1*					
• RegFile	:: 32 x 32 Regi	ister file			
ICS * for ICS	(Intelligent C	onfigurable Switch)RI	SC Core(source file for regFil	le)	
* Copyrig	ght(c) 2005 by	Chul KIM, All right r	reserved		
* Author:	: Chul KIM(cl	kim@student.ecu.edu.	au)		
• File nar	me: regi ile cp	/p 			
* Revisio	n mstory: ver	SIUTE			
• Date: 5/	202003				
4					
#include "	regFile.h"				
void regFi	le::do_regFile	204			
	if (wbEn) {		_		
	1	switch (wbldx.read())		b l.	
		case 0: gpr	U.WINE(WDData);	orcax;	
		case 1: gpr	1.write(wbData);	break-	
		case 3: out	3.write(wbData);	break:	
		case 4; gpr	4.write(wbData);	break;	
		case 5: gpr	5.write(whData);	break;	
		case 6: gpr	6.write(whData);	break;	
		case 7: gpr	7.write(wbData);	hreak;	
		case 8: gpr	8.write(wbData);	break;	
		cese 9: gpr	9.write(wbData);	Break; broak;	
		case 10: gp	n rowrite(wobata); witeweite(whData);	break:	
		case 12: gp	rl2.write(wbData):	break;	
		case 13: 20	w13.write(wbData);	break;	
		case 14: gr	or14.write(wbData);	break;	
		case 15: gr	or15.write(wbData);	break;	
		case 16: gg	or16.write(wbData);	break;	
		case 17: gr	or17.write(wbData);	break;	
		case 18: gr	r18.write(wbData);	break;	
		case 19: gp	or 17. Write(WDData); pr70. genite(wbData);	urtak; break•	
		case 20: gj	wite(whData);	break:	
		Case 22: ar	x22.write(whData):	break:	
		case 23: gr	or23.write(wbData);	break;	
		case 24: pr	or24.write(wbData);	break;	
		case 25: gr	pr25.write(wbData);	break;	
		case 26: gr	pr26.write(wbData);	break;	
		case 27: gj	pr27.write(wbData);	break;	
		case 28: gj	pr28.write(wbData);	break;	
		case 29: gj	pr29.write(WDData); ar10.write(wbData);	orcak; break:	
		case 50: gi	prov.Write(wovata); ar31 write(aa);	orvak; bregk://for PC	
		LASC 31; E	********	strong state	

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes

	octanti:	Dreak;
}		
}		
if (rdAOEn) {		
switch	(rdAldx.read()) {	
	case 0: rdAData = gpr0;	break;
	case 1: rdAData = gpr1;	break;
	case 2: rdAData = gpr2;	break;
	case 3: rdAData = gpr3;	break;
	case 4: rdAData = gpr4;	break;
	case 5: rdAData = gpr5;	Dreak;
	case o: realizata = gpro;	Dreak;
	case 7: rua Data = $gpr/;$	Urvak;
	case 9: rdAData = gpro;	break:
	case 10: rdAData = gpr);	break:
	case 11: rdAData = gpr10;	break:
	case 12: rdAData = $gpr12$:	break:
	case 13: rdAData = gpr13;	break:
	case 14: rdAData = gpr14;	break:
	case 15: rdAData = gpr15;	break;
	case 16: rdAData = gpr16;	break;
	case 17: rdAData = gpr17;	break;
	case 18: rdAData = gpr18;	break;
	case 19: rdAData = gpr19;	break;
	case 20: rdAData = gpr20;	break;
	case 21: rdAData = gpr21;	break;
	case 22: rdAData = gpr22;	break;
	case 23: rdAData = gpr23;	break;
	case 24: rdAData = gpr24;	break;
	case 25: rdA Data = gpr25;	break;
	case 26: rdAData = gpr26;	break;
	case 27: $rdAData = gpr27;$	break;
	case 20; roA Data = gpr20;	oreak;
	case 29; $ruA Data = gpr29;$	Urcak;
	case 31; rdA Data = per	break,
	case 31: rdAData = pc;	break; break; //for PC
١	case 31: rdAData = pc; default:	break; //for PC break;
,)	case 31: rdAData = pc; default:	break; //for PC break;
} } if (rdBOEn) {	case 31: rdAData = pc; default:	break; //for PC break;
} } if (rdBOEn) { switch	case 31: rdAData = pr; default:	break; //for PC break;
} } if (rdBOEn) { switch	(rdBldx.rcad()) { case 0: rdBData = pr; default: (rdBldx.rcad()) {	break; //for PC break;
} if (rdBOEn) { switch	<pre>case 30: rdAData = pp:00, case 31: rdAData = pc; default: (rdBldx.rcad()) { case 0: rdBData = gpr0; case 1: rdBData = por1;</pre>	break; //for PC break; break;
} if (rdBOEn) { switch	<pre>case 30: rdAData = pp:00, case 31: rdAData = pc; default: (rdBldx.read()) { case 0: rdBData = gpr0; case 1: rdBData = gpr1; case 2: rdBData = gpr2;</pre>	break; //for PC break; break; break; break;
} if (rdBOEn) { switch	<pre>(rdBldx.rcad()) { case 31: rdAData = pc; default: (rdBldx.rcad()) { case 0: rdBData = gpr0; case 1: rdBData = gpr1; case 2: rdBData = gpr2; case 3: rdBData = gpr3;</pre>	break; //for PC break; break; break; break; break; break;
} if (rdBOEn) { switch	<pre>(rdBldx.rcad()) { case 31: rdAData = pc; default: (rdBldx.rcad()) { case 0: rdBData = gpr0; case 1: rdBData = gpr1; case 2: rdBData = gpr2; case 3: rdBData = gpr3; case 4: rd2Data = gpr4; } }</pre>	break; //for PC break; break; break; break; break; break; break;
} if (rdBOEn) { switch	<pre>(rdBldx.rcad()) { case 31: rdAData = pc; default: (rdBldx.rcad()) { case 0: rdBData = gpr0; case 1: rdBData = gpr1; case 2: rdBData = gpr2; case 3: rdBData = gpr3; case 4: rd2Data = gpr4; case 5: rdBData = gpr5;</pre>	break; break; break; break; break; break; break; break; break;
} if (rdBOEn) { switch	<pre>(rdBldx.rcad()) { case 31: rdAData = pc; default: (rdBldx.rcad()) { case 0: rdBData = gpr0; case 1: rdBData = gpr1; case 2: rdBData = gpr3; case 3: rdBData = gpr4; case 5: rdBLata = gpr5; case 6: rdBData = gpr6;</pre>	break; break; break; break; break; break; break; break; break; break;
} if (rdBOEn) { switch	<pre>(rdBldx.rcad()) { case 31: rdAData = gpr0; default: (rdBldx.rcad()) { case 0: rdBData = gpr0; case 1: rdBData = gpr1; case 2: rdBData = gpr2; case 3: rdBData = gpr3; case 4: rd2Data = gpr4; case 5: rdBLata = gpr6; case 6: rdBData = gpr7; }; }</pre>	break; //for PC break; break; break; break; break; break; break; break; break; break;
} if (rdBOEn) { switch	<pre>(rdBldx.rcad()) { case 31: rdAData = gpr0; default: (rdBldx.rcad()) { case 0: rdBData = gpr0; case 1: rdBData = gpr1; case 2: rdBData = gpr2; case 3: rdBData = gpr4; case 5: rdBData = gpr5; case 6: rdBData = gpr7; case 6: rdBData = gpr7; case 8: rdBData = gpr8; } </pre>	break; break; break; break; break; break; break; break; break; break; break; break;
} if (rdBOEn) { switch	<pre>(rdBldx.rcad()) { case 31: rdAData = gpr0; default: (rdBldx.rcad()) { case 0: rdBData = gpr0; case 1: rdBData = gpr1; case 2: rdBData = gpr2; case 3: rdBData = gpr3; case 4: rd2Data = gpr5; case 5: rdBData = gpr5; case 6: rdBData = gpr7; case 8: rdBData = gpr8; case 9: rdBData = gpr9; } }</pre>	break; break; break; break; break; break; break; break; break; break; break; break; break; break;
} if (rdBOEn) { switch	<pre>(rdBldx.rcad()) { case 31: rdAData = pc; default: (rdBldx.rcad()) { case 0: rdBData = gpr0; case 1: rdBData = gpr1; case 2: rdBData = gpr2; case 3: rdBData = gpr3; case 4: rd2Data = gpr4; case 5: rdBData = gpr5; case 5: rdBData = gpr6; case 7: rdBData = gpr6; case 7: rdBData = gpr9; case 10: rdBData = gpr10; } }</pre>	break; //for PC break; break; break; break; break; break; break; break; break; break; break; break; break; break;
} if (rdBOEn) { switch	<pre>(rdBldx.rcad()) { case 31: rdAData = pc; default: (rdBldx.rcad()) { case 0: rdBData = gpr0; case 1: rdBData = gpr1; case 2: rdBData = gpr2; case 3: rdBData = gpr3; case 4: rd2Data = gpr4; case 5: rdBData = gpr6; case 6: rdBData = gpr6; case 7: rdBData = gpr6; case 9: rdBData = gpr9; case 10: rdBData = gpr10; case 11: rdBData = gpr11; } }</pre>	break; break; break; break; break; break; break; break; break; break; break; break; break; break; break; break; break;
} if (rdBOEn) { switch	<pre>(rdBldx.rcad()) { case 31: rdAData = pc; default: (rdBldx.rcad()) { case 0: rdBData = gpr0; case 1: rdBData = gpr1; case 2: rdBData = gpr2; case 3: rdBData = gpr3; case 4: rd2Data = gpr4; case 5: rdBData = gpr5; case 6: rdBData = gpr6; case 7: rdBData = gpr6; case 9: rdBData = gpr9; case 10: rdBData = gpr10; case 11: rdBData = gpr11; case 12: rdBData = gpr12; } }</pre>	break; //for PC break; break; break; break; break; break; break; break; break; break; break; break; break; break; break; break;
} if (rdBOEn) { switch	<pre>(rdBldx.rcad()) { case 31: rdAData = pc; default: (rdBldx.rcad()) { case 0: rdBData = gpr0; case 1: rdBData = gpr2; case 2: rdBData = gpr2; case 2: rdBData = gpr3; case 3: rdBData = gpr4; case 5: rdBData = gpr5; case 6: rdBData = gpr6; case 7: rdBData = gpr7; case 8: rdBData = gpr7; case 10: rdBData = gpr9; case 10: rdBData = gpr10; case 11: rdBData = gpr10; case 12: rdBData = gpr13; case 13: rdBData = gpr13; case 14: rdBData = gpr14; case 14: rdBData = gpr44; case 14: rdBData = gpr44;</pre>	break; //for PC break; break; break; break; break; break; break; break; break; break; break; break; break; break; break; break;
} if (rdBOEn) { switch	<pre>(rdBldx.rcad()) { case 31: rdAData = pc; default: (rdBldx.rcad()) { case 0: rdBData = gpr0; case 1: rdBData = gpr2; case 2: rdBData = gpr2; case 2: rdBData = gpr3; case 3: rdBData = gpr4; case 5: rdBData = gpr5; case 6: rdBData = gpr6; case 7: rdBData = gpr7; case 8: rdBData = gpr7; case 9: rdBData = gpr7; case 10: rdBData = gpr10; case 11: rdBData = gpr10; case 12: rdBData = gpr13; case 14: rdBData = gpr14; case 15: rdBData = gpr14; case 14: rdBData = gpr14; c</pre>	break; //for PC break; break; break; break; break; break; break; break; break; break; break; break; break; break; break; break; break; break;
} if (rdBOEn) { switch	<pre>(rdBldx.rcad()) { case 31: rdAData = pc; default: (rdBldx.rcad()) { case 0: rdBData = gpr0; case 1: rdBData = gpr1; case 2: rdBData = gpr2; case 2: rdBData = gpr3; case 4: rd2Data = gpr4; case 5: rdBData = gpr6; case 7: rdBData = gpr6; case 7: rdBData = gpr7; case 8: rdBData = gpr6; case 9: rdBData = gpr9; case 10: rdBData = gpr10; case 11: rdBData = gpr11; case 12: rdBData = gpr13; case 14: rdBData = gpr14; case 15: rdBData = gpr15; case 16: rdBData = gpr14; case 16: rdBData = gpr14; case 16: rdBData = gpr15; case 16: rdBData = gpr14; case 16: rdBData = gpr15; c</pre>	break; break;
} if (rdBOEn) { switch	<pre>(rdBldx.rcad()) { case 31: rdAData = pc; default: (rdBldx.rcad()) { case 0: rdBData = gpr0; case 1: rdBData = gpr1; case 2: rdBData = gpr2; case 3: rdBData = gpr3; case 4: rd2Data = gpr4; case 5: rdBData = gpr6; case 7: rdBData = gpr7; case 8: rdBData = gpr6; case 9: rdBData = gpr7; case 8: rdBData = gpr7; case 10: rdBData = gpr1; case 11: rdBData = gpr1; case 12: rdBData = gpr1; case 13: rdBData = gpr1; case 11: rdBData = gpr1; case 12: rdBData = gpr1; case 13: rdBData = gpr1; case 14: rdBData = gpr1; case 15: rdBData = gpr1; case 16: rdBData = gpr1; case 16: rdBData = gpr1; case 16: rdBData = gpr16; case 17: rdBData = gpr15; case 16: rdBData = gpr16; case 17: rdBData = gpr16; case 11: rdBData = gpr14; case 12: rdBData = gpr14; case 13: rdBData = gpr14; case 15: rdBData = gpr15; case 16: rdBData = gpr15; case 16: rdBData = gpr16; case 17: rdBData = gpr16; case 16: rdBData = gpr16; case 17: rdBData = gpr16; case 11: rdBData = gpr15; case 11: rdBData = gpr14; case 11: rdBData = gpr14; case 11: rdBData = gpr14; case 11: rdBData = gpr15; case 11: rdBData = gpr15; case 11: rdBData = gpr14; case 11: rdBData = gpr15; case 11: rdB</pre>	break; break;
} if (rdBOEn) { switch	(rdBldx.rcad()) { case 31: rdAData = pc; default: (rdBldx.rcad()) { case 0: rdBData = gpr0; case 1: rdBData = gpr1; case 2: rdBData = gpr2; case 3: rdBData = gpr3; case 4: rd2Data = gpr3; case 5: rdBData = gpr6; case 5: rdBData = gpr7; case 8: rdBData = gpr6; case 9: rdBData = gpr1; case 10: rdBData = gpr10; case 11: rdBData = gpr11; case 12: rdBData = gpr12; case 13: rdBData = gpr12; case 14: rdBData = gpr14; case 15: rdBData = gpr16; case 17: rdBData = gpr16; case 17: rdBData = gpr17; case 18: rdBData = gpr16; case 17: rdBData = gpr17; case 18: rdBData = gpr16; case 17: rdBData = gpr17; case 19: rdBData = gpr16; case 17: rdBData = gpr17; case 19: rdBData = gpr16; case 19: rdBData = gp	break; break;
} if (rdBOEn) { switch	(rdBldx.rcad()) { case 31: rdAData = pc; default: (rdBldx.rcad()) { case 0: rdBData = gpr0; case 1: rdBData = gpr1; case 2: rdBData = gpr2; case 3: rdBData = gpr3; case 4: rd2Data = gpr4; case 5: rdBData = gpr6; case 5: rdBData = gpr6; case 7: rdBData = gpr7; case 8: rdBData = gpr9; case 10: rdBData = gpr10; case 11: rdBData = gpr10; case 12: rdBData = gpr14; case 13: rdBData = gpr14; case 14: rdBData = gpr14; case 15: rdBData = gpr14; case 15: rdBData = gpr14; case 15: rdBData = gpr16; case 17: rdBData = gpr16; case 17: rdBData = gpr16; case 18: rdBData = gpr16; case 19: rdBData = gpr16; case 10: rdBData = gp	break; //for PC break;
} if (rdBOEn) { switch	(rdBldx.rcad()) { case 31: rdAData = pc; default: (rdBldx.rcad()) { case 0: rdBData = gpr0; case 1: rdBData = gpr1; case 2: rdBData = gpr2; case 3: rdBData = gpr3; case 4: rd2Data = gpr4; case 5: rdBData = gpr6; case 5: rdBData = gpr6; case 6: rdBData = gpr9; case 10: rdBData = gpr19; case 11: rdBData = gpr10; case 12: rdBData = gpr14; case 13: rdBData = gpr14; case 13: rdBData = gpr14; case 15: rdBData = gpr14; case 15: rdBData = gpr14; case 16: rdBData = gpr15; case 16: rdBData = gpr16; case 17: rdBData = gpr18; case 19: rdBData = gpr19; case 19: rdBData =	break; //for PC break;
} if (rdBOEn) { switch	(rdBldx.rcad()) { case 31: rdAData = pc; default: (rdBldx.rcad()) { case 0: rdBData = gpr0; case 1: rdBData = gpr1; case 2: rdBData = gpr2; case 3: rdBData = gpr3; case 4: rd2Data = gpr4; case 5: rdBData = gpr6; case 5: rdBData = gpr6; case 6: rdBData = gpr7; case 8: rdBData = gpr1; case 10: rdBData = gpr1; case 11: rdBData = gpr1; case 12: rdBData = gpr1; case 13: rdBData = gpr15; case 14: rdBData = gpr14; case 15: rdBData = gpr16; case 16: rdBData = gpr16; case 16: rdBData = gpr16; case 16: rdBData = gpr16; case 16: rdBData = gpr16; case 17: rdBData = gpr18; case 19: rdBData = gpr19; case 20: rdBData = gpr20; case 21: rdBData = gpr20	break; //for PC break;
} if (rdBOEn) { switch	<pre>(rdBldx.rcad()) { case 31: rdAData = pc; default: (rdBldx.rcad()) { case 0: rdBData = gpr0; case 1: rdBData = gpr1; case 2: rdBData = gpr2; case 3: rdBData = gpr3; case 3: rdBData = gpr4; case 5: rdBData = gpr6; case 6: rdBData = gpr6; case 7: rdBData = gpr6; case 9: rdBData = gpr6; case 9: rdBData = gpr1; case 10: rdBData = gpr1; case 11: rdBData = gpr1; case 12: rdBData = gpr1; case 13: rdBData = gpr1; case 14: rdBData = gpr16; case 16: rdBData = gpr16; case 17: rdBData = gpr16; case 19: rdBData = gpr17; case 18: rdBData = gpr16; case 19: rdBData = gpr16; case 12: rdBData = gpr17; case 13: rdBData = gpr16; case 12: rdBData = gpr16; case 12: rdBData = gpr18; case 20: rdBData = gpr20; case 21: rdBData = gpr21; case 21: rdBData = gpr21; case 22: rdBData = gpr21; case 23: rdBData = gpr21; case 24: rdBData = gpr20; case</pre>	break; //for PC break;
} if (rdBOEn) { switch	<pre>(rdBldx.read()) { case 31: rdAData = pc; default: (rdBldx.read()) { case 0: rdBData = gpr0; case 1: rdBData = gpr1; case 2: rdBData = gpr2; case 3: rdBData = gpr3; case 4: rd2Data = gpr4; case 5: rdBData = gpr5; case 6: rdBData = gpr6; case 7: rdBData = gpr6; case 7: rdBData = gpr6; case 9: rdBData = gpr6; case 10: rdBData = gpr10; case 11: rdBData = gpr10; case 12: rdBData = gpr12; case 13: rdBData = gpr13; case 14: rdBData = gpr14; case 15: rdBData = gpr16; case 17: rdBData = gpr16; case 19: rdBData = gpr17; case 18: rdBData = gpr18; case 19: rdBData = gpr21; case 20: rdBData = gpr21; case 21: rdBData = gpr22; case 21: rdBData = gpr22; case 21: rdBData = gpr24; case 23: rdBData = gpr24; c</pre>	break; //for PC break;
} if (rdBOEn) { switch	<pre>(rdBldx.read()) { case 31: rdAData = pc; default: (rdBldx.read()) { case 0: rdBData = gpr0; case 1: rdBData = gpr1; case 2: rdBData = gpr2; case 3: rdBData = gpr3; case 4: rd2Data = gpr4; case 5: rdBData = gpr5; case 6: rdBData = gpr6; case 7: rdBData = gpr7; case 8: rdBData = gpr7; case 8: rdBData = gpr7; case 8: rdBData = gpr7; case 7: rdBData = gpr7; case 8: rdBData = gpr7; case 10: rdBData = gpr10; case 11: rdBData = gpr10; case 12: rdBData = gpr10; case 13: rdBData = gpr13; case 14: rdBData = gpr14; case 15: rdBData = gpr16; case 17: rdBData = gpr16; case 19: rdBData = gpr18; case 19: rdBData = gpr20; case 20: rdBData = gpr21; case 21: rdBData = gpr22; case 21: rdBData = gpr23; case 24: rdBData = gpr24; case 24</pre>	break; break;
} if (rdBOEn) { switch	<pre>(rdBldx.read()) { case 31: rdAData = pc; default: (rdBldx.read()) { case 0: rdBData = gpr0; case 1: rdBData = gpr1; case 2: rdBData = gpr2; case 2: rdBData = gpr3; case 4: rd2Data = gpr3; case 5: rdBData = gpr3; case 6: rdBData = gpr6; case 7: rdBData = gpr7; case 8: rdBData = gpr7; case 8: rdBData = gpr7; case 6: rdBData = gpr6; case 7: rdBData = gpr7; case 8: rdBData = gpr7; case 10: rdBData = gpr10; case 11: rdBData = gpr10; case 12: rdBData = gpr12; case 13: rdBData = gpr13; case 14: rdBData = gpr14; case 15: rdBData = gpr15; case 18: rdBData = gpr16; case 19: rdBData = gpr17; case 18: rdBData = gpr18; case 19: rdBData = gpr20; case 21: rdBData = gpr21; case 22: rdBData = gpr24; case 23: rdBData = gpr24; case 25: rdBData = gpr25; case 25: rdBData = gpr24; case 25: rdBData = gpr25; case 25: rdBData = gpr24; case 25: rdBData = gpr24; case 25: rdBData = gpr24; case 25: rdBData = gpr25; case 25: rdBData = gpr24; case 25: rdBData = gpr24; case 25: rdBData = gpr24; case 25: rdBData = gpr24;</pre>	break; break;
} if (rdBOEn) { switch	<pre>(rdBldx.rcad()) { case 31: rdAData = gpr0; default: (rdBldx.rcad()) { case 0: rdBData = gpr0; case 1: rdBData = gpr1; case 2: rdBData = gpr2; case 2: rdBData = gpr3; case 3: rdBData = gpr3; case 4: rd2Data = gpr3; case 5: rdBData = gpr3; case 6: rdBData = gpr6; case 7: rdBData = gpr7; case 8: rdBData = gpr9; case 10: rdBData = gpr1; case 10: rdBData = gpr1; case 11: rdBData = gpr10; case 12: rdBData = gpr12; case 13: rdBData = gpr14; case 14: rdBData = gpr14; case 15: rdBData = gpr15; case 16: rdBData = gpr14; case 17: rdBData = gpr14; case 18: rdBData = gpr18; case 19: rdBData = gpr19; case 20: rdBData = gpr20; case 21: rdBData = gpr21; case 22: rdBData = gpr23; case 23: rdBData = gpr25; case 24: rdBData = gpr25; case 26: rdBData = gpr25; c</pre>	break; //for PC break;

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes

		case 28: rdBData = gpr2 case 29: rdBData = gpr2 case 30: rdBData = gpr3 case 31: rdBData = pc; default:	28; break; 19; break; 10; break; break; break; break;	//for PC
}	}			
/* * aluDo * Copy * Auth * File r	ef: Definition of the ALU right(c) 2005 by Chal Kl or: Chal KlM(ckim@stu tame: aluDef.h	functions M, All right reserved dent.ccu.edu.su)		
* Revis * Date: */	: 2/2/2005			
#ifndef #define	_ALU_DEFINE_H_ _ALU_DEFINE_H_			
// ALU I #define	Function Definitions CMD_MOVA 0x0			
#define	CMD_MOVB 0x1			
#define	CMD_AND	0x2		
#deline	CMD_UK CMD_XOP	0x3 6x4		
#define	CMD_NOT	0x5		
#define	CMD_ADD	0x6		
#definc	CMD_SUB	0x7		
#define	CMD_CMP	0x8		
#endif				
/* * alul(* Copy	CS: ALU for ICS(Intellig yright(c) 2005 by Chul K	ent Configurable Switch)R IM, All right reserved	USC Core(header file for aluIC	'S)
• Auth	ior: Chui KIM(ckim@stu name: aluICS/h sion history: Version1	ident.ecu.edu.au)		
 File Revi Date 	: 2/2/2005			
* File * Revi * Date */	: 2/2/2005			
 File (Revi Date */ #includ #includ 	: 2/2/2005 e ''systemc.h'' e ''aluDef.h''			
 File (Revi Date */ #includ \$C_MO 	: 2/2/2005 e ''systemc.h'' c ''aluDef.b'' DDULE(alu1CS) {			
* File (* Revi * Date */ #includ #includ SC_MC	: 2/2/2005 e ''systemc.h'' c ''aluDef.b'' DDULE(aluICS) { sc_in <sc_uint<32> > sc_in<sc_uint<32> ></sc_uint<32></sc_uint<32>	aluAln; aluBln:		
* File * Revi * Date */ #includ SC_MO	: 2/2/2005 e ''systemc.h'' e ''aluDef.h'')DULE(aluICS) { sc_in <sc_uint<32> > sc_in<sc_uint<32> > sc_in<sc_uint<32> ></sc_uint<32></sc_uint<32></sc_uint<32>	aluAIn; aluBIo; aluCtl;		
* File * Revi * Date */ #includ #includ SC_MO	: 2/2/2005 e "systenic.h" e "aluDef.h" DDULE(alu1CS) { sc_in <sc_uint<32> > sc_in<sc_uint<32> > sc_in<sc_uint<32> > sc_in<sc_uint<32> ></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32>	aluAIn; aluBIn; aïuCti; cIn;	//carry input	
* File * Revi * Date */ #includ SC_MC	: 2/2/2005 e ''systemc.h'' e ''aluDef.h'')DULE(aluICS) { sc_in <sc_uint<32> > sc_in<sc_uint<32> > sc_in<sc_uint<32> > sc_in<bool> sc_out<sc_uint<32> > sc_out<sc_uint<32> ></sc_uint<32></sc_uint<32></bool></sc_uint<32></sc_uint<32></sc_uint<32>	aluAIn; aluBIn; aluCtl; cIn; > aluOut; condFlag;	//carry input //conditional flags	
* File * Revi * Date */ #includ #includ SC_MC	: 2/2/2005 e ''systemc.h'' e ''aluDef.h'')DULE(aluICS) { sc_in <sc_uint<32> > sc_in<sc_uint<32> > sc_in<sc_uint<4> > sc_in<bool> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<4> ></sc_uint<4></sc_uint<32></sc_uint<32></bool></sc_uint<4></sc_uint<32></sc_uint<32>	aluAIn; aluBIn; aluCt; cIn; > aluOut; condFlag;	//carry input //conditional flags	
* File * Revi * Date */ #includ SC_MC	: 2/2/2005 e ''systemc.h'' e ''aluDef.h'')DULE(aluICS) { sc_in <sc_uint<32> > sc_in<sc_uint<32> > sc_in<sc_uint<32> > sc_in<bool> sc_out<sc_uint<32> sc_out<sc_uint<4> > sc_out<sc_uint<4> > sc_out<sc_uint<4> > sc_out<sc_uint<4> ></sc_uint<4></sc_uint<4></sc_uint<4></sc_uint<4></sc_uint<32></bool></sc_uint<32></sc_uint<32></sc_uint<32>	aluAIn; aluBIn; aluCtl; cIn; > aluOut; condFlag; cf, vf, nf, zf;	//carry input //conditional flags	
* File * Revi * Date */ #includ SC_MC	: 2/2/2005 e "systemc.h" e "aluDef.h" DULE(alu1CS) { sc_in <sc_uint<32> > sc_in<sc_uint<32> > sc_in<sc_uint<32> > sc_in<bool> sc_out<sc_uint<4> > void do_alu(); sc_signal<bool> SC_CTOR(alu1CS) SC_ME</bool></sc_uint<4></bool></sc_uint<32></sc_uint<32></sc_uint<32>	<pre>aluAIn; aluBIn; aluCtl; cIn; aluOut; condFlag; (FHOD(do_alu);</pre>	//carry input //conditional flags	
• File + • Revi • Date */ #includ #includ SC_MC // // #ifdet 5	: 2/2/2005 e "systemc.h" e "aluDef.h" DDULE(alu1CS) { sc_in <sc_uint<32>> sc_in<sc_uint<32>> sc_in<sc_uint<32>> sc_in sc_uint<32> sc_out<sc_uint<4>> void do_alu(); sc_signal<bool> SC_CTOR(alu1CS) SC_ME sensitive</bool></br></sc_uint<4></sc_uint<32></sc_uint<32></sc_uint<32>	aluA In; aluBIn; aluCtl; cIn; > aluOut; condFlag; cf, vf, nf, zf; [[[[[[HOD(do_alu); << aluBIn << a	//carry input //conditional flags :	·
• File + • Revi • Date */ #includ #includ SC_MC // #ifdef S	: 2/2/2005 e "systemc.h" e "aluDef.h" DDULE(alu1CS) { sc_in <sc_uint<32> > sc_in<sc_uint<32> > sc_in<sc_uint<32> > sc_in sc_uint<32> > sc_out<sc_uint<32> > sc_out<sc_uint<32> void do_alu(); sc_signal<bool> SC_CTOR(alu1CS) SC_ME sensitive SIM aluOut.i</bool></br></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32>	<pre>aluAIn; aluBIn; aluCtl; cIn; aluOut; condFlag; fHOD(do_alu); <<< aluBIn << aluBI</pre>	//carry input //conditional flags ; aluCti;	
* File + * Revi * Date */ #includ SC_MC // // #ifdef S	: 2/2/2005 e "systemc.h" e "aluDef.h" DDULE(alu1CS) { sc_in <sc_uint<32> > sc_in<sc_uint<32> > sc_in<sc_uint<32> > sc_in sc_uint<32> > sc_in sc_uint<32> > sc_out<sc_uint<32> > sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> sc_out<sc_uint<32> s</sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></sc_uint<32></br></sc_uint<32></sc_uint<32></br></sc_uint<32></sc_uint<32></sc_uint<32>	<pre> aluAIn; aluBIn; aluCtl; cIn; cIn; aluOut; condFlag; fHOD(do_alu); <<< aluBIn << aluBIn << an itialize(0); g.inltialize(0); </pre>	//carry input //conditional flags	· · · · · · · · · · · · · · · · · · ·

A Novel 3D Vertically Integrated Adaptive Computing System

Appendiv C-SystemC Codes

1.	}			
0				
/* * aluICS * Copyri, * Author * File nai * Revisio * Date: 2 */	: ALU for ICS(Intelligent Configura ght(c) 2005 by Chul KIM, All right : Chul KIM(ckim@student.ecu.edu me: aluICS.cpp n history: Version1 /2/2005	able Switch)RISC Core reserved .au)	(source file l	for aluICS)
#include "	aluICS.h"			
#define co	mp(a,b) (((c)>(b))?1: (((a)==(b))?0:	-1))		
// ALU void aluIC	:S::do_alu() {			
Helse	signe i short result = 0;			
#endif	signed short result;			
//	<pre>signet' short src1 = aluAin.read(); signed short src2 = aluBin.read(); signed short tcin = cin.read(); signed short cmd = aluCtl.read()</pre>);		
	sc_uint<4> tmpCond;			
	<pre>switch (end & 0xF) { case 0: result = src1; case 1: result = src2; case 2: result = src1 & case 3: result = src1 case 4: result = src1; case 5: result = src1; case 7: result = src1 + case 8: result = src1; scase 8: result = src1; scase 8: result = src1; scase 8: result = src1; case 8: result = src1; scase 8: result = src1;</pre>	src2; rc2; src2; src2; src2; src1,src2);	break; break; break; break; break; break; break; break; break;	//CMD_MOVA //CMD_MOVB //CMD_AND //CMD_OR //CMD_XOR //CMD_NOT //CMD_ADD //CMD_SUB //CMD_CMP
// Conditio	nal Flags			
	if (result & 0xFFFF0000) else cf.write(0):	cf.write(1); //carry/Bo	rrow flag	
	if (result & 0xFFFF0000) clse vf.write(0); result &= 0xFFFF:	vf.write(1); //overflow	flag	
	if (result === 0) else zf.write(0):	zf.write(1); //zero flag		
	if (result & 0x8000) else nf.write(0);	nf.write(1);//negative	flag	
	aluOut.write(result);			
	tmpCond[3]=nf; tmpCond[2]=zf; tmpCond[1]=cf; tmpCond[0]=vf; condFlag=tmpCond;			
ł				

/*
 * MUL: multiplier
 * for ICS(Intelligent Configurable Switch)RISC Core(header file for multiplier)

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes

```
* Copyright(c) 2005 by Chui KIM, All right reserved
 * Author: Chul KIM(ckim@student.ccu.cdu.au)
 • File name: mul.h
 * Revision history: Version1
 * Date: 14/3/2005
 */
#include "systemc.h"
SC_MODULE(mul) {
           sc_in<bool>
                                             clock:
           sc_in<bool>
                                             resel;
           sc_in<sc_uint<32>>
                                             mulAIn;
                                             mulB1n;
           sc_in<sc_uint<32> >
           sc_out<sc_uint<32>>
                                             mulOut;
           void do_mul();
           SC_CTOR(mul) {
                      SC_METHOD(do_mul);
                      sensitive << clock << reset << mulAIn << mulBIn;
#ifdef SIM
                      mulOut.initialize(0);
#endif
           }
);
/*
 * MUL: multiplier
 * for ICS(Intelligent Configurable Switch)RISC Core(source file for multiplier)
 * Copyright(c) 2005 by Chul KIM, All right reserved
 * Author: Chul KIM(ckim@student.ecu.cdu.au)
 * File name: mul.cpp
 * Revision history: Version1
 * Date: 14/3/2005
 •7
#include "mul.h"
void mul::do_mul() {
           sc_uint<32> src1, src2, result;
           srci = mulAIn;
           src2 = mulBIn;
           result = src1 * src2;
           mulOut.write(result);
ł
/*
 * Shifter: Shifter for ICS(Intelligent Configurable Switch)RISC Core(beader file for Shifter)
 * Copyright(c) 2005 by Chul KIM, All right reserved
 * Author: Chul KIM(ckim@student.ecu.edu.au)
 * File name: shifter.h
 • Revision history: Version1
 * Date: 2/2/2005
 */
#include "systeme.h"
SC_MODULE(shifter) {
           sc_in<sc_uint<32> >
                                                                   //shifter input
                                             shiftIn;
```

shiftAmt;
shiftCtl;

shiftOut;

//shifter amount

//control input

//shifter output

sc_in<sc_uint<5>>

sc_in<sc_ulnt<3> >
sc_out<sc_ulnt<32> >

3D-SoftChip A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes

```
void do_shift();
           SC CTOR(shifter) {
                       SC_METHOD(do_shift);
                       sensitive << shiftIn << shiftAmt << shiftCtl;
#ifdef SIM
                       shiftOut.initialize(0);
#endif
            }
Ŀ
14
 * Shifter: Shifter for ICS(Intelligent Configurable Switch)RISC Core(source file for Shifter)
 * Copyright(c) 2005 by Chul KIM, All right reserved
 * Author: Chul KIM(ckim@student.ecu.edu.au)
 * File name: shifter.cpp
 * Revision history: Version1
 * Date: 2/2/2005
 •/
#include "shifter.h"
void shifter::do_shift() {
           sc_uint<32> w_shiftIn,w_shiftOut;
            sc_uint<5> w_shiftAmt;
            sc_uint<3> w_shiftCtl;
            w_shiftIn=shiftIn;
            w shiftAmt=shiftAmt:
            w_shiftCtl=shiftCtl;
           switch (w_shiftCtl) {
                       case 0: w_shiftOut = w_shiftIn << w_shiftAmt; break;</pre>
                                                                                //logical shift left
                       case 1: w_shiftOut = w_shiftIn >> w_shiftAmt; break; //logical shift right
//
                       case 2: w_shiftOut = ({32{w_shiftIn[31]}}<<(32.w_shiftAmt))|(w_shiftIn>>w_shiftAmt); break;
11
                       Arithmetic Shift Right should be modified.
11
                       case 2: w_shiftOut = ({w_shiftIn[32]{w_shiftIn[31]}}<<(32-w_shiftAmt))(w_shiftIn>>w_shiftAmt);
break:
                       case 2: w_shiftOut = w_shiftIn >> w_shiftAnt; break;
                       case 3: w_shiftOut = (w_shiftIn >> w_shiftAmt) | (w_shiftIn << (32-w_shiftAmt)); break;
                                  //Rotate
                       default:
                        break:
           shiftOut.write(w_shiftOut);
ł
```

1*

- * Datapath: Data-path architecture
- * for ICS(Intelligent Configurable Switch)RISC Core(header file for datapath)
- * Copyright(c) 2005 by Chul KIM, All right reserved
- * Author: Chul KIM(ckim@student.ecu.edu.au)
- * File name: datapath.h
- * Revision history: Version1
- * Date: 30/4/2005
- •/

#include "systemc.h"
#include "pc.h"
#include "sr.h"
#include "If.h"
#include "lf.h"
#include "aluDef.h"
#include "aluDef.h"
#include "aluICS.h"
#include "shifter.h"
#include "mul.h"

3D-SoftChip A Novel 3D Vertically Integrated Adaptive Computing System Appendix C-SystemC Codes

SC_MODU	LE(datapat	ih) {			
	sc_in <bool:< td=""><td>></td><td>clock;</td><td></td><td></td></bool:<>	>	clock;		
	sc_in <book< td=""><td>> </td><td>reset;</td><td>(7</td><td>- D-4-</td></book<>	> 	reset;	(7	- D-4-
	sc_in <sc_u< td=""><td>int<32> ></td><td>ir. m;</td><td>//immedia</td><td></td></sc_u<>	int<32> >	ir. m;	//immedia	
	sc_m <book< td=""><td></td><td>endering;</td><td>//Compare //Statue Da</td><td>riag oister Knable</td></book<>		endering;	//Compare //Statue Da	riag oister Knable
	sc_in <book< td=""><td><u>_</u></td><td>srWhEn:</td><td>//Status Re</td><td>gister Read/Write Enable</td></book<>	<u>_</u>	srWhEn:	//Status Re	gister Read/Write Enable
	sc in <sc td="" ti<=""><td>- int<4> ></td><td>aluCti:</td><td>//ALU Cor</td><td>trol Signal</td></sc>	- int<4> >	aluCti:	//ALU Cor	trol Signal
	sc in <bool:< td=""><td>></td><td>aluOEn:</td><td>//ALU Out</td><td>put Enable</td></bool:<>	>	aluOEn:	//ALU Out	put Enable
	sc in <sc td="" ui<=""><td>int<3>></td><td>shiftCtl;</td><td>//Shifter C</td><td>ontrol Signal</td></sc>	int<3>>	shiftCtl;	//Shifter C	ontrol Signal
	sc_in <bool:< td=""><td>></td><td>shiftOEn;</td><td>//Shifter O</td><td>utput Enable</td></bool:<>	>	shiftOEn;	//Shifter O	utput Enable
	sc_in <boob< td=""><td>></td><td>mulOEn;</td><td>//Multiplie</td><td>r Output Enable</td></boob<>	>	mulOEn;	//Multiplie	r Output Enable
	sc_in <sc_ui< td=""><td>int<5> ></td><td>opAIdx;</td><td>//Operand</td><td>A Index</td></sc_ui<>	int<5> >	opAIdx;	//Operand	A Index
	sc_in <sc_ui< td=""><td>int<5>></td><td>opBldx;</td><td>//Operand</td><td>B Index</td></sc_ui<>	int<5>>	opBldx;	//Operand	B Index
	sc_in <bool:< td=""><td>></td><td>rdAOEn;</td><td>//Read A O</td><td>utput Enable</td></bool:<>	>	rdAOEn;	//Read A O	utput Enable
	sc_in <bool:< td=""><td>></td><td>rdBOEn;</td><td>//Read B C</td><td>utput Enable</td></bool:<>	>	rdBOEn;	//Read B C	utput Enable
	sc_in <sc_ui< td=""><td>int<5>></td><td>wbldx;</td><td>//Writebac</td><td>k Index</td></sc_ui<>	int<5>>	wbldx;	//Writebac	k Index
	sc_in <book< td=""><td>></td><td>WDEN;</td><td>// Writebac</td><td>k Enable</td></book<>	>	WDEN;	// Writebac	k Enable
	sc_in <book< td=""><td>></td><td>immOEn;</td><td>//immedia</td><td>e Output Enable</td></book<>	>	immOEn;	//immedia	e Output Enable
	sc_in <book< td=""><td>></td><td>A now City</td><td>//Instruction</td><td>na Address Register Control</td></book<>	>	A now City	//Instruction	na Address Register Control
	sc_in <book< td=""><td>></td><td>diaCth</td><td>//Data Inni</td><td>ress Register Condition</td></book<>	>	diaCth	//Data Inni	ress Register Condition
	sc inchool	- -	dOntCtl:	//Data Out	nut Control
	se incse ui	inte325 5	din:	//Data Inni	it
	sc in <book< td=""><td>></td><td>lbEn:</td><td>//Loon Bul</td><td>fer Enable</td></book<>	>	lbEn:	//Loon Bul	fer Enable
	sc in <book< td=""><td>></td><td>lbRWEn:</td><td>//Loop Buf</td><td>fer Read/Write Enable</td></book<>	>	lbRWEn:	//Loop Buf	fer Read/Write Enable
				•	
	//Output Si	gnals			
	sc_out <boo< td=""><td>۵.</td><td>zFlag;</td><td>//Zero Flag</td><td>1</td></boo<>	۵.	zFlag;	//Zero Flag	1
	sc_out <sc_o< td=""><td>uint<32> ></td><td>iAddr;</td><td>//Instructio</td><td>on Address</td></sc_o<>	uint<32> >	iAddr;	//Instructio	on Address
	sc_out <sc_u< td=""><td>uint<32> ></td><td>dAddr;</td><td>//Data Add</td><td>ress</td></sc_u<>	uint<32> >	dAddr;	//Data Add	ress
	sc_out <sc_u< td=""><td>uint<32> ></td><td>dOut;</td><td>//Data Out</td><td>put</td></sc_u<>	uint<32> >	dOut;	//Data Out	put
	//Torne Sine	anle			
	within allow	$r_{\rm main}$	hues hus	է հաշԾ։	
	se_signales	$c_{uin} < 32 > 2$	s aluOut	//ALU Out	nut Signal
	sc signales	e uinteds s	s condFlag	1711100 (Jul	//Conditional Flag
	sc signal <s< td=""><td>$c_{uint<32>>}$</td><td>s shiftOut:</td><td>,, ,</td><td>//Shifter Output Signal</td></s<>	$c_{uint<32>>}$	s shiftOut:	,, ,	//Shifter Output Signal
	sc signal <s< td=""><td>c uint < 32 > ></td><td>s mulOut:</td><td>•</td><td>//Multiplicr Output Signal</td></s<>	c uint < 32 > >	s mulOut:	•	//Multiplicr Output Signal
	sc_signal <s< td=""><td>c_uint<4> ></td><td>tmpbusA, t</td><td>tmpbusW;</td><td>//Temp Signals for SR</td></s<>	c_uint<4> >	tmpbusA, t	tmpbusW;	//Temp Signals for SR
	sc_signal <s< td=""><td>c_uint<5>></td><td>s_shiftAmt</td><td>1</td><td>//Temp Signal for Shifter</td></s<>	c_uint<5>>	s_shiftAmt	1	//Temp Signal for Shifter
	sc_signal <s< td=""><td>c_uint<32> ></td><td>pIAddr;</td><td></td><td>//Instruction Address from PC</td></s<>	c_uint<32> >	pIAddr;		//Instruction Address from PC
	sc_signal <s< td=""><td>c_uint<32> ></td><td>llAddr;</td><td></td><td>//Instruction Address from LF</td></s<>	c_uint<32> >	llAddr;		//Instruction Address from LF
	يرم مار الارس	Caller	//Eunation	for Output	Control
	void do inf	ncu(); AntReg():	/Function	for Data In	Out Register
	void do sie	Div() {	//Function	for Gen. Sic	mals for Status Register
	sc uint< 32	> busA1, busW1, busBi	l:	tor och bij	, mis for builds register
	busA1 = bu	ISA:	-,		
	busW1 = bt	usW;			
	busB1 = bu	sB;			
	tnipbusA =	busA1.range(31,28);			
	tmpbusW =	= busW1.range(31,28);			
	s_shiftAmt	= busB1.range(15,11);	//Signal for	Shift Amou	int Control
	ł				
	*	1			
	he.	ipe;			
	100	151; 117:			
	 rceFile*	irceFile:			
	aluICS*	ialulCS:			
	shifter*	ishifter:			
	mu]•	imul;			
		-			
	SC_CTOR	(datapath) {			
		ipc=new pc("pc");			· · · · · · · · · · · · · · · · · · ·
		ipe->clock(clock);	<pre>ipc->reset("do.")</pre>	resci);	ipc->iAregUU(iAregUU);
		ipe->aAregUii(dArege	ես);	ipe->au01	11(3_410001/;

A Novel 3D Vertically Integrated Adaptive Computing System Appendix C-SystemC Codes

		ipc->iAddr(pIAddr);	ipc->dAddr(dAddr);		
		isr=new sr("sr"); isr->clock(clock); isr->wbData(tmphusW); isr->srWbEn(srWbEn);	lsr->reset(reset); lsr->wbSel(cmpFlag); isr->zFlag(zFlag);	isr->condF isr->srOEı isr->rdDat	lag(s_condFlag); 1(srOEn); a(tmpbusA);
		ialuICS=new aluICS("aluICS"); ialuICS->aluAIn(busA); ialuICS->aluOut(s_aluOut);	ialuICS->aluBIn(busB) ialuICS->condFlag(s_c); ondFlag);	jaluICS->aluCtf(aluCti);
		<pre>ishifter=new shifter("shifter"); ishifter->shiftIn(busA);ishifter->sl ishifter->shiftCtl(shiftCtl);</pre>	hiftAmt(s_shiftAmt); ishifter->shiftOut(s_sh	iftOul);	
		imul=new mul("mul"); imul->clock(clock); imul->mul=In(busB);	imul->reset(reset); lmul->mulOut(s_mulO	ut);	lam1->nmlAIn(busA);
		m=new u(' u'); ilf->clock(clock); ilf->lbEn(lbEn); int->lbEn(lbEn);	ilf->reset(reset); ilf->1AddrIn(p1Addr);		ilf->lbRWEn(lbRWEn); ilf->iAddrOut(llAddr);
		<pre>irrgFile=acw regrate (rgFate), irrgFile=>clock(clock); irrgFile=>rdAOEn(rdAOEn); irrgFile=>wbData(busW); irrgFile=>rdAData(busA);</pre>	iregFile->rdAIdx(opAl iregFile->rdBOEn(rdB iregFile->wbEn(wbEn) iregFile->rdBData(bus	(dx); ;OEn);); ;B);	iregFile->rdBldx(opBldx); iregFile->wbIdx(wbIdx); iregFile->pc(iAddr);
		SC_METHOD(do_outCll); sensitive << immOEn << abuOEn SC_METHOD(do_inOutReg); sensitive << dInCtl << dOutCll; SC_METHOD(do_sigDiv); considiue << constitute ===================================	<< shiftOEn << স্থাOEt	1 << lbEn <	<< clock << reset;
#ifdef SIM		sensitive << clock << reset;			
WELCO SEAS		zFlag.initialize(0); iAddr.initialize(0); dAddr.initialize(0); dOut.initialize(0);			
#endif	}				
); ;;					
/* * Datapat * for ICS(* Copyrig * Author: * File nan * Revision * Date: 30 */	h: Data-pa Inteiligent ht(c) 2005 : Chul KIM ne: datapat a history: V)/4/2005	th architecture Configurable Switch)RISC Core(S by Chul KIM, All right reserved (ckim@studen1.ccu.edu.au) h.cpp /ersion1	ource file for datapath)		
#include "(dalapath.h'	•			
void datap	ath::do_ou if (immOI	tC(l() { Sn) { busB = lmm;			
	} if (aluOE)	n) { busW = s_aluOut;			
	} if (shiftOI	En) { busW = s_shiftOut;			
	} if (mulOE	in) { busW = s_mulOut;			
	} if (lbEn) {	iAddr = l[Addr;	//Loop Buffer Address //Instruction Address	ing from LB	
}	} } eise (iAddr = pIAddr;	//Instruction Address	from PC	

3D-SoftChip A Novel 3D Vertically Integrated Adaptive Computing System Appendix C-SystemC Codes

void da	tapath::do_inOutReg() { if (dInCti) { busW = dIn;	//Data Input Register	
	} if (dOutCtl) { dOut = busB;	//Data Oulput Register	
}	}		

3.2 **Control Architecture**

/*		
* Def: Macros for ICS_RISC		
 Copyright(c) 2005 by Chul KIM, All right reserved 		
* Author: Chul KIM(ckim@student.ecu.edu.au)		
* File name: def.h		
* Revision history: Version1		
* Date: 2/5/2005		
*/		
	•	
#define INST_ALUIS	0	(ALU Imm. Short(1 Inst. word)
#define INST_ALUIL	1	//ALU Imm. Long(2 Inst. word)
#define INST_ALUK	2	WALU Register
#define INST_ALULB	3	WALU Loop Butter Addressing
#define INST_SHRO	4	//Smit/Rotate
#define INST_LUAD	5	//L020
#define INST_STORE	0	//Store
#define INST_BRANCH	7	//Branch (DE Control
#define INST_FECON	ð	WPE Control
#define INST_DMA	y 10	//DMA Control
#denne INSI_MUL	10	//multiply
#define COND_EO	0	/Faual
#define COND_EQ	1	//Not Ecusi
#define COND_NE	1	MNOT EQUAL
#define COND_AL	2	//Always
//#DRING COND_NV	3	<i>maever</i>
#define OP_MOVA	0	
#define OP MOVB	1	
#define OP_AND	2	
#define OP OR	3	
#define OP XOR	4	
#define OP_NOT	5	
#define OP ADD	6	
#define OP_SUB	7	
#define OP_CMP	8	
#define OP_MSR	9	
#define OP_MRS	10	
—		
#define SII_LSL	0	
#define SI1_LSR	1	
#dcfine SH_ASR	2	
#define SH_ROT	3	

/*

* Fetch: Fetch Unit for ICS_RISC(header file for fetch)

- Copyright(c) 2005 by Chul KIM, All right reserved
 Author: Chul KIM(ckim@student.ccu.edu.au)
- File name: fetch.h
- * Revision history: Version1
- * Date: 1/5/2005
- +/
| SC MODULEVIA | 1 | | |
|--|--|---|--|
| sc inchor | 1 | clock: | |
| se inese | n/~
wint<32> > | din: | //Instruction Data |
| sc_out <sc< td=""><td>_uint<32> ></td><td>finst;</td><td>//Fetched Data</td></sc<> | _uint<32> > | finst; | //Fetched Data |
| void do_f | etch(); | | |
| SC_CTO | R(fetch) { | | |
| | SC_METHOD(do | _fetch); | |
| | sensitive << clock ₁ | pos() << dIn; | |
| #ifdef SIM | et | | |
| Hondit | linst.imiliauze(0); | | |
| } | | | |
| k ' | | | |
| | | | |
| /*
 | 100 | | |
| * Fetch: Fetch Unit | for IC5_RISC(Source | e file for fetch) | |
| Copyright(c) 2005 | by Chul KIM, All rig | ght reserved | |
| * Author: Chul KIN | l(ckim@student.ccu) | edu. av) | |
| File name: fetch.cj | γ ρ | | |
| * Revision history: \ | /ersion1 | | |
| * Date: 1/5/2005 | | | |
| •/ | | | |
| #include ''fetch.h'' | | | |
| | | | |
| void fetch::do_fetch()
finst = dI |) {
n read(): | | |
| 10121 = 01 | LI CAU(7; | | |
| F | | | |
| | | | |
| /*
* Decode: Instructio | n Decoder Unit for I | CS_RISC(hcader file f | for decode) |
| Decode: Instructio Copyright(c) 2005 Author: Chul KIN File name: decode Revision history: V Date: 1/5/2005 | n Decoder Unit for 10
by Chul KIM, All rig
l(ckim@student.ecu.o
h
/crsion1 | CS_RISC(beader file f
ht reserved
edu.au) | for decode) |
| Decode: Instructio Copyright(c) 2005 Author: Chul KIX File name: decode Revision history: V Date: 1/5/2005 */ | n Decoder Unit for 10
by Chul KIM, All rig
l(ckim@student.ecu.c
h
/crsion1 | CS_RISC(header file f
ht reserved
edu.au) | or decode) |
| * Decode: Instructio
* Copyright(c) 2005
* Author: Chul KIX
* File name: decode
* Revision history: V
Date: 1/5/2005
*/ | n Decoder Unit for 10
by Chul KIM, All rig
l(ckim@student.ecu.c
h
/crsion1 | CS_RISC(header file f
ht reserved
edu.au) | or decode) |
| * Decode: Instructio
* Copyright(c) 2005
* Author: Chul KIX
* File name: decode
* Revision history: V
Date: 1/5/2005
*/
#include "systemc.h" | n Decoder Unit for 10
by Chul KIM, All rig
l(ckim@student.ecu.c
h
/crsion1 | CS_RISC(header file f
ht reserved
edu.au) | or decode) |
| * Decode: Instructio
* Copyright(c) 2005
* Author: Chul KIX
* File name: decode
* Revision history: V
* Date: 1/5/2005
*/
#include "systemc.h" | n Decoder Unit for 10
by Chul KIM, All rig
I(ckim@student.ecu.c
h
/crsion1 | CS_RISC(header file f
pht reserved
edu.au) | or decode) |
| * Decode: Instructio
* Copyright(c) 2005
* Author: Chul KIX
* File name: decode
* Revision history: V
* Date: 1/5/2005
*/
#include "systemc.h"
#include "def.h"
SC MODUL-Efdecod | n Decoder Unit for 10
by Chul KIM, All rig
l(ckim@student.ecu.c
h
/crsion1 | CS_RISC(header file f
ht reserved
edu.au) | or decode) |
| Decode: Instructio Copyright(c) 2005 Author: Chul KIN File name: decode Revision history: \ Date: 1/5/2005 */ #include "systemc.h" #include "def.h" SC_MODULE(decod
se in-chor | n Decoder Unit for 10
by Chul KIM, All rig
l(ckim@student.ecu.c
h
/crsion1
e) { | CS_RISC(header file f
ht reserved
edu.au)
clock: | or decode) |
| Decode: Instructio Copyright(c) 2005 Author: Chul KIX File name: decode Revision history: N Date: 1/5/2005 */ #include "systemc.h" #include "def.h" SC_MODULE(decod
sc_in<box colors<="" li=""> </box> | n Decoder Unit for 1(
by Chul KIM, All rig
l(ckim@student.ecu.c
h
/crsion1
e) {
b> | CS_RISC(header file f
pht reserved
edu.au)
clock;
reset; | or decode) |
| Decode: Instructio Copyright(c) 2005 Author: Chul KIX File name: decode Revision history: N Date: 1/5/2005 */ #include "systemc.h" #include "def.h" SC_MODULE(decod
sc_in<box
sc_in<box
sc_in<box< li=""> </box<></box
</box
 | n Decoder Unit for I(
by Chul KIM, All rig
I(ckim@student.ecu.o
h
/crsion1
e) {
e) {
e) { | CS_RISC(header file f
pht reserved
edu.au)
clock;
reset;
flost: | or decode)
//Fetched Instruction |
| * Decode: Instructio
* Copyright(c) 2005
* Author: Chul KIX
* File name: decode
Revision history: V
* Date: 1/5/2005
*/
#include ''systemc.h''
#include ''def.h''
SC_MODULE(decod
sc_in <box
sc_in<box
sc_in<box
sc_in<box< td=""><td>n Decoder Unit for 10
by Chul KIM, All rig
l(ckim@student.ecu.c
h
/crsion1
e) {
d>
d>
d>
uint<32> ></td><td>CS_RISC(header file f
pht reserved
edu.au)
clock;
reset;
finst;
flush;</td><td>or decode)
//Fetched Instruction
//Pipeline Flush</td></box<></box
</box
</box
 | n Decoder Unit for 10
by Chul KIM, All rig
l(ckim@student.ecu.c
h
/crsion1
e) {
d>
d>
d>
uint<32> > | CS_RISC(header file f
pht reserved
edu.au)
clock;
reset;
finst;
flush; | or decode)
//Fetched Instruction
//Pipeline Flush |
| * Decode: Instructio
* Copyright(c) 2005
* Author: Chul KIX
* File name: decode
* Revision lustory: V
* Date: 1/5/2005
*/
#include "systemc.h"
#include "def.h"
SC_MODULE(decod
sc_in <box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_in<box< td=""><td>n Decoder Unit for I(
by Chul KIM, All rig
l(ckim@student.ecu.d
h
/crsion1
/crsion1
uint<32> >
il></td><td>CS_RISC(beader file f
ght reserved
edu.au)
clock;
reset;
flast;
flusb;
refill:</td><td>'or decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill</td></box<></box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
 | n Decoder Unit for I(
by Chul KIM, All rig
l(ckim@student.ecu.d
h
/crsion1
/crsion1
uint<32> >
il> | CS_RISC(beader file f
ght reserved
edu.au)
clock;
reset;
flast;
flusb;
refill: | 'or decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill |
| Decode: Instructio Copyright(c) 2005 Author: Chul KIN File name: decode Revision history: V Date: 1/5/2005 */ #include "systemc.h" #include "def.h" SC_MODULE(decod sc_in<box sc_in<box="" sc_in<br=""></box>sc_in<box sc_in<box="" sc_in<br=""></box>sc_in<box sc_in<br=""></box>sc_in<box sc_in<br=""></box>sc_in<box sc_in<br=""></box>sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc_in
sc | n Decoder Unit for 10
by Chul KIM, All rig
l(ckim@student.ecu.d
h
/crsion1
e) {
.sl>
uint<32> >
.l>
vol>
.uint<4> > | CS_RISC(header file f
ht reserved
edu.au)
clock;
reset;
flast;
flush;
refill;
instId; | for decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill
//Instruction ID |
| * Decode: Instructio
* Copyright(c) 2005
* Author: Chul KIN
* File name: decode
Revision history: V
Date: 1/5/2005
*/
#include ''systemc.h''
#include ''def.h''
SC_MODULE(decod
sc_in <box
sc_in<box
sc_in<box
sc_out<box
sc_out<box
sc_out<box< td=""><td>n Decoder Unit for 10
by Chul KIM, All rig
l(ckim@student.ecu.d
h
/crsion1
e) {
d>
uint<32> >
d>
uint<32> >
uint<32> ></td><td>CS_RISC(header file f
ht reserved
edu.au)
clock;
reset;
finst;
flush;
refill;
instId;
cond;</td><td>for decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill
//Instruction ID
//Branch condition</td></box<></box
</box
</box
</box
</box
 | n Decoder Unit for 10
by Chul KIM, All rig
l(ckim@student.ecu.d
h
/crsion1
e) {
d>
uint<32> >
d>
uint<32> >
uint<32> > | CS_RISC(header file f
ht reserved
edu.au)
clock;
reset;
finst;
flush;
refill;
instId;
cond; | for decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill
//Instruction ID
//Branch condition |
| * Decode: Instructio
* Copyright(c) 2005
* Author: Chul KIX
* File name: decode
* Revision history: N
Date: 1/5/2005
*/
#include ''systemc.h''
#include ''def.h''
SC_MODULE(decod
sc_in <box
sc_in<box
sc_in<box
sc_in<box
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc< td=""><td>n Decoder Unit for 10
by Chul KIM, All rig
l(ckim@student.ecu.d
h
/crsion1
/crsion1
/ol>
uint<32> >
uint<32> >
uint<4> >
_uint<3> >
_uint<3> ></td><td>CS_RISC(header file f
ht reserved
edu.au)
clock;
reset;
finst;
flush;
refill;
instId;
cond;
opcode:</td><td>for decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill
//Instruction ID
//Branch condition
//Op code</td></sc<></sc
</sc
</sc
</box
</box
</box
</box
 | n Decoder Unit for 10
by Chul KIM, All rig
l(ckim@student.ecu.d
h
/crsion1
/crsion1
/ol>
uint<32> >
uint<32> >
uint<4> >
_uint<3> >
_uint<3> > | CS_RISC(header file f
ht reserved
edu.au)
clock;
reset;
finst;
flush;
refill;
instId;
cond;
opcode: | for decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill
//Instruction ID
//Branch condition
//Op code |
| * Decode: Instructio
* Copyright(c) 2005
* Author: Chul KIX
* File name: decode
* Revision history: N
Date: 1/5/2005
*/
#include "systemc.h"
#include "def.h"
SC_MODULE(decod
sc_in <box
sc_in<box
sc_in<box
sc_in<box
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc< td=""><td>n Decoder Unit for 1(
by Chul KIM, All rig
[(ckim@student.ecu.d
h
/crsion1
/crsion1
/crsion1
/crsion1
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2</td><td>CS_RISC(header file f
pht reserved
edu.au)
clock;
reset;
flast;
flast;
flusb;
refill;
instId;
cond;
opcode;
shift;</td><td>For decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill
//Instruction ID
//Branch condition
//Op code
//Shit control signal</td></sc<></sc
</sc
</sc
</sc
</sc
</box
</box
</box
</box
 | n Decoder Unit for 1(
by Chul KIM, All rig
[(ckim@student.ecu.d
h
/crsion1
/crsion1
/crsion1
/crsion1
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2 | CS_RISC(header file f
pht reserved
edu.au)
clock;
reset;
flast;
flast;
flusb;
refill;
instId;
cond;
opcode;
shift; | For decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill
//Instruction ID
//Branch condition
//Op code
//Shit control signal |
| * Decode: Instructio
* Copyright(c) 2005
* Author: Chul KIX
* File name: decode
* Revision history: V
Date: 1/5/2005
*/
#include "systemc.h"
#include "def.h"
SC_MODULE(decod
sc_in <box
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box< td=""><td>n Decoder Unit for 1(
by Chul KIM, All rig
[(ckim@student.ecu.o
h
/crsion1
/crsion1
/uint<32> >
.uint<32> >
.uint<3> >
_uint<3> >
_uint<3> >
_uint<3> >
_uint<3> ></td><td>CS_RISC(header file f
pht reserved
edu.au)
clock;
reset;
finst;
flust;
flust;
fush;
refill;
instId;
cond;
opcode;
shift;
rs1Idx:</td><td>for decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill
//Instruction ID
//Branch condition
//Op code
//shift control signal
//Rb/Rs1 Index</td></box<></box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
 | n Decoder Unit for 1(
by Chul KIM, All rig
[(ckim@student.ecu.o
h
/crsion1
/crsion1
/uint<32> >
.uint<32> >
.uint<3> >
_uint<3> >
_uint<3> >
_uint<3> >
_uint<3> > | CS_RISC(header file f
pht reserved
edu.au)
clock;
reset;
finst;
flust;
flust;
fush;
refill;
instId;
cond;
opcode;
shift;
rs1Idx: | for decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill
//Instruction ID
//Branch condition
//Op code
//shift control signal
//Rb/Rs1 Index |
| Decode: Instructio Copyright(c) 2005 Author: Chul KIX File name: decode Revision lustory: N Date: 1/5/2005 */ #include "systeme.h" #include "def.h" SC_MODULE(decod
sc_in<box
sc_in<box
sc_in<box
sc_in<box
sc_out<box
sc_out<box
sc_out<box
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc< li=""> </sc<></sc
</sc
</sc
</sc
</sc
</sc
</box
</box
</box
</box
</box
</box
</box
 | n Decoder Unit for 1(
by Chul KIM, All rig
[(ckim@student.ecu.d
h
/crsion1
/crsion1
/crsion1
/crsion1
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2
/crsion2 | CS_RISC(header file f
pht reserved
edu.au)
clock;
reset;
finst;
flush;
refill;
instId;
cond;
opcode;
shift;
rs1Idx;
rs2Idx: | For decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill
//Instruction ID
//Branch condition
//Op code
//shift control signal
//Rb/Rs1 Index
//Rb/Rs1 Index |
| * Decode: Instructio
* Copyright(c) 2005
* Author: Chul KIN
* File name: decode
Revision history: V
* Date: 1/5/2005
*/
#include "systemc.h"
#include "def.h"
SC_MODULE(decod
sc_in <box
sc_in<box
sc_in<box
sc_in<box
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc< td=""><td>n Decoder Unit for 10
by Chul KIM, All rig
(l(ckim@student.ecu.d
h
/crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2</td><td>CS_RISC(header file f
ht reserved
edu.au)
clock;
reset;
flast;
flush;
refill;
instId;
cond;
opcode;
shift;
rsIIdx;
rsIIdx;
rdItx:</td><td>For decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill
//Instruction ID
//Branch condition
//Op code
//shift control signal
//Rb/Rs1 Index
//Rs2 Index
//Rs2 Index</td></sc<></sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</box
</box
</box
</box
 | n Decoder Unit for 10
by Chul KIM, All rig
(l(ckim@student.ecu.d
h
/crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion1
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2 | CS_RISC(header file f
ht reserved
edu.au)
clock;
reset;
flast;
flush;
refill;
instId;
cond;
opcode;
shift;
rsIIdx;
rsIIdx;
rdItx: | For decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill
//Instruction ID
//Branch condition
//Op code
//shift control signal
//Rb/Rs1 Index
//Rs2 Index
//Rs2 Index |
| * Decode: Instructio
* Copyright(c) 2005
* Author: Chul KIN
* File name: decode
Revision history: V
Date: 1/5/2005
*/
#include ''systemc.h''
#include ''def.h''
SC_MODULE(decod
sc_in <box
sc_in<box
sc_in<box
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc< td=""><td>n Decoder Unit for 10
by Chul KIM, All rig
[(ckim@student.ecu.d
h
/crsion1
//crsion1
//crsion1
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
/</td><td>CS_RISC(header file f
ht reserved
edu.au)
clock;
reset;
finst;
flush;
refill;
instId;
cond;
opcode;
shift;
rsIIdx;
rs2idx;
rdIdx;
imm:</td><td>For decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill
//Instruction ID
//Branch condition
//Op code
//shift control signal
//Rb/Rs1 Index
//Rs2 Index
//Rs2 Index
//Rd Index
//Rd Index</td></sc<></sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</box
</box
</box
 | n Decoder Unit for 10
by Chul KIM, All rig
[(ckim@student.ecu.d
h
/crsion1
//crsion1
//crsion1
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
/ | CS_RISC(header file f
ht reserved
edu.au)
clock;
reset;
finst;
flush;
refill;
instId;
cond;
opcode;
shift;
rsIIdx;
rs2idx;
rdIdx;
imm: | For decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill
//Instruction ID
//Branch condition
//Op code
//shift control signal
//Rb/Rs1 Index
//Rs2 Index
//Rs2 Index
//Rd Index
//Rd Index |
| * Decode: Instructio
* Copyright(c) 2005
* Author: Chul KIX
* File name: decode
* Revision history: V
Date: 1/5/2005
*/
#include ''systemc.h''
#include ''def.h''
SC_MODULE(decod
sc_in <box
sc_in<box
sc_in<box
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc< p=""></sc<></sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</box
</box
</box
 | n Decoder Unit for 10
by Chul KIM, All rig
[(ekim@student.ecu.d
h
/crsion1
//crsion1
//crsion1
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
/ | CS_RISC(header file f
ht reserved
edu.au)
clock;
reset;
flast;
flush;
refill;
instId;
cond;
opcode;
shift;
rs1Idx;
rs2idx;
rdIdx;
imm;
immFlag. | For decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill
//Instruction ID
//Branch condition
//Op code
//shift control signal
//Rb/Rs1 Index
//Rs2 Index
//Rs2 Index
//Rd Index
//Rd Index
//Rd Index |
| * Decode: Instructio
* Copyright(c) 2005
* Author: Chul KIX
* File name: decode
* Revision history: N
Date: 1/5/2005
*/
#include "systemc.h"
#include "def.h"
SC_MODULE(decod
sc_in <box
sc_in<box
sc_in<box
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc< td=""><td>n Decoder Unit for 10
by Chul KIM, All rig
[(ckim@student.ecu.d
h
/crsion1
//crsion1
//crsion1
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
/</td><td>CS_RISC(header file f
pht reserved
edu.au)
clock;
reset;
flast;
flast;
flusb;
refill;
instId;
cond;
opcode;
shift;
rs1Idx;
rs2idx;
rdIdx;
immFlag;
cmmFlag;</td><td>For decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill
//Instruction ID
//Branch condition
//Op code
//shift control signal
//Rb/Rs1 Index
//Rs2 Index
//Rs2 Index
//Rd Index
//Immediate data
//Immediate data
//Immediate data</td></sc<></sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</box
</box
</box
 | n Decoder Unit for 10
by Chul KIM, All rig
[(ckim@student.ecu.d
h
/crsion1
//crsion1
//crsion1
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
/ | CS_RISC(header file f
pht reserved
edu.au)
clock;
reset;
flast;
flast;
flusb;
refill;
instId;
cond;
opcode;
shift;
rs1Idx;
rs2idx;
rdIdx;
immFlag;
cmmFlag; | For decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill
//Instruction ID
//Branch condition
//Op code
//shift control signal
//Rb/Rs1 Index
//Rs2 Index
//Rs2 Index
//Rd Index
//Immediate data
//Immediate data
//Immediate data |
| * Decode: Instructio
* Copyright(c) 2005
* Author: Chul KIX
* File name: decode
Revision history: N
Date: 1/5/2005
*/
#include "systemc.h"
#include "def.h"
SC_MODULE(decod
sc_in <box
sc_in<box
sc_in<box
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc< td=""><td>n Decoder Unit for 1(
by Chul KIM, All rig
[(ckim@student.ecu.d
h
/crsion1
//
bb
uint<32> >
.uint<32> >
.uint<32> >
.uint<3> >
_uint<3> >
_uint<3> >
_uint<5> >
_uint<5> >
_uint<5> >
_uint<3>>
_uint<3>>
_uint<3>>
_uint<5> >
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<</td><td>CS_RISC(header file f
ht reserved
edu.au)
clock;
reset;
flast;
flast;
flush;
refill;
instId;
cond;
opcode;
shift;
rs1Idx;
rs2Idx;
rdIdx;
immFlag;
cmpFlag;
branchener</td><td>For decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill
//Instruction ID
//Branch condition
//Op code
//shift control signal
//Rb/Rs1 Index
//Rs2 Index
//Rs2 Index
//Rs1 Index
//Rs1 Index
//Rs1 Index
//Inmediate data
//Inmediate operand flag
//Compare flag(update status register/no writeback)
//Branch Elao</td></sc<></sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</box
</box
</box
 | n Decoder Unit for 1(
by Chul KIM, All rig
[(ckim@student.ecu.d
h
/crsion1
//
bb
uint<32> >
.uint<32> >
.uint<32> >
.uint<3> >
_uint<3> >
_uint<3> >
_uint<5> >
_uint<5> >
_uint<5> >
_uint<3>>
_uint<3>>
_uint<3>>
_uint<5> >
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint<3>
_uint< | CS_RISC(header file f
ht reserved
edu.au)
clock;
reset;
flast;
flast;
flush;
refill;
instId;
cond;
opcode;
shift;
rs1Idx;
rs2Idx;
rdIdx;
immFlag;
cmpFlag;
branchener | For decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill
//Instruction ID
//Branch condition
//Op code
//shift control signal
//Rb/Rs1 Index
//Rs2 Index
//Rs2 Index
//Rs1 Index
//Rs1 Index
//Rs1 Index
//Inmediate data
//Inmediate operand flag
//Compare flag(update status register/no writeback)
//Branch Elao |
| * Decode: Instructio
Copyright(c) 2005
* Author: Chul KIX
* File name: decode
Revision history: N
Date: 1/5/2005
*/
#include "systemc.h"
#include "def.h"
SC_MODULE(decod
sc_in <box
sc_in<box
sc_in<box
sc_in<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<bbox
sc_out<bbox
sc_out<bbox
sc_out<bbox
sc_out<box
sc_</box
</bbox
</bbox
</bbox
</bbox
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
 | n Decoder Unit for 10
by Chul KIM, All rig
[(ckim@student.ecu.d
h
/crsion1
/crsion1
/uint<32> >
.uint<32> >
.uint<3> >
_uint<3> >
_uint<3> >
_uint<3> >
_uint<5> >
_uint<5> >
_uint<5> >
_uint<32> >
.oil> | CS_RISC(header file f
ht reserved
edu.au)
clock;
reset;
flast;
flust;
flust;
refill;
instId;
cond;
opcode;
shift;
rsIIdx;
rsIIdx;
rdIdx;
immFlag;
cmpFlag;
branchFlag;
srifter. | For decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill
//Instruction ID
//Branch condition
//Op code
//shift control signal
//Rb/Rs1 Index
//Rs2 Index
//Rs2 Index
//Rs2 Index
//Rd Index
//Immediate data
//Immediate operand flag
//Compare flag(update status register/no writeback)
//Branch Flag
//Compare flag. |
| * Decode: Instructio
* Copyright(c) 2005
* Author: Chul KIN
* File name: decode
Revision history: N
* Date: 1/5/2005
*/
#include "systemc.h"
#include "def.h"
SC_MODULE(decod
sc_in <box
sc_in<box
sc_in<box
sc_in<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<bbox
sc_out<bbox
sc_out<box
s</box
</bbox
</bbox
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
 | n Decoder Unit for 10
by Chul KIM, All rig
((ckim@student.ecu.d
h
/crsion1
//crsion1
//crsion1
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2 | CS_RISC(header file f
ht reserved
edu.au)
clock;
reset;
flast;
flush;
refill;
instId;
cond;
opcode;
shift;
rsIIdx;
rs2Idx;
rdIdx;
immFlag;
cmpFlag;
branchFlag;
extIFlag;
extIFlag; | For decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill
//Instruction ID
//Branch condition
//Op code
//shift control signal
//Rb/Rs1 Index
//Rs2 Index
//Rs2 Index
//Rs2 Index
//Rs2 Index
//Inmediate data
//Immediate data
//Immediate operand flag
//Compare flag(update status register/no writeback)
//Branch Flag
//End of simulaton flag
//End of simulaton flag |
| * Decode: Instructio
* Copyright(c) 2005
* Author: Chul KIX
* File name: decode
Revision history: V
Date: 1/5/2005
*/
#include "systemc.h"
#include "def.h"
SC_MODULE(decod
sc_in <box
sc_in<box
sc_in<box
sc_out<box
sc_out<box
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<box
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<box
sc_out<sc
sc_out<sc
sc_out<sc
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<box
sc_out<</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</box
</sc
</sc
</sc
</box
</sc
</sc
</sc
</box
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</sc
</box
</box
</box
</box
</box
 | n Decoder Unit for 10
by Chul KIM, All rig
[(ckim@student.ecu.d
h
/crsion1
//crsion1
//crsion1
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsion2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
//crsio2
// | CS_RISC(header file f
ht reserved
edu.au)
clock;
reset;
finst;
flush;
refill;
instId;
cond;
opcode;
shift;
rs1Idx;
rs2idx;
rdIdx;
immFlag;
cmpFlag;
branchFlag;
exlIFlag;
srOEn;
srOEn; | For decode)
//Fetched Instruction
//Pipeline Flush
//Pipeline Refill
//Instruction ID
//Branch condition
//Op code
//shift control signal
//Rb/Rs1 Index
//Rs2 Index
//Rs2 Index
//Rs2 Index
//Rs1 Index
//Immediate data
//Immediate operand flag
//Compare flag(update status register/no writeback)
//Branch Flag
//End of simulation flag
//Status register output enable |

3D-SoftChip A Novel 3D Vertically Integrated Adaptive Computing System Appendix C-SystemC Codes

```
//For Loop Buffer
           sc_out<bool>
                                                                  //Loop Buffer Enable
                                             lbEn;
           sc out<bool>
                                            lbRWEn:
                                                                  //Loop Buffer Read/Write Enable
           //PE Control
                                                                  //PE Execute Operations
           sc_out<sc_uint<3>>
                                            PEOp;
           sc_out<sc_uint<2> >
                                             PEOpmode;
                                                                  //PE Operation Mode Selection
           sc_out<sc_uint<2>>
                                             PEConfig;
                                                                  //PE Configuration
           sc_out<sc_uint<4> >
                                            PESel:
                                                                  //PE Sel
           //DMA Control
           sc_out<sc_uint<3> >
                                            DMAOp;
                                                                  //DMA Execute Operations Selection
           sc_out<bool>
                                             DFBSel;
                                                                  //Data Frame Buffer Selection(2 Sets)
           sc_out<sc_uint<4> >
                                             dataAmt:
                                                                  #Amount of Data to Transfer
           sc_out<sc_uint<6> >
                                             startAddrDFB;
                                                                  //Start Address of DFB(Source/Dest.)
           sc_out<bool>
                                            SRAMRegSel;
                                                                  //Select between SRAM/ICS_RISC Reg(Source/Dest.)
           sc_out<sc_uint<5>>
                                             startAddrSRAMReg; //Start Address of SRAM/ICS_RISC Reg(S/D)
           sc_out<bool>
                                            memSel;
                                                                  //Memory Sclection(Program/Data)
           sc_out<sc_uint<10>> startAddrProgDaMem; //Start Address of Program/Data Memory(S/D)
                                                                  //finst[31:25] for extract Instruction ID
           sc uint<7>
                                            finstid:
           sc_uint<4>
                                            instIdTmp;
                                                                  //Describe the 10 sets of Instruction ID
           void do_pipelineCtl();
                                            //Function for pipeline control
                                            //Function for extract instruction ID
           void do_instId();
                                            //Function for condition
           void do_cond();
           void do_fieldExt();
                                            //Function for instruction field extraction
           SC_CTOR(decode) {
                      SC_METHOD(do_pipetineCil);
                      sensitive << clock.pos() << flush;
                      SC_METHOD(do_instId);
                      sensitive << clock.pos() << fInst;</pre>
                      SC_METHOD(do_cond);
                      sensitive << clock.pos();
                      SC_METHOD(do_fieldExt);
                      sensitive << clock.pos() << finst;
          }
 * Decode: Instruction Decoder Unit for ICS_RISC(source file for decode)
 * Copyright(c) 2005 by Chul KIM, All right reserved
 * Author: Chul KIM(ckim@student.ecu.cdu.au)
 * File name: decode.cpp
 Revision history: Version1
 * Date: 2/5/2005
#include "decode.h"
void decode::do_pipelineCtl() {
                                            //Function for Pipeline Control
          bool
                      refillTmp;
          If (reset) {
                      refillTmp = 1;
          } else {
                      if (flush) {
                                 refiliTmp = 1;
                                 refill = 1;
                      } else {
                                 refullTmp = 0;
                                 refill = refillTmp;
                      }
          ł
```

}

k /+

*/

* Execute: Execute Unit for ICS_RISC(header file for execute)

* Copyright(c) 2005 by Chul KIM, All right reserved

* Author: Chul KIM(ckim@student.ecu.edu.au)

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes

* File name: execute.h

- Revision history: Version1
- * Date: 2/5/2005
- */

#include "systemc.h" #include "def.h"

//Opernad A Control		
#define ARD	0;	//Operand A : Rd
#define ARSI	1;	//Operand A : Rs1
#define APC	2;	//Operand A : PC
//Operand B Control		-
#define BIM	0;	//Operand B : Immediate
#define BRS2	1;	//Operand B : Rs2/ShiftAmt
#define BRD	2;	//Operand B : Rd

SC_MODULE(execute) {

sc_in <bool></bool>	clock;	
sc_in <sc_uint<4>></sc_uint<4>	instId;	//Instruction ID
sc_in <sc_uint<3>></sc_uint<3>	cond;	//Condition
sc_in <sc_uint<4>></sc_uint<4>	opcode;	//Op code
sc_in <sc_uint<3>></sc_uint<3>	shift;	//Shift Type
sc_ln <sc_uint<5>></sc_uint<5>	rs1ldx;	//Rs1 Index
sc_in <sc_uint<5> ></sc_uint<5>	rs2ldx;	//Rb/Rs2 Index
sc_la <sc_uint<5>></sc_uint<5>	rdldx;	//Rd Index
<pre>sc_in<sc_uint<32> ></sc_uint<32></pre>	imm;	//Immediate data
sc_in <bool></bool>	immFlag;	//Immediate Operand Flag
sc_in <bool></bool>	cmpFlag;	//Compare Flag (update SR, No writeback)
sc_in <booi></booi>	srOEn;	//Status Register Output Enable
sc_in <bool></bool>	sr WbEn;	//Status Register Writeback Enable
//Output Signals		-
sc_out <sc_uint<4>></sc_uint<4>	aluCtl;	//ALU Control
sc_out <bool></bool>	aluOEn;	//ALU Output Enable
sc_out <sc_uint<3>></sc_uint<3>	shiftCtl;	//Shifter Control
sc_out <bool></bool>	shiftOEn;	//Shifter Output Enable
sc_out <bool></bool>	mulOEn;	//Multiplier Output Enable
sc_out <sc_uint<5>></sc_uint<5>	opAIdx;	//Operand A Index
sc_out <sc_uint<5>></sc_uint<5>	opBIdx;	//Operand B Index/Shift Amt
sc_out <bool></bool>	rdAOEn;	//Read A Output Enable
sc_eut <bool></bool>	rdBOEn;	//Read B Output Enable
sc_out <sc_uint<5>></sc_uint<5>	wbIdx;	//Writeback Index
sc_out <bool></bool>	wbEn;	//Writeback Enable
sc_out <bool></bool>	immOEn;	//Immediate Output Enable
sc_out <bool></bool>	iAregCtl;	//Instruction Address Register Control
sc_out <bool></bool>	dAregCil;	//Data Address Register Control
sc_out <bool></bool>	dInCtt;	//Data Input Control
sc_out <bool></bool>	dOutCtl;	//Data Output Control
sc_out <sc_uint<5>></sc_uint<5>	shiftAmt;	//Shift Amount

11

void do_ctlSigGen(); //Function for Control Signal Generate void do_opSel(); //Function for Select Input OperandA,B void do_aluCtl(); //Function for Arrange AluCtl Signals vold do_shiftCtl(); //Function for Arrange ShiftCtl Signals

sc_uint<4> opcodeTmp; sc_uint<2> opA, opB;

SC_CTOR(execute) (

SC_METHOD(do_ctlSigGen); sensitive << clock.pos() << instId; SC_METHOD(do_opScl); sensitive << clock.pos() << rdIdx << rs1Idx << rs2Idx; SC_METHOD(do_aluCtl); sensitive << clock.pos() << opcode; SC_METHOD(do_shiftCtl); sensitive << clock.pos() << shift;

#ifdef SIM

3D-SoftChip A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes

	aluCtl.initlalize(0); aluOEn.initialize(0) shiftCtl.initialize(0) shiftOEn.initialize(0) opAldx.initialize(0) opBldx.initialize(0); roAOEn.initialize(1) wbfdx.initialize(0); immtOEn.initialize(1) iAregCtl.initialize(1) dAregCtl.initialize(1) jimtOEn.initialize(1) jimtOEn.initialize(1) jimtOEn.initialize(1) jimtOEn.initialize(1) jimtOEn.initialize(1) dAregCtl.initialize(1) jimtOED.initialize(1)); 0); 0); ;; ;; 0); 0); 0);
#endlf		<i>"</i> ;
	1	
}; }		
/* * Execute * Copyrig * Author: * File nar * Revisio * Date: 4/ */	e: Execute Unit for ICS_RISC(so ght(c) 2005 by Chuł KIM, All rig : Chul KIM(ckim@student.ecu.e ne: execute.epp n history: VersionI (5/2005	urce file for execute) ht reserved du.au)
#include "	evenite h ^{ti}	
#1111111	CACCOR:	
void execu //	te::do_ctlSigGen() { sc_uint<4> topcodeTmp; sc_uint<2> topA, topB; bool taluOEn, tshiftOEn bool wbEnTmp; sc_uint<4> instIdTmp; instIdTmp = instId.read(); sc_uint<5> tshiftAmt;	, tmulOEn, twbEn, tiAregCtl, tdAregCtl, tdInCtl, tdOutCtl;
	if (instidTmn 0) / //INST	41 110
	topcodeTmp	= opcode.read();
	taluOEn	= 1;
	tshiftOEn troutOEn	= 0;
	topA	= 0; = ABD:
	topB	= BIM;
	twbEn	= 1;
	tiAregCt1	= 0;
	IdAregUti (dIoCil	= 0;
	tdOutCtl	= 0;
	} else if (instIdTmp == 1) {	//INST_ALUIL
	topcodeTmp	= opcode.rear!();
		= 1;
	tmulOEn	= 0; 0;
	topA	= 3; = ARD:
	topB	= BIM;
	twbEn	= 1;
	llAregCli 13 Area Cti	= 0;
	tdInCtl	= 0; = 0:
	tdOutCt1	= 0;
	} else if (instidTmp == 2) {	//INST_ALUR
	topcodeTmp	= opcode.read();
	CELUUED tshipOFn	= 1; - 0;
	tmulOEn	= 0;
	topA	= ARSI;
	topB	= BRS2;

twbEn	= 1;
tiAregCtl	= 0;
tdAregCtl	= 0;
tđInCtl	= 0;
tdOutCtl	= 0;
clsc if (instIdTmp == 3) {	//INST_ALULB
topcodeTmp	= opcode.read();
taluOEn	= 1;
tshiftOEn	= 0;
tmulOEn	= 0:
topA	= ARSI:
tonB	= BBS2:
twhFn	= 1.
tiå rea Cil	
tdA rea Ctl	- 0;
tdlaCtl	- 0; - 0:
taben +dOutCtl	-0,
	-v; /INFT CUD/D
e ise in (instituting) == 4) (//M31_3/IKO
topcode i mp	= 0;
LAUUEN	= U;
IsmitOEn	= 1;
ImulOEn	= U;
topA	= ARS1;
topB	= BRS2;
tshiftAmt	= BRS2; //BRS2 = ShiftAmt
twhEn	= 1;
tiAregCtl	= 0;
tdAregCtl	= 0;
tdInCtl	= 0;
tdOutCil	= 0;
} else if (instIdTmp == 5) (//INST_LOAD
topcodeTmp	= OP_MOVA;
taluOEn	= 0;
tshiftOEn	= 0;
tmulOEn	= 0;
topA	= ARS1;
topB	= BRS2;
twbEn	= 1;
tiAregCtl	= 0;
tdAregCtl	= 1;
tdInCtl	= 1;
tdOutCtl	= 0;
clse if (instIdTmp == 6) {	//INST_STORE
topcodeTmp	= OP_MOVA;
taluOEn	= 0;
tshiftOEn	= 0;
tmulOEn	≃ 0;
topA	= ARS1:
topB	= BRD:
twbEn	= 0;
tiAregCtl	= 0:
tdAregCtl	= 1:
·	-,
tdlnCtl	= U:
tdIaCti tdfmCti	= U; = 1:
tdinCti tdÖutCti i cise if (instIdTmn == 7) {	= 0; = 1; //INST_BRANCH
tdinCil tdOutCti clse if (instIdTmp == 7) topcodeTmp	= 0; = 1; //INST_BRANCH = OP_ADD:
tdInCil tdOutCil } clse if (instIdTmp == 7) { topcodeTmp taluGEn	= 0; = 1; //INST_BRANCH = OP_ADD; = 1:
tdInCil tdOutCil clse if (instIdTmp == 7) { topcodeTmp taluOEn tsbitOEn	= 0; = 1; //INST_BRANCH = 0P_ADD; = 1; = 0;
tdInCil tdOutCil clse if (instIdTmp == 7) { topcodeTmp taluOEn tshiftOEn tshiftOEn	= 0; = 1; //INST_BRANCH = OP_ADD; = 1; = 0; = 0;
tdinCii tdOutCii i clse if (instIdTmp == 7) { topcodeTmp taluOEn tshiftOEn tmulOEn topA	= 0; = 1; //INST_BRANCH = OP_ADD; = 1; = 0; = 0; = 0;
tdInCil tdOutCil clse if (instIdTmp == 7) { topcodeTmp taluOEn tshiftOEn tmulOEn topA topB	= 0; = 1; //INST_BRANCH = OP_ADD; = 1; = 0; = 0; = APC; = BIM:
tdInCil tdOutCil clse if (instIdTmp == 7) { topcodeTmp taluOEn tshiftOEn tmulOEn topA topB twbEn	= 0; = 1; //INST_BRANCH = OP_ADD; = 1; = 0; = 0; = 0; = APC; = BIM; = 0:
tdInCil tdOutCil iclse if (instIdTmp == 7) { topcodeTmp taluOEn tshiftOEn tmulOEn topA topB twbEn twbEn	= 0; = 1; //INST_BRANCH = OP_ADD; = 1; = 0; = 0; = 0; = APC; = BIM; = 0; = 1;
tdinCii tdOutCii iclse if (instIdTmp == 7) { topcodeTmp taluOEn tshiftOEn tmulOEn topA topB twbEn tiAregCii	= 0; = 1; //INST_BRANCH = OP_ADD; = 1; = 0; = 0; = APC; = BIM; = 0; = 1; = 0:
tdinCti tdOutCti idOutCti iclse if (instIdTmp == 7) { topcodeTmp taluOEn tshiftOEn tshiftOEn topA topB twbEn tiAregCti tdAregCti	= 0; = 1; //INST_BRANCH = OP_ADD; = 1; = 0; = 0; = 0; = APC; = BIM; = 0; = 1; = 0; = 0;
tdInCti tdOutCti clse if (instIdTmp == 7) { topcodeTmp taluOEn tshiftOEn tmulOEn topA topA topB twbEn tiAregCti tdAregCti tdInCti tdOutCti	= 0; = 1; //INST_BRANCH = OP_ADD; = 1; = 0; = 0; = APC; = BIM; = 0; = 1; = 0; = 0; = 0; = 0;
tdinCii tdOutCii idOutCii iclse if (instIdTmp == 7) { topcodeTmp taluOEn tshiftOEn tmuIOEn topA topB twbEn tiAregCii tdAregCii tdAregCii tdInCii tdOutCii	= 0; = 1; //INST_BRANCH = OP_ADD; = 1; = 0; = 0; = APC; = BIM; = 0; = 1; = 0; = 0; = 0; = 0; = 0; = 0; = 0;
tdinCil tdOutCil idOutCil iclse if (instIdTmp == 7) { topcodeTmp taluOEn tshiftOEn tmuIOEn topA topB twbEn tiAregCil tdAregCil tdAregCil tdInCil tdOutCil } clse if (instIdTmp == 10) {	= 0; = 1; //INST_BRANCH = OP_ADD; = 1; = 0; = 0; = 0; = APC; = BIM; = 0; = 1; = 0; = 0; = 0; = 0; = 0; = 0; = 0; = 0
tdinCti tdOutCti idOutCti clse if (instIdTmp == 7) { topcodeTmp taluOEn tshiftOEn tmulOEn topA topB twbEn tiAregCti tdAregCti tdAregCti tdInCti idOutCti } clse if (instIdTmp == 10) { topcodeTmp taluOEn topCodeTmp	= 0; = 1; //INST_BRANCH = OP_ADD; = 1; = 0; = 0; = 0; = APC; = BIM; = 0; = 1; = 0; = 0; = 0; = 0; = 0; = 0; = 0; = 0
tdinCii tdOutCii idOutCii clse if (instIdTmp == 7) { topcodeTmp taluOEn tshiftOEn tmulOEn topA topB twbEn tiAregCii tdAregCii tdAregCii tdInCii tdOutCii } clse if (instIdTmp == 10) { topcodeTmp taluOEn (shiftOEn	= 0; = 1; //INST_BRANCH = OP_ADD; = 1; = 0; = 0; = 0; = APC; = BIM; = 0; = 1; = 0; = 0; = 0; = 0; = 0; = 0; = 0; = 0

**

A Novel 3D Vertically Integrated Adaptive Computing System Appendix C-SystemC Codes

		tmulOEn		= l;		
		topA		=ARS1;		
		topB		= BRS2;		
		twbEn		= 1;		
		tiAregCil		= 0;		
		tdAregCtl		= 0;		
				= U;		
	1 -1 re d	IdOutCu	0.1	= 0	CON CLARK	
	} eise ii (int	sita i mp == toneodoTr	· D) {	//INSI_PE	CONShould E	e modified
		taluOCn	u h	= 0pcoue.r	cau();	
		tchiftOFn		= 0;		
		tmulOEn		= 0;		
		topA		= ARS1;		
		topB		= BRS2;		
		twbEn		= 1;		
		tiAregCtl		= 0;		
		tdAregCtl		= 0;		
		tdInCtl		= 0;		
		tdOutCtl		= 0;		
) else if (in:	stIdTmp ==	9) (//INST_DN	IA Should be I	nodified
		topcodeTn	np 🛛	= opcode.re	ead();	
		taluOEn		= 1;		
		tshiftOEn		= 0;		
		tmulOEn		= 0;		
		topA		= ARS1;		
		topB		= BRS2;		
		IWDEN ALA HANG		≃ 1; _ 0:		
		uAregCu		= 0;		
		tdInCtl		-0;		
		tdOut('fl		= 0;		
	}	NOUTON		- •,		
	•					
	opcodeTm	p	= topcodeT	împ;		
	aluOEn		= taluOEn	;		
	shiftOEn		= tshiftOE	n;		
	mulOEn		= tmulOE);		
	opA		= topA;			
	opB		= lopB;			
	WOEnTmp		= IWDEN;			
	LATESCU ResearCu		= td t rog C	1; 41.		
	diaco		= tdIaCet	ц;		
	dOutCil		= 1dOutCl	•		
#	shiftAmt		- Ishift Am	4) 4•		
				••		
	rdAOEn =	-srOEn:				
	if (opB != 0	D) {				
		rdBOEn =	1;			
	}					
	if ((empFla	ig==0) &&	(wbEnTmp:	==1)){		
		wbEn = 1;				
	}					
	if (opB ==	0) {				
	,	IMMOED =	= 1;			
	1					
1						
void evecu	terido onSe	101				
TUIG CALL	if (onA ==	0) (//ARD	
		opAIdx = 1	rdidx:			
	} else if (or	A == 1) {	//ARS1			
	•*	opAIdx =:	rsl Idx;			
	} else if (op	A == 2) {	//APC			
	•	opAIdx =	15;			
	}					
	··· -	-				
	if (opB ==	0) {			//BIM	

//

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes



- * File name: control.h
- * Revision history: Version1
- * Date: 5/5/2005
- •7

#include "systemc.h" #include "def.h" #include "fetch.h" #include "decode.h" #include "execute.h" #include "debug.h"

SC_MODULE(control) {

sc_in <bool></bool>	clock;	//Clock
sc_in <bool></bool>	reset;	
sc_in <sc_uint<32> ></sc_uint<32>	din;	//Data Input
sc_in <bool></bool>	zFlag;	//Zero Flag
<pre>sc_out<sc_uint<32> ></sc_uint<32></pre>	lmm;	//Immediate Data
sc_out <bool></bool>	cmpFlag; //Comp	are Flag
sc_out <bool></bool>	srOEn;	//SR Output Enable
sc_out <bool></bool>	srWbEn;	//SR Writeback Enable
sc_out <sc_uint<4> ></sc_uint<4>	aluCtl;	//ALU Control

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes

sc_out<bool> aluOEn; //ALU Output Enable sc out<sc uint<3>> shiftCtl: //Shifter Control sc_out<bool> shiftOEn; //Shifter Output Enable sc_out<bool> molOEn: //Multiplicr Output Enable sc_out<sc_uint<5>> opAIdx; //Operand A Index sc out<sc uint<5>> opBidx: //Operand B Index sc_out<bool> rdAOEn; //Read A Output Enable sc_out<bool> rdBOEn; //Read B Output Enable sc out<sc uint<5>> wbldx: //Writeback Index sc_out<bool> wbEn; //Writeback Enable sc_out<bool> ImmOEn: //Immediate Output Enable sc_out<bool> iAregCtl; //Instruction Address Register Control sc_out<bool> dAregCtl; //Data Address Register Control sc_out<bool> dInCil: //Data Input Control sc out<boot> dOutCil; //DAta Output Control //Extended Output sc_out<sc_uint<5>> shiftAmt; //Shift Amount sc out<bool> lbEn: //Loop Buffer Enable lbRWEn: sc_out<bool> //Loop Buffer Read/Write Enable sc_out<sc_ulnt<3>> PEOp; //PE Execution Operation sc_out<sc_uint<2>> PEOpmode; //PE Operation Mode Selection sc_out<sc_uint<2>> PEConfig; //PE Configuration sc_out<sc_uint<4>> PEScl; //PE Selection sc_out<sc_uint<3>> DMAOp; //DMA Operation Selection sc_out<bool> //Data Frame Buffer Selection DFBSel: sc_out<sc_uint<4>> dataAmt; //Amount of Data to Transfer sc_out<sc_uint<6>> startAddrDFB: //Start Address of DFB(Source/Dest.) //Select between SRAM/ICS_RISC Reg(S/D) SRAMRegSel; sc_out<bool> sc_out<sc_uint<5> > startAddrSRAMReg; //Start Address of SRAM/ICS_RISC Reg(S/D) sc_out<bool> //Memory Selection(Program/Data) memSel: sc_out<sc_uint<10>> startAddrProgDaMem; //Start Address of Program/Data Memory(S/D) sc_signal<sc_uint<32> > finst; //Fetched Instruction Data sc signal<tool> flush: //Pipeline Flush sc_signal<bool> refill; //Pipeline Refill sc_signal<sc_uint<4>> dInstId; //Instruction ID sc_signal<sc_uint<3>> //Condition dCond; sc_signal<sc_uint<4>> opcode; //Opcode sc_signal<sc_uint<3> > shift: //Shift Control sc_signal<sc_uint<5>> rsl Idx: //Rs1 Index sc signal<sc ulnt<5>> //Rs2 Index rs2Idx: sc_signal<sc_uint<5>> rdldx: //Rd Index sc_signal<sc_uint<32>> dlmm; //Immediate Data sc_signal<bool> immFlag; //Immediate Operand Flag sc_signal<bool> dCmpFlag;//Compare Flag sc_signal<bool> dBranchFlag;//Branch Flag sc_signal<bool> dExitFlag; //End of Simulation Flag sc_signal<bool> dSrOEn, dSrWbEn; //SR Read/Write Enable sc_signal<bool> dLbEn, dLbRWEn; //LB Enable Read/Write Enable dPEOp; //PE Execution Operation sc_signal<sc_ulnt<3>> sc_signal<sc_uint<2>> dPEOpmode; //PE Operation Mode Selection sc_signal<sc_uint<2>> dPEConfig: **//PE Configuration** sc_signal<sc_uint<4>> dPESel: //PE Selection sc_signal<sc_ulnt<3>> dDMAOp; //DMA Operation Selection sc_signal<bool> dDFBSel; //DFB Selection sc_signal<sc_uint<4>> dDataAmi; //Amount of Data to Transfer sc_signal<sc_uint<6> > dStartAddrDFB; sc_signal<bool> dSRAMRegSel; dStartAddrSRAMReg; sc_signal<sc_uint<5>> sc_signal<bool> dMemSel; sc_signal<sc_uint<10>> dStartAddrProgDaMem; sc_signal<sc_uint<4>> dAluCtl; //ALU Control sc_signal<bool> dAluOEn; //ALU Output Enable sc_signal<sc_uint<3> > dShiftCtl; //Shift Control dShiftOEn;//Shift Output Enable sc_signal<bool> dMulOEn; //Multiplier Output Enable sc_signal<bool> sc_signal<sc_uint<5>> dOpAIdx; //Operand A Index sc_signal<sc_uint<5>> dOpBldx; //Operand B Index sc_signal<bool> dRdAOEn;//Read A Output Enable

	se slanatst	nol>	dRdBOEn:	//Read B Ou	tout Enable		
	se signales	e nint<5>>	dWbIdx:	//Writeback	Index		
	se signale	nolo	dWbEn:		/Writeback Enable		
	sc signal<	1001>	dimmOEn;	//Immediate	Output Enable		
	sc signal<	ool>	dIAregCtl;	//Instruction	Address Register Con	atrol	
	sc signal<	pool>	dDAregCtl	; /	/Data Address Registo	r Control	
	sc signal<	bool>	dDInCtl;	//Data Input	Control		
	sc signal<	ioal>	dDOutCil;	//Data Outp	ut Control		
//	sc_signal <s< td=""><td>sc_uint<5>></td><td>dShiftAmt;</td><td>//Shift Amou</td><td>int</td><td></td><td></td></s<>	sc_uint<5>>	dShiftAmt;	//Shift Amou	int		
		_					
	sc_signal <i< td=""><td>nt></td><td>instIdText;</td><td>//Instruction</td><td>1D Debug Informatio</td><td>n</td><td></td></i<>	nt>	instIdText;	//Instruction	1D Debug Informatio	n	
	sc_signal <i< td=""><td>nt></td><td>aluText;</td><td>//ALU Debu</td><td>g Information</td><td></td><td></td></i<>	nt>	aluText;	//ALU Debu	g Information		
	- •						
	//Pipeline I	Registers					
	sc_ulnt<4>	,	instId;		/Instruction ID		
	sc_uint<3>	,	cond;		//Execution Condition		
11	bool		cmpFlag;		//Compare Flag		
	bool		branchFlag	;;	//Branch Flag		
	bool		eExitFlag;		//Exit Flag		
//	bool		srOEn;		//SR Output Enable	-	
	bool		eSrWDEn;		//SK Writeback Enabl	e	
//	sc_uint<4>	•	aluCli;		ALU Control		
11	bool		aluOEn;	4	ALU Output Enable		
//	sc_uint<3>	•	shincu;	L	Shuter Output Cont	701	
11	bool		shiftOEn;	L	Shuter Output Enab	18	
#	bool		mulOEn;		Multiplier Output E	nadie	
11	sc_uint<5>	•	opAldx;		Operand A Index		
//	sc_uint<5>	•	opBidx;		//Operand B Index		
//	bool		rdAOEn;		Read A Output Enal	ne	
11	bool		rdBUEn;		/Read B Output Enai	ne	
//	sc_uint<5>	•	wbldx;		//Writeback Index		
	bool		eWbEn;		//Writeback Enable		
//	sc_uint<32	>	imm;		//Immediate Data		
11	bool		immOEn;		//Immediate Output E	nadie	
	bool		elAregCtl;		//Instruction Address	Register Cor	illo!
//	bool		dAregCtl;		//Data Address Regist	er Control	
11	bool		dInCtl;		//Data Input Control		
	bool		eDOutCtl;		//Data Output Contro	1	
	void do_pi	pekeg();					
	void do_co	ondExe();					
		18. a. h.					
	fetch*	Melco;					
	decone*	laecode;					
	execute	lexecule;					
	debug*	idebug;					
	SO CTO	Manadural) (
	SC_CION	(COMFUI) (ah!!\}				
		if stok selectivelocity	ifateh->dir	(dIn)	ifeich->fInct(fInst):		
		idecode=pow_docode("Jocode")	.((1117))	Inter Principality		
		idecode-sclock(clock)	· · · ·	idecode-pro	set(reset): idecode->	finst(finst):	
		Idecode South (Dech)		Idecode->re	fill(rcfill): idecode->	inst[d(dInst]	(d);
		idecode.scond(dCon	• d):	idecode-sou	code(opcode):	idecode->	hif((shift):
		idecode.src1Idv(rc11	dy).	idecode->rs	2Idx(rs2Idx):	idecode->r	dIdx(rdIdx):
		idecade.simm(dInm)	1. I.	idecode-sin	mFlag(lmmFlag);	idecode-	,
SemnFlag	dCmnFlag), 1967-006->MBU(QANNI	71	10000000			
Zubi ing	(ucmpring	/› idecode->hranchFlag	(dBranchFla	g); idecode->	exitFlag(dExitFlag);	idecode->sr()En(dSrOEn);
		idecode->srWhEn(dS	srWbEn);	idecode->lt	En(dLbEn);		idecode-
SINRWER	dL bRWEn):					
>JUX: DI		idecode->PEOp(dPE	On):	idecode->P	EOpmode(dPEOpmo	de):	idecode-
SPFConfi	a/dPEConfi	a):	5 P/I				
>1 LCOLL	E(ur noon	idecode->PESel(dPE)	Sch:	idecode->D	MAOp(dDMAOp);		idecode-
SDERSelf	dDFBSel):	Integer Pr Eber(al	~~~,,,		F (E //		
>proort		idecode->dataAmt(d)	DataAmt):	idecode->st	artAddrDFB(dStartA	.ddrDFB);	
		idecode->SRAMRep!	SekdSRAMR	tegSel): idece	de->startAddrSRAM	Reg(dStartA	ddrSRAMReg);
		Idecode->memSel/dN	lemSel):	idecode->st	artAddrProgDaMem	(dStartAddr	ProgDaMem);
		iexecute=new execute	e("execute"):		-0		
		iexecute->clock(cloc)	d:	iexecute->i	nst[d(dInstId);	iexecute->	cond(dCond);
		iexecute->opcode(op	code);	iexccute->s	hift(shift); lexecute-	>rsildx(rsll	dx);

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes

	<pre>iexecute->rs2Idx(rs2Idx); iexecute->immFlag(immFlag); iexecute->srWbEn(dSrWbEn); iexecute->shlftCtl(dShiftCtl); iexecute->opAIdx(dOpAIdx);</pre>	iexecute->rdIdx(rdIdx); iexecute->cnipilag(dCmpFlag); id iexecute->aluCil(dAluCil); iexecute->shiftOEn(dShiftOEn); i iexecute->opBIdx(dOpBidx);	lexecute->Imm(dImm); execute->srOEn(dSrOEn); lexecute->aluOEn(dAluOEn); execute->mulOEn(dAluOEn); iexecute-
>rdAOEn(dRdAOEn)); iexecute->rdBOEn(dRdBOEn); iexecute->immOEn(dImmOEn); >-	iexecute->wbIdx(dWbIdx); iexecute->iAregCtl(dIAregCtl);	icxecute->wbEn(dWbEn); icxecute-
>oAregut(aDAregut	i); iexecute->dInCtl(dDInCtl);	iexecute->dOutCtl(dDOutCtl);	//iexecute-
>instIdText(instIdText	' idebug=new debug("debug''); idebug->InstId(dInstId); t);	ldebug->aluCü(dAluCü);	idebug-
#ifdef SIM	SC_METHOD(do_pipeReg); sensitive << clock.pos() << reset; SC_METHOD(do_condExe); sensitive << clock.pos();		
	imm.initialize(0); cmpFlag.initialize(0); srOEn.initialize(0); srWbEn.initialize(0); aluCU.initialize(0); aluOEn.initialize(0);		
	shiftCtt.initialize(0); shiftOEn.initialize(0); mulOEn.initialize(0); opAIdx.initialize(0); opBIdx.initialize(0); ctAOE= baitialize(0);		
	rdBOEn.initialize(0); wbIdx.initialize(0); wbEn.initialize(0); immOEn.initialize(0); iAregC(11.initialize(0); iAregC(11.initialize(0);		
	dInCtl.initialize(0); dOutCtl.initialize(0);		
#endif } };			
/* * Control: Control A * Copyright(c) 2005 * Author: Chul KIM * File name: control. * Revision history: V * Date: 5/5/2005 */	rch. for ICS_RISC(source file for co by Chul KIM, All right reserved (ckim@student.ecu.edu.au) cpp ersion1	ontrol)	
#include ''control.h''			
void control::do_pipel if (reset) {	Reg() (
	cond = 0; eSrWbEn = 0; eWbEn = 0; eIAregCtl = 0; branchFlag = 0; eExiFlag = 0;		
) else {	instid = dInstId		
	cond = dCond; cmpFlag = dCmpFlag; branchFlag= dBranchFlag; eExltFlag = dExitFlag;		

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes

```
srOEn
                             = dSrOEn;
                    eSrWbEn = dSrWbEn;
                    aluCtl
                             = dAluCtl;
                    aluOEn
                             = dAluOEn;
                             = dShiftCtl;
                    shiftCtl
                    shiftOEn = dShiftOEn;
                    mulOEn = dMulOEn;
                    opAidx
                             = dOpAIdx;
                    opBldx
                             = dOpBIdx;
                    rdAOEn = dRdAOEn;
                    rdBOEn = dRdBOEn;
                              = dWbIdx;
                    wbldx
                    eWbEn
                             = dWbEn;
                    immOEn = dImmOEn;
                              = dImm;
                    imm
                    eIAregCil = dIAregCil;
                    dAregCtl = dDAregCtl;
                              = dDInCtl:
                    dInCil
                    cDOutCtl = dDOutCtl;
                    lbEn
                             ⇒ dLbEn:
                    lbRWEn
                            = dLbRWEn;
                    PEOp
                              = dPEOp;
                    PEOpmode= dPEOpmode;
                    PEConfig = dPEConfig;
                    PESel
                             = dPESel;
                    DMAOp
                            = dDMAOp;
                    DFBSel
                            = dDFBSel;
                    dataAmt = dDataAmt;
                    startAddrDFB =dStartAddrDFB;
                    SRAMRegScl=dSRAMRegSel;
                    startAddrSRAMReg = dStartAddrSRAMReg;
                    memSel = dMemSel;
                    startAddrProgDaMem = dStartAddrProgDaMem;
          ł
ł
void control::do_condExe() {
                    execFlag; //Execute Flag
          bool
          bool
                    exitFlag;
          if ((cond==COND_AL) || ((cond==COND_EQ) && (zFlag==1)) || ((cond==COND_NE) && (zFlag==0))) {
                    execFlag = 1;
          }
          flush = branchFlag & execFlag;
          wbEn = (execFlag && -refill) ? eWbEn : 0;
          srWbEn = (execFlag && -refill) ? eSrWbEn : 0;
          iAregCil = (execFlag && ~refill) ? clAregCil : 0;
          dOutCtl = (execFlag && ~refill) ? eDOutCtl : 0;
          exitFlag = (execFlag && ~refill) ? eExitFlag : 0;
ł
```

/*

* Debug: Debug Information for ICS_RISC(header file for debug)

- * Copyright(c) 2005 by Chul KIM, All right reserved
- Author: Chul KIM(cklm@student.ecu.edu.au)
- File name: debug.h
- Revision history: Version1
- Date: 5/5/2005
- •/

#include "systemc.h" #include "def.h"

SC_MODULE(debug) (

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes

	sc_in <sc_ui sc_in<sc_ui sc_out<int> sc_out<int></int></int></sc_ui </sc_ui 	int<4> > int<4> >	instId; aluCtl; instIdText; aluText;	
	void do_del	bug();		
	SC CTOD	(dobuo) (
	SC_CTOR	SC_METI: sensitive <	IOD(do_debug); < instId << aluCtl;	
#ifdef SIM		1 41 100 4	1	
		aluText ini	.1011211226(0); itialize(0):	
#endif		alateanin		
	}			
};				
/* * Debug: * Copyrig * Author: * File nar * Revisio: * Date: 5/ */ #include ''	Debug Infor 3bt(c) 2005 b Chui KIM(ne: debug.cp n history: Ve 5/2005 debug.h''	mation for y Cbul KIN ckim@stud pp crsion1	ICS_RISC(source file for debug) M, All right reserved lent.ecu.edu.au)	
Helofino A I	IIIS	A ,		
#define AI		U; 1.		
#define A1	TIR .	2:		
#define AI	JILB	3.		
#define SI	IRO	4:		
#define LC	DAD	5:		
#define ST	ORE	6:		
#define Bl	ANCI	7:		
#define M	UL	8:		
#define Ph	CON	9:		
#define D!	MA .	10;		
vold debuj	g::do_debug	0{		
	sc_unt<4>	Instia i mp); 	
	sc_unt<4>		p; tra	
	aluCtiTmp	= instituted	ad(); ead();	
	switch (ins	ildTmp) { case INST case INST	<pre>'_ALUIS : instIdText = ALUIS; printf("ALUIS \n"); '_ALUIL : instIdText = ALUIL; printf("ALUIN \n"); '_ALUR : instIdText = ALUR; printf("ALUIN \n"); '_ALULB : instIdText = ALULB; printf("ALULB \n"); '_SHRO : instIdText = SHRO; printf("ALULB \n"); '_LOAD : instIdText = LOAD; printf("CALULB \n"); '_STORE : instIdText = BCANCH; printf("CADD \n"); '_BRANCH: instIdText = BRANCH; printf("BRANCH \n"); '_MUL : instIdText = BRANCH; printf("BRANCH \n"); '_PECON : instIdText = PECON; printf("MUL\"); '_DMA : instIdText = DMA; printf("DMA \n");</pre>	break; break; break; break; break; break; break; break; break; break;
ı	ł			
1				

/*

BusCtl: I/O Bus Control for ICS_RISC(header file for busCtl)
Copyright(c) 2005 by Chul KIM, All right reserved
Author: Chul KIM(ckim@student.ecu.edu.au)

^{*} File name: busCtl.h

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes

```
* Revision history: Version1
 * Date: 5/5/2005
 •/
#include "systemc.h"
SC_MODULE(busCtl) {
          sc_in<bool>
                               nRW;
          sc_in<sc_uint<32>>
                              dataOut;
          sc_out<sc_uint<32>> dataIn;
          sc_inout_rv<32>
                               data;
          void do_busCtl();
          SC_CTOR(busCtl) (
                    SC_METHOD(do_busCti);
                    sensitive << nRW << dataOut;
          ł
Ŀ
н
 * BusCtl: I/O Bus Control for ICS_RISC(source file for busCtl)
 * Copyright(c) 2005 by Chul KIM, All right reserved
 * Author: Chul KIM(ckim@student.ecu.edu.au)
 * File name: busCtl.cpp
 * Revision history: Version1
 * Date: 5/5/2005
 */
#include "busCtl.h"
void busCtl::do_busCtl() {
          dataIn = sc_uint<32> (data);
          if (nRW) (
                     data = sc_lv<32> (dataOut);
          } else {
                     1
١
 * ICS_RISC: Top module for ICS_RISC(header file for ICS_RISC)
 * Copyright(c) 2005 by Chul KIM, All right reserved

    Author: Chul KIM(ckim@student.ecu.edu.au)

 * File name: ICS_RISC.h
 Revision history: Version1
```

- Date: 5/5/2005
- •/

#include "systemc.b" #include "datapath.h" #include "control.h" #include "busCU.h"

```
SC_MODULE(ICS_RISC) {
                                          clock;
          sc_in<bool>
          sc_in<bool>
                                          reset;
                                          iData;
                                                     //Instruction Data
          sc_in<sc_uint<32>>
                                          nRW;
          sc_out<bool>
          sc_out<sc_uint<32> >
                                                     //Instruction Address
                                          iAddr;
          sc_out<sc_uint<32>>
                                          dAddr;
                                                     //Data Address;
                                                     //Data Bus:
          sc_inout_rv<32>
                                          dData:
          //Extended Output
          sc_out<sc_uint<3>>
                                          PEOp;
                                                     //PE Execution Operation
```

A Novel 3D Vertically Integrated Adaptive Computing System

Appendix C-SystemC Codes

s	c out <sc th="" u<=""><th>int<2>></th><th>PEOpmode</th><th>;</th><th>//PE Operation Mode</th><th></th><th></th></sc>	int<2>>	PEOpmode	;	//PE Operation Mode		
5	c_out <sc_u< td=""><td>int<2>></td><td>PEConfig;</td><td>•</td><td>//PE Configuration Mod</td><td>le</td><td></td></sc_u<>	int<2>>	PEConfig;	•	//PE Configuration Mod	le	
5	c_oul <sc_u< td=""><td>int<4> ></td><td>PESel;</td><td></td><td>//PE Selection</td><td></td><td></td></sc_u<>	int<4> >	PESel;		//PE Selection		
s	ic_out <sc_u< td=""><td>int<3>></td><td>DMAOp;</td><td></td><td>//DMA Operation Select</td><td>lion</td><td></td></sc_u<>	int<3>>	DMAOp;		//DMA Operation Select	lion	
S	c_out <bool< td=""><td>></td><td>DFBSel;</td><td></td><td>//DFB Selection (2 Sets)</td><td></td><td></td></bool<>	>	DFBSel;		//DFB Selection (2 Sets)		
S	ic_out <sc_u< td=""><td>int<4> ></td><td>dataAmt;</td><td>//Amount I</td><td>Data to Transfer</td><td></td><td></td></sc_u<>	int<4> >	dataAmt;	//Amount I	Data to Transfer		
5	c_out <sc_u< td=""><td>int<6> ></td><td>startAddrD</td><td>FB;</td><td>//Start Address of DFB</td><td></td><td>Fal</td></sc_u<>	int<6> >	startAddrD	FB;	//Start Address of DFB		Fal
5	c_out <bool< td=""><td>></td><td>SKAMRegs</td><td>iel; Dabati</td><td>//SKAN/IUS</td><td>S_KISU KIY</td><td>SCI Pag</td></bool<>	>	SKAMRegs	iel; Dabati	//SKAN/IUS	S_KISU KIY	SCI Pag
5	ic_out <sc_u< td=""><td>int<5>></td><td>startAddrs</td><td>клмкед;</td><td>//Start Address of SNAD</td><td>mico_motol</td><td>. KCg</td></sc_u<>	int<5>>	startAddrs	клмкед;	//Start Address of SNAD	mico_motol	. KCg
5	c_out <bool< td=""><td>> tastatila a stastikaldal</td><td>memsel;</td><td></td><td>//Memory Sciecholigrid</td><td>am/Data Mi</td><td>וחזי</td></bool<>	> tastatila a stastikaldal	memsel;		//Memory Sciecholigrid	am/Data Mi	וחזי
3	ic_out <sc_u< td=""><td>ini((10)) stariAutori</td><td>nugoamun,</td><td>1</td><td>Instart Addites of Freg</td><td></td><td></td></sc_u<>	ini((10)) stariAutori	nugoamun,	1	Instart Addites of Freg		
5	sc_signal <sc< td=""><td>_uint<32> ></td><td>imnı;</td><td></td><td></td><td></td><td></td></sc<>	_uint<32> >	imnı;				
5	sc_signal be	ool>	cmpFlag;				
5	sc_signal b	ool>	srOEn;				
5	sc_signal be	ool>	srWbEn;				
5	sc_signal <sc< td=""><td>:_uint<4> ></td><td>aluCü;</td><td></td><td></td><td></td><td></td></sc<>	:_uint<4> >	aluCü;				
5	sc_signal <se< td=""><td>:_uint<3> ></td><td>shiftCil;</td><td></td><td></td><td></td><td></td></se<>	:_uint<3> >	shiftCil;				
	sc_signal b	00 >	aluOEn;				
5	sc_signal <d< td=""><td>001></td><td>SantOEn;</td><td></td><td></td><td></td><td></td></d<>	001>	SantOEn;				
1	se_signal <n< td=""><td>001> intefa</td><td>muloca;</td><td></td><td></td><td></td><td></td></n<>	001> intefa	muloca;				
1	st_signalesc	_uun<3>>	opRidy:				
-	sc_signal <ss es_cionsl </ss 	c_uuii<52.2	rdAOEn:				
	se signaleb	001>	rdBOEn:				
	se_signalese	nint<5>>	whidx:				
	se signaleh	_u	wbEn:				
	sc signal <b< td=""><td>ool></td><td>immOEn;</td><td></td><td></td><td></td><td></td></b<>	ool>	immOEn;				
	sc signal <b< td=""><td>ool></td><td>iAregCtl;</td><td></td><td></td><td></td><td></td></b<>	ool>	iAregCtl;				
	sc signal <b< td=""><td>ool></td><td>dAregCil;</td><td></td><td></td><td></td><td></td></b<>	ool>	dAregCil;				
:	sc_signal <b< td=""><td>ool></td><td>dlnCtl;</td><td></td><td></td><td></td><td></td></b<>	ool>	dlnCtl;				
1	sc_signal <b< td=""><td>ool></td><td>dOutCil;</td><td></td><td></td><td></td><td></td></b<>	ool>	dOutCil;				
1	sc_signal <se< td=""><td>c_uint<32> ></td><td>dIn;</td><td></td><td></td><td></td><td></td></se<>	c_uint<32> >	dIn;				
	sc_signal <se< td=""><td>c_uint<32> ></td><td>dOut;</td><td></td><td></td><td></td><td></td></se<>	c_uint<32> >	dOut;				
:	sc_signal <b< td=""><td>00l></td><td>zFlag;</td><td></td><td></td><td></td><td></td></b<>	00l>	zFlag;				
1	sc_signal <b< td=""><td>ool></td><td>lbEn;</td><td></td><td></td><td></td><td></td></b<>	ool>	lbEn;				
:	sc_signal <b< td=""><td>00 ></td><td>ibRWEn;</td><td></td><td></td><td></td><td></td></b<>	00 >	ibRWEn;				
1	sc_signal <b< td=""><td>00l></td><td>tnRW;</td><td></td><td></td><td></td><td></td></b<>	00l>	tnRW;				
	void do_Ou	tCtl();					
	harCilla	ikurCili					
	control*	icontrol:					
	datanath*	idatanath:					
		······					
	SC_CTOR((ICS_RISC) {					
		ibusCil=new busCil("	BUSCH");	ik	Pata Out/dOut)	iburCtLade	(oIn(dIn))
		IDUSUU->NKW(INKW)	16	100SC4-24	tatiton(dont);	10430-11-244	******(*****)1
		icontrol-new canterol	"control")				
		icontrol-sclock/clock)	·	icontrol->	reset(reset):	icontrol->d)	In(IData);
		icontrol->vious(cious)	,):	icontrol->	imm(imm):	icontrol->ci	npFlag(cmpFlag);
		icontrol->srOEn(srO)		icontrol->	srWbEn(srWbEn);	(control->al	uCtl(aluCtl);
		icontrol->aluOEn(alu	OEn);	icontrol->	shiftCtl(shiftCtl);	icontrol->sh	uftOEn(shiftOEn);
		icontrol->mulOEn(m	ılOEn);	icontrol->	opAIdx(opAIdx);	icontrol->0	pBIdx(opBIdx);
		icontrol->s dAOEn(rd	AOEn);	icontrol->	rdBOEn(rdBOEn);	icontrol->w	bldx(wbldx);
		icontrol->wbEn(wbEi	1);	icontrol->	immOEn(immOEn);	icontrol->i/	regCtl(iAregCtl);
		icontrol->dArcgCtl(d,	AregCtl);	icontrol->	dInCtl(dInCtl);	icontrol->d	OutCtl(dOutCtl);
		icontrol->lbEn(lbEn);	l	icontrol->	lbRWEn(lbRWEn);		·•
		icontrol->PEOp(PEO	p);	icontrol->	PEOpmode(PEOpmode)	;	10001201-
>PEConfig	(PEConfig);		J).	lannt-1-	DMAGn/DMAGn/		icontrol.
SDVDE-1/h	FRSelie	icouror->i,F26(1,F26	au;	ROUIL01->	manophonanoph		**************************************
Sht boci(n	rbaciji	icontrol->dataAmt(da	itaAmt):	icontrol->	startAddrDFB(startAdd	rDFB);	
		icontrol->SRAMRc25	el(SRAMRe	gSel);	Icontrol->startAddrSR	AMReg(sta	rtAddrSRAMReg);
		icontrol->mcmSel(me	mSel);	icontrol->	startAddrProgDaMem(s	tartAddrPro	ogDaMem);
		idatapath=new datap	ath("datapat	th'');	-		
		Idatapath->clock(cloc	k);	idatapath	->resci(reset);	idatapath->	-imm(imm);
		idatapath->cmpFlag(cmpFlag);	idatapath	->srOEn(srOEn);	idatapath-	
>srWbEn(s	srWbEn);						

>rdAOEn(rdAOEn); >dArcgCtl(dArcgCtl);	idatapaih->aluCtl(aluCtl); idatapaih->shiftOEn(shiftOEn); idatapaih->opAldx(opAldx); idatapaih->rdBOEn(rdBOEn); ldatapath->idBOEn(rdBOEn); ldatapath->idBOEn(rdBOEn); idatapath->idBOEn(rdBOEn); idatapath->ibEn(lbEn); idatapath->lbEn(lbEn); idatapath->lAddr(iAddr); SC_METHOD(do_OutCtl); sensitive << clock.pos();	idatapath->aluOEn(aluOEn); idatapath->mulOEn(mulOEn); idatapath->cpBldx(opBldx); idatapath->cpBldx(wbIdx); idatapath->iArcgCtl(iArcgCtl); idatapath->dOutCtl(dOutCtl); idatapath->lbRWEn(lbRWEn); ldatapath->dAddr(dAddr);	idatapath->shiftCtl(shiftCtl); idatapath- idatapath->wbEn(wbEn); idatapath- ldatapath->dIn(dIn); idatapath->zFlag(zFlag); ldatapath->dOut(dOut);
<pre>}; /* * ICS_RISC: Top module for ICS_RISC(source file for ICS_RISC) * Copyright(c) 2005 by Chul KIM, All right reserved * Author: Chul KIM(ckim@student.ecu.edu.au) * File name: ICS_RISC.cpp * Revision history: Version1 * Date: 5/5/2005 */ #include "ICS_RISC.h"</pre>			
<pre>void ICS_RISC::do_OutCll() { tnRW = dOutCll; nRW = tnRW; }</pre>			