Serial Code Accelerators for Heterogeneous Multi-core Processor with 3D Stacked Memory.

By

Philip Jacob

A Thesis Submitted to the Graduate
Faculty of Rensselaer Polytechnic Institute
in Partial Fulfillment of the
Requirements for the degree of
DOCTOR OF PHILOSOPHY
Major Subject: Computer and Systems Engineering

Approved by the
Examin ing Committee:

_________________________________________
John F. McDonald, Thesis Adviser

_________________________________________
Christopher D. Carothers, Member

_________________________________________
Tong Zhang, Member

_________________________________________
Paul Schoch, Member

Rensselaer Polytechnic Institute
Troy, New York
August 2010
CONTENTS

LIST OF TABLES ........................................................................................................... ix
LIST OF FIGURES ......................................................................................................... x
ACKNOWLEDGMENT .................................................................................................... xvii
ABSTRACT .................................................................................................................... xviii

1. Introduction ................................................................................................................. 1

2. Motivation for the research ......................................................................................... 4
   2.1 Supercomputers ....................................................................................................... 4
   2.2 MPI .......................................................................................................................... 9
   2.3 Memory wall: .......................................................................................................... 11
   2.4 Interconnect delay: ................................................................................................. 12

3. 3D Process ................................................................................................................... 14
   3.1 Wafer bonding: ....................................................................................................... 14
       3.1.1 Wafer direct bonding: .................................................................................... 14
       3.1.2 Polymer bonding: .......................................................................................... 17
   3.2 3D fabrication .......................................................................................................... 19

4. Processor Memory Stack Architecture ....................................................................... 21
   4.1 Architectural simulators: ....................................................................................... 21
   4.2 Metric: ...................................................................................................................... 22
   4.3 Benchmarks: ........................................................................................................... 22
   4.4 Processor Implementation: ...................................................................................... 22
   4.5 Ultra high bus-width: ............................................................................................ 23
       4.5.1 Dependence on Clock speed: ......................................................................... 25
       4.5.2 Interleaved cache: ........................................................................................... 26
       4.5.3 Improving L1 speeds and L2 speeds to improve CPI: .................................... 27
       4.5.4 3D L2 cache memory: ..................................................................................... 28
       4.5.5 Multi tier, Multi bank, Multi-port, 3D memories: ........................................... 29
LIST OF TABLES

Table 1: Supercomputers of current era. .................................................................6
Table 2: Simulator tools used for evaluations in 3D processor memory system study....21
Table 3: Studying the impact of multiple memory management units and wider bus. ....32
Table 4: Shared memory – multi-core processor setup for 3D architecture............36
Table 5: Architecture setup for 3D vs. 2D base processor comparison. ...............38
Table 6: Opcode Descriptions.................................................................................44
Table 7: Pre ALU Control Lines and their Function............................................48
Table 8: ALU Simulation Results..........................................................................53
Table 9: CPU testing code......................................................................................56
Table 10: 3D Cache memory architectural features.............................................60
Table 11: Power dissipation of the processor core and the dimensions of the blocks ....67
Table 12: Voltage levels used in 3-level CML design............................................88
Table 13: List of functions implemented in 74181 4 bit ALU. ............................97
Table 14: List of functions implemented in 74381 - 4 bit ALU. ............................97
Table 15: Signals of the Pipeline Controller........................................................109
Table 16: Pipeline Controller State Transition Table/ Decoding table .................111
Table 17: Pipeline Controller State Transition Table.............................................111
Table 18: Comparison of Rise, fall times and voltage levels for IIL and CML inverter circuit logic. .................................................................134
Table 19: Sample data obtained from sim-cache simulator simulating a cache system having L1 and L2 caches both levels being Harvard architecture. ...............148
Table 20: sim-outorder simulator used to simulate a processor- cache system having the above L1 and L2 caches. .................................................................149
Table 21: MPICH instruction set ..........................................................................163
LIST OF FIGURES

Figure 1: 3-D ICs are characterized by through-die interconnections, as shown in this cross-sectional view. (Source: Tezzaron Semiconductor) ........................................ 2

Figure 2a: 3D MIT Lincoln labs process - Three tiers that will be integrated to form a 3D integrated circuit at the completion of conventional integrated circuit fabrication. b. Three-tier assembly shown after bond pad etch and prior to testing the 3D circuit. ........ 2

Figure 3: 3D integration concept using wafer-bonding, showing bonding interface, vertical interchip vias and bonding approaches of "face-to-face" and "face-to-back"......3

Figure 4: Columbia supercomputer at NASA ............................................................................. 4

Figure 5: From processor to supercomputer – building of supercomputers ..................... 5

Figure 6: Projected performance of supercomputers. ................................................................. 7

Figure 7: Implementation of interconnection network to communicate between the different nodes in a supercomputer. ....................................................................................... 7

Figure 8: 80 core implementation by Intel .................................................................................. 8

Figure 9: MPI interaction between processors ........................................................................... 10

Figure 10: Memory wall because of processor memory performance difference. ........ 11

Figure 11: Interconnect length comparison of 2D vs 3D. ......................................................... 13

Figure 12: Process flow diagram of wafer bonding of two oxidized Silicon wafer at room temperature. Additional heat treatment is carried out to strengthen the bonds for additional steps such as polishing and etching.[Proc of IEEE Dec 2006] .......................... 15

Figure 13: Schematic of two hydrophilic oxidized silicon wafers with mono-layers of adsorped water: (left) at room temperature with low bond strength and (right) after annealing with high bond strength due to covalent Si-bonds at the interface. .............. 16

Figure 14: Schematic of two hydrophobic silicon wafers, terminated with hydrogen at room temperature with low bond strength, annealed to form high bond energy Si-Si interface. ......................................................................................................................... 17

Figure 15: Polymer bonding used in RPI process for wafer bonding. ................................. 18

Figure 16: Metallic bonding showing Cu pads which are bonded together by contact and pressure. ................................................................................................................................. 19

Figure 17: The impact of bus width and block size on CPI is examined. Increasing block size and bus width gives the minimum CPI ................................................................. 24
Figure 18: CPI is reduced by increasing the bus width between the cache levels. Initially this is done by using interleaved cache design that can typically provide bus width up to 128 bits. Beyond this 3D architecture provides ultra wide bus widths between L2 and L1 cache levels.

Figure 19: The variation of CPI is examined for varying bus width and clock frequency. Large bus widths would give lower CPIs for higher clock frequencies. A CPI of about 9 is seen at buswidth of 512 bits and 16 GHz clock frequency.

Figure 20: Comparison of access time composition of L1 cache made with CMOS and BiCMOS: 1. Decoder data, 2. Word Line, 3. Sense Amp data, 4. Comparator, 5. Mux, 6. Sel Inverter, 7. Output Driver. 70 nm CMOS L1 cache access time = 0.718 ns, BiCMOS L1 cache access time= 0.431 ns.

Figure 21: a: Evaluating 3D cache memory access time using a 2D cache access time evaluation tool. Cache access times evaluated for 1 MB, 70 nm process with 1 layer and 8 banks, 2 layers and 4 banks per layer, 4 layers and 2 banks per layer etc. as represented in 7b.

Figure 22: CPU with 3D memory and multiple memory management units. a. Ports in Multitier. b. Ports in Multiple banks in same tier c. Multiple ports for same bank. Each MMU unit can access a separate layer of the 3D memory and thus effectively increase the memory bandwidth.

Figure 23: CPI reduction strategies evaluated for a 32 GHz core.

Figure 24: CPI variation with multithreading and buswidth evaluated at 16 GHz and 32 GHz clock frequencies. Table 2 describes the simulated system.

Figure 25: The CPI is reduced from a high of around 23 all the way to 2.5 at 16 GHz processor speeds by systematically applying various architectural techniques. The last strategy assumes context switching with multiple memory management units. The architectural setup is as described in Table 2.

Figure 26: Execution time of 6 programs when executed on a single core, multiple cores (2, 4) and a 16, 32 GHz processors designed in the preceding sections (with 3D memory and BiCMOS memory and context switching). The impact of bus width is explored in the case of multi-core processor. The architectural setup is described in Table 2.

Figure 27: RSIM architecture adapted to 3D multi-core processor memory stack.
Figure 28: A multi-core system with 3D memory simulations to show reduction in execution time using RSIM simulator. For the simulations an FFT program is being programmed as a 4-way, 16-way and 64-way process is used. ..................................37
Figure 29: CPI has been evaluated for a base system and compared with a 3D processor memory stack described in Table 4, for SPECint and BioBench programs. ..................39
Figure 30: A 3D processor memory system architecture that is proposed. ..................41
Figure 31: FPGA implementation of the core – block schematic..................................42
Figure 32: Instruction decoder schematic........................................................................45
Figure 33: Instruction decoder simulation.........................................................................46
Figure 34: Register file design with pipeline structure.........................................................47
Figure 35: Register file schematic with 2 stage pipeline and multi-thread support............47
Figure 36: The Pre ALU stage design................................................................................49
Figure 37: Simulating the Pre ALU ..................................................................................50
Figure 38: ALU Schematic Diagram..................................................................................51
Figure 39: Carry look ahead adder structure.......................................................................52
Figure 40: ALU simulation..................................................................................................53
Figure 41: Write back stage logic.......................................................................................54
Figure 42: State machine logic...........................................................................................56
Figure 43: CPU simulation..................................................................................................57
Figure 44: Altera cyclone 3 board.......................................................................................58
Figure 45: 3D FDSOI fabrication process...........................................................................59
Figure 46: FDSOI vs. PDSOI transistor................................................................................60
Figure 47: 2D SRAM vs. 3D SRAM Test Structures ...............................................................61
Figure 48: 3D L2 cache Memory Access time: 1-Decoder; 2-Wordline + Wordline_driver; 3-Bitline; 4-Sense_Amp..........................................................61
Figure 49: 3D cache – Floor plan & Microphotograph.........................................................62
Figure 50: Reduction in performance with increasing temperature of chip. .....................64
Figure 51: 3D floorplan that is used for thermal analysis. 2 layers of memory are stacked on the CPU core. .................................................................65
Figure 52: Floor-plan of processor core .............................................................................66
Figure 53: Thermal dissipation of CPU chip placed on 1mm Cu. Chip thickness= 2000um ..........................................................68

Figure 54: Thinning down the Si wafer to 2000um to 25um allows the heat source in the circuit layer to be brought closer to the heat spreader. This allows better heat spreading and thereby lower peak temperature in the hot spots ........................................68

Figure 55: Single MOS device places on top of a hot spot. 800Si on 1mm Cu ..........69

Figure 56: 25um Si, 50um Diamond sheet 1mm Cu heatsink. Thermal conductivity of Si: 100W/K, Diamond thermal conductivity: 2000W/k,Cu plate conductivity: 400 W/k.....70

Figure 57: 25um Si, 50um Diamond sheet 1mm Cu heatsink ..................................71

Figure 58: 25um Si, 50um Diamond sheet 1mm Cu heatsink with intermediate heat sink ..................................................................................................................71

Figure 59: 3D chip bonded to diamond wafer for better heat spreading. Through vias are made in the bottom SiGe wafer to the diamond sheet........................................72

Figure 60: Horizontal displacement of the wafer tiers resulting in shear stress on 3D via ........................................................................................................................................72

Figure 61: Strain mechanisms seen on wafer bonding especially on the interconnect via. ........................................................................................................................................73

Figure 62: SiGe HBT structure compared with Si BJT structure ..................................74

Figure 63: Doping profile of the wafer for a SiGe HBT device ..................................75

Figure 64: Energy band diagram of a graded base SiGe HBT compared to Si BJT ......76

Figure 65: Strained devices formed by bending of the processed wafer. 1a : Fully processed IC wafer that has been thinned. 1b: Mechanical stress applied on the wafer and subsequently bonded to polymer carrier. 1c: Wafer bonded to permanent quartz wafer to retain stress ...........................................................................................................77

Figure 66: Growth of Si on SiGe results in strained lattice in Si layer ....................78

Figure 67: Combining low temperature epitaxial techniques with ultra high vacuum chemical vapor deposition, IBM scientists have thinned the SiGe HBTs vertical transport layer by half to push its cut off frequency to 210GHz in 8HP technology. ..........79

Figure 68: Non self aligned process for fabricating SiGe HBT..................................81

Figure 69: Fabrication steps for SiGe HBT on SOI....................................................82

Figure 70: SiGe HBT on SOI .....................................................................................83
Figure 71: Unity gain cut off frequency and Unity power gain frequency variations for different technology nodes. (a) shows variation of ft for different transistor sizes for 5HP node. (b) shows variation of ft and fmax for different transistor cross sections for 8HP. (c) shows variation of ft and fmax for the next generation with transistors having ft=350GHz. (d) shows technology node variation. ...............................84

Figure 72: Variation of ft+fmax across several generations of SiGe technology. ........85

Figure 73: CML buffer with gated voltage reference........................................87

Figure 74: Addition of emitter followers at the output of CML buffer to form ECL. ....87

Figure 75: Emitter followers used for level shifting- Level 1 to Level 2 and Level 3 respectively. ...................................................................................................................89

Figure 76: Basic logic cell that can be used for implementation of 2 input AND, NAND, NOR, OR gates..........................................................................................................................90

Figure 77: Basic logic cell that can be used for implementation of 3 input AND, NAND, NOR, OR gates..........................................................................................................................91

Figure 78: Logic cell that can be used for implementation of XOR and XNOR gates...91

Figure 79: Logic cell that can be used for implementation of Multiplexer ...............92

Figure 80: Logic cell that can be used for implementation of de-multiplexer .............93

Figure 81: Logic cell that can be used for implementation of D latch .......................94

Figure 82: a. D flip-flop formed from D-latch b. Combined Schematic of D flip-flop. ..95

Figure 83: Gate level schematic diagram of 74381.....................................................98

Figure 84: Layout of block D signal select shown in Figure 67.................................99

Figure 85: Testing Dsignal logic................................................................................100

Figure 86: DSignal output validation - D4, D7 output.............................................100

Figure 87: Layout of ABSelect .................................................................................101

Figure 88: F0 block layout ......................................................................................102

Figure 89: F1 block layout ......................................................................................102

Figure 90: F2 block layout ......................................................................................103

Figure 91: F3 block layout ......................................................................................103

Figure 92: P & G terms that could be used for CLA..............................................104

Figure 93: 4 bit operand preparation layout ............................................................104

Figure 94: Simulation set up for 4 bit operand preparation block .........................105
Figure 95: Layout of operand preparation ..........................................................105
Figure 96: Simulation setup for full chip operand preparation block ..................106
Figure 97: Droop analysis for Vee and gnd! ...................................................107
Figure 98: Color coding for droop analysis .....................................................107
Figure 99: Operand Preparation chip microphotograph ....................................108
Figure 100: Operand Preparation pad layout .................................................108
Figure 101: Pipeline controller state transition diagram ...................................110
Figure 102: Pipeline controller simulation ......................................................112
Figure 103: Block schematic of Pipeline controller chip ...................................113
Figure 104: Simulation of finite state machine used in pipeline controller ..........113
Figure 105: Layout of Finite state machine .....................................................114
Figure 106: Layout of the 32 bit D Flip flop (pipeline stage) and buffer array. ....115
Figure 107: Complete layout of state machine and pipeline stages ..................115
Figure 108: Chip layout of pipeline controller ...............................................116
Figure 109: Simulation of pipeline stage output ..............................................117
Figure 110: Pad layout for pipeline controller chip .........................................117
Figure 111: Microphotograph of fabricated chip: pipeline controller ...............118
Figure 112: Amdahl's law .............................................................................120
Figure 113: Serial code presents a limiting case as more cores are added ...........121
Figure 114: Repeater count increasing with cmos scaling ...............................122
Figure 115: Heterogeneous integration of HCRU with low power cores ..........123
Figure 116: Package limitations on bandwidth mitigated through use of 3D memory ...124
Figure 117: Modeling asymmetric multi-core processor with 3D memory ..........125
Figure 118: CMP model with shared cache ....................................................126
Figure 119: 3D memory on Multi-core architecture explored through two configurations .................................................................127
Figure 120: Integrated Injection logic using PNP as constant current injector ......129
Figure 121: Inverter outputs after stage 1 and stage 2 ....................................130
Figure 122: PNP constant current source varying because of slower PNP. .........131
Figure 123: An NPN only multi-valued logic structure ....................................132
Figure 124: Faking a multi-valued logic through the use of two NPN transistors. ......132
Figure 125: Two inverter stages implemented using the modified multi-valued transistors.................................................................133
Figure 126: Waveforms showing fast rise times comparable to CML logic.................133
Figure 127: Memory cell implemented using CML circuit........................................134
Figure 128: Memory circuit design implemented using III.........................................135
Figure 129: Showing the architectural benefits of 3D memory stacked on processor...136
Figure 130: Simplescalar simulators and their descriptions-(simple scalar tutorial).....147
Figure 131: Access time vs. technology node as calculated by CACTI, which uses a linear scaling approach to calculate access time for different nodes. .......................164
Figure 132: Access time variation with the Associativity of the cache. .......................165
Figure 133: Access time variation with the number of banks used in the cache.........165
Figure 134: Access time variations in 3D memory a: without sub-banks b: with sub-banks......................................................................................................................166
ACKNOWLEDGMENT

This research was sponsored in part by DARPA/SPAWAR IBM 3DI contract number N66001-0408032 and also the DARPA HPCS IBM PERCS contract number HBCH3039004. I would also like to thank the ECSE Department for funding me as a teaching assistant through the rest of my PhD.

I would like to thank my thesis committee members Prof. Christopher Carothers, Prof. Tong Zhang and Prof. Paul Schoch for their insights and help during my PhD. I would also like to thank former Professor at RPI Dr. Khaled Salama for his help for my candidacy committee. I was fortunate to receive their support and guidance.

I would like to thank my advisor Prof. John F. McDonald in mentoring me even before I got to join his group over a prolonged period of time. He had a major impact on my research and career. This would not have been possible without the support he gave me throughout.

I would like to thank my fellow F-RISC group members Peter Curran, Jong-Ru Guo, Okan Erdogan, Paul Belemjian, Jin-woo Kim, Michael Chu and Aamir Zia. Special thanks to Russell Kraft for the extended review sessions held in drafting the research papers and technical knowhow. Also, special regards needs to be mentioned to Steve Nicholas who ensured that the graduate students in the FRISC lab stayed nourished and exercised regularly.

I would like to acknowledge the help rendered by Kerry Bernstein, IBM for his guidance on the 3Di project.

I would like to thank my parents and my brother in keeping me motivated to pursue my education beyond the bachelor’s degree. And finally, I would like to thank my wife Sherin for the support and encouragement that she gave me through the last two years of my PhD.
ABSTRACT

Asymmetric multi-core would be the pathway for future microprocessors with different cores performing different functionality. Of particular interest is the handling of serial and parallel code present in a given application. Performance improvement in multi-core microprocessors is limited by Amdahl’s Law which states that speed up achieved by adding more cores gets saturated because of the presence of serial code in all applications. This calls for heterogeneous core integration of a fast core that accelerates serial code along with other CMOS cores that execute parallel component of the application. Performance improvement of parallel code can be improved by adding more cores, while higher clock rates can benefit serial. In either case, there is an associated memory wall problem due to limited bandwidth. We therefore require 3D stacked memory to overcome this problem. In this thesis, I evaluate high clock rate processors as well as shared memory processors with large number of cores with 3D stacked memory. Since clock rates for CMOS have tended to saturate due to wire scaling problems and excessive heat dissipation, one must look to an alternate three terminal device, which is compatible with CMOS i.e. BJTs. SiGe Hetero-junction Bipolar Transistor (HBT) BiCMOS process is used to build such fast digital chips that can clock in the 20-30GHz at reasonable power levels and densities. 3D memory stacked on top of a processor core can provide several advantages from wide bandwidth to multiport caches serving multiple cores. Memory Processor chip stacking reduces this Memory Wall problem by using a large number of vertical vias between tiers in the stack, for ultra wide bit path transfer of data and address information to and from various levels of cache. Chips have been successfully fabricated and tested in the 3D MITLL process as well as SiGe BiCMOS process. Thermal modeling for possible integration of these two processes has been carried out. The research is progressing towards the heterogeneous core integration with 3D memory.
1. Introduction

3D interconnect technology provides an alternative to conventional scaling of CMOS structures by physically placing chips one above the other and interconnecting them using 3D through vias. This approach would be necessary to ensure that Moore’s law continues to hold true. The law had been held true by successfully scaling down of transistor structures and placing more transistors per die. However as this trend progresses, interconnects become a dominating factor in the chip. With a practical limit on the number of interconnect layers, and a limit on the number of global interconnects that can be packed, it will soon require that the die area for interconnects will be larger than the area for the transistors. In 3D interconnect a large number of long interconnects are replaced by short vertical vias. This in turn reduces the RC delay seen in long interconnects.

3D technology involves stacking of dies or wafers together and bonding them. The wafer on wafer approach has a higher manufacturing throughput and yield. The diameter of the through silicon vias is reducing as the process technology improves. Although the size of the through silicon vias are still larger than a regular metal to metal interconnect via, one still can place a large number of vias in a given area. The area lost due to the size of the vias is made up by the improvement achieved in terms of global interconnect length reduction through the use of vertical vias. Yield is another concern in 3D. If a defect exists on one of the dies forming the stack, the whole stack is defective. So redundancy needs to be built into the chip to counter these defects. Also issues due to bonding or alignment could result in defective chips too. A third concern seen in 3D IC is about handling heat removal. The floor planning of the chip needs to be handled better to avoid hot spots being created which would in turn cause heat to be transmitted to the other wafers through the interconnect through vias. These issues could be resolved by having through silicon vias in the substrate of the bottom wafer that would connect to the external heat sink and the use of better heat spreaders such as diamond. Also the intermediate interfaces could have metal spreaders that could be routed to the periphery of the chip so as to allow access to direct cooling using conventional strategies. Some research is being carried out in the creating micro channels which would pass through the wafers. Air would be forced through them to cool the chip down.
While the disadvantages are significant, the advantages in 3D chips far outweigh the disadvantages in comparison to 2D chips. This chapter discusses some of the common 3D strategies that are being researched upon in the industry and academia.

Figure 1: 3-D ICs are characterized by through-die interconnections, as shown in this cross-sectional view. (Source: Tezzaron Semiconductor)

Figure 1 shows a cross section of a typical 3D bonding and 3D via interconnection as seen in a die. Our research group has been using the MIT-LL process for building 3D memory chips. This process consists of bonding 3 wafers with the first two being bonded face to face and the third wafer being bonded face to back. Figure 2 shows the process involved for bonding the three wafers together.

Figure 2:a. 3D MIT Lincoln labs process - Three tiers that will be integrated to form a 3D integrated circuit at the completion of conventional integrated circuit fabrication. b. Three-tier assembly shown after bond pad etch and prior to testing the 3D circuit.
Figure 3: 3D integration concept using wafer-bonding, showing bonding interface, vertical interchip vias and bonding approaches of "face-to-face" and "face-to-back".

Figure 3 above shows the RPI process for 3D integration. 3D Integrated circuits open the gateway to circumvent many problems seen in conventional 2D circuits. Other approaches for 3D include die bonding, face to back approach, transferred substrate approach.

As one reaches the limits of scaling (12nm or 16nm process), the researchers will have to think of 3D circuits to keep the pace of the Moore’s law going. The following chapters explore the motivation behind the 3D research (chapter 2), analysis of the benefits of 3D chip stacking (chapter 4), and design of the processor architecture (chapter 5). That is followed by a study on thermal dissipation in a 3D stack as well as understanding the SiGe process and SiGe HBTs(chapter 8). Chapter 9 explores CML circuit design using SiGe BiCMOS. This is followed by implementation studies of operand preparation circuit (chapter 10) and the finite state machine that controls the pipeline of the processor (chapter 11). Chapter 12 evaluates Heterogeneous multi core with 3D memory and this is followed by exploratory studies of Integrated Injection Logic in chapter 13.
2. Motivation for the research

Research on 3D IC processor memory stack has been motivated by multiple factors such as low performance efficiency of supercomputers, Amdahl’s law, Memory wall, larger interconnect delays. Faster processor nodes are called for to improve the overall efficiency of super computers. These nodes could provide for handling message packet processing and communication. The need for 3D processor memory stack is also seen as a necessity to overcome the memory wall problem which has limited the clock race of microprocessors. 3D stacking also provides for shorter inter-connects from vertical integration. These factors are explored in more detail in this chapter.

2.1 Supercomputers

Supercomputers have formed the backbone of several important applications such as weather simulations, nuclear reaction simulations or molecular level simulations of carbon nanotubes etc. Larger supercomputers are constantly being built to solve larger problems. Figure 4 shows a super computer at NASA.

Figure 4 : Columbia supercomputer at NASA.
Figure 5: From processor to supercomputer – building of supercomputers

Figure 5 shows a schematic of super-computer integration going from a processor to board level to racks and ultimately the complete system. Table 1 shows a list of supercomputers made over the years.

<table>
<thead>
<tr>
<th>Period</th>
<th>Supercomputer</th>
<th>Peak speed</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1943-1944</td>
<td>Colossus</td>
<td>5000 characters per second</td>
<td>Bletchley Park, England</td>
</tr>
<tr>
<td>1945-1950</td>
<td>Manchester Mark I</td>
<td>500 instructions per second</td>
<td>University of Manchester, England</td>
</tr>
<tr>
<td>1950-1955</td>
<td>MIT Whirlwind</td>
<td>20 KIPS (CRT memory), 40 KIPS (Core)</td>
<td>Massachusetts Institute of Technology, Cambridge, MA</td>
</tr>
<tr>
<td>1956-1958</td>
<td>IBM 704</td>
<td>40 KIPS</td>
<td></td>
</tr>
<tr>
<td>1958-1959</td>
<td>IBM 709</td>
<td>40 KIPS</td>
<td></td>
</tr>
<tr>
<td>1960-1961</td>
<td>LARC</td>
<td>500 kiloflops (2 CPUs)</td>
<td>Lawrence Livermore Laboratory, California</td>
</tr>
<tr>
<td>1961-1964</td>
<td>IBM 7030 &quot;Stretch&quot;</td>
<td>1.2 MIPS, ~600 kiloflops</td>
<td>Los Alamos National Laboratory, New Mexico</td>
</tr>
<tr>
<td>1965-1969</td>
<td>CDC 6600</td>
<td>10 MIPS</td>
<td>Lawrence Livermore Laboratory, California</td>
</tr>
<tr>
<td>1969-1975</td>
<td>CDC 7600</td>
<td>36 megaflops</td>
<td>Lawrence Livermore Laboratory, California</td>
</tr>
<tr>
<td>1974-1975</td>
<td>CDC Star-100</td>
<td>100 megaflops (vector), ~2 megaflops (scalar)</td>
<td>Lawrence Livermore Laboratory, California</td>
</tr>
<tr>
<td>Year</td>
<td>Supercomputer</td>
<td>Performance</td>
<td>Location</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------</td>
<td>--------------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>1975-1983</td>
<td>Cray-1</td>
<td>80 megaflops (vector), 72 megaflops (scalar)</td>
<td>Los Alamos National Laboratory, New Mexico (1976)</td>
</tr>
<tr>
<td>1975-1982</td>
<td>ILLIAC IV</td>
<td>150 megaflops, &lt;100 megaflops (average)</td>
<td>NASA Ames Research Center, California Had serious reliability problems.</td>
</tr>
<tr>
<td>1981-1983</td>
<td>CDC Cyber-205</td>
<td>400 megaflops (vector), average much lower.</td>
<td>Los Alamos National Laboratory, New Mexico</td>
</tr>
<tr>
<td>1983-1985</td>
<td>Cray X-MP</td>
<td>500 megaflops (4 CPUs)</td>
<td>Lawrence Livermore Laboratory and NASA Lawrence Berkeley National Laboratory (the only 8 CPU system)</td>
</tr>
<tr>
<td>1985-1990</td>
<td>Cray-2</td>
<td>1.95 gigaflops (4 CPUs) 3.9 gigaflops (8 CPUs)</td>
<td>Lawrence Livermore Laboratory, California</td>
</tr>
<tr>
<td>1989-1990</td>
<td>ETA-10G</td>
<td>10.3 gigaflops (vector) (8 CPUs), average much lower.</td>
<td>Lawrence Livermore Laboratory, California</td>
</tr>
<tr>
<td>1990-1995</td>
<td>Fujitsu Numerical Wind Tunnel</td>
<td>236 gigaflops</td>
<td>National Aerospace Lab</td>
</tr>
<tr>
<td>1995-2000</td>
<td>Intel ASCI Red</td>
<td>2.15 teraflops</td>
<td>Sandia National Laboratories, New Mexico</td>
</tr>
<tr>
<td>2000-2002</td>
<td>IBM ASCI White, SP Power3 375 MHz</td>
<td>7.226 teraflops</td>
<td>Lawrence Livermore Laboratory, California</td>
</tr>
<tr>
<td>2002-2004</td>
<td>Earth Simulator</td>
<td>35.86 teraflops</td>
<td>Yokohama Institute for Earth Sciences, Japan</td>
</tr>
<tr>
<td>2004-2007</td>
<td>Blue Gene/L prototype</td>
<td>70.72 teraflops †</td>
<td>IBM, Rochester, Minnesota[2]</td>
</tr>
<tr>
<td>2009</td>
<td>Roadrunner</td>
<td>1.04 petaflops</td>
<td>DOE/NNSA/LANL</td>
</tr>
<tr>
<td>2010</td>
<td>Jaquar</td>
<td>1.75petaflops/s</td>
<td>Oak Ridge National Lab</td>
</tr>
</tbody>
</table>

**Table 1: Supercomputers of current era.**

While the supercomputers of the previous generation appear to be rather slow having only a computation power of current desktops, they still were technologically advanced in its time. Many of the ideas developed in that age for the supercomputers have now been implemented on single chip processors used in current desktops. Figure 6 shows the trend in performance improvement of supercomputers. Intel is reported to be working on an 80 core wafer processor on which 3D memory is to be stacked to form the cache layer; in effect a 3D processor memory stack based super computer all in 1 wafer. The possible reason for moving to such single wafer supercomputer is to avoid the long delay seen when one attempts to communicate between the different nodes in a
supercomputer which calls for an MPI implementation. The communication needs to be done over optical fibers provided in the Interconnection network as shown in Figure 7.

Figure 6: Projected performance of supercomputers.

Figure 7: Implementation of interconnection network to communicate between the different nodes in a supercomputer.
In the current age, the circuits have been miniaturized so much that Intel has made prototypes of chips that contain 80 processor nodes in just 1 chip. Figure 8 shows the fabricated wafer. The fabricated 80 processor core will require 3D memory to operate. As this is still in research phase, it would be exciting to imagine future computers of current age being shrunk to the level of 1 chip that could fit in a desktop.

![Figure 8: 80 core implementation by Intel](image)

Figure 8: 80 core implementation by Intel

Currently, computational efficiencies are as low as 5 percent for some applications running on Tera-scale Parallel Processor Installations. It has been observed that most of the efficiency is lost during the processing of message packets that are transmitted between the CPUs in the cluster. Although the packing and unpacking of packets seem to be a trivial process for most advanced processors developed today, these processors are not fast enough to handle this application to deliver higher overall productivity.

Increasing the efficiency of modern supercomputers has become that major focus of many researchers. The huge cluster of processors is offering a very low efficiency on the order of 1 to 5%. The delay incurred is not in the individual processor performance. But in the message passing interface that is used to communicate between the processors present in the cluster. It has been observed that several thousand cycles of time is wasted
for forming, packing and unpacking message packets that need to be sent across to other processors.

2.2 MPI

MPI is a widely available communications library that enables parallel programs to be written in C, FORTRAN, and Python etc. It is thus language independent computer communications descriptive applications programmer interface, with defined semantics, and with flexible interpretations. MPI does not define the protocol by which these operations are to be performed as seen in the layer 4 of OSI model. It rather corresponds to layer 5+ type set of interfaces. MPI is a de-facto standard for communications among processes modeling a parallel program on a distributed memory system. Often these programs are mapped to cluster, actual distributed memory supercomputers. MPI is supported on shared-memory, NUMA (Non-uniform Memory Access) architecture as well.

MPI thus provides synchronization and communication functionality between a set of processes that have been mapped to different node or processors in a cluster or supercomputer. When one tries to get maximum performance, one process per processor or core is selected as part of the mapping activity. This mapping activity happens at runtime through the agent that starts the MPI program.

MPI [11] is a protocol standard that has been developed to handle inter-processor communication in huge clusters. Codes such as MPICH [9] and MPI-LAM [10] have been developed for this functionality. Multithreaded collective communication is best accomplished by using multiple copies of communicator. Communicators are groups of processes in the MPI session with their own virtual communication fabric for point to point operations. They also have independent communications addressability or space for collective communication.

The communication is done in the form of message packets transmitted over optical fibers. The processes which are assigned to different processors could form different groups. The communication could be between the groups or internally inside a given group. The former uses inter-communicator while the latter uses intra-communicator. Every time a process needs to communicate with another process, whether inside the group or outside the group, it calls on a set of MPI subroutines on the MPI system, by
passing it a set of arguments. The arguments would identify the destination process and the parameters to be sent etc. These are packed into an envelope and sent across to another communicator. The send and receive functions are monitored at these MPI systems. For the communication, the packets have to be packed at the processor where it is generated, transmitted over the fibers to the destination processor. It is unpacked over there, and depending on the contents, it would be retransmitted or processed. Most supercomputers are replete with such inter processor communications in order to operate as well as to carry out certain tasks. For the functioning of an MPI based processor network, several standard routines are called by the processes deployed on different processors. These routines are implemented and executed on the MPI system over and over. Speeding up this process could reduce the network latencies significantly. The processor core that is being discussed in the paper focuses on speeding up this execution.

**Figure 9 : MPI interaction between processors.**

Figure 9 shows an MPI implementation that interacts with several processes. As network bit rates increase, various computing elements serving the network will also need to increase in speed. An example of this is found in store and forward nodes seen in MPI. As shown in Figure 9 such nodes must inspect one or more incoming packets for their destinations and other information, and route them to exiting ports. If a required port is already occupied with prior routing operations, the data must be stored at that node, and later forwarded when the target port becomes free. Prioritization of message passing may require generic programmability for these functions, and this in turn implies faster instruction processing rates, and wider data paths.

Thus speeding up the MPI processing can significantly improve the multiprocessor performance. The goal is therefore to study the instructions used in the MPICH program
and implement them on a fast processor that can execute the code extremely fast. Such a fast processor would however require the 3D processor memory stack.

2.3 Memory wall:

A common problem seen in computer designs is the memory wall problem. The rate at which processor speeds have increased in the past few years is much higher than the rate at which memory speeds have increased. This difference in speed causes the processors to stall for many cycles waiting for a response from the memory. This problem is accentuated at higher clock speeds on the order of 16GHz or 32GHz that would be needed to handle applications such as Message Passing Interface (MPI) or spice. These processor speeds could be achieved in RISC processors built using SiGe Hetero-junction Bipolar Transistor (HBT) BiCMOS. To solve this memory wall problem, 3D stacking of chips and its benefits are explored in the context of microprocessors.

Figure 10: Memory wall because of processor memory performance difference.

Memory speeds are increasing at a much slower rate compared to processor speeds [3]. Figure 10 shows the rate at which the processor-memory gap is increasing over the years. The figure makes an inherent assumption that the ratio of memory size to processor size should be constant [8]. Because of the slower speeds of the memory compared to the processor, the processor stalls waiting on the memory to access some
block of data. So even though the processors are becoming faster, the overall performance is reduced because of the slower memories.

This problem has been alleviated to a certain extent by the introduction of multiple levels of cache with different sizes and speeds. Smaller caches operate faster while larger caches which can hold more data are slower. Pipelining caches in this fashion from small to large between the processor and memory has reduced the impact of memory wall on the processor performance. However a cache miss would result in severe penalties as the number of cache levels increase.

The effects of Memory Wall is seen at a larger scale when building SiGe RISC processors that can clock at frequencies on the order of 16GHz, as more clock cycles are wasted while waiting for a memory access to get through. Some strategies such as 3D memory, ultra wide bus widths etc. are explored to reduce the stall cycles observed.

2.4 Interconnect delay:
Shrinking dimensions has helped improve the performance of microprocessors by allowing more transistors and logic to be squeezed in the same footprint. As one goes from 130nm process to 90nm and then to 65nm there is more room for building complicated architectures, space for additional cores and larger on die caches. However performance advantage obtained so far has begun to hit a limiting factor as dimensions approach molecular sizes. The delay seen in global interconnect wires have not scaled with the transistor sizes as the technology nodes shrink. This is severely impacting the performance of CMOS based processors to operate beyond 4GHz. Although lithography process technology is exploring integrated circuits fabrication at smaller nodes like 45nm and 22nm, it is already examining the boundary of optical lithography. The masks needed for fabrication of chips are proving to be too expensive. One would need to shift to plate less e-beam lithography for these kinds of dimensions which at present is expensive.
Figure 11: Interconnect length comparison of 2D vs 3D.

Figure 11 shows how 3D stacking can reduce the interconnect length compared to a 2D implementation of the same circuit.
3. 3D Process

3D integration has matured over the years and has been rechristened several times from names such as giga scale integration to hyper integration. They represent several stages of development where the last term refers to the current generation involving bonding of wafers of different types of material such GaN on SiGe or Silicon wafer. The advantage of such integration has been manifold in that many technologies available on different materials can now be integrated into a single chip. 3D also has inbuilt advantages in that it is resistant to reverse engineering, which being a common problem seen by many hi-end chip designers. This chapter focuses on the various approaches involved in bonding and aligning the wafer, integration with SOI process, heterogeneous integration and finally a few applications envisaged including processor, RAM and ROM etc.

3.1 Wafer bonding:
3.1.1 Wafer direct bonding:
This constitutes of bonding two wafers without the use of any external adhesive or application of external forces. The bonding is done by the adhesion of two solids with sufficiently clean and flat surfaces. This relies on intermolecular attractive forces. The bonding so formed is reversible and requires additional steps such as annealing to strengthen the bond. Vanderwaal’s force is one such attractive force that is used to bond wafers. These bonds are formed by hydrogen bridge formation. Subsequent annealing is required to achieve much higher bond strengths. Such wafer bonding is also carried out at elevated temperatures as in the case of III-V compounds which is done in a hydrogen containing atmosphere. Alternatively, bonding in ultra high vacuum is carried out for some materials such as silicon to obtain maximum bond strength.
Wafer direct bonding requires clean surfaces. Particle contamination, organic contamination and ionic contamination put a serious limitation on the efficiency of such bonding process as these form interfacial voids or weak bonds. Flatness and smoothness of the wafers is also important as they induce a strain on the wafer if they are bonded. The other forms of direct bonding include Hydrophilic bonding, Hydrophobic bonding, Wafer bonding in UHV and Twist bonding. Hydrophilic and Hydrophobic bonding exploit vanderwaal’s force. The surface of the wafer is cleaned with strongly oxidizing solutions, to form oxide containing covalently bound hydrogen. This reacts with water to form Si-OH at the surface. This makes the surface hydrophilic as shown in Figure 13.
Figure 13: Schematic of two hydrophilic oxidized silicon wafers with mono-layers of adsorbed water: (left) at room temperature with low bond strength and (right) after annealing with high bond strength due to covalent Si-bonds at the interface.

The electrically insulating oxide formed at the interface is undesirable in certain applications. In such cases, a hydrophobic bonding approach is taken. This is done by treating the surface with diluted hydrofluoric acid or ammonium fluoride leaving a surface that is temporarily covered with covalently bonded hydrogen. Such a surface would be hydrophobic as it cannot be wetted with water. The two wafers are then bonded together using van der waal’s force and under thermal treatment the hydrogen can be removed as seen in Figure 14.
Hydrophobic bonding is more prone to contamination by hydrocarbons. In UHV bonding, the two wafers which are coated with Hydrogen are first cleaned of the hydrogen by heating in an UHV environment and then cooled to room temperature and brought in contact. This approach helps form the Si-Si bonds spontaneously across the interface at room temperature itself. Twist bonding makes use of deformities formed when two wafers are bonded and then twisted by an angle. This is primarily used certain applications involving nanotechnology.

3.1.2 Polymer bonding:
This approach to bond two wafers does not involve high temperature. The use of a glue to bond two wafers also has the advantage over direct wafer bonding in that interface voids that could be formed from contamination etc in direct wafer bonding can be reduced significantly as the polymer tends to provide a uniform interface. Figure 15 shows a schematic of polymer bonding as used in the RPI process for 3D integration.
Metallic bonding:

Two types of Cu-Cu pads are provided. One set is used for providing interconnects while the other is used to increase the bonding area. Using metal as bonding interface is an attractive choice because metal is a good heat conductor and this will help circumvent the heat dissipation problem encountered in 3-D IC. At the same time, a metal interface allows additional wiring and routing. Cu is a metal of choice because it is a mainstream CMOS material, and it has good electrical and thermal conductivities and longer electromigration lifetime. Another advantage offered by metal bonding interface is better crosstalk immunity between device layers. This is shown in Figure 16.
Figure 16: Metallic bonding showing Cu pads which are bonded together by contact and pressure.

Wafer bonding can be extended to include heterogeneous integration of wafers formed from different processes as in SiGe handle wafer with CMOS wafers or SiGe with GaN (110 with 111) for different applications. Integration of SiGe with GaN is very attractive for building power chips while SiGe with CMOS is useful for building high speed processors.

3.2 3D fabrication

Several companies are exploring the idea of stacking wafers one on top another. Researchers have grown 3D memories by depositing layers of devices in the same fashion metal layers and such are deposited on current 2D wafers [68]. They state that this approach is superior in that the cost would be cheaper to add layers to existing silicon substrate than stacking multiple chips one on top of other. However, from fabrication point of view, vias need to be provided from the top layer to the bottom layer for interconnects. The size of these vias would be large and thus reduces the circuit density. Also the complexity of the fabrication and mask design is only going to be more complex with each additional fabrication step. Others have followed the approach of
processing chips separately and stacking them [69]. The process is again different from main stream 3D in that the stacking is chip to chip. The yield is good as only known good dies are stacked on top of another. However, since this would require a standard size for the chiplets, it would not be helpful in ASIC design. Also the processing time would be significantly high if one has to fabricate chip by chip. Tezzaron[64] has a process called FaStack which takes the approach of wafer stacking connected by “super vias”. They have also built a microcontroller with SRAM on top of the processor wafer. The research in our group is in a similar direction. However we are targeting higher clock speeds on the order of 16GHz using SiGe as well as multi-core CMOS systems operating at 4GHz. Researchers have announced 3D memory that is comprised of 8 layers [8]. The individual layers are made of NAND type flash memories and hence different from typical SRAM or DRAM expected to be used in processors. These flash memories are however better suited for flash memory drives, mobile phone memories etc.
4. Processor Memory Stack Architecture

4.1 Architectural simulators:
Most architectural tools currently in use are not easily extendable to explore 3D stack architecture. Our initial architectural simulations used DineroIV [18] and CACTI[5] to study the cache memory hierarchy and the impact of bus width on CPI and cache miss. As DineroIV is a trace based simulator, it is difficult to generate traces as it is a fairly old tool. So the traces for this were generated on alternate tools such as Simplescalar [17] and converted to Dinero format. The trace file contains a sequence of instructions as it would be seen running in a single issue processor. The tool simulates the fetching of these instructions, executing then and writing back to the cache. The trace files for Dinero were generated using Simplescalar’s processor simulator program: sim-outorder. The authors have used CACTI to analyze the cache memory access time for a given size and technology node. This data along with the simulated data from Dinero was fitted into equations to evaluate the average memory access time and CPI.

Subsequent simulations were done on Simplescalar itself, which is limited to 2 levels of cache. The code was modified to incorporate a 3rd level of cache between processor and main memory to include L1, L2, L3 caches. Finally the shared memory parallel processor simulator RSIM has been used to analyze a multicore system [19][20]. RSIM’s architecture has been adapted to our analysis of a 3D multi-core processor-memory system.

<table>
<thead>
<tr>
<th>Simulators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dinero</td>
<td>Trace level simulator for cache hierarchy system</td>
</tr>
<tr>
<td>CACTI</td>
<td>Cache memory analysis tool</td>
</tr>
<tr>
<td>Simplescalar</td>
<td>Uniprocessor execution driven system simulator</td>
</tr>
<tr>
<td>RSIM</td>
<td>Multi-processor shared memory system simulator</td>
</tr>
</tbody>
</table>

Table 2: Simulator tools used for evaluations in 3D processor memory system study.

Portions of this chapter previously appeared as: Jacob, P. Zia, A. Erdogan, O. Belemjian, P.M. Jin-Woo Kim Chu, M. Kraft, R.P. McDonald, J.F. Bernstein, K., “Mitigating Memory wall in High clock rate and CMOS multicore 3D-processor memory stack”, Proceedings of the IEEE, Jan 2009, pp 108-122
4.2 Metric:
Performance of the processor can be measured by several metrics such as Clocks Per Instruction (CPI), Instructions Per Cycle (IPC), clock frequency, fabrication cost, power consumed, and area of the chip die. We have identified CPI as the best metric to evaluate the processor throughput. CPI is comprised of two components- contribution from the execution in the processor pipeline and contribution from stalling from cache misses and branches.

To compute the CPI for the execution component, we took an ideal situation where every instruction can be computed in one cycle and all memory fetches can be completed in one cycle. This CPI component is sometimes called the CPU latency component of CPI. Although the CPU is designed to execute with a peak CPI of 1.0, pipeline latencies (the inability to keep parts of the processor busy due to code dependencies) result in reduced efficiency in CPI. Running such a scenario through Simplescalar’s sim-outorder simulator gave us a CPI of 1.45. This was corroborated by the research done by Philhower [58]. The other component of CPI is known as the stall component. Some percentage of the instructions executed by any given code will result in data cache accesses, and the overall CPI of the processor depends on the cache's ability to perform the requested transaction in the time allotted. In the single issue processor with a single pipeline, whenever the primary cache needs to retrieve data from the secondary cache, the processor is stalled preventing it from accomplishing any useful work. The execution time, the time taken to execute a given program, has also been used in the analysis to study a given architecture and its performance.

4.3 Benchmarks:
To evaluate the processor memory stack architecture, the authors have used a set of programs from both SPECint and BioBench [81] – the latter being a collection of programs for bioinformatics applications. This provides a comprehensive evaluation of the design space for different types of applications.

4.4 Processor Implementation:
High clock rate processors can be implemented using Heterojunction Bipolar Transistors (HBT). Although this approach is not common in digital applications, microprocessors made before the advent of CMOS were essentially made with bipolar logic. Lately,
Bipolar, which has been developed primarily for Radio frequency applications, could stage a comeback into digital applications as they provide extremely high cutoff frequency \( f_c \) making them attractive to achieve high clock speeds for processors. With each successive generation of the technology, faster transistors are being incorporated into the technology node and there are reports of transistors that have a \( f_c \) of 500 GHz being researched [52]. With such transistors, it is easy to conceive microprocessors operating at 32 GHz and beyond. Belemjian et al. have already demonstrated a processor test vehicle [59] operating at 16 GHz using the current generation of IBM’s SiGe process. Thus with such a promising future from SiGe, the authors feel that it would provide the optimum technology for exploring the Memory Wall as would be seen in the next decade. A processor designed with HBT is also being compared with multi-core processor simulations that could be implemented in CMOS at current peak operating clock frequencies of 4 GHz.

4.5 Ultra high bus-width:

Larger buses allow the transfer of more words in one clock cycle. With 3D, designers can draw thousands of interconnections between the wafer layers without having the routing and spacing problems in 2D designs for ultra-wide buses. Bus widths in the order of 1,024 bits or more are easily envisioned. Because these vias span vertically from one layer to another, the interconnection length is also much smaller. 3D design can also support multiple buses that could connect more than one bank and hence can support even larger buses that cannot be supported in similar 2D designs. Ultra-wide buses between the processor and cache also reduce the miss rate because the processor can bring more data into the cache in one clock cycle, thereby reducing CPI considerably. Figure 17 shows the variation of CPI with varying bus widths and block sizes. The best CPI values are seen for large block sizes (1,024 bits) and large bus widths (1,024 bits), both qualities that a 3D chip would facilitate.
Figure 17: The impact of bus width and block size on CPI is examined. Increasing block size and bus width gives the minimum CPI.

Figure 18 shows results of simulation of other programs belonging to the Standard Performance Evaluation Corporation (SPEC) suite to validate the dependence of CPI on the bus width.

Figure 18: CPI is reduced by increasing the bus width between the cache levels. Initially this is done by using interleaved cache design that can typically provide bus width up to 128 bits. Beyond this 3D architecture provides ultra wide bus widths between L2 and L1 cache levels.
4.5.1 Dependence on Clock speed:

CPI values may also be calculated from the data supplied by DineroIV and CACTI to show its dependence on Clock frequency. CPI calculations for a 3 level cache system (L0, L1, L2) with each level implementing unified architecture are described below. We initially make the assumption that there will not be significant cache misses beyond L2 because the footprint of the program would be restricted to L2 cache. However, we later evaluate for much larger programs that would require embedded 3D DRAM memory in the form of additional layers in the 3D chip.

CPI calculations for a 3 level cache system (L0, L1, L2) with each level implementing unified architecture are described below. We make the assumption that there will not be significant cache misses beyond L2 because the footprint of the program would be restricted to L2 cache.

Average Memory access time is given by the following equation

\[ \text{AMAT} = \text{L0} \text{memory Hit time} + (\text{L1} \text{Fetches/instruction}) \times (\%\text{L1 cache miss}) \times \text{L1 AMAT} \quad (2) \]

Where L0 miss penalty and L1 memory access time are given by

\[ \text{L0 miss penalty} = \text{L1 AMAT} \quad (3) \]

\[ \text{L1 AMAT} = \text{L1 memory access time} + (\text{L1 total fetches /instruction}) \times (\%\text{L1 cache miss}) \times \text{L2 AMAT} \quad (4) \]

AMAT however is the time the CPU waits for a cache memory access. So the number of stall cycles can be obtained from AMAT as

\[ \text{CPI stalls} = \text{AMAT} \times \text{clock frequency} \quad (5) \]

Total CPI = CPI_{datapath} + CPI_{stalls} \quad (6)

CPI can be calculated similarly for more complex cache systems involving split caches at L0 or L1 levels. Similar equations for AMAT can be derived for a cache architecture involving split caches. For a cache architecture having split L0, L1 caches and a unified L2 cache, the equations are given by

\[ \text{AMAT} = \text{L0 Mem Access time} + \%\text{instructions} \times \text{L0 icache misses} \times \text{L0 icache miss penalty} + \%\text{data} \times \text{L0 dcache misses} \times \text{L0 dcache miss penalty} \quad (7) \]

where the L0 icache and dcache miss penalties are given by

\[ \text{L0 icache miss penalty} = \text{L1 memory access time} + \text{L1 icache miss} \times \% \times \text{L2 access time} \times \text{Bus factor} \quad (8) \]

\[ \text{L0 dcache miss penalty} = \text{L1 memory access time} + \text{L1 dcache miss} \times \% \times \text{L2 access time} \times \text{Bus factor} \quad (9) \]
Bus factor reduces the access time to a memory in the case of wider buses as it allows bringing multiple blocks every time an access is made to the L2 cache. CPI is then evaluated using Eq. 5 and 6.

At higher clock frequencies, the number of cycles spent waiting for the data accessed in the cache increases. The number of cycles spent is given by the Access time (second) x Clock frequency (cycles/second). Thus a higher value for CPI is obtained at higher clock frequencies. The performance is actually degraded from the CPI point of view by increasing the processor clock speed without decreasing the memory access time. The Memory Wall becomes larger at higher clock frequencies. This can be understood from the simulated results in Figure 19 where at low bus widths of around 64 bits, the CPI is seen to be very high on the order of 16 for a 16 GHz clock frequency.

![Figure 19: The variation of CPI is examined for varying bus width and clock frequency. Large bus widths would give lower CPIs for higher clock frequencies. A CPI of about 9 is seen at buswidth of 512 bits and 16 GHz clock frequency.](image)

### 4.5.2 Interleaved cache:

As it would be difficult to have a memory block being written in one clock cycle through a write port that is not wide enough, a bus controller would be required to write the data into the cache in multiple cycles. Alternatively, there could be an interleaved cache architecture consisting of multiple banks. Each bank has its own read/write port. This way, the whole block could be written in one cycle if there is sufficient width for the memory provided through interleaved architecture. Conventional 2D architectural
implementations of cache memory would still limit the number of bus lines that can be drawn in two dimensions. This however would be much simpler if 3D cache memory is used as upper level cache.

4.5.3 Improving L1 speeds and L2 speeds to improve CPI:
Cache access time heavily depends on the technology used. A cache implemented in a 130 nm process will be much slower than one implemented in a 90 nm or 65 nm or 45 nm CMOS process. This is the direct result of lower gate capacitances for smaller transistors as well as shorter interconnect seen when the spacing between the memory cells decrease. With the process technology improving, the use of a 45 nm or 32 nm processes for building cache memories and thereby building faster caches can be anticipated. There is still the problem of requiring large drivers for driving the wires as the wires are narrower. The authors are also exploring the possibility of implementing some levels of cache in BiCMOS. As BJT transistors are better drivers, they provide better driving capacity of the interconnect wires thus obtaining shorter access time. Also, the BiCMOS cache can be implemented in the form of multiple banks and the bank decoders can be implemented in Bipolar HBTs which can speed up the cache significantly.

Figure 20: Comparison of access time composition of L1 cache made with CMOS and BiCMOS: 1. Decoder data, 2. Word Line, 3. Sense Amp data, 4. Comparator, 5. Mux, 6. Sel Inverter, 7. Output Driver. 70 nm CMOS L1 cache access time = 0.718 ns, BiCMOS L1 cache access time = 0.431 ns
Figure 20 shows a comparison of cache access time components when implemented in CMOS and BiCMOS. Alternatively implementing the cache in purely Bipolar would result in much faster caches although lower density of the memory cells.

4.5.4 3D L2 cache memory:
In the previous sections, we have explored the advantages of having ultra high bandwidth and large amounts of cache as close to the processor as possible. For our research we are exploring heterogeneous integration of 3D memory consisting of more than one layer on top of the bottom wafer containing the processor. 3D processing has improved a lot in that it is now conceivable to bond two wafers made from different technologies including BJT to CMOS technology. So for our research, we are exploring a structure wherein we have the SiGe processor wafer bonded to CMOS SOI wafers made from the MIT Lincoln Labs process. The MIT Lincoln Labs process themselves have 3 wafers, with the bottom wafer forming the handle wafer. This handle wafer is to be replaced by the SiGe wafer. The advantage would be that with SiGe it is possible to explore higher speeds for the processor, thereby permitting analysis of upper limits of Memory Wall well before CMOS reaches those levels of performance. 3D memory also brings in other advantages such as access time reduction, reduction in chip area and cost, reduced interconnect length, and increased cache size. In order to analyze access times of 3D cache memory, the authors adapted the use of existing 2D tools such as CACTI. Taking a single layer of cache having 8 banks, it can be implemented in two layers with 4 banks each as shown in Figure 21. The access time is then calculated for 1 layer with 4 banks. The TSV interconnect delay is found to be very small for such theoretical analysis. This approach can be extended to more layers.

![Figure 21: a: Evaluating 3D cache memory access time using a 2D cache access time evaluation tool. Cache access times evaluated for 1 MB, 70 nm process with 1](image-url)
layer and 8 banks, 2 layers and 4 banks per layer, 4 layers and 2 banks per layer etc. as represented in 7b.

4.5.5 Multi tier, Multi bank, Multi-port, 3D memories:
3D memory not only provides a reduced access time through shorter interconnects, it also allows a large memory bandwidth. This bandwidth can be exploited to provide multiple ports, one per layer as in Figure 22a, or multiple ports in a given layer itself accessing different banks as shown in Figure 22b, or finally multiple ports to a given bank as shown in Figure 22c. As the number of vertical vias that can be drawn from the memory to the processor is in the order of thousands, it is easy to build architectures that would have been impossible in 2D architecture. The advantage of having such multiple ports would be exploited by multiple threads that access different parts of the memory at the same time.

Figure 22: CPU with 3D memory and multiple memory management units. a. Ports in Multitier. b. Ports in Multiple banks in same tier c. Multiple ports for same
bank. Each MMU unit can access a separate layer of the 3D memory and thus effectively increase the memory bandwidth.

4.5.6 3D Embedded DRAM:
Having a 3D L2 cache reduces to a large extent the Memory Wall between the high clock rate processor and cache. It is possible to have additional tiers on the chip by adding more wafers to the 3D chip fabrication. Since 3D allows heterogeneous integration of wafers from different processes, it is possible to integrate DRAM memory from a different process onto the existing chip. Since DRAM is much more compact, a large on-chip memory i.e. embedded memory can be implemented adjacent to the processor in addition to the main memory present on the board. This way, it is possible to get around the delay associated with having to access the main memory through the front side bus.

4.6 Performance beyond 16 GHz:
As processor speeds increase beyond 16 GHz, it is necessary to implement the processor itself in multiple layers to reduce the interconnect lengths. At these clock speeds, wire delays become very critical and hence it is important to reduce the interconnect length at every stage. Figure 23 summarizes all the strategies that are used to lower the CPI for a 32GHz core.
Figure 23: CPI reduction strategies evaluated for a 32 GHz core.

Belemjian et al. has explored the advantages of wire length shortening in implementing the processor in multiple layers as compared to 2D [59]. With the availability of faster bipolar transistors that are currently being researched upon, it is even possible to implement 32 GHz processor cores too in SiGe HBTs.

4.7 Multi-threading vs. Multi-core processors:
A multi-threaded processor running at higher speeds can offer better efficiencies since applications which are multi-threaded can be executed as multiple threads, while single thread applications can be executed as such without any deprecation in speed. The current trend in processor development has been towards slower multi-core processors, as the impact of Memory Wall is less significant. Multi-core processors also have the advantage of reusing the core design and hence are easier to implement. They also have fewer performance anomalies compared to multithreaded processors. 3D chip stacking can support both approaches.

<table>
<thead>
<tr>
<th>3D processor memory system</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>L0 cache</td>
<td>256 bytes each for dcache, icache, SiGe HBT 0.0625 ns</td>
</tr>
<tr>
<td>L1 cache - BiCMOS</td>
<td>16 kB each for dcache and icache, 70 nm, 0.431 ns</td>
</tr>
</tbody>
</table>
L2 cache: 1 MB unified cache (2 layers- 3D), 70 nm, 1.32 ns access time

L0 – L1 cache bus width: 128 bits wide

L1 – L2 cache bus width:
- 32 bits to 512 bits for single threaded
- 1024 bits for two threaded core with 2 memory management units each supporting 512 bits
- 2048 bits for two threaded core with 2 memory management units each supporting 512 bits.

Processor speeds: 16 GHz, 32 GHz

Table 3: Studying the impact of multiple memory management units and wider bus.

| Table 3: Studying the impact of multiple memory management units and wider bus. |

Figure 24: CPI variation with multithreading and buswidth evaluated at 16 GHz and 32 GHz clock frequencies. Table 2 describes the simulated system.

Figure 24 shows an analysis for multithreaded architecture with wide bus and multiple memory management units. Multi-threaded processors would have lower CPI as a 16 GHz core having two threads would be time sharing between them. So they would effectively look like two threads operating at 8 GHz each. The cache systems also see memory accesses for a particular block more spaced out in time. As long as there are multiple memory access points in the L2 cache, these threads can be seen as operating as
separate 8 GHz threads. For our design, we have context switching as a method to reduce CPI. This would require two or more sets of register files in the processor depending on the number of threads that would be supported. Whenever a given thread faces a stall due to cache miss, a context switch takes place to the next register file and the execution can continue.

Figure 25: The CPI is reduced from a high of around 23 all the way to 2.5 at 16 GHz processor speeds by systematically applying various architectural techniques. The last strategy assumes context switching with multiple memory management units. The architectural setup is as described in Table 2.

Figure 25 summarizes in the form of a bar graph the strategies that have been examined to reduce CPI for the various benchmark programs.

3D would still be beneficial in this case as multiple cores would require ultra-wide bandwidth to access the unified large cache [13]. Even if the bandwidth per core is only about 64 bits or so, the total bandwidth for 10 cores would be 10 times than that of a single core which would be impossible to implement in 2D. The trend towards multiple cores is mainly because a task that can be done at 16 GHz on a single processor can be divided over two cores running at 8 GHz or 4 cores running at 4 GHz each and so forth. Thus the Memory Wall problem can be avoided yet giving the same throughput. However, currently implemented architectures are not flexible enough to handle most
programs like Spice, MPICH etc. that are not written for multiple cores. In such cases, the performance takes a hit when a certain process has to be executed as fast as possible.

For multi-core implementation, we have analyzed a simple architecture wherein each core would have its own L0 and L1 cache and a common 3D memory lying on top of it. Each layer of the 3D memory can be assigned to a particular core. As the design is not limited by the number of layers for the 3D memory, such a configuration can be thought of as two uniprocessors operating in parallel executing their own individual programs. The programs considered so far from the SPEC suite are not easily parallelizable either. So for simulations, we have assumed a multi-programmed workload environment where there is no sharing of cache or inter-processor communication. The effective CPI of running two such programs on two cores would be the average of the CPIs running independently on individual cores.

![Execution time for 6 spec programs](image)

**Figure 26:** Execution time of 6 programs when executed on a single core, multiple cores (2, 4) and a 16, 32 GHz processors designed in the preceding sections (with 3D memory and BiCMOS memory and context switching). The impact of bus width is explored in the case of multi-core processor. The architectural setup is described in Table 2.
Since we have simulated for the same number of instructions, the effective CPI would be the (avg CPI)/(number of cores). For the simulations, we have considered 6 SPEC programs (perl, go, mk88, lisp, vortex, jpeg) each having 10M instructions totaling 60M instructions being executed on each type of processor (1, 2, 4) CMOS cores as well as on 16, 32 GHz SiGe cores. The multi-core processors are assumed to execute at 4 GHz as contemporary CMOS processors. Figure 26 shows the relationship of the number of cores to execution time in the presence of 3D memory given a wide bus between the L2 and L1 cache. This is compared to the execution time of a SiGe processor operating at 16 GHz and 32 GHz with a BiCMOS L1 cache, 3D memory and context switching. The performance of a 4-core processor is seen to be comparable to that of the SiGe processor. The advantage obtained from ultra-wide bandwidth is partially hidden by the BiCMOS cache, 3D memory and context switching.

The high speed processors such as 16 GHz and 32 GHz, show better performance advantages in terms of execution time for cases where the programs considered do not parallelize.

4.8 3D Shared memory system:
The previous system of multi-core processors with a 3D cache operated with accesses to the 3D cache memory being independent with no memory access conflicts and to separate layers. However practical systems would behave more like a shared memory system with different processes sharing the memory system. It is necessary to implement cache coherent protocols to maintain valid data in all the cache banks. Such a system can be simulated by considering a shared memory multiprocessor simulator such as RSIM. Figure 27 shows the RSIM architecture that has been adapted to a multi-core 3D memory system.
Figure 27: RSIM architecture adapted to 3D multi-core processor memory stack.

The simulator provides the option to implement cache coherence protocols such as MESI (Modified, Exclusive, Shared, Invalid) and MSI (Modified, Shared, Invalid) protocols. These represent the status of each cache line. The simulations here are carried out with the MSI protocol to maintain coherence between the different memory banks. In order to broaden this aspect, we carried out simulations with RSIM to evaluate the multi-core processors with programs that can be parallelized. Figure 28 shows the simulated results with an FFT program and its benefits evaluated for both increasing the bus width and increasing the number of cores. The architecture implementation is described in Table 4 below.

<table>
<thead>
<tr>
<th>3D multi-core processor memory system</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulator adapted to 3D setup</td>
<td>RSIM–x86</td>
</tr>
<tr>
<td>Number of cores/ nodes</td>
<td>4, 16, 64</td>
</tr>
<tr>
<td>L1 cache</td>
<td>16 kB each for dcache and icache, 70 nm, 0.71 ns</td>
</tr>
<tr>
<td>L2 cache</td>
<td>1 MB unified cache (2 layers- 3D), 70 nm, 1.32 ns access time</td>
</tr>
<tr>
<td>bus width</td>
<td>128-1024 Bits</td>
</tr>
<tr>
<td>Processor speeds</td>
<td>4.0 GHz, CMOS (multiplication factor of 15)</td>
</tr>
</tbody>
</table>

Table 4: Shared memory – multi-core processor setup for 3D architecture.
Figure 28: A multi-core system with 3D memory simulations to show reduction in execution time using RSIM simulator. For the simulations an FFT program is being programmed as a 4-way, 16-way and 64-way process is used.

The above results also confirm Amdahl’s prediction that without the code being sufficiently parallel, the benefits obtained from going from 16 core to 64 core would be marginal [28].

4.9 Summary:
Our research has explored various strategies by which it is possible to reduce the CPI for processors that run at high clock speeds such as 16 GHz. The graphical results show that 3D integrated chips can help overcome the Memory Wall. As CPI is seen to be directly proportional to clock, increasing clock frequencies to say 32 GHz or 64 GHz would again make the CPIs unmanageable. At this stage, embedded 3D DRAM memory would be required too. The analysis then continued to explore the benefit of 3D memory over multi-core processors. The CPI metric is ideal for uni-processor evaluations but doesn’t reflect the exact performance in a multi-processor environment. For such a multi-core processor, the performance is measured by the execution time instead of CPI. Figure 12 shows a histogram where a multi-core processor is visualized running programs that are single threaded in nature and hence cannot be parallelized.
<table>
<thead>
<tr>
<th></th>
<th>Base</th>
<th>3D processor memory system</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>L0 cache</strong></td>
<td>none</td>
<td>L0 Bipolar cache, 128 bytes each for dcache and icache, SiGe HBT</td>
</tr>
<tr>
<td><strong>L1 cache</strong></td>
<td>L1 CMOS 70 nm, 16 kB each for dcache and icache, 70 nm, 0.718 ns</td>
<td>L1 BiCMOS cache, 16 kB each for dcache and icache, 130 nm, 0.431 ns</td>
</tr>
<tr>
<td><strong>L2 cache</strong></td>
<td>L2 CMOS cache, 1 MB unified 2D cache, 70 nm, 2.05 ns access time</td>
<td>L2 3D cache, 1 MB unified cache (2 layers- 3D), 70 nm, 1.32 ns access time</td>
</tr>
<tr>
<td><strong>Main Memory</strong></td>
<td>off-chip</td>
<td>3D DRAM memory on the processor</td>
</tr>
<tr>
<td><strong>Total Instructions</strong></td>
<td>100M</td>
<td>100M</td>
</tr>
<tr>
<td><strong>Processor Speed</strong></td>
<td>32 GHz</td>
<td>32 GHz</td>
</tr>
</tbody>
</table>

Table 5: Architecture setup for 3D vs. 2D base processor comparison.

Table 5 describes the architecture setup for comparing the two 32 GHz processor systems – one built as a 2D conventional processor and the other with multiple features associated with 3D memory and BiCMOS cache and the results are shown in Figure 29. The simulations were carried out on Simplescalar. The code was modified to include a 3rd level of cache in the case of the 3D system. The simulations were extended to include Biobench benchmarks in order to widen the simulation design space.
Figure 29: CPI has been evaluated for a base system and compared with a 3D processor memory stack described in Table 4, for SPECint and BioBench programs.

Memory Wall problems would need to be addressed immediately for multi-core or high clock rate processors. In the case of multi-core processors, as the number of cores increases beyond 6 to 8 cores in one chip, the performance will not increase proportionately due to insufficient memory and bandwidth problems. For high clock rate processors, the SiGe process technology is already well established for RF circuits. It’s only a matter of the industry adopting it into mainstream processor design. As the future of microprocessors unfolds, whether as high clock speeds or multi-cored processors, 3D integrated chips will be an essential requirement. Additional cache techniques such as dynamic data pre-fetching to avoid cache pollution will have to be considered for smallest and fastest cache, which is L0 in our case, to enhance the performance [23].
5. Processor architecture:

The processor core is being implemented using Silicon Germanium (SiGe) Heterojunction Bipolar Transistors (HBT). Since these transistors have high \( f_t \) compared to CMOS or Si based BJTs, they are broadly used in both analog and digital applications where high frequencies of operation are required. There are different levels of integration such 5HP, 7HP, 8HP in the IBM BiCMOS process. The \( f_t \) of the transistors is highest for 8HP at 210GHz, at 120GHz for 7HP process. Future technologies may have an \( f_T \) as high as 375 GHz. Since these transistors have such high \( f_t \), it would be ideal technology to make extremely high speed digital circuits. 7HP IBM SiGe BiCMOS process having matured is chosen as the technology for implementation of the processor core. The technology would also benefit from the interconnection advantages of 3D. It has been observed that circuits made from bipolar devices are roughly 2.5 times faster than comparable CMOS circuits, assuming that the same level of photolithography is used in both technologies. In conjunction with their high speed the IBM BiCMOS SiGE process offer relatively good yields compared to other bipolar processes and good current driving capability. They do not include new low-k dielectric materials, but do use Cu for a minimized interconnection related delay. The yield of the process enables one to make more complex circuits, which may be incorporated in a microprocessor. This extra complexity allows the 3D processor – memory test vehicle to contain interconnection lengths which approximate some of the critical paths in more realistic processor cores.

The SiGe HBT’s speed, yield, linear characteristics and high gain make it very suitable for many applications. The current applications of SiGe are mainly in the telecommunications industries where high frequency analog and mixed signal circuits are used. As a consequence, contemporary SiGe technology has been somewhat optimized for analog applications. The original IBM SiGe HBTs were designed for mainframe computers. The contemporary SiGe HBT satisfies a blend of analog and digital requirements, but its readily available speed advantages made it a prime choice for our study.
5.1 Architecture of the processor:

Shown in Figure 30 is the architecture of the 3D processor memory system envisioned.

Figure 30: A 3D processor memory system architecture that is proposed.

Altera provides a CAD tool called Quartus. This tool allows one to implement the core using existing blocks of cores such as memory, logic gates, multiplexers, ALUs etc. These blocks have been connected together to form the processor core as shown in Figure 31.
The implementation details of the individual blocks are explained in subsequent sections. From the architecture shown above, it is possible to see the flow of instructions from the instruction cache to the write-back stage. The instruction cache feeds 4 instructions to the instruction queue. The queue feeds 1 instruction per clock cycle to the instruction decoder. Depending on the type of instruction (Load/store, Branch, ALU), the decoder instructs the register file to decode the source, destination addresses in the instruction word. The operands obtained from the register file are fed into the operand preparation circuit which along with the adder forms the ALU. Upon completion of the ALU operation, the data is written back into the register file. Details of handling of each instruction are explained in the next section.

5.2 Instruction set and their handling:
The instructions that are supported by the architecture include two memory operations (Load, store), 8 ALU operations as supported by the 74381 operand preparation circuit, 1 Branch instruction and a NOOP. The instruction is 32 bit long. The bits are segregated in the following manner-

Op code: 4 bits
SRC reg: 3 bits (to identify one of 8 locations in the register file)
16 bit address that can be used for LOAD, STORE and BRANCH instructions.

**LOAD instruction**: This instruction has the following format
LD SRC, Dest address

The Destination address here is a 16 bit address. However the address size seen in accessing memory is 32 bit. So the instruction handles this by adding this 16 bit prefixed by 16 zeros to a base register contents in the adder. The resulting address is passed to the cache memory. Since a load instruction results in waiting for the cache contents to return, the pipeline controller is triggered to stall the pipeline. When the cache returns the data to the waiting load instruction, it is written to the register file and thus the data is available for the following ALU instructions.

In conventional processors, these delays from load and store are hidden by performing out of order execution and non blocking execution i.e. allowing other instructions which needn’t access the memory to continue its execution.

**STORE instruction**: This instruction has a similar format to LOAD.
ST SRC, Dest address

The SRC register contains the data that needs to be stored in the destination address. Again the destination address is calculated similar to the load instruction address by adding the 16 bit address to the contents of a base register. This 32 bit address along with the contents of the store data obtained from the register file is written to the data cache.

**BRANCH instruction**: A generic branch instruction has been provided. At this stage its functionality is simple in that no branch condition is associated with it. The instruction simply forces the branch to a different address which is calculated by adding the 16 bit address to the base register contents. This new address is fed to the program counter that is remotely located in the instruction cache.

**ALU instructions**: This includes the instructions supported by 74381 in addition to 32 bit adder/ subtractor. The details of the 74381 operand preparation chip are studied in a separate chapter.

The ALU instruction has the format
Op code SRC1, SRC2, DEST
<table>
<thead>
<tr>
<th>Opcode</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>NOOP</td>
<td>No operation</td>
</tr>
<tr>
<td>0001</td>
<td>CLR</td>
<td>Clear</td>
</tr>
<tr>
<td>0010</td>
<td>ST</td>
<td>Store</td>
</tr>
<tr>
<td>0100</td>
<td>BR</td>
<td>Branch to target address</td>
</tr>
<tr>
<td>0101</td>
<td>SUB</td>
<td>A MINUS B</td>
</tr>
<tr>
<td>0110</td>
<td>LD</td>
<td>Load</td>
</tr>
<tr>
<td>0111</td>
<td>ADD</td>
<td>A PLUS B</td>
</tr>
<tr>
<td>1001</td>
<td>XOR</td>
<td>A XOR B</td>
</tr>
<tr>
<td>1011</td>
<td>OR</td>
<td>A OR B</td>
</tr>
<tr>
<td>1101</td>
<td>AND</td>
<td>A AND B</td>
</tr>
<tr>
<td>1111</td>
<td>PRESET</td>
<td>Set</td>
</tr>
</tbody>
</table>

Table 6: Opcode Descriptions

The first stage of the processor is the instruction decoder. It receives a sixteen bit address and a sixteen bit instruction. The address is simply passed onto the next stage of the pipeline while the instruction is broken down into separate components and used to generate control signals. The first four bits of the instruction make up the opcode. The opcodes and their descriptions can be seen in Table 6. The fourth bit of the opcode is always high for ALU operations. This is done to easily generate an ALU operation control signal that can be used in the later stages of the processor. When executing a ST or LD -- functions that require the ALU’s ADD operation -- logic is performed on the first three bits of the instruction before being passed to the next stage. The remainder of bits in the instruction are broken down into source one, source two, and destination.
Figure 32: Instruction decoder schematic

5.3 Simulation:
5.4 Register File:

The register file is adapted from the 3 stage pipeline shown in Figure 35. The first stage (Address Decode, AD) decodes the addresses to be used in writing and reading the contents of the register file. The second pipeline stage (MEMory operations, MEM) deals with the memory writing and reading, as well as an initial selection process. In order to reduce the latency any overlapping reading addresses and writing address is compared in the AD pipeline, and it is fed to the second pipeline stage with a match signal stored in AD stage’s pipeline. This match signal selects between the memory bank output and the data fed into the register file to be written. After this selection process, the correct data is stored in the MEM pipeline register. The third stage shown in the Figure 45 is removed and adapted to the processor design. This stage is to support multithreading access to the same register file. Instead future designs would have multiple separate register banks for each thread.
The other way to fill the register contents is by write back during the WB2 stage. After the execution of the instruction the WB2 sends the result and the corresponding register to the register file. The WE (write enable) is kept active whenever something needs to be written to the register file.
5.5 **Operand preparation stage:**

The operand preparation circuits is an adaptation of 74381 chip that is available off the shelf. The chip performs 8 types of ALU operations. This in combination with the full adder forms the complete set of ALU instructions. The implementation details of this block are detailed in a separate chapter.

The Pre ALU stage, simply called *ALU stage 1* in the design, serves as an intermediary step between instruction decoding and ALU operations. In this stage, the proper data is selected for ALU manipulation.

As made evident by Figure 36 below, certain control signals simply pass through the stage. Buffers were placed on these lines to mitigate potential skewing between the control signals and multiplexed data. The lines being passed through the stage are described in Table 7 below.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESTr</td>
<td>3-bit bus containing the address to which ALU operations will be written</td>
</tr>
<tr>
<td>ALU_SEL</td>
<td>3-bit bus determining which operation is used in the ALU stage</td>
</tr>
<tr>
<td>NOP</td>
<td>No-op command</td>
</tr>
<tr>
<td>LOAD</td>
<td>Load command</td>
</tr>
<tr>
<td>STORE</td>
<td>Store command</td>
</tr>
<tr>
<td>BR</td>
<td>Branch command</td>
</tr>
<tr>
<td>DATA_A1</td>
<td>32-bit bus containing data from the registry to be used in a STORE</td>
</tr>
</tbody>
</table>

**Table 7: Pre ALU Control Lines and their Function**

The main functionality of this stage is to select the proper data to be manipulated for the forthcoming ALU stage. Should a LOAD or a STORE command enter the PRE ALU stage, the selection bit on both multiplexers becomes logic high, multiplexing the second data input. This causes the base address to be outputted on the DATA_A_OUT bus, and the instruction address to be outputted on the DATA_B_OUT bus. DATA_A_OUT and DATA_B_OUT are sent to the ALU stage to be added together. This is a necessary procedure to provide the needed address offset for LOAD and STORE commands.
Conversely, if an ALU function enters the stage (non LOAD/STORE) the multiplexers are set to select the first data input. This places the data from the register file held on inputs DATA_A1, and DATA_B1 onto the output buses DATA_A_OUT, and DATA_B_OUT respectively. These output buses later serve as the operands in the ALU stage.

![Diagram of Pre ALU stage design]

**Figure 36: The Pre ALU stage design**

Inputs were chosen to test all aspects of the Pre ALU stage. LOAD and STORE instructions were tested as were ALU instructions. As seen by the simulation diagram below (Figure 8), the Data of DATA_A_OUT alternated between, input DATA_A1.
(READ_OUT_A in the simulation) and 0x00003A98 (the base register in hex) as LOAD/STORE and ALU operations entered the simulation. Moreover, DATA_B_OUT alternated between DATA_B1 (READ_OUT_B in the simulation) and the data of the ADDR bus. After a series of additional simulations, the Pre ALU stage was deemed fully functional and ready for integration.

![Simulation Waveforms](image)

**Figure 37: Simulating the Pre ALU**

### 5.6 ALU

The ALU is capable of executing the logical instructions (AND, OR, XOR), ADD and SUB as well as Branch condition instructions. The result is selected using a 4 input mux based on the instruction that is being executed. The branch signals are passed on to the next stage.

The Arithmetic Logic Unit (ALU) does the computation for the processor. The $ALU_{Sel[2:0]}$ signal decides what operation is to be performed while 74381s (4 bit ALU chips inside the $op_{prep}_{32bit}$ block) do the logical calculations. Addition and subtraction are done using a 32 bit adder ($adder_{32}$). All other signals coming to this
stage are propagated through to the output via buffers. Given below is the schematic diagram of ALU.

Figure 38: ALU Schematic Diagram

The adder block of the ALU is implemented as a Carry look ahead adder. Blocks arranged in a pseudo-carry look-ahead tree as shown in Figure 39.
Figure 39: Carry look ahead adder structure.

The ALU is then simulated on quartus and the simulation results validating the structure are shown in Figure 40.
The results are tabulated below in Table 8.

<table>
<thead>
<tr>
<th>ALU_SEL</th>
<th>DATA_A</th>
<th>DATA_B</th>
<th>OPERATION</th>
<th>ALU_OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>75757575</td>
<td>40414243</td>
<td>CLR</td>
<td>00000000</td>
</tr>
<tr>
<td>010</td>
<td>75757575</td>
<td>40414243</td>
<td>SUB</td>
<td>35343332</td>
</tr>
<tr>
<td>011</td>
<td>75757575</td>
<td>40414243</td>
<td>ADD</td>
<td>B5B6B7B8</td>
</tr>
<tr>
<td>100</td>
<td>75757575</td>
<td>40414243</td>
<td>XOR</td>
<td>35343736</td>
</tr>
<tr>
<td>101</td>
<td>75757575</td>
<td>40414243</td>
<td>OR</td>
<td>75757777</td>
</tr>
<tr>
<td>110</td>
<td>75757575</td>
<td>40414243</td>
<td>AND</td>
<td>40414041</td>
</tr>
<tr>
<td>111</td>
<td>75757575</td>
<td>40414243</td>
<td>PRESET</td>
<td>FFFFFFFF</td>
</tr>
</tbody>
</table>

Table 8: ALU Simulation Results

These simulation results show that the ALU works satisfactorily for all the instructions. Several simulations were done with different data values but only one is shown in order to maintain conciseness.

5.7 Write back stage and post processing stage:

This stage consists of a store queue that is used to write back the evaluated contents in a register to the d-cache. The stage also handles write back of the result data back to the register file.
Figure 41: Write back stage logic

5.8 Finite State machine (Pipeline controller):

The finite state machine is being designed as a separate block that would generate a set of output signals for each state that the CPU is in. The CPU can operate in 5 states—program start, Normal operation, Stall operation, Pipeline reload and stop. Initially when the CPU starts running the instruction set, the state changes to Normal and when it hits a stall either from cache miss or Branch it moves to Stall state. If the stall is from branch, the CPU moves to next state of pipeline reload to refresh the pipeline with a fresh set of instructions, and then return back to Normal state. If the stall is from cache miss, it could wait for the cache block to be available or alternatively tell the CPU to switch to a different thread to continue its operation. When the CPU has completed all its operations, it would move to the stop state. This state machine could provide a useful technique to watch the CPI or branch frequency of the whole program that is being tested by just watching the frequency of occurrence of certain state signals in the program execution. This block is described in more detail in a subsequent chapter.
The implementation might involve implementation of a control store which would take in a set of input variables and send out a set of output signals to control the pipeline.
5.9 Core testing:

To test the core, a 25 instruction long program was prepared that would examine all the testing cases and logic instructions. The initial part of the code tests the various logic instructions while the last part of the code tests the branch instruction operating in a loop. The instruction set and output is shown in Table 9 below.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Instruction</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOOP</td>
<td>0000 0000</td>
<td>F</td>
</tr>
<tr>
<td>LD R6 &lt;- mem(100)=A</td>
<td>6600 0000</td>
<td></td>
</tr>
<tr>
<td>CLR R7 &lt;- 0</td>
<td>1700 0000</td>
<td>0</td>
</tr>
<tr>
<td>LD R0 &lt;- mem(109)=1</td>
<td>6000 0009</td>
<td></td>
</tr>
<tr>
<td>LD R4 &lt;- mem(105)=5</td>
<td>6400 0005</td>
<td></td>
</tr>
<tr>
<td>Preset R5 &lt;- FFFFFFFF</td>
<td>F500 0000</td>
<td>F</td>
</tr>
<tr>
<td>LD R6 &lt;- mem(112)=D</td>
<td>6600 0000</td>
<td></td>
</tr>
<tr>
<td>ST mem(255) &lt;- R6 =A</td>
<td>2060 009B</td>
<td></td>
</tr>
<tr>
<td>ADD R1 &lt;- R0+R0 =2</td>
<td>7100 0000</td>
<td>2</td>
</tr>
<tr>
<td>SUB R2 &lt;- R4-R0 =4</td>
<td>5240 0000</td>
<td>4</td>
</tr>
<tr>
<td>CLR R5 &lt;- 0</td>
<td>1500 0000</td>
<td>0</td>
</tr>
<tr>
<td>LD R6 &lt;- mem(255)=A</td>
<td>6600 009B</td>
<td></td>
</tr>
<tr>
<td>XOR R3 &lt;- R4 XOR R6 =8</td>
<td>9346 0000</td>
<td>8</td>
</tr>
<tr>
<td>ADD R1 &lt;- R1+R7 =2</td>
<td>7117 0000</td>
<td>2</td>
</tr>
<tr>
<td>ADD R0 &lt;- R0+R7=1</td>
<td>7007 0000</td>
<td>1</td>
</tr>
<tr>
<td>NOOP</td>
<td>0000 0000</td>
<td></td>
</tr>
<tr>
<td>AND R5 &lt;- R6+R7 =0</td>
<td>D567 0000</td>
<td>0</td>
</tr>
<tr>
<td>OR R4 &lt;- R6 OR R4 =F</td>
<td>B446 0000</td>
<td>F</td>
</tr>
<tr>
<td>ADD R2 &lt;- R2+R7 =4</td>
<td>7227 0000</td>
<td>4</td>
</tr>
<tr>
<td>ADD R1 &lt;- R1+R7 =2</td>
<td>7117 0000</td>
<td>2</td>
</tr>
<tr>
<td>ADD R0 &lt;- R0+R7=1</td>
<td>7007 0000</td>
<td>1</td>
</tr>
<tr>
<td>ADD R1 &lt;- R1+R7 =2</td>
<td>7117 0000</td>
<td>2</td>
</tr>
<tr>
<td>ADD R2 &lt;- R2+R7 =4</td>
<td>7227 0000</td>
<td>4</td>
</tr>
<tr>
<td>ADD R3 &lt;- R3+R7 =8</td>
<td>7337 0000</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 9: CPU testing code
Simulation results of the core testing are shown in the waveform below. The outputs are also observed on the board Cyclone 3 shown in Figure 45.

Figure 43: CPU simulation
Figure 44: Altera cyclone 3 board
6. 3D Memory Design

6.1 Introduction:
3D memory was fabricated on this research project using the MIT Lincoln Lab process which is a 180nm FDSOI process. The process provides integration of 3 wafers to form a 3 tier structure.

6.2 Fabrication steps:
The fabrication steps starts with 3 fully depleted Silicon on Insulator wafers that are fully processed. Wafer 2 is flipped over wafer 1 and they are bonded face to face through oxide bonding. The handle silicon of the second wafer is now etched away with the oxide layer used as an etch stop. Through silicon vias are now etched into the wafer through the oxide. This is shown in step 2 and 3 of Figure 45 below. Following this, wafer 3 is flipped and bonded face to back with the previous bonded wafers. And the process of etching away the handle wafer and drawing TSVs is repeated.

Figure 45: 3D FDSOI fabrication process
The MITLL process uses tungsten for TSVs as they have shown to have better reliability. Also the thermal coefficient of expansion of tungsten is comparable to that of
Silicon. Previous attempts to use Cu TSVs resulted in copper vias popping out as Cu had a thermal coefficient of expansion that was 4 times that of tungsten. So the use of heat for oxide bonding or heat from circuit operation etc. resulted in stress on the silicon oxide bonds with the expansion of Cu vias.

6.3 FDSOI vs. PDSOI

Silicon on Insulator provides two forms of transistor technology: Fully Depleted SOI (FDSOI) and Partially Depleted SOI (PDSOI). The difference between the two SOI technologies is in the thickness of the depletion region of FDSOI vs. PDSOI. FDSOI has a thinner top layer resulting in the channels being fully depleted. PDSOI on the other hand has a bit of bulk under the channel. The two technologies have their own advantages and disadvantages. Figure 46 shows the schematic of transistors built in the two technologies. Table 10 lists the advantages and disadvantages of these two processes.

![Figure 46: FDSOI vs. PDSOI transistor](image)

<table>
<thead>
<tr>
<th></th>
<th>FDSOI</th>
<th>PDSOI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design compatibility</td>
<td>Similar to bulk</td>
<td>Additional circuit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>techniques</td>
</tr>
<tr>
<td>Floating Body effects</td>
<td>Moderate</td>
<td>Significant</td>
</tr>
<tr>
<td>Manufacturability</td>
<td>Additional processing</td>
<td>Similar to bulk</td>
</tr>
<tr>
<td></td>
<td>steps</td>
<td></td>
</tr>
<tr>
<td>Speed</td>
<td>Moderate</td>
<td>Faster</td>
</tr>
</tbody>
</table>

Table 10: 3D Cache memory architectural features
6.4 3D Memory chip design

A comparison is made between a two dimensional layout vs. three dimensional layout in Figure 47. The 2D structure has 3 banks in the same plane while the 3D structure has the 3 sub-banks one on top of the other. An access time comparison is done between these two structures. It can be seen that between the two structures, the decoder and sense amplifier have the same delays. The latency has reduced marginally for the wordline and bit line as seen from the shorter wire interconnects obtained from 3D chip design. This suggests that the gain from 3D memory comes from the wider bandwidth that can be obtained from the large number of thru-silicon vias and not from the latency improvements. The other advantage is that a larger memory can be laid out in the same given area.

![Figure 47: 2D SRAM vs. 3D SRAM Test Structures](image)

![Figure 48: 3D L2 cache Memory Access time: 1-Decoder; 2-Wordline + Wordline_driver; 3-Bitline; 4-Sense_Amp.](image)
A simple cache was designed with a size of 192KB. The cache is divided into 4 banks. Each bank has 4 ways. Each “way” has a size of 12k. Each layer of the 3 tier sub-array has 4k. Each subarray is formed by 128 bit lines x 256 rows in a given tier thereby forming 128*3 bit lines per 3 tier sub array. When all 4 banks are taken into account, this can translate to a maximum bandwidth of 1536 bits lines.

**Figure 49: 3D cache – Floor plan & Microphotograph**

Figure 49 above shows the floor plan and the chip microphotograph of the 3D memory chip. The microphotograph shows the 3D vias in the center of the chip. This is placed in a way that it overlaps on the junction between L1 cache and the processor core.
7. Thermal modeling of 3D processor memory stack

3D chips require extensive thermal analysis to analyze power and thermal dissipation distribution. Active circuits such as microprocessor components as in ALU and register files tend to dissipate more heat than passive blocks like memory causing the formation of hot spots. The heat generated remains trapped especially in 3D chips. This heat if not removed could result in increased junction temperatures of the transistors resulting in reduced performance. Heat generated from bottom layer could also conduct to other tiers in the stack. This chapter explores strategies to spread the heat generated from the hot spots and remove it from the chip. Thermal analysis has been carried out on the floor plan based on actual building blocks that have been previously designed. The 3D chip considered has the bottom layer fabricated in SiGe BiCMOS technology, and the CMOS 3D SRAM is stacked on top to obtain better performances.

Thermal analysis is a critical step in the design and implementation of 3D chips. Without effective removal of heat from the circuit, any advantages gained in implementation of 3D circuits over the corresponding 2D circuits would be lost. 3D chip stacking typically involves wafer bonding of multiple wafers and thereafter dicing them into individual chips. Because of the loss in surface area exposed to the heat sink, one needs to come up with innovative means to spread the heat and thereby reduce the power density in the circuit. Heat spreading by use of diamond is studied for its effectiveness. Variation in the thickness of the wafers used is also studied for dissipation characteristics.

[41] focuses on uniform dissipation of heat in the logic layer bonded with memory. However 3D circuits such as processor with 3D memory stacking, would have hot spots in certain regions and hence the importance of spreading the heat generated in these hot spots is paramount. [41] also is primarily focused on interconnect losses instead of evaluating power density peaks as would be the case when the lithography shrinks.

Currently available tools for Finite Element Method Analysis include ANSYS, COMSOL-, AMG(algebraic multigrid) solvers, Geometric multigrid GMD solvers, ATAR, Patran. There are other spice based models such as Thermodel.
7.1 3D chip stack:
3D chip stacking provides potential benefits in performance improvement such as faster clock speeds, ultra wide bus-width between the tier using the large number of vertical vias between the wafers. These advantages are particularly useful when trying to build microprocessors that clock at high speeds and hence is starved of data arriving from the slower cache memory. [12] describes these performance advantages that can be gained from 3D chip stacking. However, if appropriate thermal modeling is not carried out, the heat generated from the “hot” processor that forms the bottom layer of the wafer stack, can end up slowing not only the processor but the 3D memory too as heat conducts through the vertical vias into it. This would kill any speed gain that one could have obtained from the ultra-wide bus widths that 3D chip stacking provides. Figure 50 shows a reduction in performance of a register file as the ambient temperature of the chip is raised through simulations.

Figure 50: Reduction in performance with increasing temperature of chip.

7.2 3D chip model:
Figure 51 shows the 3D block floorplan of the chip stack that is being studied for thermal dissipation. The bottom wafer forms the SiGe wafer processor that is designed to operate at high clock rates. The bottom tier consists of the CPU, L0 cache and L1 cache. The 3D L2 cache memory is stacked on top of this tier.
Figure 51: 3D floorplan that is used for thermal analysis. 2 layers of memory are stacked on the CPU core.

7.3 CPU core Floor-plan and power estimates:
For a detailed analysis the CPU floor-plan is laid out based on the dimensions of existing blocks. The power consumption of each block is listed in Table 11.
Figure 52: Floor-plan of processor core

In order to analyze the thermal dissipation estimates, the authors evaluated the power dissipation for each of the block forming the processor core by calculating the total current consumption and operating voltage. Since the bottom tier processor core is designed in CML, the only power dissipation is from static power. CML only steers a constant current around in the current trees, thereby producing a constant power consumption at all times. This power dissipation is directly related to the thermal dissipation.

<table>
<thead>
<tr>
<th>Blk No.</th>
<th>Name</th>
<th>Length</th>
<th>Breadth</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Instruction Queue</td>
<td>600u</td>
<td>1250u</td>
<td>1.4w</td>
</tr>
<tr>
<td>2</td>
<td>Pipeline register</td>
<td>346u</td>
<td>800u</td>
<td>0.4w</td>
</tr>
<tr>
<td>3</td>
<td>Adder</td>
<td>620u</td>
<td>900u</td>
<td>2.5w</td>
</tr>
<tr>
<td>4</td>
<td>Pipeline register</td>
<td>346u</td>
<td>800u</td>
<td>0.4w</td>
</tr>
</tbody>
</table>
Table 11: Power dissipation of the processor core and the dimensions of the blocks

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th><em>t</em> (μs)</th>
<th><em>n</em> (μs)</th>
<th><em>W</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Register File</td>
<td>1250</td>
<td>1300</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>Pipeline register</td>
<td>346</td>
<td>800</td>
<td>0.4</td>
</tr>
<tr>
<td>7</td>
<td>Write/Store Queue</td>
<td>1250</td>
<td>600</td>
<td>1.4</td>
</tr>
<tr>
<td>8</td>
<td>Pipeline register</td>
<td>346</td>
<td>800</td>
<td>0.4</td>
</tr>
<tr>
<td>9</td>
<td>Pipeline Controller</td>
<td>600</td>
<td>1400</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>Serdes</td>
<td>1875</td>
<td>950</td>
<td>2.5</td>
</tr>
<tr>
<td>11</td>
<td>L0 dcache1 Register File</td>
<td>1250</td>
<td>1300</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>L0 dcache2 Register File</td>
<td>1250</td>
<td>1300</td>
<td>5</td>
</tr>
<tr>
<td>13</td>
<td>L0 dcache3 Register File</td>
<td>1250</td>
<td>1300</td>
<td>5</td>
</tr>
<tr>
<td>14</td>
<td>L0 dcache 4 Register File</td>
<td>1250</td>
<td>1300</td>
<td>5</td>
</tr>
<tr>
<td>15</td>
<td>L0 icache 1 Register File</td>
<td>1250</td>
<td>1300</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>L0 icache 2 Register File</td>
<td>1250</td>
<td>1300</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Total power</td>
<td>5000</td>
<td>5000</td>
<td>41.5</td>
</tr>
</tbody>
</table>

7.4 Thermal studies:

Thermal studies were carried out using a finite element analysis program- COMSOL. The program is fed with parameters to identify the dimensions of the blocks, the material properties and the power dissipation estimates. The finite element analysis is then run to evaluate the heat dissipation pattern, thereby identifying the hot spots. For the first analysis, we evaluated just 1 tier consisting of the SiGe processor core. The peak temperature is seen to occur mainly on the register file blocks used as L0 cache components. Since the substrate is not a good conductor, a thick substrate doesn't allow proper heat dissipation into the heat sink (Cu chuck used for die testing).
Figure 53: Thermal dissipation of CPU chip placed on 1mm Cu. Chip thickness= 2000um

7.4.1 Wafer Thinning:
The finite element analysis was then extended to carry out simulations for a thinner wafer. One can see a reduction in peak temperatures seen as one reduces the thickness.

Figure 54: Thinning down the Si wafer to 2000um to 25um allows the heat source in the circuit layer to be brought closer to the heat spreader. This allows better heat spreading and thereby lower peak temperature in the hot spots.
7.4.2 Impact of these hot spots on devices placed on top:
Placing a MOSFET device on top of a hot spot causes the junction temperature in the device to increase. For a MOSFET, the junction temperature is given by

\[ T_j = T_a + P_{chip} \times R_{ja} \]

where \( T_j \) is the junction temperature, \( T_a \) is the ambient temperature and \( R_{ja} \) is the thermal resistance.

\[ R_{ja} = \frac{1}{4K(L \times W)^{0.5}} \]

where \( L \) and \( W \) specify the dimensions of the chip block. \( K \) is the thermal conductivity of Si. The increased temperature source tends to degrade both performance of the MOSFET in terms of speed as well as long term reliability. The high temperatures tend to result in metal failures resulting from electro-migration, as well damage to the weak oxides.

![Figure 55: Single MOS device places on top of a hot spot. 800Si on 1mm Cu](image)

7.4.3 Diamond sheet heat-sink:
Diamond has a much higher thermal conductivity than Silicon. Thus reducing the Si substrate thickness and bringing it closer to a better heat spreader as in a Diamond, one can easily remove any tent pole formation in the heat spots. This has been modeled using COMSOL and shown in Figure 56 below.
Figure 56: 25um Si, 50um Diamond sheet 1mm Cu heatsink. Thermal conductivity of Si: 100W/K, Diamond thermal conductivity: 2000W/k, Cu plate conductivity: 400 W/k

7.4.4 3D stack thermal modeling:

Having modeled with a Single MOSFET on top of a hot spot, the authors proceeded to model 3D memory stacked on the processor. In the model considered below, the bottom SiGe processor core has 3D L2 cache (2 tiers) stacked on top. The substrate thinned die is bonded to diamond. Although there will be some loss in thermal conductivity due to kapitsa resistance which be seen at the bonding interface between Silicon and Diamond [al-joseph], one still obtains reasonable heat spreading properties.
One can consider additional more complicated stack involving a diamond wafer stacked in between the hot processor core and the 3D memory. This allows better heat spreading as well as preventing peak temperatures in the core affecting the overlying 3D memory as the thermal sources are spread out better. Figure 58 shows the peak temperature as low as 309°F for such a model structure.

Figure 57: 25um Si, 50um Diamond sheet 1mm Cu heatsink

Figure 58: 25um Si, 50um Diamond sheet 1mm Cu heatsink with intermediate heat sink
Figure 59: 3D chip bonded to diamond wafer for better heat spreading. Through vias are made in the bottom SiGe wafer to the diamond sheet.

7.5 Thermoelectric cooling:
Peltier effect can be used effectively to remove heat from the hot spots in the IC. This thermoelectric phenomena works on the peltier cooler principle where current is passed through the device to remove the heat from the hotter spot to the colder region when a temperature gradient is provided across the device.

\[ Q = \alpha T_1 I - I^2 R - K(T_2 - T_1) \]

This of course results in increased power consumption.

7.6 Stress analysis impact on 3D vias connecting chip tiers:
Stress analysis also needs to be done on a chip with heterogeneous integration and non uniform thermal expansions. Of particular interest is the impact on the die and the via chains when the expansion rates are different. As shown in Figure 60, shear strain happens when the top layer expands outward while the lower layer doesn’t.

Figure 60: Horizontal displacement of the wafer tiers resulting in shear stress on 3D via
7.6.1 Horizontal and Vertical normal strain:
The other types of strains that can result on the vias are horizontal / normal strain. This will be seen when the wafers are pulled apart either because of expansion of the bonding material or non-uniform bending.

![Strain mechanisms](image)

Figure 61: Strain mechanisms seen on wafer bonding especially on the interconnect via.

7.7 Multi-core processors with 3D memory:
While the study here was focused on SiGe HBTs based microprocessor, it can be very well extended to CMOS based multi-core processors. As the industry is currently focusing on increasing the number of cores to improve the performance, one is likely to see a shortage in memory bandwidth as the number of cores integrated in one chip increases. This would force the industry to use 3D memory stacked on top of the bottom multi-core processor. As the lithography shrinks to accommodate more processors, one is also essentially increasing the power density in the chip. Even though CMOS may dissipate lower power compared to SiGe BiCMOS HBTs, but at 45nm or lower, the amount of circuitry integrated into a tiny die is so large that the power dissipated per unit area is comparable to the levels of the high speed core considered in this paper. Hence the thermal analysis studies carried out can be extended to multi-core processors with 3D memory to obtain similar results.
8. SiGe HBT Device

8.1 Introduction:
The idea of mixing Ge in Si to form an alloy and be used in transistors has existed for several years. But the difficulty of growing a SiGe epitaxy that is lattice matched with underlying Si made the SiGe transistors a reality only as recent as in 1990s. The original SiGe process was developed to cater the digital world for building high speed computers. But because of the large power consumptions, this later became a predominant technology in RF circuits. As the frequency of operation of the transistors increased, it gradually displaced several other technologies like GaAs in RF communications. The basic operational theory of HBTs i.e. having traditional wide band gap emitter plus narrow band gap base approach seen in III-V HBTs as well as the graded base approach used in SiGe fabrication were pioneered by Kroemer. Current SiGe devices include carbon doping to suppress boron out diffusion during fabrication. The idea of using Ge in Si was extended to FETs as well as other devices.

8.2 SiGe Device:
A simple structure of the SiGe HBT device is shown in Figure 62a. While most of the structure is similar to a regular bipolar transistor shown in Figure 62b, the base of the transistor is a very thin layer.

![SiGe HBT structure compared with Si BJT structure](image)

Figure 62: SiGe HBT structure compared with Si BJT structure

Figure 63 shows the doping profile of the wafer. The first part of the region is doped with P type dopant As. This is followed by n- type base doping Boron. The base also
contains graded doping of Ge to form the SiGe epitaxy. The collector is doped p type as seen with Phosphorous doping.

Figure 63: Doping profile of the wafer for a SiGe HBT device

8.3 Device physics:
In the device structure the Ge content in the base is graded from 0% to a high value near the collector base junction and then rapidly brought down to 0% again. The resultant energy band diagram is shown in the Figure 64 below. The dotted profile represents the case for SiGe HBT while the strong line represents a Si BJT. The grading of the Ge across the neutral base induces a built in quasi drift field in the neutral base induces a built in quasi drift field in the base. The injection of Ge into the base region reduces the potential barrier to injection of electrons from emitter into the base. Thus there will be more electron injection for the same applied Vbe, This gives one a higher collector current and hence higher current gain. The presence of the Ge induced drift field across the base will also accelerate the injected minority electrons across the base. This reduces the base transit time. Also the presence of finite Ge content in the CB junction will positively influence the output conductance of the transistor, yielding higher Early voltage. The smaller base band gap near the CB junction effectively weights the base profile, such that the backside depletion of the neutral base with increasing Vcb (Early effect) is suppressed compared to a comparably doped Si BJT. This translates to higher Early voltage compared to Si BJT.
Figure 64: Energy band diagram of a graded base SiGe HBT compared to Si BJT

8.4 Device Properties:
SiGe device exhibits high $g_m$, low $NF_{min}$, very low power dissipation at $NF_{min}$. They also have low $1/f$ noise corner and phase noise. These characteristics make it the ideal choice for mixed signal circuits. The suppression of the back channel depletion results in very high output resistance and high $\beta VA$ product. This is an important figure of merit that most analog designers would be keen on achieving. Since the transistors exhibit extremely high cut off frequencies, this can be traded with lower power consumption circuits. The device has high power gain and good linearity and can be operated at both extremes of temperature. Since the SiGe base is very thin, it has built in total dose radiation tolerance. The fact that the device is Silicon based process; it can be easily integrated with existing CMOS devices.

Strain in the device channel, improves the mobility by modifying the energy band structure as well as increasing the lattice constant. Strain can be introduced in the channel by simply bending the wafer.
Figure 65: Strained devices formed by bending of the processed wafer. 1a: Fully processed IC wafer that has been thinned. 1b: Mechanical stress applied on the wafer and subsequently bonded to polymer carrier. 1c: Wafer bonded to permanent quartz wafer to retain stress.

Figure 65(a) shows a fully processed IC wafer that has been thinned sufficiently to enhance elasticity. A mechanical stress is then applied to incorporate a stress in the directions required. The wafer is then bonded to a polymer carrier as shown in Figure 65(b). This is then wafer bonded to a more permanent structure as a quartz wafer to retain the mechanical stress. The advantage of this approach to introduce strain is that it is not process limited and hence not limited by feature sizes of the devices either. The mechanical strain can be applied in any direction required. The other approach to
introduce strain is by actually modifying the processing steps to include a layer of SiGe in the device. The formation of such a layer introduces a mismatch in the energy band structure. Such structures are therefore called heterostructures because of the formation of separate layers of Si and SiGe. These structures also have a mismatch in the lattice size, causing a strain in the upper layer. Figure 66 shows a Si lattice being grown on a relaxed SiGe. As the lattice constant of SiGe is larger, the Si grown on top tends to match its lattice with the underlying layer, thus forming a strain in the lattice. This strained lattice increases the space available for the electrons to move through the lattice and thus improves the mobility.

![Diagram](image)

**Figure 66:** Growth of Si on SiGe results in strained lattice in Si layer.

### 8.5 Device fabrication:

#### 8.5.1 Epitaxy:

SiGe devices are grown by using SiGe epitaxy on Si substrate. Several methods to grow the epitaxy exists—such as by RTCVD, MBE, UHV/CVD. The critical thickness of the epitaxy and its thermal stability cause severe limitations for the integration into Si process technology. The deposition of SiGe layers requires low-temperature process technique. Hetero epitaxy by CVD is generally used as the proven process for the growth of SiGe layers. In Rapid thermal CVD (RTCVD), the process temperature can be
switched rapidly. This allows one to minimize the thermal budget. In this technique, a stable gas flow is maintained over the wafer at low temperature and deposition is switched on or off by rapidly heating and cooling the wafer. MBE method allows the growth of crystalline silicon germanium layers with well-defined dopants by very low growth temperatures. In the case of SiGe MBE surface segregation is observed for alloy atoms and dopants like boron, phosphorus or antimony. An MBE machine generates molecular beams of matrix material such as silicon or germanium, and doping species and their interaction with the substrate surface to form a single crystal deposit under UHV conditions. In UHV/CVD, deposition of Si and SiGe from chemical vapors is based on the thermal decomposition of the appropriate compounds- (SiH4, Si2H6 etc) or chlorosilanes and germanes (GeH4, Ge2H6) or chlorogermaines.

![SiGe Transistor](image)

**Figure 67:** Combining low temperature epitaxial techniques with ultra high vacuum chemical vapor deposition, IBM scientists have thinned the SiGe HBTs vertical transport layer by half to push its cut off frequency to 210GHz in 8HP technology.

There are 3 basic classes of bipolar structures: non–self aligned, self aligned epitaxial base transistor with inner side wall. And self aligned with outer sidewall.

### 8.5.2 SiGe HBT self aligned:

Self aligned structures are formed by either using an inner wall on the emitter region or an outer wall. In the inner wall based self aligned structures, a double polysilicon structure is used. Figure 67 shows the process mechanism for the growth of such transistors.

The other category of self aligned HBTs makes use of a outer sidewall on a sacrificial structure. The advantage being that the growth and deposition are non-
selective and the structures have low topography. Figure 68 shows a schematic for such a growth technique.

8.5.3 SiGe HBT non self aligned (BiCMOS):

This structure is the simplest epitaxial base bipolar structure and thus has simpler processes. The process begins with the formation of the buried layer into a CMOS compatible P substrate by patterning an arsenic implant. The buried layer implant is then followed by a high temperature anneal. This is done with an oxidizing implant to drive the implant deep into the silicon and thus consume the damaged silicon layer. The lightly doped n-epitaxial layer is then grown on this at high temperature. The base CMOS process is then carried out. After the CMOS process, the etch that protects the bipolar regions is removed. Then an in situ doped SiGe (with Carbon doping) is grown. Single crystal material is then grown in the region of exposed silicon and polysilicon over the regions where dielectrics are exposed. Landing pads for metal contacts are made in the regions where polysilicon growth takes place outside the active regions. A high quality emitter base dielectric is formed by oxidation. This is followed by the deposition of thicker dielectrics. The emitter opening is formed by patterning and etch step. The emitter polysilicon is deposited, doped, capped with a dielectric layer, patterned and etched. The extrinsic base is then implanted using the emitter poly-layer as a mask for the extrinsic base implant. The extrinsic base layer is then patterned and etched. A dielectric deposition and etch forms a sidewall on both emitter polysilicon and the base polysilicon layer. CMOS implants, an RTA anneal and Ti silicide formation completes the front end of the line process.
8.6 SiGe HBT on SOI:

IBM has manufactured a SiGe HBT on SOI process. This allows easy integration of HBTs with CMOS built on SOI. This also had the advantage of reducing power consumption significantly. The process steps involved in developing this structure is shown in Figure 69.
Figure 69: Fabrication steps for SiGe HBT on SOI

The device has a raised extrinsic base to kill of the bleeder paths associated with the Van Der Ziel lateral base pushout effect. The effect causes small emitter stripes fringing current paths to travel way out through the extrinsic base to the collector and these longer paths slow the transistor down as the emitter stripe width is scaled down. The raised base on oxide blocks these bleeder paths very well, enabling much lower
operating currents. Raised base introduces high resistivity which hurts Rb. So CoSi2 is slathered all over the base. This is shown in Figure 70.

![Figure 70: SiGe HBT on SOI](image)

**Figure 70: SiGe HBT on SOI**

**8.7 Comparison of 5HP, 6HP, 7HP, 8HP:**

Two common figures of merit associated with SiGe transistors are the unity current gain cut off frequency $f_t$ and the unity power gain frequency $f_{max}$. Over the several generations of processing viz. 5HP, 6HP, 7HP and 8HP, these frequencies has been increasing. The Cut off frequencies were at 50GHz for 5HP and 6HP, 120GHz for 7HP and 210GHz for 8HP. The processing lithography has also shrunk from 0.5um for 5HP, 0.25um for 6HP, 0.18um for 7HP all the way to 0.13um for 8HP. In the next generation, the transistors are projected to go up to 350GHz $f_t$ in 90nm process. Figure 8 shows the current vs Cut off frequency curves for the different SiGe technology nodes.
Figure 71: Unity gain cut off frequency and Unity power gain frequency variations for different technology nodes. (a) shows variation of $f_t$ for different transistor sizes for 5HP node. (b) shows variation of $f_t$ and $f_{max}$ for different transistor cross sections for 8HP. (c) shows variation of $f_t$ and $f_{max}$ for the next generation with transistors having $f_t=350\text{GHz}$. (d) shows technology node variation.

Figure 72 shows the trend in which $f_t+f_{max}$ increased across several generations. The current generation is marked as the 3rd generation.
8.8 Summary:
21st century communications market is demanding higher frequency bands. Such high speeds require faster transistors. While GaAs filled this market, it was too expensive and the yield was also low. SiGe HBTs being Silicon based process, gave higher yields and also allowed integration with CMOS easily. Also SiGe devices themselves are undergoing rapid process development with speeds of the transistors doubling every other year. Cressler’s group in GaTech is working on HBT devices that operate at 500GHz at liquid helium temperatures and they have also reported simulated devices in 1THz range [4]. In the immediate future the development of SiGe on SOI has been a significant step in that the devices can now be integrated with CMOS on SOI. This helps reduce power levels as well as increase the cut off frequency to 350GHz.
9. CML design with SiGe

The Current Mode Logic is so named as current is steered through the logic to obtain different logic functions. CML logic can be both CMOS and Bipolar based, although one typically uses bipolar. Emitter Coupled Logic is similar to CML except that it has emitter followers at the output to allow for better driving capabilities. CML logic uses a differential signal to switch a pair of transistors connected as in differential amplifiers. The transistors are however biased to operate in digital mode i.e. Saturation and cutoff in the case of MOSFETs and Active, cutoff for Bipolar based logic. The primary advantage of this logic is the high switching speeds possible. This is so because the swing voltages of the output and input are on the order of 150 to 200mV. However this comes at a price of higher power consumption because of the constant path between the power and ground. Also the ability of the circuit to drive multiple gates requires large emitter followers which to consume power. The logic can introduce multiple levels of differential signals to create more complex gates with multiple inputs. This chapter explores the basics of CML based design that are used extensively in the design of all circuits in our research group. Bipolar transistors are used preferentially over MOSFETs as they have higher cut-off frequencies which in turn yield faster circuits.

9.1 CML Logic cells

The simplest of the CML logic constitutes a buffer shown in Figure 73. The T1 transistor acts as a constant current source MOSFET. The transistor is biased in the saturation region for this purpose. The logic is fed by differential digital signal in and in_b. Depending on which signal has the high signal, the corresponding transistor is switched on (active) while the other transistor is turned off. Thus a current path is formed between the Vcc and ground through the active transistor. The amount of current flowing is determined from the resistance used and the voltage levels of the MOSFET transistor. When the ‘in’ signal goes high, the corresponding transistor is ON and thus the collector output of that transistor is pulled low. On the other hand the transistor that is fed in_b is turned OFF and thus its collector remains high in this case at Vcc. One can thus see that the outputs have to be taken from the opposite branch. One main advantage of CML logic is that a signal and its complementary are generated at the same time.
Thus to invert the output signal, one only needs to switch the output terminals and take out\_b instead of out.

Figure 73: CML buffer with gated voltage reference.
The gated voltage reference is primarily used to turn off certain blocks of the circuit when not in use for saving power.

Figure 74: Addition of emitter followers at the output of CML buffer to form ECL.
Figure 74 shows a modified buffer with emitter followers to form the ECL logic. ECL logic are better at driving output signals.
9.2 Voltage levels:

Having multiple levels in a CML allows for multiple inputs in a gate thereby permitting more complex gates. However, this increases the voltage level of the power supply too and thus the net power consumption. Typical circuits could be 2-level logic which uses a 2.5 volt power supply or a 3 level logic which uses a 3.4 voltage supply. All the design used for CPU uses 3.4 voltage level thus permitting 3 levels of inputs stacked one on top of the other.

The three levels of signals are shown in Table 12.

<table>
<thead>
<tr>
<th>Level</th>
<th>Voltage values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>0 to -250mv</td>
</tr>
<tr>
<td>Level 2</td>
<td>-950mv to -1.250mv</td>
</tr>
<tr>
<td>Level 3</td>
<td>-1.95mv to -2.25mv</td>
</tr>
</tbody>
</table>

Table 12: Voltage levels used in 3-level CML design

There is a difference of 0.7V between each voltage level which corresponds to a diode voltage. Also some margin of voltage is accounted for the constant current source MOSFET at the bottom of the tree.

9.3 Logic Gates

9.3.1 Emitter Followers:

One may use emitter follower level shifters shown in Figure 75 as a separate logic element. When multiple voltage levels are used, it is necessary to shift signals from one voltage level to another. This is done by the use of emitter followers especially when a signal needs to be shifted from a higher level to lower level. To shift from lower level signal to higher level, one makes use of a buffer instead.
9.3.2 2 input cells (NAND, OR, AND, NOR)

All the logic functions NAND, AND, OR, NOR have the same structure. The output varies the way the input is connected. The structure shown below is for NAND gate. When A and B are high, the output $Z_1$ is pulled low. For all other input combinations, $Z_1$ is high. This corresponds to a NAND gate. Swapping the output lines of $Z_1$, $Z_1b$ gives one the logic for AND gate.

An OR gate can be formed from the same structure by swapping the inputs A and A-bar and doing the same for B and B-bar. This effectively makes use of Boolean logic where in A-bar.B-bar = A + B. Thus the output at Z1-bar with the switched inputs would correspond to OR gate. Switching the Z output results in NOR gate.
As the logic is at 3.4 volts and only two inputs are used, a dummy diode connected transistor is used to for the third level.

Figure 76: Basic logic cell that can be used for implementation of 2 input AND, NAND, NOR, OR gates.

9.3.3 3 level 3 input cells (NAND, AND, NOR, OR)
The logic can be explained in a similar fashion to the 2 input logic explain above. The only difference being that the bottom third level diode connected transistor is replaced by a differential pair.
Figure 77: Basic logic cell that can be used for implementation of 3 input AND, NAND, NOR, OR gates

9.3.4 XOR (XNOR)

Figure 78: Logic cell that can be used for implementation of XOR and XNOR gates
XOR gates are frequently used in digital logic for implementing circuits such as comparators, phase detectors etc. The schematic shown in Figure 78 is intuitive in the way it functions. Assigning various combinations of the truth table, one can verify that the outputs from the logic match that of an XOR gate. A XNOR gate can be obtained from the same schematic by swapping the output terminals.

9.3.5 Multiplexer / De-multiplexer

![Image of multiplexer circuit](image)

**Figure 79: Logic cell that can be used for implementation of Multiplexer**

A multiplexer is often used to select one of two signals using a control signal. One typical application that is commonly used in our research is to select between an external clock and internal VCO. It is also used extensively in the data path to select different data words. A de-multiplexer on the other hand retrieves both the signals that were originally multiplexed. The activated pair of transistors which are fed by A or B depends on the control signal Φ. The inactivated pair leaves both the output nodes at a high voltage.
9.3.6 D Latch:

The D flip-flop forms one of the most frequently used logic block in designing sequential circuits such as state machines as well as high speed memory functions such as queues, register files and shift registers. A D flip flop is formed from two D latches (Figure 81) connected in master-slave configuration. The slave latch is fed with the inverted clock as shown in Figure 82a.
Figure 81: Logic cell that can be used for implementation of D latch

9.3.7 D FlipFlop
b).

**Figure 82:** a. D flip-flop formed from D-latch b. Combined Schematic of D flip-flop.

### 9.4 Summary:

CML logic thus can be used to form all circuits that are generally used in digital circuits. For high speed operations, it should be ensured that the transistors stay in the active or cut off regions for bipolar transistors. A saturated transistor is slower as it takes more time to clear the base charge. Multiple voltage levels are handled which complicates the design to some extent as one need to add level shifters to connect from one gate to another. Also the power supply voltage requirements are also much higher at 3 V or higher to supply a 3 level signal. This when compared to a 1 V CMOS design is significantly higher. The power dissipation is also higher for bipolar. However, the speed advantage that can be gained from bipolar compensates for these negative aspects of CML design and one can use this logic for specific applications in digital circuit where speed is the primary criteria.
10. Operand Preparation circuit based on 74181 and 74381

An Arithmetic Logic Unit is a combinational logic that performs a set of arithmetic instructions such as addition, subtraction, logical operations such as OR, AND etc. The ALU can thus be divided into two parts- the operand preparation circuits and the Adder/Subtractor unit. The operand preparation circuits implements the function select inputs that decides the operation to be performed. This chapter discusses the implementation of the operand preparation circuit which together with the adder implemented by Belemjian would constitute the ALU. The circuit has been adapted from 74381 logic chip that are commonly available. The 74381 is itself a modified version of the 74181.

10.1 Instruction set:

74181 is a 4 bit ALU that has 4 select inputs which can be used to select 16 different operations. The operations implemented by 74181 are listed below

<table>
<thead>
<tr>
<th>S3</th>
<th>S2</th>
<th>S1</th>
<th>S0</th>
<th>Function M=0 (arithmetic)</th>
<th>Function M=1 (logic)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>F=A minus 1 plus CIN</td>
<td>F=A’</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>F=A . B minus 1 plus CIN</td>
<td>F=A’+B’</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>F=A . B’ minus 1 plus CIN</td>
<td>F=A’+B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>F=1111 plus CIN</td>
<td>F=1111</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>F=A plus(A’+B’) plus CIN</td>
<td>F=A’ . B’</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>F=A.B plus (A+B’) plus CIN</td>
<td>F= B’</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>F= A minus B minus 1 plus CIN</td>
<td>F= A ^ B’</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>F=A+B’ plus CIN</td>
<td>F=A ^ B</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>F= A + B’ plus CIN</td>
<td>F = A + B’</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>F=A plus (A+B) plus CIN</td>
<td>F = A’ . B</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>F= A plus B plus CIN</td>
<td>F = A^ B</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>F= A . B’ plus(A+B)plus CIN</td>
<td>F = B</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>F=A plus A plus CIN</td>
<td>F= 0000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>F=A. B plus A plus CIN</td>
<td>F = A . B’</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>F= A. B’ plus A plus CIN</td>
<td>F= A . B</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>F=A plus CIN</td>
<td>F= A</td>
</tr>
</tbody>
</table>
Table 13: List of functions implemented in 74181 4 bit ALU.
Some of the functions listed about in 74181 are almost never used, but are obtained anyway the way the logic is connected. 74381 encodes its select signals more compactly and provides only eight different but useful functions.

10.2 Reduced Instruction set:
A smaller set of instructions are implemented for the actual processor core. The 74381 chip is examined in this regard which supports 9 instructions which are listed in Table 14. The corresponding select signals and their output are shown below.

<table>
<thead>
<tr>
<th>S2</th>
<th>S1</th>
<th>S0</th>
<th>ALU</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>CLEAR</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>H</td>
<td>B MINUS A</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>L</td>
<td>A MINUS B</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>H</td>
<td>A PLUS B</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>L</td>
<td>A xor B</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>H</td>
<td>A + B</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>L</td>
<td>AB</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>H</td>
<td>PRESET</td>
</tr>
</tbody>
</table>

Table 14: List of functions implemented in 74381 - 4 bit ALU.
74381 also provide the group carry look ahead generate and propagate signals. These can be used for implementing the CLA adder. The 74381 operand preparation circuit along with the CLA as implemented in the dissertation by Belemjian will form the complete ALU. The complete block diagram of the 74381 is shown in Figure 83. The schematic can be divided into the following sub-sections for better understanding viz. D signals, ABSelect, F0, F1, F2, F3, P & G. Each of these sub-sections is explained later.
Figure 83: Gate level schematic diagram of 74381
10.3 Implementation

10.3.1 Dsignals:
This block converts the 3 bit select into one of the 8 select signals. So it effectively functions as a 3-8 decoder. Table 9 lists the 3 select signals S0, S1, S2 which are used to decode the ALU operation that is being performed. These select signals are determined in the previous stage of instruction decoding. Figure 84 shows the layout of the 3-8 decoder / DSignals block. At the time of design, DTI sharing wasn’t allowed. This affected the spacing and wire lengths to be longer than required.

Figure 84: Layout of block D signal select shown in Figure 67.
Figure 85 and Figure 86 show a test bench and simulation result that validates the circuit.
Figure 85: Testing Dsignal logic

Figure 86: DSsignal output validation - D4, D7 output
10.3.2 ABSelect, ABSelect

The two blocks ABSelect and ABSelect' are almost similar except one logic gate (3 input vs 2 input). Hence the layouts are almost identical. These blocks take in the inputs $a_i$ and $b_i$. The outputs are fed to next stage gates F0, F1, F2 and F3 as shown in Figures 87-91. These gates produce the final 4 bit pattern.

Figure 87: Layout of ABSelect
10.3.3 F0, F1, F2 and F3 blocks

Figure 88: F0 block layout

Figure 89: F1 block layout
Figure 90: F2 block layout

Figure 91: F3 block layout.
10.3.4 P & G

Figure 92: P & G terms that could be used for CLA.

10.3.5 Operand block:

Figure 93: 4 bit operand preparation layout
Figure 94: Simulation set up for 4 bit operand preparation block

10.3.6 16 bit 74381

Figure 95: Layout of operand preparation
10.4 Droop Analysis

Given the problem of wires behaving more like resistors, there is a voltage drop of the power lines from its source pads to the actual circuit. It is there for essential to analyse the drop in voltage across the chip. Running the tool voltage storm on the chip allows analysis of both the Vee and gnd power lines. The color code can be set to have different colors for different voltage drops. Figure 97 show the droop analysis carried out on the chip and the results validate that the power distribution has been properly done. Improper distribution could result in voltage mismatches between different logic levels at different parts of the chips. As the noise margins are very low, these mismatches can affect the performance of the chip severely.
Figure 97: Droop analysis for Vee and gnd!

Figure 98: Color coding for droop analysis

Figure 99 shows the chip microphotograph of the operand preparation block. The pad layout is shown in Figure 100.
Figure 99:Operand Preparation chip microphotograph

<table>
<thead>
<tr>
<th>D1</th>
<th>S0</th>
<th>S1</th>
<th>Ve</th>
<th>Vc</th>
<th>S2</th>
<th>D7</th>
<th>Vc</th>
<th>Ve</th>
<th>A0</th>
<th>B0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trig</td>
<td>F0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vee</td>
<td>F1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vcc</td>
<td>Vee</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td>Vcc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B1</td>
<td>F2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vcc</td>
<td>F3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vee</td>
<td>Vcc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AB</td>
<td>Vee</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SET</td>
<td>Clk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Trigger</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 100:Operand Preparation pad layout.
11. Pipeline Controller State Machine

The pipeline controller is a finite state machine, responsible for generating the control signals for the pipeline stages. The table below describes the inputs and outputs to the pipeline controller.

<table>
<thead>
<tr>
<th>SIGNAL NAME</th>
<th>TYPE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET</td>
<td>Input</td>
<td>Start Operation</td>
</tr>
<tr>
<td>HLT</td>
<td>Input</td>
<td>Stop the operation and clear the pipelines</td>
</tr>
<tr>
<td>STALL_CACHE</td>
<td>Input</td>
<td>Wait for the data coming from the memory (for LOAD)</td>
</tr>
<tr>
<td>UNSTALL_CACHE</td>
<td>Input</td>
<td>Data has come, resume operation</td>
</tr>
<tr>
<td>CLEAR_BRANCH</td>
<td>Input</td>
<td>Clear the pipelines (for BR, it clears the instructions following BR)</td>
</tr>
<tr>
<td>CLK</td>
<td>Input</td>
<td>Global Clock</td>
</tr>
<tr>
<td>CLEAR</td>
<td>Output</td>
<td>Consists of Clear_1 and Clear_2345, used to clear pipelines</td>
</tr>
<tr>
<td>FREEZE</td>
<td>Output</td>
<td>Freezes the operation in case we need to wait for memory</td>
</tr>
</tbody>
</table>

Table 15: Signals of the Pipeline Controller

The state machine has four states – PWR, EXEC, FRZ and CLR. The description of these states is as follows:

- **PWR** – This is the Power-Up state. When a HLT (Halt) input comes, the controller goes into this state and remains there until it receives a SET signal. During this state, the controller sends out a CLEAR signal to all the pipelines.
- **EXEC** – This is the normal execution state. The processor resumes its normal operation when it is in this state, until it receives a HLT, STALL_CACHE or a CLEAR_BRANCH input.
- **FRZ** – The controller enters this state when it receives a STALL_CACHE signal. This tells the controller to wait until the memory sends the data requested during a LOAD operation. The controller cannot come out of this state unless an
UNSTALL_CACHE pulse is received. The controller sends out FREEZE signal when in this state.

- CLR – When a BRANCH occurs, the controller is informed through the signal CLEAR_BRANCH. The controller responds by entering the CLR state and clears the pipeline so that the new instruction (according to the target address) can enter the pipeline and the old results are not written back into the register file or memory. The controller sends out a CLEAR signal when in this state.

11.1 State Machine:
The state transition diagram depicted below in Figure 101, illustrates the operation of the pipeline controller.

Now derived, are the State Equations. The derivations were carried out by following the standard procedure of making a circuit diagram. Three intermediate signals (X, Y, Z) are generated based on the five inputs to the controller. If the number of gates in series (critical path) seems to restrict the speed performance of the processor, these signals will help insertion of another pipeline in the middle of the controller. The table 16 given below shows how these signals are generated.

![Figure 101: Pipeline controller state transition diagram](image)
From the above table, equations are derived to find the following signals:

\[ X = (\text{SET})' (\text{HLT})' (\text{ST\_CACHE})' (\text{CLR\_BR})' (\text{UNST\_CACHE}) \]

\[ Y = (\text{UNST\_CACHE})' (\text{ST\_CACHE} \oplus \text{CLR\_BR}) + \]

\[ (\text{UNST\_CACHE})(\text{ST\_CACHE})' (\text{CLR\_BR})' \]

\[ Z = (\text{UNST\_CACHE})' (\text{SET})' (\text{CLR\_BR})' (\text{ST\_CACHE} \oplus \text{HLT}) \]

The columns 000, 001… etc are the values of XYZ respectively. Now, the next state equations can be derived from the state transition table using K-Maps. The results are:

\[ A_1 = B_0 X Y Z' + A_0 Y Z' + A_0 X' Z' + X' Y' Z' B_0' + A_0 B_0' X' Y \]

\[ B_1 = X' Y' Z' + B_0 X Z' + A_0' B_0 X' Y + A_0 X' Y Z + A_0 B_0' Y Z' \]

Both A and B are the state variables. Now the controller was simulated for its correctness as it is the instrumental part of the whole design. The next section describes the simulation procedure and results.
11.2 Verification

Figure 102: Pipeline controller simulation

As shown by Figure 102 above, the pipeline controller successfully transitions to all of its states. The simulation began with a HLT signal; this signal correctly outputted a high CLEAR_2345. When the pipeline controller receives the SET signal set the controller in motion. Afterwards, when the Stall_Cache is received, the Freeze_b output goes low signaling the pipelines to freeze. The controller, properly, stays frozen until the Unstall_Cache signal is received. Finally, as the Clear_Br signal is received, meaning a BRANCH instruction is present, the CLEAR_2345 signal goes high, clearing the pipelines.
Figure 103: Block schematic of Pipeline controller chip

Figure 104: Simulation of finite state machine used in pipeline controller
11.3 Chip Design:
The chip has been divided into 4 blocks: Finite state machine which controls the various states; Pipeline register which separate the different stages of a proposed processor; Buffer stages which act as dummy stages between the pipeline registers; 3 bit counter which feeds as input to the first buffer stages. Figure 105 shows the layout of the Finite state machine that was designed in section 10.1 and 10.2.

Figure 105: Layout of Finite state machine

The state machine forms the heart of the processor as it decides how the pipeline operates—whether to stall it (in case of cache miss) or clear it (in case of branch) or to resume normal operation (when the stall or branch is resolved). Current design requires this decision to be made in one clock cycle. It would be essential to pipeline this design further if higher clock rates are required. Simulations on altera have shown that the state machine is currently limiting the maximum possible clock frequency that can be achieved.
Figure 106: Layout of the 32 bit D Flip flop (pipeline stage) and buffer array.

Figure 107: Complete layout of state machine and pipeline stages
11.4 Chip testing:
Simulation results of the chip schematic were carried out. The counter output is fed into the pipeline/buffer stages and the output is collected on pins Z31, Z30 shown in the pad layout Figure 110. The state machine can control the flow of this data depending on the output of pipeline clear and freeze. The clock to the pipeline stages is implemented as an AND output of external clock and stall_b from state machine.

Simulations results shown in Figure 109 shows that there is a setup time required for the D flip-flop pipeline register before clock input is fed into the pipeline stage. This proves to be a limiting factor on the maximum possible frequency.
Figure 109: Simulation of pipeline stage output

Figure 110: Pad layout for pipeline controller chip
Figure 110 above shows the chip’s pad layout. The bottom probe configuration is the power probe. The left and right probes are signal probes. The power probes can sink in a current of 0.5A per Vcc/Vee pair. The signal probes on the other hand can sink in 0.25 A per Vcc/Vee pair. Figure 111 below shows the chip layout for pipeline controller finite state machine.

Figure 111: Microphotograph of fabricated chip: pipeline controller
12. Heterogeneous Core with 3D Memory

12.1 Introduction:
Recent advances in microprocessor design have been predominantly focused on homogeneous multi-core architectures in order to increase throughput of the programs as well as to keep pace with Moore’s law of increasing the transistor count. However this trend doesn’t alleviate the performance of code that cannot be parallelized. Performance improvement in multiple core microprocessors is limited by Amdahl’s Law. Consequently it is prudent to pursue a balanced technology approach, in which both parallelizable and serial codes enjoy speed improvements. This calls for heterogeneous core integration of a fast core that accelerates serial code with other CMOS cores that execute parallel operation. Scaling of MOSFETs has also reached the molecular boundaries where a few dopant atoms have to control the threshold voltage of the transistors resulting in increased leakage power. Clock rates for CMOS based designs have tended to saturate due to wire scaling problems and excessive heat dissipation. Alternate technologies for wires as well need to be considered to go beyond CMOS scaling. Industry has relied on increasing the number of processor cores to increase performance. Part of this approach has relied on Moore’s law [1] to increase circuit density to increase number of processor nodes, and memory per node while lowering the power. Increases in clock speeds were enabled by scaling. The progress was codified in the Dennard’s scaling rules [2], which linked device, voltage, and interconnection scaling strategies. However, even at the outset, wire parasitic were a known challenge to continuation of this clock race and the clock speeds have peaked and even tapered off at 3-4GHz. Because of limitations established by Amdahl’s law and it’s not yet time to say the clock race has ended if other fabrication technologies are considered.

12.2 Computing issues:
12.2.1 Amdahl’s law:
Amdahl’s law discusses the performance improvement that can be achieved in a system by speeding up a certain component. This law can be used to evaluate the amount of performance benefits by increasing the speed of the core or the number of cores. The industry has fully embraced the multi-core approach to continuing the processor
throughput issue in an era where the scaling limits for wiring, repeater quantities and associated power dissipation have forced this paradigm change. The objective of the multi-core approach is that code that can be parallelized can be speeded up by introducing more processor nodes to execute them. However, most application code has a serial component and a parallel component. Amdahl created a figure of merit (or FOM) given by

\[
\text{Performance} = \frac{S + P}{S + \frac{P}{n}}
\]

where \( P \) is the fraction of the code that can be parallelized, \( S \) is the fraction that cannot be parallelized and \( n \) is the number of cores. As \( n \) goes to infinity the figure of merit becomes \( 1 + \frac{P}{S} \). The serial code acts as a limiting factor on the performance improvement that can be achieved by increasing number of cores.

Figure 112: Amdahl's law
12.3 Application scalability:
Apart from a few HPC applications, most applications do not scale with more threads and cores. Performance is seen to degrade as managing parallel threads in sync becomes a tough task as their count increases. This suggests that adding more cores doesn't necessarily help in improving current applications.

12.4 Moore’s law and CMOS scaling:
Lithographic scaling has helped maintain the Moore’s law of doubling transistors. The scaling however has reached a limit at around 22nm. At these dimensions, the transistors are showing high leakage currents because of both thinner oxides and their inability to turn off completely because of smaller gate lengths. The cost of preparing mask sets for circuits designed in these nodes have also escalated and hence affordable only by a select few companies. At nodes smaller than 16nm, the use of EUV will be essential. This however limits its use in mass production. Scaling has also resulted in having transistors that require wires to be scaled accordingly in order to reduce the capacitance. This however has resulted in increasing the resistivity of the wires. Without scaling, the industry has hit a road block in attempting to increase the logic count in the form of increasing number of cores or the size of memory in the chip. 3D circuit design has been regarded as a means to continue the trend of Moore’s law of doubling transistors without having to resort to scaling. Other benefits such as ultra wide bandwidth have also enabled the possibility of increasing the clock speed and overcoming the memory wall problem.
Additional performance improvements beyond scaling were needed through the years, one of which was instruction level parallelism. While appealing, it is not without its own problems, one of which is implied by Amdahl’s Law.

12.5 Wire scaling:
With CMOS scaling to gain transistor performance, the wires did not scale accordingly. Wires and contacts increased in resistance. To combat this, repeaters were introduced [3] to help reshape resistance-capacitance limited charging events. Consequently, unless there is a breakthrough in carbon nano-tubes, graphenes, spintronics, cryo CMOS, or low temperature superconductors, this becomes one of the ultimate defining limitations of scaling. This repeater explosion has resulted in the introduction of multiple cores at modest clock rate.

Figure 114: Repeater count increasing with cmos scaling

12.6 Serial code accelerator:
A solution to the presence of serial code in applications would be the addition of a high clock rate unit core that would act as a serial code accelerator. Going by the Amdahl’s law on speed up achieved by improving the performance of serial code acceleration by a factor of m, the overall speed up is represented by the equation below. As in the previous case, a highly scalable application that can be parallelized on a large number of cores, the limiting condition would be m (1+P/S).

\[
\lim_{n \to \infty} FOM_2 = \lim_{n \to \infty} \frac{(S + P)}{\left(\frac{S}{m} + \frac{P}{n}\right)} = m \left(1 + \frac{P}{S}\right)
\]
The proposed integrated chip is shown below, with a HCRU along with a set of parallel low power CMOS cores. The integration could be on the same die using compatible transistor technologies or perhaps through the use of Silicon carriers.

![Diagram of MCU cores and HCRU]

**Figure 115: Heterogeneous integration of HCRU with low power cores**

Even if the programs have a high percentage of parallelizable code, there is an added burden on the system to maintain memory consistency between the various threads that are running on different cores. These operations implemented through the use of barriers etc end up being additional serial code. They are in effect additional burden on the operating system to maintain the operation of parallel code. Thus having a serial code accelerator benefits the above thread synchronization and memory consistency handling code too.

### 12.7 High Clock Rate Processor:

Implementing a serial code accelerator calls for a high clock rate processor. Implementing it would require review of other technologies to implement high speeds given the limitations of CMOS and wire scaling that limits high clock rate performances. Options include use of carbon nanotubes or graphenes to replace wires to overcome wire resistance; use of air gap or low-k dielectric technology to overcome wire capacitance; use of SiGe MOSFETs or FINFETs to have faster and smaller CMOS transistors; low temperature electronics or cryo electronics to achieve higher speeds. While these options look attractive they have some disadvantages that prevent its possibility for the solution the authors are interested. Carbon nanotubes and graphenes haven’t yielded itself for mass production in the field of electronics. Air gap and low-k dielectric improves RC
delay through lower capacitance, yet the resistance still stays high. Similarly, SiGe MOSFET and FINFETs help improving the speed and controllability of MOSFETs but do nothing about wires. An often ignored technology that can fill this gap would be the use of SiGe Hetero-junction Bipolar transistors that have high cut off frequencies and are good drivers of wires. While the technology has predominantly developed for RF technologies, it is very much a viable option for making a Bipolar processor.

12.8 3D Memory:
3D memory integration would be essential in a high clock rate processor [Mitigating memory wall ref]. 3D memory also gives the designers the option to have multiple ports and wide buses. These can be used to bring multiple and large blocks from the last level 3D cache memory to the processor core to ensure that the core doesn’t idle while waiting for data from the cache. Having multiple ports to different tiers allow for carrying out multiple load requests simultaneously thereby improving performance. 3D memory will also prove vital when it comes to chip multi-core package where having a large cache is constrained by both space requirements as well as limited bandwidth between the cores and 3D memory. Fig 5 shows the limitations of 2D chip designs where multiple cores attempt to access the last level of cache and the limited bandwidth ends up throttling down the performance because of the requirement of arbitration. 3D memory on chip multi-core provides wide bandwidth with Through-Silicon-Vias (TSVs).

Figure 116: Package limitations on bandwidth mitigated through use of 3D memory.

12.9 Asymmetric Core/ Heterogeneous integration:
Modeling the Asymmetric Multi-core processor along with a shared 3D cache memory would be essential before building the chip. Other considerations like thermal and power modeling will need to be done to successfully implement a working chip. The authors
evaluate architectural benefits of having asymmetric multi-core processor with 3D memory. This in turn resolves problems with Amdahl’s law and memory wall that would be seen in multi-core processors. Fig. 2 below shows a representation of the system that is to be analyzed.

![Diagram of asymmetric multi-core processor with 3D memory]

**Figure 117: Modeling asymmetric multi-core processor with 3D memory.**

Figure 117 shows a model for heterogeneous asymmetric multi-core processor with 3D memory. Serial code is sent to the fast core while parallel code is sent to the multiple cores. OpenMP compiler directives could be used to identify parallel blocks of code.

**12.10 Modeling Heterogeneous asymmetric multi-core processor:**
System analysis of a heterogeneous asymmetric core processor would require applications to take advantage of the fast core to obtain maximum benefits. Applications today are typically multi threaded with limited parallelism. Serial components of the code can be assigned to the fast core while the parallel part of the code is assigned to the symmetric multi-cores. For proper functioning, synchronization between the asymmetric cores and their non uniform memory access is examined.

**12.11 Shared 3D cache memory:**
Maintaining cache coherency in a multi core processor typically makes use of protocols such as MESI, MSI in the cache system. In an asymmetric system, there will be more complications arising out of the different rate of clock speed as well as nature of application thread being run by the core. Maintaining coherency in the presence of asymmetric cores needs to be examined and techniques need to be developed to ensure
that the cache access is uniformly divided between all the cores while at the same time maintaining coherency. Fast cores and slow cores could operate at different buses, thus implementing improved bus snooping protocols to handle this situation would be essential.

12.12 Simulation Model:
Before designing at chip level an asymmetric multi-core processor, one has to model the architecture and evaluate the model against various benchmarks. Two models are of interest here as shown below.

12.12.1 Shared Memory Model:
Shared memory architecture model as shown in Figure 118 shows a 3D cache memory that acts as the last level cache. Each core has its own private L1 and L2 caches. The L2 caches are connected to the last level cache through a crossbar. However, there needs to be an arbiter to decide the connection priority between the core and the cache. The model however does not exploit the full advantage of 3D cache memory in the form of ultra-wide bandwidth. All cores communicate to the cache through a given port and therefore it is bandwidth limited when multiple requests are made simultaneously.

Figure 118: CMP model with shared cache
12.12.2 Distributed Memory Model:
A second model of interest is similar to distributed memory model as shown in Figure 120. Each core is assigned a set of banks. Coherence is maintained between the different set of banks through a directory based system. This option allows the implementation of multiple ports access to different bank layers, thus exploiting the ultra-wide bandwidth feasible in 3D memory over chip multi-processor.

Figure 119: 3D memory on Multi-core architecture explored through two configurations.

12.13 Research Impact and Applications:
Current general purpose processors with 2 to 4 cores have kept most users content with its processing capabilities where the need has been limited to web browsing, multimedia and productivity applications. Increased throughput has also been achieved for graphics applications by the addition of more cores in a Homogeneous multi-core processor system. The scientific community would however benefit a lot in having asymmetric multi-core processor that is designed for their needs. Typical applications in the scientific domain such as SPICE simulations, finite element analysis have limited parallelism. Accelerating the serial code with a specialized fast core along with the additional cores for the parallel code could improve significantly the overall performance of the system. Utilization of such an asymmetric core with 3D memory
could also benefit supercomputing applications which are currently limited in performance due to the latency involved in inter processor communications. Standard libraries calls in a Message Passing Interface could be executed at high clock rates on the fast core while the parallel cores prepare the packets for transmission. Such asymmetric cores can find other applications in cloud computing or virtual conferencing where the application would have a mix of serial and parallel code.

12.14 Conclusions:
The chapter explores the need for a high clock rate processor to operate in conjunction with low power multi-core processors to deal with serial code present in most applications. Amdahl's law shows that this serial code presents a limiting condition to the performance gain that can be achieved from adding more cores. Heterogeneous integration with 3D memory of a fast core and parallel cores provides a solution to overcome Amdahl's law.
13. Integrated Injection Logic:

As power consumption for CML based SiGe BiCMOS design is a big concern, alternate logic styles are explored that have lower power consumption. Of interest in particular is the Integrated Injection Logic. The logic works on the fact the PNP transistor is used as a constant current source injector feeding current into the NPN transistor.

13.1 PNP-NPN IIL

As shown in Figure 120, Q1 acts as the current injector. Q2 acts as both the active load for Q3 as well as the current source for next stage for Q5. Q4 again acts as a similar active load for Q5. As Q1 is biased to GND, the transistor is saturated and thus acts as a constant current source. When input "In" is high, Q3 is turned on, thus giving a path for the current from Q1 to flow through Q3 to the ground. When input "In" is low or in other words input tied to ground, the current from the Q1 PNP flows to the ground thus leaving Q3 off. When Q3 is on for high input, the output is pulled low. This thus acts as an inverter. Figure 120 acts as a two stages of inverter that is cascaded.

![Integrated Injection logic using PNP as constant current injector](image)

Figure 120: Integrated Injection logic using PNP as constant current injector

Figure 121 shows the simulated waveform of the above circuit. The results show that speeds using BiCMOS8wl are not anywhere close to the CML logic speeds. Although
the $f_i$ of 8wl is only comparable to BiCMOS 7HP transistor, the simulations still showed much lower speeds than expected.

**Figure 121: Inverter outputs after stage 1 and stage 2.**

Figure 122 below indicates the current in the collector of PNP transistor. The PNP transistor was supposed to provide a constant current to the NPN transistor. The variations in the current show that, the PNP transistor is not in a position to react to the faster current variations in the NPN transistor.
Given the problems of PNP transistor being too slow, we explored an alternative IIL circuit using NPN only transistor [74]. Figure 123: An NPN only multi-valued logic structure below shows two inverters connected together. As memory circuits are primarily cross coupled inverters, the below configuration gives a rough estimate of what speeds can be obtained from a memory cell. The transistor is shown to be a multi-valued logic. As the transistor kit doesn’t have such multi-collector transistors, we have mimicked the operation by tying two transistors together as shown in Figure 124.
Figure 123: An NPN only multi-valued logic structure

Figure 124: Faking a multi-valued logic through the use of two NPN transistors.

Figure 125 shows the modified circuit with the faked multi-collector transistor. The simulations results show promising speeds that are capable of replacing CML based logic while operating at low power. However as seen in Figure 126, the fall times are still very slow. This is seen to be as a result of the transistor going into saturation and it takes a longer time to remove the charge from the base. A solution to avoid this is to have a schottky diode to clamp the output at around 300 to 400mV. This could improve the fall time speeds and make it comparable to the rise time.
Figure 125: Two inverter stages implemented using the modified multi-valued transistors.

Table 18 shows a comparison of speeds and voltage levels of 3 different bipolar logic inverter circuits. It is seen that the CML logic which has been widely considered the fastest but most power consuming can be matched with the NPN only Integrated Injection Logic. The NPN only IIL can also operate at much lower supply voltages, thus reducing the static power.
<table>
<thead>
<tr>
<th>Logic</th>
<th>Supply Voltage</th>
<th>Voltage Swing</th>
<th>Rise Time</th>
<th>Fall Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>CML (8HP)</td>
<td>2.5V</td>
<td>300mV</td>
<td>4ps</td>
<td>4ps</td>
</tr>
<tr>
<td>NPN-PNP IIL (8WL) ~7HP f&lt;sub&gt;T&lt;/sub&gt;</td>
<td>1.1V</td>
<td>300mV</td>
<td>30ps</td>
<td>36ps</td>
</tr>
<tr>
<td>NPN only IIL (8HP)</td>
<td>1.1V</td>
<td>300mV</td>
<td>4.4ps</td>
<td>22ps</td>
</tr>
</tbody>
</table>

Table 18: Comparison of Rise, fall times and voltage levels for IIL and CML inverter circuit logic.

13.3 Memory design

The exploration of Integrated Injection Logic was carried out mainly to find a means to replace the register file implementation which is seen currently having high power consumption on the order of 5W for 8 word entry. CML based design for the register file was proposed to implement the L0 cache banks too. This contributes to significantly large power for the overall CPU. Replacing this logic with IIL, one can possibly implement much lower power memory. Figure 127 and Figure 128 show the memory circuit comparison of CML and IIL.

![Figure 127: Memory cell implemented using CML circuit](image-url)
The above circuit is similar to the 6T SRAM cell made with MOSFET. T1, T2, T5, T6 acts as the cross coupled inverters. T3, T4 are connected to the base of T1 and T2 and they function as the access transistors.

13.4 Summary
IIL promises a low power alternative to CML design while at the same time having almost comparable speeds especially with the NPN only IIL. Further exploration of this logic circuit will be needed towards the implementation of complex logic.
14. Research goals ahead

14.1 Review:
My PhD dissertation has explored the study of 3D Processor Memory stack architecture. The research background of my thesis advisor has been into building high clock rate processors. High clock rate processors however would suffer from memory wall problem as the memory cannot keep up with the processor speeds. To counter the memory wall problem, my research explored the advantages of stacking 3D memory on top of the processor core. 3D chip stacking sustains the pace of Moore’s law of doubling transistor count and density every two years. The researchers in the last decade had predominately focused on reducing the lithography size of CMOS devices from 250nm to 22nm. But, this scaling is reaching a limit as the physical dimensions of the MOSFET and wires shrink. The logical way to continue doubling the transistors is to stack chips vertically through wafer stacking. Wires have become resistive while MOSFETs have become leaky resulting in increased leakage power. 3D stacking opens up the microprocessor design to many more possibilities than previous 2D configuration. It not only allows increasing the logic count in a given area, but also solves other issues such as increased capacity of last level cache in the form of 3D memory and providing ultra wide bandwidth between the cache levels. 3D stacking also provides for shorter interconnects from vertical integration which reduces the access time for the memory blocks compared to a 2D configuration. These solutions have reduced the impact of memory wall problem that would be seen in a high clock rate processor.

![Diagram of 3D memory stack on processor](image_url)

**Figure 129:** Showing the architectural benefits of 3D memory stacked on processor.
Multiple access ports and wide bus can be drawn vertically. Availability of wide bus allows simultaneous access to different banks and layers of the 3D memory stacked on the processor.

The research goals funded by a DARPA program on 3D IC, included evaluation of the implementation details of a pipelined processor core and the cache memory hierarchy. As a computer hardware architect, I got to evaluate the various design choices including implementation at circuit level to determine the best options. Alternate technologies for implementing the processor core such as SiGe BiCMOS technology which have transistors with high cut off frequency $f_t$ on the order of 210GHz was evaluated in this regard. SiGe technology allows the feasibility of building a simple processor core that could operate at 16GHz or higher. The goal was to build a cache hierarchy around such a high clock rate processor. I got to validate the cache hierarchy using various benchmarks such as SPEC and Biobench by modifying processor-memory simulators available to the academic community, such as Simplescalar and CACTI, to suit to the 3D processor memory stack I was evaluating. Various strategies such as ultra wide bus, faster L1 and L0 cache using BiCMOS technologies, increasing the number of layers of 3D cache memory as well as multiple memory management units accessing multiple banks and layers in 3D memory were explored as means to reduce the Clocks per Instruction metric. I went on to create a pipelined processor hardware model on an FPGA to test and validate the architecture of the processor. As it is important to carry out a thermal aware design of a processor especially in a 3D chip where heat dissipation is a problem, I carried out thermal analysis of the 3D processor memory system. This allowed me to redesign the processor with a power and thermal budget in mind. The analysis also permitted me to devise strategies such as use of diamond and copper interface layers to remove heat from the 3D stack. My study on the architecture of a high clock rate processor was extended to analyze multi-core processor with 3D memory. As multi-core processors will also face the memory wall problem due to the simultaneous contention for the bus by multiple cores, performance improvements in this regard were carried out. Existing symmetric multiprocessor simulators (RSIM) was adapted to simulate a chip multi-core processor with wide bus as well as multiple access ports to the 3D cache memory. The research carried out so far has opened up additional research topics which
I can look into. Evaluating asymmetric multi-core processor with different clock speed along with 3D memory is an exciting area that the research group could start exploring.

14.1.1 Research Impact and Applications:
Current general purpose processors with 2 to 4 cores have kept most users content with its processing capabilities where the need has been limited to web browsing, multimedia and productivity applications. Increased throughput has also been achieved for graphics applications by the addition of more cores in a Homogeneous multi-core processor system. The scientific community would however benefit a lot in having asymmetric multi-cored processor that is designed for their needs. Typical applications in the scientific domain such as SPICE simulations, finite element analysis have limited parallelism. Accelerating the serial code with a specialized fast core along with the additional cores for the parallel code could improve significantly the overall performance of the system. Utilization of such an asymmetric core with 3D memory could also benefit supercomputing applications which are currently limited in performance due to the latency involved in inter processor communications. Standard libraries calls in a Message Passing Interface could be executed at high clock rates on the fast core while the parallel cores prepare the packets for transmission. Such asymmetric cores can find other applications in cloud computing or virtual conferencing where the application would have a mix of serial and parallel code.

14.1.2 Heterogeneous integration:
One can also take up certain tasks of implementing the processor in multiple levels of wafer, which would allow shortening of the wires in the processor too and thus one can achieve speeds on the order of 32GHz. Implementations in 9HP is also under consideration for this. This also holds the advantage in that the processor design cannot be retrieved by reverse engineering as one can move the blocks in the processor between 2 layers thus preventing easy identification of the connections between the blocks that have been used in the processor design. One also would need to consider the different heat generated from different processes, which makes this task more complex to understand in an heterogeneously integrated chip. Design of heat sinks for this would make an important topic of research.
15. Bibliography

ARCHITECTURE


THERMAL


SiGe


IC DESIGN


**Bipolar Logic**


**Misc.**

Appendix A. Simulators

A.1 DineroIV
Sample command for Dinero
./dineroIV -l1-isize 4k -l1-dsize 4k -l1-ibsize 1024 -l1-ibsize 1024 -l1-dbsize 1024 -l1-dbsize 1024 -l2-usize 32k -l2-ubsize 1024 -informat d<cc1.din> 4k4k8k_bus1024_1.out

A.2 Simplescalar simulator:
The different set of simulators included in the package and their characteristics are shown in Fig 5. Sim-safe provides the basic simulator. Sim-profile is used for profiling the instructions present in the program that has been fed to the simulator. Sim-cache provides a simulator for simulating a cache system. Sim-outorder provides a complete processor-cache system simulation. As the complexity increases, the speed of the simulation drops. The sim-outorder has been rated for 150KIPS.

![Standard Models](image)

**Figure 130: Simplescalar simulators and their descriptions-(simple scalar tutorial)**

Simple scalar provides 5 simulators- the most useful being sim-cache and sim-outorder. sim-cache is similar to dinero and sim-outorder simulates a processor and cache system.

Benchmark programs belonging to Spec95 suite have been packaged with the tool. So its easy to run an analysis on any of the benchmark. Many universities are also using the tool for architecture simulations. It has features like branch prediction built in. However, it doesn't allow cache hierarchy more than 2 levels. Sim-cache simulator doesn't have the provision to change the bus width for memory access. The data
generated from processor simulator (sim-outorder) carries out simulations at very low frequencies like 150KIPS. Thus the CPI figures, obtained from these simulations, don’t depict the CPI figures that would be seen at higher clock frequencies especially at 16GHz. The sim-outorder processor architecture is also limited in that it has a rigid pipeline structure. One cannot vary the pipeline depth although it allows changing the number of execution units etc.

A cache system with the following architecture was simulated using sim-cache. The data generated has been shown in Table 1.

- cache: dl1  
  dl1:32:32:1:1 [32 sets, block size=32, associativity=1, LRU (l) =1K]
- cache: dl2  
  dl2:256:64:1:1 [256 sets, block size=64, associativity=1, LRU(l) =16K]
- cache: il1  
  il1:32:32:1:1
- cache: il2  
  il2:256:64:1:1

<table>
<thead>
<tr>
<th>sim_num_insn(total instructions)</th>
<th># 5000000</th>
<th>il1.accesses (icache L1)</th>
<th>5000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>sim_num.refs(load/store)</td>
<td>1715057</td>
<td>il1.miss_rate</td>
<td>0.1271</td>
</tr>
<tr>
<td>sim_inst_rate</td>
<td>1000000</td>
<td>il2.accesses (icache L2)</td>
<td>635257</td>
</tr>
<tr>
<td>sim_elapsed_time</td>
<td>5 secs</td>
<td>il2.miss_rate</td>
<td>0.0049</td>
</tr>
<tr>
<td>dl1.accesses (dcache L1)</td>
<td>1796294</td>
<td>dl2.accesses (dcache L2)</td>
<td>352746</td>
</tr>
<tr>
<td>dl1.miss_rate</td>
<td>0.1141</td>
<td>dl2.miss_rate</td>
<td>0.2052</td>
</tr>
</tbody>
</table>

Table 19: Sample data obtained from sim-cache simulator simulating a cache system having L1 and L2 caches both levels being Harvard architecture.

Sim-outorder simulator was used to simulate a processor-cache system. The processor was configured to be a single issue, inorder, single execution stream unit while the cache system used was identical to the one used for above simulations. Sim-outorder simulator has a provision to vary the memory access bandwidth. CPI was estimated for different bandwidths. Table 2 shows the values and it is seen that CPI is saturated. This is so as the clock speed at which the simulation of the processor carried out is very low.
and there isn’t any impact of ultra wide bus width at such low frequencies. Sim-outorder doesn’t have a provision to change the clock frequency.

<table>
<thead>
<tr>
<th>sim_cycle (total cycles)</th>
<th>Memory width</th>
<th>CPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>10432709</td>
<td>32</td>
<td>2.0865</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>2.0852</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>2.0852</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>2.0852</td>
</tr>
</tbody>
</table>

Table 20: sim-outorder simulator used to simulate a processor-cache system having the above L1 and L2 caches.

The above data indicates that CPI doesn’t show any significant improvement by increasing memory bandwidth at low clock speeds as simulated by sim-outorder. This is in contrast with the huge CPI gain that would be seen at frequencies on the order of 16GHz as shown in fig 6b.

Sample commands:

```
./sim-outorder -max:inst 5000000 -config config/regress.cfg -redir:sim Current-outorder.txt m88ksim.ss dcrand.big
./sim-cache -max:inst 5000000 -config config/cachetest.cfg -redir:sim cachetest1.txt m88ksim.ss dcrand.big
./sim-cache -max:inst 5000000 -config config/cachetest.cfg -redir:sim cachetest3.txt cc1.ss anagram
./sim-profile -max:inst 5000000 -redir:sim cachetest2_out.txt m88ksim.ss dcrand.big
```

**sim-outorder:**

sim: command line: ./sim-outorder -max:inst 5000000 -config config/regress.cfg -redir:sim Current-outorder_256.txt ./spec95-big/m88ksim.ss ../../../downloads/spec95-big/inputs/124.m88ksim/input1_bendian/dcrand.big

sim: simulation started, options follow:

sim-outorder: This simulator implements a very detailed out-of-order issue superscalar processor with a two-level memory system and speculative execution support. This simulator is a performance simulator, tracking the
latency of all pipeline operations.

# -config # load configuration from a file
# -dumpconfig # dump configuration to a file
# -h false # print help message
# -v false # verbose operation
# -d false # enable debug message
# -i false # start in Dlite debugger

-seed 1 # random number generator seed (0 for timer seed)

# -q false # initialize and terminate immediately
# -chkpt <null> # restore EIO trace execution from <fname>
# -redir:sim Current-outorder_256.txt # redirect simulator output to file (non-
interactive only)

# -redir:prog <null> # redirect simulated program output to file

-nice 0 # simulator scheduling priority

-max:inst 5000000 # maximum number of inst's to execute

-fastfwd 0 # number of insts skipped before timing starts

# -ptrace <null> # generate pipetrace, i.e., <fname|stdout|stderr> <range>

-fetch:ifqsize 4 # instruction fetch queue size (in insts)

-fetch:mplat 3 # extra branch mis-prediction latency

-fetch:speed 1 # speed of front-end of machine relative to execution core

-bpred bimod # branch predictor type

{nottaken|taken|perfect|bimod|2lev|comb}

-bpred:bimod 2048 # bimodal predictor config (<table size>)

-bpred:2lev 1 1024 8 0 # 2-level predictor config (<l1size> <l2size> <hist_size>

-xor)

-bpred:comb 1024 # combining predictor config (<meta_table_size>)

-bpred:ras 8 # return address stack size (0 for no return stack)

-bpred:btb 512 4 # BTB config (<num_sets> <associativity>)

# -bpred:spec_update <null> # speculative predictors update in {ID|WB} (default

non-spec)

-decode:width 4 # instruction decode B/W (insts/cycle)
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-issue:width</td>
<td>4</td>
<td># instruction issue B/W (insts/cycle)</td>
</tr>
<tr>
<td>-issue:inorder</td>
<td>false</td>
<td># run pipeline with in-order issue</td>
</tr>
<tr>
<td>-issue:wrongpath</td>
<td>true</td>
<td># issue instructions down wrong execution paths</td>
</tr>
<tr>
<td>-commit:width</td>
<td>4</td>
<td># instruction commit B/W (insts/cycle)</td>
</tr>
<tr>
<td>-ruu:size</td>
<td>16</td>
<td># register update unit (RUU) size</td>
</tr>
<tr>
<td>-lsq:size</td>
<td>8</td>
<td># load/store queue (LSQ) size</td>
</tr>
<tr>
<td>-cache:dl1</td>
<td>dl1:128:32:4:1</td>
<td># l1 data cache config, i.e., {&lt;config&gt;</td>
</tr>
<tr>
<td>-cache:dl1lat</td>
<td>1</td>
<td># l1 data cache hit latency (in cycles)</td>
</tr>
<tr>
<td>-cache:dl2</td>
<td>none</td>
<td># l2 data cache config, i.e., {&lt;config&gt;</td>
</tr>
<tr>
<td>-cache:dl2lat</td>
<td>6</td>
<td># l2 data cache hit latency (in cycles)</td>
</tr>
<tr>
<td>-cache:il1</td>
<td>il1:512:64:1:1</td>
<td># l1 inst cache config, i.e., {&lt;config&gt;</td>
</tr>
<tr>
<td>-cache:il1lat</td>
<td>1</td>
<td># l1 instruction cache hit latency (in cycles)</td>
</tr>
<tr>
<td>-cache:il2</td>
<td>none</td>
<td># l2 instruction cache config, i.e., {&lt;config&gt;</td>
</tr>
<tr>
<td>-cache:il2lat</td>
<td>6</td>
<td># l2 instruction cache hit latency (in cycles)</td>
</tr>
<tr>
<td>-cache:flush</td>
<td>false</td>
<td># flush caches on system calls</td>
</tr>
<tr>
<td>-cache:icompress</td>
<td>false</td>
<td># convert 64-bit inst addresses to 32-bit inst equivalents</td>
</tr>
<tr>
<td>-mem:lat</td>
<td>6 1</td>
<td># memory access latency (&lt;first_chunk&gt; &lt;inter_chunk&gt;)</td>
</tr>
<tr>
<td>-mem:width</td>
<td>256</td>
<td># memory access bus width (in bytes)</td>
</tr>
<tr>
<td>-tlb:itlb</td>
<td>none</td>
<td># instruction TLB config, i.e., {&lt;config&gt;</td>
</tr>
<tr>
<td>-tlb:dtlb</td>
<td>none</td>
<td># data TLB config, i.e., {&lt;config&gt;</td>
</tr>
<tr>
<td>-tlb:lat</td>
<td>30</td>
<td># inst/data TLB miss latency (in cycles)</td>
</tr>
<tr>
<td>-res:ialu</td>
<td>4</td>
<td># total number of integer ALU's available</td>
</tr>
<tr>
<td>-res:imult</td>
<td>1</td>
<td># total number of integer multiplier/dividers available</td>
</tr>
<tr>
<td>-res:memport</td>
<td>2</td>
<td># total number of memory system ports available (to CPU)</td>
</tr>
<tr>
<td>-res:fpalu</td>
<td>4</td>
<td># total number of floating point ALU's available</td>
</tr>
<tr>
<td>-res:fpmult</td>
<td>1</td>
<td># total number of floating point multiplier/dividers available</td>
</tr>
<tr>
<td># -pcstat</td>
<td>&lt;null&gt;</td>
<td># profile stat(s) against text addr's (mull uses ok)</td>
</tr>
<tr>
<td>-bugcompat</td>
<td>true</td>
<td># operate in backward-compatible bugs mode (for testing only)</td>
</tr>
</tbody>
</table>

Pipetrace range arguments are formatted as follows:

```
{{@|#}<start>}:{{@|#|+}<end>}
```
Both ends of the range are optional, if neither are specified, the entire execution is traced. Ranges that start with a `@' designate an address range to be traced, those that start with an `#' designate a cycle count range. All other range values represent an instruction count range. The second argument, if specified with a `+', indicates a value relative to the first argument, e.g., 1000:+100 == 1000:1100. Program symbols may be used in all contexts.

Examples:  
- ptrace FOO.trc #0:#1000  
  - ptrace BAR.trc @2000:  
  - ptrace BLAH.trc :1500  
  - ptrace UXXE.trc :  
  - ptrace FOOBAR.trc @main:+278

Branch predictor configuration examples for 2-level predictor:

Configurations:  
N, M, W, X

N  # entries in first level (# of shift register(s))  
W  width of shift register(s)  
M  # entries in 2nd level (# of counters, or other FSM)  
X  (yes-1/no-0) xor history and address for 2nd level index

Sample predictors:

GAg  : 1, W, 2^W, 0  
GAp  : 1, W, M (M > 2^W), 0  
PAg  : N, W, 2^W, 0  
PAp  : N, W, M (M == 2^(N+W)), 0  
gshare : 1, W, 2^W, 1

Predictor `comb' combines a bimodal and a 2-level predictor.

The cache config parameter <config> has the following format:

<name>:<nsets>:<bsize>:<assoc>:<repl>

<name>  - name of the cache being defined
<nsets> - number of sets in the cache
<bsize> - block size of the cache
<assoc> - associativity of the cache
<repl> - block replacement strategy, 'l'-LRU, 'f'-FIFO, 'r'-random

Examples:  -cache:dl1 dl1:4096:32:1:l
           -dtlb dtlb:128:4096:32:r

Cache levels can be unified by pointing a level of the instruction cache
hierarchy at the data cache hierarchy using the "dl1" and "dl2" cache
configuration arguments. Most sensible combinations are supported, e.g.,

A unified l2 cache (il2 is pointed at dl2):
  -cache:il1 il1:128:64:1:1 -cache:il2 dl2

Or, a fully unified cache hierarchy (il1 pointed at dl1):
  -cache:il1 dl1

sim: ** starting performance simulation **

sim: ** simulation statistics **

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sim_num_insn</td>
<td>5000000</td>
<td># total number of instructions committed</td>
</tr>
<tr>
<td>sim_num.refs</td>
<td>1715057</td>
<td># total number of loads and stores committed</td>
</tr>
<tr>
<td>sim_num.loads</td>
<td>532431</td>
<td># total number of loads committed</td>
</tr>
<tr>
<td>sim_num.stores</td>
<td>1182626.0000</td>
<td># total number of stores committed</td>
</tr>
<tr>
<td>sim_num.branches</td>
<td>923947</td>
<td># total number of branches committed</td>
</tr>
<tr>
<td>sim_elapsed_time</td>
<td>30</td>
<td># total simulation time in seconds</td>
</tr>
<tr>
<td>sim_inst_rate</td>
<td>166666.6667</td>
<td># simulation speed (in insts/sec)</td>
</tr>
<tr>
<td>sim_total_insn</td>
<td>5375741</td>
<td># total number of instructions executed</td>
</tr>
<tr>
<td>Metric</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>sim_total.refs</td>
<td>1885446</td>
<td># total number of loads and stores executed</td>
</tr>
<tr>
<td>sim_total.loads</td>
<td>696686</td>
<td># total number of loads executed</td>
</tr>
<tr>
<td>sim_total.stores</td>
<td>1188760.0000</td>
<td># total number of stores executed</td>
</tr>
<tr>
<td>sim_total.branches</td>
<td>1069932</td>
<td># total number of branches executed</td>
</tr>
<tr>
<td>sim_cycle</td>
<td>2365434</td>
<td># total simulation time in cycles</td>
</tr>
<tr>
<td>sim_IPC</td>
<td>2.1138</td>
<td># instructions per cycle</td>
</tr>
<tr>
<td>sim_CPI</td>
<td>0.4731</td>
<td># cycles per instruction</td>
</tr>
<tr>
<td>sim_exec_BW</td>
<td>2.2726</td>
<td># total instructions (mis-spec + committed) per cycle</td>
</tr>
<tr>
<td>sim_IPB</td>
<td>5.4116</td>
<td># instruction per branch</td>
</tr>
<tr>
<td>IFQ_count</td>
<td>7965131</td>
<td># cumulative IFQ occupancy</td>
</tr>
<tr>
<td>IFQ_fcount</td>
<td>1728237</td>
<td># cumulative IFQ full count</td>
</tr>
<tr>
<td>ifq_occipancy</td>
<td>3.3673</td>
<td># avg IFQ occupancy (insn's)</td>
</tr>
<tr>
<td>ifq_rate</td>
<td>2.2726</td>
<td># avg IFQ dispatch rate (insn/cycle)</td>
</tr>
<tr>
<td>ifq_latency</td>
<td>1.4817</td>
<td># avg IFQ occupant latency (cycle's)</td>
</tr>
<tr>
<td>ifq_full</td>
<td>0.7306</td>
<td># fraction of time (cycle's) IFQ was full</td>
</tr>
<tr>
<td>RUU_count</td>
<td>28135850</td>
<td># cumulative RUU occupancy</td>
</tr>
<tr>
<td>RUU_fcount</td>
<td>590588</td>
<td># cumulative RUU full count</td>
</tr>
<tr>
<td>ruu_occipancy</td>
<td>11.8946</td>
<td># avg RUU occupancy (insn's)</td>
</tr>
<tr>
<td>ruu_rate</td>
<td>2.2726</td>
<td># avg RUU dispatch rate (insn/cycle)</td>
</tr>
<tr>
<td>ruu_latency</td>
<td>5.2339</td>
<td># avg RUU occupant latency (cycle's)</td>
</tr>
<tr>
<td>ruu_full</td>
<td>0.2497</td>
<td># fraction of time (cycle's) RUU was full</td>
</tr>
<tr>
<td>LSQ_count</td>
<td>9979122</td>
<td># cumulative LSQ occupancy</td>
</tr>
<tr>
<td>LSQ_fcount</td>
<td>674235</td>
<td># cumulative LSQ full count</td>
</tr>
<tr>
<td>lsq_occipancy</td>
<td>4.2187</td>
<td># avg LSQ occupancy (insn's)</td>
</tr>
<tr>
<td>lsq_rate</td>
<td>2.2726</td>
<td># avg LSQ dispatch rate (insn/cycle)</td>
</tr>
<tr>
<td>lsq_latency</td>
<td>1.8563</td>
<td># avg LSQ occupant latency (cycle's)</td>
</tr>
<tr>
<td>lsq_full</td>
<td>0.2850</td>
<td># fraction of time (cycle's) LSQ was full</td>
</tr>
<tr>
<td>sim_slip</td>
<td>42288765</td>
<td># total number of slip cycles</td>
</tr>
<tr>
<td>avg_sim_slip</td>
<td>8.4578</td>
<td># the average slip between issue and retirement</td>
</tr>
<tr>
<td>bpred_bimod.lookups</td>
<td>1122270</td>
<td># total number of bpred lookups</td>
</tr>
<tr>
<td>bpred_bimod.updates</td>
<td>923943</td>
<td># total number of updates</td>
</tr>
</tbody>
</table>
bpred_bimod.addr_hits       887083 # total number of address-predicted hits
bpred_bimod.dir_hits         892304 # total number of direction-predicted hits (includes addr-hits)
bpred_bimod.misses          31639 # total number of misses
bpred_bimod.jr_hits          136717 # total number of address-predicted hits for JR's
bpred_bimod.jr_seen          141646 # total number of JR's seen
bpred_bimod.jr_non_ras_hits.PP         45 # total number of address-predicted hits for non-RAS JR's
bpred_bimod.jr_non_ras_seen.PP        77 # total number of non-RAS JR's seen
bpred_bimod.bpred_addr_rate    0.9601 # branch address-prediction rate (i.e., addr-hits/updates)
bpred_bimod.bpred_dir_rate    0.9658 # branch direction-prediction rate (i.e., all-hits/updates)
bpred_bimod.bpred_jr_rate     0.9652 # JR address-prediction rate (i.e., JR addr-hits/JRs seen)
bpred_bimod.bpred_jr_non_ras_rate.PP     0.5844 # non-RAS JR addr-pred rate (ie, non-RAS JR hits/JRs seen)
bpred_bimod.retstack_pushes    144964 # total number of address pushed onto ret-addr stack
bpred_bimod.retstack_pops     148872 # total number of address popped off of ret-addr stack
bpred_bimod.used_ras.PP       141569 # total number of RAS predictions used
bpred_bimod.ras_hits.PP       136672 # total number of RAS hits
bpred_bimod.ras_rate.PP       0.9654 # RAS prediction rate (i.e., RAS hits/used RAS)
il1.accesses                 5514847 # total number of accesses
il1.hits                     5511994 # total number of hits
il1.misses                   2853 # total number of misses
il1.replacements             2441 # total number of replacements
il1.writebacks               0 # total number of writebacks
il1.invalidations            0 # total number of invalidations
il1.miss_rate                0.0005 # miss rate (i.e., misses/ref)

il1.repl_rate 0.0004 # replacement rate (i.e., repls/ref)
il1.wb_rate 0.0000 # writeback rate (i.e., wrbks/ref)
il1.inv_rate 0.0000 # invalidation rate (i.e., invs/ref)
dl1.accesses 1737940 # total number of accesses
dl1.hits 1605550 # total number of hits
dl1.misses 132390 # total number of misses
dl1.replacements 131878 # total number of replacements
dl1.writebacks 129072 # total number of writebacks
dl1.invalidations 0 # total number of invalidations
dl1.miss_rate 0.0762 # miss rate (i.e., misses/ref)
dl1.repl_rate 0.0759 # replacement rate (i.e., repls/ref)
dl1.wb_rate 0.0743 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate 0.0000 # invalidation rate (i.e., invs/ref)
sim_invalid_addrs 0 # total non-speculative bogus addresses seen (debug var)
ld_text_base 0x00400000 # program text (code) segment base
ld_text_size 286672 # program text (code) size in bytes
ld_data_base 0x10000000 # program initialized data segment base
ld_data_size 130436 # program init'ed `.data' and uninit'ed `.bss' size in bytes
ld_stack_base 0x7fffc000 # program stack segment base (highest address in stack)
ld_stack_size 16384 # program initial stack size
ld_prog_entry 0x00400140 # program entry point (initial PC)
ld_environ_base 0x7fff8000 # program environment base address
dl_target_big_endian 1 # target executable endian-ness, non-zero if big endian
mem.page_count 1030 # total number of pages allocated
mem.page_mem 4120k # total size of memory pages allocated
mem.ptab_misses 1031 # total first level page table misses
mem.ptab_accesses 37692665 # total page table accesses
mem.ptab_miss_rate 0.0000 # first level page table miss rate
A.3 RSIM simulator:

RSIM simulator is used to explore shared memory multi-core architecture. It is being used to approximate a chip multi-core simulator by zeroing out some of the associated network delays, flit delays etc. A single chip multi-core have these components as almost zero compared as all these delays are on the order of microseconds compared to several milliseconds in the case of network architecture. Below is the list of commands for various benchmarks. Also listed is the configuration file for the RSIM simulator.

Benchmark commands:

fft
./rsim -A 5000 -i1 -f fft_rcopt -S fft_rcopt -D../outputs -0 /dev/null -z rsim.config -- -t -p2 -m14

ocean
./rsim${T} -A 5000 -i1 -f ocean_rc -S ocean_rc-n34-p2 -D../outputs -0 /dev/null -z rsim.config -- -n34 -p2

gauss
./rsim -i1 -f em3d_rc -S em3d_rc -D../outputs -0 /dev/null -z rsim.config -- 2 10 5 1 10

raytrace
./rsim -i1 -f raytrace_rc -S raytrace_rc -D../outputs -0 /dev/null -z rsim.config -- -p2 -m10 teapot.env

radix
./rsim -f radix_rc -S radix_rc-rcc -D../outputs -0 /dev/null -z rsim.config -- -t-p4 -n1024 -m4096

lu
./rsim${T} -i1 -f lu_rcopt -S lu_rcopt${T} -D../outputs -0 /dev/null -z rsim.config -- -n64 -p4

lucont
./rsim -i1 -f lu_rc -S lu_rc -D../outputs -0 /dev/null -z rsim.config -- -n100 -p4

tomcatv
./rsim -A 5000 -i1 -f tomcatv_rc -S tomcatv_rc -D../outputs -0 /dev/null -z rsim.config -- 100 10 4
Configuration:

numnodes 16
reqsz 16
bpbtpe 2bit
bpbsize 512
shadowmappers 8
rassize 4
numalus 2
numfpsus 2
numadrs 2
regwindows 8
latint 1
repint 1
latmul 3
repmul 1
latdiv 9
repdv 1
latshift 1
repshift 1
latflt 3
repflt 1
latfmov 1
repfmov 1
latfconv 4
repfconv 2
latfdiv 10
repfdiv 6
latfesqrt 10
repfesqrt 6
maxstack 1024
l1type WT
linesize 64
l1size 16
l1assoc 1
l1ports 2
l1taglatency 1
l2size 64
l2assoc 4
l2taglatency 3
l2datalatency 5
wrbbufextra 0
ccprot mesi
wbufsize 8
mshrcoal 16
buswidth 32
buscycle 3
busarbdelay 1
memorylatency 18
dircycle 3
meminterleaving 4
dirbufsize 64
dirpacketcreate 12
dirpacketcreateaddtl 6
flitsize 8
flitdelay 4
arbdelay 4
pipelinedsw 2
netbufsize 64
netportsize 64
portsz1wbreq 2
portszwb11rep 1
portszwbl2req 1
portszl2wbrep 1
portszl1l2req 2
portszl1l1rep 1
portszl1l1cohe 1
portszl1l2cr 1
portszl2busreq 8
portszb12rep 2
portszb12cohe 2
portszbuscr 8
portszbusother 16
portszdir 64
STOPCONFIG 1
### Appendix B. MPICH instruction set

The MPICH code was compiled on PowerPC processor (MAC system) using the gcc compiler with the option “–s”. These instructions formed the basis for identifying the set of instructions that are to be implemented in the processor core.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Instruction category</th>
<th>Description</th>
<th>Percentage %</th>
</tr>
</thead>
<tbody>
<tr>
<td>addc r3,r3,r10</td>
<td>Integer arithmetic</td>
<td>Add carrying</td>
<td>0.00</td>
</tr>
<tr>
<td>adde r2,r2,r9</td>
<td>Integer arithmetic</td>
<td>Add extended</td>
<td>0.00</td>
</tr>
<tr>
<td>addi r0,r30,120</td>
<td>Integer arithmetic</td>
<td>Add Immediate</td>
<td>1.20</td>
</tr>
<tr>
<td>addis r2,r31, ha16(L_MPIR_Process$non_lazy_ptr-$&quot;L00000000001$pb&quot;)</td>
<td>Integer arithmetic</td>
<td>Add Immediate Shifted</td>
<td>8.01</td>
</tr>
<tr>
<td>add r0,r0,r9</td>
<td>Integer arithmetic</td>
<td>Add</td>
<td>2.30</td>
</tr>
<tr>
<td>addze r0,r0</td>
<td>Integer arithmetic</td>
<td>Add to zero</td>
<td>0.01</td>
</tr>
<tr>
<td>bcl 20,31,&quot;L00000000001$pb&quot;</td>
<td>Branch</td>
<td>Branch Conditional</td>
<td>1.32</td>
</tr>
<tr>
<td>bcl 20,31,L2$_MPID_Abort</td>
<td>Branch</td>
<td>Branch unconditionally to CTR</td>
<td>0.00</td>
</tr>
<tr>
<td>bctr</td>
<td>Branch</td>
<td>Branch unconditionally to CTR setting LR</td>
<td>1.04</td>
</tr>
<tr>
<td>Bctrl</td>
<td>Branch</td>
<td>Branch unconditionally to CTR setting LR</td>
<td>0.07</td>
</tr>
<tr>
<td>beq cr7,L6</td>
<td>Branch</td>
<td>Branch if Equal</td>
<td>2.90</td>
</tr>
<tr>
<td>bge cr7,L27</td>
<td>Branch</td>
<td>Branch if Greater than or equal to</td>
<td>0.24</td>
</tr>
<tr>
<td>bgt cr7,L29</td>
<td>Branch</td>
<td>Branch if Greater than</td>
<td>0.32</td>
</tr>
<tr>
<td>b L7</td>
<td>Branch</td>
<td>Branch</td>
<td>4.16</td>
</tr>
<tr>
<td>ble cr7,L41</td>
<td>Branch</td>
<td>Branch If Less Than Or Equal To</td>
<td>0.38</td>
</tr>
<tr>
<td>bl L_MPIR_Err_create_code$sstub</td>
<td>Branch</td>
<td>Branch</td>
<td>3.29</td>
</tr>
<tr>
<td>blr</td>
<td>Branch</td>
<td>Branch unconditionally to LR</td>
<td>0.29</td>
</tr>
<tr>
<td>blt cr7,L29</td>
<td>Branch</td>
<td>Branch If Less Than</td>
<td>0.00</td>
</tr>
<tr>
<td>bne cr7,L8</td>
<td>Branch</td>
<td>Branch if Not</td>
<td>1.87</td>
</tr>
<tr>
<td></td>
<td>Instruction</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---</td>
<td>--------------------------------------------------</td>
<td>---------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>20</td>
<td><code>cmplwi cr7,r0,1</code></td>
<td>Integer</td>
<td>Integer comparison=cpli</td>
</tr>
<tr>
<td>21</td>
<td><code>cmpw cr7,r0,r2</code></td>
<td>Integer</td>
<td>Compare Logical Word</td>
</tr>
<tr>
<td>22</td>
<td><code>cmpwi cr7,r0,1</code></td>
<td>Integer</td>
<td>Compare Logical Word</td>
</tr>
<tr>
<td>23</td>
<td><code>divw r2,r9,r0</code></td>
<td>Integer</td>
<td>Integer arithmetic</td>
</tr>
<tr>
<td>24</td>
<td><code>extsh r0,r0</code></td>
<td>Logical</td>
<td>Extend sign half word</td>
</tr>
<tr>
<td>25</td>
<td><code>extsb r0,r0</code></td>
<td>Logical</td>
<td>Extend sign byte</td>
</tr>
<tr>
<td>26</td>
<td><code>fadd f0,f12,f0</code></td>
<td>FP</td>
<td>FP add</td>
</tr>
<tr>
<td>27</td>
<td><code>fadds f0,f13,f0</code></td>
<td>FP</td>
<td>FP add to single</td>
</tr>
<tr>
<td>28</td>
<td><code>fcmpu cr7,f0,f13</code></td>
<td>FP</td>
<td>Floating point comparison</td>
</tr>
<tr>
<td>29</td>
<td><code>fsub f0,f12,f0</code></td>
<td>FP</td>
<td>FP sub</td>
</tr>
<tr>
<td>30</td>
<td><code>fsubs f0,f12,f0</code></td>
<td>FP</td>
<td>FP sub single</td>
</tr>
<tr>
<td>31</td>
<td><code>fmr f0,f1</code></td>
<td>FPSCR</td>
<td>FP move</td>
</tr>
<tr>
<td>32</td>
<td><code>fmul f12,f13,f0</code></td>
<td>FP</td>
<td>FP mul</td>
</tr>
<tr>
<td>33</td>
<td><code>fmuls f0,f13,f0</code></td>
<td>FP</td>
<td>FP mul single</td>
</tr>
<tr>
<td>34</td>
<td><code>la r5,lo16(_FCNAME.0-&quot;L00000000001$pb&quot;)(r5)</code></td>
<td>Addi</td>
<td>Equivalent mnemonic</td>
</tr>
<tr>
<td>35</td>
<td><code>lbz r9,0(r9)</code></td>
<td>Storage access</td>
<td>Load Byte and Zero</td>
</tr>
<tr>
<td>36</td>
<td><code>lfd f13,152(r30)</code></td>
<td>FP</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td><code>lfs f13,0(r9)</code></td>
<td>FP</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td><code>lha r2,236(r30)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td><code>lh2 r2,8(r2)</code></td>
<td>Storage access</td>
<td>Load half word and Zero</td>
</tr>
<tr>
<td>40</td>
<td><code>li r0,0</code></td>
<td>Addi</td>
<td>Equivalent mnemonic</td>
</tr>
<tr>
<td>41</td>
<td><code>lis r2,0x400</code></td>
<td>Addis</td>
<td>Equivalent mnemonic</td>
</tr>
<tr>
<td>42</td>
<td><code>lmw r30,-8(r1)</code></td>
<td>Storage access</td>
<td>Load Multiple word</td>
</tr>
<tr>
<td>43</td>
<td><code>lwz r0,0(r2)</code></td>
<td>Storage access</td>
<td>Load Word and Zero</td>
</tr>
<tr>
<td>44</td>
<td><code>lwz r2,lo16(L_MPIR_Process$n on_layz_ptr-&quot;L00000000001$pb&quot;)(r2)</code></td>
<td>Storage access</td>
<td>Load Word and Zero</td>
</tr>
<tr>
<td>Line</td>
<td>Instruction</td>
<td>Category</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------------</td>
<td>----------------------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td>45</td>
<td>lwzu r12,lo16(L_MPIR_Err_return_comm$lazy_ptr-LO$, MPIR_Err_return_comm)(r11)</td>
<td>Storage access</td>
<td>Load Word and Zero with Update</td>
</tr>
<tr>
<td>46</td>
<td>mfcr r0</td>
<td></td>
<td>Move from condition register</td>
</tr>
<tr>
<td>47</td>
<td>mflr r0</td>
<td>Equivalent to Mfspr</td>
<td>Move from special purpose register</td>
</tr>
<tr>
<td>48</td>
<td>mr r30,r1</td>
<td>Equivalent to Mfspr</td>
<td>Move from special purpose register</td>
</tr>
<tr>
<td>49</td>
<td>mtctr r12</td>
<td>Mtspres</td>
<td>Move to Special purpose register</td>
</tr>
<tr>
<td>50</td>
<td>mtlr r0</td>
<td>Mtspres</td>
<td>Move to Special purpose register</td>
</tr>
<tr>
<td>51</td>
<td>muli r9,r0,200</td>
<td>Integer arithmetic</td>
<td>Multiply low immediate</td>
</tr>
<tr>
<td>52</td>
<td>mulhwu r9,r2,r0</td>
<td>Integer arithmetic</td>
<td>Multiply high word unsigned</td>
</tr>
<tr>
<td>53</td>
<td>mullw r0,r2,r0</td>
<td>Integer arithmetic</td>
<td>Multiply low word</td>
</tr>
<tr>
<td>54</td>
<td>neg r0,r0</td>
<td>Integer arithmetic</td>
<td>Negate</td>
</tr>
<tr>
<td>55</td>
<td>ori r5,r5,271</td>
<td>Logical Instruction</td>
<td>OR immediate</td>
</tr>
<tr>
<td>56</td>
<td>oris r0,r0,0x8000</td>
<td>Logical Instruction</td>
<td>OR immediate shifted</td>
</tr>
<tr>
<td>57</td>
<td>or r0,r0,r2</td>
<td>Logical Instruction</td>
<td>OR</td>
</tr>
<tr>
<td>58</td>
<td>rlwinmm r0,r0,0,2,5</td>
<td>Rotate &amp; shift</td>
<td>Rotate Left Word Immediate then Insert Mask</td>
</tr>
<tr>
<td>59</td>
<td>slwi r0,r0,2</td>
<td>Rotate &amp; shift</td>
<td>Shift left word immediate</td>
</tr>
<tr>
<td>60</td>
<td>slw r2,r2,r0</td>
<td>Rotate &amp; shift</td>
<td>Shift Left Word</td>
</tr>
<tr>
<td>61</td>
<td>srawi r0,r0,26</td>
<td>Rotate &amp; shift</td>
<td>Shift Right Algebraic Word Immediate</td>
</tr>
<tr>
<td>62</td>
<td>srwi r0,r0,30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>stb r0,8(r2)</td>
<td>Storage access</td>
<td>Store Byte</td>
</tr>
<tr>
<td>64</td>
<td>std f0,144(r30)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>stfs f0,0(r11)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>sth r0,8(r9)</td>
<td>Storage access</td>
<td>Store Half Word</td>
</tr>
<tr>
<td></td>
<td>Instruction</td>
<td>Category</td>
<td>Description</td>
</tr>
<tr>
<td>---</td>
<td>-------------------</td>
<td>-------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>67</td>
<td>stmw r30,-8(r1)</td>
<td>Storage access</td>
<td>Store Multiple Word</td>
</tr>
<tr>
<td>68</td>
<td>stw r0,8(r1)</td>
<td>Storage access</td>
<td>Store Word</td>
</tr>
<tr>
<td>69</td>
<td>stwu r1,-144(r1)</td>
<td>Storage access</td>
<td>Store Word with Update</td>
</tr>
<tr>
<td>70</td>
<td>subf r0,r0,r2</td>
<td>Integer arithmetic</td>
<td>Subtract From</td>
</tr>
<tr>
<td>71</td>
<td>xori r0,r0,2048</td>
<td>Logical Instruction</td>
<td>XOR Immediate</td>
</tr>
<tr>
<td>72</td>
<td>xor r0,r2,r0</td>
<td>Logical Instruction</td>
<td>XOR</td>
</tr>
</tbody>
</table>

**Table 21: MPICH instruction set**
Appendix C. Cache and its analysis

C.1 Access time analysis:
Cache studies carried out previously to understand certain design parameters involved in designing the cache. Variation of the access time with Technology node is shown in Figure 131. CACTI calculates the access time by linear scaling against another node whose technology parameters have been included in the program. The access time is seen to decrease as the transistors become smaller and the separation between the transistors decreases. This in turn reduces the local wire delays.

Figure 131: Access time vs. technology node as calculated by CACTI, which uses a linear scaling approach to calculate access time for different nodes.

Cache access time variations with associativity are also analyzed in Figure 132. The access time is seen to increase as the complexity of the circuits for decoder etc increases for higher associativity.
**Figure 132:** Access time variation with the Associativity of the cache.

Access time variation with the number of banks is seen in Figure 133.

**Figure 133:** Access time variation with the number of banks used in the cache.

Figure 108 and 109 help us decide the optimal configuration of the cache. Increasing banks is seen to decrease the access time. But after a certain number of banks, the decrease is gradual. Also the complexity of the cache implementation is also higher as the number of banks is increased.
Figure 134: Access time variations in 3D memory a: without sub-banks b: with sub-banks
Appendix D. Verilog files for CPU core

**Instruction Decoder:**

```verilog
`timescale 1 ns / 1 ps
module inst_decode(
    ADDR,
    INSTR,
    LD,
    ST,
    BR,
    NOOP,
    ALU_OP,
    ADDR_O,
    DESt,
    Select_ALU,
    SRC1,
    SRC2
);

input [15:0] ADDR;
input [31:16] INSTR;
output LD;
output ST;
output BR;
output NOOP;
output ALU_OP;
output [15:0] ADDR_O;
output [26:24] DESt;
output [2:0] Select_ALU;
output [22:20] SRC1;
output [18:16] SRC2;

wire [2:0] Sel_ALU;
wire SYNTHESIZED_WIRE_0;
wire SYNTHESIZED_WIRE_1;
wire SYNTHESIZED_WIRE_2;
wire SYNTHESIZED_WIRE_3;
wire SYNTHESIZED_WIRE_4;
wire SYNTHESIZED_WIRE_5;
wire SYNTHESIZED_WIRE_6;

assign ST = SYNTHESIZED_WIRE_5;
assign BR = SYNTHESIZED_WIRE_1;
assign SYNTHESIZED_WIRE_0 = 1;
assign SYNTHESIZED_WIRE_3 = 1;
assign SYNTHESIZED_WIRE_4 = 1;
assign SYNTHESIZED_WIRE_6 = 1;
```
tri_buf3 b2v_inst1(
    .enabledt(SYNTHESIZED_WIRE_0),
    .data(INSTR[26:24]),
    .tridata(DESTr)
);

assign Sel_ALU[0] = SYNTHESIZED_WIRE_1 | INSTR[29];

decoder2_4 b2v_inst3(
    .enable(SYNTHESIZED_WIRE_2),
    .data(INSTR[30:29]),
    .eq0(NOOP),
    .eq1(SYNTHESIZED_WIRE_5),
    .eq2(SYNTHESIZED_WIRE_1),
    .eq3(LD));

tri_buf3 b2v_inst4(
    .enabledt(SYNTHESIZED_WIRE_3),
    .data(INSTR[22:20]),
    .tridata(SRC1)
);

tri_buf3 b2v_inst5(
    .enabledt(SYNTHESIZED_WIRE_4),
    .data(INSTR[18:16]),
    .tridata(SRC2)
);

assign Sel_ALU[1] = SYNTHESIZED_WIRE_5 | INSTR[30];

tri_buf16 b2v_inst7(
    .enabledt(SYNTHESIZED_WIRE_6),
    .data(ADDR),
    .tridata(ADDR_O)
);

soft b2v_inst8(
    .in(INSTR[31]),
    .out(Sel_ALU[2]));

assign SYNTHESIZED_WIRE_2 = ~INSTR[28];
assign    ALU_OP = INSTR[28];
assign    Select_ALU = Sel_ALU;

endmodule

RF2:

module RF2(
    LOAD,
    STORE,
    R_WB_enable_A_B,
    NOP,
    CLK,
    BR,
    ALU_OP,
    Freeze_b,
    clear,
    ADDR,
    ALU_SEL,
    DATA_IN,
    DestR,
    SRC1_IN,
    SRC2_IN,
    WB_REG,
    NOOP,
    LOAD_o,
    STORE_o,
    BR_O,
    ALU_OP_O,
    ADDR_O,
    DESTr_OUT,
    READ_OUT_A,
    READ_OUT_B,
    SEL_ALU
);

input LOAD;
input STORE;
input R_WB_enable_A_B;
input NOP;
input CLK;
input BR;
input ALU_OP;
input Freeze_b;
input clear;
input [15:0] ADDR;
input [2:0] ALU_SEL;
input [31:0] DATA_IN;
input [2:0] DestR;
input [8:6] SRC1_IN;
input [5:3] SRC2_IN;
input [2:0] WB_REG;
output   NOOP;
output   LOAD_o;
output   STORE_o;
output   BR_O;
output   ALU_OP_O;
output   [15:0] ADDR_O;
output   [11:9] DESTr_OUT;
output   [31:0] READ_OUT_A;
output   [31:0] READ_OUT_B;
output   [2:0] SEL_ALU;

wire SYNTHESIZED_WIRE_2;
wire SYNTHESIZED_WIRE_17;
wire SYNTHESIZED_WIRE_18;
wire SYNTHESIZED_WIRE_4;
wire SYNTHESIZED_WIRE_7;
wire SYNTHESIZED_WIRE_10;
wire SYNTHESIZED_WIRE_11;
wire SYNTHESIZED_WIRE_12;
wire SYNTHESIZED_WIRE_13;

assign SYNTHESIZED_WIRE_2 = 1;
assign SYNTHESIZED_WIRE_11 = 1;
assign SYNTHESIZED_WIRE_12 = 1;
assign SYNTHESIZED_WIRE_13 = 1;

rf_mem   b2v_inst(  
    .wren(R_WB_enable_A_B),  
    .clock(CLK),  
    .data(DATA_IN),  
    .rdaddress_a(SRC1_IN),  
    .rdaddress_b(SRC2_IN),  
    .wraddress(WB_REG),  
    .qa(SYNTHESIZED_WIRE_17),  
    .qb(SYNTHESIZED_WIRE_19));

soft   b2v_inst1(  
    .in(LOAD),  
    .out(LOAD_o));

DFF_32   b2v_inst10(  
    .clock(CLK),  
    .enable(SYNTHESIZED_WIRE_16),
.aclr(clear),
.data(SYNTHESIZED_WIRE_17),
.q(SYNTHESIZED_WIRE_4));

DFF_1 b2v_inst14(
  .data(Freeze_b),
  .clock(CLK),
  .enable(SYNTHESIZED_WIRE_2),
  .aclr(clear),
  .q(SYNTHESIZED_WIRE_18));

data32x2_mux b2v_inst16(
  .sel(SYNTHESIZED_WIRE_18),
  .data0x(SYNTHESIZED_WIRE_4),
  .data1x(SYNTHESIZED_WIRE_17),
  .result(READ_OUT_A));

data32x2_mux b2v_inst17(
  .sel(SYNTHESIZED_WIRE_18),
  .data0x(SYNTHESIZED_WIRE_7),
  .data1x(SYNTHESIZED_WIRE_19),
  .result(READ_OUT_B));

soft b2v_inst2(
  .in(STORE),
  .out(STORE_o));

assign SYNTHESIZED_WIRE_10 = ~Freeze_b;

soft b2v_inst3(
  .in(NOP),
  .out(NOOP));

assign SYNTHESIZED_WIRE_16 = SYNTHESIZED_WIRE_18 & SYNTHESIZED_WIRE_10;

soft b2v_inst4(
  .in(BR),
  .out(BR_O));

tri_buf3 b2v_inst5(
DFF_32 b2v_inst9(  
  .clock(CLK),  
  .enable(SYNTHESIZED_WIRE_16),  
  .aclr(clear),  
  .data(SYNTHESIZED_WIRE_19),  
  .q(SYNTHESIZED_WIRE_7));

endmodule

**Operand preparation stage:**

`timescale 1ns / 1ps

module op_prep_32bit(  
  A,  
  B,  
  SEL_ALU,  
  Z  
);  

input [31:0] A;  
input [31:0] B;  
input [2:0] SEL_ALU;  
output [31:0] Z;
wire [31:0] Z_ALTERA_SYNTHESIZED;
wire SYNTHESIZED_WIRE_0;
wire SYNTHESIZED_WIRE_1;
wire SYNTHESIZED_WIRE_2;
wire SYNTHESIZED_WIRE_3;
wire SYNTHESIZED_WIRE_4;
wire SYNTHESIZED_WIRE_5;
wire SYNTHESIZED_WIRE_6;
wire SYNTHESIZED_WIRE_7;

assign SYNTHESIZED_WIRE_0 = 0;
assign SYNTHESIZED_WIRE_1 = 0;
assign SYNTHESIZED_WIRE_2 = 0;
assign SYNTHESIZED_WIRE_3 = 0;
assign SYNTHESIZED_WIRE_4 = 0;
assign SYNTHESIZED_WIRE_5 = 0;
assign SYNTHESIZED_WIRE_6 = 0;
assign SYNTHESIZED_WIRE_7 = 0;

\74381   b2v_inst(  
    .A1(A[29]),
    .CIN(SYNTHESIZED_WIRE_0),
    .B1(B[29]),
    .A0(A[28]),
    .B0(B[28]),
    .A3(A[31]),
    .A2(A[30]),
    .B3(B[31]),
    .B2(B[30]),
    .S2(SEL_ALU[2]),
    .S1(SEL_ALU[1]),
    .S0(SEL_ALU[0]),
    .F0(Z_ALTERA_SYNTHESIZED[28]),
    .F1(Z_ALTERA_SYNTHESIZED[29]),
    .F3(Z_ALTERA_SYNTHESIZED[31]),
    .F2(Z_ALTERA_SYNTHESIZED[30])
  );

\74381   b2v_inst11(  
    .CIN(SYNTHESIZED_WIRE_1),
    .B1(B[9]),
    .A0(A[8]),
    .B0(B[8]),
    .A2(A[10]),
    .B3(B[11]),
    .B2(B[10]),
    .S2(SEL_ALU[2]),
  );
.S1(SEL_ALU[1]),
.S0(SEL_ALU[0]),
.F0(Z_ALTERA_SYNTHESIZED[8]),
.F1(Z_ALTERA_SYNTHESIZED[9]),
.F3(Z_ALTERA_SYNTHESIZED[11]),
.F2(Z_ALTERA_SYNTHESIZED[10])
);

\74381 b2v_inst13(
 .CIN(SYNTHESIZED_WIRE_2),
 .B1(B[5]),
 .A0(A[4]),
 .B0(B[4]),
 .A3(A[7]),
 .A2(A[6]),
 .B3(B[7]),
 .B2(B[6]),
 .S2(SEL_ALU[2]),
 .S1(SEL_ALU[1]),
 .S0(SEL_ALU[0]),
 .F0(Z_ALTERA_SYNTHESIZED[4]),
 .F1(Z_ALTERA_SYNTHESIZED[5]),
 .F3(Z_ALTERA_SYNTHESIZED[7]),
 .F2(Z_ALTERA_SYNTHESIZED[6])
);

\74381 b2v_inst15(
 .A1(A[1]),
 .CIN(SYNTHESIZED_WIRE_3),
 .B1(B[1]),
 .A0(A[0]),
 .B0(B[0]),
 .A3(A[3]),
 .A2(A[2]),
 .B3(B[3]),
 .B2(B[2]),
 .S2(SEL_ALU[2]),
 .S1(SEL_ALU[1]),
 .S0(SEL_ALU[0]),
 .F0(Z_ALTERA_SYNTHESIZED[0]),
 .F1(Z_ALTERA_SYNTHESIZED[1]),
 .F3(Z_ALTERA_SYNTHESIZED[3]),
 .F2(Z_ALTERA_SYNTHESIZED[2])
);
\74381  b2v_inst3(  
  .CIN(SYNTHESIZED_WIRE_4),  
  .B1(B[25]),  
  .A0(A[24]),  
  .B0(B[24]),  
  .A3(A[27]),  
  .A2(A[26]),  
  .B3(B[27]),  
  .B2(B[26]),  
  .S2(SEL_ALU[2]),  
  .S1(SEL_ALU[1]),  
  .S0(SEL_ALU[0]),  
  .F0(Z_ALTERA_SYNTHESIZED[24]),  
  .F1(Z_ALTERA_SYNTHESIZED[25]),  
  .F3(Z_ALTERA_SYNTHESIZED[27]),  
  .F2(Z_ALTERA_SYNTHESIZED[26])
);

\74381  b2v_inst5(  
  .CIN(SYNTHESIZED_WIRE_5),  
  .B1(B[21]),  
  .A0(A[20]),  
  .B0(B[20]),  
  .A3(A[23]),  
  .A2(A[22]),  
  .B3(B[23]),  
  .B2(B[22]),  
  .S2(SEL_ALU[2]),  
  .S1(SEL_ALU[1]),  
  .S0(SEL_ALU[0]),  
  .F0(Z_ALTERA_SYNTHESIZED[20]),  
  .F1(Z_ALTERA_SYNTHESIZED[21]),  
  .F3(Z_ALTERA_SYNTHESIZED[23]),  
  .F2(Z_ALTERA_SYNTHESIZED[22])
);

\74381  b2v_inst7(  
  .A1(A[17]),  
  .CIN(SYNTHESIZED_WIRE_6),  
  .B1(B[17]),  
  .A0(A[16]),  
  .B0(B[16]),  
  .A3(A[19]),  
  .A2(A[18]),
\74381 b2v_inst9(
  .CIN(SYNTHESIZED_WIRE_7),
  .B1(B[13]),
  .A0(A[12]),
  .B0(B[12]),
  .A3(A[15]),
  .A2(A[14]),
  .B3(B[15]),
  .B2(B[14]),
  .S2(SEL_ALU[2]),
  .S1(SEL_ALU[1]),
  .S0(SEL_ALU[0]),
  .F0(Z_ALTERA_SYNTHESIZED[16]),
  .F1(Z_ALTERA_SYNTHESIZED[17]),
  .F3(Z_ALTERA_SYNTHESIZED[19]),
  .F2(Z_ALTERA_SYNTHESIZED[18])
);

assign    Z = Z_ALTERA_SYNTHESIZED;
endmodule

ALU Stg 1:
`timescale 1ns / 1ps
module ALU_Stgl(
  NOP,
  LOAD,
  STORE,
  BR,
  ALU_OP,
  ADDR,
  ALU_SEL,
  DATA_A1,
  DATA_B2,
  DESTr,
input NOP;
input LOAD;
input STORE;
input BR;
input ALU_OP;
input [15:0] ADDR;
input [2:0] ALU_SEL;
input [31:0] DATA_A1;
input [31:0] DATA_B2;
input [2:0] DEST;
output NOOP;
output LOAD_o;
output STORE_o;
output BR_O;
output ALU_OP_o;
output [31:0] DATA_A_OUT;
output [31:0] DATA_B_OUT;
output [2:0] SEL_ALU;
output [31:0] STORE_DATA;
output [2:0] ylatch_o;
wire [31:0] data;
wire [1:1] SELB;
wire [31:0] SYNTHESIZED_WIRE_0;
wire SYNTHESIZED_WIRE_1;
wire [31:0] SYNTHESIZED_WIRE_2;
wire SYNTHESIZED_WIRE_3;
wire SYNTHESIZED_WIRE_4;
wire SYNTHESIZED_WIRE_5;
assign SYNTHESIZED_WIRE_1 = 1;
assign SYNTHESIZED_WIRE_3 = 1;
assign SYNTHESIZED_WIRE_4 = 1;
assign SYNTHESIZED_WIRE_5 = 1;

baseReg b2v_inst(
    .result(SYNTHESIZED_WIRE_0));
data32x2_mux b2v_inst1(
    .sel(SELB),
    .data0x(DATA_A1),
    .data1x(SYNTHESIZED_WIRE_0),
    .result(SYNTHESIZED_WIRE_2));

soft b2v_inst10(
    .in(LOAD),
    .out(LOAD_o));

soft b2v_inst11(
    .in(STORE),
    .out(STORE_o));

soft b2v_inst16(
    .in(BR),
    .out(BR_O));

soft b2v_inst17(
    .in(ALU_OP),
    .out(ALU_OP_O));

assign SELB = STORE | LOAD;

data32x2_mux b2v_inst4(
    .sel(SELB),
    .data0x(DATA_B2),
    .data1x(data),
    .result(DATA_B_OUT));

tri_buf32 b2v_inst5(
    .enabledt(SYNTHESIZED_WIRE_1),
    .data(SYNTHESIZED_WIRE_2),
    .tridata(DATA_A_OUT)
);

tri_buf3 tri_buf6(
    .enabledt(SYNTHESIZED_WIRE_3),
    .data(DESTr),
    .tridata(ylatch_o)
);

ALU stage:

module ALU(
    NOOP_IN,
    LOAD,
    STORE,
    }
BR,
ALU_SEL,
DATA_A,
DATA_B,
DestR,
ST_DATA,
NOOP_OUT,
LOAD_o,
STORE_o,
BR_o,
ALU_OUT,
DestR_o,
STO_DATA
);

input NOOP_IN;
input LOAD;
input STORE;
input BR;
input [2:0] ALU_SEL;
input [31:0] DATA_A;
input [31:0] DATA_B;
input [2:0] DestR;
input [31:0] ST_DATA;
output NOOP_OUT;
output LOAD_o;
output STORE_o;
output BR_o;
output [31:0] ALU_OUT;
output [2:0] DestR_o;
output [31:0] STO_DATA;

wire ADD;
wire SUB;
wire SYNTHESIZED_WIRE_0;
wire SYNTHESIZED_WIRE_1;
wire SYNTHESIZED_WIRE_2;
wire [31:0] SYNTHESIZED_WIRE_3;
wire [31:0] SYNTHESIZED_WIRE_4;
wire SYNTHESIZED_WIRE_5;
wire SYNTHESIZED_WIRE_6;

assign SYNTHESIZED_WIRE_0 = 1;
assign SYNTHESIZED_WIRE_6 = 1;

decode_3_8 b2v_inst(
    .data(ALU_SEL),
    .eq2(SUB),
    .eq3(ADD));

assign SYNTHESIZED_WIRE_5 = ~SUB;
soft b2v_inst10(  
    .in(STORE),  
    .out(STORE_o));

tri_buf3 b2v_inst11(  
    .enabledt(SYNTHESIZED_WIRE_0),  
    .data(DestR),  
    .tridata(DestR_o)  
);

soft b2v_inst14(  
    .in(BR),  
    .out(BR_o));

op_prep_32bit b2v_inst2(  
    .A(DATA_A),  
    .B(DATA_B),  
    .SEL_ALU(ALU_SEL),  
    .Z(SYNTHESIZED_WIRE_3));

assign    SYNTHESIZED_WIRE_2 = SUB ^ ADD;

adder_32 b2v_inst4(  
    .add_sub(SYNTHESIZED_WIRE_1),  
    .dataa(DATA_A),  
    .datab(DATA_B),  
    .result(SYNTHESIZED_WIRE_4));

data32x2_mux b2v_inst5(  
    .sel(SYNTHESIZED_WIRE_2),  
    .data0x(SYNTHESIZED_WIRE_3),  
    .data1x(SYNTHESIZED_WIRE_4),  
    .result(ALU_OUT));

assign    SYNTHESIZED_WIRE_1 = ADD & SYNTHESIZED_WIRE_5;

tri_buf32 b2v_inst7(  
    .enabledt(SYNTHESIZED_WIRE_6),  
    .data(ST_DATA),  
    .tridata(STO_DATA)  
);

soft b2v_inst8(  
    .in(NOOP_IN),  
    .out(NOOP_OUT));

soft b2v_inst9(  
    .in(LOAD),  
    .out(LOAD_o));

endmodule
soft b2v_inst7(
    .in(NOP),
    .out(NOOP));

tri_buf3 b2v_inst8(
    .enabledt(SYNTHESIZED_WIRE_4),
    .data(ALU_SEL),
    .tridata(SEL_ALU)
);

tri_buf32 b2v_inst9(
    .enabledt(SYNTHESIZED_WIRE_5),
    .data(DATA_A1),
    .tridata(STORE_DATA)
);

assign data[15:0] = ADDR;
assign data[31:16] = 16’b0000000000000000;
endmodule

Pipeline controller:

`timescale 1 ns / 1 ps
module pipe_controller(
    SET,
    HLT,
    Stall_Cache,
    Unstall_Cache,
    Clear_Br,
    Clk,
    Clear_2345,
    A,
    B,
    X,
    Y,
    Z,
    Clear_1,
    Freeze_b
);

input SET;
input HLT;
input Stall_Cache;
input Unstall_Cache;
input Clear_Br;
input Clk;
output Clear_2345;
output A;
output B;
output X;
output Y;
output Z;
output Clear_1;
output Freeze_b;

reg [1:0] data;
wire data0_b;
wire data1_b;
wire X_ALTERA_SYNTHESIZED;
wire X_b;
wire Y_ALTERA_SYNTHESIZED;
wire Y_b;
wire Z_ALTERA_SYNTHESIZED;
wire Z_b;
wire SYNTHESIZED_WIRE_0;
wire SYNTHESIZED_WIRE_1;
wire SYNTHESIZED_WIRE_2;
wire SYNTHESIZED_WIRE_3;
wire SYNTHESIZED_WIRE_4;
wire SYNTHESIZED_WIRE_5;
wire SYNTHESIZED_WIRE_6;
wire SYNTHESIZED_WIRE_59;
wire SYNTHESIZED_WIRE_8;
wire SYNTHESIZED_WIRE_9;
wire SYNTHESIZED_WIRE_10;
wire SYNTHESIZED_WIRE_60;
wire SYNTHESIZED_WIRE_12;
wire SYNTHESIZED_WIRE_61;
wire SYNTHESIZED_WIRE_62;
wire SYNTHESIZED_WIRE_18;
wire SYNTHESIZED_WIRE_19;
wire SYNTHESIZED_WIRE_20;
wire SYNTHESIZED_WIRE_21;
wire SYNTHESIZED_WIRE_22;
wire SYNTHESIZED_WIRE_26;
wire SYNTHESIZED_WIRE_27;
wire SYNTHESIZED_WIRE_64;
wire SYNTHESIZED_WIRE_32;
wire SYNTHESIZED_WIRE_33;
wire SYNTHESIZED_WIRE_34;
wire SYNTHESIZED_WIRE_35;
wire SYNTHESIZED_WIRE_38;
wire SYNTHESIZED_WIRE_39;
wire SYNTHESIZED_WIRE_40;
wire SYNTHESIZED_WIRE_41;
wire SYNTHESIZED_WIRE_42;
wire SYNTHESIZED_WIRE_43;
wire SYNTHESIZED_WIRE_44;
wire SYNTHESIZED_WIRE_45;
wire SYNTHESIZED_WIRE_46;
wire SYNTHESIZED_WIRE_47;
wire SYNTHESIZED_WIRE_48;
wire SYNTHESIZED_WIRE_49;
wire SYNTHESIZED_WIRE_50;
wire SYNTHESIZED_WIRE_51;
wire SYNTHESIZED_WIRE_52;
wire SYNTHESIZED_WIRE_53;
wire SYNTHESIZED_WIRE_54;
wire SYNTHESIZED_WIRE_55;
wire SYNTHESIZED_WIRE_56;
wire SYNTHESIZED_WIRE_57;
wire SYNTHESIZED_WIRE_58;

assign Clear_1 = SYNTHESIZED_WIRE_51;
assign Y_ALTERA_SYNTHESIZED = SYNTHESIZED_WIRE_0 | SYNTHESIZED_WIRE_1 | SYNTHESIZED_WIRE_2;
assign SYNTHESIZED_WIRE_5 = data1_b & data[0];
assign SYNTHESIZED_WIRE_6 = X_b & Y_ALTERA_SYNTHESIZED;
assign SYNTHESIZED_WIRE_41 = SYNTHESIZED_WIRE_3 & SYNTHESIZED_WIRE_4;
assign SYNTHESIZED_WIRE_19 = data[1] & X_b;
assign SYNTHESIZED_WIRE_20 = Y_ALTERA_SYNTHESIZED & Z_ALTERA_SYNTHESIZED;
assign SYNTHESIZED_WIRE_21 = data[1] & data0_b;
assign SYNTHESIZED_WIRE_22 = Y_ALTERA_SYNTHESIZED & Z_b;
assign SYNTHESIZED_WIRE_47 = SYNTHESIZED_WIRE_5 & SYNTHESIZED_WIRE_6;
assign SYNTHESIZED_WIRE_61 = ~SET;
assign SYNTHESIZED_WIRE_62 = ~HLT;
assign SYNTHESIZED_WIRE_64 = ~Stall_Cache;
assign SYNTHESIZED_WIRE_63 = ~Unstall_Cache;
assign SYNTHESIZED_WIRE_59 = ~Clear_Br;
assign Z_b = ~Z_ALTERA_SYNTHESIZED;
assign X_b = ~X_ALTERA_SYNTHESIZED;
assign SYNTHESIZED_WIRE_8 = X_b & Y_b;
assign   Y_b = ~Y_ALTERA_SYNTHESIZED;
assign   data0_b = ~data[0];
assign   data1_b = ~data[1];
assign   SYNTHESIZED_WIRE_9 = Z_b & data0_b;
assign   SYNTHESIZED_WIRE_27 = Unstall_Cache & SYNTHESIZED_WIRE_59;
assign   SYNTHESIZED_WIRE_49 = SYNTHESIZED_WIRE_8 & SYNTHESIZED_WIRE_9;
assign   SYNTHESIZED_WIRE_0 = SYNTHESIZED_WIRE_10 & SYNTHESIZED_WIRE_60;
assign   SYNTHESIZED_WIRE_1 = SYNTHESIZED_WIRE_12 & SYNTHESIZED_WIRE_60;
assign   SYNTHESIZED_WIRE_26 = SYNTHESIZED_WIRE_61 & SYNTHESIZED_WIRE_62;
assign   SYNTHESIZED_WIRE_60 = SYNTHESIZED_WIRE_61 & SYNTHESIZED_WIRE_62;
assign   Freeze_b = ~SYNTHESIZED_WIRE_18;
assign   SYNTHESIZED_WIRE_54 = SYNTHESIZED_WIRE_19 & SYNTHESIZED_WIRE_20;
assign   SYNTHESIZED_WIRE_53 = SYNTHESIZED_WIRE_21 & SYNTHESIZED_WIRE_22;
assign   SYNTHESIZED_WIRE_33 = SYNTHESIZED_WIRE_61 & SYNTHESIZED_WIRE_59 & SYNTHESIZED_WIRE_63;
assign   X_ALTERA_SYNTHESIZED = SYNTHESIZED_WIRE_26 & SYNTHESIZED_WIRE_27;
assign   SYNTHESIZED_WIRE_55 = SYNTHESIZED_WIRE_64 & SYNTHESIZED_WIRE_63 & Clear_Br;
assign   SYNTHESIZED_WIRE_10 = Stall_Cache & SYNTHESIZED_WIRE_59 & SYNTHESIZED_WIRE_63;
assign   SYNTHESIZED_WIRE_32 = Stall_Cache ^ HLT;
assign   Z_ALTERA_SYNTHESIZED = SYNTHESIZED_WIRE_32 & SYNTHESIZED_WIRE_33;
always@(posedge Clk)
begin
  begin
    data[1] = SYNTHESIZED_WIRE_34;
  end
end

always@(posedge Clk)
begin
  begin
    data[0] = SYNTHESIZED_WIRE_35;
  end
end

assign    SYNTHESIZED_WIRE_50 = X_b & data[1] & Z_b;
assign    SYNTHESIZED_WIRE_46 = X_b & Y_b & Z_b;
assign    SYNTHESIZED_WIRE_42 = data[1] & Y_ALTERA_SYNTHESIZED;
assign    SYNTHESIZED_WIRE_12 = SYNTHESIZED_WIRE_64 &
SYNTHESIZED_WIRE_59 & Unstall_Cache;
assign    SYNTHESIZED_WIRE_40 = X_ALTERA_SYNTHESIZED & Z_b &
data[0];
assign    SYNTHESIZED_WIRE_45 = data[0] & X_ALTERA_SYNTHESIZED &
& Z_b;

lpm_decode0     b2v_inst55(
  .data(data),
  .eq0(SYNTHESIZED_WIRE_51),
  .eq1(SYNTHESIZED_WIRE_18),
  .eq2(SYNTHESIZED_WIRE_52)
);

assign    SYNTHESIZED_WIRE_44 = SYNTHESIZED_WIRE_38 |
SYNTHESIZED_WIRE_39;
assign    SYNTHESIZED_WIRE_43 = SYNTHESIZED_WIRE_40 |
SYNTHESIZED_WIRE_41;
assign    SYNTHESIZED_WIRE_38 = SYNTHESIZED_WIRE_42 & Z_b;
assign    SYNTHESIZED_WIRE_34 = SYNTHESIZED_WIRE_43 |
SYNTHESIZED_WIRE_44;
assign    SYNTHESIZED_WIRE_48 = SYNTHESIZED_WIRE_45 |
SYNTHESIZED_WIRE_46;
assign    SYNTHESIZED_WIRE_58 = SYNTHESIZED_WIRE_47 |
SYNTHESIZED_WIRE_48;
assign SYNTHESIZED_WIRE_39 = SYNTHESIZED_WIRE_49 | SYNTHESIZED_WIRE_50;

assign Clear_2345 = SYNTHESIZED_WIRE_51 | SYNTHESIZED_WIRE_52;

assign SYNTHESIZED_WIRE_57 = SYNTHESIZED_WIRE_53 | SYNTHESIZED_WIRE_54;

assign SYNTHESIZED_WIRE_2 = SYNTHESIZED_WIRE_55 & SYNTHESIZED_WIRE_60;

assign SYNTHESIZED_WIRE_35 = SYNTHESIZED_WIRE_57 | SYNTHESIZED_WIRE_58;

assign SYNTHESIZED_WIRE_3 = data[1] & data0_b;

assign SYNTHESIZED_WIRE_4 = X_b & Y_ALTERA_SYNTHESIZED;

assign A = data[1];
assign B = data[0];
assign X = X_ALTERA_SYNTHESIZED;
assign Y = Y_ALTERA_SYNTHESIZED;
assign Z = Z_ALTERA_SYNTHESIZED;

endmodule

Data RAM:
`timescale 1 ps / 1 ps
// synopsys translate_on
module d_ram (data, rdaddress, rdclock, wraddress, wrclock, wren, q);

input [31:0] data;
input [7:0] rdaddress;
input rdclock;
input [7:0] wraddress;
input wrclock;
input wren;
output [31:0] q;
wire [31:0] sub_wire0;
wire [31:0] q = sub_wire0[31:0];

altsyncram altsyncram_component (}
.wren_a (wren),
.clock0 (wrclock),
.clock1 (rdclock),
.address_a (wraddress),
.address_b (rdaddress),
.data_a (data),
.q_b (sub_wire0),
.aclr0 (1'b0),
.aclr1 (1'b0),
.addressstall_a (1'b0),
.addressstall_b (1'b0),
.byteena_a (1'b1),
.byteena_b (1'b1),
.clocken0 (1'b1),
.clocken1 (1'b1),
.clocken2 (1'b1),
.clocken3 (1'b1),
.data_b ({32{1'b1}}),
.eccstatus (),
.q_a (),
.rden_a (1'b1),
.rden_b (1'b1),
.wren_b (1'b0));
defparam
  altsyncram_component.address_aclr_b = "NONE",
  altsyncram_component.address_reg_b = "CLOCK1",
  altsyncram_component.clock_enable_input_a = "BYPASS",
  altsyncram_component.clock_enable_input_b = "BYPASS",
  altsyncram_component.clock_enable_output_b = "BYPASS",
  `ifdef NO_PLI
    altsyncram_component.init_file = "d_mem.rif"
  `else
    altsyncram_component.init_file = "d_mem.hex"
  `endif,
  altsyncram_component.intended_device_family = "Cyclone III",
  altsyncram_component.lpm_type = "altsyncram",
  altsyncram_component.numwords_a = 256,
  altsyncram_component.numwords_b = 256,
  altsyncram_component.operation_mode = "DUAL_PORT",
  altsyncram_component.outdata_aclr_b = "NONE",
  altsyncram_component.outdata_reg_b = "UNREGISTERED",
  altsyncram_component.power_up_uninitialized = "FALSE",
  altsyncram_component.widthad_a = 8,
  altsyncram_component.widthad_b = 8,
  altsyncram_component.width_a = 32,
  altsyncram_component.width_b = 32,
  altsyncram_component.width_byteena_a = 1;
Instruction RAM:

`timescale 1 ns / 1 ps
// synopsys translate_on
module i_ram (  
data,  
rdaddress,  
rdclock,  
wraddress,  
wrclock,  
wren,  
q);
input [31:0]  data;  
input [7:0]  rdaddress;  
input  rdclock;  
input [7:0]  wraddress;  
input  wrclock;  
input  wren;  
output  [31:0]  q;
wire [31:0]  sub_wire0;  
wire [31:0]  q = sub_wire0[31:0];  
altsyncram altsyncram_component (  
  .wren_a (wren),  
  .clock0 (wrclock),  
  .clock1 (rdclock),  
  .address_a (wraddress),  
  .address_b (rdaddress),  
  .data_a (data),  
  .q_b (sub_wire0),  
  .aclr0 (1'b0),  
  .aclr1 (1'b0),  
  .addressstall_a (1'b0),  
  .addressstall_b (1'b0),  
  .byteena_a (1'b1),  
  .byteena_b (1'b1),  
  .clocken0 (1'b1),  
  .clocken1 (1'b1),  
  .clocken2 (1'b1),  
  .clocken3 (1'b1),  
  .data_b ((32{1'b1})),  
  .eccstatus (),  
  .q_a (),  
  .rden_a (1'b1),  
  .rden_b (1'b1),  
  .wren_b (1'b0));
defparam
  altsyncram_component.address_aclr_b = "NONE",
  altsyncram_component.address_reg_b = "CLOCK1",
  altsyncram_component.clock_enable_input_a = "BYPASS",
  altsyncram_component.clock_enable_input_b = "BYPASS",
  altsyncram_component.clock_enable_output_b = "BYPASS",
  `ifdef NO_PLI
    altsyncram_component.init_file = "i_mem.rif"
  `else
    altsyncram_component.init_file = "i_mem.hex"
  `endif,
  altsyncram_component.intended_device_family = "Cyclone III",
  altsyncram_component.lpm_type = "altsyncram",
  altsyncram_component.numwords_a = 256,
  altsyncram_component.numwords_b = 256,
  altsyncram_component.operation_mode = "DUAL_PORT",
  altsyncram_component.outdata_aclr_b = "NONE",
  altsyncram_component.outdata_reg_b = "UNREGISTERED",
  altsyncram_component.power_up_uninitialized = "FALSE",
  altsyncram_component.widthad_a = 8,
  altsyncram_component.widthad_b = 8,
  altsyncram_component.width_a = 32,
  altsyncram_component.width_b = 32,
  altsyncram_component.width_byteena_a = 1;
endmodule

WriteBack:
`timescale 1ns / 1ps
module Z(
  NOOP_IN,
  L0_Data_Wr,
  BR,
  ALU_OP,
  LD,
  ST,
  Freeze_b,
  L0Data,
  WB_DEST,
  Z.data_i,
  Z.destR,
  NOOP_OUT,
  WB_wren,
  BR_O,
  ALU_OP_O,
  LD_O,
ST_O,
  WB_Data_o,
  WB_DEST_o
);

input NOOP_IN;
input L0_Data_Wr;
input BR;
input ALU_OP;
input LD;
input ST;
input Freeze_b;
input [31:0] L0Data;
input [2:0] WB_DEST;
input [31:0] Z_data_i;
input [2:0] Z_destR;
output    NOOP_OUT;
output    WB_wren;
output    BR_O;
output    ALU_OP_O;
output    LD_O;
output    ST_O;
output    [31:0] WB_Data_o;
output    [2:0] WB_DEST_o;

wire SYNTHESIZED_WIRE_0;
wire SYNTHESIZED_WIRE_1;
wire SYNTHESIZED_WIRE_2;
wire SYNTHESIZED_WIRE_3;
wire SYNTHESIZED_WIRE_10;
wire SYNTHESIZED_WIRE_5;
wire SYNTHESIZED_WIRE_8;
wire SYNTHESIZED_WIRE_9;

soft  b2v_inst(
  .in(NOOP_IN),
  .out(NOOP_OUT));

soft  b2v_inst1(
  .in(SYNTHESIZED_WIRE_0),
  .out(WB_wren));

assign    SYNTHESIZED_WIRE_10 = L0_Data_Wr & LD;
assign    SYNTHESIZED_WIRE_2 = SYNTHESIZED_WIRE_1 & ALU_OP;
assign SYNTHESIZED_WIRE_9 = SYNTHESIZED_WIRE_2 & SYNTHESIZED_WIRE_3;

assign SYNTHESIZED_WIRE_8 = SYNTHESIZED_WIRE_10 & SYNTHESIZED_WIRE_5;

mux_3_2 b2v_inst2(
    .sel(SYNTHESIZED_WIRE_10),
    .data0x(Z_destR),
    .data1x(WB_DEST),
    .result(WB_DEST_o));

assign SYNTHESIZED_WIRE_1 = ~L0_Data_Wr;
assign SYNTHESIZED_WIRE_5 = ~Freeze_b;

data32x2_mux b2v_inst3(
    .sel(SYNTHESIZED_WIRE_10),
    .data0x(Z_data_i),
    .data1x(L0Data),
    .result(WB_Data_o));

soft b2v_inst4(
    .in(BR),
    .out(BR_O));

soft b2v_inst5(
    .in(ALU_OP),
    .out(ALU_OP_O));

soft b2v_inst6(
    .in(LD),
    .out(LD_O));

assign SYNTHESIZED_WIRE_0 = SYNTHESIZED_WIRE_8 | SYNTHESIZED_WIRE_9;

soft b2v_inst7(
    .in(ST),
    .out(ST_O));

assign SYNTHESIZED_WIRE_3 = Freeze_b ^ LD;
endmodule