A MICRO POWER HARDWARE FABRIC FOR EMBEDDED COMPUTING

by

Gayatri Mehta

B.Tech in Electronics and Communications, National Institute of Technology, India, 1999

M.Tech in Microelectronics, Panjab University, India, 2001

M.S in Telecommunications, University of Pittsburgh, 2003

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2009

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This dissertation was presented

by

Gayatri Mehta

It was defended on

July 20, 2009

and approved by

Alex K. Jones, Associate Professor, Department of Electrical and Computer Engineering

Jun Yang, Associate Professor, Department of Electrical and Computer Engineering

Allen Cheng, Assistant Professor, Department of Electrical and Computer Engineering

James T. Cain, Professor, Department of Electrical and Computer Engineering

Brady Hunsaker, Adjunct Professor, Department of Industrial Engineering, Google, Inc.

Dissertation Director: Alex K. Jones, Associate Professor, Department of Electrical and Computer Engineering

A MICRO POWER HARDWARE FABRIC FOR EMBEDDED COMPUTING

Gayatri Mehta, PhD

University of Pittsburgh, 2009

Field Programmable Gate Arrays (FPGAs) mitigate many of the problems encountered with the development of ASICs by offering flexibility, faster time-to-market, and amortized NRE costs, among other benefits. While FPGAs are increasingly being used for complex computational applications such as signal and image processing, networking, and cryptology, they are far from ideal for these tasks due to relatively high power consumption and silicon usage overheads compared to direct ASIC implementation. A reconfigurable device that exhibits ASIC-like power characteristics and FPGA-like costs and tool support is desirable to fill this void.

In this research, a parameterized, reconfigurable fabric model named as domain specific fabric (DSF) is developed that exhibits ASIC-like power characteristics for Digital Signal Processing (DSP) style applications. Using this model, the impact of varying different design parameters on power and performance has been studied. Different optimization techniques like local search and simulated annealing are used to determine the appropriate interconnect for a specific set of applications. A design space exploration tool has been developed to automate and generate a tailored architectural instance of the fabric.

The fabric has been synthesized on 160 nm cell-based ASIC fabrication process from OKI and 130 nm from IBM. A detailed power-performance analysis has been completed using signal and image processing benchmarks from the MediaBench benchmark suite and elsewhere with comparisons to other hardware and software implementations. The optimized fabric implemented using the 130 nm process yields energy within 3X of a direct ASIC implementation, 330X better than a Virtex-II Pro FPGA and 2016X better than an Intel XScale processor.

TABLE OF CONTENTS

1.0 INTRODUCTION 1

1.1 Related Work 5

1.2 Background 8

1.2.1 SuperCISC Architecture 8

1.2.2 Super Data Flow Graphs 9

2.0 STATEMENT OF THE PROBLEM 11

2.1 Domain Specific Fabric 12

2.2 Manual design space exploration 13

2.3 Mapping of benchmarks onto the domain specific fabric using Simulated Annealing 14

2.4 Power and performance analysis of benchmarks implemented on the fabric 14

2.5 Design space exploration tool for domain-specific fabric 17

3.0 DOMAIN SPECIFIC FABRIC 19

4.0 MANUAL DESIGN SPACE EXPLORATION 27

4.1 Power modeling and analysis 27

4.2 Design space case studies 29

4.2.1 Functional unit implementation tradeoffs 29

4.2.2 Impact of datawidth on power, performance and area of functional units . . . 30

4.2.3 Dedicated Pass-gates 33

4.2.4 Multiplexer cardinality impact on power 34

4.2.5 Benchmark Driven Interconnect Design 35

4.2.5.1 Interconnect profiling 36

5.0 MAPPING OF BENCHMARKS ONTO THE FABRIC USING SIMULATED ANNEALING . . . 45
5.1 Optimizing the interconnect . . . 49
6.0 POWER AND PERFORMANCE ANALYSIS OF THE BENCHMARKS IMPLEMENTED ON THE FABRIC . . . 52
6.1 Homogeneous and heterogeneous interconnects . . . 53
6.2 Dedicated vertical routes . . . 56
6.3 Heterogeneous ALUs . . . 61
6.4 Combining heterogeneity and dedicated vertical routes . . . 65
6.5 Energy results for the fabric designs implemented on 130 nm IBM ASIC standard cells . . . 65
7.0 DESIGN SPACE EXPLORATION TOOL FOR DOMAIN-SPECIFIC FABRIC . . . 68
8.0 CONCLUSIONS AND FUTURE WORK . . . 77
8.1 Conclusions . . . 77
8.2 Future work . . . 78
8.2.1 Implementing the fabric design to 90 nm and below target technologies . . . 79
8.2.2 Designing fabric architectures to map encryption algorithms . . . 80
8.2.3 Improvements in the design space exploration tool . . . 80
BIBLIOGRAPHY . . . 82

LIST OF TABLES

1 Parameters of the fabric model. For the design space exploration considered here, fixed or limited parameters are indicated in **bold**. 20
2 Area results for ALUs of different datawidths 31
3 Power comparison of ALU used as a pass-gate and a dedicated pass-gate. 35
4 Runtime and solution quality for the simulated annealing algorithm. 48
5 Rows added by various mapping strategies with different interconnects. A hyphen (-) indicates that no solution was found. 49
6 Runtime in (seconds) of various mapping strategies with different interconnects. A star (*) indicates that no solution was found. 50
7 Fabric size (Width x Height) for mapping various benchmarks onto different interconnects using the heuristic mapper. 54
8 Fabric size (Width x Height) for mapping various benchmarks onto different interconnects with different percentages of dedicated pass gates using the heuristic mapper. 59
9 Number of ALUs used as pass gates for various interconnect strategies. 60
10 Increase in overall path length by addition of dedicated pass gates. 60
11 Area results for various instances. 64

LIST OF FIGURES

1 Power consumption features of a Xilinx Virtex-2 3000 FPGA [Sheng et al. 2002]. . 1

2 The domain specific fabric (DSF) is comprised of Arithmetic and Logic units and a reconfigurable interconnect. 2

3 The multiplexer-based interconnection stripe structure. 3

4 Schematic for a 5:1 multiplexer equivalent using 4:1 multiplexers 4

5 SuperCISC architecture. 8

6 Software code and DFG showing control flow in ADPCM encoder 10

7 Example data flow graph (DFG) 15

8 Example mapping. 16

9 Design space exploration flow of our fabric model. 18

10 Domain Specific Fabric conceptual model. 19

11 VHDL code for a 4-wide homogeneous ALU stripe. 23

12 FIM file example for 5:1 style interconnect. 24

13 FIM file example for an 8 ops ALU, 8:1 interconnect with 50% dedicated pass gates. 25

14 VHDL code for a heterogeneous ALU stripe generated from the FIM file of Figure 13. 26

15 4-D plots of p, d, and s versus power for ADD and MULT operations of an ALU and 2:1 multiplexer synthesized with standard cells for a 160nm OKI ASIC process. Power is indicated as a color between black and white where solid white represents the least power consumed by the device and black indicates the most power consumed by the device. Measurements taken at 0.1 intervals in each dimension p, d, and s. 28

16 Power results for several functional unit implementation techniques. Results for a standard cell 160nm OKI ASIC process. 30
17 Power consumption of ALU for different datawidths. 31
18 Delay results of ALU for different datawidths. 32
19 Comparison of Energy consumption of 32-bit ALU, 16-bit ALU, 32-bit ALU used for 16-bit operations, 8-bit ALU and 32-bit ALU used for 8-bit operations. 33
20 Power profiles for various vertical routing structures. 34
21 Power consumption in multiplexers of different cardinalities. Results for a standard cell 160nm OKI ASIC process. 36
22 Design flow for creating SDFGs to run on the fabric. 37
23 Mapped SDFG of an ADPCM decoder onto a particular fabric instance. 38
24 Multiplexer cardinality usage. 39
25 Connectivity using 4:1 multiplexers. 39
26 Partial SDFG that occurs frequently in image and signal processing applications. . . 40
27 Attempt to embed the graph from Figure 26 into the connectivity supplied by Figure 25. 40
28 Ways to overlap node pairs with the 4:1 connectivity. 41
29 Embedding the graph from Figure 26 with the connectivity provided by Figure 4. . . 41
30 Embedding the graph from Figure 26 into a 3-5-5:1 interconnect. 42
31 Embedding the graph from Figure 26 into a 3-5-5-3:1 interconnect. 43
32 Embedding the dense subgraph from ADPCM encoder into a 5:1 interconnect with 25% of ALUs replaced with dedicated pass-gates. 43
33 Schematic for building a 6:1 reaching multiplexer interconnect using 4:1 multiplexers. 44
34 Embedding the graph from Figure 26 into a 6:1 interconnect with 25% dedicated pass-gates. 44
35 Mapping of a benchmark onto the fabric. 46
36 Multiplexer cardinality usage for a variety of signal and image processing applications using ASAP scheduling, Simulated Annealing, and IP. 51
37 Design flow using FIM. 52
38 Impact of interconnect cardinality on energy consumption. 55

39 Potential for energy reduction by reducing interconnect cardinality. 56
40 Comparison of ALUs used for routing and for computation. 57
41 Energy trend when adding dedicated pass gates for 5:1 interconnect. 58
42 Energy trend when adding dedicated pass gates for 6:1 interconnect. 58
43 Energy trend when adding dedicated pass gates for 8:1 interconnect. 59
44 Comparison of best energy result between 0% and 50% dedicated pass gates with heuristic mapper. 62
45 Potential best energy result using between 0% and 50% dedicated pass gates, mappings provided by the IP program where feasible. 63
46 Impact of reducing the number of operations of ALUs on energy. 64
47 Combining dedicated pass gates with heterogeneous functional units. 66
48 Energy comparison of fabric designs implemented on 130 nm IBM ASIC standard cells 67
49 Design space exploration flow for the domain specific fabric. 69
50 Results from the design space exploration tool for various architectures for threshold value of the average path length increase set to 2. 72
51 ALUs used as pass gates for various fabric architectures. 73
52 Results from the design space exploration tool for various architectures for threshold value of the average path length increase to be 3. 74
53 ALUs used as pass gates for various fabric architectures. 75
54 Energy comparison between the manual architectural solution and the architectural solutions generated by the tool. 76
55 Fabric compared with other hardware and software implementations. 79
56 Fabric design with memories using segmented bus architecture. 80

1.0 INTRODUCTION

Rapidly increasing mask and non-recurring engineering (NRE) costs, expensive Computer Aided Design (CAD) tools, long manufacturing times, and lack of flexibility of Application Specific Integrated Circuits (ASICs) necessitate new solutions to produce custom logic devices on time and on budget. Field Programmable Gate Arrays (FPGAs) address many of these problems by offering the benefits of fast time-to-market, flexibility, relatively low CAD tooling costs, and a per part cost that amortizes NRE costs over all users. While FPGAs are increasingly being used for complex computational applications such as multimedia, signal processing, networking, etc., they are far from ideal for these tasks due to high power consumption and silicon usage overheads compared to direct ASIC implementation.

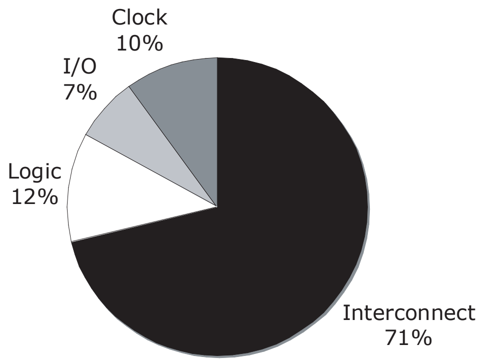


Figure 1: Power consumption features of a Xilinx Virtex-2 3000 FPGA [Sheng et al. 2002].

The dynamic power consumption in FPGAs has been shown to be dominated by interconnect power [1]. For example, as shown in Figure 1, the reconfigurable interconnect in the Virtex-2 FPGA consumes more than 70% of the total power dissipated in the device. Power consumption is exacerbated by the necessity of bit-level control for the computational and switch blocks.

A reconfigurable device that exhibits ASIC-like power qualities and FPGA-like costs and tool support is desirable to fill this void. Several coarse-grained fabric architectures proposed during the last decade have been focused on performance and area-efficient architectural techniques. Even though power is becoming one of the critical design concerns for semiconductor industry, this issue has not been adequately addressed in the existing coarse-grained fabric architectures.

In this dissertation research, an energy-efficient, parameterized, reconfigurable fabric model named as domain specific fabric (DSF) was designed for Digital Signal Processing (DSP) style applications. The DSF has a striped configuration like that of PipeRench [2, 3] but without register files. It is comprised of coarse-grained Arithmetic and Logic Units (ALUs) and multiplexer-based interconnect. As shown in Figure 2, ALUs are organized into rows or *computational stripes* within which each functional unit operates independently. The results of these ALU operations are then fed into *interconnection stripes* constructed using multiplexers. The multiplexer based interconnect is shown in Figure 3.

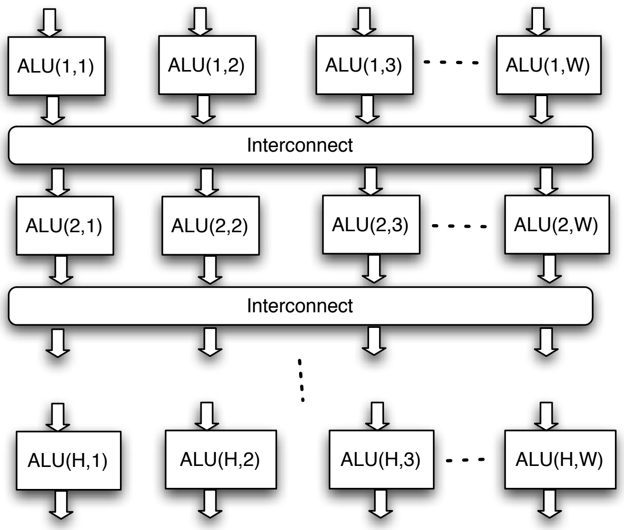


Figure 2: The domain specific fabric (DSF) is comprised of Arithmetic and Logic units and a reconfigurable interconnect.

Using this model, the impact of varying different architectural design parameters on power and performance was studied. The tradeoffs of potential energy savings from reducing the flexibility of

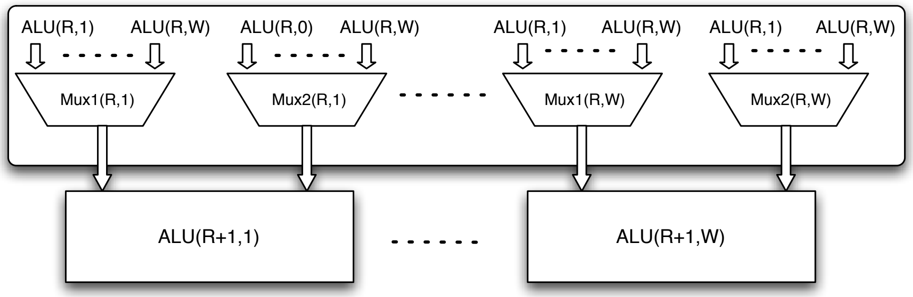


Figure 3: The multiplexer-based interconnection stripe structure.

the architecture compared to the ease of mapping the applications onto the device were examined. The impact of varying the cardinality (number of inputs/fanin of multiplexers) and the orientation (overlap of multiplexers for different operands) of multiplexers was considered. The interconnect strategies considered include baseline interconnect architectures built from fully connected rows, 8:1, and 4:1 multiplexers. A 5:1 multiplexing strategy shown in Figure 4, developed from mirroring 4:1 multiplexers connected to the functional unit input operands was also explored. In order to further simplify the device the possibility of using a heterogeneous interconnect to reduce the multiplexing cardinality at some locations was explored. 33% and 50% of the 5:1 interconnect were replaced with 3:1 multiplexers, built from mirrored 2:1 multiplexers. This is an attempt to reduce the delay and power required by the multiplexers and interconnect wires. These strategies are called 3-5-5:1 (33% 3:1), and 3-5-5-3:1 (50% 3:1). Different optimization techniques like local search and simulated annealing were used to determine the appropriate interconnect for a domain of applications. Simulated Annealing was also used to map benchmarks onto the fabric.

Then the impact of heterogeneity in the functional unit design was explored by introducing non-uniform arithmetic and logic units in a stripe. When mapping a data flow graph to a stripe-style structure, data dependency edges often traverse multiple rows. In these fabric structures, ALUs must often pass these values through without doing any computation. These operations in the graph are called pass gates. The benefit of adding dedicated pass-gates to prevent functional units from being used as routing was examined. The incorporation of dedicated pass-gates requires

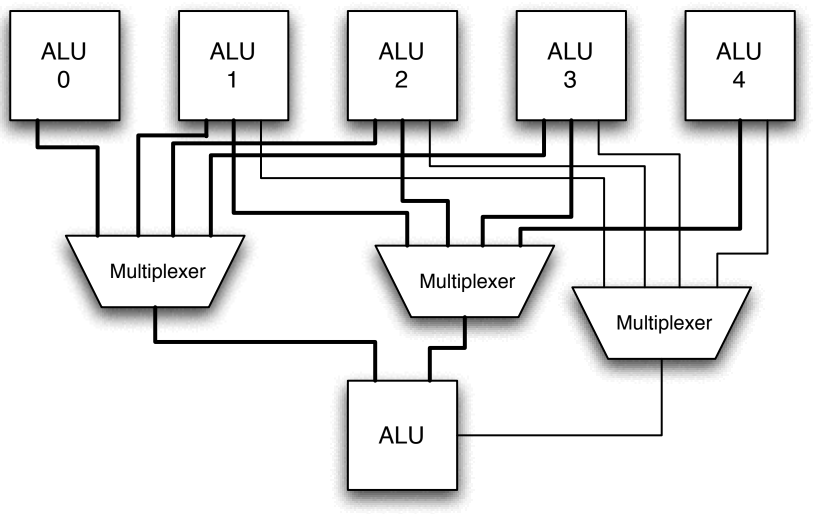


Figure 4: Schematic for a 5:1 multiplexer equivalent using 4:1 multiplexers

potential tradeoffs in the cardinality and orientation of the multiplexers in the interconnect. Another approach used in this research to reduce the complexity of the ALUs is to reduce the number of operations each ALU can support and spread the total number of required operations out over the stripe. While the ALUs within the stripe are heterogeneous, each stripe in the fabric remains identical.

Exploring the design space manually is time consuming and may not even be feasible for the most complex designs. A design space exploration tool was developed to automate the architectural exploration case studies. The tool generates a tailored architectural instance based on the needs of the applications to reduce power and area, and improve performance for a given suite of applications. Super data flow graphs of the applications were examined to get information like number of functional units, type of functional units, granularity of the functional units and interconnect, and fan-in and fan-out of different functional units. Several statistics like width and height of the fabric, utilization of the fabric in terms of number of computational elements and pass gates, multiplexer cardinality usage, granularity of the functional units and the interconnect were determined. The tool determines the best candidate by minimizing the multiplexer cardinality and the number of operations supported per ALU, and maximizing the number of dedicated pass gates in a fabric architecture. Power and performance analysis was done by implementing a set of core signal processing benchmarks from the MediaBench benchmark suite and some edge-detection benchmarks from the image processing domain onto the fabric. These benchmarks are of particular interest due to applications such as speech processing and digital communications, and platforms such as cell phones and cameras.

1.1 RELATED WORK

Recently, the development and use of coarse-grained fabrics for computationally complex tasks has received a lot of attention as a possible alternative to FPGAs. Coarse-grained fabrics consist of multi-bit logic units and multi-bit datapaths that significantly reduces the routing overhead. Many architectures have been proposed and developed both in academia and industry during the last decade such as MATRIX [4], Garp [5], Chimaera [6], MorphoSys [7], RaPiD [8, 9], PipeRench [2, 3], CFPA [10], HFPGA [11], RAP [12], XPP [13], and the FPOA [14]. Some of these architectures are mentioned below:

MATRIX (Multiple ALU architecture with Reconfigurable Interconnect eXperiment) [4] is comprised of a two-dimensional array of identical 8-bit functional units with a configurable network. Each functional unit consists of a 256x8-bit memory, an 8-bit ALU and a control logic. The Garp [5], the Chimaera [6], the MorphoSys [7], and the SuperCISC [15] architectures combine a reconfigurable computing device with a processor in order to do hardware acceleration. RaPiD (Reconfigurable Pipelined Datapath) [8, 9], mainly intended for computation-intensive applications, consists of a linear array of application-specific function units. PipeRench [2, 3] has a striped configuration and is comprised of an interconnected network of configurable logic blocks and storage elements. It consists of a set of physical pipeline stages called stripes and each stripe contains a set of processing elements, register files and an interconnection network.

The CFPA (Computational Field Programmable Architecture) [10] consists of Partial Add, Subtract, and Multiply (PASM) blocks for implementing data path operations of computational intensive applications. The PASM block operates on 4-bit operands and can be connected together to implement adders, subtracters, and multipliers of various sizes. The HFPGA (Hierarchical Field

Programmable Gate Array) [11] allows the creation of coarse grain blocks built from traditional 4-input lookup tables. These coarse grain blocks have dedicated routing channels.

The Reconfigurable Algorithm Processor (RAP) from Elixent [12] is comprised of an array of 4-bit ALUs and register/buffer blocks that can be cascaded to suit different data widths. The ALUs are arranged in a chessboard-style array, alternating with adjacent switchboxes.

Pact XPP Technologies [13] proposed the XPP architecture which has a hierarchical array of coarse-grained adaptive computing elements called Processing Array Elements (PAEs) and a packet-oriented communication network. An XPP core is comprised of a rectangular array of ALU-PAEs and RAM-PAEs with I/O.

MathStar [14] proposed Field Programmable Object Array (FPOA) which consists of a 2-D array of Silicon Objects (SOs). Silicon Objects are 16-bit configurable machines such as ALU, Multiply-Accumulate Unit or Register File. Both Silicon Object behavior and the interconnection among Silicon Objects are field-programmable. Rapport Incorporated has created Kilocore based on the previously mentioned PipeRench [16].

Unlike MATRIX whose basic functional unit consists of an 8-bit ALU and a SRAM, the basic functional unit in the DSF is a coarse-grained ALU having variable datawidth. Our approach differs from GARP and Chimaera in so much as we tailor the hardware co-processor to the application domain. Compared to RaPiD which has small RAMs and registers to store data and intermediate results, the DSF is purely combinational. The programmable connections in the datapath interconnect in the DSF are modeled as multiplexers somewhat similar to those in RaPiD. Unlike RAP whose ALUs are arranged in a chessboard style, the DSF has a striped configuration like that of PipeRench but without register files. Compared to the XPP architecture which is comprised of a mixture of ALU-PAEs and RAM-PAEs, the DSF consists of only an array of ALUs with no memory elements.

Application Specific Instruction Set Processors (ASIPs) attempt to extend normal processor cores with custom instructions, often in reconfigurable hardware, designed to tailor the processor to a particular domain of applications or even a single application for improved performance. One of the earliest works on this topic is the dynamic instruction set computer (DISC) developed at Brigham Young University [17]. DISC uses partial reconfiguration available in FPGAs to swap in and out custom instructions based on demand of the application. Cong et al. from University of California, Los Angeles described a technique to generate the instructions included in an ASIP using pattern matching of input applications [18]. In addition to these notable examples, there has been a lot of work in the development of ASIPs. Many are described in this survey paper [19]. However, ASIPs have transitioned into the commercial realm with the most notable example being the Xtensa processor from Tensillica [20].

Shen et al. describe a video specific instruction set architecture including both single instruction multiple data (SIMD) and custom video specific instructions [21]. Fanucci et al. describe a processor architecture for non-linear image processing algorithms [22]. In terms of the design flows, Brisk et al. describes an optimal polynomial time solution to determining how many registers to include in an ASIP [23]. Finally, Dinh et al. describes a method to use resource sharing in the custom instructions of ASIPs to reduce area and improve performance [24]. In this research, a reconfigurable architecture, was designed which can be customized for the applications it executes in a similar manner as ASIPs. However, we customize the reconfigurable fabric, rather than utilizing a generic reconfigurable fabric to customize a processor.

The generic and parameterized fabric model presented in this dissertation is used to explore the architectural design space of the reconfigurable architectures. Several methods have been proposed in the past few years for design space exploration of reconfigurable architectures [25, 26, 27]. However, these methods are either too technology-dependent or too architecture-dependent. They deal with low level of abstraction and are too specific as only a small and limited design space can be explored around the target architecture. In order to overcome this limitation, Bossuet et al [28] proposed a design space exploration method which can be used to cover a wide domain of reconfigurable fabrics, from fine-grained to coarse-grained fabrics, as well as heterogeneous fabrics. They used the architectural processing use rate and the communication hierarchical distribution as metrics to investigate a power-efficient architecture. In contrast, this research work is focused on the study of the application needs that drive the construction of the fabric architecture. The impact of varying different architectural design parameters on power and performance of the fabric has been studied.

1.2 BACKGROUND

1.2.1 SuperCISC Architecture

The DSF was designed to operate within the super-complex instruction-set computing (SuperCISC) embedded processor architecture summarized in Figure 5. SuperCISC is a heterogeneous, multi-core processor architecture designed to exceed performance of traditional embedded processors while maintaining a reduced power budget compared to low-power embedded processors.

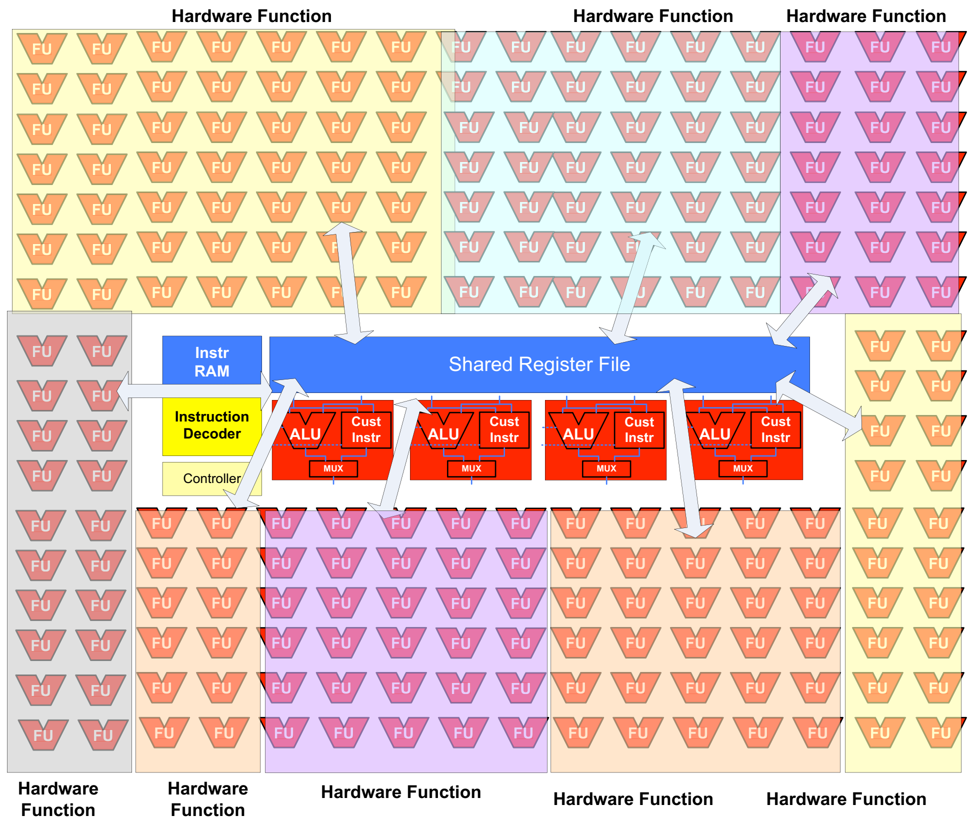


Figure 5: SuperCISC architecture.

The SuperCISC processor was developed with a 4-way very long instruction word (VLIW) core with a shared register file. This shared register file is also shared with one or more application specific hardware functions [15, 29]. The idea is to accelerate the high incidence code segments (e.g. loops) that require large portions of the application runtime, called kernels, while also accelerating the remaining non-kernel code with the VLIW. These kernels are converted into entirely combinational hardware functions generated automatically from the C using a design automation flow [30].

1.2.2 Super Data Flow Graphs

Like many synthesis flows, the portion of code to be implemented in hardware is converted into a Control Data Flow Graph (CDFG) representation. CDFG representation consists of a set of blocks interconnected by control flow edges. The details of the creation of a CDFG representation from a high level language can be found in [31]. These control flow edges are created by control statements within the code such as loop boundaries or conditional statements. Using hardware predication, these control dependencies can be converted into data dependencies and a Super Data Flow Graph (SDFG) can be made. For example, a conditional statement, such as an *if-then-else* C code segment, is implemented as a multiplexer acting as a binary switch to predicated output datapaths. In software, an *if-then-else statement* is implemented as a stream of six instructions composed of comparisons and branch statements. Software code and a data flow diagram for a 2:1 multiplexer equivalent are shown in Figure 6.

SDFGs retain a data flow structure allowing computational results to be computed in one ALU and flow onto others in the system. The DSF was designed to mimic this computational style. Figure 2 illustrates this top-down data flow concept. The functional units that surround the processor are the ASIC implementation of the kernels. The energy-efficient DSF designed in this research can be used to replace the ASIC functional units.

```
If (bufferstep) {
  delta = inputbuffer & 0xf;
} else {
  inputbuffer = *inp++
  delta = (inputbuffer >> 4) & 0xf;
}
```

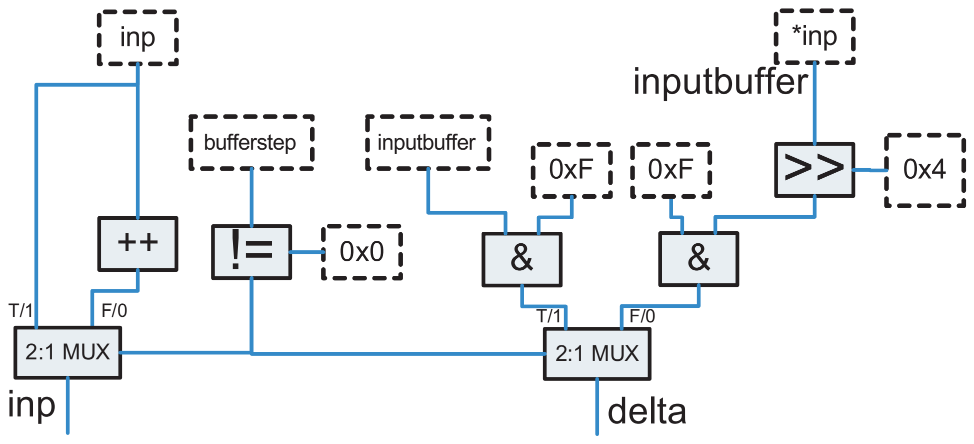


Figure 6: Software code and DFG showing control flow in ADPCM encoder

2.0 STATEMENT OF THE PROBLEM

FPGAs are the most commonly used general purpose reconfigurable hardware devices. They contain bit-level logic and independent bit-level routes. They are increasingly being used as ASIC replacements for computational intensive applications such as signal processing, image processing, security, networking, etc. This trend has been encouraged by an increase of heterogeneity in the FPGA devices with multi-bit ASIC functional units supplementing the bit-level logic. Unfortunately, FPGAs still have relatively poor power consumption characteristics compared to ASICs. The dynamic power consumption in FPGAs has been shown to be dominated by interconnect power [1]. The static power consumption of FPGAs is exacerbated by the necessity of bit-level control of the computational and switch blocks. Because the device is designed to handle sequential logic, clock trees and storage registers are required, which also contribute to power consumption. Much of this flexibility is not even needed for many classes of applications such as signal processing, image processing, security, etc. It is also difficult to program these devices because they use hardware description languages and have complex CAD tool flow. Thus, to create a low-power computational device to fit the needs of a particular class of applications, it is desirable to remove or reduce as many of these power consuming characteristics as possible. Additionally, the device should mimic the computational style of the data flow graphs (DFGs) generated from the compiler/synthesis engines so that applications can be easily mapped. This dissertation research provides the following contributions to solve this problem.

The goals of this dissertation research are: (1) to design an energy-efficient domain specific fabric by finding the best tradeoff between a reduced complexity device and the ability of design tools to map the applications onto the device, and (2) to develop a design space exploration tool to explore architectural tradeoffs efficiently and reach solutions quickly. In this research, a number of design space case studies were developed. As an initial application domain for our case studies, some of the core signal processing benchmarks from the MediaBench benchmark suite and some edge-detection benchmarks from the image processing domain were selected. In order to explore the architectural space, a parameterized, coarse-grained reconfigurable fabric model was designed. Using this model, the impact of varying different design parameters was studied for their implications on power and performance. Different optimization techniques like local search and simulated annealing were used to determine the appropriate interconnect for a domain of applications. A design space exploration tool was developed to automate the architectural exploration case studies. The tool generates a tailored architectural instance based on the needs of the applications to reduce power and area, and improve performance for a given suite of applications. The main contributions of this dissertation are described as follows:

2.1 DOMAIN SPECIFIC FABRIC

Stripe-based hardware fabrics are designed to mimic the computational style of data flow graphs (DFGs). The domain specific fabric (DSF) presented in this research works in a similar way, retaining a data flow structure allowing computational results to be computed in one ALU and flow onto others in the system. Each ALU in the fabric can be represented by a number of parameters such as the number of operands, O, datawidth of each operand, DW, and the number of operations, OP. The multiplexer cardinality, C, determines the width of each multiplexer and as a result connectivity of the interconnection stripe. The fabric size is determined by the parameters specifying the width, W, and the height, H, of the fabric. The width of the fabric determines the number of ALUs in each computational stripe and the height determines the number of computational and interconnection stripes in the fabric. The fabric model was implemented in parameterized VHDL using the `generic` capability of the VHDL. As the fabric model is generic and parameterized, various architectural instances can be generated and evaluated for power/performance. Redesigning the fabric architecture by hand for each new configuration was impractical. To solve this problem, I and other team members of our research group created the fabric interconnect model (FIM). The FIM was designed as a textual representation to describe the interconnect and the layout and make-up of the ALUs in the system. I developed the fabric generator to integrate the FIM in the design flow to automate the generation of fabric architectures. The generator takes the FIM file as an input and extracts information like the number and the type of operations supported by each ALU, the type of the interconnect, the number of dedicated pass gates, and generates the fabric architectural instance with the features defined in the FIM file.

2.2 MANUAL DESIGN SPACE EXPLORATION

To study the impact of various parameters of the fabric, the ALU and multiplexer elements were studied individually. The cardinality of the multiplexers was varied from 2 to 32 in powers of 2 and profiled for power consumption. Different architectural techniques for implementation of the ALU were also explored. ALUs of different data widths like 8, 16, 32 were synthesized and profiled for power and latency for several ALU operations.

The interconnect strategies considered in this research include baseline interconnect architectures built from fully connected rows, 8:1, and 4:1 multiplexers. A 5:1 multiplexing strategy developed from mirroring 4:1 multiplexers connected to the functional unit input operands was also explored. In order to further simplify the device the possibility of using a heterogeneous interconnect to reduce the multiplexing cardinality at some locations was explored. 33% and 50% of the 5:1 interconnect were replaced with 3:1 multiplexers, built from mirrored 2:1 multiplexers. This is an attempt to reduce the delay and power required by the multiplexers and interconnect wires. These strategies are called 3-5-5:1 (33% 3:1), and 3-5-5-3:1 (50% 3:1). An ALU used as a pass gate and a dedicated pass gate were power profiled. This is done to calculate the energy consumption overhead of using an ALU for pass operations. The details of the design space exploration studies are presented in Chapter 4 and [32, 33, 34].

2.3 MAPPING OF BENCHMARKS ONTO THE DOMAIN SPECIFIC FABRIC USING SIMULATED ANNEALING

A mapping of a data flow graph onto a fabric consists of an assignment of operators in the data flow graph to ALUs of the fabric such that the logical structure of the data flow graph is preserved and the parameters of the fabric are respected. This mapping problem is central to the use of the fabric, as a solution must be available in order for the fabric to be reprogrammed for a specific data flow graph. Because of the layered nature of the fabric, the mapping is also allowed to use ALUs as "pass gates," which take a single input and pass the input value to one or more outputs. In general, not all of the available ALUs and edges will be used. An example DFG and a corresponding mapping are shown in Figure 7 and Figure 8 respectively.

One of the more complicated parts of creating a mapping is the introduction of pass gates to fit the layered structure of the fabric. A successful approach that has been used works in two stages. In the first stage, pass gates are introduced heuristically and operators assigned to rows so that all edges go from one row to the next. The second stage assigns the operators to columns so that the fabric interconnect is respected. This second stage is called Feasible Mapping with Fixed Rows [35]. Depending on the interconnect design, there may or may not exist a feasible mapping. In this dissertation research, we have formulated the Simulated Annealing algorithm to solve the problem of Feasible Mapping with Fixed Rows. Simulated Annealing is a popular algorithm for computer aided design flows targeting custom hardware (either for standard cell ASICs or FPGAs) particularly for placement of cells. It was also used to study and optimize the interconnect for a domain of applications.

2.4 POWER AND PERFORMANCE ANALYSIS OF BENCHMARKS IMPLEMENTED ON THE FABRIC

In order to evaluate power and performance, a set of core signal processing benchmarks were selected from the MediaBench benchmark suite including the ADPCM encoder, ADPCM decoder, GSM channel encoder, and the MPEG II decoder. The Sobel and Laplace edge detection algo-

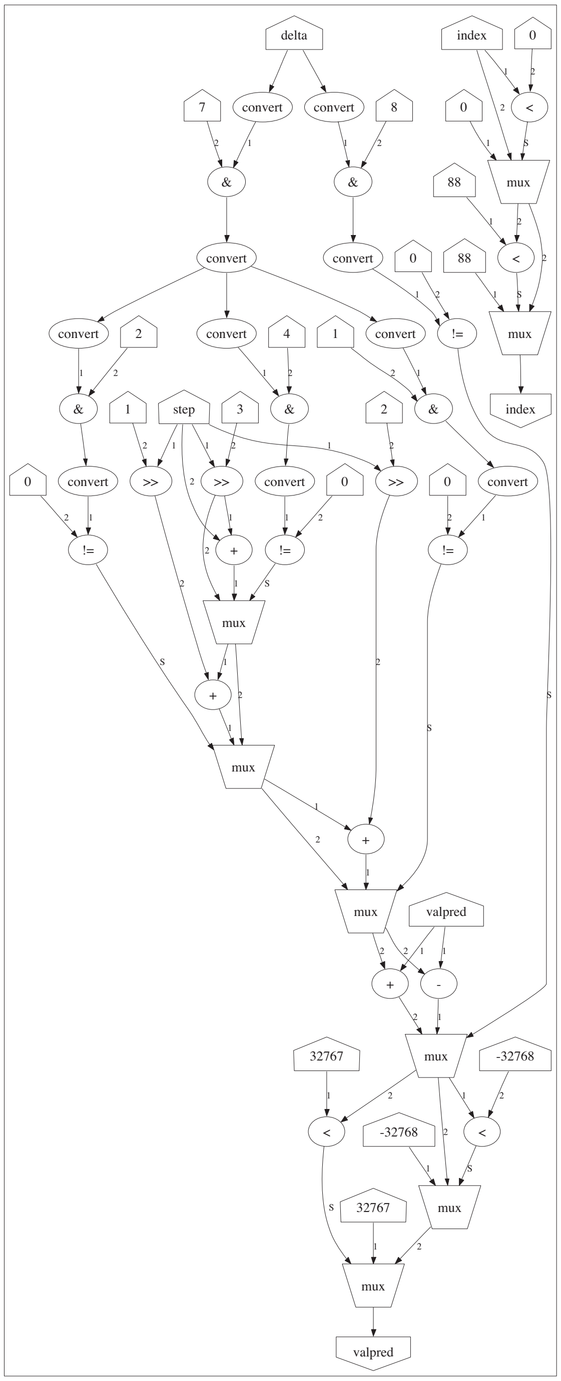


Figure 7: Example data flow graph (DFG)

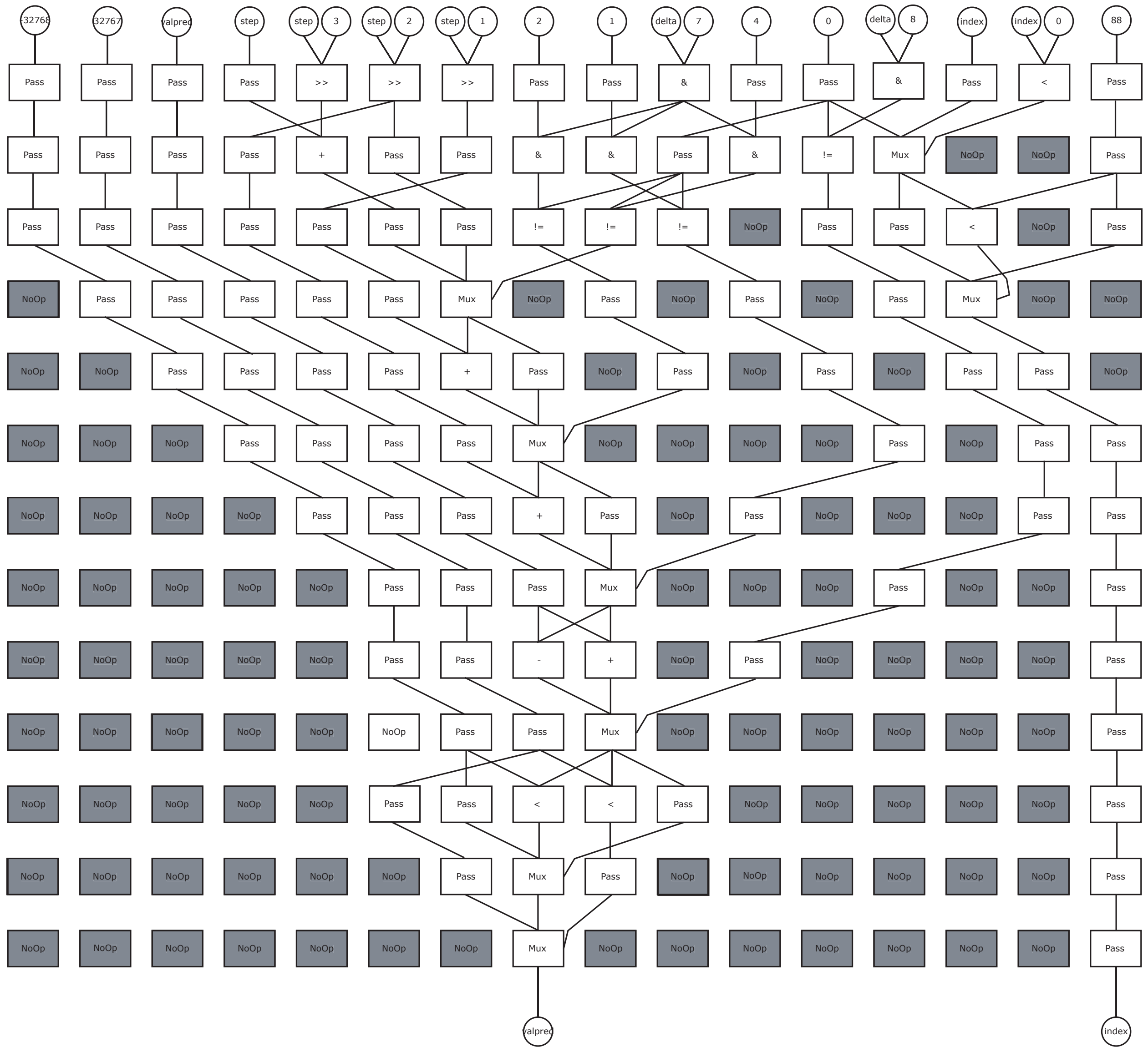


Figure 8: Example mapping.

rithms were also analyzed for power and performance. Using the SuperCISC compilation flow [30], computational kernels were extracted for these applications and converted into super data flow graphs (SDFGs) as described in Chapter 1. These SDFGs were then mapped to the fabric model. Several fabric architectures were designed and evaluated for power and performance for a set of signal and image processing applications. Architectures with homogeneous (fully connected, 8:1, 4:1, 5:1, 6:1) and heterogeneous interconnects (3-5-5-3:1, and 3-5-5:1), dedicated pass-gates (25%, 33%, and 50% of functional units replaced with dedicated pass-gates) were analyzed for power and performance. The impact of reducing the number of operations supported by the ALUs on energy was studied. The number of ALU operations were reduced from 23 to 16, 10 and 8. The combined effect of having reduced instruction set ALU and dedicated pass gates on energy was also examined.

2.5 DESIGN SPACE EXPLORATION TOOL FOR DOMAIN-SPECIFIC FABRIC

In this dissertation research, a design space exploration tool for domain specific reconfigurable computing was developed to automate the design space case studies that were done manually. The design space exploration flow for our fabric model is shown in Figure 9. It generates a tailored architectural instance based on the needs of the applications to reduce power and area, and improve performance for a given suite of applications. Super data flow graphs of the applications were examined to get information like number of functional units, type of functional units, granularity of the functional units and interconnect, and fan-in and fan-out of the different functional units. Several statistics like width and height of the fabric, utilization of the fabric in terms of the number of computational elements and pass gates, multiplexer cardinality usage, granularity of the functional units and the interconnect were determined. The tool determines an energy-efficient fabric for a given domain of applications by minimizing the multiplexer cardinality, minimizing the number of operations supported per ALU, and maximizing the number of dedicated pass gates.

The remainder of this dissertation is organized as follows: The fabric architecture and the configurable parameters are described in Chapter 3. Chapter 4 describes how the fabric model is used for architectural space exploration. Chapter 5 covers the study and optimization of the interconnect for a specific domain of applications. Power and performance analysis of the benchmarks is described in Chapter 6. Chapter 7 describes the automated design space exploration flow for the reconfigurable fabric. Conclusions and future work are discussed in Chapter 8.

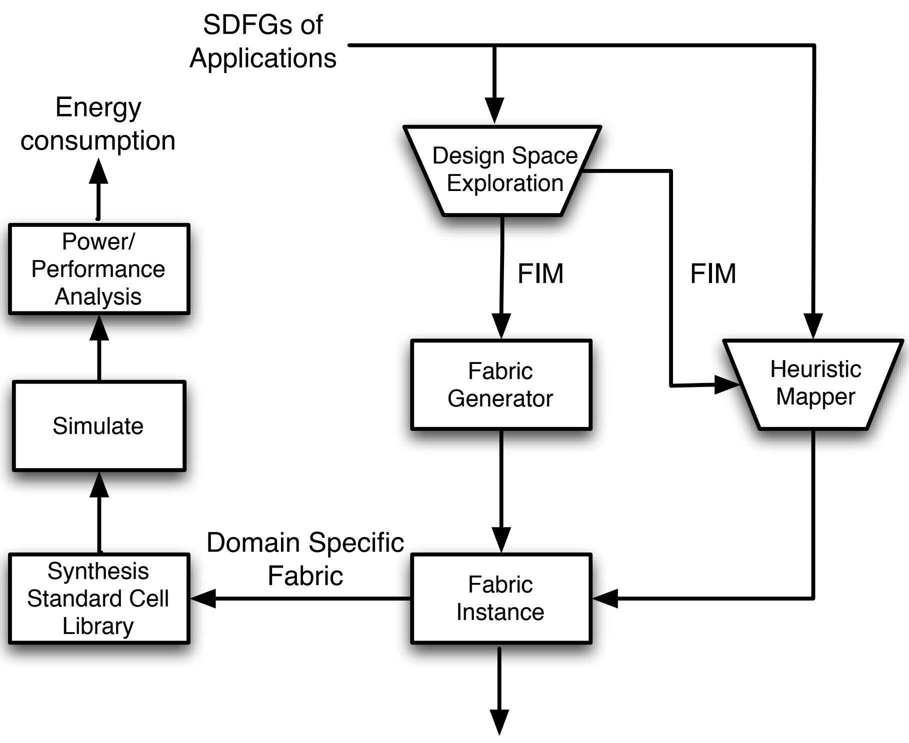


Figure 9: Design space exploration flow of our fabric model.

3.0 DOMAIN SPECIFIC FABRIC

Stripe-based hardware fabrics are designed to easily map data flow graphs (DFGs) from the application onto the device. The domain specific fabric (DSF) designed in this research works in a similar way, retaining a data flow structure allowing computational results to be computed in one ALU and flow onto others in the system. ALUs are organized into rows or *computational stripes* within which each functional unit operates independently. The results of these ALU operations are then fed into *interconnection stripes* constructed using multiplexers. Since the model does not include registers or other internal storage, it does not permit feedback between stripes. Figure 10 illustrates this top-down data flow concept.

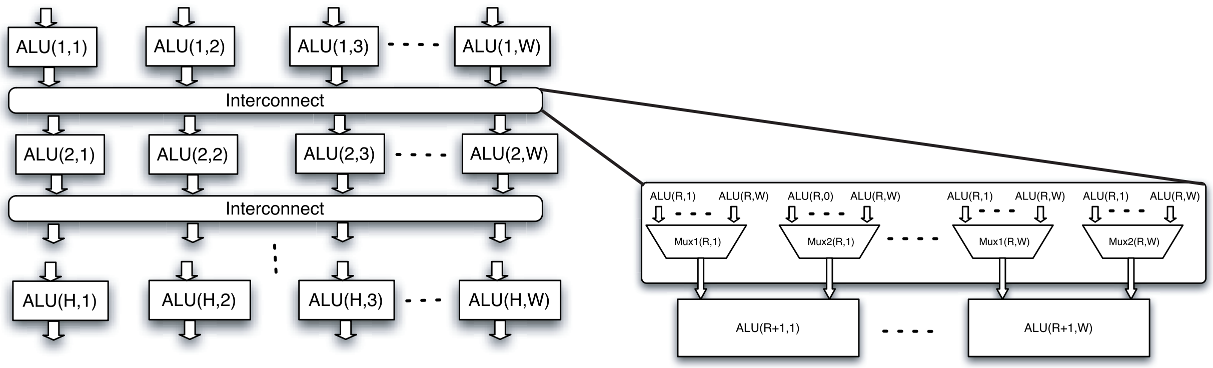


Figure 10: Domain Specific Fabric conceptual model.

.

The fabric model was implemented in parameterized VHDL using the `generic` capability of the VHDL language. Several parameters were included in the fabric model and are listed in Table

Table 1: Parameters of the fabric model. For the design space exploration considered here, fixed or limited parameters are indicated in **bold**.

Global Parameters	
ALU Datawidth	P= $\{\mathbf{8, 16, 32}\}$
Fabric Parameters	
Width of the fabric	W
Height of the fabric	H
Arithmetic and Logic Unit Parameters	
Number of operands	O= **3**
Number of operations	$ALU - OP$= $\{\mathbf{8, 10, 16, 20}\}$
Interconnect Parameters	
Multiplexer cardinality	C= $\{\mathbf{2, 3^*, 4, 5^*, 6^*, 8, 16, 32}\}$

1. For example, each ALU in the fabric can be represented by a number of parameters such as the number of operands, O, datawidth of each operand, DW, the number of operations, OP. The multiplexer cardinality, C, determines the width of each multiplexer and as a result connectivity of the interconnection stripe.

The fabric size is determined with the parameters specifying the width of the fabric, W, height of the fabric, H, and datawidth, DW. W dictates the number of ALUs in each computational stripe. The number of multiplexers in each interconnection stripe is a function of both W and O as the input to each ALU operand is configurable. H determines the number of computational and interconnection stripes in the fabric model. Thus, with the parameterizable model, a fabric contains WxH DW-wide ALUs segregated into H computational stripes. These computational stripes are interconnected by $H - 1$ stripes each containing OxW DW-wide $C : 1$ multiplexers.

The ALU which is the computational unit of the fabric not only performs arithmetic and logic operations but it is also used as a pass gate and a multiplexer in the fabric grid. This multiplexer requires a third, single bit operand to be included in the ALU. This bit specifies which of the two input operands to propagate to the output, acting as a selector. Thus, the interconnection stripe contains a third set of single bit $C : 1$ multiplexers for controlling this operand. This multiplex operation is added to the ALU as a separate operator and is different from the interconnect multiplexers.

Figure 11 shows an example of a vhdl code for a 4-wide homogeneous ALU stripe. All ALUs in a stripe are identical and all ALU stripes are identical. The fabric parameters are set using `generics` in the vhdl code. For the initial design space case studies, the number of operations supported by an ALU was fixed to 23. In the example code shown in Figure 11, the width of the fabric W is 4 , the data width P is 32, and the opcode OP of an ALU is 5-bits long. These parameters generate an ALU stripe with 4 ALUs in a stripe, where the `op_sel_bus` signal in the stripe is 20-bits long. The appropriate offsets are generated for each ALU signal using `generate` statements.

As the fabric model is generic and parameterized, various fabric architectural instances can be generated and evaluated for power/performance for a domain of applications. Redesigning the mapping flow and target fabric hardware by hand for each new configuration was impractical. To solve this problem, I and other team members of our research group created the fabric interconnect model (FIM). The FIM was designed as a textual representation to describe the interconnect and the layout and make-up of the ALUs in the system. The FIM becomes an input file to the mapper as well as the tool that generates a particular instance of the fabric with the appropriate interconnect.

The FIM file is written in the Extensible Markup Language (XML) [36]. XML was selected as it allowed the FIM specification to easily evolve as new features and descriptions were required. For example, while the FIM was initially envisioned to describe the interconnect only, it has evolved to describe dedicated pass-gates and other heterogeneous ALU structures.

Figure 12 shows an example partial FIM file that describes a 5:1-based interconnect. The pattern repeats the interconnect pattern for `alu0`, whose zeroth operand can read from two units to the left and one unit to the right, and the first operand is the mirror. The second operand is the selection bit if the ALU is configured as a multiplexer and follows the first operand. The ranges in the FIM can be discontinuous by supplying additional range flags. The file can contain a heterogeneous interconnect by defining additional `FTUs` with different interconnect ranges. The pattern can either repeat or can be arbitrarily customized without a repeating pattern for a fixed size fabric. In the FIM file, the types of operations supported by each ALU can also be specified.

Figure 13 shows an example partial FIM file that describes a fabric having an 8 operations ALU, 8:1 interconnect and 50% dedicated pass gates.

The FIM file is used to automatically generate the VHDL for the fabric instance described by the FIM. The FIM file shown in Figure 13 dictates the fabric generator to generate a heterogeneous ALU stripe with ALUs and dedicated pass gates. As each ALU supports 8 operations, the fabric generator determines that only a 3-bit opcode is needed for an ALU and a 1-bit opcode for a dedicated pass gate as shown in the generics of Figure 14. The fabric generator also generates a vhdl code for an ALU where ALU supports only the operations defined in the FIM file. Each dedicated pass gate can act as a pass gate or a *noop*. The `ALU`, and `pass gate` pattern repeats in the vhdl code as defined in the FIM file. The generated vhdl code has two ALUs (A) and two pass gates (P) in the $APAP$ pattern. The fabric instance VHDL is then synthesized using commercial tools such as Synopsys Design Compiler to generate a netlist tied to ASIC standard cells.

```
entity stripe is
generic (W:integer:=4; P:integer:=32; OP:integer:=5);
port (
inp1,inp2 : in std_logic_vector (W*P-1 downto 0);
op_sel_bus : in std_logic_vector (W*OP-1 downto 0);
.......
dout_bus : out std_logic_vector (W*P-1 downto 0)
);
end stripe;
architecture struct of stripe is
component ALU_0
port (
op_sel : in std_logic_vector (OP-1 downto 0);
s1,s2 : in std_logic_vector (P-1 downto 0);
......
dout : out std_logic_vector (P-1 downto 0)
);
end component;
:
:
begin
I1 : for N in W-1 downto 0 generate
I2 : if N = W-1 generate
I3 : ALU_0
port map(
s1 => inp1(P-1 + N*P downto N*P),
s2 => inp2(P-1 + N*P downto N*P),
op_sel => op_sel_bus(19 downto 15),
.....
dout => dout_bus(P-1 + N*P downto N*P)
);
end generate;
:
:
end generate;
end struct;
```

Figure 11: VHDL code for a 4-wide homogeneous ALU stripe.

```
<rowpattern repeat="forever">
 <row>
  <ftupattern repeat="forever">
   <FTU type="alu0">
    <operand number="0">
     <range left ="-2" right ="1"/>
    </operand>
    <operand number="1">
     <range left ="-1" right ="2"/>
    </operand>
    <operand number="2">
     <range left ="-1" right ="2"/>
    </operand>
   </FTU>
  </ftupattern>
 </row>
</rowpattern>
```

Figure 12: FIM file example for 5:1 style interconnect.

```
<ftudefine name="alu[0]" useic="false" noop="000" type="alu">
<op code="001" commutative="true"> + </op>
<op code="010" commutative="false"> - </op>
<op code="011" commutative="true"> * </op>
<op code="100" commutative="false"> &gt;&gt; </op>
<op code="101" commutative="false"> mux </op>
<op code="110" commutative="false"> pass </op>
<op code="111" order="reverse" commutative="false"> pass </op>
<op code="000" commutative="true"> noop </op>
 </ftudefine>
<ftudefine name="pass" useic="false" noop="0" type="pass">
<op code="0" commutative="true"> noop </op>
<op code="1" commutative="false"> pass </op>
 </ftudefine>
<rowpattern repeat="forever">
<row>
<ftupattern repeat="forever">
<FTU type="alu[0]">
<operand number="0">
<range left ="-3" right="4"/>
 </operand>
<operand number="1">
<range left ="-3" right="4"/>
</operand>
<operand number="2">
<range left ="-3" right="4"/>
</operand>
</FTU>
<FTU type="pass">
<operand number="0">
<range left ="-3" right="4"/>
 </operand>
 </FTU>
 </ftupattern>
</row>
</rowpattern>
```

Figure 13: FIM file example for an 8 ops ALU, 8:1 interconnect with 50% dedicated pass gates.

```
entity stripe is
generic(W:integer:=4;P:integer:=32;OP_ALU:integer:=3;OP_DP:integer:=1);
port (
inp1,inp2 : in std_logic_vector (W*P-1 downto 0);
op_sel_bus: in std_logic_vector((W*OP_ALU)+(W*OP_DP)-1 downto 0);
.....
dout_bus : out std_logic_vector (W*P-1 downto 0)
);
end stripe;
architecture struct of stripe is
component ALU_0
port (
op_sel : in std_logic_vector (OP-1 downto 0);
......
dout : out std_logic_vector (P-1 downto 0)
);
end component;
component pass_gate
port (
op_sel : in std_logic;
.......
dout : out std_logic_vector (P-1 downto 0)
);
end component;
begin
I1 : for N in W-1 downto 0 generate
I2 : if N = W-1 generate
I3 : ALU_0
port map(
s1 => inp1(P-1 + N*P downto N*P),
op_sel => op_sel_bus(7 downto 5),
......
dout => dout_bus(P-1 + N*P downto N*P)
);
end generate;
I4 : if N = W-2 generate
I5 : pass_gate
port map(
op_sel => op_sel_bus(4),
.......
dout => dout_bus(P-1 + N*P downto N*P)
);
end generate;
:
:
end generate;
end struct;
```

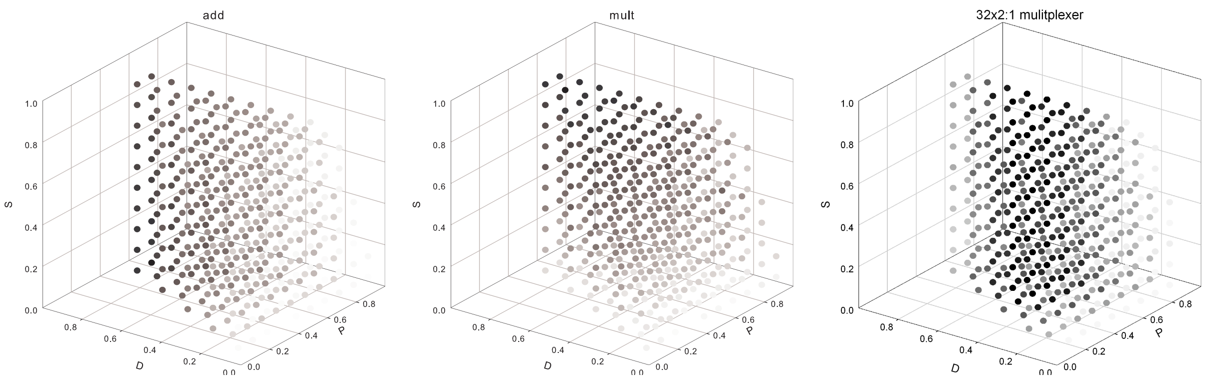
Figure 14: VHDL code for a heterogeneous ALU stripe generated from the FIM file of Figure 13.

4.0 MANUAL DESIGN SPACE EXPLORATION

In order to evaluate different design parameters of the fabric, the power consumption and performance of different parameterized fabric components were studied. Because power consumption is of particular interest the following section provides some details on how the power analysis was done.

4.1 POWER MODELING AND ANALYSIS

Power has become one of the critical design concerns for semiconductor industry. A significant amount of effort has been devoted to study low power hardware design techniques [37, 38, 39, 40, 41, 42, 43]. Power macromodeling is one of the most commonly used approaches to estimate power dissipation in digital circuits. Power macromodeling when done at the Register Transfer Level (RTL) generates a mapping between the power dissipation of the circuit and certain statistics of the input signals applied to the circuit. Liu et al. [44] suggested that the statistics of the input signals, like average input signal probability p, average transition density d and spatial correlation s can be used for accurate estimation of the power dissipated in digital circuits. Using a technique similar to [45], various functional units and interconnect multiplexers synthesized with standard cells for a 160nm OKI ASIC process over all values of p, d and s were power profiled. This provides a more accurate power analysis than considering switching alone. The actual input vectors for power simulation were generated using the technique described in [44]. All power measurements taken in this work are weighted averages across all valid p, d, and s values with 0.1 probability increments ranging from 0.5 to 0.95. The synthesis was completed using Design Compiler and the power was estimated using PrimePower; both of these tools are from Synopsys [46].



(a) Adder (b) Multiplier (c) 32-bit 2:1 Multiplexer

Figure 15: 4-D plots of p, d, and s versus power for ADD and MULT operations of an ALU and 2:1 multiplexer synthesized with standard cells for a 160nm OKI ASIC process. Power is indicated as a color between black and white where solid white represents the least power consumed by the device and black indicates the most power consumed by the device. Measurements taken at 0.1 intervals in each dimension *p*, *d*, and *s*.

The power consumption for several ALU operations and interconnect multiplexers synthesized with standard cells for a 160nm OKI ASIC process is displayed in Figure 15 . Power is indicated as a color between black and white where solid white represents the least power consumed by the device and black indicates the most power consumed by the device. Measurements were taken at 0.1 intervals in each dimension *p*, *d*, and *s*. The actual input vectors for power simulation were generated using the technique described in [44]. The synthesis was completed using Design Compiler and the power was estimated using PrimePower; both of these tools are from Synopsys [46]. The reason for the wedge-like shape in the graphs is that several combinations of p, d, and s are not possible. For example, if the probability of '1's, p, is extremely low or high, it reduces the amount of switching, d, that is possible between vectors.

While transition density d is often considered the only metric of interest, Figure 15 provides an indication of how it can be insufficient. The results shown in Figure 15(a) reveal that high transition density dominates power consumption in the adder. However, the plot for the multiplier in Figure 15(b) shows higher power consumption correlating with high spatial correlation. The multiplexer in Figure 15(c) appears to be dominated by transition density, but has the most power consumption when d is neither too high nor too low. As a result of these findings, all power measurements taken in this work are weighted averages across all valid p, d, and s values with 0.1 probability increments ranging from 0.5 to 0.95.

4.2 DESIGN SPACE CASE STUDIES

To begin studying the impact of various parameters of the fabric, the ALU and multiplexer elements were studied individually. Different architectural techniques for implementation of the ALU were studied including a study of varying the width of the ALU. ALU used as a pass gate and a dedicated pass gate were also power profiled. The cardinality of the multiplexers was varied from 2 to 32 in powers of 2 and profiled for power consumption.

4.2.1 Functional unit implementation tradeoffs

The power consumed in the fabric is also heavily dependent on the ALU power consumption. Several architectural techniques for implementing computations in the functional units were profiled including a high performance ALU, a power optimized ALU and individual functional units directly. These blocks were synthesized using 160nm OKI standard cells. The results are shown in Figure 16. The $Hardware$ bar corresponds to the power results for the blocks independently synthesized for each operation. The ALU bar corresponds to a synthesizable ALU built structurally from the Mentor Moduleware components. This ALU was originally designed for maximizing performance. It executes each function in parallel and selects the result using a multiplexer after the computation completes. Finally, the $Optimized\ ALU$ bar represents a low-power ALU in which latches are used at the input to each operation to avoid unnecessary switching of the rest of the hardware blocks when only a single operation is executed. For example, when an ADD operation is being executed, the other components like multiplier, shifter, etc. are latched to the last computation completed by that hardware block. Thus, power profiling provides an insight into the impact of power on the functional units. The power consumed by the performance optimized

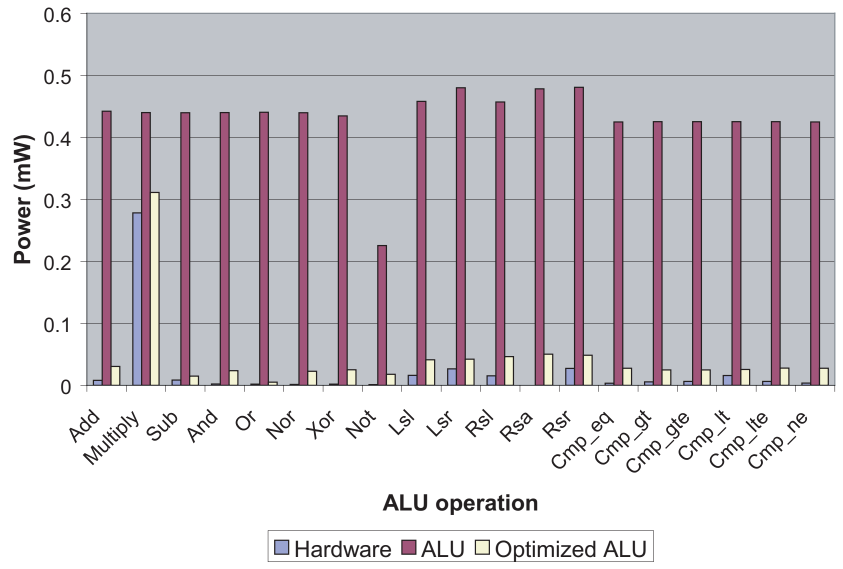


Figure 16: Power results for several functional unit implementation techniques. Results for a standard cell 160nm OKI ASIC process.

ALU is most likely dominated by the multiplier power, making it a poor power choice in the fabric. While, the additional latches may create an area increase, the delay increase is added to all paths in parallel, thus creating a minimal delay overhead. The optimized ALUs are used as the computational elements of the fabric.

4.2.2 Impact of datawidth on power, performance and area of functional units

While many hardware fabric projects agree that coarse-grained functional units are important for performance improvement of reconfigurable devices, the actual granularity varies across projects. The datawidth of each functional unit has a significant impact on the power dissipation of the fabric. Thus, 8, 16 and 32-bit ALUs, which are candidates to be used as computational elements, have been power profiled for several ALU operations. A parameterized model of an ALU was written in VHDL and ALUs of different datawidths was synthesized and profiled for latency and power. The area results are shown in Table II. The area is reduced to more than half when datawidth is decreased from 32 to 16 bits. The same trend is seen when datawidth is reduced from 16 to 8 bits. This area change is likely due to the multiplier, which is the most complex functional unit included in the ALU.

Table 2: Area results for ALUs of different datawidths

Design Unit	**Area**(μm^2)
32-bit ALU	18492.2
16-bit ALU	7787.63
8-bit ALU	3670.5

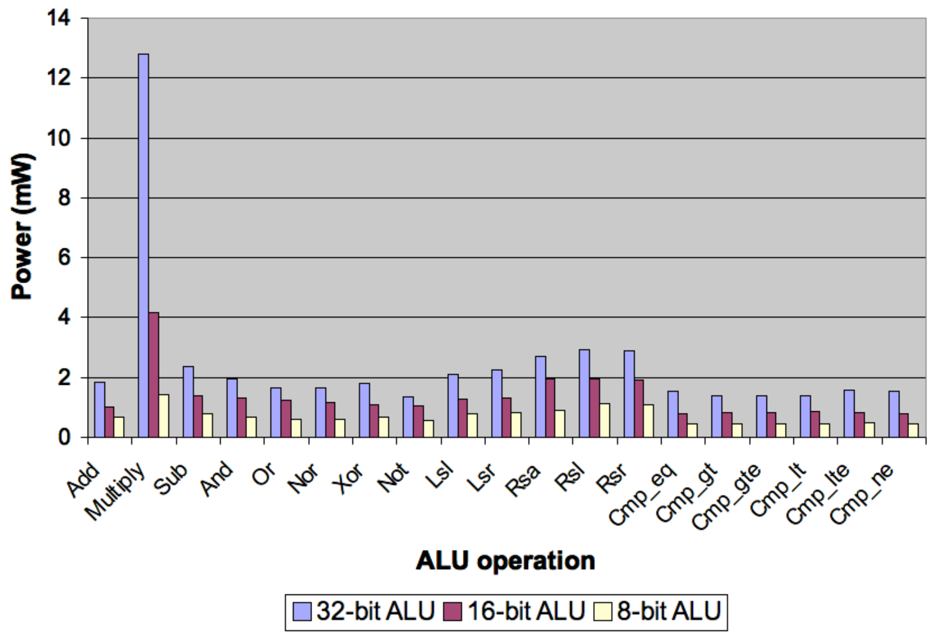


Figure 17: Power consumption of ALU for different datawidths.

The results shown in Figure 17 reveal that as datawidth decreases by half, power dissipation also decreases by nearly 50% for all cases excepting multiplication. Because combinational multiplication has a stacking complexity it grows faster than other functional units, and the power reflects that decreasing to about 30% of the 32-bit version. There is a similar power trend between

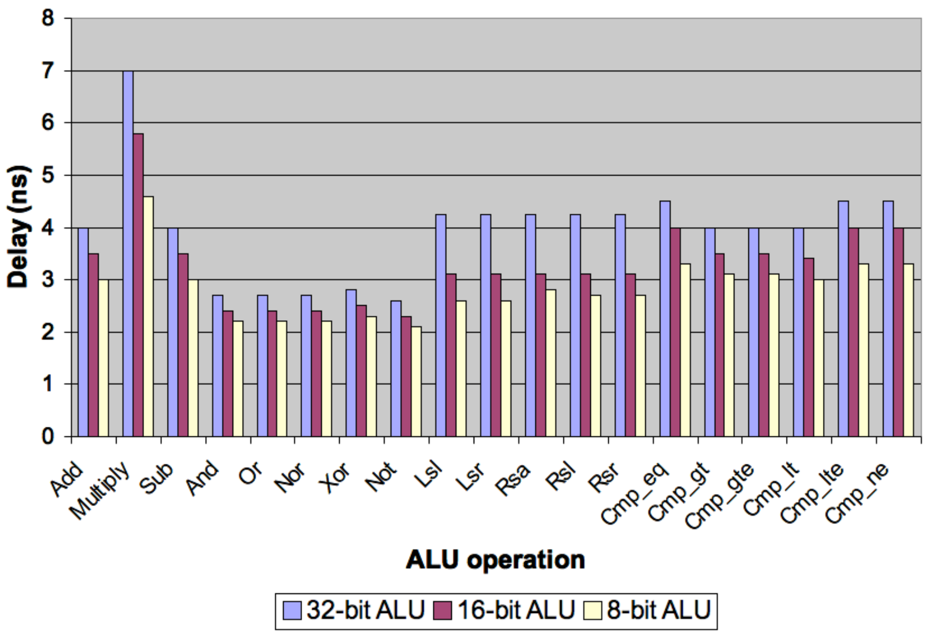


Figure 18: Delay results of ALU for different datawidths.

16-bit and 8-bit operations. Figure 18 describes the latency of each bit-width. While as expected, the latency is lowest for the 8-bit ALU operations the change is only a nominal decrease over a 16-bit ALU. Even compared to a 32-bit ALU, the delay improvement is less than 50% at the cost of 3/4 of the bandwidth for the computation.

The energy results of ALUs of different datawidths are shown in Figure 19. This chart includes the power consumption of using a 32-bit wide ALU to compute both 16-bit and 8-bit values in comparison to computing them directly on a 16-bit or an 8-bit wide ALU. This was done to calculate the energy consumption overhead of a 32-bit ALU used for lower bit width operations

Consider the dedicated width case (from left to right, the first, second, and fourth bars of each operation), as datawidth decreases by half, energy dissipation also decreases almost by half for most of the ALU operations except multiplier. In the case of multiplier, the energy is reduced to almost one-fourth when datawidth decreases by half. However, the overhead in using a 32-bit ALU for 16-bit operations as compared to a 16-bit ALU is 2.5X on the average. The same trend

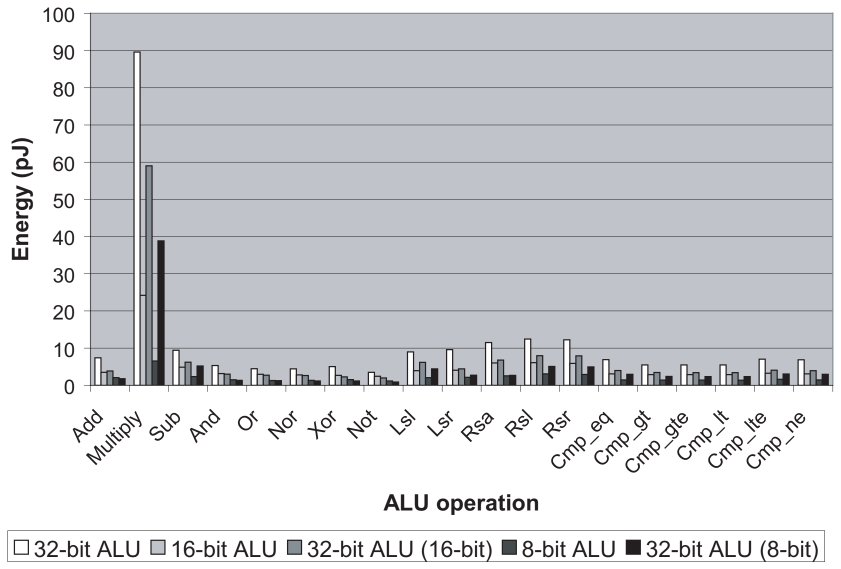
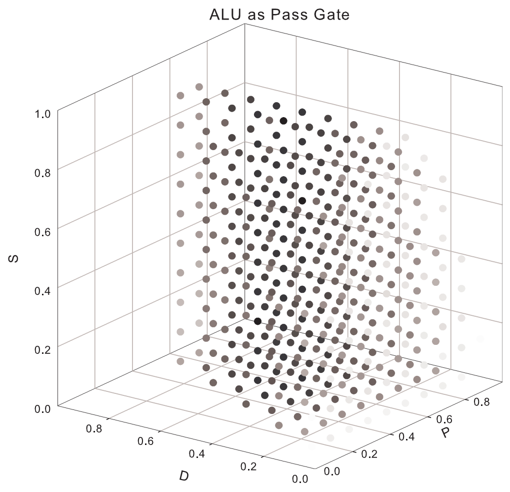


Figure 19: Comparison of Energy consumption of 32-bit ALU, 16-bit ALU, 32-bit ALU used for 16-bit operations, 8-bit ALU and 32-bit ALU used for 8-bit operations.

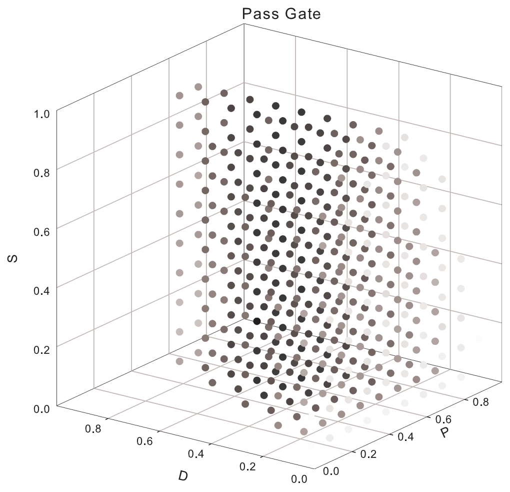
is observed if we compare the energy results of a 32-bit ALU used for 8-bit operations and a 8-bit ALU. However, when the multiplier is removed from consideration, the overheads are much lower. While, it appears that the overhead actually decreases for some of the logic operations, this is unlikely, and the difference between the two calculations does not exceed the estimated inaccuracy from using the PrimePower simulation over a SPICE level simulation. We used 32-bit ALUs in our domain-specific fabric.

4.2.3 Dedicated Pass-gates

When mapping a DFG to a stripe-style structure, data dependency edges often traverse multiple rows. In these fabrics, ALUs must often pass these values through without doing any computation. We call these operations in the graph, pass-gates. However, these ALUs used as pass-gates are certainly an area-inefficient and possibly also power-inefficient method for vertical routing. The power distribution using average input signal probability, p, average transition density, d, and spatial correlation s for two vertical routing options shown in Figure 20 was studied. The first option was to use an ALU as a pass-gate shown in Figure 20(a) and the second is a simple routing structure that could only pass a value from the left operand, the right operand, or act as a NOOP is shown in Figure 20(b).



(a) ALU used as pass-gate.



(b) Dedicated pass-gate.

Figure 20: Power profiles for various vertical routing structures.

The power trends indicate a similar power profile between the two vertical routing implementations, relating to the probabilities that produce the minimum and maximum power, as would be expected. However, the power scales are very different between the two structures as shown in Table 3, using an ALU as a pass-gate requires over an order of magnitude more power than a dedicated pass gate implementation. As a result, if the structure of the benchmarks requires vertical routes there is an opportunity to reduce power consumption by using these simpler dedicated pass gates.

4.2.4 Multiplexer cardinality impact on power

The power required in the interconnect depends heavily on the cardinality of the multiplexers in the interconnection stripe. The power impact of the cardinality of the multiplexer is shown by the trends from Figure 21. As shown in the figure, the maximum, minimum and average powers consumed in the multiplexers increases linearly with the cardinality. While not shown explicitly,

Table 3: Power comparison of ALU used as a pass-gate and a dedicated pass-gate.

	ALU as pass-gate	**Dedicated pass-gate**
Minimum Power	0.78 mW	0.03 mW
Maximum Power	6.46 mW	0.28 mW
Average Power	4.02 mW	0.17 mW

reducing the cardinality also reduces the delay of the multiplexer. Thus, reducing the multiplexer complexity is desirable if allowed by the needs of the applications of interest.

4.2.5 Benchmark Driven Interconnect Design

In order to study the system level interconnect, a set of core signal processing benchmarks were examined. Algorithms of interest were selected from different categories such as voice compression, image and video coding and wireless communications. The software codes examined were taken from the MediaBench benchmark suite. SDFGs were created for these benchmark circuits and mapped to the fabric model for verification and analysis. Mapping was done both by hand and through an automated mapping flow. The number of operands of an ALU and the number of operations of each ALU are fixed in this particular analysis. The design flow of the fabric model is shown in Figure 22.

Figure 23 shows an example mapping for the ADPCM decoder kernel SDFG from the automated mapping flow that is part of the flow in Figure 22. In this example, the circular blocks represent input and output values at the top and bottom, respectively. The rectangular blocks represent ALU resources. Multiplexers to allow routing are not shown in this representation, however, the actual routes are shown. The grayed boxes represent ALUs that are idle for this mapping. This representation illuminates a property of the SDFGs, they tend to be triangular requiring many ALU resources near the top and relatively few near the bottom.

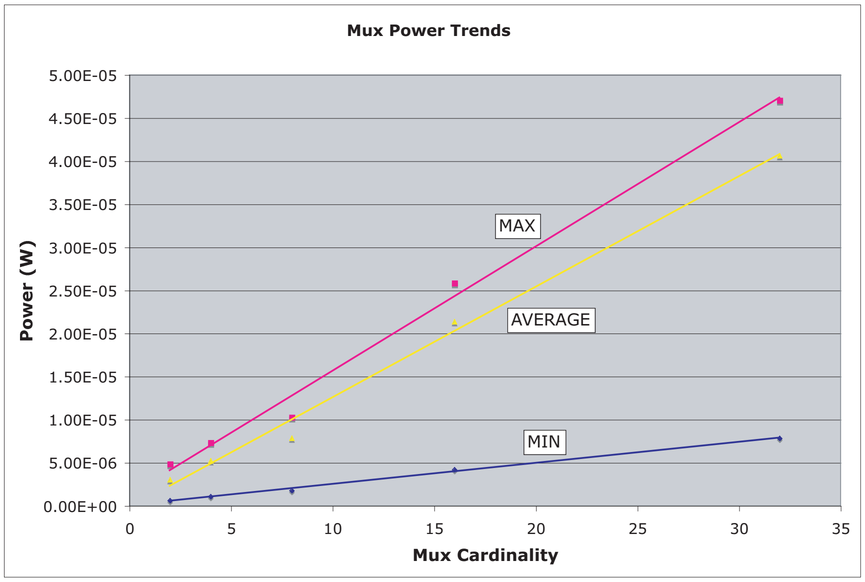


Figure 21: Power consumption in multiplexers of different cardinalities. Results for a standard cell 160nm OKI ASIC process.

4.2.5.1 Interconnect profiling To examine the need for complex multiplexers, a mapping assuming full interconnection stripe connectivity was performed without any particular bias to reduce lateral distance between computations in successive rows. The multiplexers were then categorized based on whether they would have been satisfied with a smaller multiplexer from the group of 3:1, 5:1, 9:1, 17:1 and 33:1 multiplexers. As it can be seen in Figure 24, the connections satisfied with 3:1 multiplexers is greater than 45% in almost all the applications, and the need for higher cardinality multiplexers like 17:1 and 33:1 is minimal.

Due to the trend of power consumption versus multiplexer cardinality shown in Figure 21 in this Chapter, it is desirable to utilize 2:1 multiplexers wherever possible. It is also well known that low cardinality multiplexers reduce delay, allowing improved performance. Based on the trends from Figure 24, it appears that a fabric with mostly 2:1 multiplexers and possibly a handful of larger multiplexers would be a good architectural choice.

While 2:1 multiplexers are desirable for these reasons, 4:1 multiplexers are more practical for two reasons, first hardware predication requires three operands, which for our architecture makes

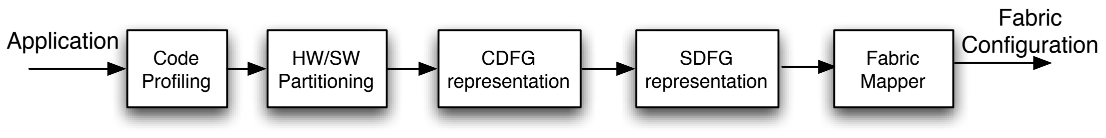


Figure 22: Design flow for creating SDFGs to run on the fabric.

2:1 multiplexers insufficient and automated mapping based on 2:1 multiplexers is too limiting to achieve a solution without adding a lot of extra pass gates in the design. It is for these reasons a uniform fabric architecture was considered, which limits connectivity from four locations in the prior stripe to the current functional unit. The fabric model with 4:1 multiplexers was considered that allows connections from above, one column left, and two columns right of the prior stripe to the current stripe as shown in Figure 25. However, depending on the needs of the applications to be implemented, this regular structure may not be appropriate.

While many structures can be successfully mapped using this interconnection strategy, this configuration is somewhat limited, as it provides only a 4-way fanout from one node to other nodes in the circuit. Consider a subgraph that appears frequently in signal and image processing applications as shown in Figure 26. Between the first and second rows, this graph consists of three pairs of nodes (A,B), (C,D), and (E,F) that communicate with three other pairs of nodes (G,H), (I,J), and (K,L) in the subsequent stage. However, between the second and third stage nodes G-H are grouped into three different pairs (G,I), (H,K), and (J,L) which communicate with three new pairs in the third stage (M,N), (O,P), and (Q,R). While separately, either of these rows could be implemented with the configuration in Figure 25, the edges connecting H with P and J with R are not possible in this configuration as shown in Figure 27.

The graph in Figure 26 is not a subgraph of the graph created by the connectivity in the fabric supplied by the 4:1 multiplexers as oriented in Figure 25. Two edges represented by dashed lines are not available in this structure. This is not solved by permuting the nodes. Figure 28 presents all four orientations possible for two pairs of nodes with dependency edges under this connectivity. Unfortunately, none of these configurations can be overlapped with another pair of nodes, which

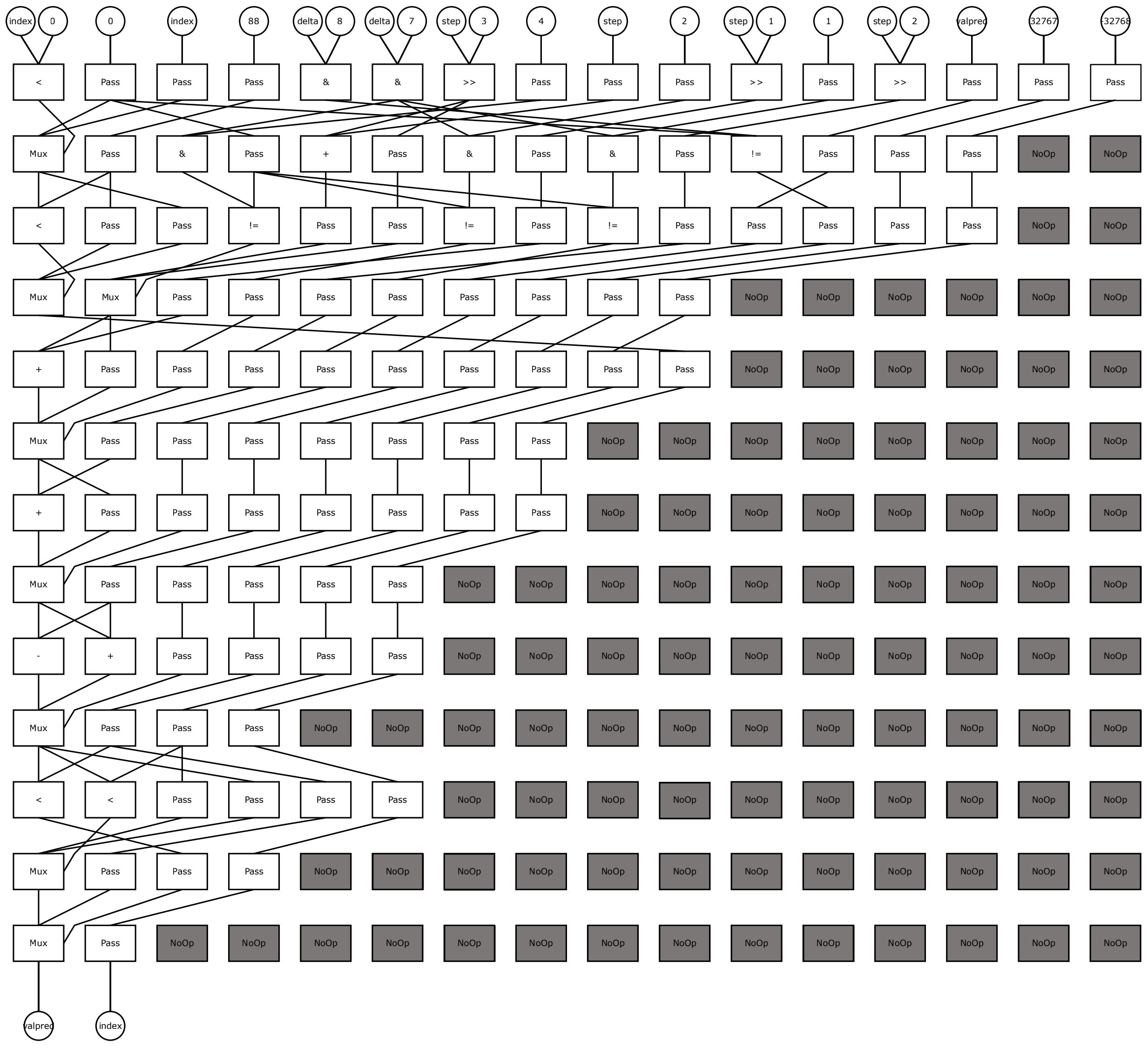


Figure 23: Mapped SDFG of an ADPCM decoder onto a particular fabric instance.

shows that permutation of the nodes is not possible. To effectively map this structure, a 5:1 multiplexer is required. However, a true 5:1 multiplexer would require an additional control bit and an additional level of logic, which is undesirable from a power and performance perspective.

But there is a way to make a 5:1 multiplexer using two 4:1 multiplexers. The connectivity shown in Figure 4 allows an emulation of a 5:1 multiplexer without increasing the architectural

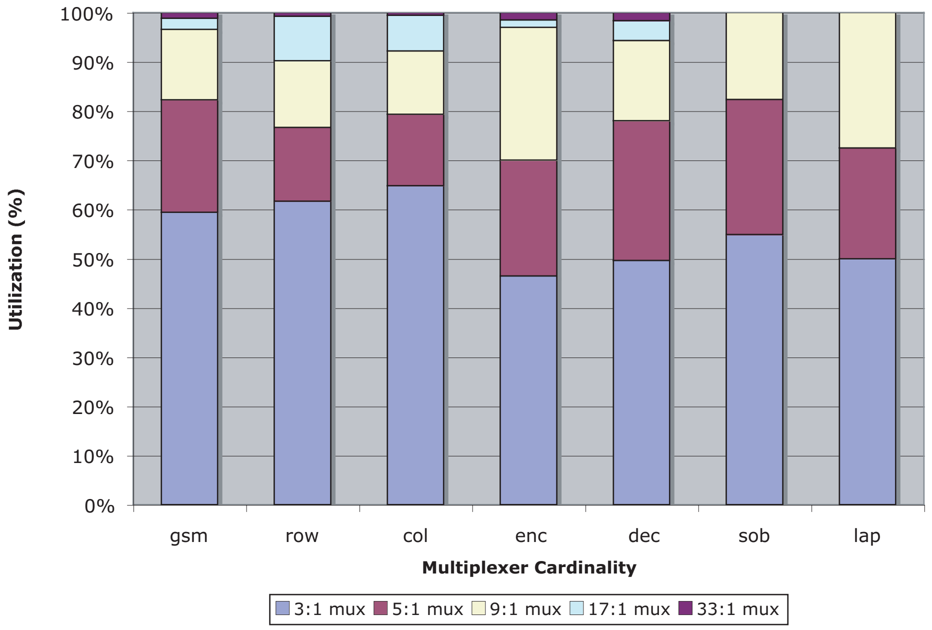


Figure 24: Multiplexer cardinality usage.

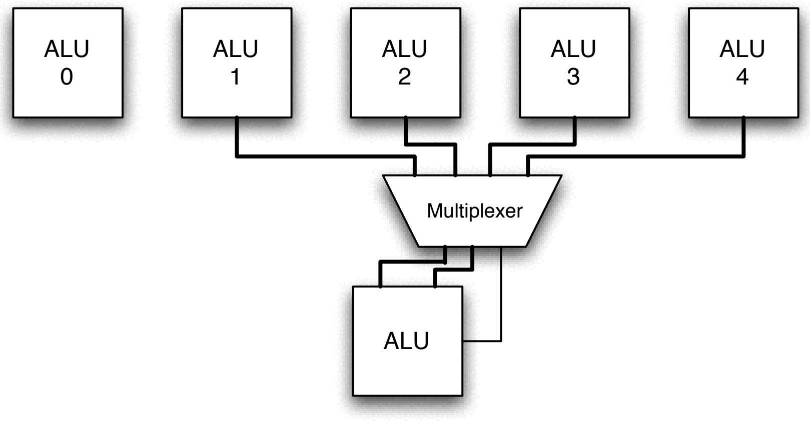


Figure 25: Connectivity using 4:1 multiplexers.

complexity beyond 4:1 multiplexers. One 4:1 multiplexer is the mirror image of the other one. In this case, the three internal ALUs, 1-3, are shared on both operand's multiplexers. The outermost

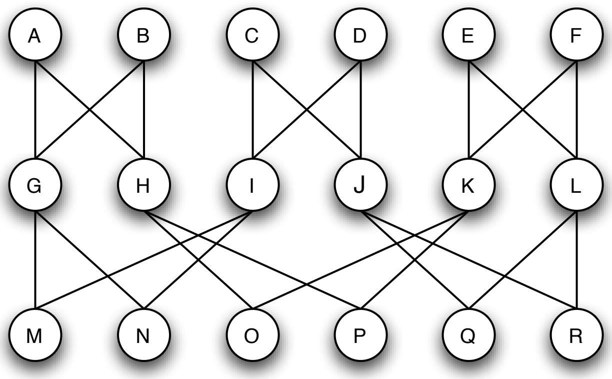


Figure 26: Partial SDFG that occurs frequently in image and signal processing applications.

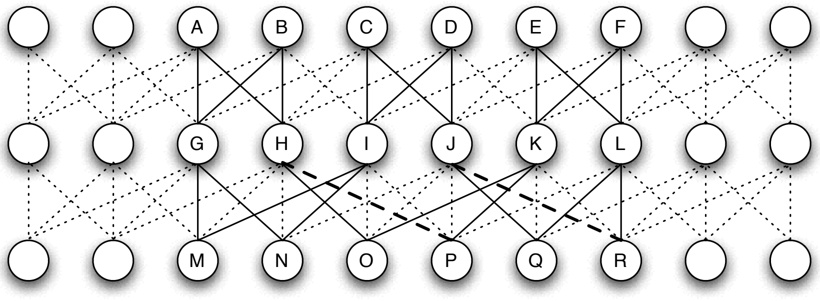
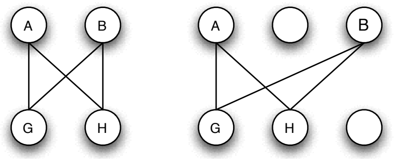
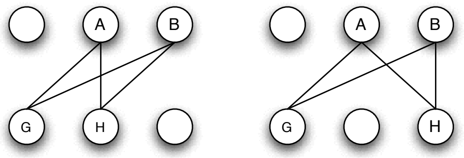


Figure 27: Attempt to embed the graph from Figure 26 into the connectivity supplied by Figure 25.

ALUs, 1 and 5, are only available on the left and right operand multiplexer, respectively. As shown in Figure 29, it is relatively easy to embed the graph from Figure 26 into the connectivity provided by Figure 4. Based on the study of several applications graphs, the fabric with 5:1 interconnect multiplexers has been found to provide a good baseline architecture to relatively easily map the applications on to the fabric. The biggest limitation to this approach is that for non-commutative operations such as subtract, there is some restriction as to which operand may be retrieved from the far left or far right. But the limitation of non-commutative operation mapping can be overcome in the architecture of the fabric by providing a separate operation that executes the operation right to



(a) Standard orientation. (b) Top node shifted right.



(c) Top two nodes shifted right. (d) Bottom node shifted left.

Figure 28: Ways to overlap node pairs with the 4:1 connectivity.

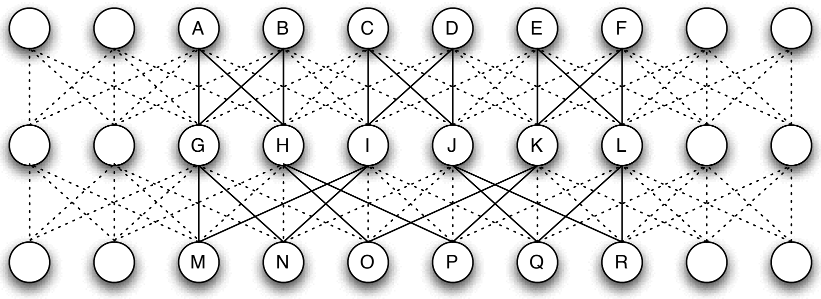


Figure 29: Embedding the graph from Figure 26 with the connectivity provided by Figure 4.

left in addition to left to right. Thus, the ALUs of the fabric can be extended to have capability of computing non-commutative operations.It may be possible to further optimize a 5:1 interconnect and still provide the necessary interconnection to implement the appropriate graphs. For example, if some number of these 5:1 multiplexers could be replaced with 3:1 multiplexers, built from 2:1 multiplexers in a similar manner as Figure 4 power and delay could be reduced if no or minimal overhead is increased due to the mapping.

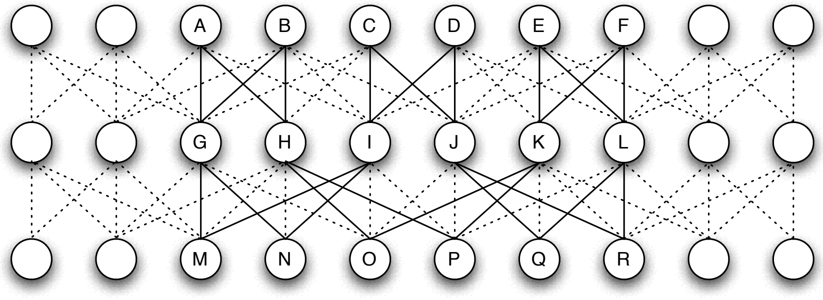


Figure 30: Embedding the graph from Figure 26 into a 3-5-5:1 interconnect.

Consider the fabric structure in which one-third of the 5:1 multiplexers are replaced with 3:1 multiplexers. Consider the structure where there are two 5:1 multiplexers adjacent, followed by a 3:1 multiplexer and this interconnect pattern is repeated throughout the fabric. This interconnect is named as the 3-5-5:1 interconnect. The graph from Figure 26 can relatively easily be mapped into this structure as shown in Figure 30.

Consider the fabric structure where half of the 5:1 multiplexers are replaced with 3:1 multiplexers and a 3-5-5-3:1 interconnect pattern has been repeated throughout the fabric. The graph embedded in an interconnect with one-half 3:1 multiplexers is shown in Figure 31. The fabric model is very generic in a sense that different parameters of the interconnect multiplexers can be set up in the model to generate fabric instances with different interconnect patterns.

Then a dense subgraph from the ADPCM encoder benchmark that contains several pass-gates in addition to the multiplexers and other logic has been tried to map onto the fabric. In this case, dedicated pass-gates in the architecture can be heavily utilized by the mapped graph. Figure 32 shows a mapping of the subgraph onto a 5:1 interconnect where 25% of the ALUs have been replaced with fixed, dedicated pass-gates equally spaced within each stripe. In this representation, fixed pass-gates are represented by squares, and ALUs configured to be pass-gates are circles with a `P` inside. The multiplexers (trapezoids) do not imply fixed selectors, just ALUs configured for selection. As shown in the figure, the graph can be mapped without increasing the number of rows. 8 of the 13 total pass-gates can be implemented by the fixed dedicated pass-gates as shown in the figure. By leveraging the fact that a pass-gate is unary, the multiplexers can be rearranged to

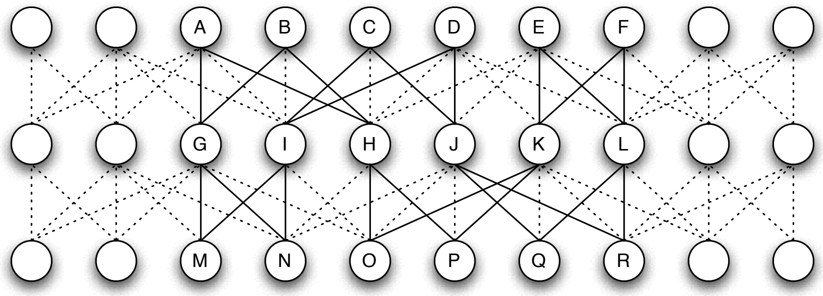


Figure 31: Embedding the graph from Figure 26 into a 3-5-5-3:1 interconnect.

mimic an 8:1 interconnect for the dedicated pass-gates. This can increase the number of dedicated pass-gates used to 9 of 13, or 69%.

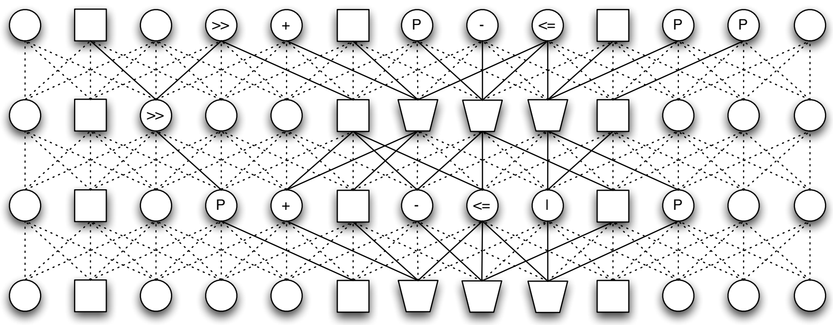


Figure 32: Embedding the dense subgraph from ADPCM encoder into a 5:1 interconnect with 25% of ALUs replaced with dedicated pass-gates.

However, the introduction of these dedicated pass-gates can make it more difficult to map dense subgraphs. The addition of even a single pass-gate within the partial butterfly from Figure 26 makes it impossible to be mapped with a 5:1 interconnect. Thus, we have created a 6:1 multiplexer as shown in Figure 33. The 6:1 interconnect allows the right operand to be read from a maximum distance of three to the right. This allows the graph to be mapped in a system where 25% of the ALUs are replaced with dedicated pass-gates. The solution is based on Figure 31 and is shown in Figure 34. By utilizing this extra distance to read from the right, the edges (D,I) and (O,K) can be accommodated, which is not possible with the 5:1-based interconnect.

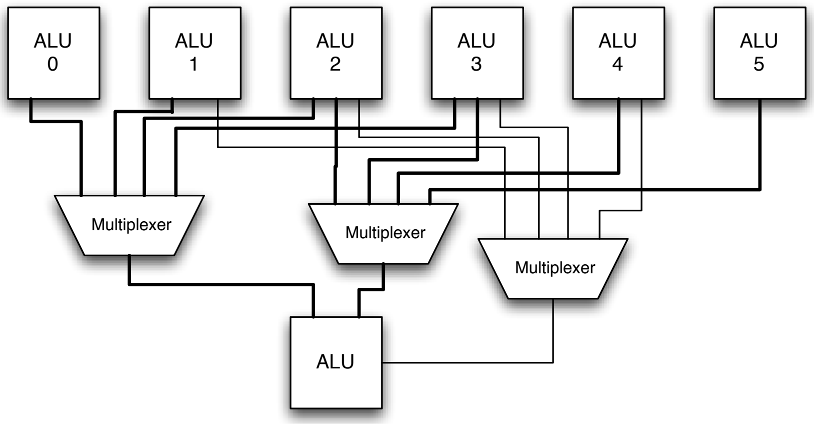


Figure 33: Schematic for building a 6:1 reaching multiplexer interconnect using 4:1 multiplexers.

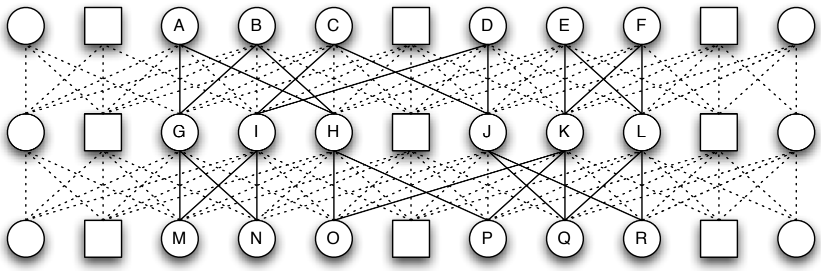


Figure 34: Embedding the graph from Figure 26 into a 6:1 interconnect with 25% dedicated pass-gates.

5.0 MAPPING OF BENCHMARKS ONTO THE FABRIC USING SIMULATED ANNEALING

A mapping of a data flow graph onto a fabric consists of an assignment of operators in the data flow graph to ALUs of the fabric such that the logical structure of the data flow graph is preserved and the parameters of the fabric are respected. This mapping problem is central to the use of the fabric, as a solution must be available in order for the fabric to be reprogrammed for a specific data flow graph.

Because of the layered nature of the fabric, the mapping is also allowed to use ALUs as "pass gates," which take a single input and pass the input value to one or more outputs. In general, not all of the available ALUs and edges will be used. An example mapping is shown Figure 35.

One of the more complicated parts of creating a mapping is the introduction of pass gates to fit the layered structure of the fabric. A successful approach that has been used works in two stages. In the first stage, pass gates are introduced heuristically and operators assigned to rows so that all edges go from one row to the next. The second stage assigns the operators to columns so that the fabric interconnect is respected. This second stage is called Feasible Mapping with Fixed Rows [35, 47]. Depending on the interconnect design, there may or may not exist a feasible mapping. In this dissertation research, Simulated Annealing was used to solve the problem of Feasible Mapping with Fixed Rows. Simulated Annealing is a popular algorithm for computer aided design flows targeting custom hardware (either for standard cell ASICs or FPGAs) particularly for placement of cells. It was also used to study and optimize the interconnect for a specific domain of applications.

The Simulated Annealing formulation uses a graded cost function based on the multiplexing cardinality required to satisfy the edges. Lower costs have been assigned to the nearby edges and the cost gets higher for the distant edges. In this case, the starting solution is the assignment of the operators to the ALUs of the fabric according to ASAP scheduling. The neighborhood

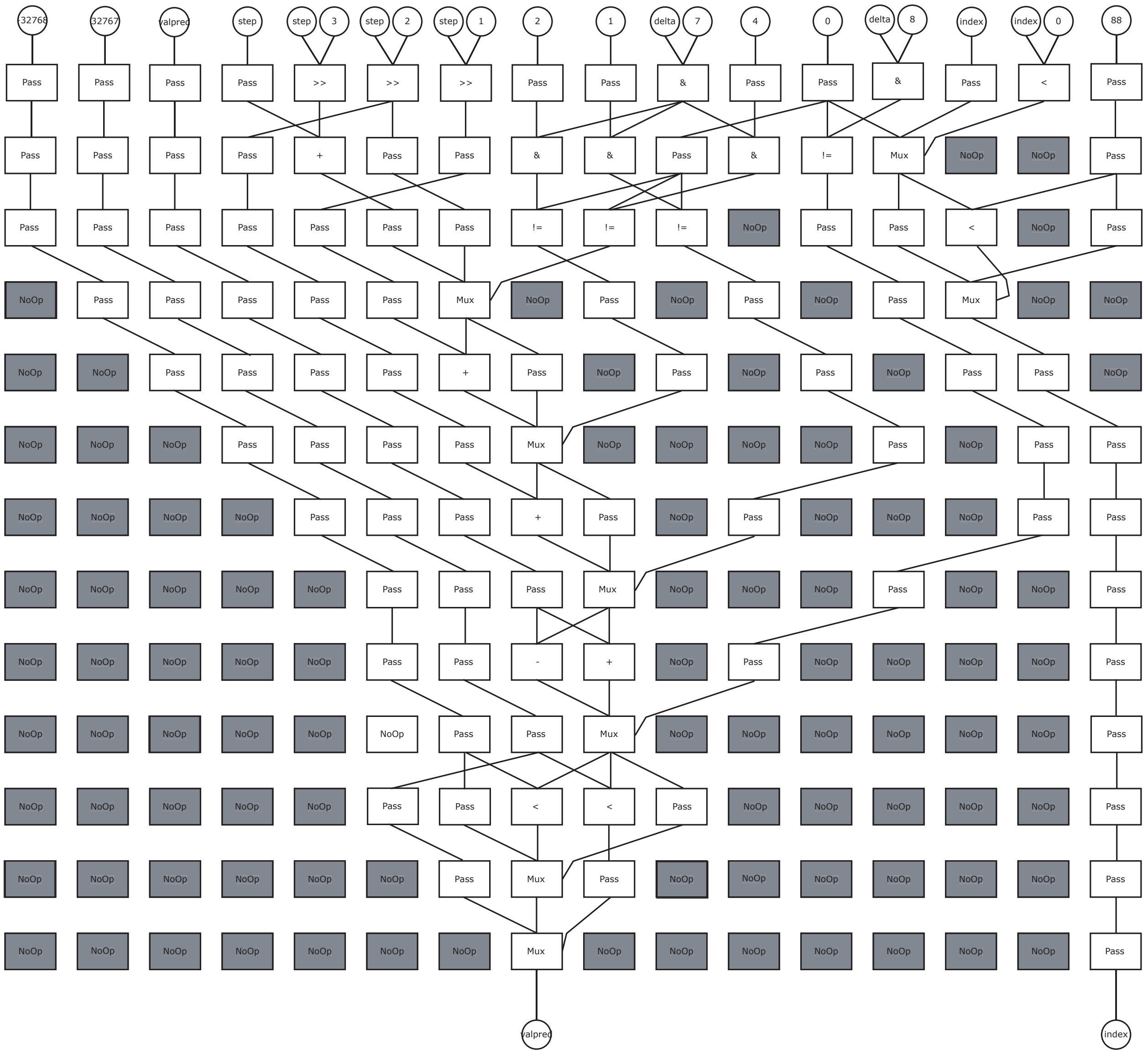


Figure 35: Mapping of a benchmark onto the fabric.

for the model consists of two possible moves: (1) a node is moved from one column location to an unoccupied column within the same row or (2) two nodes within the same row are selected and swapped. A candidate move is selected uniformly at random from the choices above. The simulated annealing algorithm accepts all moves that decrease the overall cost of the system and accepts moves that increase the overall cost with the probability described in Equation 5.1 based on the Boltzmann equation from thermodynamics. k_B is the Boltzmann constant and T is the temperature in the system.

$$P(\Delta C) = e^{\frac{-\Delta C}{k_B T}} \tag{5.1}$$

The exponential cooling schedule shown in Equation 5.2 was used to decrease the probability that bad moves will be accepted as the algorithm proceeds. α is a multiplier in the range between 0 and 1. We selected $\alpha = 0.9$ based on initial experimental results.

$$T_{new} = \alpha T_{old} \tag{5.2}$$

The above formulation is able to solve six of the instances, particularly for the cardinality 8 interconnect shown in Table 4. However, the results for the cardinality 5 interconnect were not as successful: the approach failed to find feasible solutions for three of the instances.

The possibility of a larger neighborhood that would allow moving a node up or down one row was considered. Unfortunately, due to the data dependencies in the flow graphs such moves could affect the positions of many other nodes in the graph—including some nodes that are many rows away. This dependency makes such a neighborhood harder to implement and slower to solve, and initial tests did not find that it provided significant benefits.

Post-processing: In most cases, upon completion of the Simulated Annealing run there are still several violated edges. In this case, a local search algorithm was run with the same neighborhood of moves from Simulated Annealing to ensure that a local optimum has been discovered.

The Simulated Annealing algorithm was implemented in C++ and run on a hyper-threaded Pentium 4 operating at 3.2 GHz with 4G of memory and running Linux. The results for mapping size and algorithm runtime are shown in Table 4 for the Feasible Mapping with Fixed Rows problem. For the cardinality 5 interconnect, four of the benchmarks are able to be mapped. For the cardinality 8 interconnect, six of the benchmarks are possible to be mapped. The algorithm takes between a half minute to two and half minutes to execute, primarily depending on the size of the graph to be mapped. There is no difference in the algorithm when using cardinality 5 or cardinality 8 interconnect; the same graded cost function is used in either case. Therefore, a single

Table 4: Runtime and solution quality for the simulated annealing algorithm.

	Cardinality 5		Cardinality 8	
	size	runtime (s)	size	runtime (s)
sobel	10x9	25	10x9	25
laplace	15x8	27	15x8	27
gsm	16x18	125	16x18	125
adpcm_decoder	16x13	82	16x13	82
adpcm_encoder	-	160	20x16	160
idctcol	-	165	-	165
idctrow	-	120	17x10	120

run tests both interconnects. This explains why the run times are identical for the two interconnects. Simulated Annealing was compared with other mapping strategies like mixed-integer linear programming (MILP) [48], constraint programming (CP) [35], and a custom heuristic (H) [47]. MILP is a common modeling and solution technique for combinatorial optimization. Constraint programming is a modeling and solution methodology based on the ideas of intelligent enumeration and reduction of the search space through careful analysis of constraints. The greedy heuristic mapper follows a top-down mapping approach to provide a feasible mapping for a given benchmark. Both the cardinality 5 interconnect and the cardinality 8 interconnect (which is easier to map) were tested for all the mapping approaches. Table 5 gives the number of extra rows needed by each method. Table 6 shows the run times for each approach. The Simulated Annealing approach is promising for those instances that it can solve, but it cannot solve several of the instances. The MILP approach is valuable in showing that a mapping is possible using the minimum number of rows in each case. However, the run times for the MILP are quite long compared to the other approaches, as may be expected. In practice, the MILP is likely to be a good choice if minimizing the number of rows is of high importance and using several hours of processing time is acceptable. For larger instances, however, it is likely that even several hours will not be sufficient for the current MILP formulation. Both the constraint program and the heuristic algorithm offer reasonably

Table 5: Rows added by various mapping strategies with different interconnects. A hyphen (-) indicates that no solution was found.

	Cardinality 5				Cardinality 8			
	MILP	SA	CP	Heuristic	MILP	SA	CP	Heuristic
sobel	0	0	0	0	0	0	0	0
laplace	0	0	1	0	0	0	1	0
gsm	0	0	0	1	0	0	0	0
adpcm_decoder	0	0	0	0	0	0	0	0
adpcm_encoder	0	-	7	2	0	0	0	0
idctcol	0	-	0	7	0	-	0	0
idctrow	0	-	1	3	0	0	0	0

good performance in terms of number of rows added, though each has one instance where many rows are added. The heuristic algorithm has by far the best execution times, with none over 10 seconds. Each mapping technique has its own advantages and disadvantages. There exists a tradeoff between the quality of the solution and the execution time.

5.1 OPTIMIZING THE INTERCONNECT

To examine the needs of target applications, the representative signal and image processing benchmarks were mapped to a fabric with fully interconnected stripes (e.g. any ALU from a previous stripe could be connected to any ALU in the current stripe). At first, no particular emphasis was made to optimize the locations within the fabric to reduce wire length, this routing requirement is shown as the left most bar for each benchmark in Figure 36. This non-optimized mapping shows that over 60% of the multiplexers required are only 3:1.

Using a mixed-integer linear programming (MILP) formulation that provided an increasing penalty to using 5:1, 9:1, and 17:1 routes or higher, the bar in Figure 36 marked (IP) was created.

Table 6: Runtime in (seconds) of various mapping strategies with different interconnects. A star (*) indicates that no solution was found.

	Cardinality 5				Cardinality 8			
	MILP	SA	CP	Heuristic	MILP	SA	CP	Heuristic
sobel	23	25	1	1	37	25	< 1	< 1
laplace	101	27	47	1	13	27	27	< 1
gsm	3104	125	2	3	514	125	1	< 1
adpcm_decoder	2916	82	2	4	155	82	1	< 1
adpcm_encoder	2555	160*	277	10	306	160	1	4
idctcol	2825	165*	7	8	894	165*	10	1
idctrow	1175	120*	36	6	460	120	1	< 1

This technique was run either until completion of an optimal solution or for 48 hours, whichever was less, so not all solutions are optimal. The IP technique eliminates the need for nearly all multiplexers greater than 5:1 for some of the benchmarks. Then the results obtained from Simulated Annealing, the bar in Figure 36 marked (SA) were compared with the results obtained from MILP. The Simulated Annealing technique eliminates the need for nearly all the multiplexers greater than 9:1 for five of the benchmarks.

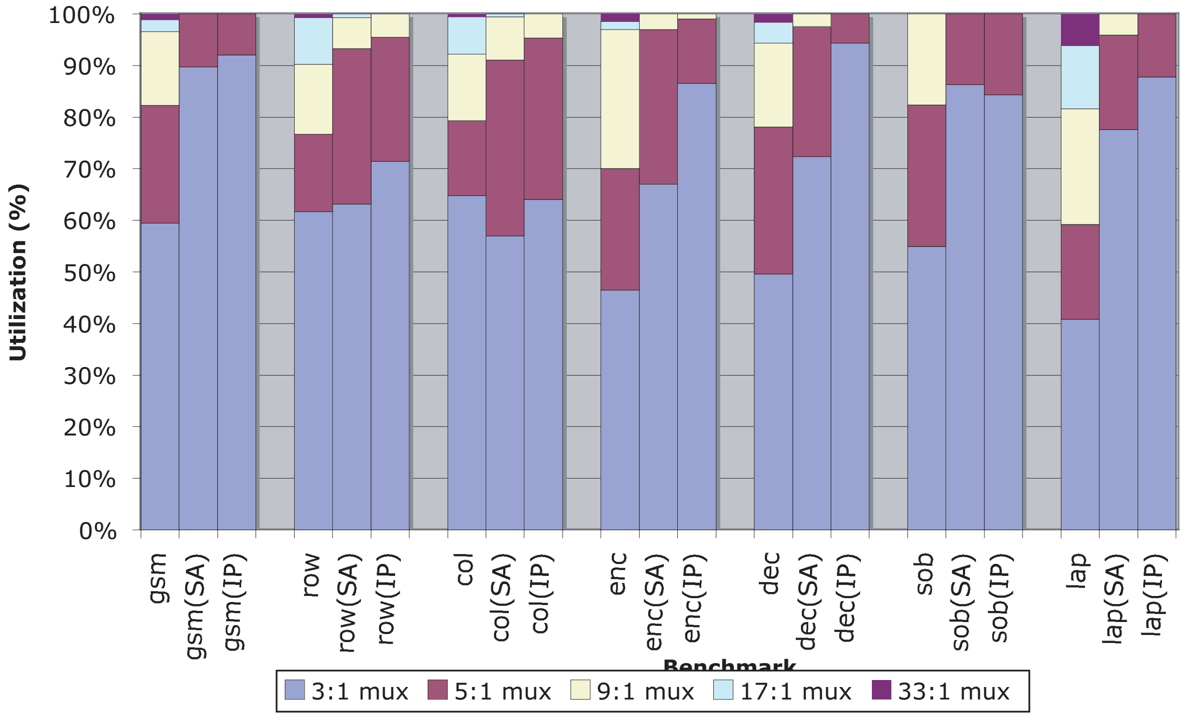


Figure 36: Multiplexer cardinality usage for a variety of signal and image processing applications using ASAP scheduling, Simulated Annealing, and IP.

6.0 POWER AND PERFORMANCE ANALYSIS OF THE BENCHMARKS IMPLEMENTED ON THE FABRIC

In order to evaluate power and performance, a set of core signal processing benchmarks were selected from MediaBench benchmark suite including the ADPCM encoder, ADPCM decoder, GSM channel encoder, and the MPEG II decoder. We added the Sobel and Laplace edge detection algorithms to the benchmark suite. The design flow overview using the FIM is shown in Figure 37. The

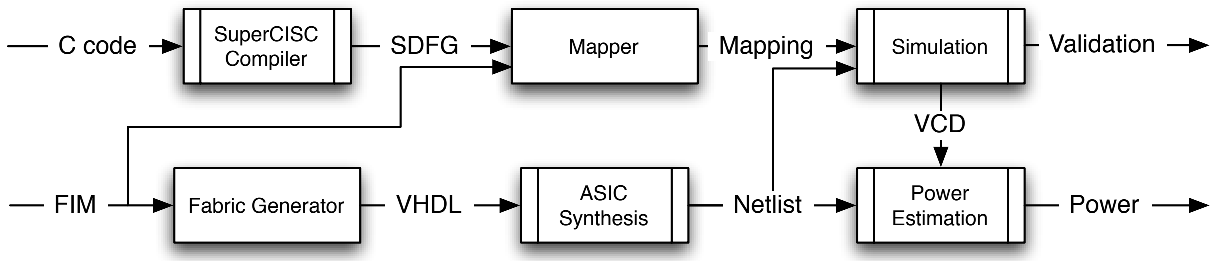


Figure 37: Design flow using FIM.

SuperCISC Compiler [30, 29] takes C code input, which is compiled and converted into an SDFG. The SDFG is then mapped into a configuration for the fabric described by the FIM. The FIM is also used to automatically generate the VHDL for the fabric instance described by the FIM. The fabric instance VHDL is synthesized using commercial tools such as Synopsys Design Compiler to generate a netlist tied to ASIC standard cells. This netlist and the mapping of the application are then fed into ModelSim where correctness can be checked. The mapping is communicated to the simulator to program the fabric device in the form of ModelSim `do` files. The input data to the power simulations was generated by the benchmark datasets supplied with the Mediabench suite or from sample photograph images in the case of the edge detection algorithms. A value change file (VCD) output from the simulation of the design netlist can then be used to determine the power consumed in the design. Synopsys PrimePower and PrimeTime tools are used to estimate the power consumption of the device.

The core components of the fabric model were implemented in parameterized VHDL using the `generic` capability of the VHDL language [33]. However, much of the fabric is generated using scripts that read from the FIM and generate the properly customized synthesizable VHDL.

The mapping heuristic follows a top-down approach to provide a feasible mapping for any given benchmark. Nodes in the SDFG are initially assigned a row in the fabric in a similar fashion to an as soon as possible schedule in high-level synthesis. During this step, pass-gates are introduced in the SDFG for edges that traverse multiple rows. During the second phase, starting with the top row, nodes in each row are completely placed using a limited look-ahead of two rows. Nodes are assigned specific column positions within each row based on the locations of their parent nodes, children nodes, slack, etc. When a node cannot be mapped within the row it is assigned (e.g. it violates the interconnect of the fabric) it is postponed until a later row and pass-gates may be introduced to propagate the values to the next row. After each row is mapped, the mapper will not modify the mapping of any portion of that row.

While the limited information available to the mapper does not often allow it to produce optimal or minimum-size mappings, its runtime is typically a few seconds or less. The FIM is incorporated into the mapping flow as a set of restrictions on what interconnect lines are available, the capabilities of particular functional units (e.g. dedicated vertical routes versus fully capable ALUs) in the system, etc. Additional details on this mapping flow can be found in [35, 49, 50, 47].

6.1 HOMOGENEOUS AND HETEROGENEOUS INTERCONNECTS

The fabric hardware was created using 8:1 and 4:1 multiplexers. We also targeted our 6:1, 5:1, 3-5-5:1, and 3-5-5-3:1 multiplexer-based interconnects as described in Chapter 5 and include a fully connected interconnect for comparison. Fully connected refers to the case where each ALU

Table 7: Fabric size (Width x Height) for mapping various benchmarks onto different interconnects using the heuristic mapper.

	adpcm enc	**adpcm dec**	**idctrow**	**idctcol**	**gsm**	**sobel**	**laplace**
Fully Connected	17x16	16x13	17x10	20x12	14x18	10x9	15x8
8:1	17x16	16x13	17x10	20x12	14x18	10x9	15x8
6:1	17x18	16x14	17x13	20x18	14x18	10x9	15x9
5:1	20x18	16x13	17x13	20x19	20x19	10x9	20x8
4:1	20x20	16x15	17x18	20x28	20x21	10x9	20x8
3-5-5:1	20x22	16x15	20x17	20x24	14x19	20x10	20x9
3-5-5-3:1	17x30	20x21	17x19	20x29	20x22	20x10	20x9

in a row can read from every ALU in the preceding row. Table 7 provides a summary of the area requirements of the benchmarks mapped to the fabric using the different multiplexing cardinalities and using the previously mentioned mapping strategies. Once all benchmarks were mapped to a fabric using a particular interconnect, the fabric size was fixed to the smallest size that could fit all seven benchmarks. The number of rows required by a benchmark implementation has a strong correlation to both the power and performance of the benchmark implementation. Energy is the product of the power and time of the benchmark implementation and as such provides a summary of the impact of the mapping on both factors. Figure 38 shows a comparison of the energy consumption of mapping the benchmark suite onto various interconnect cardinalities.

As the sizes of the mappings for both fully connected and 8:1 interconnections are identical, the reduction the difference in energy consumption demonstrates the impact that the interconnect complexity has on the power and performance of the fabric. Some other items that jump out are that 4:1 and 3-5-5-3:1 interconnects perform very poorly compared to most of the alternatives. The remainder of the interconnects do not show a clear winner, however, 6:1, 5:1, and 3-5-5:1 do outperform 8:1, though not by a large margin.

To highlight the potential impact of the interconnect cardinality, we developed an 0-1 IP formulation that attempts to discover an optimal mapping (e.g. no additional rows) for the benchmark

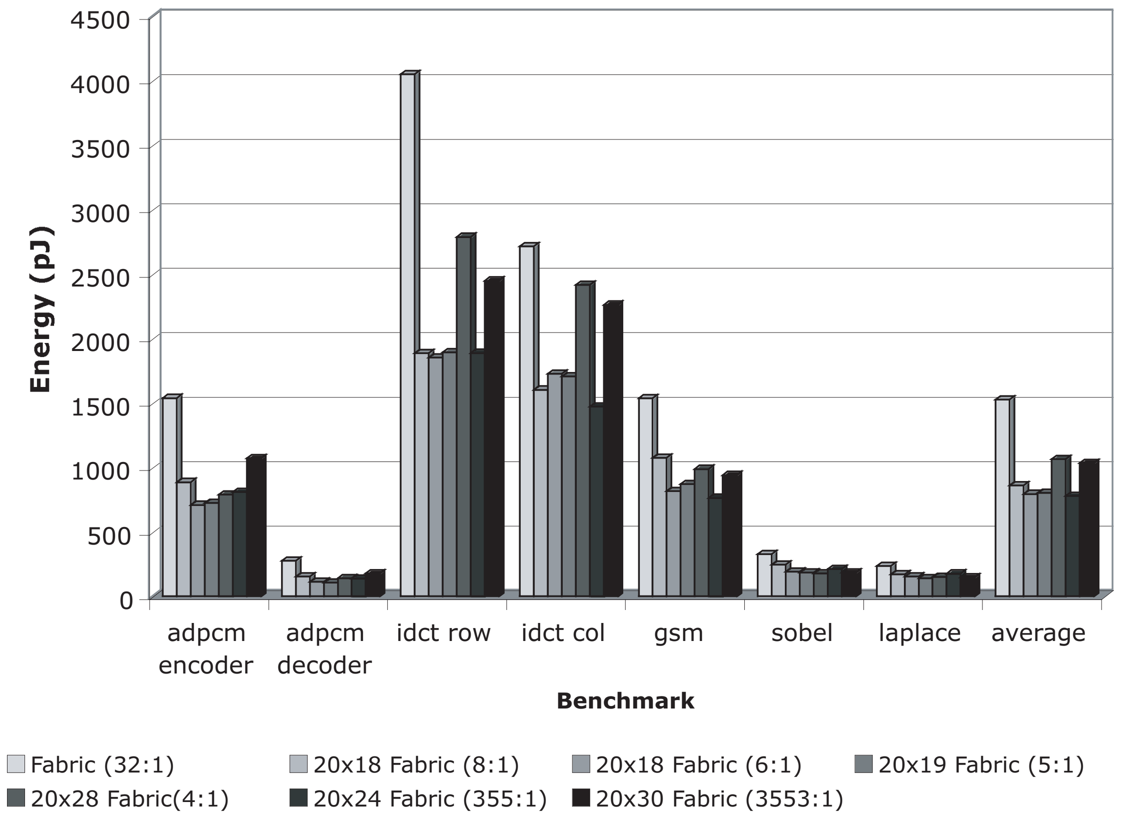


Figure 38: Impact of interconnect cardinality on energy consumption.

if possible. Figure 39 shows the energy results for mappings from this technique. In many cases the IP program can prove that the mapping is infeasible, and in these cases the heuristic solution is used as the best available mapping. All IP solutions were feasible for fully connected, 8:1, 6:1, and 5:1. Three of the benchmarks; idctrow, idctcol, and adpcm enc; were not feasible for 4:1 or 3-5-5:1. Sobel in addition to these other three were not feasible for 3-5-5-3:1. The results show a clear advantage to the 5:1 and 6:1 interconnects with slightly better results for 5:1. Unfortunately, these results cannot be obtained without improvement to the heuristic algorithm and running times for the IP mapping make the technique impractical.

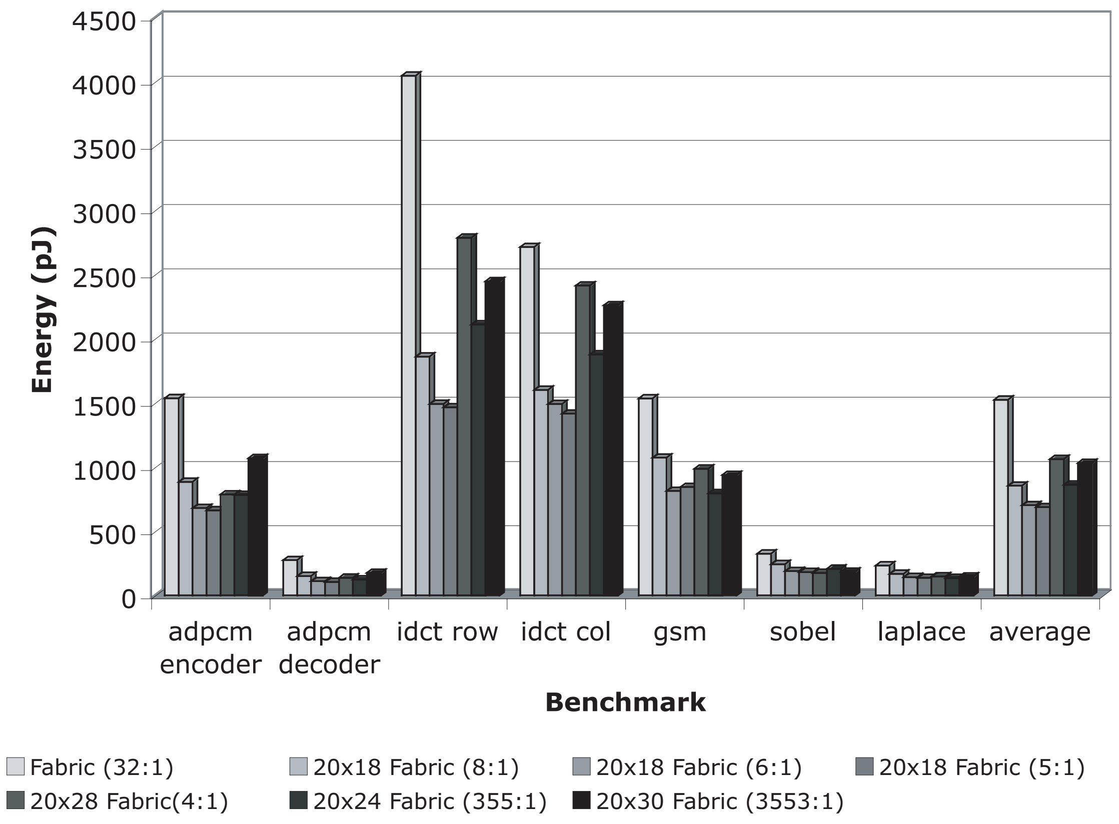


Figure 39: Potential for energy reduction by reducing interconnect cardinality.

6.2 DEDICATED VERTICAL ROUTES

Figure 40 provides a comparison of ALUs used in the graph, showing that more than 50% of the ALUs in the fabric will be used for routing by configuring the ALU as a pass-gate. Thus, some percentage of the ALUs in the fabric can be replaced with *dedicated* pass-gates to reduce complexity of the device. The top performers were chosen from Section 6.1, 5:1, 6:1, and 8:1 and varied the percentage of ALUs replaced with dedicated pass gates at levels of 25% (1 out of 4), 33% (1 out of 3), and 50% (1 out of 2) dedicated pass gates. While the study in Section 4.2.3 pointed to 25% stretching the 5:1 and 6:1 interconnect to its limits, the 8:1 interconnect has enough

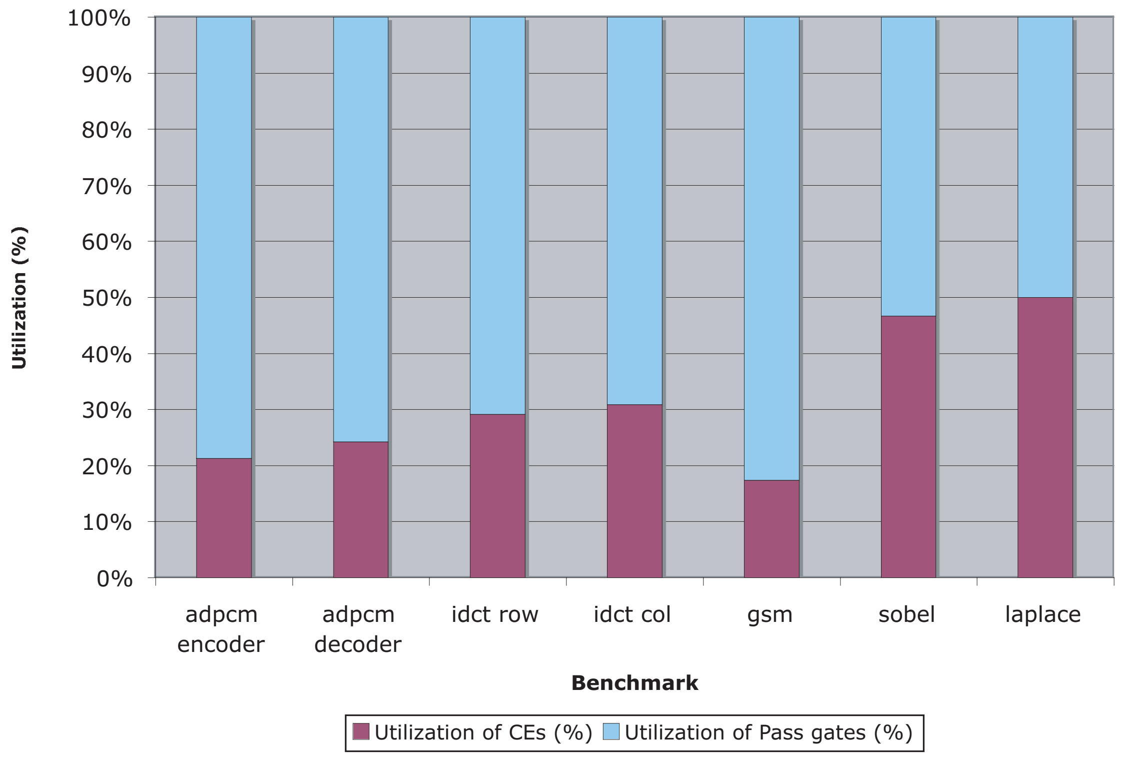


Figure 40: Comparison of ALUs used for routing and for computation.

flexibility and the benchmarks contain enough need for pass gates as seen by Figure 40 that 33% and 50% may be reasonable, particularly for sparse subgraphs.

Figure 41, Figure 42, and Figure 43 show energy results for varying each of the percentage of dedicated pass gates for the three chosen interconnects. The trend for 5:1 shown in Figure 41 shows that including dedicated pass gates does not help the overall energy improvement. In fact, when using 25% dedicated pass gates, the energy increases. When the level of dedicated pass gates is 50%, this tradeoff returns to a similar energy compared to no dedicated pass gates. The trend for 6:1 shown in Figure 42 is similar to 5:1 except that it returns to relatively even only at 33% dedicated pass gates. The trend for 8:1 is the best overall, dropping below the initial level with 25% dedicated pass gates and dropping further with 33%. The change between 33% and 50% is nominal. The fabric size requirements for including dedicated pass gates are shown in Table 8. The width number includes full ALUs and dedicated pass gates, so a width of 20 with 25% dedicated

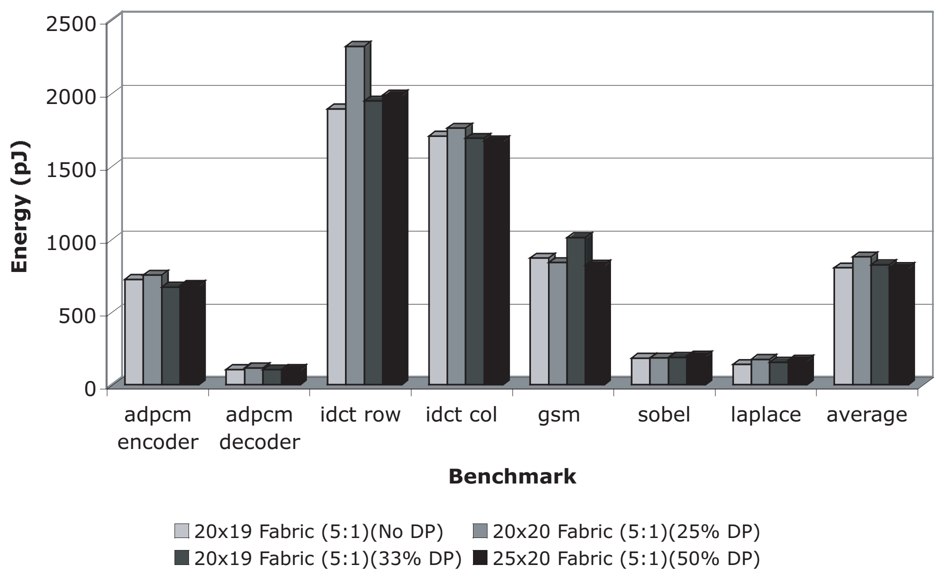


Figure 41: Energy trend when adding dedicated pass gates for 5:1 interconnect.

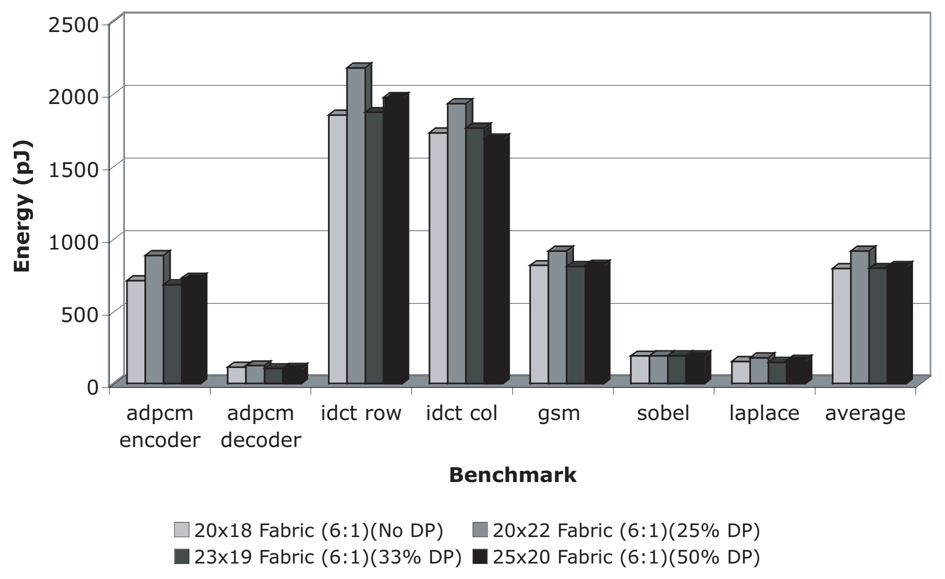


Figure 42: Energy trend when adding dedicated pass gates for 6:1 interconnect.

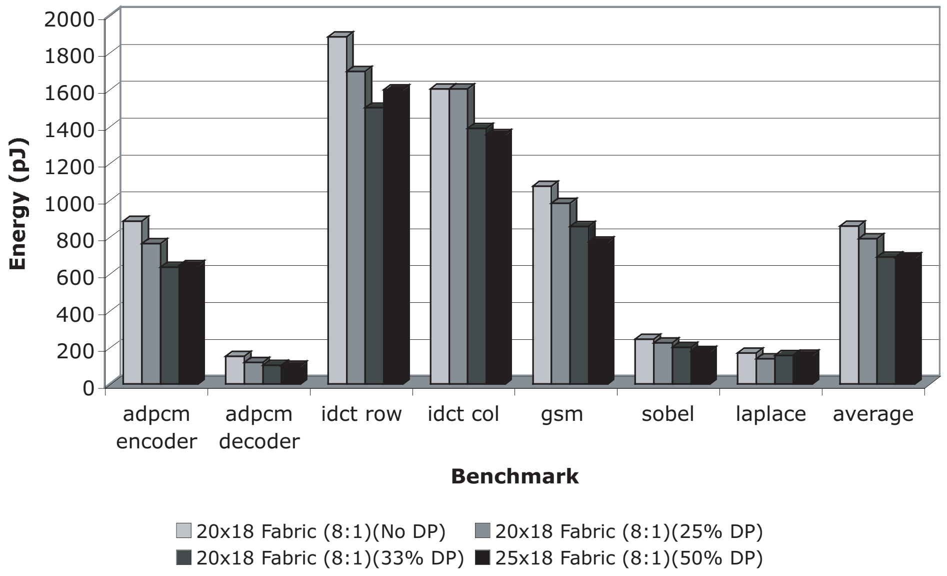


Figure 43: Energy trend when adding dedicated pass gates for 8:1 interconnect.

Table 8: Fabric size (Width x Height) for mapping various benchmarks onto different interconnects with different percentages of dedicated pass gates using the heuristic mapper.

	adpcm enc	**adpcm dec**	**idct row**	**idct col**	**gsm**	**sobel**	**laplace**
5:1 (0% DP)	20x18	16x13	20x8	17x13	20x19	20x19	10x9
5:1 (25% DP)	17x19	16x14	20x9	18x15	20x20	14x18	10x9
5:1 (33% DP)	20x18	20x14	19x10	20x14	20x19	20x18	11x9
5:1 (50% DP)	20x18	16x14	25x11	21x14	21x20	14x19	20x13
6:1 (0% DP)	17x18	16x14	15x9	17x13	20x18	14x18	10x9
6:1 (25% DP)	17x22	16x14	17x11	17x16	20x22	20x22	10x9
6:1 (33% DP)	20x18	16x13	20x9	20x15	23x19	14x19	11x9
6:1 (50% DP)	22x20	20x14	25x10	23x14	22x19	20x18	20x10
8:1 (0% DP)	17x16	16x13	15x8	17x10	20x12	14x18	10x9
8:1 (25% DP)	17x16	16x13	17x8	17x10	20x12	14x18	10x9
8:1 (33% DP)	17x16	16x13	19x8	17x11	20x12	14x18	11x9
8:1 (50% DP)	17x16	16x13	25x8	21x10	21x12	14x18	15x9

pass gates contains 15 full ALUs and 5 dedicated pass gates. While the energy consumed in the fabric is impacted by the height of the benchmark implementation, this does not tell the whole story and in these results sizes of the implementations do not always correlate well to the energy consumed in the fabric. One reason is that the size can not change while the length of non-critical paths can change a great deal. The energy consumed by each implementation is affected by the

Table 9: Number of ALUs used as pass gates for various interconnect strategies.

	adpcm enc	adpcm dec	idct row	idct col	gsm	sobel	laplace
5:1 (0% DP)	170	85	103	165	138	20	20
5:1 (25% DP)	110	56	86	104	80	8	15
5:1 (33% DP)	77	22	43	77	55	4	7
5:1 (50% DP)	57	28	36	50	48	5	3
6:1 (0% DP)	152	91	104	160	131	19	24
6:1 (25% DP)	143	50	74	70	84	7	12
6:1 (33% DP)	85	44	42	68	66	9	6
6:1 (50% DP)	44	28	47	20	43	3	8
8:1 (0% DP)	137	81	58	88	129	19	17
8:1 (25% DP)	78	44	22	56	82	6	6
8:1 (33% DP)	57	30	16	40	63	2	1
8:1 (50% DP)	26	11	8	12	32	1	2

number of ALUs used as vertical routes and the change in the overall path length (i.e. length of all output paths). We ran a two-way analysis of variance (ANOVA) on the energy with the number of ALUs used as pass gates and path length as factors to determine the correlation. Using an alpha value of 0.05, both factors significantly influenced the energy ($p < 0.01$ and $p = 0.031$, respectively).

Table 10: Increase in overall path length by addition of dedicated pass gates.

	adpcm enc	adpcm dec	idct row	idct col	gsm	sobel	laplace
5:1 (25% DP)	5	1	9	8	-1	0	2
5:1 (33% DP)	1	-1	9	1	-1	0	2
5:1 (50% DP)	-1	1	-4	9	1	2	3
6:1 (25% DP)	8	-1	31	18	0	0	2
6:1 (33% DP)	-1	-2	13	7	1	0	0
6:1 (50% DP)	3	0	16	1	0	1	1
8:1 (25% DP)	1	0	2	10	0	0	0
8:1 (33% DP)	1	0	4	9	0	0	1
8:1 (50% DP)	1	0	12	14	0	0	2

Reducing the number of ALUs used as pass gates reduces energy. We assume that the amount of energy consumed by a dedicated pass gate is negligible compared to an ALU configured to be a pass gate. Increasing the path length increases the energy. Table 9 contains the impact of adding dedicated pass gates on the number of ALUs used as pass gates and Table 10 shows the increase in path length over the 0% dedicated pass gate case for each interconnect cardinality.

As dedicated pass gates are added, the numbers of ALUs used as pass gates decreases for each cardinality. However, for 6:1 and 5:1 cardinalities where the energy initially increases when adding 25% and 33% dedicated pass gates, we can also see spikes in the path length that overcome the savings due to reducing the number of pass gates. For both cases it requires 50% dedicated pass gates for the path length increase to finally be mitigated by the savings in ALUs used as pass gates. However, the path length was not as dramatically increased when dedicated pass gates were included for 8:1, which is why the energy decreased overall.

To study the overall impact of adding dedicated pass gates, Figure 44 presents the best energy result using the heuristic mapper and dedicated pass gates for each interconnection cardinality. For example, where 8:1 uses 33% or 50% dedicated pass gates, the more restrictive interconnects such as 4:1 or 3-5-5:1 use no dedicated pass gates because of the inherent difficulty in mapping.

The results clearly indicate that the 8:1 based interconnects provide the best result as it is the lowest power for each of the benchmarks. However, the results for 6:1, 5:1, and 3-5-5:1 all seem to generate a similar result with each providing a better energy for different benchmarks and averaging out to be similar. When exploring the potential for energy reduction with the IP program, the energy savings possible from the various interconnect strategies becomes distinguishable as shown in Figure 45. 8:1 still provides the best result, however, 5:1 becomes the second choice. Based on the results from Figure 43, 8:1 interconnect with 33% dedicated pass gates was selected as the best energy performing fabric solution of those that were tested.

6.3 HETEROGENEOUS ALUS

Another approach to reduce the complexity of the ALUs is to reduce the number of operations each can support and spread the total number of required operations out over the stripe. Each stripe remains identical. However, when creating a heterogeneous stripe, it is important to distinguish what operations to include, and how often to include each operation. By analyzing representative appli-

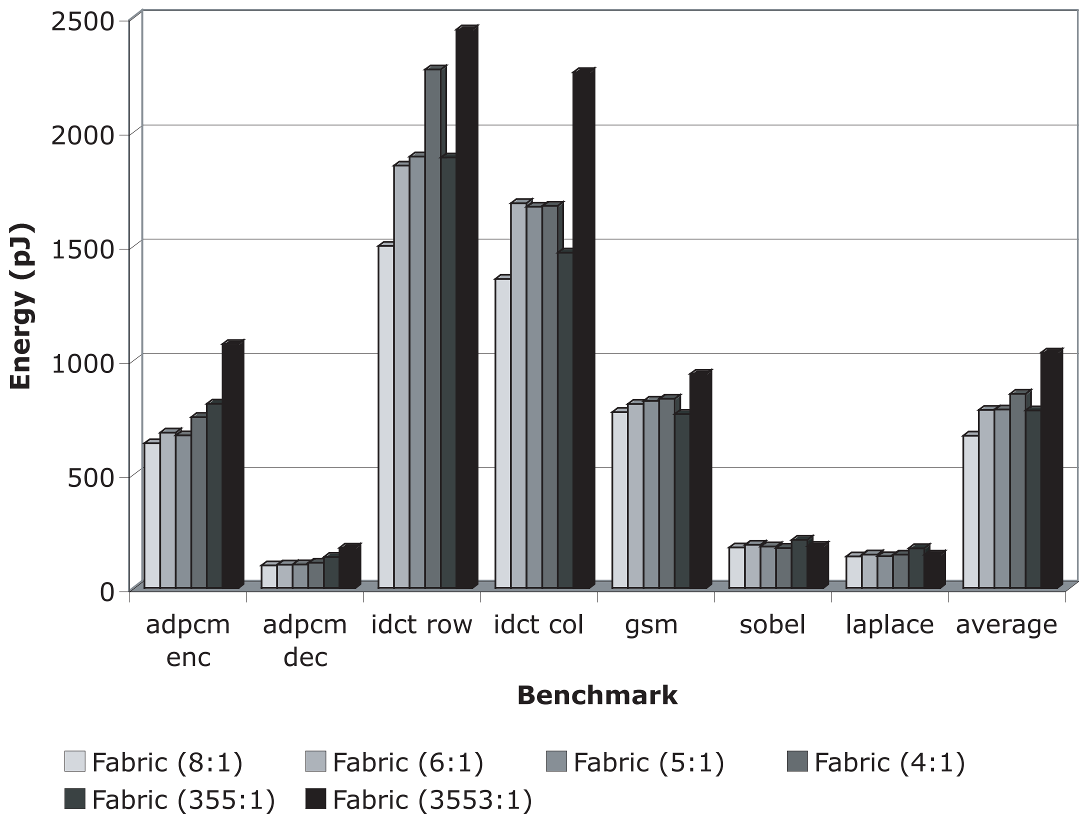


Figure 44: Comparison of best energy result between 0% and 50% dedicated pass gates with heuristic mapper.

cations, it is possible to extrapolate the needs of similar applications and create a heterogeneous fabric capable of supporting applications in the same computation class with minimal additional overheads. Each stripe is capable of supporting a fixed number of total operations. This total is determined by the number of ALUs in the row and the number of operations each ALU can support. A number of the total operations are consumed by NOOP and pass gate operations, which are supported by every ALU. The remaining operations' frequencies are calculated by first summing the frequencies of each operation across all of the benchmarks. An operation's distribution is made across the ALUs in the row based on its percentage of the total nodes. Thus, a frequently

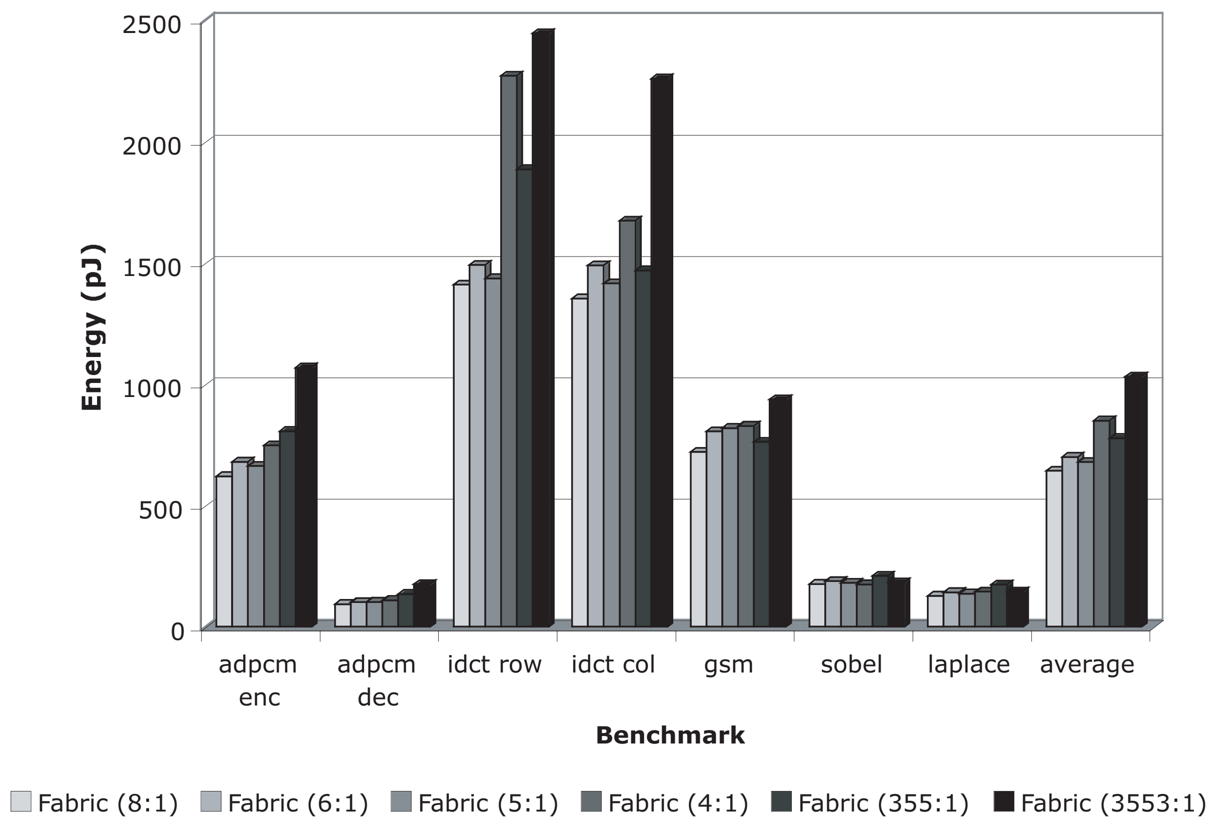


Figure 45: Potential best energy result using between 0% and 50% dedicated pass gates, mappings provided by the IP program where feasible.

used operation such as addition may take up a large percentage of the total possible operations in a row. However, the actual number of addition operations included in the fabric would be much lower. This is due to the restriction of allowing each ALU to contain only one instance of a given operation. This restriction ensures that any given operation can only occur *w* times in a given row, where *w* is the width of the fabric. By removing these extraneous operations, additional slots are made available to remaining operations. The impact of reducing the number of operations supported by the ALUs in the fabric on energy was studied. Figure 46 shows the energy results for varying the number of operations supported by the ALUs from 23 to 16, 10, and 8. As the number

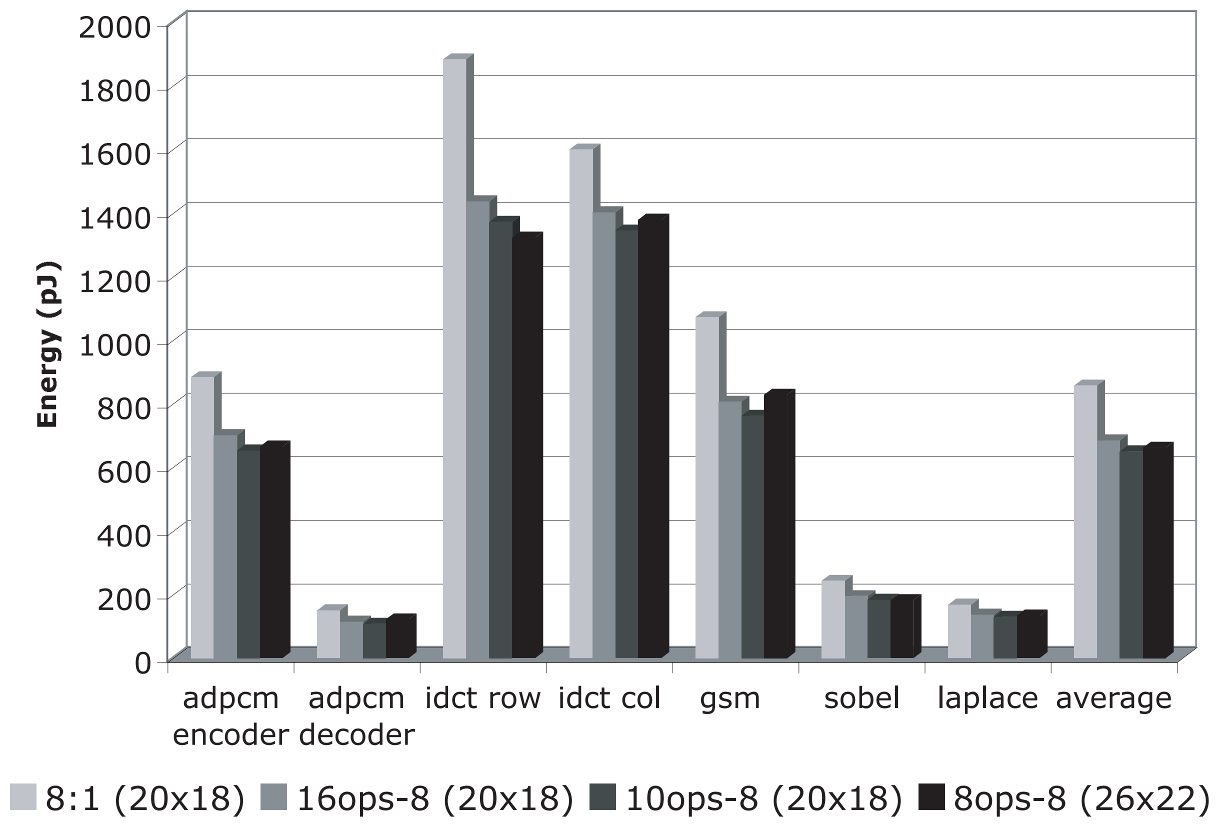


Figure 46: Impact of reducing the number of operations of ALUs on energy.

Table 11: Area results for various instances.

Fabric instances	**Area (μm^2)**	**Area with 33% DP**
23 ops	8362981	6057959
16 ops	4940466	5194891
10 ops	4190375	4463338
8 ops	5145491	no mapping

of operations reduces from 23 to 16, there is a significant decrease in energy consumption but as we go from 16 to 10, the decrease is nominal. As it can be seen that the fabric with 10-operations per ALU is the best candidate as it has the lowest energy for all the benchmarks. However, when we further reduced the number of operations to 8, this makes the mapping problem more difficult and we had to use a larger fabric to fit all of the benchmarks. This leads to increase in the energy consumption in several of the cases.

6.4 COMBINING HETEROGENEITY AND DEDICATED VERTICAL ROUTES

Based on our study where we varied the percentage of ALUs replaced with dedicated pass gates at levels of 25%, 33%, and 50%, the results clearly indicate that 8:1 interconnect with 33% dedicated pass gates requires the minimal energy. We selected our top performer from Section 6.3 and studied the combined effect of using dedicated vertical routes and reducing the number of operations supported by the ALUs. We tried the 16-operations ALU and 10-operations ALU with 8:1 interconnect cardinality and 33% dedicated pass gates. The 10-operations ALU with 33% dedicated pass gates configuration consumes the least energy among all the candidates considered here for all the benchmarks. The area results for different fabric instances is shown in Table 11. As it can be seen from the area results, the fabric with 10-operations and 8:1 interconnect consumes the least area.

6.5 ENERGY RESULTS FOR THE FABRIC DESIGNS IMPLEMENTED ON 130 NM IBM ASIC STANDARD CELLS

We selected the following fabric architectures based on our design space case studies when we moved to the new target technology: homogeneous ALUs with 8:1 interconnect, homogeneous ALUs with 8:1 interconnect and 33 % dedicated pass gates, 10 operations per ALU with 8:1 interconnect, and 10 operations per ALU with 8:1 interconnect and 33 % dedicated pass gates. The fabric VHDL automatically generated using the FIM file, was synthesized into an IBM cell-based ASIC design with a feature size of 0.13 μm using Synopsys Design Compiler. The post-synthesis design was simulated in Mentor Graphics ModelSim to calculate the delay of each design and these simulations were used as stimulus to the Synopsys PrimeTime PX tool to estimate the

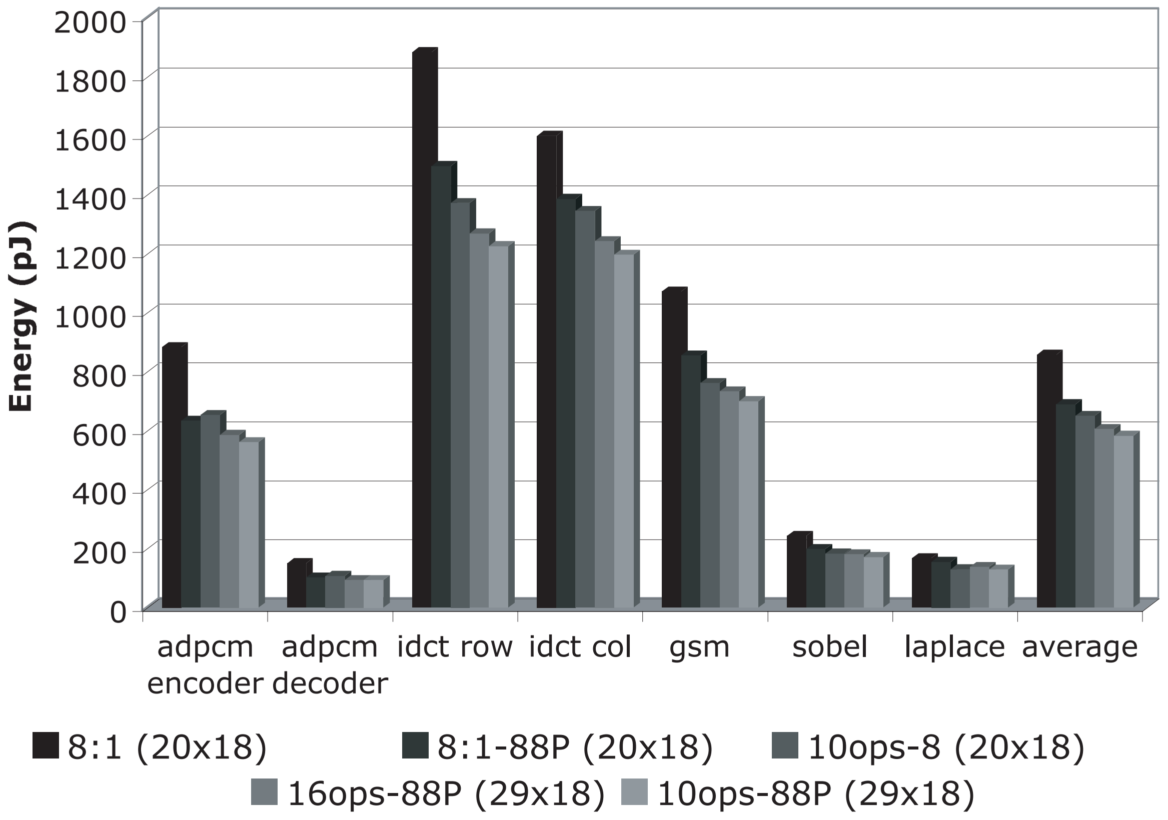


Figure 47: Combining dedicated pass gates with heterogeneous functional units.

power consumption of the device. Energy was calculated by computing the product of the power and delay of the design. Figure 48 shows energy comparison of homogeneous fabric with 8:1 multiplexer, homogeneous fabric with 8:1 multiplexer and 33% dedicated pass gates, 10 operations per ALU with 8:1 multiplexer and 10 operations per ALU with 8:1 multiplexer and 33 % dedicated pass gates. The fabric with 10 ops per ALU and 33% dedicated pass gates consumes the least energy and is the best candidate from our design space case studies.

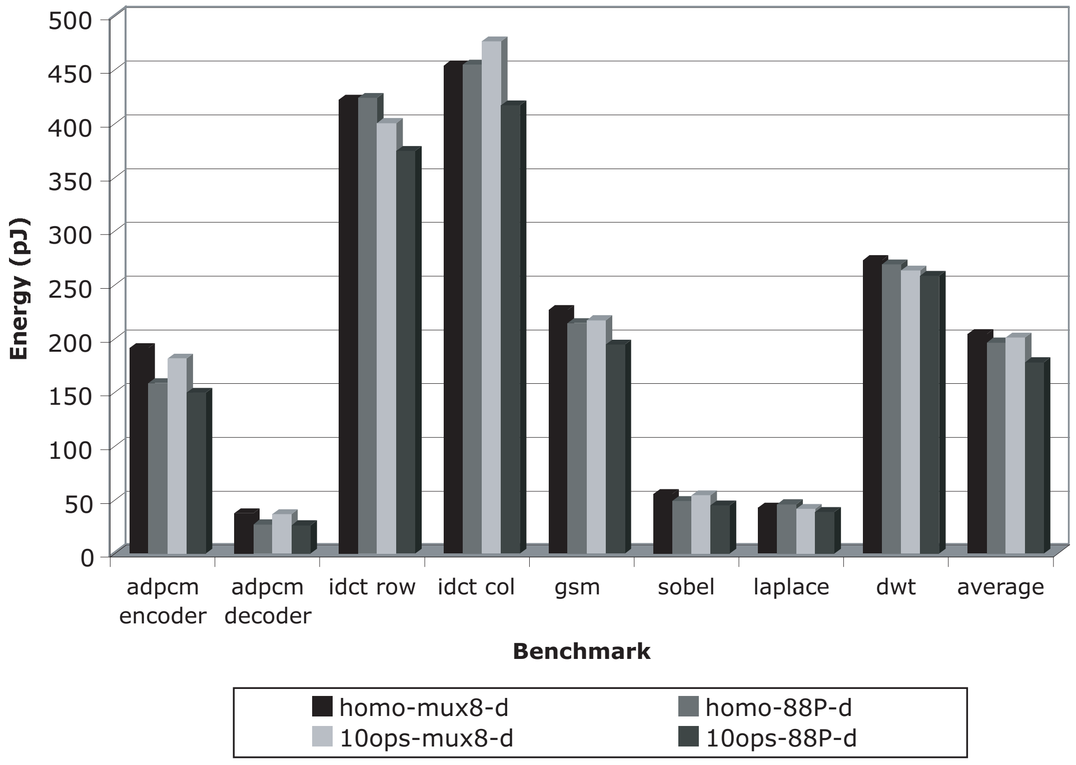


Figure 48: Energy comparison of fabric designs implemented on 130 nm IBM ASIC standard cells

7.0 DESIGN SPACE EXPLORATION TOOL FOR DOMAIN-SPECIFIC FABRIC

Exploring the design space manually would be very time consuming and may not even be feasible for large system-on-chip designs. Design space exploration tools can allow application developers to explore architectural tradeoffs efficiently and reach solutions quickly. To make these design exploration tools work, however, it is important to consider the applications of interest. When reconfigurable computing is considered as a design option, the application domain (a suite of applications to be run on the device) is often well known in advance. A general purpose design may make poor power / performance tradeoffs for that application domain, while a design fine-tuned for a single application may be inflexible. However, a reconfigurable design for an application domain can provide some of the flexibility of general purpose reconfigurable computing along with some of the optimality that can result from application specific design. This task is somewhat more challenging than application specific design, however, because the needs of different applications must be carefully balanced to achieve the targeted design goals.

In this dissertation research, a number of design space case studies were developed . As an initial application domain for our case studies, some of the core signal processing benchmarks from the MediaBench benchmark suite and some edge-detection benchmarks from the image processing domain were selected. In order to explore the architectural space, an energy-efficient, parameterized, coarse-grained reconfigurable fabric model was designed. Using this model, the impact of varying different design parameters onto physical characteristics of the device was studied. A design space exploration tool was developed to automate the design space case studies that were done manually. The tool generates a tailored architectural instance based on the needs of the applications to reduce power and area, and improve performance for a given suite of applications. The design space exploration flow is similar to the Application Specific Instruction Set Processor (ASIP) design automation flows. ASIP flows read in one or more programs into the tool and attempt to identify custom instructions which when added to the processor instruction set architecture could improve an execution metric such as power or performance. Similarly, the design space exploration flow shown in Figure 49 takes information from the applications and produces statistics from different instances of the fabric model. These statistics are used to make decisions and evaluate tradeoffs to converge on a fabric model tailored to the applications it executes.

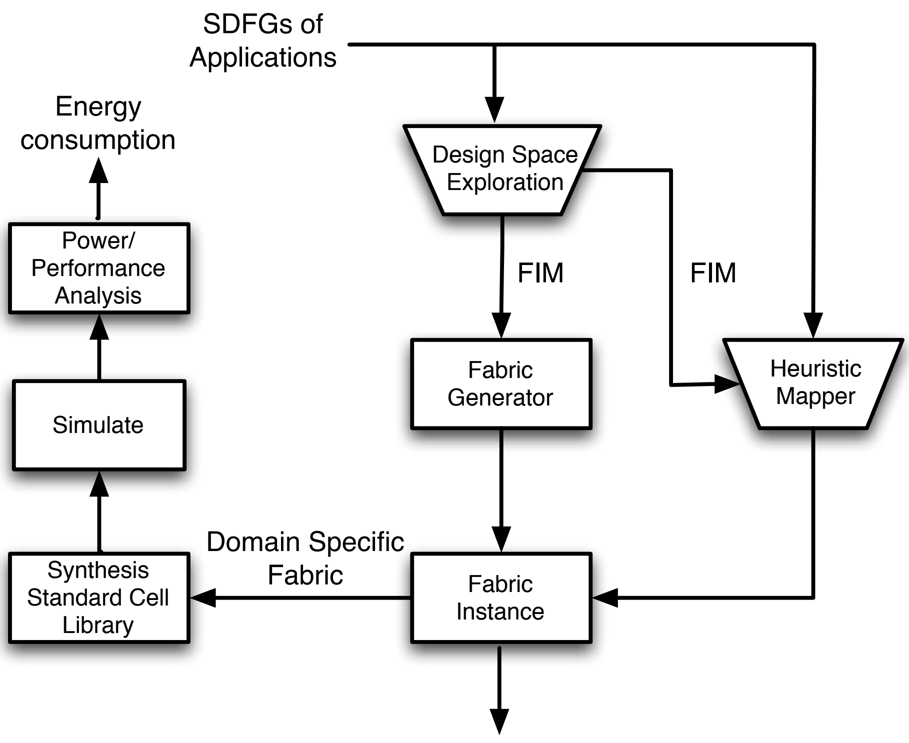


Figure 49: Design space exploration flow for the domain specific fabric.

Using the SuperCISC compilation flow [30], computational kernels were extracted for these signal and image processing applications and converted into super data flow graphs (SDFGs), which were used as the benchmark circuits. SDFGs were mapped onto the fabric with fully interconnected stripes (e.g any ALU from a previous stripe can be connected to any ALU in the current stripe) using as soon as possible (ASAP) schdeduling. SDFGs were then examined to get information about number of functional units, type of functional units, granularity of the functional units and the interconnect, and fan-in and fan-out of the functional units. Several statistics like size of the fabric, utilization of the fabric in terms of the number of computational elements and pass gates, multiplexer cardinality usage, granularity of the functional units and the interconnect were determined. The tool generates a fabric instance model (FIM) described in detail in Chapter 3. The FIM is a textual representation to describe the interconnect and the layout and make-up of the ALUs in the device. The SDFGs are then mapped to the fabric architecture described by the FIM using heuristic mapper. Even though the heuristic mapper is not the best mapping algorithm but it was chosen because of its fast execution time.

Since energy results are generated from extremely time consuming power simulations using computer-aided design tools, conducting power simulations for each possible fabric architecture in the design space would be impractical. From our manual design space case studies, we examined that the two factors that affect the energy consumption of the device are: (1) the increase in the total path length of the mapped application onto the device, and (2) the number of ALUs used as pass-gates. The total path length in the mapped design is the sum of the number of rows traversed from each input to each output. The number of ALUs used as pass-gates is useful in judging success in cases where the fabric contains dedicated pass-gates. Dedicated pass-gates are more energy efficient than complex functional units at passing a value (more than an order of magnitude described in Chapter 4). Thus, when using dedicated-pass gates, the fewer ALUs are used as pass-gates. To demonstrate that these factors influence the energy consumption of the device, we ran a two-way analysis of variance (ANOVA) on the energy with the number of ALUs used as pass-gates and path length as factors to determine the correlation. Using an alpha value of 0.05, both factors significantly influenced the energy ($p < 0.01$ and $p = 0.031$, respectively) described in Chapter 6. We used the average path length increase (average pli) as a metric in our design space exploration algorithm. We used only one metric in our algorithm for simplicity. This allows us to explore a wide range of architectures in reasonable amount of time without doing power simulations for each case. We performed power simulations only on the final fabric architecture picked by the tool to compare its energy consumption with the one picked by our manual design space case studies.

ASAP provides ideal and total path length for a suite of benchmarks. Based on our manual design space case studies, the threshold for average pli was selected for a set of fabric architectures examined here for a suite of applications. The design goal here is to determine an energy efficient fabric for a given domain of applications by minimizing the multiplexer cardinality, minimizing the number of operations supported per ALU, and maximizing the number of dedicated pass gates.

The design space exploration algorithm is described in Algorithm 1:

Algorithm 1 Design Space Exploration

```
1: while average $pli$ < threshold do
2:     Reduce multiplexer cardinality $(C)$ to next power of $2^n$+1 where $n = 5, 4, ..., 1$.
3:     Map applications to the fabric using heuristic mapper.
4:     Determine average path length increase.
5: end while
6: Revert to last $C$ where average $pli$ ¡ threshold.
7: while average $pli$ < threshold do
8:     Increase the number of dedicated pass gates $(D)$ where $D = 0\%, 25\%, 33\%, 50\%, ..., 75\%$.
9:     Map applications to the fabric using heuristic mapper.
10:    Determine average path length increase.
11: end while
12: Revert to last percentage of $D$ where average $pli$ ¡ threshold.
13: while average $pli$ < threshold do
14:    Reduce the number of operations supported by each ALU $(O)$ by one.
15:    Map applications to the fabric using heuristic mapper.
16:    Determine average path length increase.
17: end while
18: Revert to last number of ALU operations $O$ where average $pli$ ¡ threshold.
```

Figure 50 shows the results obtained from the design space exploration tool for various fabric architectures. Based on our manual design space case studies, the threshold value for the average path length increase was set to 2. As the multiplexer cardinality was reduced from 33:1 to 32:1,17:1,16:1, and 8:1, the average path increase stayed at zero. As it reached 5:1 interconnect, the average path length increase went up to 9.4 because we were restricting the connectivity of each ALU and making the mapping problem more difficult. Since it exceeded the threshold limit for a 5:1 interconnect, the tool picked an 8:1 interconnect and then started changing the percentage of dedicated pass gates from 0% to 25%, 33%, 50% for that interconnect. Now it picked an 8:1 interconnect with 33% dedicated pass gates based on the average path length increase and then it started reducing the number of operations supported per ALU for that architecture. As it started

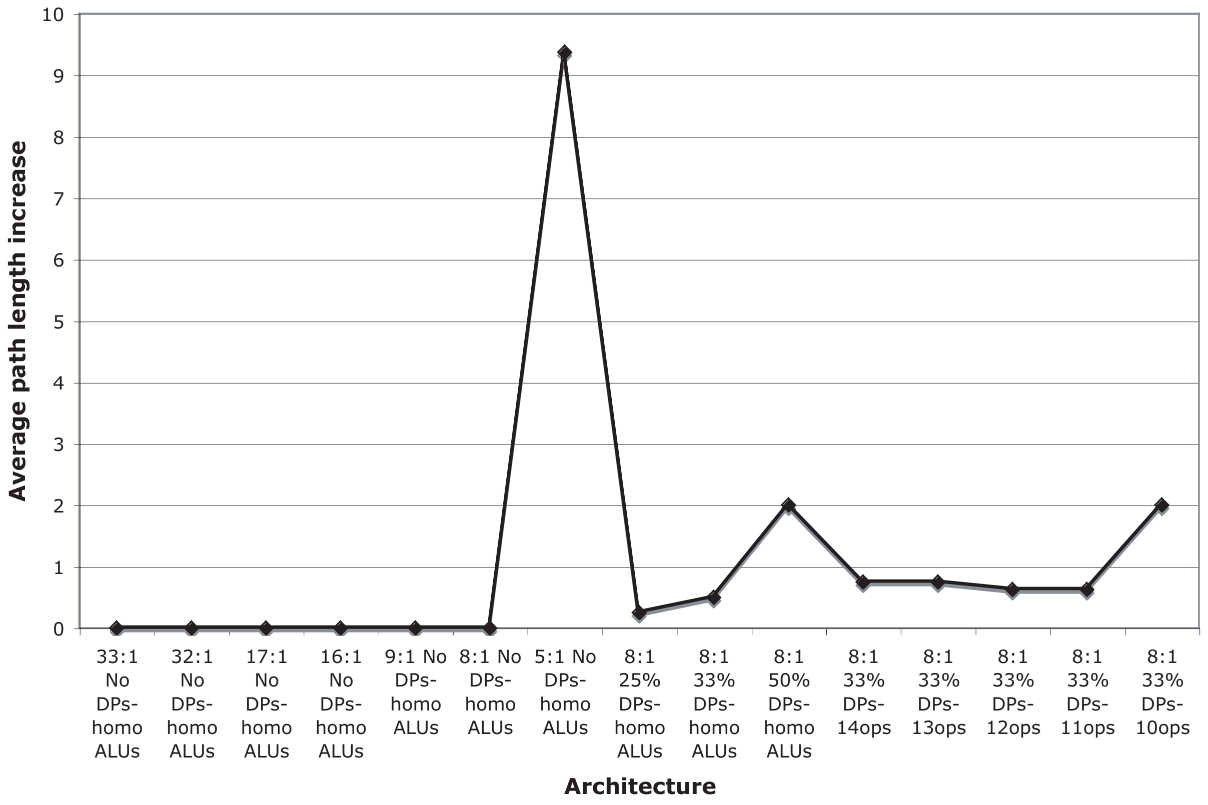


Figure 50: Results from the design space exploration tool for various architectures for threshold value of the average path length increase set to 2.

reducing the number of ALU operations, the mapping problem became even more challenging because each ALU could support only certain number and types of operations. It picked an 8:1 interconnect with 33% dedicated pass gates and 11 operations per ALU as the best candidate that could meet the target design goals. The manual design space exploration case studies picked the fabric with 10 operations per ALU, 8:1 interconnect with 33% dedicated pass gates as the best candidate in terms of energy consumption. Figure 51 shows the number of ALUs used as pass gates for various architectures explored for threshold value of average pli to be 2. As the number of dedicated pass gates increases, lesser number of ALUs are used as pass gates.

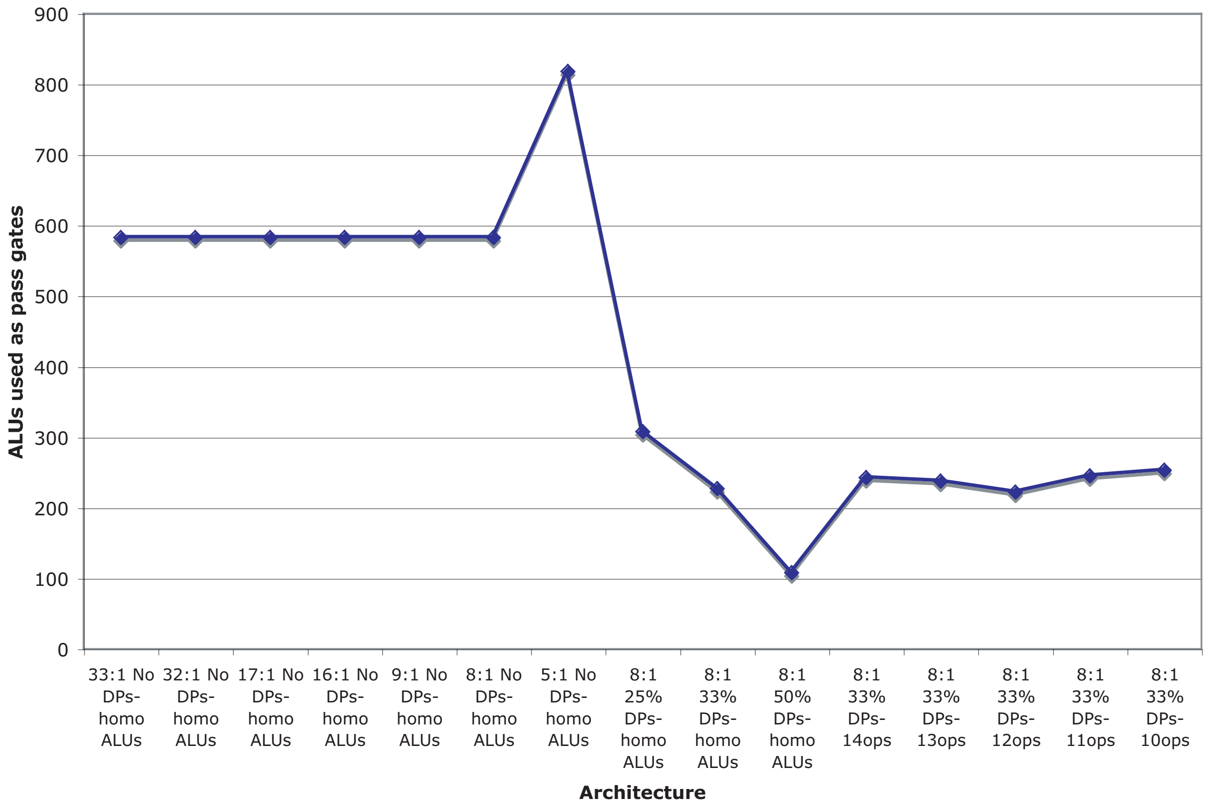


Figure 51: ALUs used as pass gates for various fabric architectures.

Figure 52 shows the results obtained from the design space exploration tool for various fabric architectures for a threshold value for the average path length increase to be 3. The initial design space exploration was same as explained in the above experiment. After the tool picked an 8:1 interconnect, it started changing the percentage of dedicated pass gates from 0% to 25%, 33%, 50%, 66% for that interconnect. Now it picked an 8:1 interconnect with 50% dedicated pass gates based on the average path length increase and then it started reducing the number of operations per ALU for that architecture. As it started reducing the number of operations supported by each ALU, this made the mapping problem even more difficult because each ALU can support only certain number and types of operations. It picked an 8:1 interconnect with 50% dedicated pass

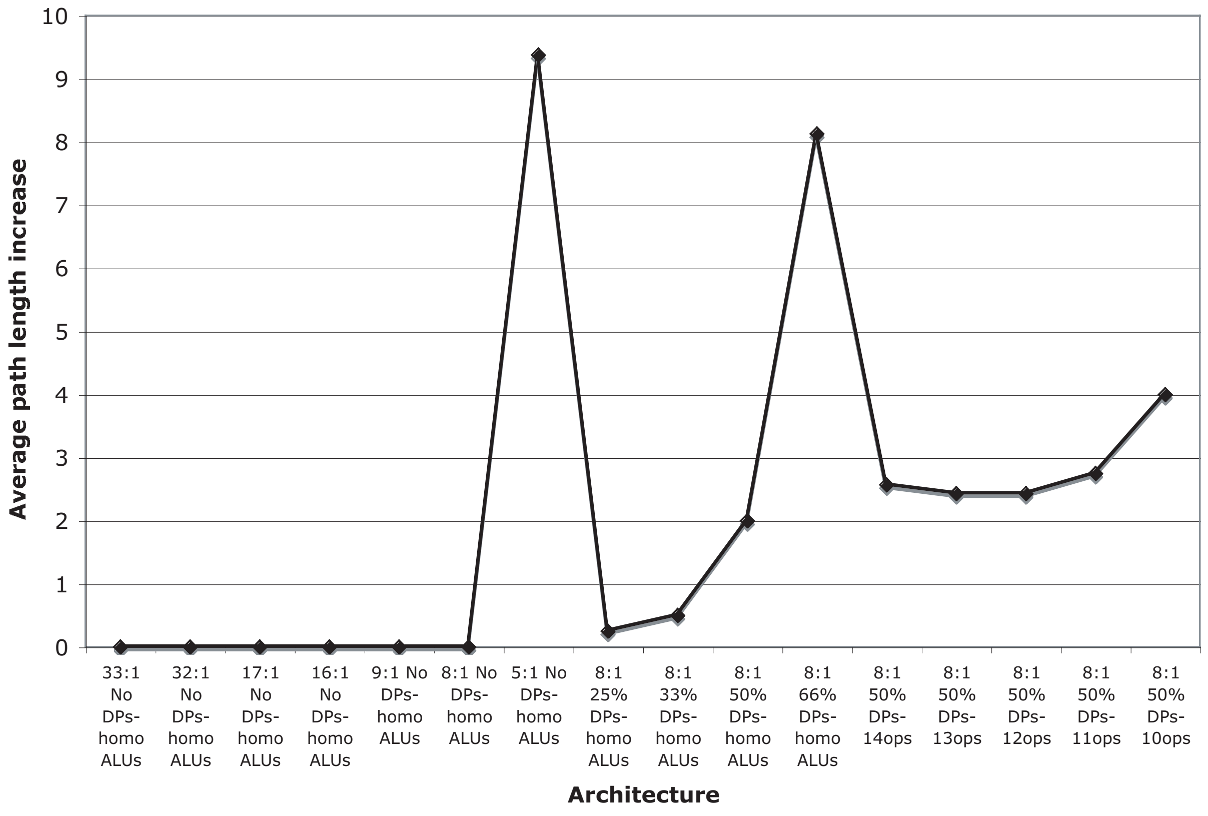


Figure 52: Results from the design space exploration tool for various architectures for threshold value of the average path length increase to be 3.

gates and 11 operations per ALU as the best candidate that could meet the target design goals. Figure 53 shows the number of ALUs used as pass gates for various architectures for threshold value of average pli to be 3. The best candidate picked by the tool uses lesser number of ALUs as pass gates as compared to the best architectural solution from the manual design space exploration.

Figure 54 compares the energy consumption of the architecture picked from manual design space case studies with the architectures picked by the design space exploration tool for threshold values of the average path length increase to be 2 and 3. 11 operations per ALU, 8:1 interconnect

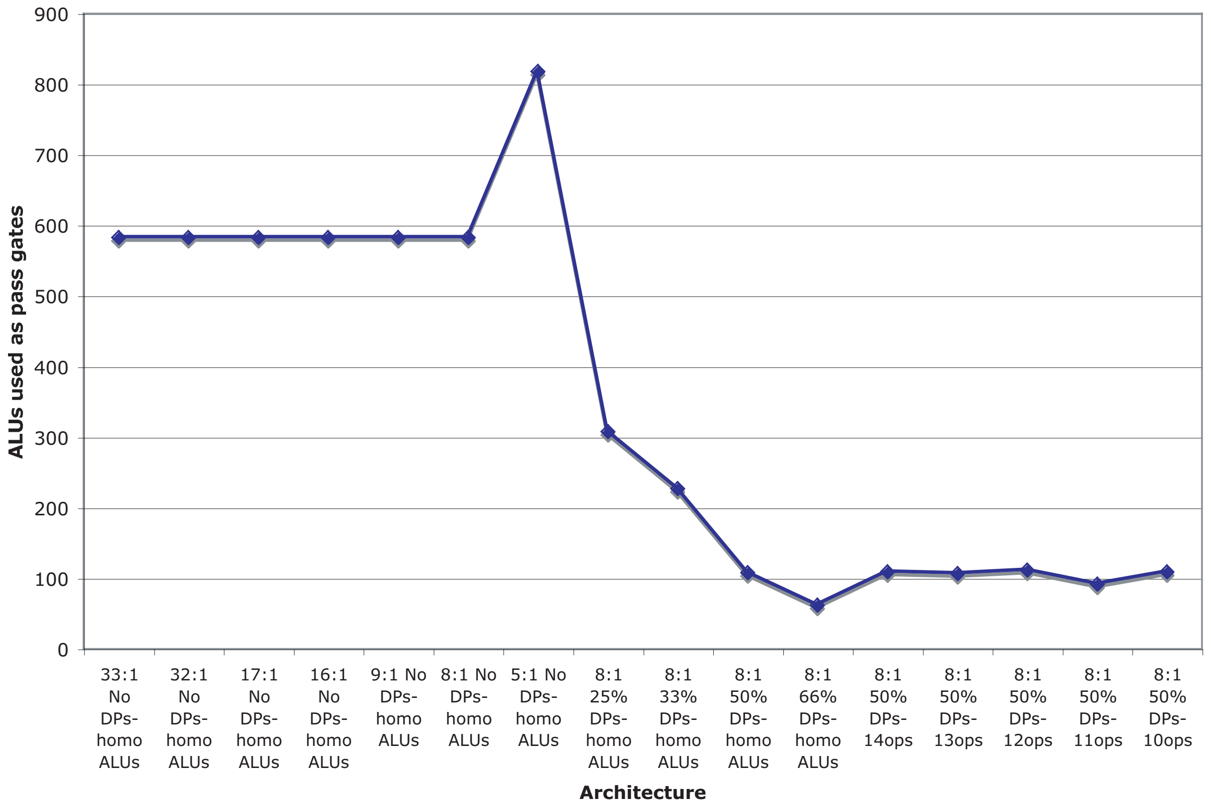


Figure 53: ALUs used as pass gates for various fabric architectures.

with 50% dedicated pass gates consumes the least energy. It consumes approximately 9% less energy on an average as compared to the best candidate from the manual design space case studies.

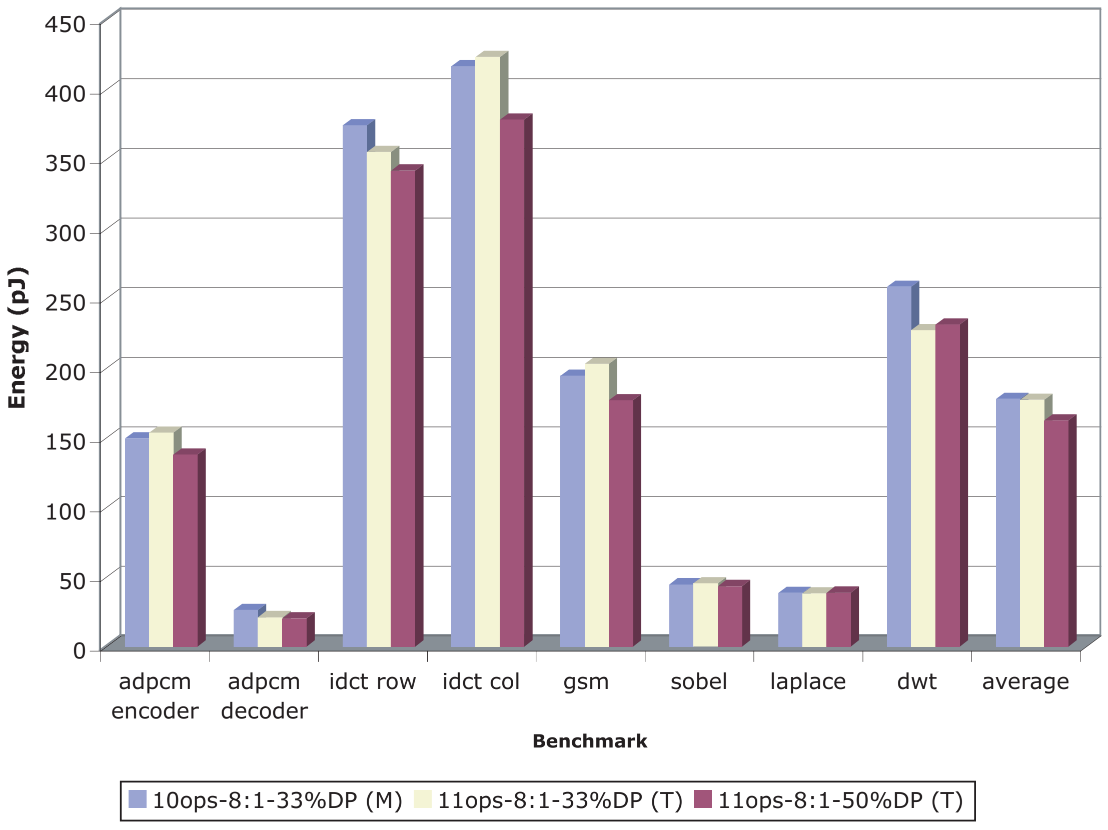


Figure 54: Energy comparison between the manual architectural solution and the architectural solutions generated by the tool.

8.0 CONCLUSIONS AND FUTURE WORK

8.1 CONCLUSIONS

In this dissertation research, an energy-efficient, parameterized, reconfigurable fabric model was designed for Digital Signal Processing (DSP) style applications. Using this model, the impact of varying different architectural design parameters on power and performance was studied. The impact of varying the cardinality (number of inputs/fanin of multiplexers) and the orientation (overlap of multiplexers for different operands) of multiplexers was considered. The interconnect strategies considered in this dissertation research include baseline interconnect architectures built from fully connected rows, 8:1, and 4:1 multiplexers. 5:1 and 6:1 multiplexing strategies , developed from mirroring 4:1 multiplexers connected to the functional unit input operands were also explored. In order to further simplify the device the possibility of using a heterogeneous interconnect to reduce the multiplexing cardinality at some locations was explored. 33% and 50% of the 5:1 interconnect were replaced with 3:1 multiplexers, built from mirrored 2:1 multiplexers.

The impact of heterogeneity in the functional unit design was explored by introducing non-uniform arithmetic and logic units in a stripe. The benefit of adding dedicated pass-gates to prevent functional units from being used as routing was examined. The incorporation of dedicated pass-gates requires potential tradeoffs in the cardinality and orientation of the multiplexers in the interconnect. Another approach used in this research to reduce the complexity of the ALUs was to reduce the number of operations each ALU can support and spread the total number of required operations out over the stripe. While the ALUs within the stripe are heterogeneous, each stripe in the fabric remains identical. Different optimization techniques like local search and simulated annealing were used to study and optimize the interconnect. Simulated Annealing was also used to solve the problem of Feasible Mapping with Fixed Rows.

Exploring the design space manually is time consuming and may not even be feasible for the most complex designs. A design space exploration tool was developed to automate the architectural exploration case studies. The tool generates a tailored architectural instance based on the needs of the applications to reduce power and area, and improve performance for a given suite of applications. Super data flow graphs of the applications were examined to get information like number of functional units, type of functional units, granularity of the functional units and interconnect, and fan-in and fan-out of the different functional units. Several statistics like width and height of the fabric, utilization of the fabric in terms of the number of computational elements and pass gates, multiplexer cardinality usage, granularity of the functional units and the interconnect were determined. It determines the best architectural solution by minimizing the multiplexer cardinality and the number of operations supported per ALU, and maximizing the number of dedicated pass gates.

Power and performance analysis was done by implementing a set of core signal processing benchmarks from the MediaBench benchmark suite and some edge-detection benchmarks from the image processing domain onto the fabric. The fabric was synthesized on 160 nm cell-based ASIC fabrication process from OKI.The power-performance of the benchmarks implemented onto the fabric was compared with other hardware and software implementations. The optimized fabric energy consumption shown in Figure 55 is within 5X of a direct ASIC implementation, is 95X better than a Virtex-II Pro FPGA and is 575X better than an Intel XScale processor. The optimized fabric was also implemented on 130 nm cell-based ASIC fabrication process from IBM. The optimized fabric yields energy within 3X of a direct ASIC implementation, 330X better than a Virtex-II Pro FPGA and 2016X better than an Intel XScale processor.

8.2 FUTURE WORK

Some of the future directions of this dissertation include (1) implementing the fabric design onto 90 nm and below target technologies, (2) designing fabric architectures to map encryption algorithms having medium to large size look-up tables, and (3) applying better control optimization algorithms to the design space exploration tool.

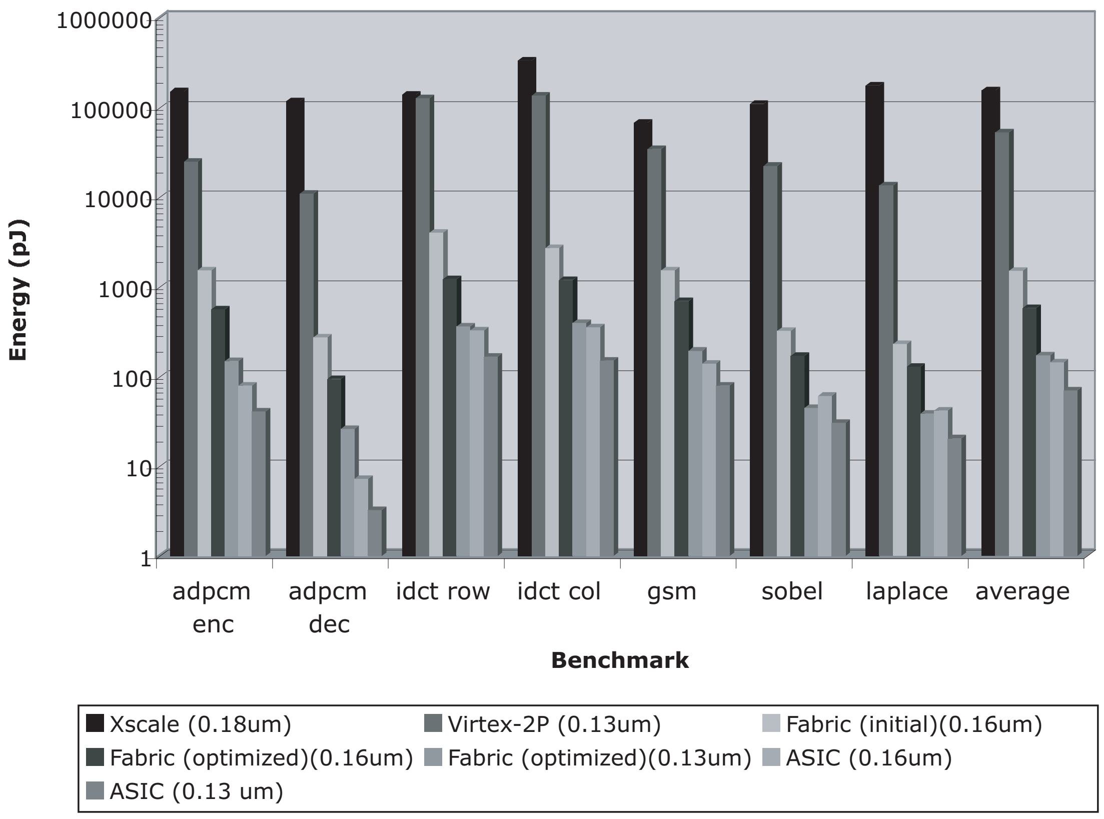


Figure 55: Fabric compared with other hardware and software implementations.

8.2.1 Implementing the fabric design to 90 nm and below target technologies

Leakage power dissipation has become one of the critical design concerns for the semiconductor industry at 90 nm and below [51, 52]. Leakage power consumption is the power consumed by the sub-threshold currents and by reverse biased diodes in a CMOS transistor. Power gating, one of the well known techniques, effective for reducing leakage power, can be applied to the fabric design. A sleep transistor is added between actual ground rail and circuit ground also called virtual ground [53, 54]. The leakage path is cut off by putting this transistor in the sleep mode. It has already been shown that this technique provides a substantial reduction in leakage power at a minimal impact on performance [55, 56, 57].

8.2.2 Designing fabric architectures to map encryption algorithms

Fabric architectures can be designed to map encryption algorithms like AES (Advanced Encryption Standard). These algorithms have read only arrays of non-changing data. Data can be pre-programmed into static random access memories (SRAMs). Depending on the needs of the applications, the size of the SRAM cells can be determined. One approach to implement large size look-up tables is by grouping together smaller memory cells. For example, 1-Kbit SRAM cell is approximately the same size as a 32-bit multiplier. 1-Kbit SRAM cell can be added to each ALU and eight of these can be grouped together to make an 8-Kbits memory. A segmented bus architecture shown in Figure 56 can be used to allow loads to be specified from any location in the group. The reason for choosing a segmented bus architecture is that it shows potential for improving both performance and power related features of a device.

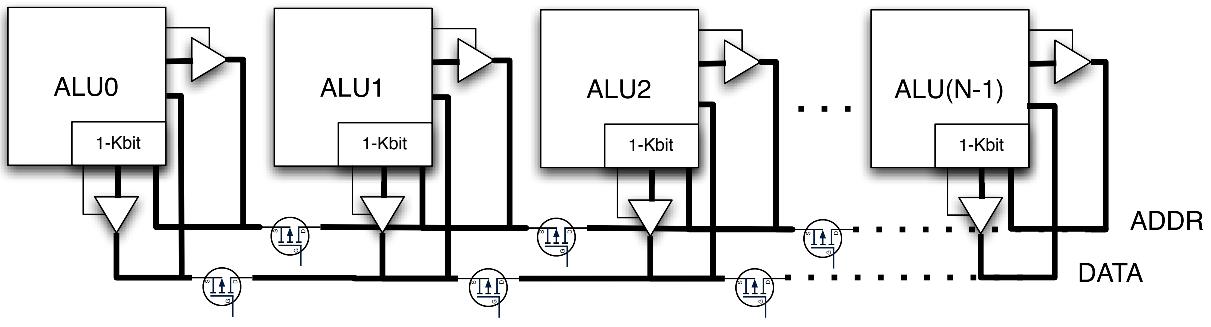


Figure 56: Fabric design with memories using segmented bus architecture.

8.2.3 Improvements in the design space exploration tool

Currently, only one parameter is varied at a time keeping others constant and the parameters of the search space exploration algorithm are explored in a specific order based on the manual design space studies. This might not lead us to the best candidate for the target goals. The search space can be explored in a more efficient way by varying all the parameters at the same time or by changing the order in which the parameters are explored for the manual architectural exploration.

Better optimization algorithms can also be applied to improve the quality of the solution obtained by the tool. The decision making process can be improved by assigning weights to various metrics of interest instead of using threshold values for the metrics.

BIBLIOGRAPHY

[1] L. Sheng, A. S. Kaviani, and K. Bathala, “Dynamic power consumption in virtex-II FPGA family,” in *FPGA*, 2002.

[2] B. Levine and H. Schmit, “Piperench: Power & performance evaluation of a programmable pipelined datapath,” presented at Hot Chips 14, Palo Alto, CA, August 2002.

[3] H. Schmit, D. Whelihan, A. Tsai, M. Moe, B. Levine, and R. R. Taylor, “Piperench: A virtualized programmable datapath in 0.18 micron technolog,” in *Proceedings of the IEEE Custom Integrated Circuits Conference*, 2002.

[4] E. Mirsky and A. Dehon, “Matrix: A reconfigurable computing architecture with configurable instruction distribution and deployable resources,” in *in Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, April 1996.

[5] J. R. Hauser and J. Wawrzynek, “Garp: A MIPS processor with a reconfigurable coprocessor,” in *IEEE Symposium on FPGAs for Custom Computing Machines*, K. L. Pocek and J. Arnold, Eds. Los Alamitos, CA: IEEE Computer Society Press, 1997, pp. 12–21. [Online]. Available: citeseer.nj.nec.com/hauser97garp.html

[6] S. Hauck, T. W. Fry, M. M. Hosler, and J. P. Kao, “The chimaera reconfigurable functional unit,” in *IEEE Symposium on FPGAs for Custom Computing Machines(FCCM)*, 1997, pp. 87–96.

[7] e. a. H. Singh, “Morphosys: An integrated re-configurable architecture,” in *Proc. of the NATO RTO Symposium on System Concepts and Integration*, 1998.

[8] C. Ebeling, D. C. Cronquist, and P. Franklin, “Rapid - reconfigurable pipelined datapath,” in *in the 6th International Workshop on Field-Programmable Logic and Applications*, 1996.

[9] C. Ebeling *et al.*, “Mapping applications to the rapid configurable architecture,” in *Proc. of the IEEE Symposium on FPGAs for Custom Computing Machines*, 1997.

[10] A. Kaviani, D. Vranesic, and S. Brown, “Computational field programmable architecture,” in *Proc. of the IEEE Custom Integrated Circuits Conference*, 1998.

[11] A. A. Aggarwal and D. M. Lewis, “Routing architectures for hierarchical field programmable gate arrays,” in *Proc. of the IEEE International Conference on Computer Design*, 1994.

[12] Elixent, "The reconfigurable algorithm processor," `http://www.elixent.com/`.

[13] PACT-XPP, "Xpp-lib core overview," `http://www.pactcorp.com/`.

[14] MathStar, "Field programmable object array architecture," `http://www.mathstar.com/literature.html`.

[15] A. K. Jones, R. Hoare, D. Kusic, J. Fazekas, and J. Foster, "An FPGA-based VLIW processor with custom hardware execution," in *ACM International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2005, pp. 107–117.

[16] Rapport, Inc, "Kilocore," website, `http://www.rapportincorporated.com`.

[17] M. J. Wirthlin and B. L. Hutchings, "A dynamic instruction set computer," in *Proc. of FCCM*, 1995, pp. 99–107.

[18] J. Cong, Y. Fan, G. Han, and Z. Zhang, "Application-specific instruction generation for configurable processor architectures," in *Proc. of the International Symposium on Field Programmable Gate Arrays (ISFPGA)*. New York, NY, USA: ACM, 2004, pp. 183–189.

[19] M. K. Jain, M. Balakrishnan, and A. Kumar, "Asip design methodologies: Survey and issues," in *International Conference on VLSI Design*, 2001.

[20] R. E. Gonzalez, "Xtensa – a configurable and extensible processor," *IEEE Micro*, vol. 20, no. 2, pp. 60–70, 2000.

[21] Z. Shen, H. He, Y. Zhang, and Y. Sun, "A video specific instruction set architecture for asip design," *VLSI Design*, vol. 2007, no. 2, pp. 1–7, 2007.

[22] L. Fanucci, M. Cassiano, S. Saponara, D. Kammler, E. M. Witte, O. Schliebusch, G. Ascheid, R. Leupers, and H. Meyr, "Asip design and synthesis for non linear filtering in image processing," in *Proceedings of the conference on Design, automation and test in Europe (DATE)*. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2006, pp. 233–238.

[23] P. Brisk, A. K. Verma, and P. Ienne, "Optimal polynomial-time interprocedural register allocation for high-level synthesis and asip design," in *Proc. of the International Conference on Computer-aided Design (CCAD)*. Piscataway, NJ, USA: IEEE Press, 2007, pp. 172–179.

[24] Q. Dinh, D. Chen, and M. D. F. Wong, "Efficient asip design for configurable processors with fine-grained resource sharing," in *Proceedings of the International Symposium on Field Programmable Gate Arrays (ISFPGA)*. New York, NY, USA: ACM, 2008, pp. 99–106.

[25] P. Benoit, G. Sassatelli, L. Torres, D. Demigny, M. Robert, and G. Cambon, "Metrics for reconfigurable architectures characterization: Remanence and scalability," in *Reconfigurable Architecture Workshop*, 2003.

[26] R. Enzler, T. Jeger, D.Cottet, and G. Troster, “High-level area and performance estimation of hardware building blocks on FPGAs,” in *Field-Programmable Logic and Applications Forum on Design Language*, 2000.

[27] S. Bilavarn, G. Gogniat, J. L. Philippe, and L. Bossuet, “Fast prototyping of reconfigurable architectures from a C program,” in *IEEE Symposium on Circuits and Systems*, 2003.

[28] L. Bossuet, G. Gogniat, and J.-L. Philippe, “Generic design space exploration for reconfigurable architectures,” in *Proc. of the Reconfigurable Architectures Workshop (RAW)*, 2005.

[29] A. K. Jones, R. Hoare, D. Kusic, G. Mehta, J. Fazekas, and J. Foster, “Reducing power while increasing performance with supercisc,” *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 5, no. 3, pp. 1–29, August 2006.

[30] R. Hoare, A. K. Jones, D. Kusic, J. Fazekas, J. Foster, S. Tung, and M. McCloud, “Rapid VLIW processor customization for signal processing applications using combinational hardware functions,” *EURASIP Journal on Applied Signal Processing*, vol. 2006, pp. Article ID 46 472, 23 pages, 2006.

[31] G. D. Micheli, *Synthesis and Optimizaton of Digital Circuits*. McGraw-Hill Inc., 1994.

[32] G. Mehta, J. Stander, J. Lucas, R. R. Hoare, B. Hunsaker, and A. K. Jones, “A low-energy reconfigurable fabric for the supercisc architecture,” *Journal of Low Power Electronics*, vol. 2, no. 2, pp. 148–164, August 2006.

[33] G. Mehta, R. R. Hoare, J. Stander, and A. K. Jones, “Design space exploration for low-power reconfigurable fabrics,” in *Proc. of the Reconfigurable Architectures Workshop (RAW)*, 2006.

[34] G. Mehta, A. K. Jones, J. Stander, M. Baz, and B. Hunsaker, “Interconnect customization for a hardware fabric,” *ACM Transactions on Design Automation for Electronic Systems (TODAES)*, vol. 14, no. 1, January 2009.

[35] M. Baz, B. Hunsaker, G. Mehta, J. Stander, and A. K. Jones, “Mapping and design of a hardware fabric,” University of Pittsburgh, Industrial Engineering, Tech. Rep., 2007, under review for *Operations Research* since April 2007.

[36] T. Bray, J. Paoli, E. M. C. M. Sperberg-McQueen, and F. Yergeau, “Extensible markup language (xml) 1.0 (fourth edition) - origin and goals,” World Wide Web Consortium, Tech. Rep. 20060816, 2006.

[37] A. K. Jones, D. Bagchi, S. Pal, P. Banerjee, and A. Choudhary, *Pact HDL: Compiler Targeting ASIC’s and FPGA’s with Power and Performance Optimizations*, R. Graybill and R. Melhem, Eds. Boston, MA: Kluwer Academic Publishers, 2002.

[38] K. Roy and S. Prasad, *Low-Power CMOS VLSI Design*. John Wiley and Sons Inc., 2000.

[39] J.-M. Chang and M. Pedram, “Module assignment for low power,” in *European Design Automation Conference*, 1996. [Online]. Available: citeseer.nj.nec.com/chang96module.html

[40] K. Khouri, G. Lakshminarayana, and N. Jha, “Impact: A highlevel synthesis system for low power control-flow intensive circuits,” in *Proc. Design Automation & Test in Europe Conf.*, 1998, pp. 848–854. [Online]. Available: citeseer.nj.nec.com/khouri98impact.html

[41] A. Raghunathan and N. K. Jha, “Behavioral synthesis for low power,” in *Proceedings of ICCD*, October 1994, pp. 318–322.

[42] Z. X. Shen and C. C. Jong, “Exploring module selection space for architectural synthesis of low power designs,” in *IEEE International Symposium on Circuits and Systems*, 1997.

[43] N. K. Jha, “Low power system scheduling and synthesis,” in *Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*. IEEE Press, 2001, pp. 259–263.

[44] X. Liu and M. C. Papaefthymiou, “A markov chain sequence generator for power macromodeling,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, July 2004.

[45] ——, “A static power estimation methodology for ip-based design,” in *Design, Automation, and Test in Europe*, March 2001, pp. 280–287.

[46] Synopsys Inc., “Design compiler and primepower manual,” `www.synopsys.com`.

[47] C. J. Ihrig, M. Baz, J. Stander, R. R. Hoare, B. A. Norman, O. Prokopyev, B. Hunsaker, and A. K. Jones, *Greedy Algorithms for Mapping onto a Coarse-grained Reconfigurable Fabric*. I-Tech Education and Publishing, 2008.

[48] M. Baz, “Optimization of mapping onto a flexible low-power electronic fabric architecture,” Ph.D. dissertation, University of Pittsburgh, Pittsburgh, Pennsylvania, July 2008.

[49] M. Baz, B. Hunsaker, G. Mehta, J. Stander, and A. K. Jones, “Application mapping onto a coarse-grained computational device,” *European Journal of Operations Research*, 2008, in review.

[50] G. Mehta, C. J. Ihrig, and A. K. Jones, “Reducing energy by exploring heterogeneity in a coarse-grain fabric,” in *Proc. of the IPDPS Reconfigurable Architecture Workshop (RAW)*, 2008.

[51] S. Borkar, “Design challenges of technology scaling,” *IEEE Micro*, vol. 19, no. 4, 1999.

[52] K. B. et al., “Design and cad challenges in sub-90 nm cmos technologies,” in *ICCAD*, November 2003.

[53] S. M. et al., “1v power supply high-speed digital circuit technology with multi threshold-voltage cmos,” *JSSC*, vol. SC-30, August 1995.

[54] J. K. et al., “Mtcmos hierarchical sizing based on mutual exclusive discharge patterns,” in *DAC*, June 1998.

[55] H. K. et al., “A super cut-off cmos (sccmos) scheme for 0.5 v supply voltage with picoampere stand-by current,” *JSSC*, vol. SC-35, October 2000.

[56] S. K. et al., “Enhanced multi-threshold (mtcmos) circuits using variable well bias,” in *ISLPED*, August 2001.

[57] M. A. et al., “Dynamic and leakage power reduction in mtcmos circuits using an automated efficient gate clustering technique,” in *DAC*, June 2002.