Imperial College of London
Department of Computing

Hybrid FPGA: Architecture and Interface

Chi Wai, Yu

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of the Imperial College
June 2010

Declaration

This thesis is a presentation of my original research work. The contributions of others are involved, every effort is made to indicate this clearly in the references to the literature and acknowledgement of collaborative researches.

Signature: ....

Date: ....

Abstract

Hybrid FPGAs (Field Programmable Gate Arrays) are composed of general-purpose logic resources with different granularities, together with domain-specific coarse-grained units. This thesis proposes a novel hybrid FPGA architecture with embedded coarse-grained Floating Point Units (FPUs) to improve the floating point capability of FPGAs. Based on the proposed hybrid FPGA architecture, we examine three aspects to optimise the speed and area for domain-specific applications.

First, we examine the interface between large coarse-grained embedded blocks (EBs) and fine-grained elements in hybrid FPGAs. The interface includes parameters for varying: (1) aspect ratio of EBs, (2) position of the EBs in the FPGA, (3) I/O pins arrangement of EBs, (4) interconnect flexibility of EBs, and (5) location of additional embedded elements such as memory.

Second, we examine the interconnect structure for hybrid FPGAs. We investigate how large and high-density EBs affect the routing demand for hybrid FPGAs over a set of domain-specific applications. We then propose three routing optimisation methods to meet the additional routing demand introduced by large EBs: (1) identifying the best separation distance between EBs, (2) adding routing switches on EBs to increase routing flexibility, and (3) introducing wider channel width near the edge of EBs. We study and compare the trade-offs in delay, area and routability of these three optimisation methods.

Finally, we employ common subgraph extraction to determine the number of floating point adders/subtractors, multipliers and wordblocks in the FPUs. The wordblocks include registers and can implement fixed point operations. We study the area, speed and utilisation trade-offs of the selected FPU subgraphs in a set of floating point benchmark circuits. We develop an optimised coarse-grained FPU, taking into account both architectural and system-level issues. Furthermore, we investigate the trade-offs between granularities and performance by composing small FPUs into a large FPU.

The results of this thesis would help design a domain-specific hybrid FPGA to meet user requirements, by optimising for speed, area or a combination of speed and area.

Acknowledgements

I am indebted to my supervisors Professor Wayne Luk and Professor Philip Leong for their patience and enthusiasm. Without their support, I would not have started my doctoral study in UK. Their useful ideas and advice help me to find a right direction of research and complete this thesis.

Besides my supervisors, I am grateful to Professor Steven Wilton for his involvement in my research. His knowledge and experience in reconfigurable architecture assist me to improve the research ideas and results.

Special thanks to Dr. Chun Hok Ho, for his proposed floating point hybrid FPGA architecture in Chapter 3, Section 3.1. I have learnt differnt design tools from him while preparing the benchmarks and experiments. I would like to acknowledge Dr. Julien Lamoureux for his invaluable suggestions in Chapter 4 and Dr. Alastair Smith for providing the common subgraph extraction program to enable the optimisation of floating point units in Chapter 6.

I would also like to thank Professor Jonathan Rose for providing useful comment on descriptions of future work.

I express my gratitude to fellow colleagues in Custom Computing Group and Computer Architecture Group of Imperial College London: Prof. Ray Cheung, Prof. Yuet Ming Lam, Dr. Brittle Tsoi Dr. Van Fu, Mr. Carlos Tavares, Dr. Lee Howes, Dr. Qiang Liu, Dr. Kubilay Atasu, Dr. Markus Koster, Prof. Qiang Wu, Mr. Gary Chow, Mr. Anson Tse, Dr. Timothy Todman, Dr. David Thomas, Mr. Qi Wei Jin, Mr. Tobias Becker, Mr. Craig Court, Mr. Abdelmalek Halawani, Mr. Philip Potter, Mr. William Osborne, Dr. Gabriel Figueiredo, Mr. Kees Van Der Bok, Mr. Adrien Le Masle, Dr. Andreas Fidjeland, Prof. Peter Jamieson, and Mr. Brahim, for their time of discussions, experience sharings and proof-reading my thesis.

In addition, it is an honor for me to be a friend of Ms. Ling Li, Ms. Lucille Cheung, Prof. Edith Ngai, Ms. Lucia Lau, Mr. Min Li, Ms. Dong Ping Zhang, Ms. Nancy Goslowsky, Ms. On Ching Lam, Ms. Wing Sze Wong, Mr. Terrence Mak, Prof. Yumi Iwashita, Ms. Erica Ngai, Mr. Mai Xu, and Mr. Yi Zhou Wang. They provide happiness and an unforgetable memory to my life in UK.

The support of all of my friends in Hong Kong is also appreciated.

The financial support of UK Engineering and Physical Sciences Research Council (EP/D060567/1) is gratefully acknowledged.

Dedication

To my parents: *Wan Wai Fong and Yu Cheong* -

who guide and take care of me and the family with their love,

To my sisters: *Yvonne, Yu Man Ki and Megan, Yu Wing Sze* -

who help and solve my daily problems in my life,

To my love: *Shirley, Leung Shuk Man* -

who loves me, shares my feeling everyday and waits for me during my PhD,

To my cat: *Liky* -

who always gives joy and happiness to me.

I am very glad to have all of you in my life. This thesis would not be finished without any of you.

Abbreviations

ALM	-	Adaptiv Logic Module
ASIC	-	Application Specific Integrated Circuit
BLE	-	Basic Logic Element
BLIF	-	Berkeley Logic Interchange Format
CAD	-	Computer-Aided Design
CB	-	Connection Box
CGU	-	Coarse-Grained Unit
CLB	-	Configurable Logic Block
CPU	-	Central Processing Unit
D_{eb}	-	Separation distance between Embedded Blocks
DP	-	Double Precision
DSP	-	Digital Signal Processor
EB	-	Embedded Block
FA	-	Floating Point Adder/Subtractor
FF	-	Fliip Flop
FM	-	Floating Point Multiplier
FP	-	Floating Point
FPFPGA	-	Floating Point FPGA
FPGA	-	Field Programmable Gate Array
FPU	-	Floating Point Unit
GPU	-	Graphics Processing Unit
HDL	-	Hardware Description Language
I/O	-	Input / Output
LSB	-	Least-Significant Bit
LUT	-	Look Up Table
MCS	-	Maximum Common Subgraph
MSB	-	Most-Significant Bit
NoC	-	Network-on-Chip
RTL	-	Register Transfer Level
SB	-	Switch Box
SoC	-	System-on-a-Chip

SP	-	Single Precision
SRAM	-	Static Random Access Memory
SRL	-	Shift Register LUT
UCF	-	User Constraint File
VEB	-	Virtual Embedded Block
VHDL	-	Very High Speed Integrated Circuit Hardware Description Language
VPH	-	Versatile Place and Route for Hybrid FPGA
VPR	-	Versatile Place and Route
WB	-	Wordblock

Publications

Journal Papers

1. C.H. Ho, C.W. Yu, P.H.W. Leong, W. Luk, and S.J.E. Wilton, “Floating point FPGA: Architecture and Modeling”, *IEEE Transactions on Very-Large Scale Integration (VLSI) Systems*, 17(2):1709-1718, Dec. 2009.
2. C.W. Yu, Julien Lamoureux, S.J.E. Wilton, P.H.W. Leong, and W. Luk, “The Coarse-Grained/Fine-Grained Logic Interface with Embedded Floating-Point Arithmetic Units”, *International Journal of Reconfigurable Computing*, vol. 2008, Article ID 736203, 2008.

Conference Papers

1. C.W. Yu, Alastair M. Smith, W. Luk, P.H.W. Leong and S.J.E. Wilton, “Optimizing Coarse-Grained Units in Floating Point Hybrid FPGA”, *In Proc. International Conference on Field-Programmable Technology (FPT)*, pp.57-64, 2008.
2. C.W. Yu, Julien Lamoureux, S.J.E. Wilton, P.H.W. Leong, and W. Luk, “The Coarse-Grained/Fine-Grained Logic Interface with Embedded Floating-Point Arithmetic Units”, *In Proc. Southern Programmable Logic Conference (SPL)*, pp.63-68, 2008. (Synplicity Best Ph.D. Student Paper Award)
3. C.H. Ho, C.W. Yu, P.H.W. Leong, W. Luk and S.J.E. Wilton, “Domain-Specific Hybrid FPGA: Architecture and Floating Point Applications”, *In Proc. International Conference on Field Programmable Logic and Applications (FPL)*, pp.196-201, 2007. (Stamatis Vassiliadis Award for Outstanding Paper)

Short Papers

1. C.W. Yu, W. Luk, S.J.E. Wilton, P.H.W. Leong, “Routing Optimization for Hybrid FPGAs”, *In Proc. International Conference on Field-Programmable Technology (FPT)*, pages 419-422, 2009.

PhD forum

1. C.W. Yu, “A Tool for Exploring Hybrid FPGAs”, *In Proc. International Conference on Field Programmable Logic and Applications (FPL)*, pages 509-510, 2007.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Application Specific Integrated Circuit (ASIC) technologies have been employed by industry to implement digital circuits or embedded systems for many years. ASIC can produce high speed, compact and low power design. However, it is expensive (e.g. over millions of dollars for the cost of building photo-masks) and usually takes a long time (several months to several years) to develop and fabricate a fixed-function ASIC devices [1].

Field Programmable Gate Arrays (FPGAs) are pre-fabricated semiconductor devices containing programmable logic components and programmable interconnects. Unlike fixed-function ASIC designs, customised applications can be developed to perform any digital function by configuring the components in the FPGA. The FPGA devices can be reconfigured for another application within a second. This highly flexible technology allows fast and low cost development for industry by removing the cost and time for fabrication. Therefore FPGA devices are used to implement complex System-on-a-Chip (SoC) designs and a wide range of applications such as broadcast video, digital TV, wireless base stations, medical equipments and automation. According to the EE Times report in March and July 2009, ASIC design starts fall by 22% in 2009.

FPGAs were first debuted in mid to late-80s and the Xilinx XC2064 FPGA had only 64 programmable logic blocks. Modern devices such as Altera's Stratix-III [2] and Xilinx's Virtex-5 [3] consist of over 200,000 programmable logic blocks. However, FPGA designs are still slower and less compact than ASICs [4]. Today, further improvements are being made by embedding coarse-grained blocks such as memories within the fine-grained programmable fabric of an FPGA to increase performance, which is called a hybrid FPGA.

Coarse-grained blocks can implement a specific function more efficiently than fine-grained programmable logic. However, since they are less flexible, they only benefit applications which utilise them. This limits the types of embedded blocks which are commercially viable in general-purpose FPGAs to common circuit blocks such as memories, adders, and multipliers [2, 3, 5, 6]. For domain-specific FPGAs, however, additional embedded blocks for specific applications may be beneficial.

Floating point computations are good examples of customised domain-specific applications. The speed of many floating point applications such as scientific computing and financial calculations can be improved by FPGA-based designs [7]. To obtain further improvement in area and speed, an FPGA that is built specifically to implement applications containing a significant amount of floating point computations would benefit from embedded floating point units. However, such domain-specific hybrid FPGAs are currently not available in commercial products. Recently, Graphics Processing Units (GPUs) [8] become more common to accelerate a wide range of floating point applications [9]. Although GPUs are high-performance many-core processors dedicated to calculating floating point operations, they are less customisable, consume more power and cannot be a stand alone device when compared to FPGAs [10, 11].

In this thesis, we propose a novel coarse-grained Floating Point Unit (FPU) for hybrid FPGAs [12] (Chapter 3) and a methodology to optimise the FPU [13] (Chapter 6). Then we optimise the interface between coarse-grained/fine-grained elements [14] (Chapter 4) and the routing structure inside this floating point FPGA [15] (Chapter 5). This optimised floating point FPGA retains the advantages of general FPGAs.

Computer-Aided Design (CAD) tools are essential to develop and evaluate an architecture or a design. Fine-grained FPGAs can be modelled using the Versatile Place and Route (VPR) tool [16, 17], which has been used to explore many architectural choices in FPGAs including the amount of logic elements and routing resources. However, a major difference in modern devices lies in the inclusion of fast carry chains, memories, multipliers and processors, which serve to improve speed, power and density [18]. We develop a tool called Versatile Place and Route for Hybrid FPGA (VPH) [19] based on VPR. The tool is publicly available [20]. VPH supports place and route of fast carry chains and coarse-grained blocks. User constraints in VPH allow specifying the position of coarse-grained blocks. The VPH tool enables the exploration of hybrid FPGA architectures.

1.2 Domain Specific Hybrid FPGAs Issue

Hybrid FPGAs are traditional island-style FPGA like architectures. They consist of a grid of identical fine-grained resources called Configurable Logic Blocks (CLBs) which are connected by configurable routing resources. There are additional coarse-grained blocks within the fine-grained fabric to improve the performance of the FPGAs. In particular, coarse-grained blocks for specific application domain such as floating point computation can further improve area, delay and power efficiency. This is because these domain-specific coarse-grained blocks can implement specific functions more efficiently than the fine-grained logic [4]. However, these domain-specific blocks are more complex and larger than embedded memories and multipliers. They waste area when they are not used by an application. FPGAs vendors must consider this trade-off to determine the type and number of coarse-grained blocks that should be embedded within their devices. In this thesis, we focus on three main issues of domain-specific hybrid FPGAs based on a novel floating point unit. They are:

1. Interface between fine-grained and coarse-grained elements (Chapter 4).

2. Routing structure (Chapter 5).

3. Internal structure of coarse-grained blocks (Chapter 6).

1.2.1 Interface between Fine-grained and Coarse-grained Elements

The first issue we investigate for the floating point hybrid FPGAs is the interface between coarse-grained and fine-grained elements. An important consideration when adding coarse-grained Embedded Blocks (EBs) to an FPGA is the interface between the coarse-grained and fine-grained resources. If this interface is not flexible enough, the usefulness of the embedded blocks will be reduced, since connections to and from the blocks will be expensive. On the other hand, if the interface is too flexible, it will require too much area and delay, possibly negating the density and performance advantages of including the embedded blocks, and resulting in unnecessary overhead for applications that do not use the embedded component. We consider four interface parameters: (1) EB position, (2) I/O pin location of EBs, (3) width of the channels surrounding the EB, (4) shape of the EB. Figure 1.1 shows an example of the interface parameters in a hybrid FPGA.

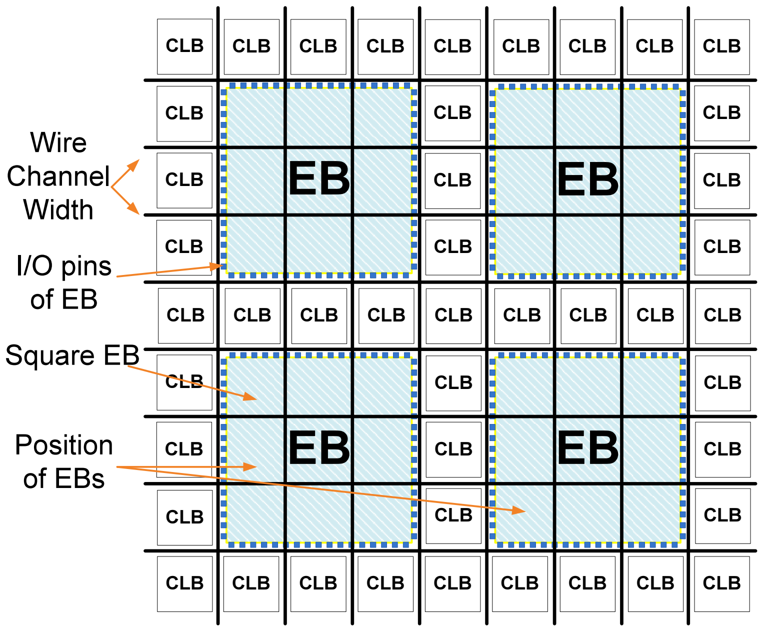


Figure 1.1: An example of interface in hybrid FPGA

1.2.2 Routing Structure

The second issue we examine is the routing structure of the hybrid FPGAs. Programmable interconnects in FPGAs have major impact on the delay and area [21]. The interconnects include connection boxes, switch boxes and wire segments connecting logic blocks as shown in Figure 1.2. In hybrid FPGAs, high density coarse-grained units usually require a high number of channel width. Optimisation of these interconnects to meet the extra routing demanded by large blocks can greatly reduce the area and delay of those FPGAs [22]. Therefore, we propose an optimised routing architecture for hybrid FPGAs. Also, we examine whether the commonly used column based architecture for small embedded memories and multipliers in commercial FPGAs is suitable for large but complex blocks such as FPUs.

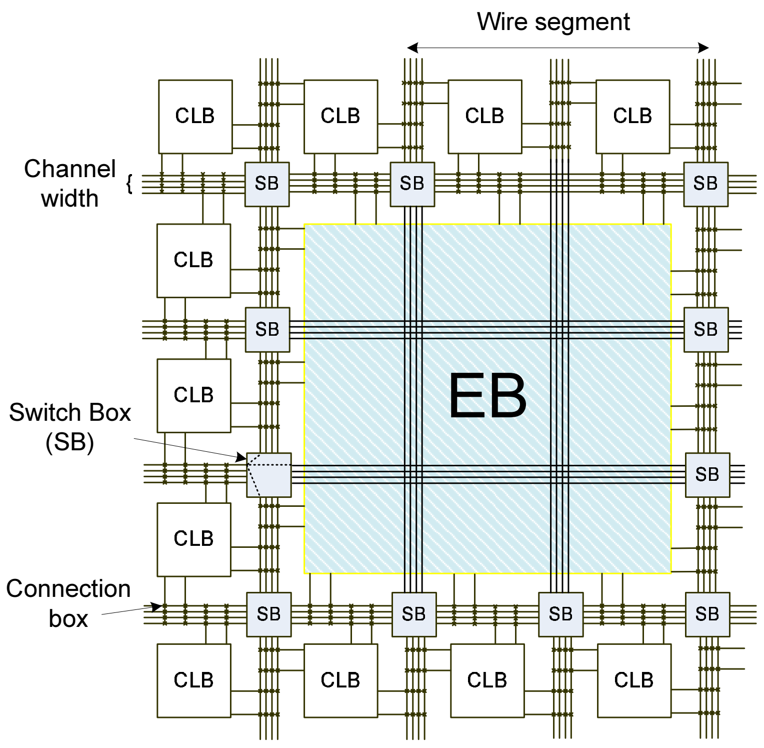


Figure 1.2: Detailed routing architecture in hybrid FPGA

1.2.3 Internal Structure of Coarse-grained Blocks

The final hybrid FPGA issue we consider is the optimisation of the internal structure of the coarse-grained embedded block. We design a novel bus based FPU embedded in the FPGA, which contains Floating point Adders/subtractors (FAs), Floating point Multipliers (FMs) and Wordblocks (WBs). FAs and FMs contain several basic functional elements such as barrel shifters, adders and multipliers. WBs are used for the bitwise operation of a floating point number such as comparison, shifting, latch and logical operation. Constructing hard circuit WBs, FAs and FMs, which are composed of basic functional elements, results in a more compact block with higher speed, but less flexibility. Optimising these hard circuits is essential. Grouping together optimised WBs, FAs and FMs to form an FPU can further improve the speed and area, since the interconnects between them use bus-based connections. Figure 1.3 shows an example of constructing an FPU by grouping together WBs, FAs and FMs. We examine the impact of FPUs on hybrid FPGAs in terms of area, speed, routing resources and flexibility.

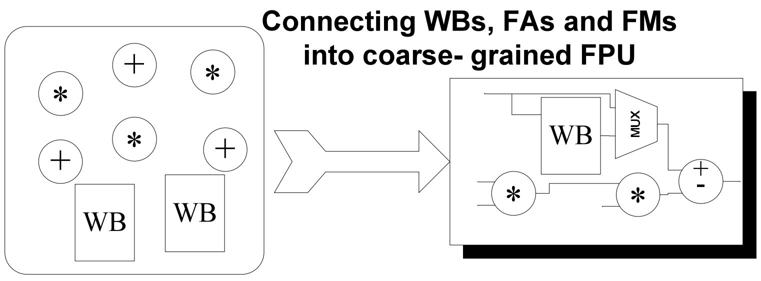


Figure 1.3: Constructing bus based coarse-grained FPUs

1.3 Research Approach and Contributions

Our research approach extends various existing optimisation algorithms for programmable logic devices. We limit the design space by defining a set of architectural parameters for optimisation. We carry out experiments on a set of floating point benchmarks to measure the area and delay trade-offs of those architectural parameters. The results are based on a CAD tool for the novel architecture.

The main contributions of this thesis are:

1. A novel bus based coarse-grained FPU to achieve improvement in speed and area for floating point applications (Chapter 3, Section 3.1).
2. The VPH tool for exploring domain-specific hybrid FPGA architectures and supporting optimisation for routing architecture (Chapter 3, Section 3.3).
3. An optimised interface between coarse-grained and fine-grained elements in hybrid FPGAs (Chapter 4).

4. An optimised routing architecture in hybrid FPGAs (Chapter 5).

5. Optimised coarse-grained units in floating point hybrid FPGAs based on common subgraph extraction (Chapter 6).

Table 1.1 summarises the contributions of this thesis.

1.4 Thesis Organisation

This thesis is organised as follows (Figure 1.4). Chapter 2 describes the background and related work in CAD tool, FPGA architecture, floating point arithmetic and application benchmarks. Chapter 3 presents our novel floating point unit and the tool for exploring the hybrid FPGAs. Chapter 4 investigates the interface between coarse-grained and fine-grained elements in the FPGAs. Chapter 5 suggests three routing optimisation schemes for hybrid FPGAs. Chapter 6 employs common subgraph extraction to determine the optimised coarse-grained FPU. Chapter 7 summarises the thesis and suggests for future work.

Table 1.1: Summary of the research contributions

Research Area	Contributions
Architecture and modelling (Chapter 3)	• An architecture for floating point hybrid FPGA. • An evaluation tool which supports floating point units and various optimised routing architectures.
Interface between coarse/fine-grained elements (Chapter 4)	• A set of parameters that describes the interface between coarse-grained and fine-grained programmable logic in FPGAs. • An empirical framework to model the impact of coarse-grained architectural parameters in terms of performance and density. • An empirical study that examines: 1. Where the coarse-grained FPUs should be embedded within FPGAs. 2. Where the pins of the FPUs should be on the periphery. 3. How flexible the interconnect between the FPUs and the fine-grained logic should be. 4. What shape the FPU should have. • A study of a hybrid FPGA interface containing embedded memories and FPUs including: 1. Where embedded memories used by the FPUs should be located. 2. How flexible the interconnect between the FPUs, embedded memories and the fine-grained logic should be.
Routing optimisation (Chapter 5)	• A study of the extra routing requirements introduced by large embedded blocks. • Optimised routing architectures for hybrid FPGA to meet the extra routing demand: 1. Separating the embedded blocks. 2. Adding routing switch on embedded blocks. 3. Inserting extra wires near the edge of embedded blocks. • Compare the optimised routing architecture to existing column based FPGA. • Ways of improving the interface between coarse/fine-grained elements in FPGA with optimised routing architecture.
Optimisation of coarse-grained FPU (Chapter 6)	• A novel methodology to optimise the floating point hybrid FPGA by considering both internal architecture of FPUs and system-level performance according to the mixture of FPUs. • A study of internal architecture of FPUs. Common arithmetic subcircuits for floating point benchmark circuits are examined, and use these subcircuits to form a hardcore FPU. • A quantitative system-level analysis of the speed, area and routing resource trade-offs of common FP hardcores. By considering the speed, the area and the routing resource of a hybrid FPGA with selection of different hardcores, optimised designs can be obtained.

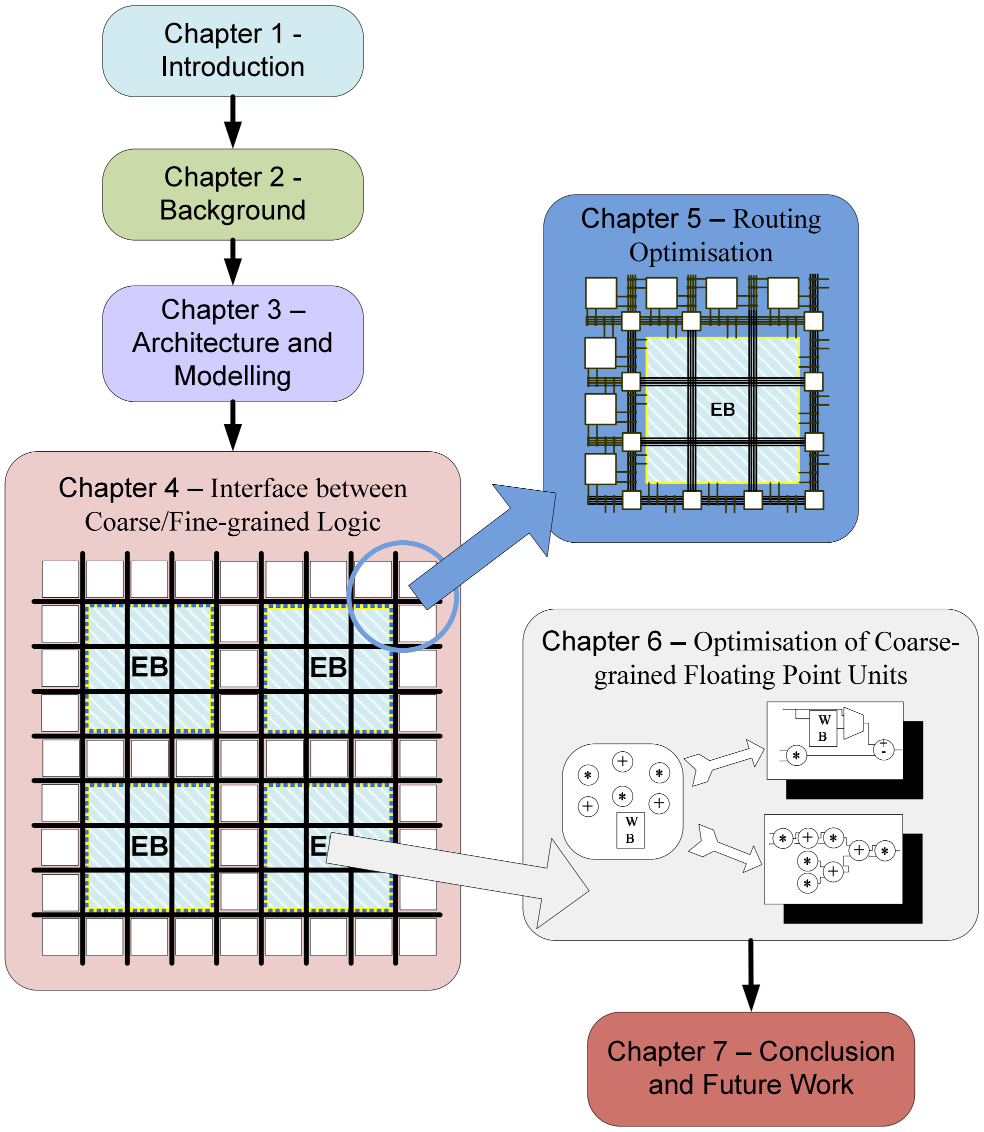

Chapter 1 - Introduction
Chapter 2 - Background
Chapter 3 – Architecture and Modelling
Chapter 4 – Interface between Coarse/Fine-grained Logic
EB
EB
EB
E
Chapter 5 – Routing Optimisation
EB
Chapter 6 – Optimisation of Coarse-grained Floating Point Units
W B
W B
Chapter 7 – Conclusion and Future Work

Figure 1.4: Organisation of this thesis

Chapter 2

Background

There is much related work. In this chapter, previous work on FPGA architecture, FPGA application, FPGA exploration, CAD tools for FPGA and different computational devices are discussed and summarised.

2.1 Fine-grained FPGA Architecture

Conventional island-style FPGAs consist of a grid of identical Configurable Logic Blocks (CLBs) and I/O blocks. The CLBs and I/O blocks are connected using horizontal and vertical wires with SRAM-based programmable switches between channel wires as shown in Figure 2.1. Any circuit can be implemented in an FPGA by configuring the logic blocks, I/O blocks and routing resources. Section 2.1.1 and 2.1.2 introduce the detailed architecture of the logic blocks and routing resources in FPGA respectively. In addition, Section 2.1.3 describes the power optimisation in fine-grained FPGAs as power efficiency is an important issue in modern computational devices.

2.1.1 Logic Block

FPGA consists of a two dimensional array of CLBs to provide logical operations. Each CLB, which is shown in Figure 2.2(a), contains a cluster of *N* Basic Logic Elements (BLEs). A BLE in early FPGAs

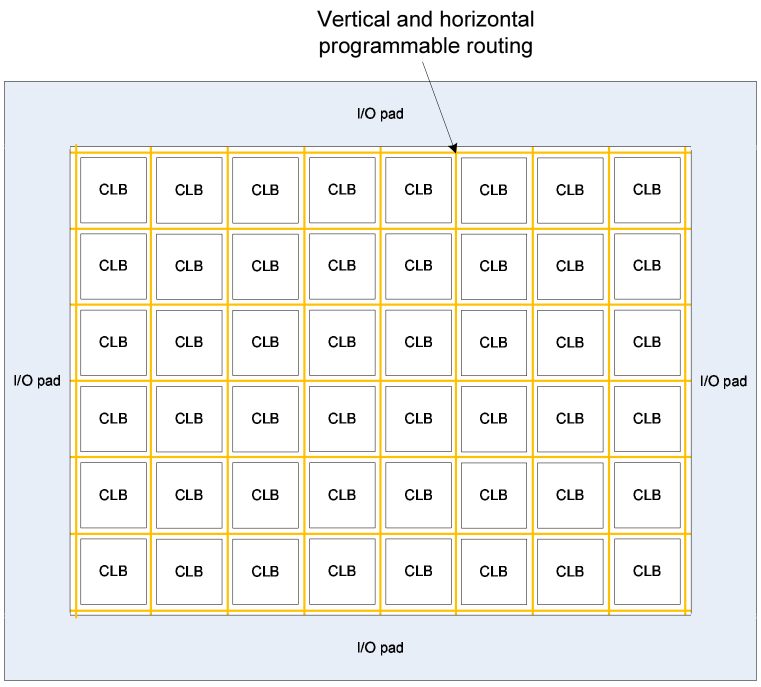


Figure 2.1: A conventional island-style FPGA

contain simply a k input Look Up Table (k-LUT) and a Flip Flop (FF) as shown in Figure 2.2(b) [1, 23]. The k-LUT can implement any single output combinational function with k inputs or less by configuring 2^k SRAM cells. Based on this CLB architecture, previous research has found that the best pin position of CLB should be full-perimeter (the pins are around all the edges of the FPGA) and the best FPGA aspect ratio is 1 [21] , LUT with 4 to 6 and cluster size (N) of 3 to 10 provides the best area-delay performance [1, 24, 25, 26]. Larger LUT (6-LUT) is used in modern commercial devices to achieve higher speed for more compact logic functions [27, 28].

Modern FPGA logic blocks such as CLBs in the Xilinx Virtex family [3] and Adaptive Logic Modules (ALMs) in the Altera Stratix family [2] include more complex functions. The CLB contains support for carry chains, shift registers, internal multiplexers and XOR gates. These functions can speed up the computation of addition and shifting operation. We will adopt this modern CLB architecture as the fine-grained element in our modelled domain-specific hybrid FPGA.

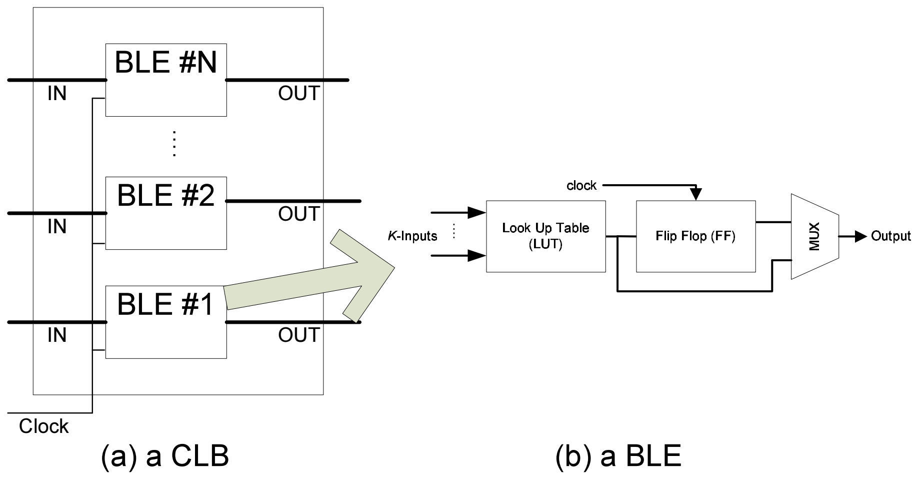


Figure 2.2: (a) A configurable logic block, (b) a basic logic elements in an island-style FPGA

2.1.2 Routing

In FPGAs, the CLBs and I/O blocks are connected by vertical and horizontal channels through Connection Boxes (CBs) [1, 29]. Each channel includes W parallel routing tracks of segment length L. Neighbouring segment channels are intersected by an SRAM-based programmable Switch Box (SB). The length of the segment is the number of CLBs it spans before connecting a switch box. Figure 2.3 shows the detailed routing architecture and an example of segment length of 1 and segment length of 2. The SRAM-based connection box in Figure 2.4(a) consists of multiplexers which allow routing wires connect to the input pins of CLBs. Pass transistors in connection boxes allow routing wires to be driven by the output pins of CLBs. The switch box in Figure 2.4(b) offers each incoming wire the ability to connect to other wire segments. The routing switch simply uses bi-directional tri-state buffers as drivers or pass-transistor to control the connections. A tri-state buffer is a two-stage buffer shown in Figure 2.5. The driving strength is determined by the size of the nMOS and pMOS transistors [30]. Rose and Brown [31] have defined useful parameters for this routing architecture. F_c is connection block flexibility, the number of tracks in a channel that can connect to a CLB pin. Specifically, F_{c_input} and F_{c_output} are the connection box flexibility of CLB input and output pins respectively. F_{c_pad} denotes the connection box flexibility of an I/O pad. F_s is the switch box flexibility; it describes the number of wires in a switch box to which each incoming wire can connect. For example, if F_s=3, the switch box offers each incoming wire the ability to connect to three other wire segments. The baseline routing architecture in Section 3.2 is based on this fine-grained routing assumption.

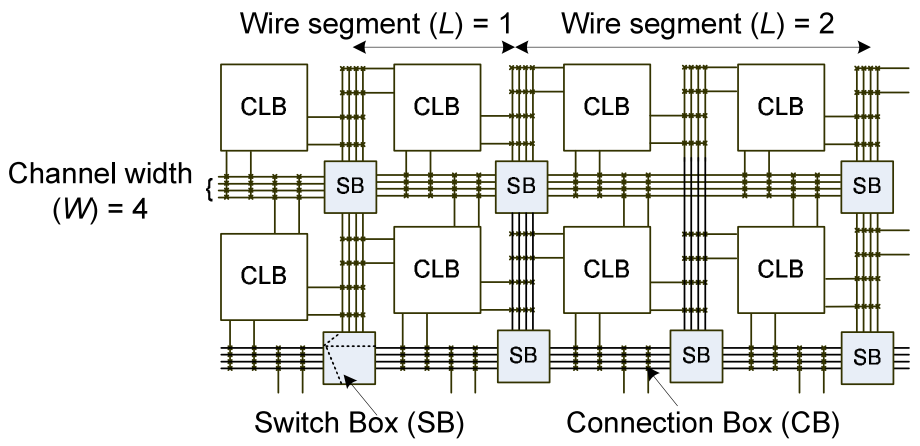


Figure 2.3: The detailed routing routing architecture in an island-style FPGA

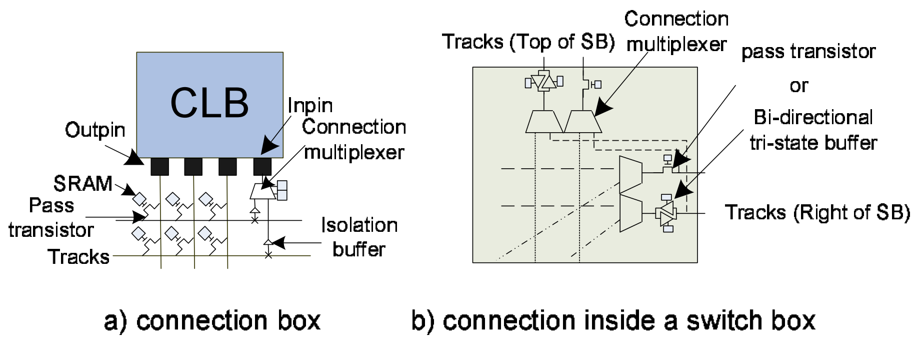


Figure 2.4: Structure of connection and switch box

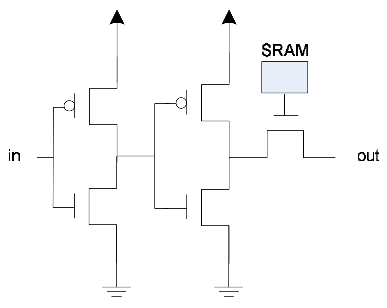


Figure 2.5: Structure of tri-state buffer

The programmable routing resources between logic and I/O pads in this traditional fine-grained island FPGA consume about 70% of the area in a die and contribute significantly to delay [4]. A significant number of studies have focused on optimising this type of FPGA routing architecture to minimise area and critical-path delay. Betz and Rose [22] have examined the impact of segmentation and buffering on FPGAs. They have concluded that wire segments of length 4 lead to the most area-efficient and fastest FPGA. Furthermore, they have evaluated the effect of the routing track distribution on FPGA area-efficiency and indicated that adding extra tracks to the center channel cannot improve routability [21]. Also the most area-efficient FPGA contains about 80% pass-transistor switches to optimise for area and only 20% tri-state buffer switches to optimise for speed. It is because pass-transistors are slower but smaller. Commercial devices use different vertical and horizontal channel widths to accompany the asymmetrical vertical and horizontal aspect ratio in rectangular FPGAs [24]. These devices contain a lot of dedicated interconnects with different fixed segment lengths enabling direct fast connection from one tile to another. For example, there are double, hex and long lines in the Xilinx Virtex-5 [3].

The routing switches have been much optimised over the past decades. The best routing transistor size in switch boxes has been determined in [23, 32, 33], as five times the minimum transistor size. Subset switch box [29], universal switch box [34] and Wilton switch box [5] are most commonly used in FPGAs. The subset switch box is the planar or domain-based switch box used in the Xilinx's FPGAs, in which wire segment in track i can only connect to other wire segments in track i. The universal switch box can route any possible two-point routing requirements. The Wilton switch box is the one leading to the most routable FPGAs. Chandra and Schmit [35] have suggested to use simultaneous sizing of driving buffers in routing switches to improve delay by 1-12%. Lemieux et al. [36] have used uni-directional and single-driver wires to improve area-delay by 32% compared to bi-directional wiring. Merging connection and switch boxes into a CS-Box structure [37] can reduce the number of the connection boxes by up to 11.81% with a small penalty of increasing the channel width and the circuit delay by less than 3%.

2.1.3 Power

Power consumption is a critical concern in the semiconductor industry. The power efficiency of FPGAs consisting of dynamic and static power consumption, is lower than ASIC [4, 38, 39]. The transistor leakage current in FPGA causes static power consumption. Leakage current is the small current that leaks from source-to-drain or through the gate oxide; this occurs even when the transistor is logically off. The smaller the physical dimensions of the CMOS transistors, the more current leaks. During switching events in the core or I/O of the device, there is power consumed, which is called dynamic power. Static power constitutes only 10% while dynamic power is 90% of total power consumption in FPGA [38, 39]. Optimisation of power consumption in an FPGA has been studied recently; pre-defined dual voltage fabric to reduce both dynamic and static power consumption has been proposed [40, 41]. This dual voltage technique has been employed in Altera Stratix-III and -IV [42]. The programmable routing fabric consumes 60%-80% of dynamic power [43, 44]. Meijer et al. [45] have demonstrated that the delay of full voltage swing, fully buffered FPGA interconnect design can be matched by low voltage swing hybrid switch, which contains buffers and pass-gates. This hybrid switch dissipates less power with no area penalty. Lin et al. [46] have proposed a monolithically stacked 3D-FPGA architecture to improve logic density 3.3 times, delay 2.35 times and dynamic power consumption 2.82 times compared to a traditional 2D-FPGA.

2.2 Hybrid FPGA Architecture

Traditional FPGAs have been used to speed up many applications because of their reconfigurability and parallel processing ability. However, the performance of applications implemented in FPGAs are worse than ASICs. Kuon and Rose [1, 4] have measured the gap between FPGAs and ASICs. They have found that an FPGA is on average 40 times larger and 3.2 times slower than an ASIC. To address this problem, *hybrid FPGAs* have been proposed [47, 48, 49] which are similar to generic FPGAs, but contain coarse-grained embedded blocks to improve the efficiency of computations. These can either be constructed out of fixed non-programmable logic or provide a limited degree of configurability to make them suitable for a wider application space. Section 2.2.1 introduces the possible coarse-grained embedded blocks in hybrid FPGA. Section 2.2.2 and 2.2.3 present the previous research of the hybrid FPGA routing and power respectively.

2.2.1 Coarse-grained Block

Currently, simple and general embedded blocks such as Digital Signal Processors (DSPs) and memories have been added to general-purpose commercial FPGA [2, 3, 5, 6] to enhance the speed of multiplication and the density of memory. Embedded processors [3] have also been added to improve FPGA processing power. In [50], the local routing resources that connect CLBs to the FPGA routing resources are shared with the embedded blocks to minimise the overall area penalty when adding the embedded blocks. This technique, called *shadow clustering*, is useful for embedded blocks with similar I/O pin densities as the existing CLBs; however, there are not sufficient routing resources for embedded blocks which have higher I/O pin densities than the existing CLBs. In [51], the QUKU architecture which merges a coarse-grained reconfigurable processing element array and an FPGA architecture is described. This two-level reconfigurable architecture provides active support for fast and efficient dynamic reconfiguration.

In order to take further advantage of coarse-grained blocks, domain-specific hybrid FPGAs target a specific application domain. In doing so, greater area and delay savings can be achieved for certain types of applications since the amount of coarse-grained logic can be tailored for those applications.

An example is an application which requires a significant amount of floating point computations. Implementing floating point operations in fine-grained FPGA technology consumes a large amount of logic and routing resources. Therefore, a number of recent approaches to optimise floating point operations in FPGAs have been proposed. Pipeline stages of floating point arithmetic in a custom computing machine have been optimised within existing fine-grained resources [52]. However, the density and speed are still worse than those implemented in ASICs. The study of embedded heterogeneous blocks for the acceleration of floating point computations has been reported. Roesler and Nelson [53] and also Beauchamp et al. [54] have concluded that employing heterogeneous blocks in a floating point unit (FPU) can improve area and delay compared to a fine-grained approach. Chong and

Parameswaran [55] have increased flexibility of an embedded FPU by providing one double-precision operation or two single-precision operations in parallel. This multi-mode embedded FPU improves 5.2 times in area and 5.8 times in delay over a set of benchmarks. Wilton et al. [56] have presented a synthesisable datapath-oriented architecture which can be used to provide post-fabrication flexibility to an SoC. We propose a novel floating point FPGA architecture in Chapter 3 [12], which consists of both fine-grained elements and word based coarse-grained FPUs based on the synthesisable datapath-oriented embedded fabric.

The common subgraph extraction technique [57] has been employed to find out the fixed point arithmetic units which are common in a set of benchmark circuits. Fused-arithmetic units generated by these common subcircuits get up to 3.3 times speed and 19.7 times area improvements on average for particular silicon cores. We employ this technique in Chapter 6 to determine floating point common subgraphs, similar to the fixed point approach. Instead of the improvement of particular cores, we focus on the system-level trade-offs in hybrid FPGAs.

Although embedding coarse-grained elements can improve performance close to ASIC, the flexibility and granularity are worse than fine-grained FPGAs. Because the embedded blocks are less configurable than fine-grained elements, they are wasted when not used. Figure 2.6 shows a graphical comparison of ASIC, fine-grained FPGA, hybrid FPGA and domain-specific FPGA.

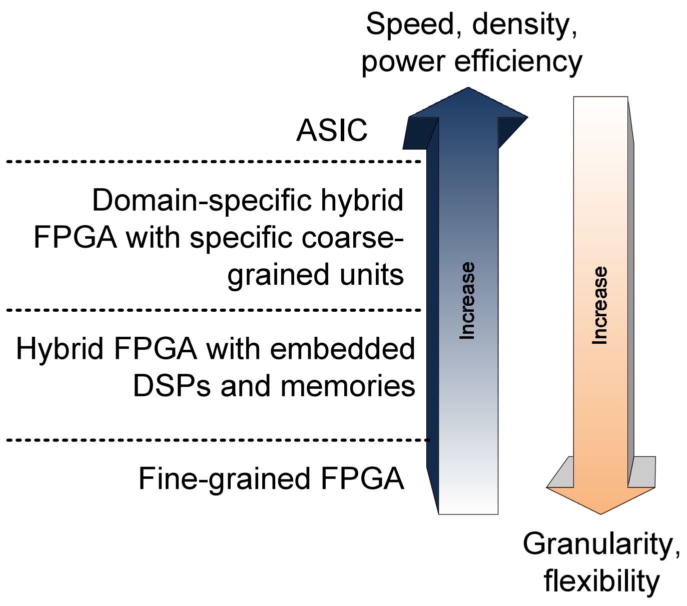

Speed, density, power efficiency
ASIC
Domain-specific hybrid FPGA with specific coarse-grained units
Hybrid FPGA with embedded DSPs and memories
Fine-grained FPGA
Increase
Increase
Granularity, flexibility

Figure 2.6: Comparison of ASIC, fine-grained FPGA, hybrid FPGA and domain-specific FPGA

2.2.2 Routing

As mentioned in Section 2.1.2, FPGA routing is the bottleneck for area and delay. It is also a major issue in hybrid FPGA.

Unlike general-purpose FPGAs which contain relatively simple and small embedded blocks (DSPs and memories), the embedded blocks in hybrid FPGAs can be larger and more complex for a specific application. High density connections can be routed inside these domain-specific blocks. As an example, the hybrid FPGA we have proposed in [12] (Chapter 3) contains large embedded Floating Point Units (FPUs) in addition to a general-purpose bit-level FPGA fabric. Each FPU is itself configurable, but at a more coarse granularity than the FPGA fabric. The key differentiating feature in our hybrid FPGAs is the presence of these large embedded blocks. It is shown that the presence of a large embedded block affects the routing demand within the fine-grained logic [58]. This extra demand causes Altera to use smaller memory blocks instead of the large MegaRAM Block in Stratix-III and Stratix-IV devices [24,42]. Existing commercial devices such as Xilinx Virtex-5 [3] arrange the small embedded blocks like memory and DSP in columns. This arrangement may not be efficient for large blocks. Therefore optimisation of the routing in hybrid FPGAs is essential.

In [59], a coarse-grained architecture with multibit bus-based connections in FPGA is proposed. Datapath circuits in large arithmetic intensive applications consist of regularly structured signals; this multibit routing architecture can improve FPGA area by 10%. Mak et al. [60] have studied seven types of FPGA-based communication architectures with embedded coarse-grained processors, embedded memories and IP cores. Network-on-Chip (NoC) architectures have been advocated and are believed to be a promising solution for on-chip communication.

In [58], the interface between embedded memory blocks and fine-grained programmable logic is examined. Memories are quite different from computation blocks, and so we expect that the interface presented in [58] would not be suitable for our FPUs. There is little research on these issues for domain-specific coarse-grained blocks. Therefore, we investigate the interface in Chapter 4 and routing structure in Chapter 5 for domain-specific hybrid FPGAs with large embedded computation blocks.

2.2.3 Power

In Section 2.1.3, we discuss power consumption in fine-grained FPGAs. Dynamic power consumption is 12 times more than ASIC, narrowing down to 9 times of ASIC with embedded memories and DSPs [4]. Implementing logic in memory arrays not only leads to an increase in power dissipation, but also increases in circuit density [61]. It is better to leave the array unused, rather than use it to implement logic. Also, smaller memory arrays are more power efficient than large arrays. As a result, choosing a suitable size and combination of the embedded elements is very important to the performance of the system. We examine area and delay trade-offs of different combinations of coarse-grained FPUs in Chapter 6; power exploration of this study will be added in the future work.

Ho et al. [62] have estimated that the floating point applications implemented on hybrid FPGA proposed in Chapter 3 can reduce dynamic power consumption by 14 times compared to the Virtex-II FPGA. Our novel hybrid FPGA architecture has advantages over traditional general FPGA in area, delay and power consumption. It is important to further improve the performance to narrow the gap between FPGAs and ASICs, therefore this thesis focuses on optimising the interface, routing and internal structure of embedded blocks of the domain-specific hybrid FPGA.

2.3 Design Space Exploration Tool

We require a way to evaluate and compare different architectures for the novel hybrid FPGA proposed in this thesis. There are existing tools for design space exploration of different architectures. First, we introduce the general CAD tools for FPGA in Section 2.3.1. Then we employ the CAD tools for hybrid FPGA architectural exploration and divide the tools into two categories:

1. Pre-fabrication evaluation tools, which are used to explore architectures before fabrication of the devices, is described in Section 2.3.2.
2. Post-fabrication evaluation tools, which are used to carry out exploration with the fabricated devices, is described in Section 2.3.3.

In addition to the CAD tool, there exist analytical modelling techniques which consist of a set of mathematical formula to relate the architectural parameters of an FPGA to area, delay and power. This technique is discussed in Section 2.3.4.

2.3.1 CAD Tool for FPGA

The programmable logic blocks and routing resources in FPGA should be configured to an appropriate state to implement a circuit. In general, the circuit is described in high level hardware specification languages. The description of the circuit is then converted into a configuration bit stream by Computer-Aided Design (CAD) programs for the programmable resources.

A typical CAD work flow for FPGA is shown in Figure 2.7 [49, 63]. In the traditional work flow, Hardware Description Languages (HDLs) like VHDL and Verilog or schematic capture are widely used on commercial reconfigurable platforms to describe the circuit to be implemented in the FPGA. The description of the circuit is written at the Register Transfer Level (RTL) which specifies the operations at each clock cycle. The description is then synthesised to netlist of logic blocks before being placed and routed the circuit onto an FPGA.

In the first stage of the synthesis process, the datapath operations in an RTL design such as control logic, memory blocks, registers, adders and multipliers are identified and elaborated into a set of basic boolean logic gates such as AND, OR and XOR. Next, the netlist of basic gates is optimised independent of the FPGA architecture. The optimisation includes: boolean expression minimisation, removing the redundant logic, buffering sharing, re-timing and finite-state machine encoding. The optimised netlist of basic gates is then mapped to the specific FPGA architecture such as Xilinx Virtex devices or Altera Stratix devices. There is further optimisation based on the specific architecture such as carry chains for adders, dedicated shift functions in logic block for shift registers. The final stage in the synthesis process is packing and clustering groups of several LUTs and registers into logic blocks like Figure 2.2(a). The packing and clustering minimise the number of connections between different logic blocks. After the synthesis process, the logic blocks in the mapped netlist are placed onto the FPGA based on the different optimisation goals, such as circuit speed, routability and wirelength.

Once the location of the logic blocks is determined, the connection between I/Os, logic blocks and other embedded elements are routed onto the programmable routing resources in FPGA. The routing process determines which programmable switches should be used to connect the logic block input and output pins. Finally, a configuration bit stream of all I/Os, logic blocks and routing resources for the circuit in specific FPGA is generated.

Modern technology allows high level general-purpose programming languages like C to be added to the tradition tool flow of FPGAs (The upper part of Figure 2.7). These languages include Handel-C [64], Haydn-C [65], Streams-C [66], SPARK [67], ASC [68] and SPC [69]. They specify the behaviour of the design without considering the hardware detailed description of the design. This can facilitate hardware development. Behaviour compilers are used to synthesise high level behavioural description of the design and generate the HDL description of the circuit. The compilers extract parallelism of computation in the source codes, and optimise for pipelining. Although the design time of this technique is faster, the performance of the resulted circuit may be worse than the one designed directly by HDL at RTL. This is because the design is not hardware oriented in the behaviour description, it is not optimised based on the specific hardware architecture. Hence, Todman et al. [70] have proposed customisable frameworks for hardware compilation which enable rapid design exploration, reusable and extensible hardware optimisation. This can be used in producing designs for signal and image processing applications, with different trade-offs in performance and resource consumption.

Besides configuring the circuit, there are CAD tools that analyse the delay, area and power consumption of the implemented circuit. This information is important to evaluate the performance of different FPGA architectures.

2.3.2 Pre-fabrication Evaluation Tool

Before the fabrication of a chip, we need to evaluate the trade-offs of different architectural parameters such as different LUT size, cluster size, aspect ratio of the FPGA, routing structure and I/O pin position. Existing commercial tools such as Xilinx ISE [71] and Altera Quartus [72] cannot evaluate the performance when changing these parameters. We use pre-fabrication evaluation tools to explore

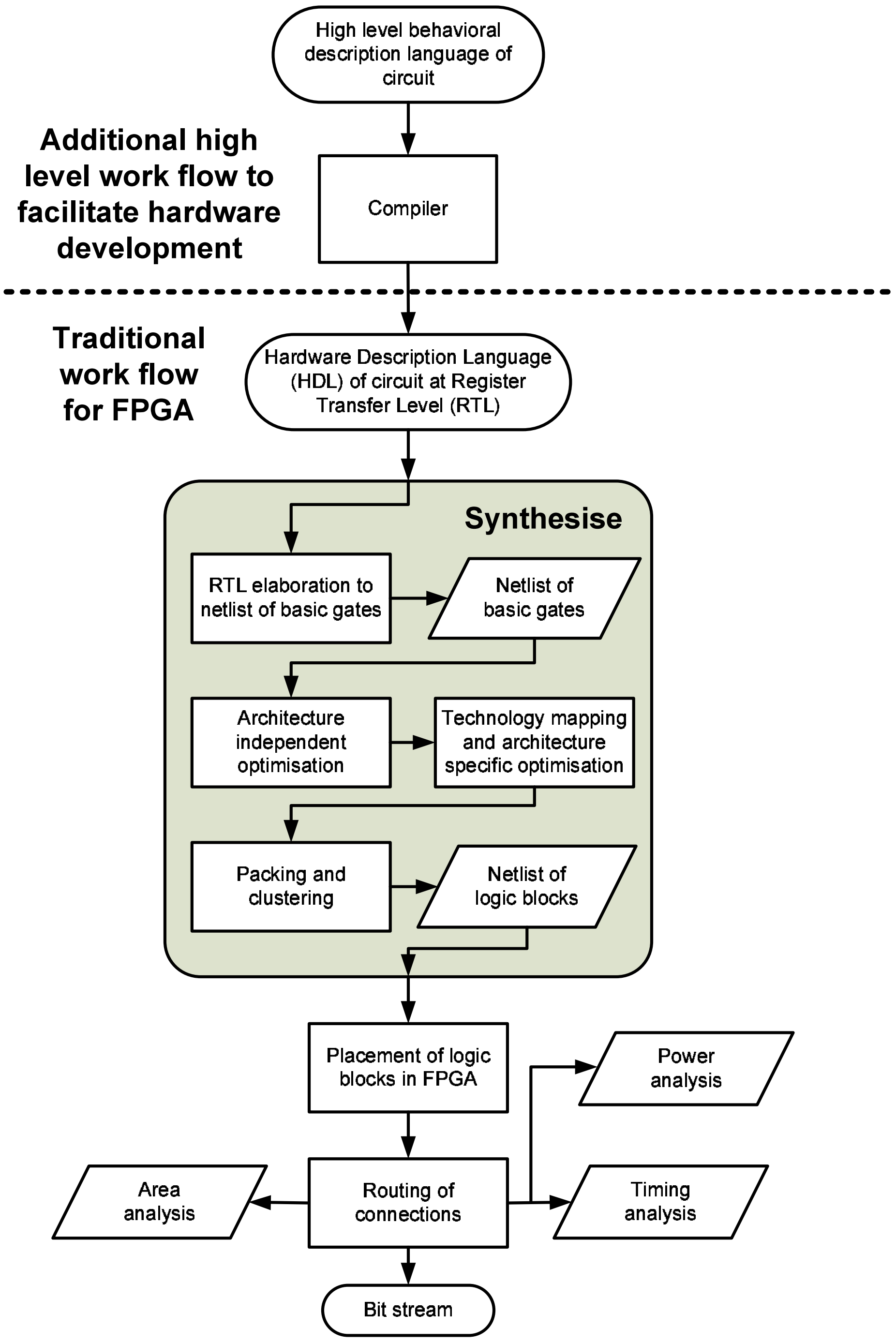

High level behavioral description language of circuit
Additional high level work flow to facilitate hardware development
Compiler
Traditional work flow for FPGA
Hardware Description Language (HDL) of circuit at Register Transfer Level (RTL)
Synthesise
RTL elaboration to netlist of basic gates
Netlist of basic gates
Architecture independent optimisation
Technology mapping and architecture specific optimisation
Packing and clustering
Netlist of logic blocks
Placement of logic blocks in FPGA
Power analysis
Area analysis
Routing of connections
Timing analysis
Bit stream

Figure 2.7: A traditional CAD work flow for FPGA

the architecture of the device before it is fabricated [73, 74].

Versatile Place and Route (VPR) [16] is an open source place and route tool that has been widely used for research into FPGA architectures. VPR is used to explore the logic block [23, 25, 26] and routing structure [21, 22, 35, 36, 37] as mentioned in Section 2.1.

VPR uses an FPGA architecture based on the Xilinx 4000X series, an obsolete device. It places and routes the FPGA resources by using simulated annealing and timing based routing estimation. In the original VPR, only three types of circuit elements are modelled: Configurable Logic Blocks (CLBs), input pads and output pads. Each basic logic element in a CLB contains a *k*-LUT and an FF. Only one output is available from each basic logic element. In the tool flow of VPR (Figure 2.8), a netlist of LUTs and FFs in .blif file format are packed and clustered into a netlist of logic blocks by the T-VPack tool included in VPR. An FPGA architecture description file (.arch file) is used to describe the architectural and electrical parameters of the FPGA such as position of I/O of logic block, delay of the driving buffer and resistance of the wire. VPR uses simulated annealing [75] to obtain a good placement result in a reasonable amount of CPU time. The routing resources of the FPGA in VPR is represented by a directed graph. This routing-resource graph is determined by global, or combined global and detailed routers [16]. The path finder algorithm [76] is used to find the shortest path between a net source node and sink node in the routing-resource graph.

The original VPR framework was released in 1997 and does not model modern features such as carry chains and embedded block RAMs (BRAMs). The centralised development of VPR version 4.30 stopped in year 2000. Recently, other researchers have modified VPR to include features in modern FPGA architectures and also optimised the algorithm used in VPR to reduce exploration time.

Wilton et al. [58] have modified VPR to support embedded memories, and investigated the interface between memories and fine-grained elements. Jamieson and Rose [77] have developed a verilog RTL synthesis tool called Odin for VPR CAD flow. Odin allows synthesis of hard blocks in heterogeneous FPGAs. It also employs simple and efficient mapping technique to generate designs comparable in area and speed to Quartus's synthesis tool. Tom and Lemieux [78] have introduced a system-level technique for fitting hard to route large circuits in limited channel width FPGAs. Beauchamp et al. [54] have modified VPR to explore architectural enhancements of floating point unit in FPGAs.

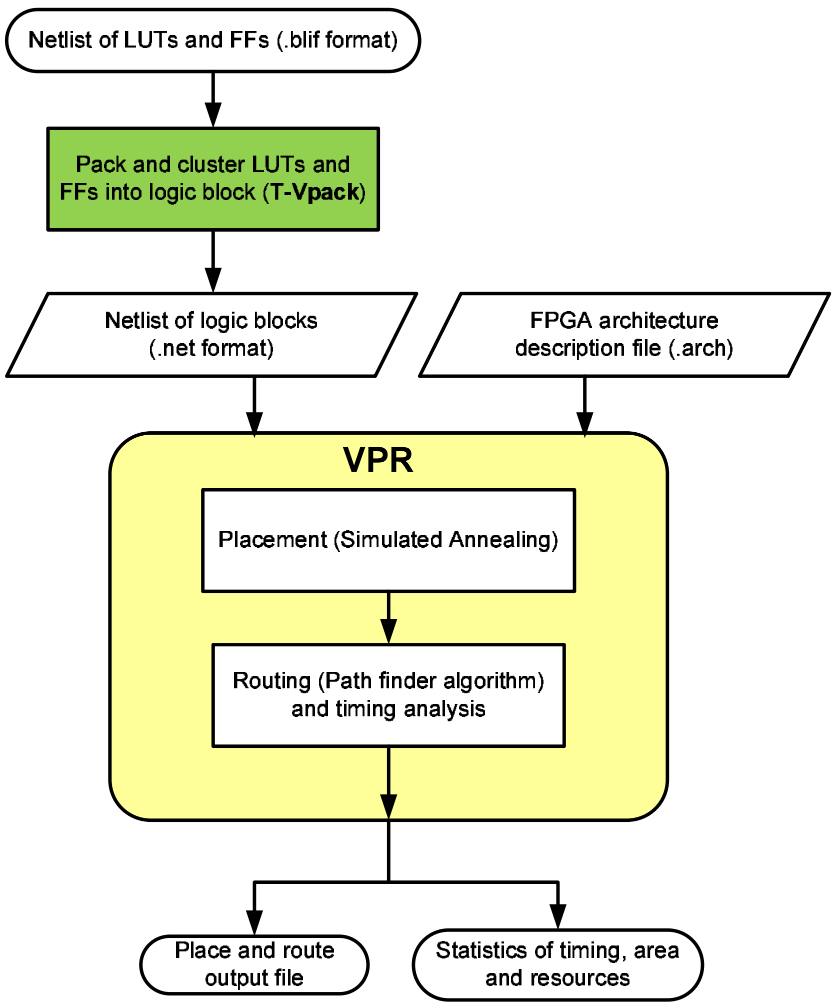

Netlist of LUTs and FFs (.blif format)
Pack and cluster LUTs and FFs into logic block (**T-Vpack**)
Netlist of logic blocks (.net format)
FPGA architecture description file (.arch)
VPR
Placement (Simulated Annealing)
Routing (Path finder algorithm) and timing analysis
Place and route output file
Statistics of timing, area and resources

Figure 2.8: Tool flow of VPR

Altera also uses an extended version of VPR called FPGA Modelling Toolkit (FMT) to explore the Stratix architecture [42]. FMT maps a set of customer designs to the Stratix Adaptive Logic Module (ALM). Heterogeneous memories are also added. It performs placement, routing, area and power analysis.

Beside the architectural exploration, place and route algorithm is also explored in VPR. Zhou et al. [79] have improved LUT-based FPGA routing by logic perturbation method. They have embedded Automatic Test Pattern Generation (ATPG) rewiring engine into VPR to reduce critical path delay. Wong and Wilton [80] have enhanced the placement and routing algorithm of VPR to provide better speed in SoC design, which contains non-rectangular embedded programmable logic cores. Chin and Wilton [81] have reduced the memory footprint of the routing step in VPR. The most memory intensive steps in the CAD flow are significantly improved while the running time increases because of the extra step to reduce memory.

In the year 2008, the latest version of VPR (VPR 5.0) was released [17, 82]. VPR 5.0 supports embedded blocks such as BRAMs and multipliers. Single driver routing is used instead of bi-directional driver to save routing area. Regression tests are included to check functionality and quality of result of the output of the tools.

In addition, VPR does not support power estimation, Lamoureux and Wilton of the University of British Columbia [83] have developed power-aware CAD flow in VPR based on the power model in [84, 85]. The CAD flow includes power-aware technology mapping and clustering algorithms with post place and route power analysis. Later, they examined various activity estimation techniques by employing a power-aware CAD tool [86]. The best estimation technique is adopted in the activity estimation tool called ACE-2.0. The tool is useful to calculate switching probabilities for every node in the circuit, which helps to estimate the dynamic power consumption of an FPGA design.

The coarse-grained embedded elements in most enhanced versions of VPR are arranged in column based fixed shape, like commercial devices [2, 3], EBs cannot be placed at any position in an FPGA. Also fast carry chain for fixed point addition is not supported. Addition is a common operation in any design; the critical path usually occurs in this operation. Modern devices use the fast carry chain to speed up addition. Without this feature in VPR, the timing analysis is not applicable to modern devices, which leads to wrong conclusions in architectural exploration. Hence, our tool Versatile Place and Route for Hybrid FPGA (VPH) enhances VPR 4.30 to support user constraints on position, area and aspect ratio of any embedded block for interface exploration. We have also modified the tool flow to support fast carry chain and our optimised routing architecture proposed in Chapter 5. The details of VPH tool flow are discussed in Chapter 3. The capabilities of different versions of VPR are summarised in Table 2.1. The table includes the original VPR 4.30, power-aware VPR by the University of British Columbia, Beauchamp's VPR for floating point unit in FPGAs, VPR 5.0 and our exploration tool VPH.

Table 2.1: Functions of various versions of VPR.
Remark: o - supports the feature, x - does not support the feature

	VPR version				
Functions	VPR 4.30 [16]	VPR 5.0 [82]	Power-aware VPR [85]	Beauchamp's VPR [54]	VPH [20]
Embedded block (EB)	x	Multiplier, memories	x	Multiplier, memories, multiply-add FPU, embedded shifter	Any embedded block
Carry chain	x	x	x	o	o
User constraint placement for EB (Chapter 4)	x	Only allow column EB	x	Only allow column EB	Any position specified by user (except overlapping of EB)
Power estimation	x	x	o	x	x
Single driver routing	x	o	x	x	x
Extra wires near EB (Chapter 5)	x	x	x	x	o
Routing switches inside EB (Chapter 5)	x	x	x	x	o

2.3.3 Post-fabrication Evaluation Tool

Measuring FPGA performance is crucial to the evaluation of different FPGA architectures. Timing and area analysis of commercial FPGAs such as Xilinx and Altera devices are carried out by using commercial tools. For example, a standard procedure to implement application circuits in Xilinx devices is to synthesise, place and route in Xilinx ISE, with timing and area analysis result output.

Although the tool VPR introduced in the last section is powerful enough to explore trade-offs between different architectural parameters, it is a relatively poor approximation to existing FPGAs. Therefore, direct comparison of a novel architecture to existing commercial FPGAs using VPR is not possible. Exploration of design space on the existing devices is possible by using a post-fabrication evaluation tool flow [73, 74].

Ho et al. [87] have proposed a methodology to model hybrid FPGA architectures using Virtual Embedded Blocks (VEBs) on commercial FPGAs, using commercial tools. This method creates dummy elements called virtual embedded blocks to model the size, position and delay of the embedded elements in hybrid FPGA. The design with VEBs is analysed using a standard commercial CAD tool. We adopt this methodology to evaluate our novel floating point architecture in Chapter 3.

The VEB methodology allows us to quantify the impact of embedding block on a modern FPGA using commercial CAD tool optimisations. This is in contrast to VPR-based methodologies which assume a bare-bone island-style FPGA (without carry chains and with a simplified routing architecture) and do not employ modern optimisations such as physical synthesis and retiming.

Figure 2.9 illustrates the modelling flow using the VEB methodology. The input is a high level application description and the output is an FPGA bitstream. The application is first broken into control logic and datapath portions manually because of a lack of suitable compiler.

The datapath portion is then mapped to the embedded blocks. The result of this step is a netlist containing black boxes representing those parts of the circuit that will be mapped to embedded blocks, and fine-grained logic elements representing those parts of the circuit that will be mapped to lookup-tables in the cases that no suitable embedded block is found or all have been used.

The basic strategy in VEB flow is to use selected logic resources of a commercial FPGA (called the *host* FPGA) to match the expected position, area and delay of an ASIC implementation of the coarse-grained units, as shown in Figure 2.10.

To estimate the area, an ASIC description of each coarse-grained block is synthesised using a comparable technology. For instance, 0.13μm technology is used in synthesising the ASIC block embedded in a Virtex-II device which in turn uses a 0.15μm process. Normalisation to the feature size is then applied to obtain a more accurate area estimation. The area of the coarse-grained block is then translated into equivalent fine-grained resources in the virtual FPGA.

In order to accurately model the delay, both the logic and the wiring delay of the virtual FPGA must match that of the host FPGA. The logic delay of the VEB can be matched by introducing delays in the FPGA resources. In the case of very small VEBs, it may not be possible to accurately match the

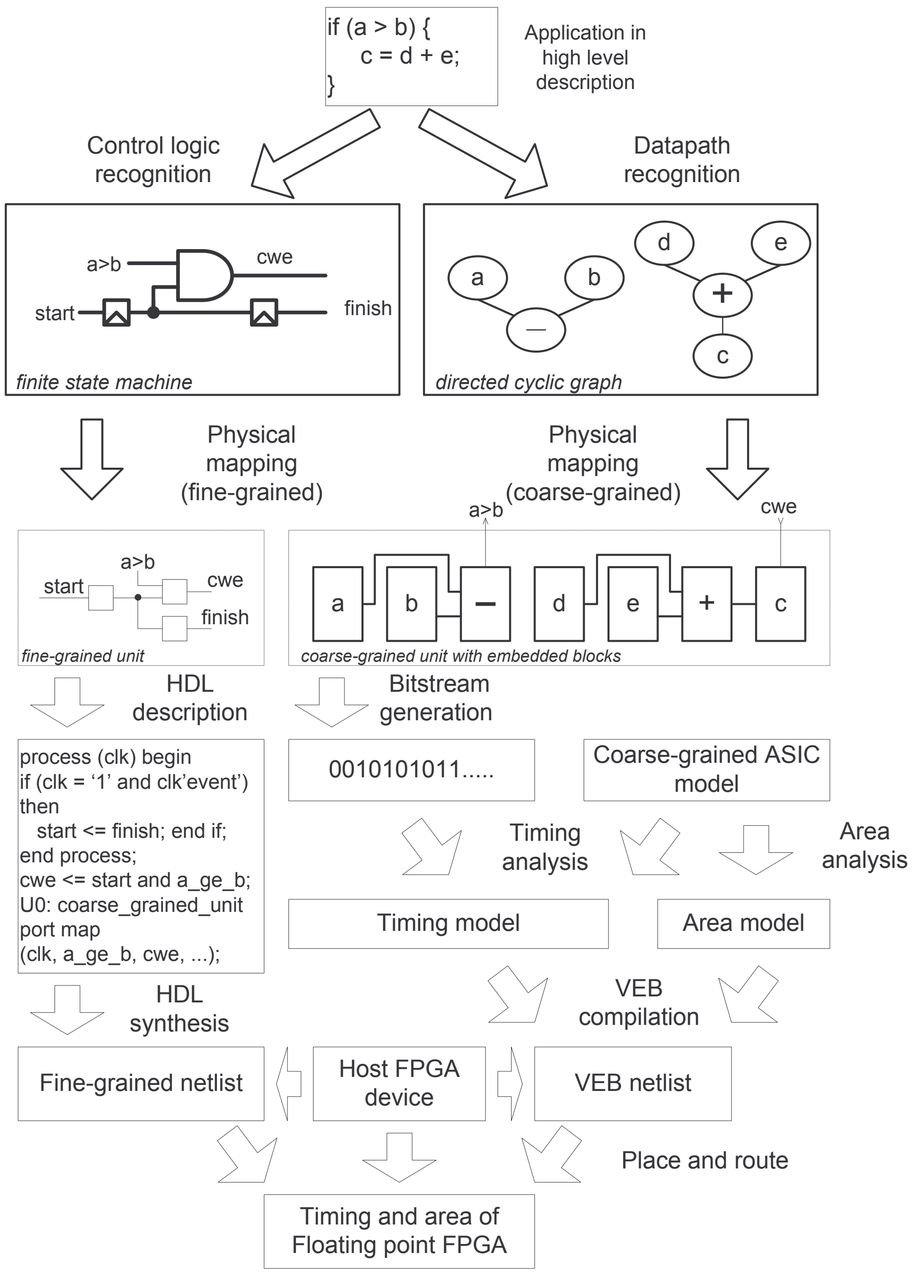


Figure 2.9: VEB modelling flow overview.

number of I/O pins, area or logic delay and it may result in inaccuracies. A complex coarse-grained unit might have many paths, each with different delays. In this case, it assumes that all delays are equal to the longest one (i.e. the critical path) as it is the most important characteristic of a coarse-grained unit in terms of timing.

To instantiate all the VEBs and connect all together, it describes the control logic and instantiates the VEBs explicitly and connects the signals between the fine-grained units and coarse-grained units. The design is then synthesised on the target device and a device-specific netlist is generated. The timing of the VEBs is also specified in the FPGA synthesis tool.

After generating the netlist of the overall circuit it uses the vendor's place and route tool to obtain the final area and timing results. This represents the characterisation of a circuit implemented on the hybrid FPGA with fine-grained units and routing resources exactly the same as the targeted FPGA.

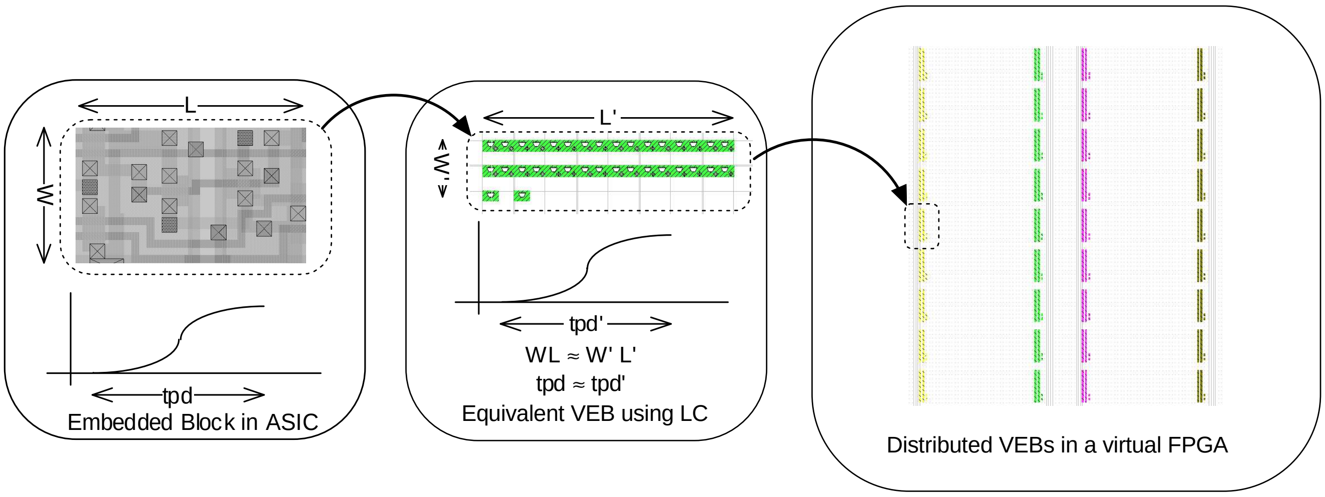


Figure 2.10: Modelling coarse-grained unit in FPGAs using Virtual Embedded Blocks.

By extending this methodology, estimation of power consumption of a novel FPGA compared to existing devices can be studied [62].

2.3.4 Analytical Modelling

Traditional FPGA architecture exploration performs numerous experiments repeatedly for each new architecture; these experiments are time consuming. Analytical modelling can be used to under-stand the relationships between architectural parameters in the early-stage FPGA architecture development [88]. The analytical model is a set of mathematical formulae relating parameters such as logic block cluster size, LUT size, number of inputs per cluster, switches and channel width to area, delay and routing demand [88,89,90,91,92]. By using the model, the design space can be searched quickly. After identifying the promising regions to study, the designer can use a traditional exploration work flow to examine precise architectural parameters. This significantly reduces the architecture design time.

2.4 General FPGA Applications

Many publications have reported that fined-grained parallelism based on FPGAs can result in outstanding performance over traditional general-purpose processors for fixed point computations. A variety of applications can be speeded up by using FPGAs. Our previous work [93] has used FPGAs to speed up DNA sequence alignment process by over 300 times compared to the one implemented in a high performance computer. Cryptography [94, 95, 96], the computation problem SAT [97, 98], medical [99, 100], finance [7, 101] and physics [102, 103] are speeded up by using FPGAs.

However, the performance of FPGA designs are still worse than ASICs [4]. Focusing the application in a particular domain can achieve further performance improvement.

2.5 Domain-specific Applications: Floating Point Applications

Although traditional FPGAs can speed up many general applications as discussed in last section, the performance of a particular application can be further improved by introducing ASIC blocks in FPGA. The ASIC block is specific to that application domain. Floating point applications are a good example of domain-specific applications which can be improved by using ASIC coarse-grained floating point units [54]. We introduce floating point arithmetic in Section 2.5.1, the floating point applications and benchmarks we use in this thesis in Section 2.5.3 and the optimisation method of floating point computation in Section 2.5.2.

2.5.1 Floating Point Arithmetic

The standard floating point numbering format is IEEE 754 [104, 105]. The floating point number is represented in the normalised formula and the base is 2:

$$fp = -1^{s}(1.f)2^{e-bias} \tag{2.1}$$

where s is sign bit, $1.f$ is positive mantissa (f is the fraction part and leading '1' is implicit in the representation), e is the exponent and bias is the constant exponent bias.

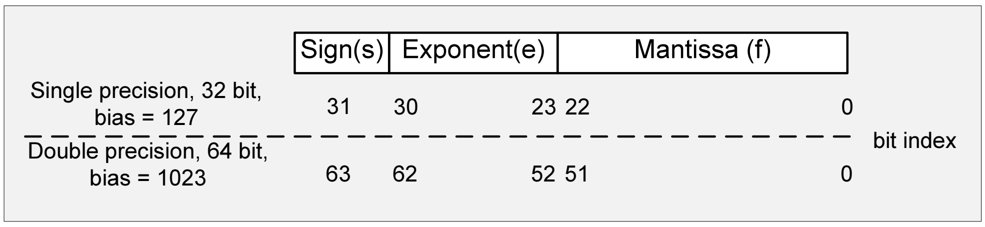

Sign(s)
Exponent(e)
Mantissa (f)
Single precision, 32 bit, bias = 127
31
30
23
22
0
Double precision, 64 bit, bias = 1023
63
62
52
51
0
bit index

Figure 2.11: IEEE floating point bit index and bias

In floating point computation, single precision (32 bit) and double precision (64 bit) are commonly used, where double precision floating point can represent a wider and more precise number range. The bit index and bias of single and double precision is shown in Figure 2.11.

Floating point addition/subtraction and multiplication require many operations as shown in Figure 2.12. In standard floating point addition/subtraction, five steps are required [106, 107, 108, 109]:

1. **Exponent comparator.** The exponents of two floating point numbers are compared, the difference between the exponents is the shifting value for mantissa denormalisation. The larger exponent is passed to final normalisation.
2. **Mantissa denormalisation.** The implicit bit '1' of $1.f$ is appended to the mantissas. The mantissa corresponding to the smaller exponent is shifted right by the difference between the two exponents. Now, the two mantissas are aligned and can be added or subtracted.
3. **Mantissa addition or subtraction.** The aligned mantissas are added or subtracted with carry output generated.

4. **Normalisation of mantissa and exponent.** The resulting mantissa and larger exponent are transformed into normalised format. The mantissa is shifted to the left by finding the leading ‘1’ and the exponent is subtracted by the shift amount.

5. **Rounding.** Rounding the mantissa after shifting.

In standard floating point multiplication, six steps are required [107, 108, 109]:

1. **XOR the sign bits.** XOR the sign bits of the floating point numbers to give the resulting sign bit.

2. **Exponent addition.** The exponents of two floating point numbers are added using a fixed point adder.

3. **Mantissa denormalisation.** The hidden bit ‘1’ of $1.f$ is appended to the mantissas.

4. **Mantissa multiplication.** The mantissas are multiplied using a fixed point multiplier.

5. **Normalisation of mantissa and exponent.** The resulting mantissa is shifted to left by finding the leading ‘1’ and the resulted exponent is subtracted by the shift amount. The final result is in normalised form.

6. **Rounding.** Rounding the mantissa after shifting.

The floating point adder/subtractor and multiplier in the floating point unit proposed in Chapter 3 are based on the above algorithms.

2.5.2 Optimisation of Floating Point Computation in FPGA

Floating point arithmetic requires many operations. These operations contribute to area, delay and power consumption in the FPGA. There are different optimisation methods for floating point computation. First, floating point representation is efficient when a large dynamic range is required. The bit-width of exponent and mantissa controls the range of numbers that can be represented. Longer

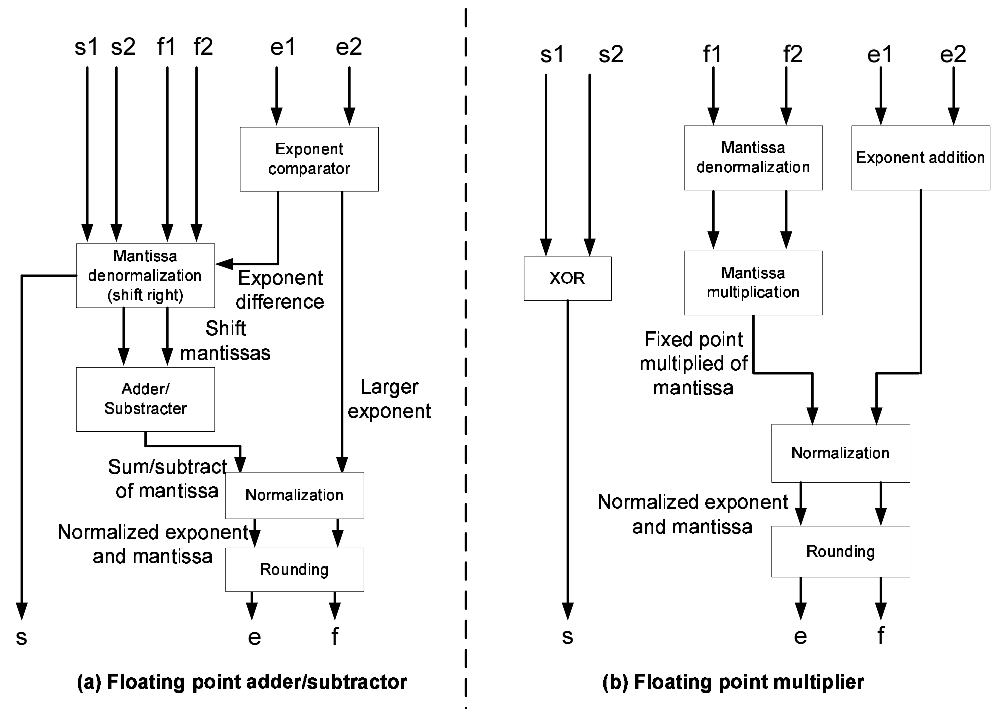


Figure 2.12: Algorithm of floating point adder/subtractor and multiplier

bit-width requires more logic resources such as priority encoders and variable shifters in the pre-normalisation and post-normalisation stage. The size of the fixed point adder and multiplier depends on the bit-width of exponent and mantissa. Applications needing less accuracy and precision can use shorter bit-width to reduce resource consumption [110, 111, 112, 113]. Second, the design of floating point units has been optimised. Besides standard floating point addition algorithm, the Leading-One-Predictor (LOP) algorithm [114] and 2-Path algorithm [115] have been proposed to simplify addition. Liang et al. [116] have designed a floating point generation tool for FPGAs to implement different floating point addition algorithms with 2 times latency improvement while consuming only half the FPGA logic area. Roesler and Nelson [53] have studied efficient ways to use FPGA features such as multipliers on the creation of floating point modules for various word widths.

Xilinx Core Generator can generate fast and complex floating point units for Xilinx FPGA devices [117] by using special features such as MULT18X18S in Spartan-III and Virtex-II FPGA, DSP48E (multiply and accumulate) in Virtex-4. Usselmann [108] has published open source floating point units for FPGAs. Ho [118] has heavily modified these open source FPUs to achieve higher improvement in speed and area. We adopt the floating point adder/subtractor and multiplier design modified by Ho in this thesis for the generation of novel floating point hybrid FPGA. Section 3.1 will give the details of the difference between the ASIC FPUs and fine-grained FPUs implemented in FPGA.

2.5.3 Applications and Benchmarks

Floating point applications are computationally intensive because of the complex floating point algorithm. FPGAs can improve the speed and area of the floating point core of these applications. LINPACK is a linear solver using gaussian elimination, which has been accelerated using a high level code transformation approach for FPGAs [119]. The speed of this approach is 6 times greater than the microprocessor. An FPGA is implemented as co-processor for the force pipeline for an N-body solver [102]. In this work, an object arithmetic library allows users to explore the trade-offs concerning precision, accuracy and speed from a high level description of the circuit.

Three Basic Linear Algebra Subroutine (BLAS) functions are examined [120], and the performance of the function implemented in FPGAs and reconfigurable computing platforms is compared to microprocessors. The FPGAs and reconfigurable platforms outperform modern microprocessors on memory bandwidth sensitive double precision floating point computation. The main limitation on the performance of FPGAs is FLoating point Operation Per Second (FLOPS) rather than memory band-width. Therefore, optimisation of the FPU can improve the overall performance.

In order to evaluate different systems, benchmarks are required. There is a set of software floating point benchmark such as SPEC CFP2006 [121] for profiling high performance computing applications. However, floating point benchmarks to evaluate hardware architecture are lacking. MCNC benchmarks are widely used for FPGA architecture evaluation [122], but they are different from floating point circuits with different operator logic, bus widths and pipeline arrangements. Synthetic floating point benchmark generation is one solution to the shortage of floating point benchmarks. Chau et al. [123] have developed a synthetic benchmark generator program for floating point circuits [124]. The generator can be used to produce floating point benchmarks with different user-specified properties similar to real computational programs such as BLAS and LINPACK.

In this thesis, we employ eight floating point benchmarks to evaluate the proposed floating point hybrid FPGA. They are *bfly*, *dscg*, *fir*, *mm3*, *ode*, *BGM*, *syn2* and *syn7*.

1. Butterfly (*bfly*)

The basic computation of Fast Fourier Transform (FFT):

$$z = y + x * w \tag{2.2}$$

where inputs and output are complex numbers. x and y are the inputs from previous stage and w is a twiddle factor. Each complex variable contains real part and imaginary part. FFT is an efficient algorithm to compute the Discrete Fourier Transform (DFT) for signal processing. Figure 2.13 is the datapath of *bfly*, where Re{} is the real part and Im{} is the imaginary part of the variables.

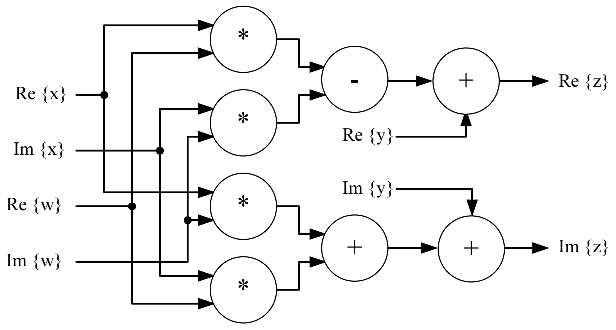


Figure 2.13: The circuit diagram of *bfly*

2. Digital Sin-Cosine Generator (*dscg*)

Digital sine-cosine generator [125] is used in wide range of digital communication applications. The circuit diagram of the generator in Figure 2.14 generates discrete values representative of either a sine wave or a cosine wave by implementing an equation in CORDIC algorithm [126]:

$$S1_{n+1} = cos(\theta)S1_n + (cos(\theta)+1)S2_n \tag{2.3}$$

$$S2_{n+1} = (cos(\theta)-1)S1_n + cos(\theta)S2_n \tag{2.4}$$

where $S1_n$ and $S2_n$ are the floating point outputs of the current samples, $S1_{n+1}$ and $S2_{n+1}$ are the next samples, and $\cos(\theta)$ is a constant floating point number.

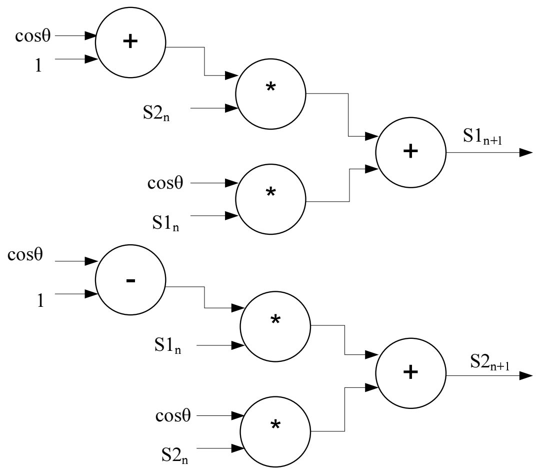


Figure 2.14: The circuit diagram of *dscg*

3. Finite Impulse Response Filter (*fir*)

A 4-tap finite impulse response filter is essential in digital filters for data sampling. The circuit in Figure 2.15 implements the equation:

$$y_i = \sum_{j=0}^{3} k_j x_{i-j} \tag{2.5}$$

where x_i is the input, k_i is the constant filter window and y_i is the output.

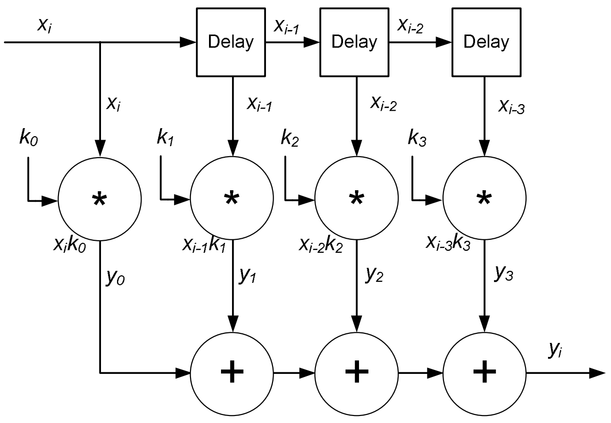


Figure 2.15: The circuit diagram of *fir*

4. Matrix Multiplier (*mm3*)

A 3x3 matrix multiplier is commonly used in scientific applications. The circuit in Figure 2.16 sequentially computes the vector dot-product:

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix} \tag{2.6}$$

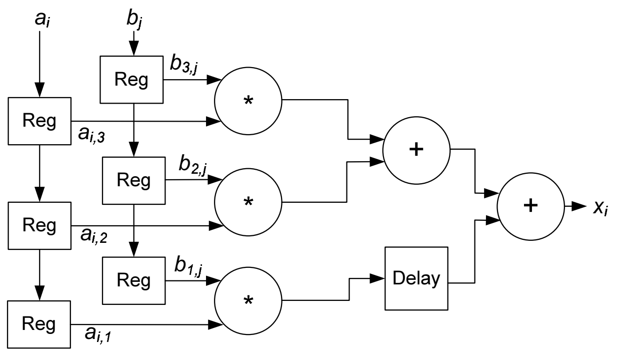


Figure 2.16: The circuit diagram of *mm3*

5. Ordinary Differential Equation (*ode*)

We choose an iterative *ode* solver (Figure 2.17) as an example to illustrate floating point computation. *ode* solver can be used to solve a simple ordinary differential equation:

$$\frac{dy}{dt} = \frac{t-y}{2}i \tag{2.7}$$

The trajectory of y is approximated by using the Euler method:

$$y_{k+1} = y_k + h\frac{t_k - y_k}{2} \tag{2.8}$$

$$t_{k+1} = t_k + h \tag{2.9}$$

where h is the step size to control the accuracy. Smaller h results in more accurate y.

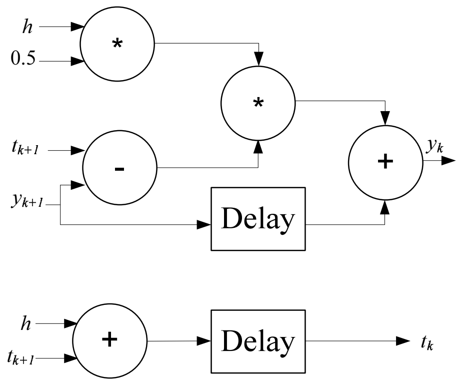


Figure 2.17: The circuit diagram of *ode*

6. BGM

Monte Carlo simulations of interest rate model derivatives in Brace, Gąatark and Musiela (BGM) framework has been sped up in FPGAs [7]. The datapath path in Figure 2.18 is the core of BGM computing the stochastic differential equation width d stochastic factors:

$$\frac{dF_n(t)}{F_n(t)} = \vec{\mu}_n(t)dt + \vec{\sigma}_n(t) \cdot d\vec{W}(t) \tag{2.10}$$

$$\vec{\mu}_n(t) = \vec{\sigma}_n(t) \cdot \sum_{i=m(t)}^{n} \frac{\tau_i F_i(t)\vec{\sigma}_i(t)}{1+\tau_i F_i(t)} \tag{2.11}$$

where $F_n(t)$ is the forward interest rate observed at time t for a period start at t_n and end at $t_n + 1$. dF_n is the change in the forward rate in time interval dt. $\vec{\mu}_n$ is the drift coefficient, where $m(t)$ is the index of the next reset date and $\vec{\sigma}_n$ is a 3-dimensional volatility vector. $d\vec{W}$ is a vector of uncorrelated Brownian motion.

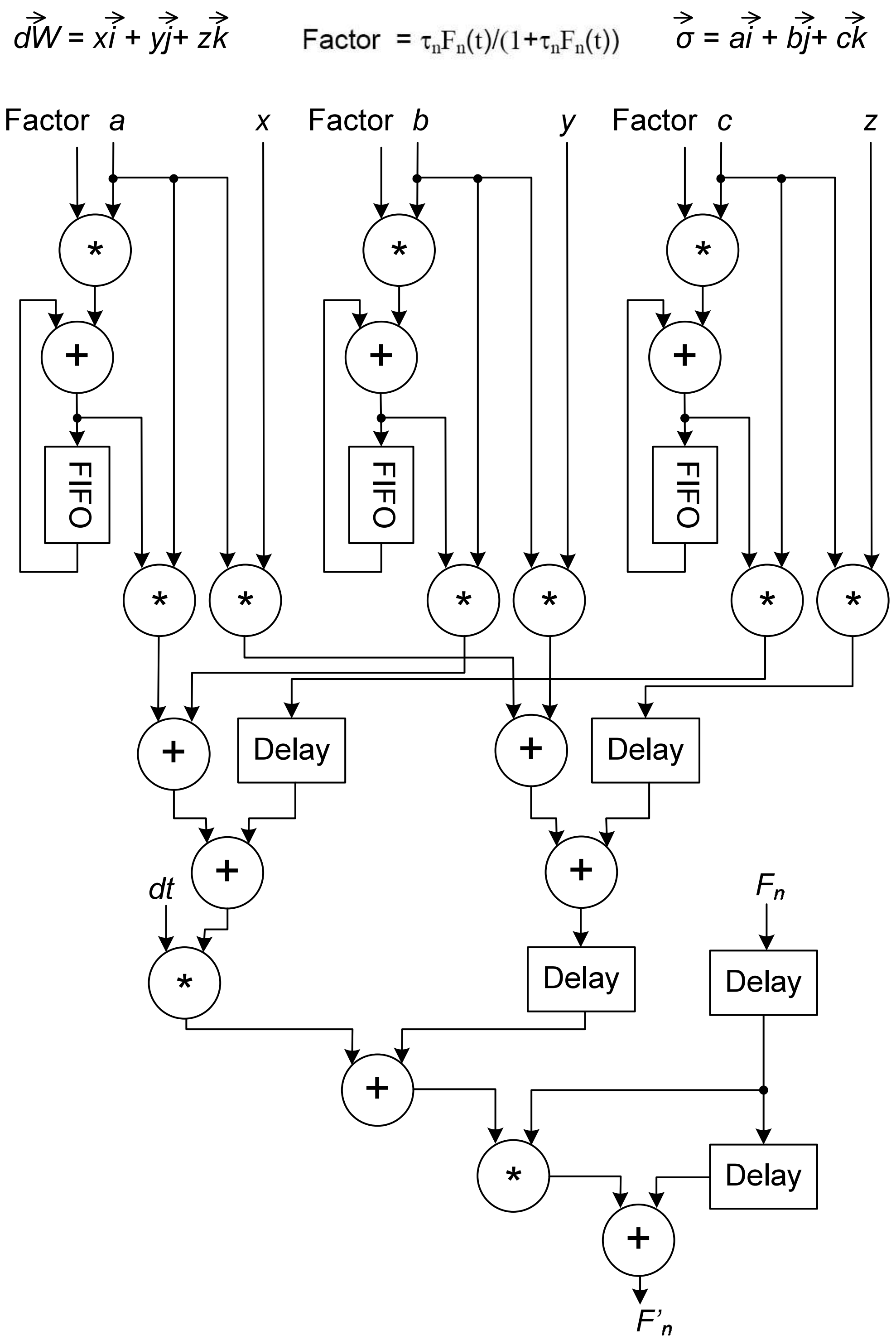

$\vec{dW} = x\vec{i} + y\vec{j} + z\vec{k}$
Factor = $\tau_n F_n(t)/(1+\tau_n F_n(t))$
$\vec{\sigma} = a\vec{i} + b\vec{j} + c\vec{k}$
Factor a
x
Factor b
y
Factor c
z
FIFO
FIFO
FIFO
Delay
Delay
dt
Delay
F_n
Delay
Delay
F'_n

Figure 2.18: The circuit diagram of *BGM*

7. Synthetic benchmark 2 (*syn2*)

A synthetic circuit contains 5 FAs and 4 FMs, generated by the synthetic benchmark circuit generator [123].

8. Synthetic benchmark 7 (*syn7*)

A synthetic circuit contains 25 FAs and 25 FMs, generated by the synthetic benchmark circuit generator [123].

Table 2.2 summarises the domain, nature and number of floating point adders/subtractors and multipliers of the eight benchmarks we used in this thesis.

Table 2.2: Summary of the benchmarks we used in this thesis

Benchmarks	Domain	Nature	no. of Add/Sub	no. of Mul
dscg	DSP	kernel	2	4
bfly	DSP	kernel	4	4
ode	DSP	kernel	3	2
mm3	Linear Algebra	kernel	2	3
fir	Linear Algebra	kernel	3	4
bgm	Finance	application	9	11
syn2	N/A	synthetic	5	4
syn7	N/A	synthetic	25	25

2.6 Processing Platforms

Parallelism is essential to speed up high performance computing applications. FPGAs can be used to perform large amounts of parallel computation. In this section, we introduce and compare processing platforms including FPGA, CPU and GPU for floating point computation.

2.6.1 CPU

Central Processing Units (CPUs) were single core when first invented. Since transistor sizes have reduced greatly in recent years, multi-core CPUs such as Intel Core2 [127] have been developed. The multi-core CPUs use the shared memory for communication, and are synchronised through shared cache.

Each core controls a thread (or two threads for hyper threading in some CPUs) at a time. The status of each thread is stored in a set of registers. A specific instruction set is used to operate the computation in CPU. Fetch, decode and execute of instructions are the main execution cycle of CPU. A management and scheduling unit in CPU is used for branch prediction, instruction ordering and execution [11]. Although the clock rate of CPU is high, memory access and execution cycles are the bottlenecks.

2.6.2 GPU

A Graphics Processing Unit (GPU) [8] is a special processor that accelerates 3D graphics for microprocessors or CPUs with high memory bandwidth. It traditionally resides on a graphics card such as NVIDIA GeForce series [128] dedicated to floating point operations. The GPU does not have the scheduling logic and caches; instead, most of its area is used for floating point units which include texture, scalar and vector processors for graphics computations [129]. This can achieve massive instruction-level parallelism and reduce memory latency. Also, thread-level parallelism is used to hide latency. Several warps execute in batches of threads, allowing each thread to execute different parts of program. However it is costly because if less threads within a warp are active, fewer parallel operations execute per cycle [11]. Besides graphics computation, the FPUs in a GPU can be used to accelerate floating point applications [9, 10, 130], captured in languages for GPUs such as CUDA [131].

2.6.3 Comparison: FPGA, CPU and GPU

FPGA, CPU and GPU have different advantages. There are many studies and comparisons between FPGA, CPU and GPU in speed and power for floating point application [10, 11, 120, 132]. CPU and GPU must share system memory, where memory access limits the speed. Also the fetch, decode and execute cycle of the instructions in CPU and GPU cannot fully utilise the high clock rate. Although GPUs are very powerful for floating point computations, they must be connected to CPUs as co-processors, and cannot work as a stand alone system.

FPGAs are limited in processing power for complex floating point computation, but they can be faster than CPU [10, 120, 132] in the floating point domain. They have advantages in low power consumption [11], high reconfigurability for system development, and massive parallelism. FPGAs do not require a specific instruction set for computation and do not need to connect to a CPU, and can be built as a stand alone device. In this thesis, we employ floating point arithmetic as an example, showing that embedding domain-specific coarse-grained units in FPGAs can increase their processing power. This domain-specific hybrid FPGA maintains the advantages of the traditional fine-grained FPGA while increasing their performance, closing the gap between FPGA and other technologies. Table 2.3 summarises the features, advantages and disadvantages of each device, including our proposed Floating Point FPGA (FPFPGA).

Table 2.3: The features, advantages and disadvantages of different devices in Floating Point (FP) domain

	Devices			
Comparison	FPGA	CPU	GPU	Proposed FPFPGA (expected performance)
Speed of FP [10, 120, 132]	medium	low	high	FPGA $<$ FPFPGA $<$ GPU [12]
Limitation on speed	speed & no. of fine-grained FPU, reconfigurable routing	no. of integrated FPUs, scheduling of floating point instruction	thread scheduling, high latency (cannot be real time)	speed & no. of coarse-grained FPU, reconfigurable routing
Power consumption [11]	low	high	high	FPFPGA $<$ FPGA [62]
Instruction execution cycle	no fixed instruction-set architecture	specific instruction-set (fetch-decode-execute cycle)	specific instruction-set (fetch-decode-execute cycle)	no fixed instruction-set architecture
Parallelism	high (depends on the size of FPGA to hold logic circuit)	low	high (depends on the thread scheduling)	high (depends on the size of FPGA to hold logic circuit)
Reconfigurability	high	low	low	FPFPGA $<$ FPGA (coarse-grained FPU is less configurable)
Others	rapid prototyping for developing system (can be stand alone)	need to connect to shared system memory	need to connect to shared system memory and CPU	rapid prototyping for developing system (can be stand alone)

2.7 Summary

In this chapter, we first introduce traditional and hybrid FPGA architectures. Then we present the CAD tools used to explore hybrid FPGA architectures and the features of our CAD tools. We also introduce applications that can be accelerated by FPGAs. Next, we describe the floating point benchmarks used in this thesis to examine the performance of FPFPGAs. Finally, we compare the performance of different devices for floating point computation.

Chapter 3

Architecture and Modelling

FPGA technology has been widely adopted to speed up computationally intensive applications. Most current FPGA devices employ an island-style fine-grained architecture [23], with additional fixed-function heterogeneous blocks such as multipliers and block RAMs; these have been shown to have severe area penalties compared with ASICs [1, 4]. In Section 3.1, we propose an architecture for FPGAs which is optimised for floating point (FP) applications as an example of a hybrid FPGA. Such devices could be used for applications in digital signal processing, control, high performance computing and other applications where the large dynamic range, convenience, and ease of verification compared with traditional fixed point designs on conventional FPGAs. The result of the FPFPGA is compared with existing FPGAs. The inclusion of ASIC floating point multipiers and adders/subtractors in a synthesisable datapath-oriented embedded FPGA fabric to facilitate FP computation is different from the previous works [56, 59], which contain fixed point computation only.

We are also interested in studying what are the trade-offs when adopting hybrid FPGA architecture with different parameters such as routing structure and interface between different elements. A tool is needed to explore different architectures. Commercial tools based on existing devices are not feasible for various architectural parameters such as I/O pin position of coarse-grained blocks and channel width. Section 3.2 defines the baseline architecture for pre-fabrication architectural exploration using VPH in later chapters. Section 3.3 presents a tool called VPH to evaluate the hybrid FPGA.

3.1 Novel Floating Point Hybrid FPGA Architecture (FPFPGA) and Post-fabrication Exploration

This section presents an architecture for a reconfigurable device which is specifically optimised for floating point applications. Fine-grained units are used for implementing control logic and bit-oriented operations, while parameterised and reconfigurable word-based coarse-grained units incorporating word-oriented lookup tables and floating point operations are used to implement datapaths. In order to facilitate comparison with existing FPGA devices, the Virtual Embedded Block (VEB) scheme is employed to model embedded blocks using existing FPGA tools. This methodology involves adopting existing FPGA resources to model the size, position and delay of the embedded elements. The standard design flow offered by FPGA and CAD vendors is then applied and static timing analysis can be used to estimate the performance of the FPGA with the embedded blocks.

3.1.1 Requirements

Before we introduce the FPFPGA architecture, common characteristics of what we consider a reasonably large class of FP applications which might be suitable for signal processing, linear algebra and simulation are first described. Although the following analysis is qualitative, it is possible to develop the architecture in a quantitative fashion by profiling application circuits in a specific domain.

In general, FPGA-based FP application circuits can be divided into control and datapath portions. The datapath typically contains floating point operators such as adders, subtractors, multipliers, and occasionally square root and division operations. The datapath often occupies most of the area in an implementation of the application. Existing FPGA devices are not optimised for FP computations and for this reason, FP operators consume a significant amount of FPGA resources. For instance, if the embedded DSP48 blocks are not used, a double precision floating point adder requires 701 slices on a Xilinx Virtex-4 FPGA, while a double precision floating point multiplier requires 1238 slices on the same device [117].

The floating point precision is usually a constant within an application. The IEEE 754 single precision format (32 bit) or double precision format (64 bit) is commonly used.

The datapath can often be pipelined and connections within the datapath may be uni-directional in nature. Occasionally there is feedback in the datapath for some operations such as accumulation. The control circuit is usually much simpler than the datapath and therefore the area consumption is typically lower. Control is usually implemented as a finite state machine and most FPGA synthesis tools can produce an efficient mapping from the boolean logic of the state machine into fine-grained FPGA resources.

Based on the above analysis, some basic requirements for FPFPGA architectures can be derived.

1. A number of coarse-grained floating point addition and multiplication blocks are necessary since most computations are based on these primitive operations. Floating point division and square root operators can be optional, depending on the domain-specific requirement.

2. Coarse-grained interconnection, fabric and bus-based operations are required to allow efficient implementation and interconnection between fixed-function operators.

3. Dedicated output registers for storing floating point values are required to support pipelining.

4. Fine-grained units and suitable interconnections are required to support implementation of state machines and bit-oriented operations. These fine-grained units should be accessible by the coarse-grained units and vice versa.

The basic requirements will be addressed by the novel architecture proposed in Section 3.1.2. The performance of the design will be quantified by using the methodology in Section 3.1.3 and the result of the comparison with existing devices will be covered in Section 3.1.4.

3.1.2 FPFPGA Architecture

This section presents the design of our FPFPGA architecture to fulfill the requirements in the Section 3.1.1. Figure 3.1 shows a top-level block diagram of the FPFPGA architecture. It employs an island-style fine-grained FPGA structure (Figure 2.1) with dedicated columns for coarse-grained units. Both fine-grained and coarse-grained units are reconfigurable. The coarse-grained part contains embedded fixed-function floating point adders and multipliers. The connection between coarse-grained units and fine-grained units is similar to the connection between embedded blocks (embedded DSP block or block RAM) and fine-grained units in existing FPGA devices.

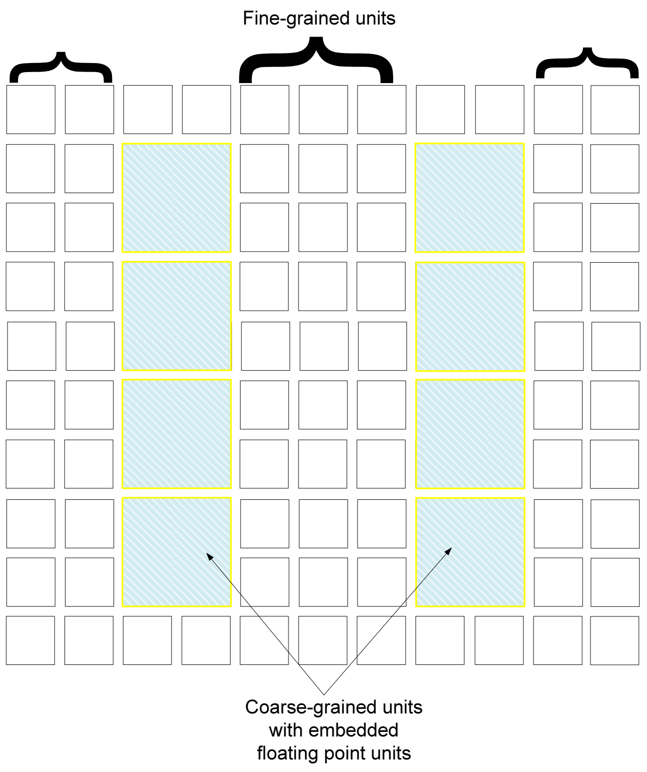


Figure 3.1: Architecture of the FPFPGA

Matching of the Requirements 1 and 2

The previous works [56, 59] have proposed a fixed point uni-directional bus-based synthesisable datapath-oriented fabric in the FPGA. The coarse-grained logic architecture in this thesis is optimised to implement the datapath portion of floating point applications. The architecture of each block, inspired by their work, is shown in Figure 3.2. Each block consists of a set of floating point multipliers, adders/subtractors, and general-purpose bitblocks connected using a uni-directional bus-based interconnect architecture. Each of these blocks will be discussed in this section. The main difference from the work [56, 59] is the inclusion of ASIC floating point multipliers and adders/subtractors in a synthesisable datapath-oriented embedded FPGA fabric to facilitate FP computation rather than fixed

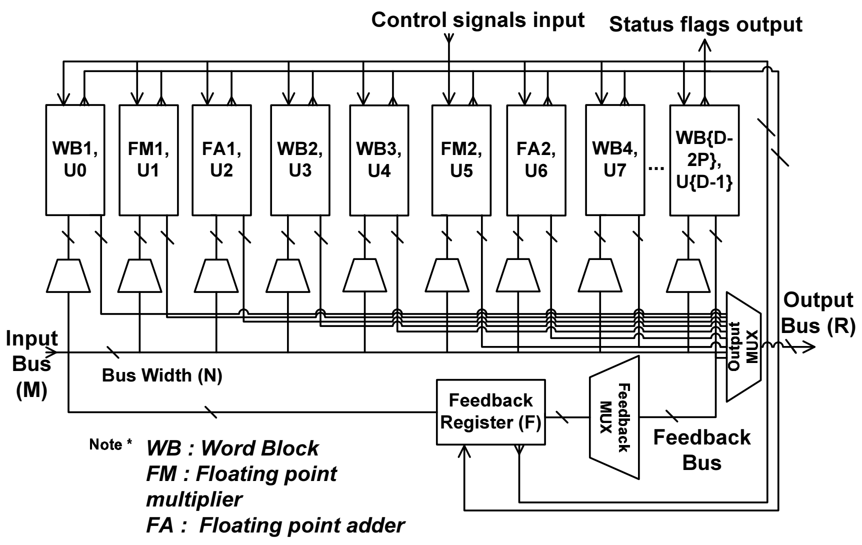


Figure 3.2: Architecture of the coarse-grained unit with different parameters

point computational blocks only. The bus-based routing in this embedded fabric provides benefits to the density. We optimise this architecture based on the FP applications. To keep our discussion general, we have parameterised the architecture as shown in Table 3.1. There are D subblocks in each coarse-grained block. P_m of these D subblocks are floating point multipliers, another P_a of them are floating point adders and the rest $(D - P_m - P_a)$ are general-purpose wordblocks. Specific values of these parameters will be given in Section 3.1.3.

Table 3.1: Parameters for the coarse-grained unit

Symbol	Parameter Description
D	Number of blocks (Including FPUs, wordblocks)
N	Bus Width
M	Number of Input Buses
R	Number of Output Buses
F	Number of Feedback Paths
P_a	Number of Floating Point Adders
P_m	Number of Floating Point Multipliers
B	Number of bitblocks in a wordblock

The core of each coarse-grained block contains P_m multiplier and P_a adder/subtractor subblocks. Each of these blocks has a reconfigurable registered output, and associated control input and status output signals. The control signal is a write enable signal that controls the output register. The status signals report the subblock's status flags and include those defined in IEEE standard as well as a zero and sign flag. The fine-grained unit can monitor these flags via the routing paths between them.

Each coarse-grained block also contains general-purpose wordblocks. Each wordblock contains B identical bitblocks, and is similar to the design in [56]. A bitblock contains two 4-input LUTs and a reconfigurable output register. The value of N depends on the bit-width of the coarse-grained block. Bitblocks within a wordblock are all controlled by the same set of configuration bits, so all bitblocks within a wordblock perform the same function. A wordblock, which includes registers, can efficiently implement operations such as fixed point addition and multiplexing. Like the multiplier and adder/subtractor blocks, wordblocks generate status flags such as most-significant bit (MSB), least-significant bit (LSB), carry out, overflow and zero; these signals can be connected to the fine-grained units.

Matching of the Requirements 3 and 4

Apart from the control and status signals, there are M input buses and R output buses connected to the fine-grained units. Each subblock can only accept inputs from the left, simplifying the routing. To allow more flexibility, F feedback registers have been employed so that a block can accept the output from the right block through the feedback registers. For example, the first block can only accept input from input buses and feedback registers, while the second block can accept input from input buses, the feedback registers and the output of the first block. Each floating point multiplier is logically located to the left of a floating point adder so that no feedback register is required to support multiply-and-add operations. The coarse-grained units can support multiply-accumulate functions by utilising the feedback registers. The bus width of the coarse-grained units is 32 bit for the single precision FPFPGA and 64 bit for double precision.

Switches in the coarse-grained unit are implemented using multiplexers and are bus-oriented. A single set of configuration bits is required to control each multiplexer, improving density compared to a fine-grained fabric.

3.1.3 Methodology: Post-fabrication Modelling of FPFPGA

To model the mapping of our benchmark circuits on the architecture described in Section 3.1.2, we employ the post-fabrication methodology: Virtual Embedded Block (VEB) introduced in Chapter 2, Section 2.3.3 to quantify the impact of FPFPGA on existing FPGA.

To adopt this methodology, we follow the VEB work flow in Figure 2.9, where area and delay models for the coarse-grained units are required. To estimate the area, we synthesise an ASIC description of each coarse-grained block in $0.13\mu m$ process technology. We normalise the feature size of the ASIC block to $0.15\mu m$ process Virtex-II device to obtain a more accurate area estimation. We employ a parameterised synthesisable IEEE 754 compliant floating point library [133] where floating point multiplier and adder are based on the work in [108]. The corresponding floating point multiplier and adder have been described in Chapter 2, Section 2.5.2. The library supports four rounding modes and denormalised numbers. A floating point multiplier and floating point adder are generated and synthesised using a regular standard cell library flow.

The normalised area of the coarse-grained block is then translated into equivalent CLB resources in the virtual FPGA. In order to make this translation, an estimate of the area of a CLB in the FPGA is required, where a CLB refers to a cluster of two 4-input lookup tables and two associated output registers similar to the one described in Figure 2.2. The area estimation includes the associated routing resources and configuration bits. All area measures are normalised by dividing the actual area by the square of the feature size, making them independent of feature size. VEB utilisation can then be computed as the normalised area of the coarse-grained unit divided by the normalised area of a CLB. This value is in units of equivalent CLBs, and the mapping enables modelling of coarse-grained units using existing FPGA resources. In addition, special consideration is given to the interface between the CLBs and the VEB to ensure that the corresponding VEBs has sufficient I/O pins to connect to the routing resources. This can be verified by keeping track of the number of inputs and outputs which connect to the global routing resources in a CLB. For example, if a CLB only has two outputs, it is not possible to have a VEB with an area of four CLBs that requires nine outputs. For such a case, the area is increased to five CLBs. In our implementation, area matching is achieved by creating a dedicated scan-chain using shift registers. A longer scan-chain consumes more CLBs and therefore the VEB is larger.

The timing of VEB is obtained in the ASIC synthesis stage. There are many options available to match the timing of a VEB to the target Virtex-II device. We utilise the fast carry-chains presented in most FPGAs to generate delays that emulate the critical path in a VEB. This choice has the added advantage that relocation of CLBs on the FPGA does not affect the timing of this circuit. A manual placement is applied to ensure that the placement of each VEB is aligned on dedicated columns while maintaining nearest displacement.

It is important to note that timing information cannot be determined before programming the configuration bits of the coarse-grained unit. Otherwise, the tool reports the worst case scenario where the longest combinational path from the first wordblock to the last wordblock is considered as the critical path and this is usually not the correct timing in most designs. To address this issue, the tool has to recognise the configuration of the coarse-grained unit before the timing analysis. Therefore, a set of configurations is generated during manual mapping, and the associated bitstream can be used in timing analysis. This bitstream can be imported to the timing analysis tool, so the tool can identify false paths during timing analysis and produce correct timing for that particular configuration.

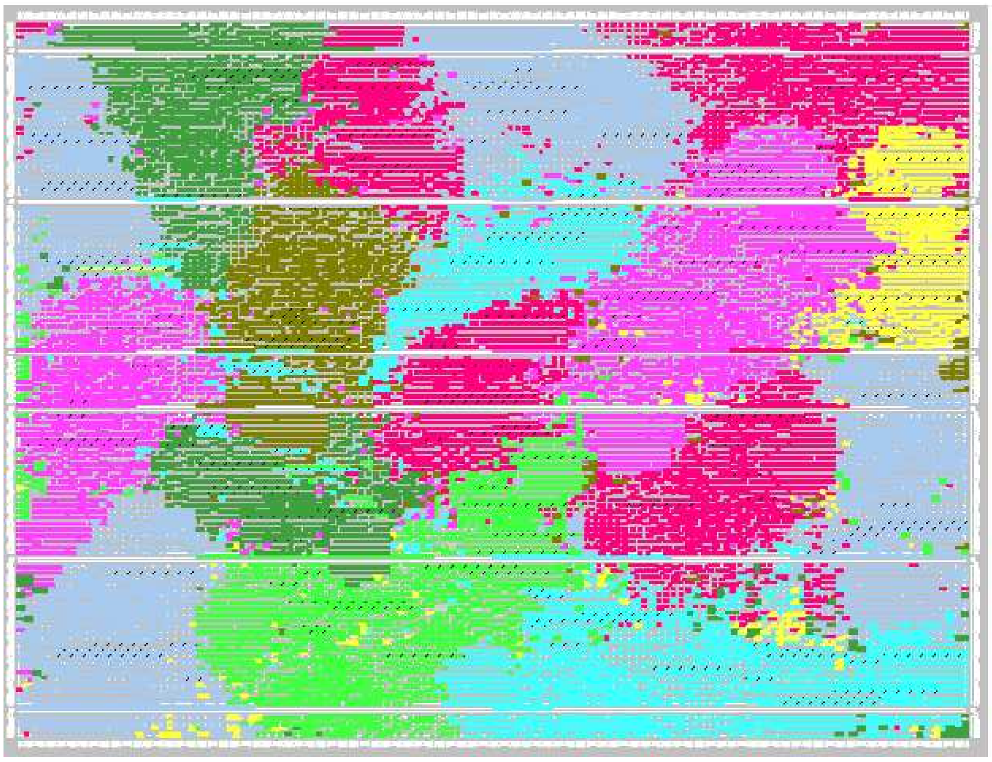
3.1.4 Results

Table 3.2: Normalisation on the area of the coarse-grained units against a Virtex-II CLB (SP - single precision, DP - double precision, and CGU - coarse-grained unit. 15% overheads and 70% of fine-grained routing resources have already been applied on the coarse-grained units area.)

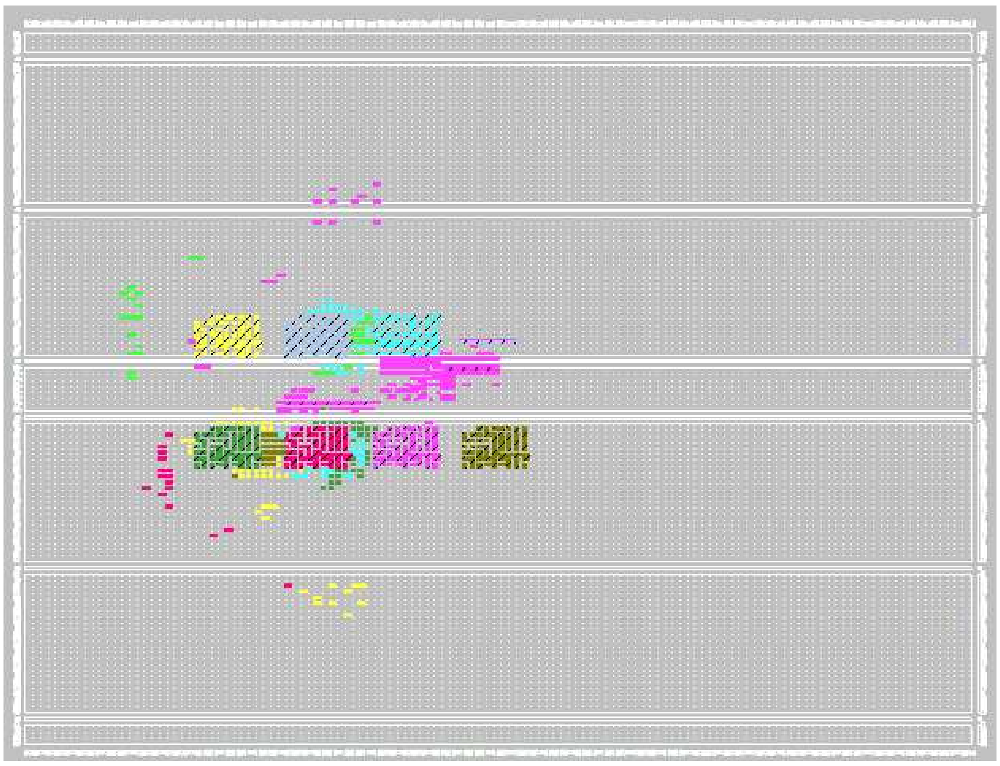
Fabric	Area (A)(μm^2)	Feature Size(L)(μm)	Normalised Area(A/L^2)	Area in CLB	Input Pin	Output Pin	Max. Delay(ns)
Virtex-II CLB	10,912	0.15	484,978	1	8(8)	4(4)	0.45
SP-CGU	848,040	0.13	50,179,876	104	157 (832)	162(416)	3.35
DP-CGU	1,743,560	0.13	103,169,254	214	285 (1,712)	258(856)	4.25

In this section, we present an evaluation of our architecture. The flow described in the previous section is employed.

The best-fit architecture can be determined by varying the parameters to produce a design with maximum density over the benchmark circuits. Additional wordblocks are included, allowing more flexibility for implementing circuits outside of the benchmark set. Manual mappings are performed for



(a) Virtex-II XC2V3000. The circuit consumes 100% of chip area



(b) FPFPGA. Coarse-grained units are identified by tightly packed CLBs in a rectangular region. The circuit consumes 5% of chip area

Figure 3.3: Floorplan of the single precision *bgm* circuit on Virtex-II FPGA and FPFPGA (Area is significantly reduced by introducing coarse-grained units.)

Table 3.3: FPFPGA implementation results
(Values in the brackets indicate the percentages of CLB used in corresponding FPGA device. CGU stands for coarse-grained unit and FGU stands for fine-grained unit.)

	Single precision FPFPGA					XC2V3000-6-FF1152				Reduction	
Circuit	number of CGU	CGU area (CLB)	FGU area (CLB)	Total Area (CLB)	Delay (ns)	FPU area (CLB)	Logic area (CLB)	Total Area (CLB)	Delay (ns)	Area (times)	Delay (times)
bfly	2	208 (1.5%)	106 (0.74%)	314 (2.2%)	2.92	5,839 (41%)	494 (3.4%)	6,333 (44%)	11.6	20.2	3.99
dscg	2	208 (1.5%)	176 (1.23%)	384 (2.7%)	2.92	4,419 (31%)	203 (1.4%)	4,622 (32%)	11.3	12.0	3.88
fir	2	208 (1.5%)	7 (0.05%)	215 (1.5%)	3.20	5,059 (35%)	109 (0.8%)	5,168 (36%)	11.2	24.0	3.51
mm3	2	208 (1.5%)	134 (0.93%)	342 (2.4%)	3.86	4,002 (28%)	505 (3.5%)	4,507 (31%)	11.8	13.2	3.06
ode	2	208 (1.5%)	19 (0.13%)	227 (1.6%)	3.24	3,329 (23%)	142 (1.0%)	3,471 (24%)	11.1	15.3	3.44
bgm	7	728 (5.1%)	323 (2.25%)	1,051 (7.3%)	4.52	13,928 (97%)	406 (2.8%)	14,334 (100%)	13.9	13.6	3.08
syn2	3	312 (2.2%)	0 (0.0%)	312 (2.2%)	2.93	5,983 (42%)	0 (0.0%)	5,977 (42%)	11.4	19.2	3.90
syn7*	16	1,664 (11.6%)	0 (0.0%)	1,664 (11.6%)	2.93	30,656 (214%)	0 (0.0%)	30,625 (214%)	13.1	18.4	4.47
									Geometric Mean:	16.6	3.64

(a) Single precision FPFPGA results. *Circuit *syn7* cannot be fitted in a XC2V3000-6 device. The area and the delay are obtained by implementing on a XC2V8000-5 device

	Double precision FPFPGA					XC2V6000-6-FF1152				Reduction	
Circuit	number of CGU	CGU area (CLB)	FGU area (CLB)	Total Area (CLB)	Delay (ns)	FPU area (CLB)	Logic area (CLB)	Total Area (CLB)	Delay (ns)	Area (times)	Delay (times)
bfly	2	428 (1.3%)	201 (0.59%)	629 (1.9%)	4.42	13,653 (40%)	963 (2.9%)	14,616 (43%)	21.7	23.2	4.91
dscg	2	428 (1.3%)	363 (1.07%)	791 (2.3%)	4.45	8,984 (27%)	202 (0.6%)	9,186 (27%)	17.3	11.6	3.89
fir	2	428 (1.3%)	6 (0.02%)	434 (1.3%)	4.38	10,310 (30%)	165 (0.5%)	10,310 (31%)	18.0	23.8	4.11
mm3	2	428 (1.3%)	229 (0.68%)	657 (1.9%)	4.25	7,529 (22%)	727 (2.2%)	8,256 (24%)	17.1	12.6	4.03
ode	2	428 (1.3%)	22 (0.07%)	450 (1.3%)	4.27	6,794 (20%)	239 (0.7%)	7,033 (21%)	18.6	15.6	4.35
bgm	7	1,498 (4.4%)	321 (0.95%)	1,819 (5.4%)	4.55	32,918 (97%)	199 (0.6%)	33,117 (98%)	22.0	18.2	4.84
syn2	3	642 (1.9%)	0 (0%)	642 (1.9%)	4.47	12,016 (36%)	0 (0%)	12,016 (36%)	19.0	18.7	4.26
									Geometric Mean:	17.1	4.33

(b) Double precision FPFPGA results. Circuit *syn7* is omitted since it cannot be fitted on any Virtex-II FPGA device each benchmark. A more in-depth analysis on how those parameters affect the application performance is on-going work.

For the single precision FPFPGA device, a Xilinx XC2V3000-6-FF1152 FPGA is used as the host and we assume 16 coarse-grained units. We emphasise that the parameter settings chosen for the coarse-grained block is fixed over the entire set of benchmarks, each coarse-grained unit having nine subblocks ($D = 9$), four input buses ($M = 4$), three output buses ($R = 3$), three feedback registers ($F = 3$), two floating point adders and two floating point multipliers ($P_a = 2$, $P_m = 2$). Number of bitblocks in each wordblock is the same as bus width ($B = N$). We assume that the two floating point multipliers in the coarse-grained unit are located at the second and the sixth subblock. The two floating point adders are located in the third and the seventh subblock. This architecture is shown in Figure 3.4.

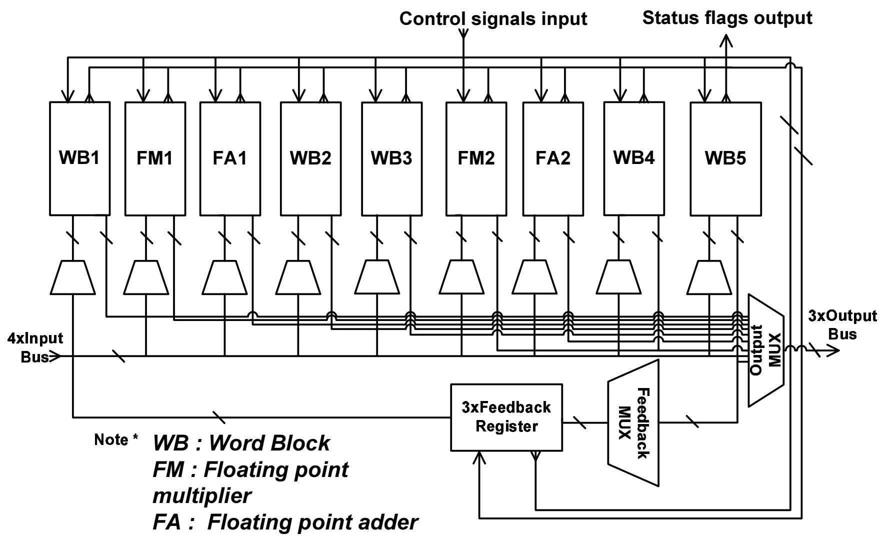


Figure 3.4: Architecture of the coarse-grained unit with appropriate parameter settings

The coarse-grained blocks constitute 7% of the total area of an XC2V3000 device. All FPGA results are obtained using Synplicity Synplify Premier 9.0 for synthesis and Xilinx ISE 9.2i design tools for place and route. All ASIC results are obtained using Synopsys Design Compiler V-2008.09 and default optimisation setting for area and delay.

The physical die area and photomicrograph of a Virtex-II device has been reported [134], and the normalisation of the area of coarse-grained unit is estimated in Table 3.2. From inspection of the die photo, we estimate that 60% of the total die area is used for CLBs.

This means that the area of a Virtex-II CLB is 10,912μm^2. We assume each CLB contains two LUTs and two FFs, which is similar to a slice in Virtex-II [135]. This number is normalised against the feature size (0.15μm). A similar calculation is used for the coarse-grained units. The ASIC synthesis tool reports that the area of a single precision coarse-grained unit is 433,780μm^2. We further assume 15% overhead after place and route the design based on our experience [56]. The area of unit does not include the routing tracks for fine-grained routing in FPGA. The routing wires consist of over 70% of total FPGA area [59] for large channel widths. Therefore, we add 70% extra area to the coarse-grained units for the vertical and horizontal routing tracks. The area values are normalised against the feature size (0.13μm). The number of equivalent CLB is obtained through the division of coarse-grained unit area by slice area. This shows that single precision coarse-grained unit is equivalent to 104 CLBs. Simply assuming each CLB has four outputs, the VEB allow maximum of 416 output pins while the coarse-grained unit consumes 162 output pins only. Therefore, we do not need to further adjust the area.

Single precision FPFPGA results are shown in Table 3.3(a). A comparison between the floorplan of the Virtex-II device and the floorplan of the FPFPGA on *bgm* circuit is illustrated in Figure 3.3.

The FPU implementation on FPGA is based on the work in [108] and described in Chapter 2, Section 2.5.2. This implementation supports denormalised floating point numbers which are required in the comparison with the FPFPGA. The FPU area for the XC2V3000 device (seventh column) is estimated from the distribution of LUTs, which is reported by the FPGA synthesis tool. The logic area (eighth column) is obtained by subtracting the FPU area from the total area reported by the place and route tool. As expected, FPU logic occupies most of the area, typically more than 90% of the user circuits. While the *syn7* circuit cannot fit in an XC2V3000 device, it can be tightly packed into a few coarse-grained blocks. The circuit *syn7* has 50 FAs and FMs which consume 214% of the total FPGA area. They can fit into 16 coarse-grained units, which constitute just 11.6% of the total FPGA area. Note that it is the minimum fine-grained resources area, we have not accounted the area of embedded fixed point multiplier in Virtex-II device as we do not have information of the multiplier area. If we account for the usage of embedded multiplier, the improvement of FPFPGA should be slightly greater.

Similar experiments for double precision floating point applications have been conducted and the results are reported in Table 3.3(b). In double precision floating point FPFPGA, we use the XC2V6000 FPGA as the host FPGA and the comparison is done on the same device.

For both single and double precision benchmark circuits, the proposed architecture reduces the area by a factor of 17 on average, a significant reduction. The saving is achieved by (1) embedded floating point operators, (2) efficient directional routing and (3) sharing configuration bits. On larger circuits, or on circuits with a smaller ratio of floating point operations to random logic, the improvement will be less significant. However, the reported ratio gives an indication of the possible improvement if the architecture is well-matched to the target applications. In essence, our architecture stands between ASIC and FPGA implementation. The authors in [4] suggest that the ratio of silicon area and delay required to implement circuits in FPGAs and ASICs is on average 35. Our proposed architecture can reduce the gap between FPGA and ASIC from 35 times to 2 times when floating point applications are implemented on such FPGAs.

The delay reduction is also significant. In our benchmark circuits, delay is reduced by 3.6 times on average for single precision applications and 4.3 times on average for double precision applications. We believe that double precision floating point implementation on commercial FPGA platform is not as effective as the single precision one. Therefore, the double precision FPFPGA offers better delay reduction than the single precision one. In our circuits, the critical path is always within the embedded floating point units, thus we would expect a ratio similar to that between normal FPGA and ASIC circuitry. Our results are consistent with [4] which suggests the ratio is between 3 to 4. As the critical paths are in the FPU, improving the timing of the FPU through full-custom design would further increase the overall performance.

We conclude that the proposed architecture meets the basic requirements for FPFPGA in Section 3.1.1 by: (1) there are FAs and FMs in the FPU for primitive FP operations (matches requirement 1), (2) the FAs, FMs and WBs inside the FPU are connected by bus-based routing (matches requirement 2), (3) feedback and output registers are included to support pipelining (matches requirement 3), (4) the fine-grained CLBs in FPFPGA support implementation of state machines and other bit-oriented operations (matches requirement 4).

3.2 Baseline Hybrid FPGA Architecture for Pre-fabrication Exploration

In the last section, we have proposed a novel FPFPGA architecture and compared it with an existing device. However, it is not flexible to use post-fabrication exploration to investigate different architecture parameters such as the size of the device, interface between fine-grained and coarse-grained logics. In order to solve this limitation, we first introduce the assumption of the fine-grained elements, coarse-grained elements and routing architecture of our hybrid FPGA model in this section. The assumed base-line architecture can then be evaluated by a flexible pre-fabrication evaluation tool called VPH which will be introduced in the next section. The assumptions are also applied to Chapter 4, 5 and 6 in this thesis.

3.2.1 Fine-grained and Coarse-grained Assumption

The physical die area and photomicrograph of a Virtex-II device have been reported [134]. We employ this area result to estimate the area of the coarse-grained floating point unit in the existing device which is shown in Section 3.1.4. Therefore, we choose Virtex-II architecture as our baseline architecture in this thesis.

In our model, we assume that the fine-grained resources in the FPGA consist of a grid of identical configurable logic blocks (CLBs) connected by horizontal and vertical wire channels which are similar to Virtex-II slices. A CLB is a cluster of two basic logic elements (BLEs) containing *4*-LUT, flip flop (FF), fast carry chains, internal multiplexers and XOR gates. Figure 3.5 shows the BLE in a CLB. The aspect ratio of the CLB is equal to *1* and the pin architecture is full-perimeter, which leads to the best area-efficiecy [21, 24]. The aspect ratio of the FPGA is equal to *1* which is the most area-efficient [21]. We assume the delay of the CLB is the same as the timing specification of a Virtex-II device, speed grade -6 [135] in the baseline architecture.

We adopt the double precision FPUs proposed in Section 3.1 as our coarse-grained elements. There are two floating point adders/subtractors, two floating point multipliers and five wordblocks in each

FPU, which are connected by bus based wires. In the hybrid FPGA, coarse-grained EBs are surrounded by fine-grained CLBs and they are connected by horizontal and vertical wire channels [23]. The placement of EBs can be anywhere in the FPGA, which is more flexible than the column based FPGA in Section 3.1, Figure 3.1. Figure 3.6 shows an example of the hybrid FPGA used in this thesis. Four EBs are positioned in the center and separated by 1 CLB distance. Each EB is 3x3 tiles and surrounded by CLBs, and the EBs are separated by distance D_{eb}. They are connected by uniform width vertical and horizontal wires. According to the usage of FPUs and CLBs in Table 3.7, we summarise the information of the components used in the baseline hybrid FPGA as shown in Table 3.4.

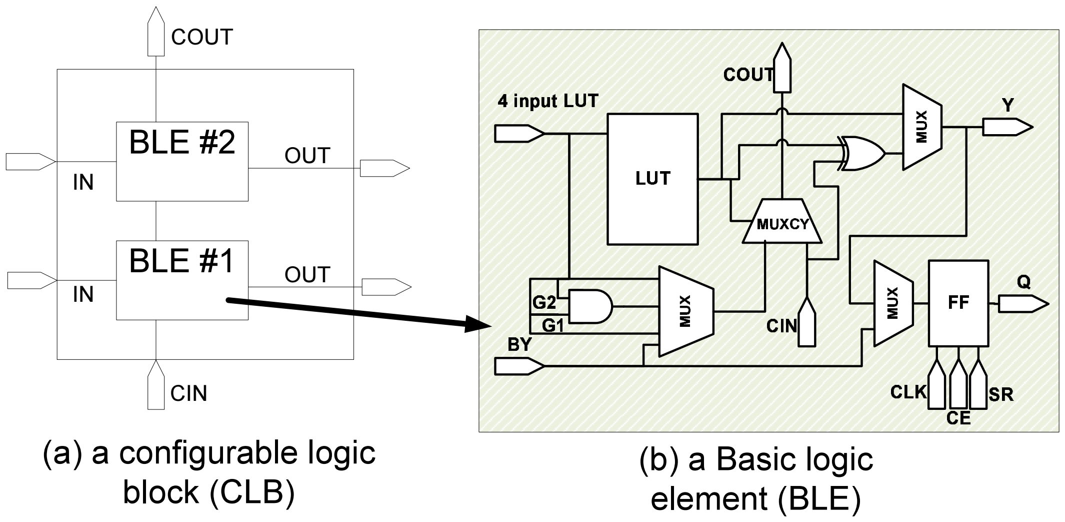


Figure 3.5: (a) A configurable logic block (CLB) in hybrid FPGA, (b) A basic logic element (BLE) in a CLB

3.2.2 Routing Architecture Assumption

Figure 3.7 shows our assumed routing architecture. CLBs and EBs are connected to W parallel routing tracks of segment length L using connection boxes. All CLBs, EBs and I/O pads are fully connected to connection boxes ($Fc_{output} = 1$, $Fc_{input} = 1$ and $Fc_{pad} = 1$, in [23]). W is constant for this baseline architecture, however we will introduce a heterogeneous channel width near EBs to meet the routing demand found in Chapter 5, Section 5.3.3. Segment channels are intersected by a switch box. The switch box offers each incoming wire the ability to connect to three other wire segments ($Fs = 3$). We use the subset switch box (also known as disjoint), in which the wire segment in track i can only

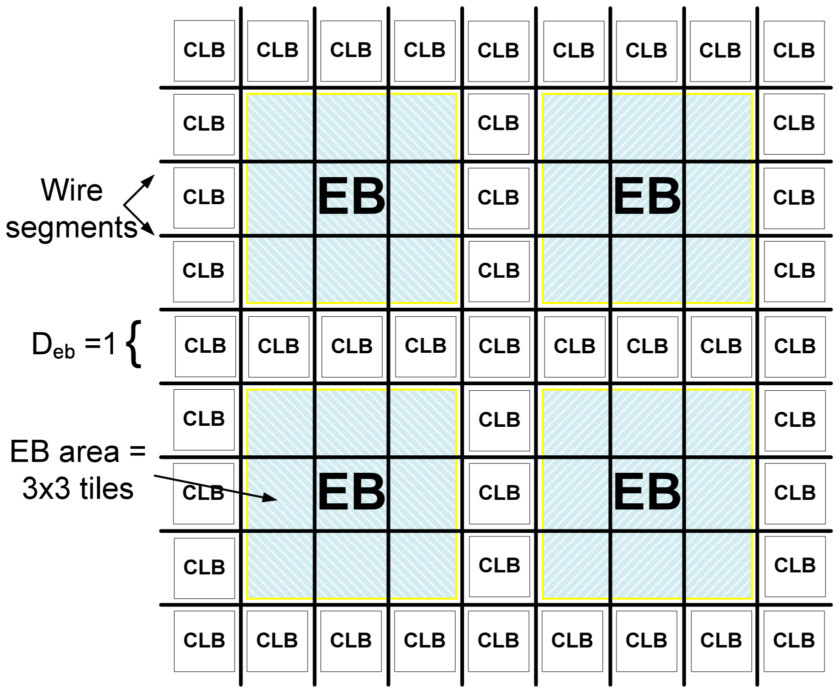


Figure 3.6: An example of the baseline hybrid FPGA: Embedded blocks (EBs) are surrounded by grid based CLBs

Table 3.4: Information of the components in the hybrid FPGA

Components	Description
Hybrid FPGA	Area = 100 x 100 CLB tiles
	Number of FPUs = 16, arranged as 4x4 grids in the middle of FPGA
CLB	Area = 1 x 1 tile
	Cluster size = 2
	Number of pins = 13 inputs + 5 outputs
	I/O density = 4.5 pins per unit CLB tile length
	Maximum delay = 0.45ns
FPU	Area = 14 x 15 CLB tiles (Total 210 CLBs, best aspect ratio ≈ 1 and area ≈ 214 CLBs)
	Number of pins = 286 inputs + 258 outputs
	I/O density = 9.7 pins per unit CLB tile length
	Maximum delay = 4.25ns
BRAM	Memory used in Chapter 4
	Area = 2 x 4 CLB tiles
	Number of pins = 90 inputs + 64 outputs
	I/O density = 12.83 pins per unit CLB tile length
	Delay = 2.1ns

connect to other wire segments in track *i*. There are no switch boxes inside EBs, so changes in wire direction are not allowed, this is common in commercial devices [3]. We will discuss the effect of adding switch boxes on top of EBs in Chapter 5, Section 5.3.2. The routing switch is simply using a tri-state buffer as driver, although the proposed routing optimization methods can be applied to any buffer such as pass-transistor.

The area and delay model of the wire is based on the PTM 0.13μm, 1.3V CMOS process [136]. We obtain the equivalent resistances, capacitances and intrinsic delay of the tri-state buffer from SPICE simulations with the PTM 0.13μm process. Table 3.5 shows the information of the routing resources used in the baseline architecture.

We estimate the routing area in our architecture by using the model stated in [22]. We count the area of connection multiplexers from logic blocks to tracks and tri-state buffers in routing switches in terms of minimum-width transistors per CLB tile.

Table 3.5: Information of the routing resources in the baseline FPGA

Routing resources	Description
Tri-state buffer	34.5x minimum width transistor area
	5x minimum driving strength
Wire	width = 1.5x width of min. width transistor
	space = 1.5x width of min. width transistor
Segment length	L = 4, the best area-delay product [22]

3.3 Pre-fabrication modelling tool: Versatile Place and Route for Hybrid FPGA - VPH

In Section 3.1, we have introduced a novel hybrid FPGA architecture for floating point application. We have compared the performance of the virtual architecture with existing devices using a commercial place and route tool. We are interested in studying what are the trade-offs when adopting different hybrid FPGA architecture parameters such as I/O pin position, aspect ratio of FPU and routing resources. The commercial tools we used in the last section which based on existing devices

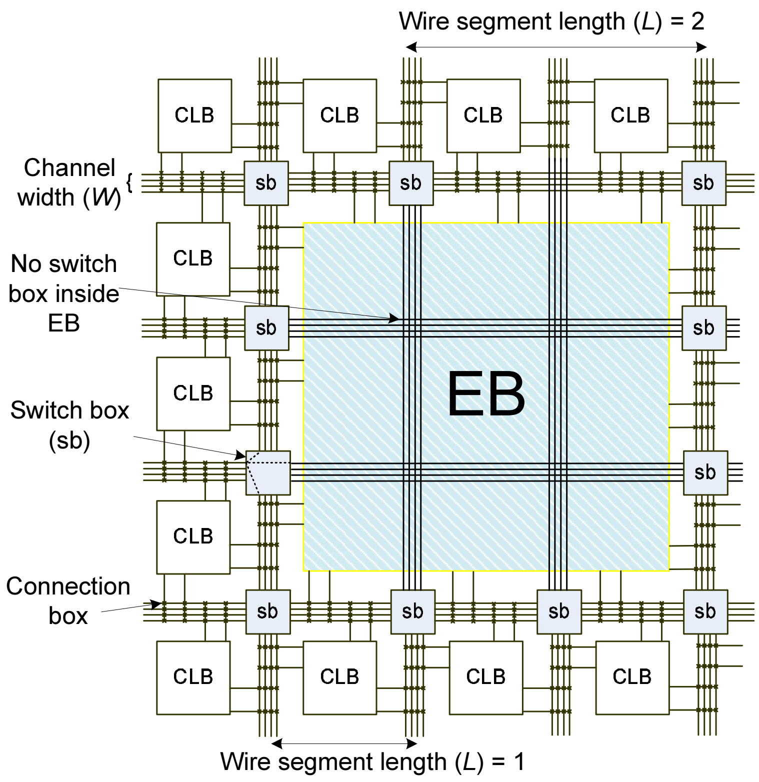


Figure 3.7: Detailed routing architecture in the baseline FPGA

are not feasible for those architectural parameters. Also the commercial device has limited resources to implement a circuit. For example, double precision *syn7* cannot be implemented in the Virtex XC2V6000 device. Therefore, a more feasible tool is needed to explore different architectures. In this section, we define the research problems, and propose our tool called Versatile Place and Route for Hybrid FPGA (VPH) to evaluate the hybrid FPGA. We also provide an initial result of packing, placing and routing of the benchmarks using VPH at the end of this section.

3.3.1 Requirements

In order to evaluate architectural trade-offs, a modelling tool must capture the salient parameters. An optimised setting of parameters should be found to minimise a cost function that normally involves delay, area and power consumption.

In a domain-specific hybrid FPGA, some important questions that must be answered include:

1. What is the best interface for the elements in hybrid FPGA? (will be covered in Chapter 4)

2. What is the optimised interconnection method between fine-grained and coarse-grained blocks? (will be covered in Chapter 5)
3. Which coarse-grained elements should be used to give the highest benefit for a specific class of applications? (will be covered in Chapter 6)

VPH is based on VPR and was designed with the following requirements.

1. Modelling modern architectures such as Xilinx and Altera devices. The work presented is based on the Xilinx Virtex-II but can be generalised to other architectures. (has been capture in Section 3.2 and will be achieved in Section 3.3.4)
2. Allowing exploration of different hybrid FPGA architectures. Hybrid FPGAs contain both fine-grained and coarse-grained elements, the architectural parameters of them should be flexible. (will be covered in Section 3.3.3)
3. Supporting optimization of the hybrid FPGA architecture such as a novel routing architecture and internal optimization of coarse-grained elements. (will be covered in Chapter 5)

3.3.2 Design flow

In the VPH design flow (Figure 3.8), applications are written in a high level hardware description language and synthesised to a mapped library netlist in VHDL format using commercial tools such as Synplicity Synplify Premier 9. The library netlist contains the usage and connection of simple units such as registers, LUTs, internal muxes and internal NOT gates. The basic logic block packing tool, VPHpack, packs these units into basic logic elements (BLEs). VPHpack clusters BLEs into CLBs. VPHpack outputs the final packed basic logic elements (BLEs) as a netlist for VPH.

A user constraint file (.ucf) is used to specify the FPGA area and the absolute position of each embedded block. A separate constraint file for each embedded block is used to specify the area, the pin position and the timing information for the EB; the area and delay information for each block is obtained using the Synopsys Design Compiler V-2008.09. A link file is used to connect the ucf file to the associated module netlist. As in VPR, an architecture file specifies the fine-grained FPGA's architectural parameters, such as timing delay of the LUT. Using these files, the VPH tool performs placement, routing, and timing analysis to produce area and delay estimates for each benchmark circuit.

Modern applications are usually written in a high level hardware description language, e.g. VHDL or Verilog. In the traditional VPR design flow, SIS [137] is used to synthesise circuits which are described in Berkeley Logic Interchange Format (BLIF). Conversion of the file format is required. The VPH design flow is simpler, since available VHDL application benchmarks for different hybrid FPGA designs can be directly processed by VPH.

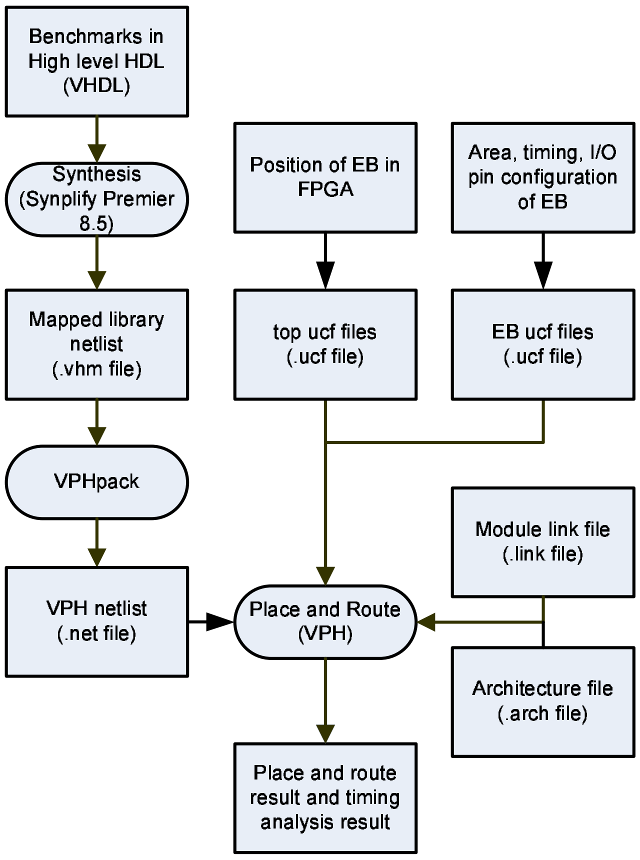

Benchmarks in High level HDL (VHDL)
Synthesis (Synplify Premier 8.5)
Mapped library netlist (.vhm file)
VPHpack
VPH netlist (.net file)
Position of EB in FPGA
Area, timing, I/O pin configuration of EB
top ucf files (.ucf file)
EB ucf files (.ucf file)
Module link file (.link file)
Place and Route (VPH)
Architecture file (.arch file)
Place and route result and timing analysis result

Figure 3.8: Design flow of VPH

3.3.3 User Constraints in VPH

The user constraint file (.ucf file) in VPH allows flexibility to explore interface and different parameters of hybrid FPGA. In summary, the user constraint file describes:

1. The width and height of the FPGA device model (for Chapter 4 to 6).
2. The position of the lower left corner of the EB(for Chapter 4 to 6).
3. Area of the EB by specifying its width and height(for Chapter 4 to 6).
4. The I/O pin configuration of the EB(for Chapter 4 to 6).
5. The delay of the EB(for Chapter 4 to 6).
6. Enable or disable switches inside EB(for Chapter 5).
7. Enable or disable extra routing wires near EB (for Chapter 5).
8. The position and distance of the extra routing wires(for Chapter 5).
9. The wire ratio of extra routing to normal routing(for Chapter 5).

The above settings allow a wide range of architectural parameters to be examined without the limitation of the physical constraint of existing devices.

3.3.4 VPHpack

We use Synplicity's Synplify Premier 9 to synthesise the high level hardware description language to a mapped library netlist (.vhm). The mapped library netlist describes the basic units occupied in a particular FPGA. We developed VPHpack which packs and clusters the units in the mapped library netlist. Table 3.6 shows the units handled by VPHpack.

The packing tool packs the units into Virtex-II BLEs and then clusters the BLEs into CLBs as shown in Figure 3.9 There are two BLEs in a CLB, fast carry chains propagate upwards , shift registers propagate downwards and FF inputs are (CLK, CE and SR) shared among two BLEs.

Table 3.6: Units in mapped library netlist handled by VPHpack

Unit name	Description	Unit name	Description
IBUF	Input buffer	XORCY	Fast carry chain XOR
OBUF	Output buffer	MUXCY	Fast carry chain mux
VCC	VCC input	MULT_AND	Internal AND
GND	GND input	FD	D-type FF
BUFGP	Clock buffer	FD{P, C, E, R, S}	FF with different combination of P: preset, C: clear, E: enable R: Reset, S: synchronous set
LUT{1,2,3,4}	Input size {1,2,3,4} LUT		
INV	Internal inverter		

To be packed into a single logic cluster, a group of BLEs must satisfy two conditions:

1. The BLEs must use the same shared inputs of FF.
2. The BLEs in different carry chain must not pack into a single cluster.

VPHpack is currently available for the Virtex-II architecture, it is easily extended to other devices by adding the mapped library and slightly modifying the packing algorithm for a specific device.

3.3.5 Elmore Delay Model and Critical Path

For the timing analysis in VPH, we calculate the Elmore delay [138] from source node to sink node to find the delay of a path. The interconnects are modelled as an RC tree as shown in Figure 3.10. Formula 3.1 is used to calculate the Elmore delay of a source-sink path. The source-sink path is also called the combinational path, which is the path from flip-flop output to flip-flop input. R_m and $C(subtree_m)$ are the resistance and total downstream capacitance of the element in the path at node m. $T_{d,m}$ is the intrinsic delay of either switch box buffer or connection box buffer in the path. The critical path is the one with highest timing delay among all source-sink paths in a circuit.

$$Delay = \sum_{m \in Source-sink\ path} R_m C(subtree_m) + T_{d,m} \tag{3.1}$$

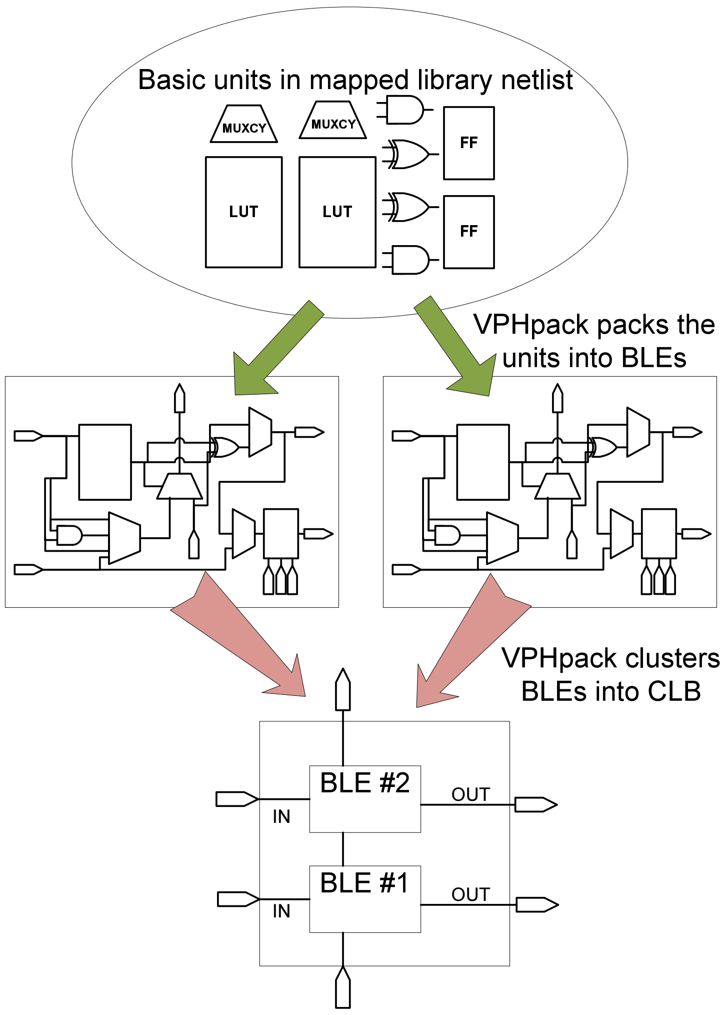

Basic units in mapped library netlist
MUXCY
MUXCY
FF
LUT
LUT
FF
VPHpack packs the units into BLEs
VPHpack clusters BLEs into CLB
BLE #2
IN
OUT
BLE #1
IN
OUT

Figure 3.9: VPHpack packs and clusters basic units into CLB

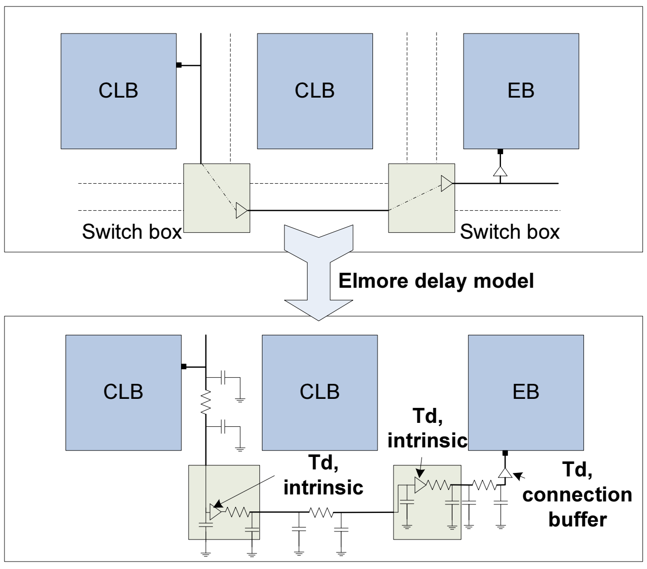


Figure 3.10: Elmore delay model for the delay path in VPH

3.3.6 Results

VPH provides feasible exploration for a hybrid FPGA. We first use VPHpack to pack and cluster the eight benchmarks described in Chapter 2, Section 2.5.3. By using the resulting netlist, we can explore different architecture parameters and optimisation for floating point hybrid FPGA by place and route the circuits from Chapter 4 to Chapter 6.

The benchmark *syn7* is not able to be implemented in the Virtex-II devices in Section 3.1 because of the limited physical logic resources. VPH can implement large circuits in a virtual FPGA as long as there in enough memory in the computer to run the program. Implementation of the double precision floating point circuit *syn7* is possible with VPH. From Table 3.3(b), the usage of fine-grained resources reduced significantly after embedding FPUs. This is not complex enough to do architectural exploration. For example, if the design is too small, the number of nets may not be enough to simulate the congestion and routability inside FPGA. To remain complexity of the circuits for architectural exploration after embedding FPUs, the core of each simple DSP kernels: *dscg, bfly, ode, mm3 and fir* is replicated four times and are connected together in a single circuit. For example, a *dscg* benchmark contains four *dscg* cores connected together. All circuits use a single global clock.

The number of CLB, FPU and computational cores of each benchmark is reported in Table 3.7. Note that there is no register in input and output pad (I/O registers) and shift register LUT (SRL) in VPH, where Virtex-II has I/O registers and SRL can be implemented as memory to save the usage of FFs. Therefore the number of CLB is different from Table 3.3(b). If we switch off the options of using I/O registers and SRL in Virtex-II device, the number of CLB used in Virtex-II is only 5.5% less than VPHpack. This deviation may be caused by different packing algorithm used in the tools.

Finally, the netlist generated by VPHpack is placed and routed by VPH. Figure 3.11 gives an example of placing and routing the benchmark *bgm* in the hybrid FPGA architecture with routing switches inside the FPU. There are sixteen FPUs in this FPGA, the purple blocks are used by *bgm* and the green blocks are the unused FPUs. Figure 3.12 shows that the direction of the nets inside the FPU can be changed. The detail of the change of direction will be discussed in Chapter 4 to optimise the routing architecture.

Table 3.7: Number of FPUs and CLBs used for each benchmark packed and clustered by VPHpack

Benchmarks	No. of CLB	No. of FPU	No. of core
dscg	649	8	4
bfly	777	8	4
ode	259	8	4
mm3	713	8	4
fir	169	8	4
bgm	6,433	7	1
syn2	288	3	1
syn7	288	16	1

VPH fulfills the research requirements of a hybrid FPGA architecture stated in Section 3.3.1 by: (1) Modelling Virtex-II likes architecture with carry chain (matches requirement 1), (2) User constraint allows flexible architecture of the device (matches requirement 2), (3) Supporting optimisation of routing architecture such as allowing routing switches in FPU (matches requirement 3),

3.4 Summary

This chapter first proposes an FPFPGA architecture which involves a combination of reconfigurable fine-grained and reconfigurable coarse-grained units for floating point computations. A parameterisable description is presented which allows us to explore different configurations of this architecture. To provide a more accurate evaluation, we adopt a methodology for estimating the effects of introducing embedded blocks to commercial FPGA devices. The approach is vendor independent and offers a rapid evaluation of arbitrary embedded blocks in existing FPGA devices. The FPFPGA reduces the area by 17 times and increases the clock frequency by 4 times on average when compared to an existing commercial FPGA device.

After that we define a baseline hybrid FPGA architecture and propose a place and route tool called VPH to explore domain-specific hybrid FPGA. The features and work flow of VPH are described. VPH supports most modern FPGA architecture such as carry chains, embedded memories and multipliers. It is based on VPR which has facilitated research on FPGA architectures. Based on the VPH and the FPFPGA architecture, optimisation of the FPFPGA will be considered in the next few chapters.

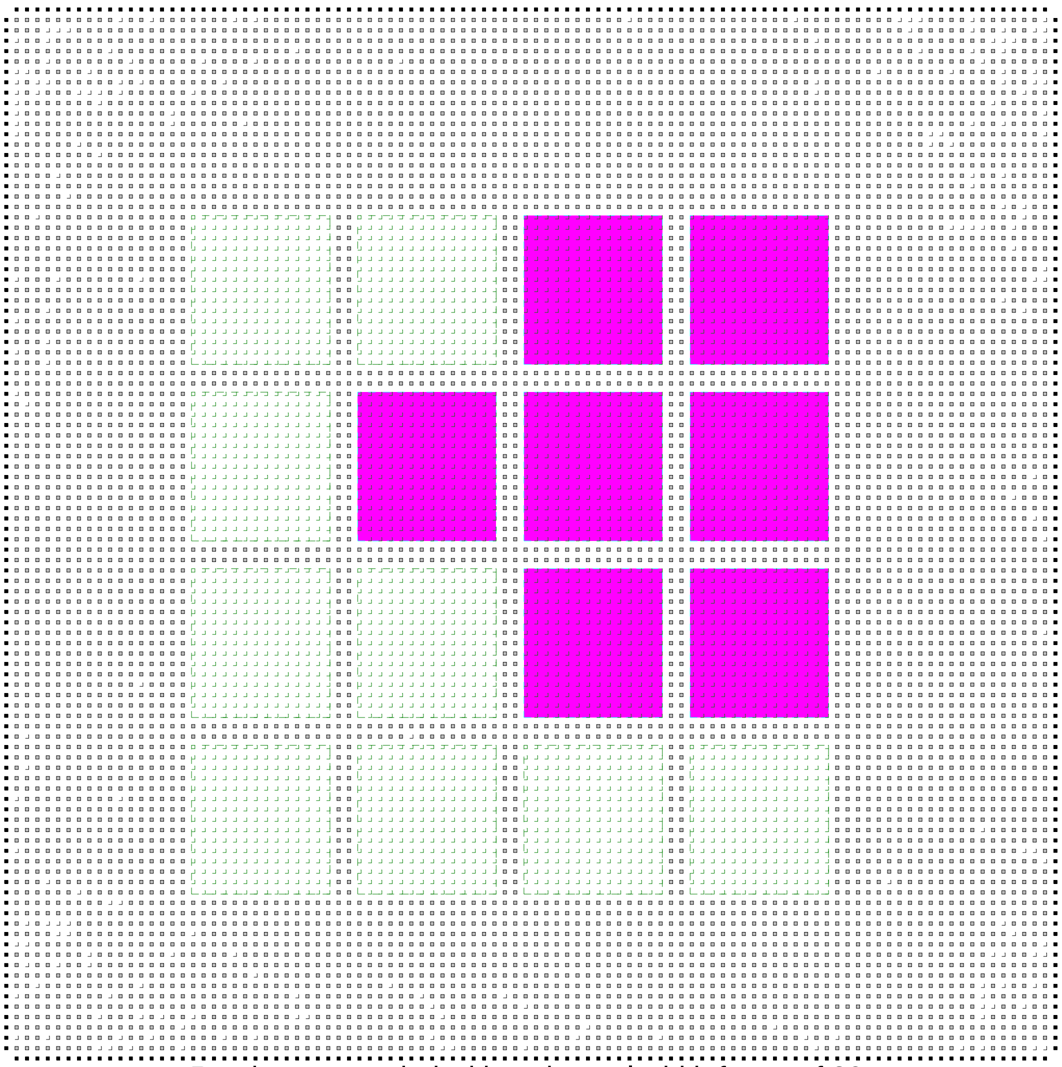


Figure 3.11: The circuit *bgm*, which has used 7 FPUs out of 16, is placed and routed using VPH (Purple blocks are the used FPUs and green blocks are unused FPUs.)

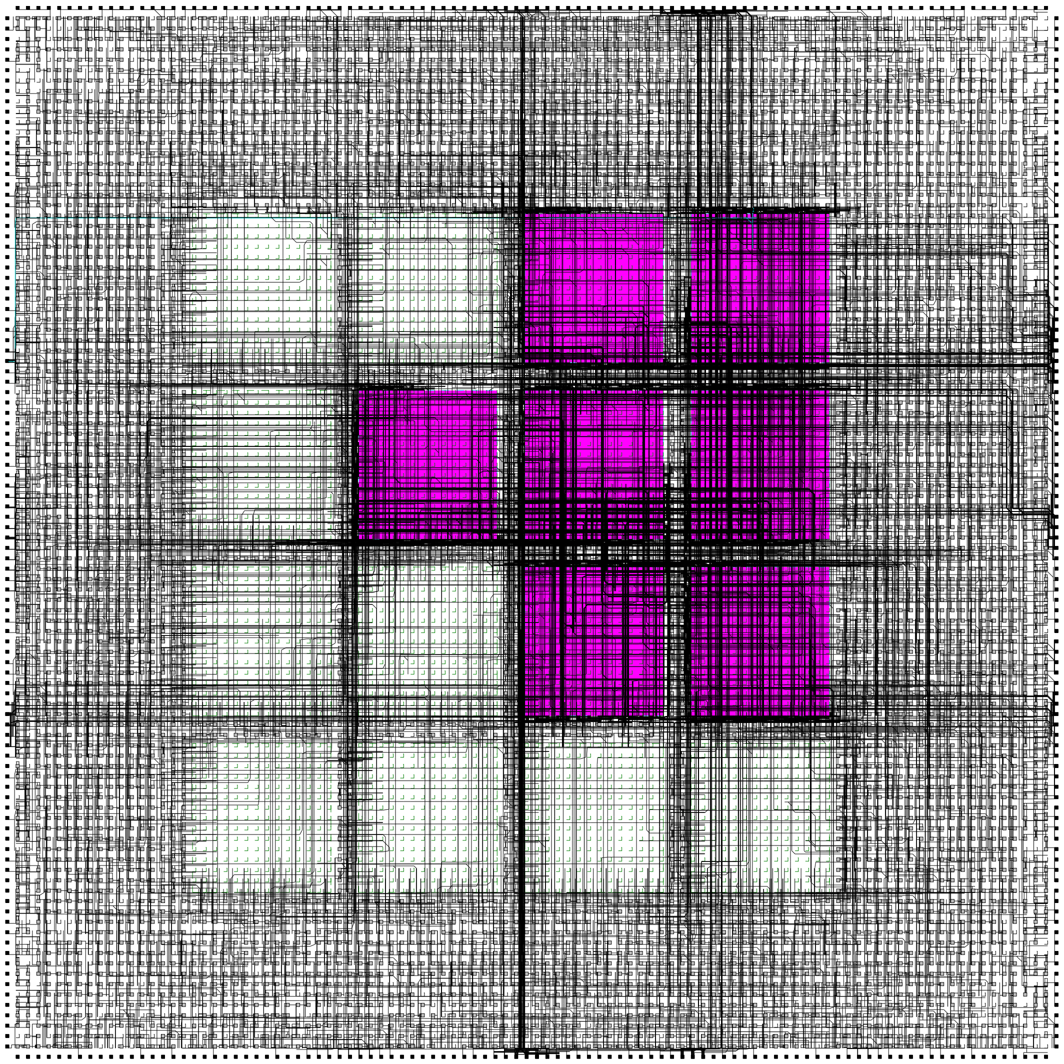


Figure 3.12: Routing switches are allowed in FPUs. The nets in FPUs are able to route and change of direction

Chapter 4

Interface between Coarse/Fine-grained Logic

We have proposed a novel hybrid FPGA architecture for floating point application domain in Chapter 3. This chapter examines the interface between fine-grained and coarse-grained programmable logic in FPGAs. Specifically, it presents an empirical study that covers the location, pin arrangement, and routing resources between embedded blocks and the fine-grained logic fabric in FPGAs. It also studies this interface in FPGAs which contain both FPUs and embedded memories.

4.1 Motivation

An important consideration when adding coarse-grained embedded elements to an FPGA is the interface between the coarse-grained and fine-grained resources. If this interface is not flexible enough, the usefulness of the embedded block will be reduced, since connections to and from the block will be expensive. On the other hand, if the interface is too flexible, it will require too much area and delay, possibly negating the density and performance advantages of including the embedded block, and resulting in unnecessary overhead for applications that do not use the embedded component.

In this chapter, we focus on architectural interface issues, such as the location of the embedded elements, and the I/O interface between the embedded elements and the fine-grained fabric. Our approach is presented in the context of the embedded FPUs described in Chapter 2, Section 2.5.3 and the baseline architecture presented in Chapter 3, Section 3.2. Chapter 5, Section 5.4 will update the interface according to the optimised routing architecture.

4.2 Interface Parameters: Single EB Type

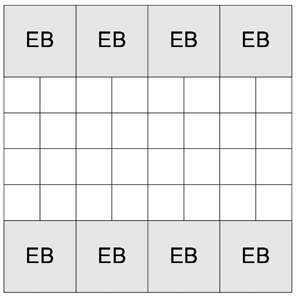
In this section, we consider a range of interface architectures. First, we explore the single EB type hybrid FPGA. To describe the space of single EB type architectures that we consider, we define the parameters: (1) EB position, (2) pin location, (3) interconnect flexibility and (4) EB aspect ratio.

4.2.1 EB Position

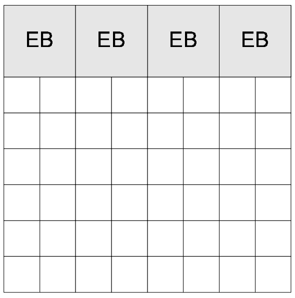
The embedded blocks can be placed in various places within the FPGA. We consider the positions as shown in Figure 4.1. There are advantage and disadvantage in each type of position. EBs are on the top and bottom of FPGA in Type 1 position, they are close to I/O of FPGA, but far away from the EB column in the opposite side. For type 2, EBs are close to I/O of FPGA too, but only one of the side can connect to CLBs directly, the routing density in EB may be very high. The EBs in type 3 are packed closely together, it may help to reduce routing delay compared with type 4. Therefore we examine the trade-offs of these configurations.

4.2.2 Pin Location

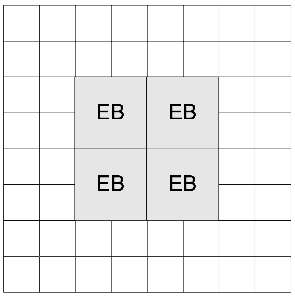
Figure 4.2 shows several strategies for positioning the pins of each EB. Strategy (a) has the highest I/O density, but may be suitable if signals from the I/O block are to be combined using a small set of CLBs. Strategies (b), (c), (d) have lower I/O density, but may result in longer connections if signals from more than one side of the EB are to be connected to the same CLB(s).



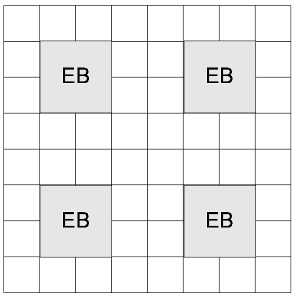
(a) Type 1: EBs are on the top and bottom of CLBs



(b) Type 2: All EBs are on the top of CLBs

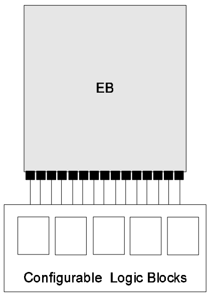


(c) Type 3: EBs are in the middle of CLBs

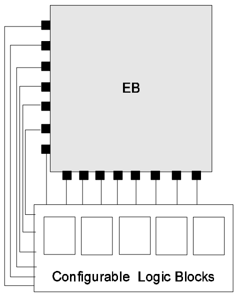


(d) Type 4: EBs are surrounded by sea of CLBs

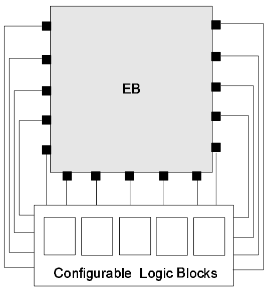
Figure 4.1: Various positions of the EBs relative to the fine-grained CLBs



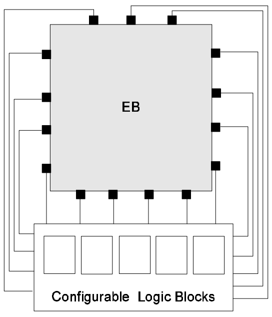
(a) CLBs connect to one side of EB with highest I/O density



(b) CLBs connect to two sides of EB



(c) CLBs connect to three sides of EB



(d) CLBs connect to four sides of EB with lowest I/O density

Figure 4.2: Different pin positions in EB

4.2.3 Interconnect Flexibility

The width of the channels surrounding the EB has a significant impact on the routability of the device. Since our EB has a large number of pins, congestion around the EB may happen so it is desirable to relieve this congestion by using wider channels.

4.2.4 EB Aspect Ratio

Several layouts of each embedded block are possible. We consider various aspect ratios as shown in Figure 4.3. The I/O density would be different as they have different perimeters. This may affect the performance of the system.

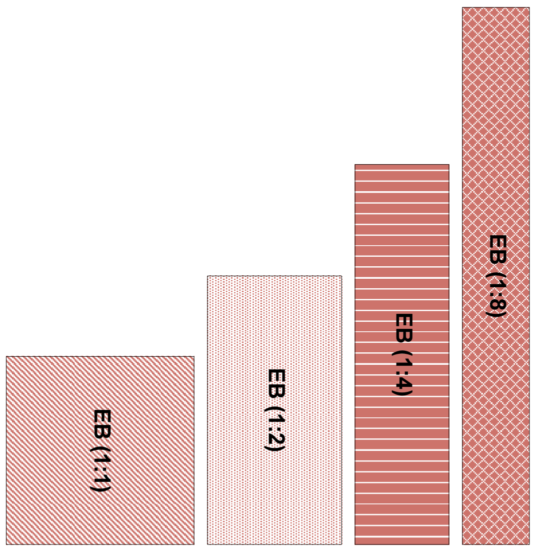


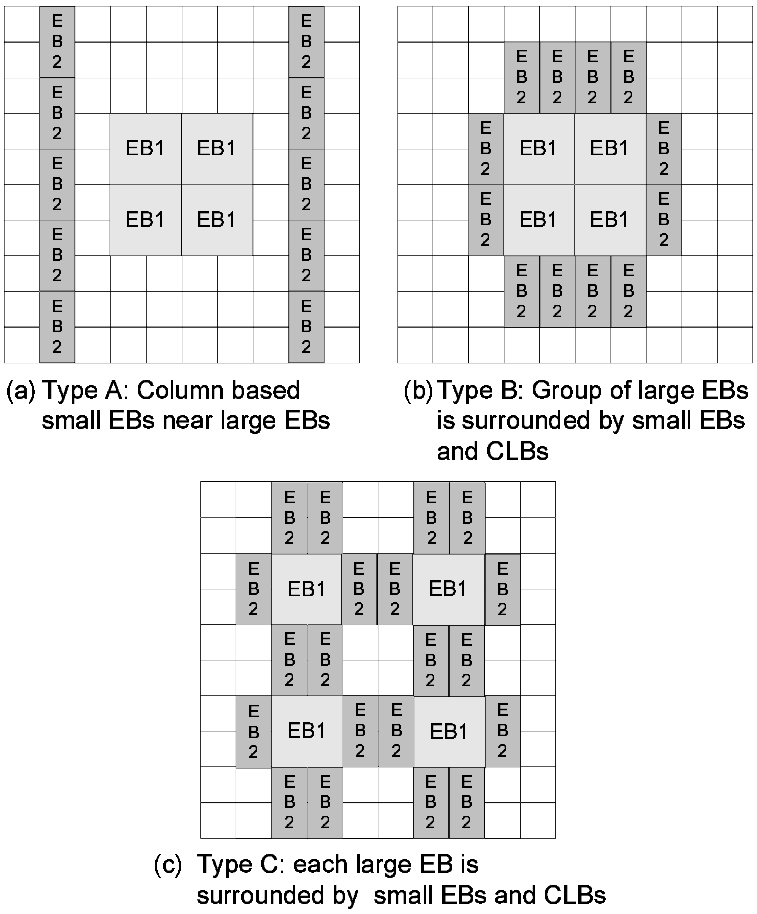
Figure 4.3: Different aspect ratio of EB

4.3 Interface Parameters: Multiple EB Type

We also study the interface between multiple EB types and the fine-grained fabric. In this case, connections exist between the two types of EBs and also between EBs and CLBs. The best interface interface architecture may be different to the single type EB FPGA. Therefore, we investigate the following parameters for the hybrid FPGAs with two types of embedded blocks.

4.3.1 EB Position

We arrange the different EB types in various ways as shown in Figure 4.4. We consider three different arrangements. The first type places the smaller EBs in columns next to the larger EBs. The second type places the smaller EBs around a group of larger EBs. The third type places the smaller EBs around individual larger EBs.



(a) Type A: Column based small EBs near large EBs

(b) Type B: Group of large EBs is surrounded by small EBs and CLBs

(c) Type C: each large EB is surrounded by small EBs and CLBs

Figure 4.4: Various positions of the multiple EBs

4.3.2 Interconnect Flexibility

Embedding additional EBs may change the amount of routing resources that are needed. The connections between EBs are usually bus based which require more routing resources. It is because if the I/O density of the additional EB is larger (the I/O density of BRAM is higher than our proposed FPU as shown in Chapter 3, Table 3.4), more wires may be needed to connect to another EB within a certain area. And the congestion in this area increases and may reduce the performance of the FPGA.

4.4 Methodology

We employ an empirical methodology to examine the impact of the interface parameters described in the previous section. This section employ the benchmark circuits in Chapter 2, Section 2.5.3, the CAD tool VPH in Chapter 3, Section 3.3, and the baseline model described in Chapter 3, Section 3.2.

For the experiment involving embedded memories, we add BRAMs to the benchmark circuits. It is more realistic to store the input and output data of the applications in internal BRAMs rather than store the data in off chip memories. The BRAM data lines are connected to primary input/output of the benchmarks. And the BRAM address lines are connected to counters which are also added to the benchmarks. The adders do not affect the performance because they are implemented by using fast carry chain, the delay is small compare to the BRAM and the FPU. The benchmark circuits now contain two different types of EB: (1) FPUs and (2) BRAMs. The number of BRAMs used in each benchmark circuit is shown in Table 4.1.

Table 4.1: Number of BRAM used in each benchmark

Benchmarks	dscg	bfly	fir	ode	mm3	bgm	syn2	syn7
No. of BRAM	22	40	12	25	12	18	18	18

4.5 Result: Single EB Type

In this section, the impact of the interface parameters in Section 4.2 on hybrid FPGAs is studied. In the experiments conducted, the channel width used is 20% more than minimum channel width to prevent the result being affected by congestion.

4.5.1 EB Position

We first examine how the position of the EBs affects the overall performance of the device. As shown in Figure 4.1, we consider positioning the EBs both around the periphery of the device, as well as in the centre. Intuitively, positioning the EBs in the centre will lead to shorter wirelengths for wires that connect multiple EBs. However, positioning the EBs around the periphery may cause less congestion since the EBs will be more spread out. In type 1 and 2 position $D_{eb} = 0$ for the EBs on the same row. Type 3 has $D_{eb} = 0$ for all EBs, while type 4 has $D_{eb} = 9$.

Delay Impact

Figure 4.5 shows the delay results for each of the positioning strategies described in Figure 4.1. The best strategy is type 3, in which the EBs are in the centre of the device, surrounded by a sea of CLBs. It achieves at least 16% in speed improvement compared to the other positioning strategies. The critical paths of our circuits tend to include nets that connect multiple EBs; thus placing the EBs close to each other is beneficial.

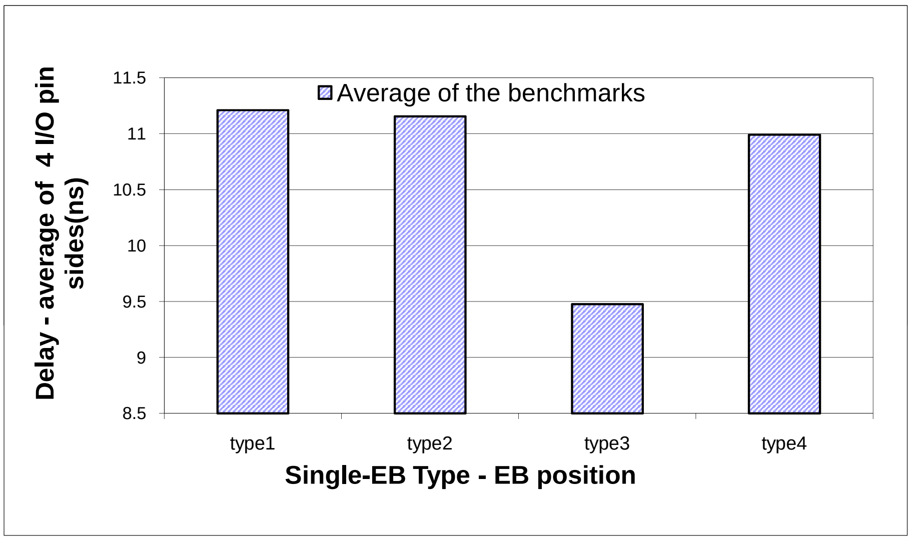


Figure 4.5: Delay against various EBs positions for the single EB type FPGA (in Figure 4.1)

Routing Area Impact

We investigate the routing area used in the different types of positioning. Routing area is more accurate than minimum channel width for evaluating routing resources as it accounts for the area of the connection multiplexers and tri-state buffers in routing switches in term of minimum-width transistors (details are in Chapter 3, Section 3.2.2). Figure 4.6 shows that type 4 configuration consumes less routing resources, which is only 62.5% of that of type 3 and about 50% of type 1 and 2. There are a lot of connection from the I/O pins of FPGA to the EBs and between EBs. In type 1 and type 2, the EBs are positioned very close to the I/O pins, which makes those regions very congested. Similarly, the EBs are close to each others in type 3 configuration. This leads to high routing demand near the edge of EBs for the connection between them. Type 4 can evenly distribute the routing nets all over the FPGA. This configuration requires less routing channels and leads to smaller routing area.

Although type 3 requires more routing resources in the baseline architecture, we will show in Chapter 5 that routing optimisation can improve the routing area with similar speed in type 3 position.

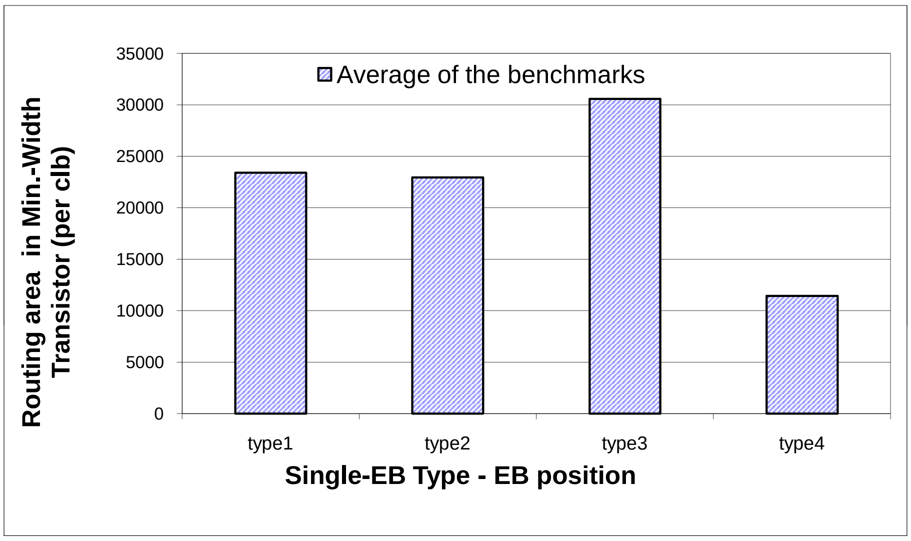


Figure 4.6: Routing area against various EBs positions for the single EB type FPGA (in Figure 4.1)

4.5.2 Pin Location

We next consider the effect of I/O pin position on the periphery of each EB. As shown earlier, pins can be distributed evenly around the EB, or can be concentrated on one or more sides of the block.

Intuitively, distributing the pins evenly will lead to a lower I/O density, possibly reducing congestion but may lead to longer wirelengths if pins from more than one side of the EB are connected.

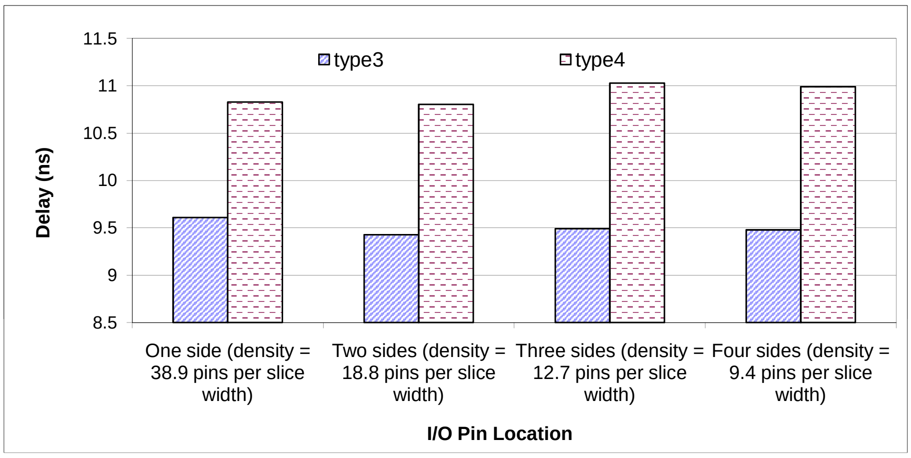


Figure 4.7: Delay against various I/O configurations for the single EB FPGA (in Figure 4.2) with type 3 and type 4 EB position

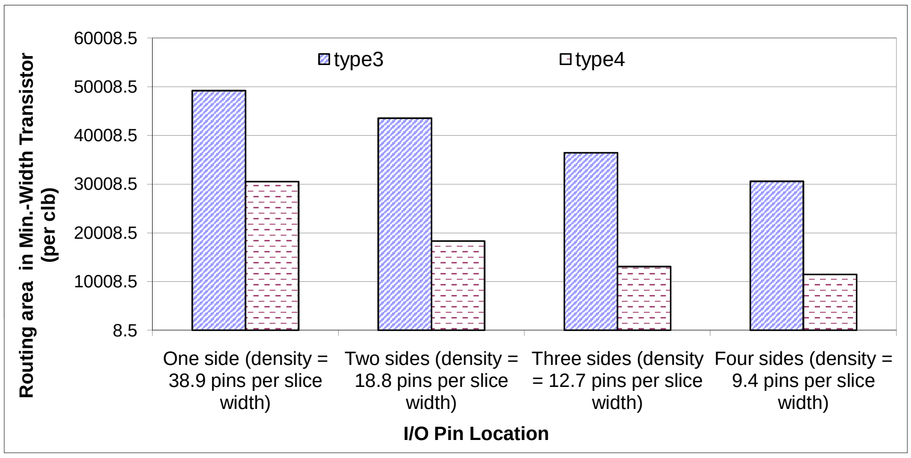


Figure 4.8: Routing area against various I/O configurations for the single EB FPGA (in Figure 4.2) with type 3 and type 4 EB position

We selected the fastest EB position FPGA (type 3) and smallest EB position FPGA (type 4) to examine the pin location of EBs. The results are shown in Figure 4.7 for delay and Figure 4.8 for routing area. The average critical path delay of the circuit is similar for all I/O pin configurations in both type 3 and type 4 EB position. The maximum variation is less than 1.9%. The I/O pins configuration does not have significant impact on delay.

On the other hand, Figure 4.8 shows that the routing demand in each channel can be reduced by distributing the pins evenly around each EB. By comparing to the configuration in which all pins are on one side of the block, evenly distributing the pins reduces the routing resources by 37.9% in type 3 EB position and 62.4% in type 4 EB position. We conclude that this is the best choice.

4.5.3 Interconnect Flexibility

We next consider the width of the channels surrounding the EBs. Intuitively, there is a high pin density on each side of each EB; this may place additional demands on the routing fabric near the EBs. If the fabric cannot provide the required flexibility, circuitous routes may be required, leading to increased delay.

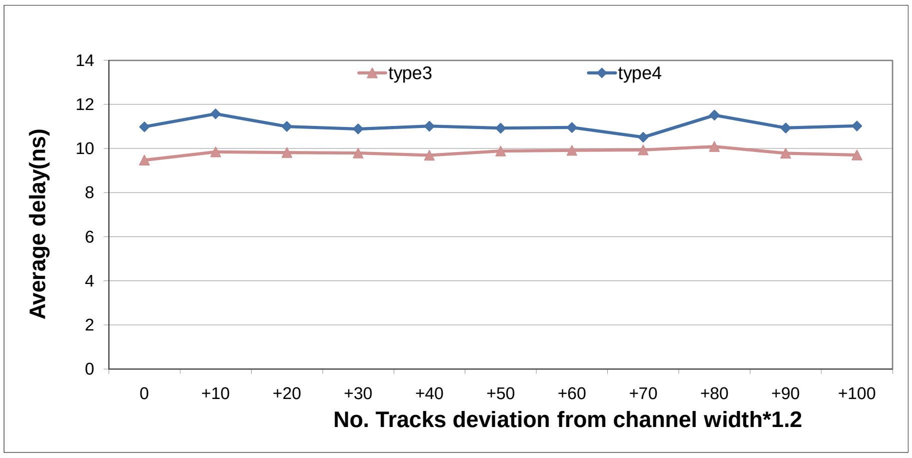


Figure 4.9: Average delay against channel width for the single EB type FPGA

The results in Figure 4.9 show the effect of routing channel width on delay. For routable circuits, rather surprisingly, the average variation is less than 1.7%. We believe this is due to critical paths being routed efficiently so once the circuit is routable, channel width does not affect delay.

4.5.4 EB Aspect Ratio

Finally, we consider how the aspect ratio of each EB affects the overall performance of the FPGA. In this experiment, the area of EB is fixed, but the aspect ratio is changed. Intuitively, changing aspect ratio will change the distance between pins on different EBs; this leads to change in the delay of the nets connecting these pins. The aspect ratio of the FPGA is also changed according to the aspect ratio of EBs.

Delay Impact

We modify the shape of the EBs from rectangular (aspect ratio of 1:8) to square (aspect ratio of 1:1). The results in Figure 4.10 show that square EBs (1:1) are the most efficient in speed for all applications. This results in a 18.5% speed improvement compared to the 1:8 EBs in type 3 EB position and 20.1% speed improvement in type 4 EB position. Square EBs lead to a better worst-case delay between the EBs, shortening the critical path in our benchmark circuits.

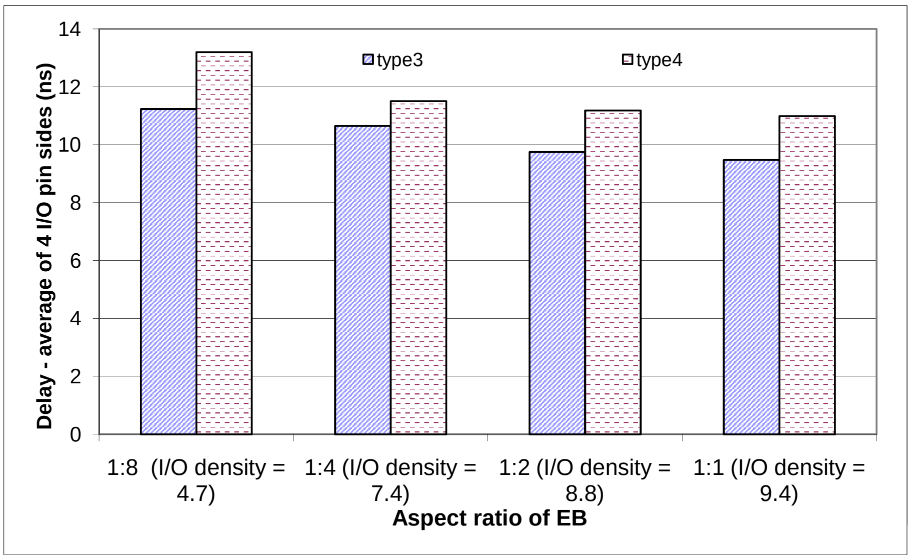


Figure 4.10: Average delay against various EBs' shape for the single EB type FPGA

Routing Area Impact

In contrast to the speed, rectangular shape can help to spread out the routing stress near EBs. As shown in Figure 4.11, the 1:8 EBs save at most 26% the routing resources in type 3 EB position when comparing to the other aspect ratio EBs. It is because the perimeter is longer and I/O density is smaller in higher aspect ratio EB. The I/O density is 4.7 in 1:8 EBs while 9.4 in 1:1 EBs. Therefore, less routing demand is introduced at the edge of higher aspect ratio EB. This effect is less significant if the routing structure is optimised like the case of type 4 EB position.

The routing area of 1:8 EBs is even 7.9% more than others in type 4 EB position. It is because the aspect ratio of FPGA is also changed when changing the EB aspect ratio. In type 4 EB position, the 16

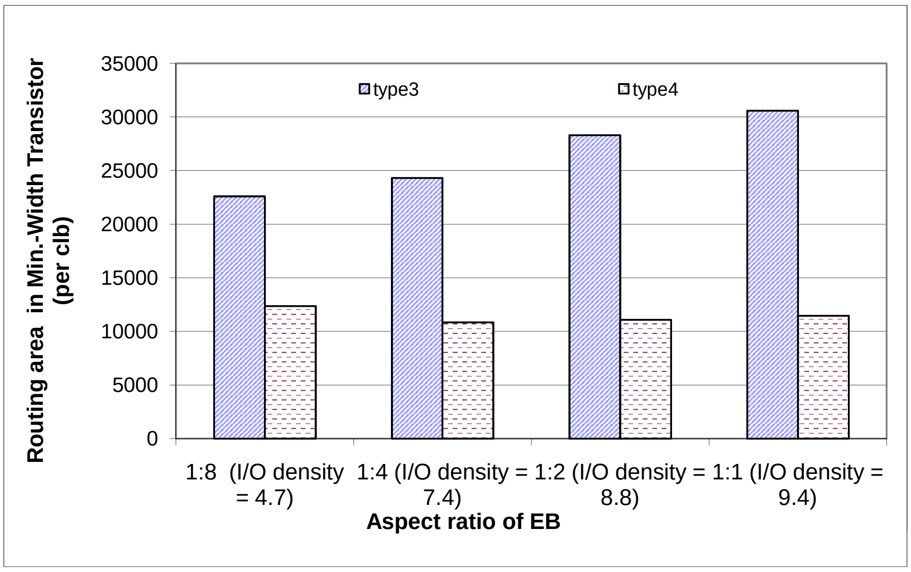


Figure 4.11: Average routing area against various EBs' shape for the single EB type FPGA

EBs are evenly distributed along the vertical and horizontal axises of FPGA (arranged as 4x4 grids of EBs). The width of the FPGA is shorter for higher aspect ratio EBs, therefore the horizontal distance between EBs (D_{eb}) is shorter. The minimum separation of 1:8 EBs is 2 CLBs away ($D_{eb} = 2$), while 1:1 EBs have $D_{eb} = 9$. The routing congestion near the edge of EBs cannot be resolved in short separation distance (see Chapter 5, Section 5.3.1). As a result 1:8 EBs consume more routing resources than others. We will consider the interface after optimised the routing structure in Chapter 5, Section 5.4, since the routing area and delay is different in the optimised hybrid FPGA.

4.6 Result: Multiple EB Types

After finding the best parameters for the single EB type case for speed or area, we examine how embedding more than one type of EB affects performance and routing demand. In our experiments, we explore the EB position and the interconnect flexibility of the fastest system (type 3 EB position), since the routing structure is not optimised here. We will examine the updated interface after optimised the routing structure in Chapter 5, Section 5.4. The size of BRAM is 2×4 CLBs in these experiments.

4.6.1 EB Position

We first explore the effect of the BRAMs position for the floating point hybrid FPGA.

Delay Impact

Figure 4.12 shows the best location is between floating point units which is the type C configuration in Figure 4.4. It is at least 2.5% faster than other configurations. This configuration performs better than the traditional column based BRAM (type A in Figure 4.4) used in Xilinx devices because the connections between floating point units and BRAMs are reduced in this case. In a similar way as shown in Section 4.5.1, placing the embedded blocks closer together reduces the length of the bus-based connections between the embedded blocks which improves performance. In addition, it is reasonable to have 3.3% higher delay than no BRAM architecture because the the BRAMs separate large EBs such that wires between EBs are longer which causes longer net delay.

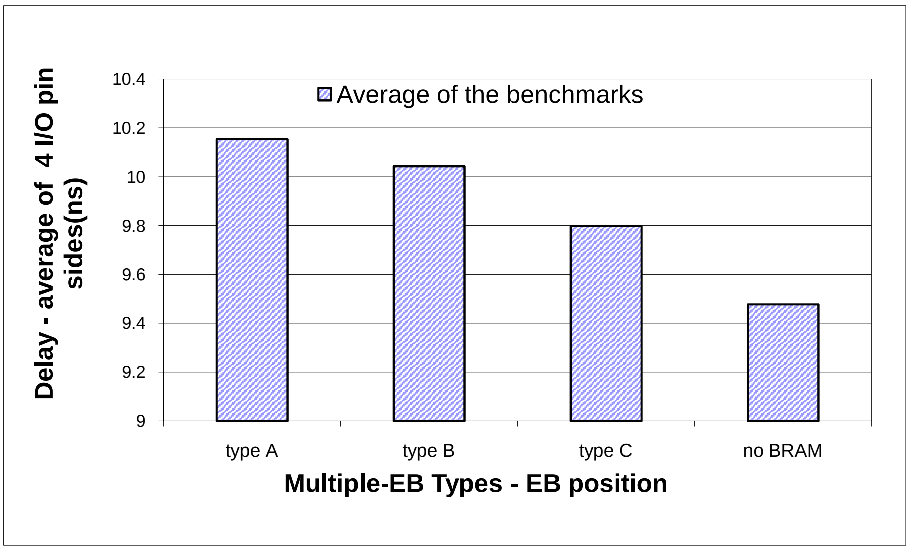


Figure 4.12: Delay against various EBs positions for the multiple EB types FPGA (Figure 4.4)

Routing Area Impact

We also examine the routing area of the multiple EB types FPGAs. We expect the routing demand should be increased after adding high I/O density BRAM between large EBs. Surprisingly, type C configuration uses less routing resources than others by at least 51.8%. And it also reduces 43.7% of routing area in no BRAM FPGA. In type A and type B configuration, the congestion is near the large EBs as they are closely positioned together. Additional BRAM causes more routing near BRAM and requires more routing resources. However in type C configuration, some unused BRAMs separate the large EBs and create more routing space for the connection between EBs. This resolves the congestion at the edge of EBs. The reduction of routing demanded between EBs is more significant than the routing resources required by nets from BRAMs to EBs. As a result, there is major reduction of routing area by using the type C configuration.

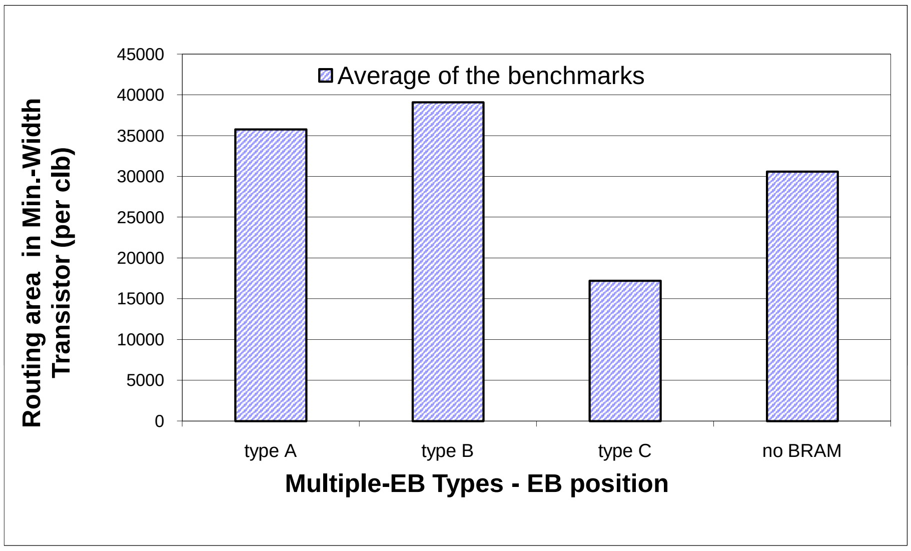


Figure 4.13: Average routing area against various EBs positions for the multiple EB types FPGA (Figure 4.4)

4.6.2 Interconnect Flexibility

Finally, we investigate routing resources for the multiple EB system. Figure 4.14 shows delay for additional channels to 1.2× of minimum channel width. The channel width is observed to be nearly constant which is similar to the case discussed in Section 4.5.3.

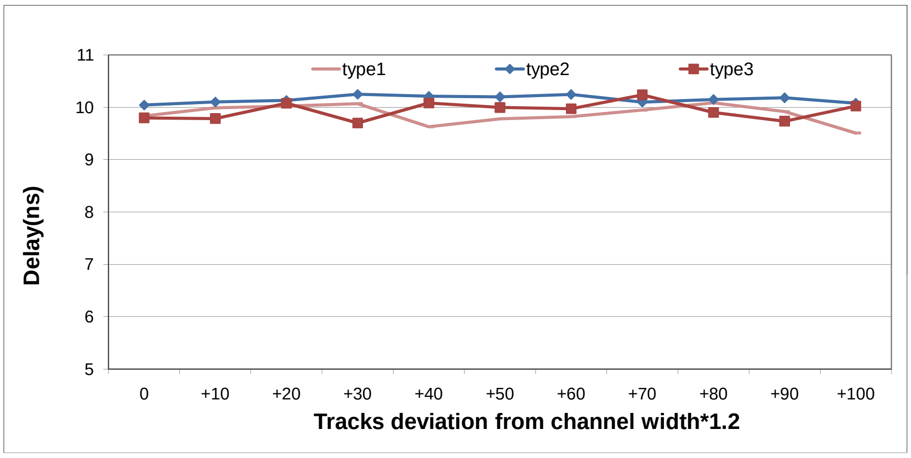


Figure 4.14: Delay against channel width for multiple EB types FPGA

4.7 Summary

This chapter investigates the architecture of the programmable interconnect between coarse-grained blocks and the fine-grained fabric in domain-specific FPGA with embedded floating point blocks. Specifically, we first examine the position of the embedded blocks (EBs) within the FPGA, the placement of the pins on the periphery of the EB, the width of the routing channels surrounding the EB, and the aspect ratio of the EB for single EB type FPGA. After that we explore the EBs position and the channel of multiple EB types FPGAs. We find that (a) the EBs should be positioned close to each other in the middle of the chip for higher speed while positioned evenly in FPGA for better area, (b) the EB's pins should be distributed evenly around the EB, (c) the width of the channels surrounding the EB have little impact on circuit speed, (d) a square EB leads to the most efficient implementation for speed while a rectangle EB reduces the routing demand, (e) smaller EBs should be located between large EB to achieve higher speed, and (f) inserting smaller EBs between large EBs leads to reduction of routing resources being consumed in highly congested hybrid FPGA. Based on the results in this section, the recommendations of designing hybrid FPGA on baseline architecture for different optimisation goal are summarised in Table 4.2. Although our results are specific to the baseline architecture studied, we believe they can be applied to FPGAs containing other types of embedded blocks for computation. Also we will optimise the routing architecture in next chapter based on the results in this chapter.

Table 4.2: Recommendations of designing hybrid FPGA on baseline architecture for different optimisation goals

Single EB type				
	Recommended configuration			
Goal	EB position	Pin location	EB aspect ratio	Channel width
Speed	positioned closely in the middle of the FPGA	arranged evenly on 4 sides of EB	1:1	1.2× of minimum channel width
Area	positioned evenly in the FPGA	arranged evenly on 4 sides of EB	1:1	1.2× of minimum channel width

Multiple EB types, include large EB and small EB				
	Recommended configuration			
Goal	EB position	Pin location	EB aspect ratio	Channel width
Speed	small EBs are positioned between closely packed large EBs	arranged evenly on 4 sides of EB	1:1	1.2× of minimum channel width
Area	small EBs are positioned between closely packed large EBs	arranged evenly on 4 sides of EB	1:1	1.2× of minimum channel width

Chapter 5

Routing Optimisation

This chapter optimises the routing structure for hybrid FPGAs, in which large and high I/O density coarse-grained units are embedded within fine-grained logic fabric. This significantly increases in the routing resources requirement between elements. We investigate the routing demand for hybrid FPGAs over a set of domain-specific applications. The trade-offs in delay, area and routability of the separation distance between coarse-grained blocks are studied. The effects of adding routing switches to the coarse-grained blocks and using wider channel near them to meet extra routing demand are examined. Our optimised architectures are compared to an existing column based architecture. Finally, we discuss the ways to improve the interface between coarse-grained/fine-grained logic of the optimised hybrid FPGA.

5.1 Motivation

The key differentiating feature to traditional fine-grained FPGA in our hybrid FPGAs proposed in Chapter 3 is the presence of large floating point units (FPUs) as embedded blocks (EBs). It has been shown that the presence of a large embedded block affects the routing demand within the fine-grained logic fabric [58]. Altera [42] has removed the large MegaRAM Blocks in Stratix-III and Stratix-IV devices since these large blocks have created a disruption for the routing fabric. In particular, Smith et al. [90] have presented a detailed wirelength model that includes the impact of embedded blocks, and applies this model to a hybrid FPGA containing floating point embedded blocks. The results from the model suggest that the expected wirelength in the general-purpose logic surrounding the embedded block can be as much as 90% larger than if the embedded block does not exist. This suggests that the extra routing demands imposed by the large embedded block should be accommodated in the design of the routing architecture of the fine-grained logic. Existing commercial devices such as Xilinx Virtex-5 [3] arrange the small embedded blocks like memory and DSP in columns. This arrangement may not be efficient for large blocks.

Therefore, we examine the routing structure of the hybrid FPGA with large embedded coarse-grained blocks. We then propose three methods of augmenting the fine-grained routing architecture: (1) separate the embedded blocks to reduce the routing stress, (2) add routing switches on top of the embedded block to increase routability, (3) use heterogeneous channel widths near the embedded blocks to meet the additional routing requirement and compare with column base architecture.

We examine the routing architecture through the benchmarks in Chapter 2, Section 2.5.3. We also employ the FPU as large EB, baseline hybrid FPGA and the VPH work flow described in Section 3.1, 3.2 and 3.3 of Chapter 3 respectively.

5.2 Routing Demand

This section shows experimentally, that the presence of the large EBs causes an increase in the routing requirements near them.

Commercial devices [2, 3] embed smaller blocks such as DSP and memory in column based arrangement as shown in Figure 5.1. The DSP and memory are normally less than 10 CLB tiles. The column based architecture may not be efficient for large embedded coarse-grained blocks, with over 100 CLB tiles. Therefore, we compare the performance of the fine-grained FPGA, column based FPGA, and the baseline architecture for large EBs.

5.2.1 Netlength Demand

First, we study the average netlength required. The average netlength is important since it determines the number of routing resources that must be included in an FPGA, longer netlength requires more routing resources for a net and results in longer net delay.

We examine the netlength and wirelength of fine-grained FPGA, column based and baseline hybrid FPGAs with FPUs as embedded blocks. The EBs in the baseline architecture are closely packed together, with D_{eb}=0. The aspect ratio of EBs in column based FPGA is 2 (rectangular shape, 9x23 CLB tiles), and 4 EB columns are evenly distributed which is similar to Figure 5.1.

Table 5.1 shows that on average, the embedded FPUs in the baseline FPGA increase the netlength of a net by 1.4 times compared to the fine-grained FPGA. In a fine-grained FPGA, all the user logic is implemented in CLBs, which are more flexibility to move closer for shorter connection. In a hybrid FPGA, most of the computation logics and nets are in the fast FPUs, which reduces the CLB usage used by 5.4 times and number of nets by 7.7 times. Therefore, the embedded FPU improves the area and delay.

The remaining small amount of nets in the hybrid FPGA are used to connect EBs and CLBs. However, the large EBs with longer perimeter are not flexible to move. They require long length routing nets to connect to other elements.

The column based FPGA is 20.8% slower and has 18.2% longer netlength than the baseline FPGA. Floating point applications require a series of addition, subtraction and multiplication. The adders and multipliers are normally connected in series (circuit diagram in Chapter 2, Section 2.5.3). The tall FPU and the long distance between FPU columns cause the longer nets and delay between EBs, as shown in Figure 5.1. This also explains why the FPUs should be closely packed together in the middle of FPGA as this can reduce the length of a net.

5.2.2 Congested Region

Next, we investigate the most congested region in both baseline and column based hybrid FPGA.

Table 5.1: Average netlength, delay and area of FPGAs over the 8 benchmarks

FPGA	Average netlength	Av. total wirelength	Av. no. of nets	Av. delay(ns)	Av. CLB area
Fine-grained	15.15	450,439	31,527	16.93	7,394
Baseline	36.61	102,549	4,089	9.37	1,379
Column based	43.30	120,687	4,089	11.32	1,379

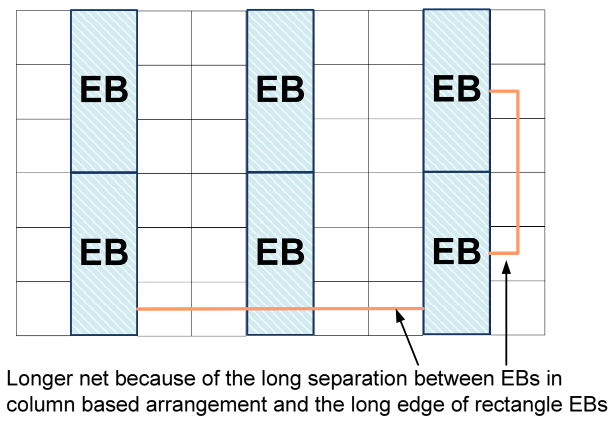

EB
EB
EB
EB
EB
EB
Longer net because of the long separation between EBs in column based arrangement and the long edge of rectangle EBs

Figure 5.1: A column based hybrid FPGA

We examine the wire segments at the edge and one segment next to the edge of EB in the baseline FPGA. On average, tracks in these segments are only 17.82% of the total tracks in the FPGA, but there are about 44% of the routed tracks used for routing near the edge, making it the most congested region.

We use *bgm* as an example to show the congested region. *bgm* uses nearly all CLBs in the baseline and the column based FPGA. Figure 5.2 and Figure 5.3 show the track usage along X and Y channel of *bgm* in the baseline FPGA and the column based FPGA respectively. The peak track usage in both systems are at the edge of EBs. This congestion is caused by the large amount of net connecting from EB to CLBs or another EB. Many CLBs move near to the EBs to reduce the net delay. Therefore, the wire density at this area is very high. As shown in the figures, the column based FPGA spreads the tracks better than the baseline one. The track usage at the EB edge in the column based FPGA is 36.2%, but 44.2% in the baseline FPGA in *bgm*. And the overall EB edge track usage in the column based FPGA is 35.8%. The difference is because of the EB columns are evenly distributed in the column based system. There are enough space to place CLBs around, so the density of connection in this area is reduced and spread to another region. As a result of the less congested region and higher routability, the column based FPGA uses minimum of 83.2 routing channels for successful routing, which is 40% less than the baseline FPGA (137.5 tracks).

Although the speed of the baseline FPGA is much faster than the column based FPGA, more routing tracks are used, which lead to 50.4% decrease in area-delay product. The area-delay product of the baseline FPGA is 345,212 and column based FPGA is 229,582. In order to retain the speed advantage of the baseline FPGA, we propose three routing optimisation schemes to reduce its routing area.

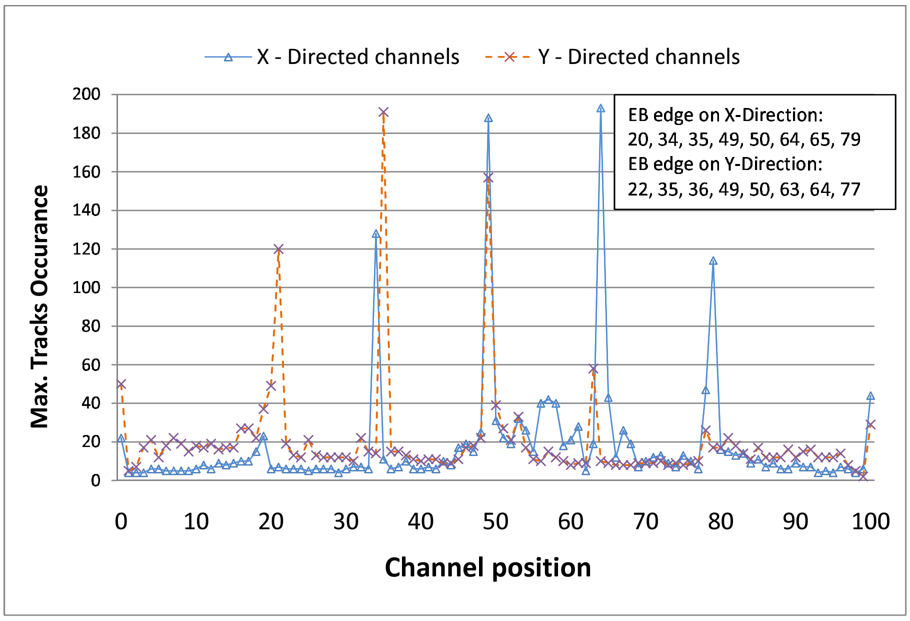


Figure 5.2: Track usage along X-Y channels of *bgm* in the baseline FPGA (100x100 CLBs)

5.3 Optimisation of Routing

In the previous section, we have showed that embedding large EBs leads to increase in routing demand. This routing demand cannot be avoided unless remove the large block, likes MegaRAM in Stratix-I and -II. In this section, we study the trade-offs of three improved architectures to solve the routing requirement, which minimise the area and delay.

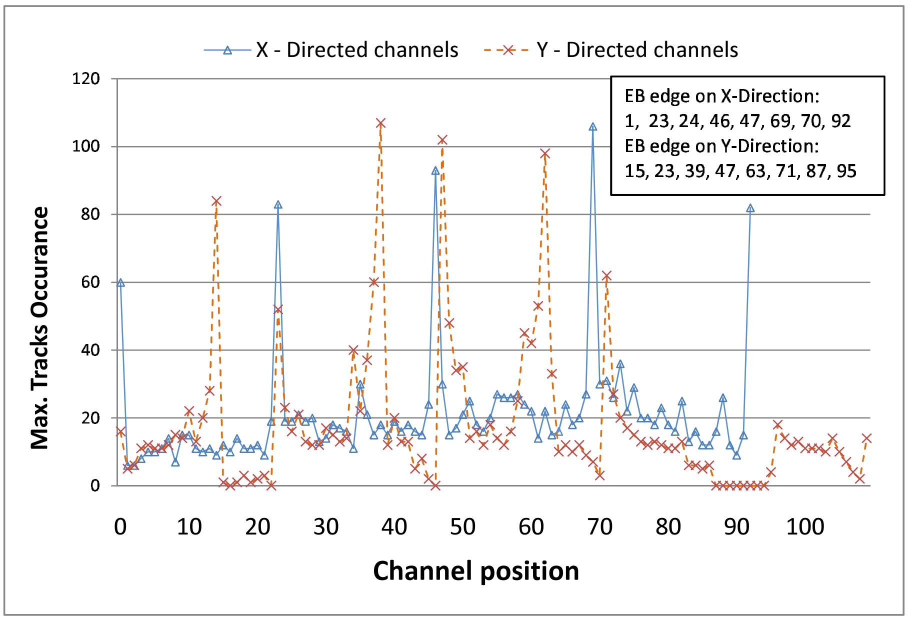


Figure 5.3: Track usage along X-Y channels of *bgm* in the column based FPGA (93x110 CLBs)

5.3.1 Separation Distance between EBs

The distance between EBs (D_{eb}) can be varied in the baseline architecture as shown in Chapter 3, Figure 3.6. Larger D_{eb} can reduce the routing stress to use less routing channel like column based FPGA, but the trade-off is having longer net delay. We examine the trade-offs of different D_{eb} in delay, minimum channel width and routing area in the baseline architecture without switches in EB.

We place and route the benchmarks in different D_{eb} by using 20% tracks more than minimum channel to avoid congestion, which may affect timing analysis. The dot line in Figure 5.4 shows the minimum channel width is optimised at D_{eb}=4, which reduces 66.07% of channel width at D_{eb}=0. That means the routing stress around EBs is minimum when the EBs are 4 CLBs away from each other. Further increases in distance cannot reduce the usage of channel, but increases the delay by at most 15%, which is shown in Figure 5.5. It is expected as the nets from one EB should use longer wires to reach another EB and resulted in lower speed. In addition, the routing area consumption is following the same trend of minimum channel width, where minimum area is at D_{eb}=4. This is because more channels use more routing resources.

Finally, we study the routing area-delay product to determine which D_{eb} is the most optimised for both area and delay. The result in Figure 5.6 (dot line) shows D_{eb}=4 is the best combination, which is at least 6% better than others. The routing area decreases since less routing channels are used. The improvement of routing area is more significant than loss in speed, and therefore D_{eb}=4 achieve the best combination of low delay and high area-efficient design.

The limitation of this method is depended the area of the hybrid FPGA and number of EBs. We cannot use this optimisation method if there is not enough space to separate the EBs.

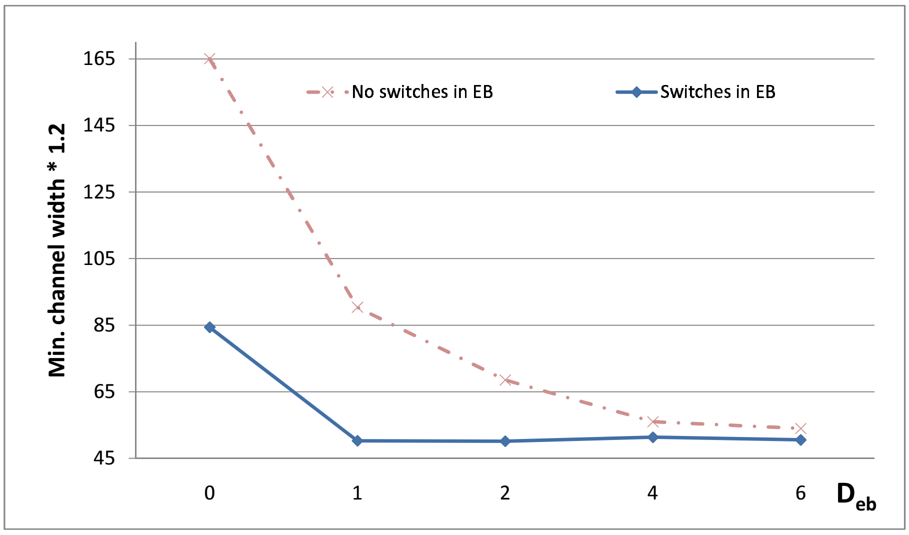


Figure 5.4: Average minimum channel width of the benchmarks at different D_{eb}

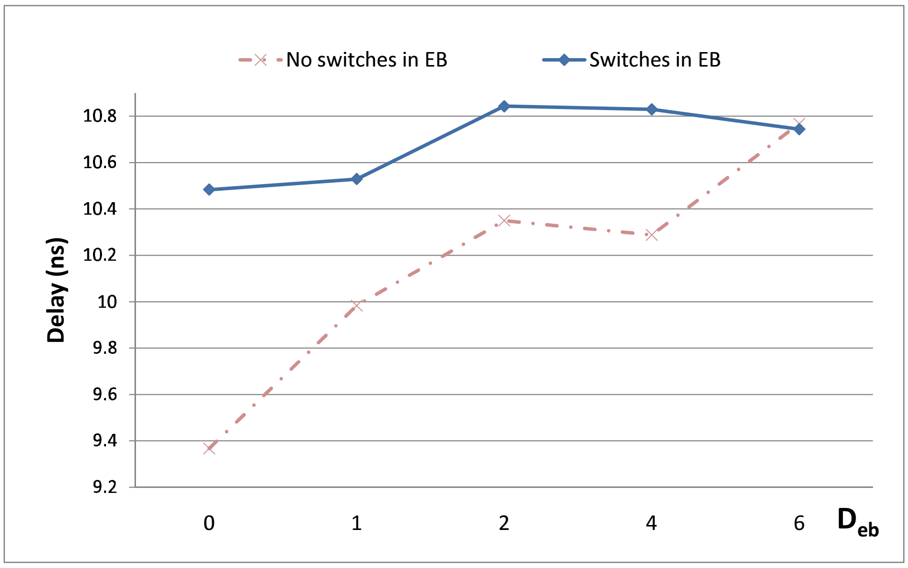


Figure 5.5: Average delay of the benchmarks at different D_{eb}

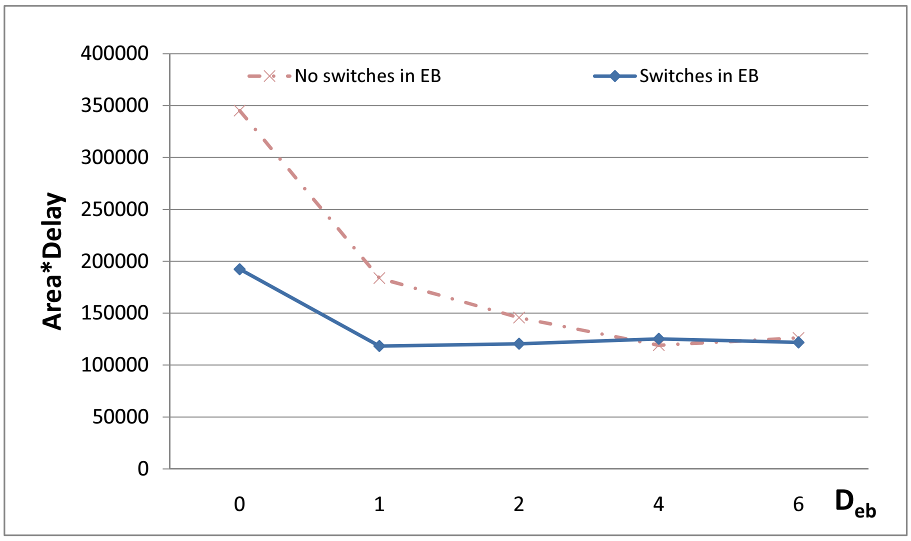
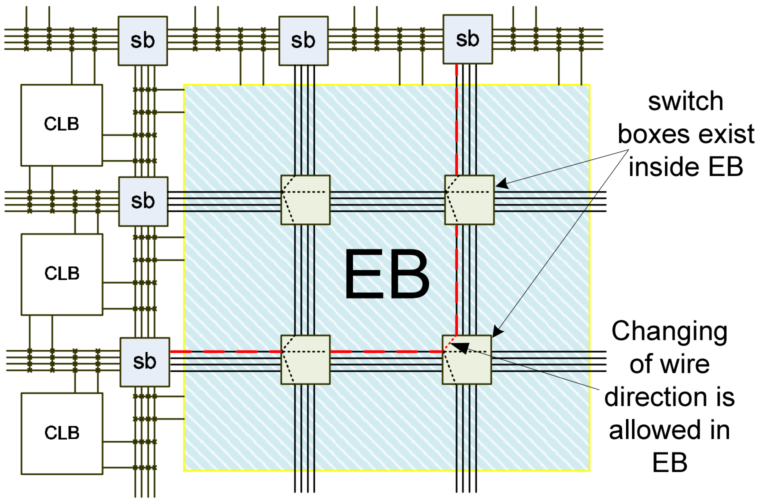


Figure 5.6: Average area*delay of the benchmarks at different D_{eb}

5.3.2 Additional Routing Switches in EBs

In this section, we show that if we extend the routing grid over the embedded block by adding switches within the embedded block, routability can be improved. Figure 5.7 shows the additional switches which allow the direction of signals inside EBs to be changed.



sb
sb
sb
CLB
sb
CLB
sb
CLB
EB
switch boxes exist inside EB
Changing of wire direction is allowed in EB

Figure 5.7: Additional routing switches in an EB

Area Overhead of Routing Switches

The routing wires and transistors of switch boxes are on different metal layers in the FPGA. When the channel width W increases, the area of routing wires and switch boxes increase by different amount. The switch box includes active component switches and SRAM to store configuration bits (the transistors). Schmit and Chandra [139] have found that 100% of the area underneath the switch box is occupied by switch point transistors when $W < 49$ in a $0.35\mu m$, four-metal layer process technology. Otherwise, the switch box area is mainly occupied by the signal routing wires. The wirebound of the routing area is W=49. The wires for the SRAM only occupy the first two metal layers, while all signals wires can occupy any upper layers. Figure 5.8 illustrates an example of this wirebound area concept. The area of switch box is larger than the routing wires in Figure 5.8(a), while the area of routing wires is larger than switch box active components in Figure 5.8(b).

We adopt this result to estimate the area trade-off when adding switch boxes in a EB. We have added 70% of area to EB for signal routing wires as stated in Chapter 3, Section 3.2. Using the area of tri-state buffers, wire width and spacing of $0.13\mu m$ process in Table 3.5. We estimate the ratio of switch box area to wire area is equal to $33.27/W$, which mean W=33 is the wirebound area. Adding switch boxes over these routing wires in EB is insignificant to the wires area if the channel width is greater than 33. We need to account for the area overhead of the additional switches when the channel width is less than 33, where the area of EB would be increased by $33.27/W$.

Performance

We study the performance of adding switches in EB by using the same methodology as in Section 5.3.1. The minimum channel width, delay and area-delay product in different EB separation distances are shown in the solid line in Figure 5.4, 5.5 and 5.6 respectively. There are no switch box area overhead through out the experiment since the minimum channel width is about 50, the routing wires are dominating the routing area.

Additional routing switches in EBs result in a significant reduction of 48.9% in the required routing channel at D_{eb}=0. This is because the additional routing switches split the long and inflexible straight

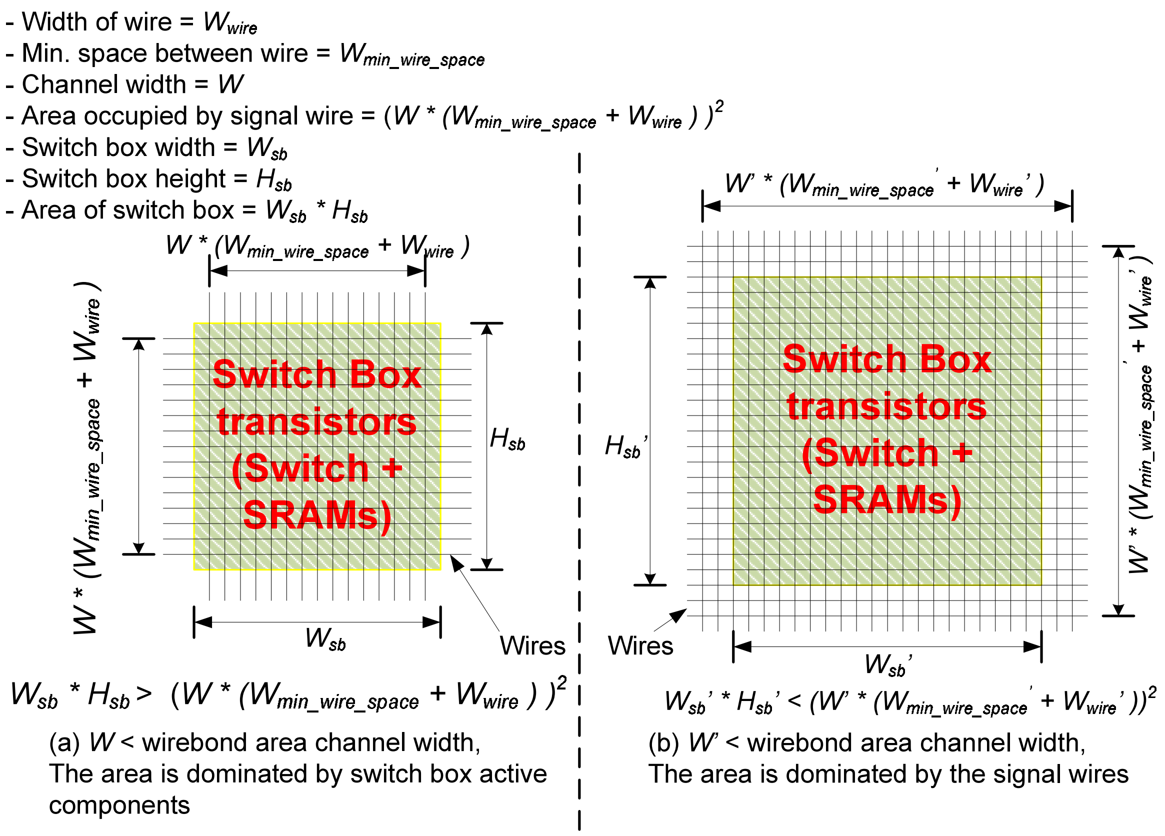

- Width of wire = W_{wire}
- Min. space between wire = $W_{min\_wire\_space}$
- Channel width = W
- Area occupied by signal wire = $(W * (W_{min\_wire\_space} + W_{wire}))^2$
- Switch box width = W_{sb}
- Switch box height = H_{sb}
- Area of switch box = $W_{sb} * H_{sb}$
$W * (W_{min\_wire\_space} + W_{wire})$
Switch Box transistors (Switch + SRAMs)
H_{sb}
W_{sb}
Wires
$W_{sb} * H_{sb} > (W * (W_{min\_wire\_space} + W_{wire}))^2$
(a) W < wirebond area channel width, The area is dominated by switch box active components
$W' * (W_{min\_wire\_space}' + W_{wire}')$
H_{sb}'
W_{sb}'
$W_{sb}' * H_{sb}' < (W' * (W_{min\_wire\_space}' + W_{wire}'))^2$
(b) W' < wirebond area channel width, The area is dominated by the signal wires

Figure 5.8: The area dominated by active components of switch box or signal wires

wires inside EBs, which are more flexible to route a net.

However, the delay is at most 11.9% higher than without EB switches. For no EB switches, the long segment in the EB is faster since the routing on this wire does not pass through a switch. And critical paths can be routed on these fast tracks. The delay is generally increased for longer wire segment not in the EB since the signal needs to pass through several internal switch points (fully populated switches for wire segments are used) and the equivalent resistance is larger.

We verify this explanation by examining the delay at different segment length which is shown in Figure 5.9. As the length increases, the effect of the speed advantage of the long wires in EBs is less significant compared to other wire segments. Therefore, the delay of the two systems are the same at segment length 12, which is close to the width and height of the EB. The optimal speed is at segment length 8 and the best area-delay efficient is at segment length 4, this matches the result in [22] for different segment lengths.

Adding switches in the EB achieves the best area-delay product at D_{eb}=1, which has the same performance as no switches in EB at D_{eb}=4. There is 65.69% reduction of area-delay product comparing to the baseline architecture and 48.4% improvement of performance to the column base architecture without switches in EB. By comparing the column base architecture with switches in EB, the optimised baseline architecture consumes the same amount of routing area, but achieves 5.07% speed improvement. The switches in EB increase the routability significantly, and separating EBs slightly is enough to minimise the minimum channel width and obtains highest performance.

The limitation of this optimisation method is the extra effort of ASIC design of EB with switches. With the additional routing switches, simple standard cell library ASIC EB design may not be suitable (an example of standard cell library ASIC design of floating point unit is shown in Chapter 3, Section 3.1). Manual place and route of routing switches inside EB is required.

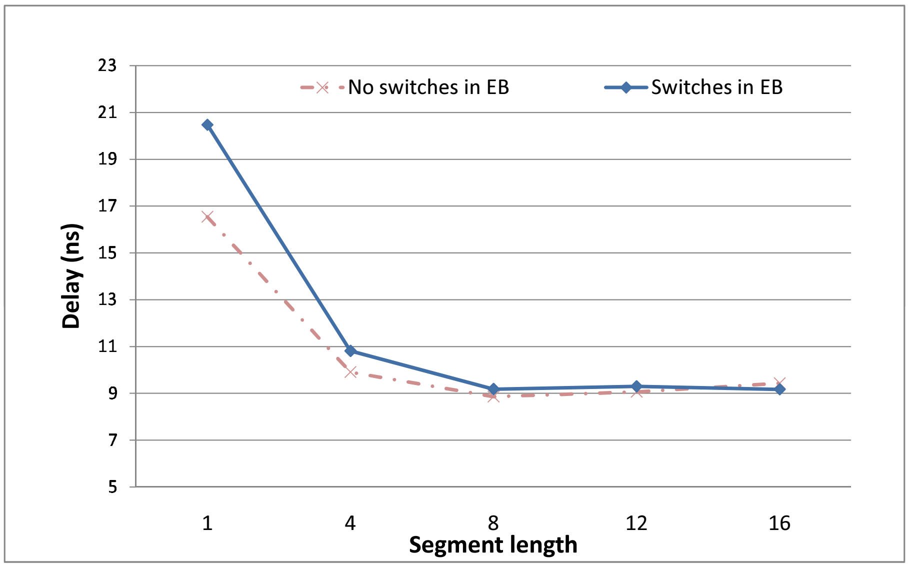


Figure 5.9: Average delay of the benchmarks at different segment length

5.3.3 Extra Routing Tracks

Betz et al. [21] have suggested using wider channel in the center of fine-grained FPGA. This architecture does not improve the routability. It is because all circuits are forced to route most of their connections through the predefined wide channel. But those connections can be spread out in uniform FPGA. The large EBs introduce large routing demand at the edge of EBs, which cannot be spread out easily. Therefore, in this section, we show how this extra routability can be provided with low overhead by strategically inserting extra tracks in the channels surrounding the EBs.

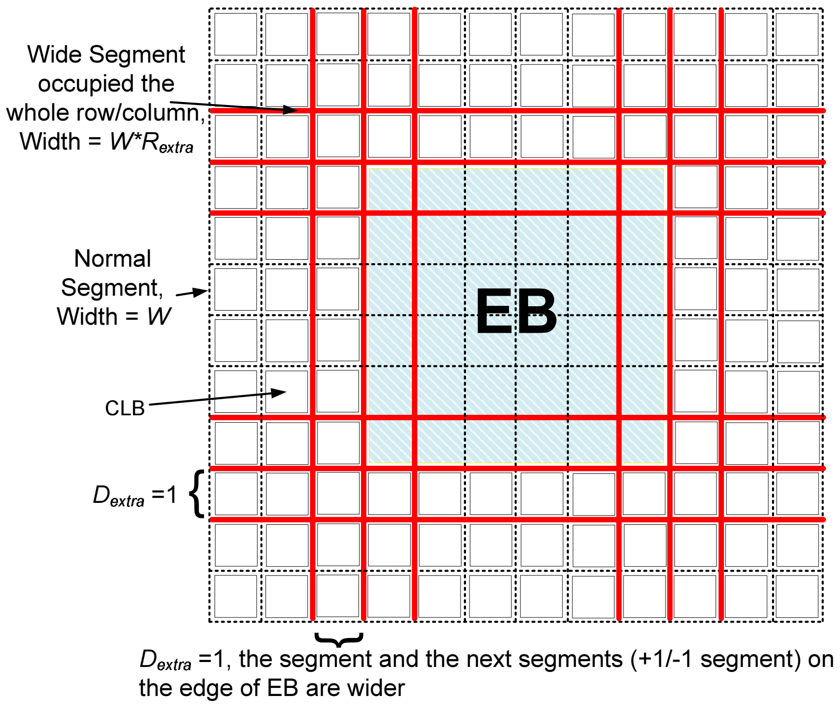

Wide Segment occupied the whole row/column, Width = W*Rextra
Normal Segment, Width = W
CLB
Dextra =1
EB
Dextra =1, the segment and the next segments (+1/-1 segment) on the edge of EB are wider

Figure 5.10: Extra routing tracks near the EB

Figure 5.10 shows an example of employing extra routing tracks around the EBs. The normal segment width is W, R_{extra} is the ratio of the number of tracks in wider segment to the tracks in normal segment. D_{extra} is the distance (in term of CLB length) from the edge of the EB, the width of the wider segment is $W * R_{extra}$ within this distance. We investigate the impact of R_{extra} and D_{extra} on routability, area and delay of the baseline hybrid FPGA and the optimised FPGA in previous sections. We select four systems to evaluate this method:

(1) $FPGA\_1$: D_{eb}=0, no switches in EBs,

(2) $FPGA\_2$: D_{eb}=0, switches in EBs,

(3) $FPGA\_3$: D_{eb}=4, no switches in EBs,

(4) $FPGA\_4$: D_{eb}=1, switches in EBs.

The area-delay product of using extra routing tracks in these systems is shown in Figure 5.11. Extra routing tracks are efficient to increase the performance of the highly congested $FPGA\_1$ by 35.5% at R_{extra}=3 and D_{extra}=2. The reason is that the wide segment routes most of the high density connections from EBs in the congested region, and only a small amount of connections are used in normal segments. As R_{extra} increases, the number of tracks in normal segments decreases. This reduces the routing area significantly. We find that the delay variation in each system is less than 3%. We believe that once there are enough tracks to route, the router can route the net efficiently in the same system although there are congestion.

Surprisingly, $FPGA\_3$ is less efficient starting from R_{extra}=1.5 in both D_{extra}=1 and D_{extra}=2. And $FPGA\_4$ does not improve by using extra routing tracks. $FPGA\_3$ and $FPGA\_4$ are very flexible to route, but the minimum channel width cannot be further reduced. Therefore, the extra tracks near the edge of EBs introduce additional area to the optimised system.

We further study the performance of highly congested $FPGA\_1$ by increasing R_{extra} and D_{extra}. Figure 5.12 shows that in general, the system is less efficient when $R_{extra} > 3$ or $D_{extra} > 3$. The best area-delay product is at R_{extra}=3, D_{extra}=3 or R_{extra}=3, D_{extra}=2, which is at least 3.9% better than others. The normal channel width decreases by using larger values of R_{extra} and D_{extra}, as there are more connections near EBs routed on the wider channels. However, not all of the wires in the wider channels are used for routing, especially the wires which are far away from EBs. The increase of the unused extra wires is greater than the decrease of the wires in normal channel when $R_{extra} > 3$ or $D_{extra} > 3$. The delay is nearly the same in different R_{extra} and D_{extra}. As a result, there is minimum area-delay product in this routing optimisation method. However, the best area-delay product of this method is still 200% of the best area-delay of FPGA_3 and FPGA_4. Although this optimisation method is not efficient than the previous two methods, this is easily implemented by adding extra wires near EBs.

5.4 Ways of Improving the Interface: Single EB Type Optimised FPGA

In Chapter 4, we have studied the interface trade-offs between coarse and fine-grained logics for the hybrid FPGAs without routing optimisation. The interface parameters may have different impacts on the delay, area and routing flexibility in the optimised routing FPGA. Therefore we need to improve

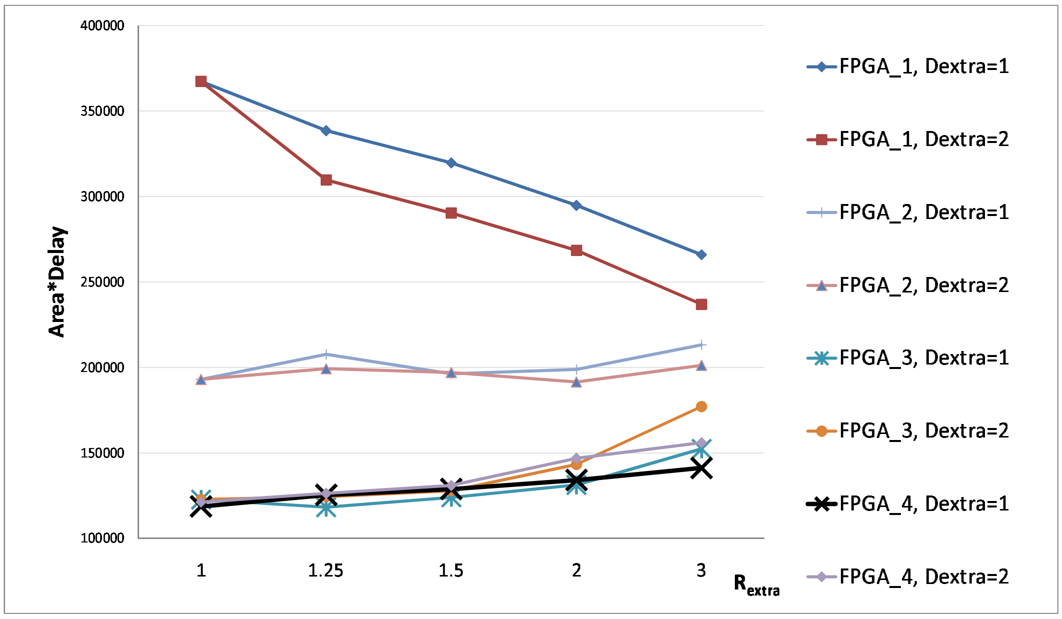


Figure 5.11: The average area-delay product of the benchmarks at different R_{extra} and D_{extra} values

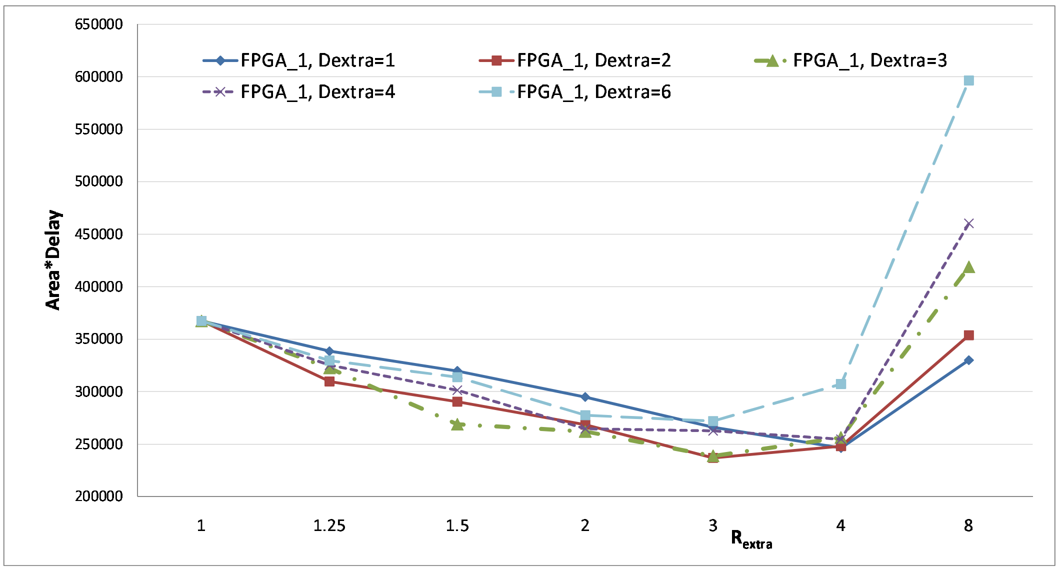


Figure 5.12: The average area-delay product of the benchmarks at different R_{extra} and D_{extra} values for $FPGA\_1$

the interface after we have optimised the routing architecture in the last section.

This section examines the same interface parameters as discussed in Chapter 4, Section 4.2. They include: EB position, pin location of EB, interconnect flexibility and EB aspect ratio of the single EB type hybrid FPGA. Assuming that we employ additional switches in EB and D_{eb}=1 for type 3 EB position as the optimisation scheme for the studies. This scheme provides the highest routing flexibility with a little decrease in speed.

5.4.1 EB Position

First, we revise the position of embedded block in the single EB type hybrid FPGA, which is defined in Chapter 4, Figure 4.1. The delay and area impacts are shown in Figure 5.13 and Figure 5.14 respectively.

The delay impact is consistent with the result in Section 4.5.1. The average delay of type 3 configuration is 10.375ns, which is at least 8.6% less than other configurations. Surprisingly, the result of routing area impact is different from Section 4.5.1. It is because the congestion near the EBs is completely solved after adding switches in EBs. However, type 1 and type 2 use 31% and 14% more in routing area than type 3 (15 tracks and 6 tracks more in each channel) respectively. EBs are close to I/O pads of FPGA in both type 1 and type 2, there are a lot of connections from I/O pads to CLBs and EBs. This results in congestion near I/O pads and EBs and consumes more routing area.

5.4.2 Pin Location

Next, we explore the pin location of EB for the optimised system. The pin location configuration is shown in Chapter 4, Figure 4.2. Type 3 and type 4 EB position are examined as the same as Chapter 4, Section 4.5.2. Figure 5.15 shows the delay of different I/O configurations of EB. The delay of all I/O configurations are similar in both type 3 and type 4 EB position, the deviation is less than 1.3%. This matches the result in Chapter 4, Section 4.5.2. The I/O configuration does not have large impact on delay. Type 4 EB position is slower than type 3 EB position since the EBs are far away from each other, the connections between them are long.

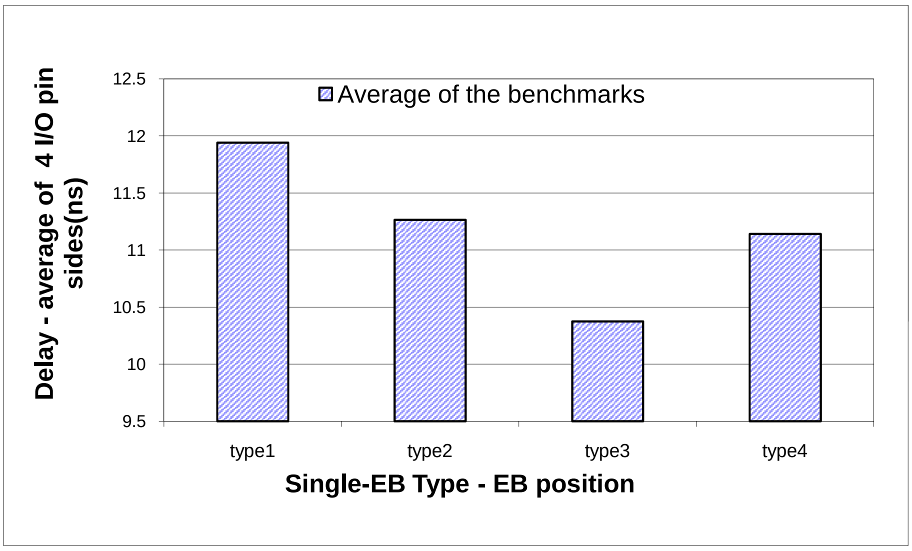


Figure 5.13: Delay against various EBs positions for the single EB type FPGA (in Figure 4.1), switches are in EB

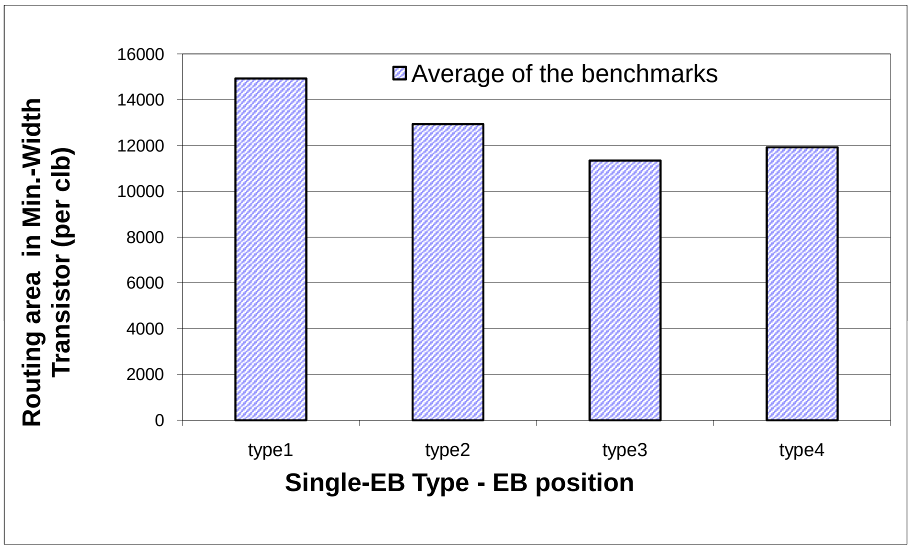


Figure 5.14: Routing area against various EBs positions for the single EB type FPGA (in Figure 4.1), switches are in EB

The routing area impact is consistent with non-optimised hybrid FPGA as shown in Figure 5.16. Four sides I/O connection has lowest I/O density which results in less routing area. The maximum reduction is about 67% when comparing to one side connection. The main different to non-optimised hybrid FPGA is the little deviation of routing resources using in both type 3 and type 4, which are less than 3%. The routing congestion in both systems is resolved, therefore they have similar routing demand for the same I/O density.

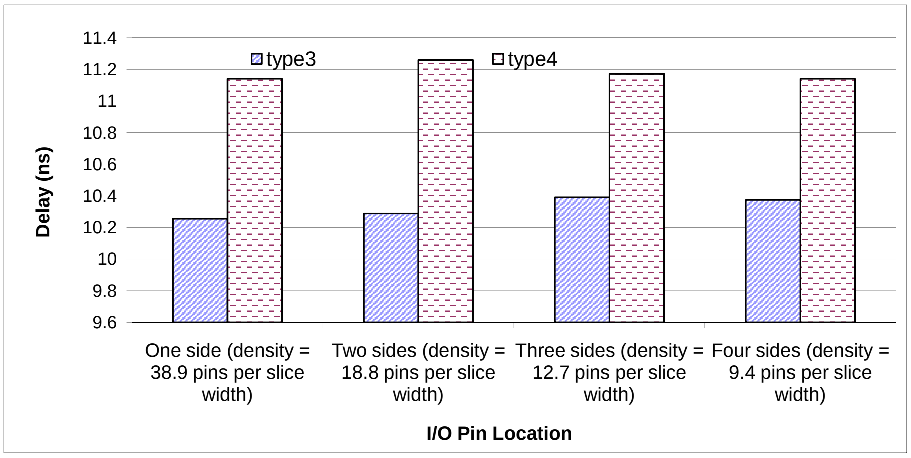


Figure 5.15: Delay against various I/O configurations for the single EB FPGA (in Figure 4.2), type 3 and type 4 EB position, switches are in EB

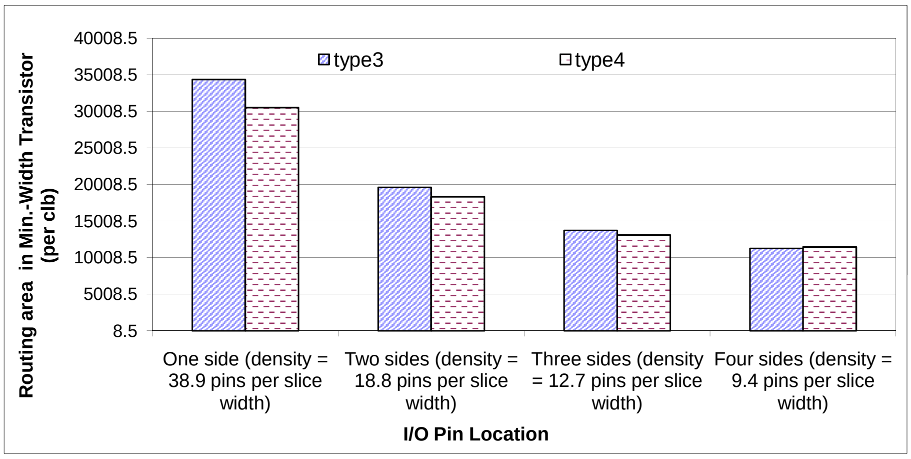


Figure 5.16: Routing area against various I/O configurations for the single EB FPGA (in Figure 4.2), type 3 and type 4 EB position, switches are in EB

5.4.3 Interconnect Flexibility

In Chapter 4, Section 4.5.3, additional channels to 1.2× of minimum channel width does not have significant impact on delay. We examine this interconnection flexibility in this section and the result is shown in Figure 5.17. The result is the same as the non-optimised hybrid FPGA as the critical paths being routed efficiently so once the circuit is routable. Therefore the delay is nearly constant even there are more tracks in a channel.

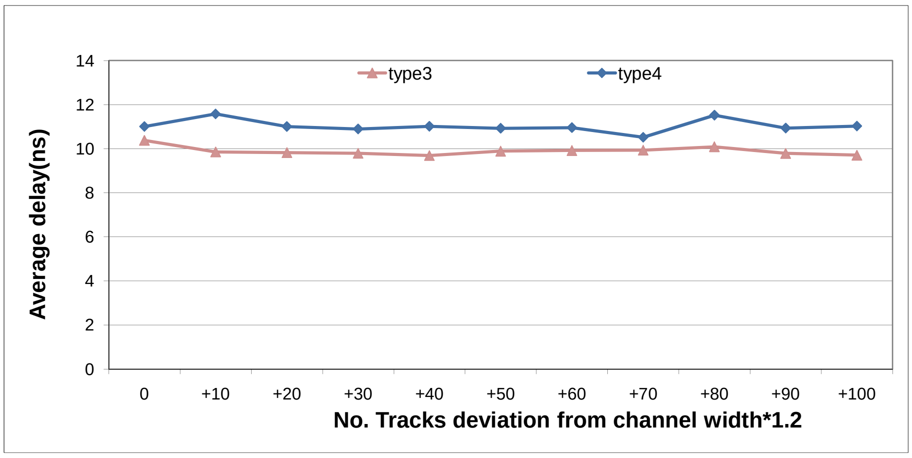


Figure 5.17: Average delay against channel width for the single EB type FPGA, switches are in EB

5.4.4 EB Aspect Ratio

Next we update the result of the EB aspect ratio after optimising the routing architecture by referring to Chapter 4, Figure 4.3. In Figure 5.18, we find that higher aspect ratio EB introduces large delay, where 1:1 EB aspect ratio is at most 10% less than others in type 3 EB position and 20.5% less than others in type 4 EB position. The tall of EB introduces long connections between EBs.

In Chapter 4, Section 4.5.4, we show that longer perimeter in higher aspect ratio EB can reduce the routing demand. Contradictorily, there are no significant improvement of routing area in highly optimised routing architecture in both type 3 and type 4 EB position. The average deviation of routing area is about 2% and 3% in type 3 and type 4 EB position respectively. The additional routing switches in EB allow very flexible routing and all EBs are separated by 1 CLB. The congestion near the edge

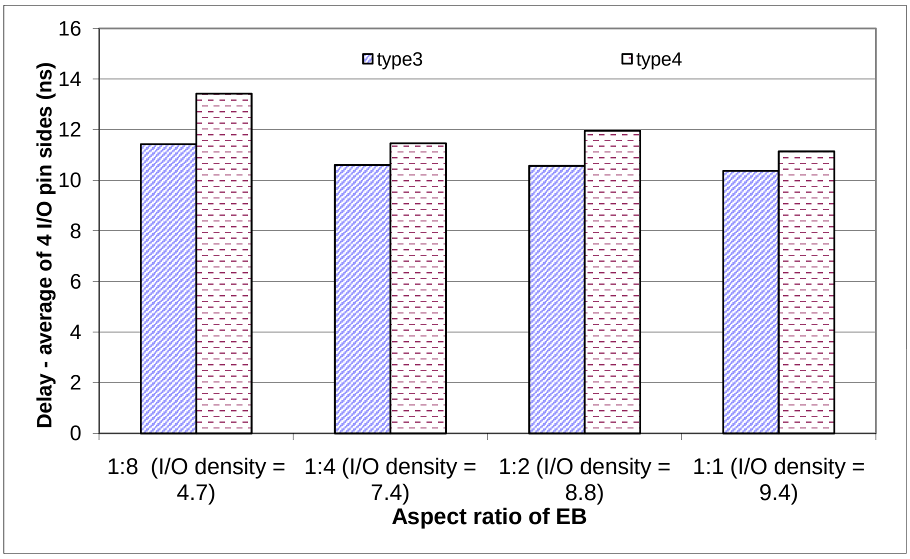


Figure 5.18: Average delay against various EB aspect ratio for the single EB type FPGA, switches are in EB

of EBs is resolved, therefore the high I/O density EB (square EB) requires similar routing resources as lowest I/O density EB (1:8 EB aspect ratio).

5.4.5 Area-Delay Product Comparison

Finally, we compare the area-delay product of four selected systems with and without routing optimisation which achieve better speed or routing area, they are:

1. FPGA_A - Type 3 EB position, no switches in EB, 4 sides I/O configuration, 1:1 EB aspect ratio (Advantage in speed),
2. FPGA_B - Type 4 EB position, no switches in EB, 4 sides I/O configuration, 1:1 EB aspect ratio (Advantage in routing area),
3. FPGA_C - Type 3 EB position, switches in EB, 4 sides I/O configuration, 1:1 EB aspect ratio (Advantage in routing area),
4. FPGA_D - Type 4 EB position, switches in EB, 4 sides I/O configuration, 1:1 EB aspect ratio (Advantage in routing area).

The area-delay product of the four systems is shown in Figure 5.19. Although FPGA_A is faster than the second fast one (FPGA_C) by 8.7%, it is highly congested at the edge of EBs, which requires large routing area and leads to high area-delay product. FPGA_C has the best area-delay product among the four systems. FPGA_C achieves the best area-delay product which is 6.5% better than FPGA_B and 11.4% better than FPGA_D. There is little routing congestion in FPGA_B, FPGA_C and FPGA_D as the routing in them is very flexible. FPGA_C consumes only 0.9% and 4.9% less routing area than FPGA_B and FPGA_D respectively. The EBs in FPGA_C are positioned closely, therefore it can have higher speed. FPGA_C is 5.6% and 6.9% faster than FPGA_B and FPGA_D respectively. Overall, FPGA_C is the best choose of the architecture, which has a balanced speed and area for the single EB type FPGA.

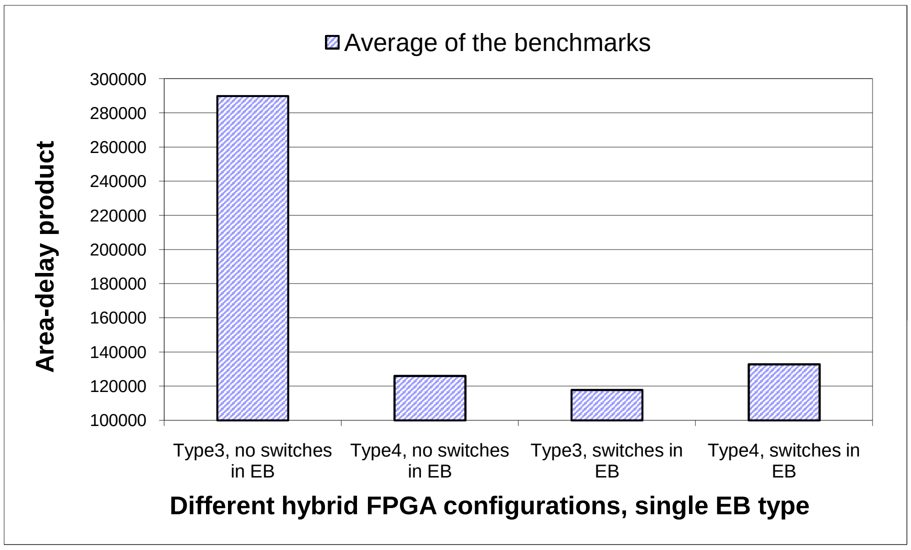


Figure 5.19: Area-delay product against various hybrid FPGA configurations, single EB type only

5.5 Ways of Improving the Interface: Multiple EB Type Optimised FPGA

After comparing the single EB type FPGA, we discuss the ways of improving the interface of the multiple EB types FPGA by following the same parameters as stated in Chapter 4, Section 4.3. The same embedded BRAM memory is used. We assume that the EBs are packed in the middle of FPGA (Type 3 EB position for the single EB type FPGA) with 4 sides I/O pin configuration. Switches are inside EB and D_{eb}=1 as the optimised FPGA. These architectural configurations have achieved the best performance in the studies in previous sections.

5.5.1 EB Position

First, we examine the EB position for the multiple EB types FPGA as defined in Chapter 4, Figure 4.4 Figure 5.20 and Figure 5.21 show the delay and area impact of different position of BRAM. The delay of type C is 2.4% and 5.4% less than type A and type B configuration respectively. It has similar delay as no BRAM existing in benchmarks, with only 0.9% different. In type A and type B, the BRAMs are not close to some of the large EBs, therefore the net delay connecting BRAM and large EB is longer. The BRAMs are between the large EBs in type C configuration, the connections between BRAMs and EBs are short and result in small net delay in type C. The small BRAMs separate the large EBs by D_{eb}=2, which is similar to the D_{eb}=1 in no BRAM FPGA and leads to similar delay between them.

The routing area is opposite to the result in Chapter 4, Section 4.6.1. Inserting small BRAMs between large EBs increases the routing demand. It is because the congestion between large EBs is small, larger separation between EBs by inserting small BRAMs close to them cannot improve the routability. Instead, the connections between high I/O density BRAMs and EBs require additional routing resources as the case in type B and type C. In type A, the BRAMs are not positioned close to the large EBs, where BRAM columns are at least 4 CLBs away from large EBs. There are enough space to spread the extra routing stress for connection between BRAMs and EBs in type A. Therefore, type B and type C configuration use 96.6% and 55% more routing area than type A. In addition, type A uses 4.8% more routing resources than the no BRAM FPGA because of the additional connections between BRAMs and EBs.

5.5.2 Interconnect Flexibility

We study to impact of additional tracks to the 1.2$\times$ minimum channel width. The result is consistent to Section 4.6.2, where the additional tracks have no impact to the delay as the critical paths being routed efficiently once the circuit is routable.

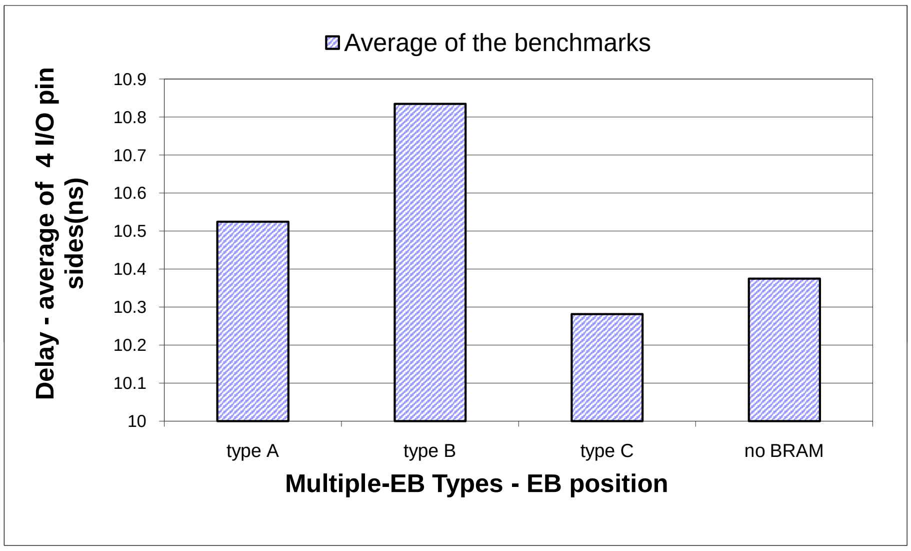


Figure 5.20: Delay against various EBs positions for the multiple EB types FPGA (Figure 4.4), where switches are in EB

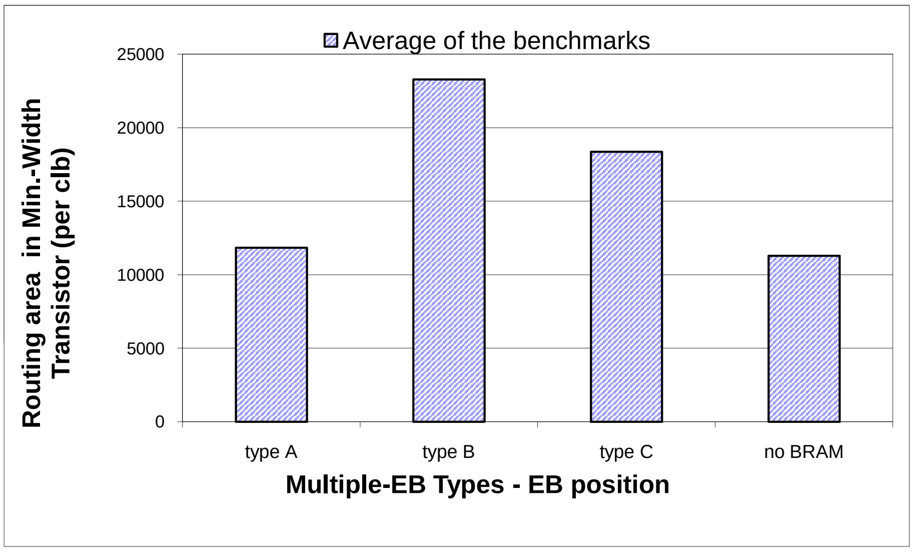


Figure 5.21: Average routing area against various EBs positions for the multiple EB types FPGA (Figure 4.4), switches are in EB

5.5.3 Area-Delay Product Comparison

Finally, we compare the area-delay product of six selected multiple EB types systems in Figure 5.22. Obviously, type A configuration with switches in EBs (The small BRAMs are arranged in columns and large EBs are closely packed in the middle of FPGA) achieves the best area-delay product, which at least 26.2% better the others. Type A uses the least routing area with reasonable speed makes it the most area-delay efficient architecture for multiple EB types FPGA.

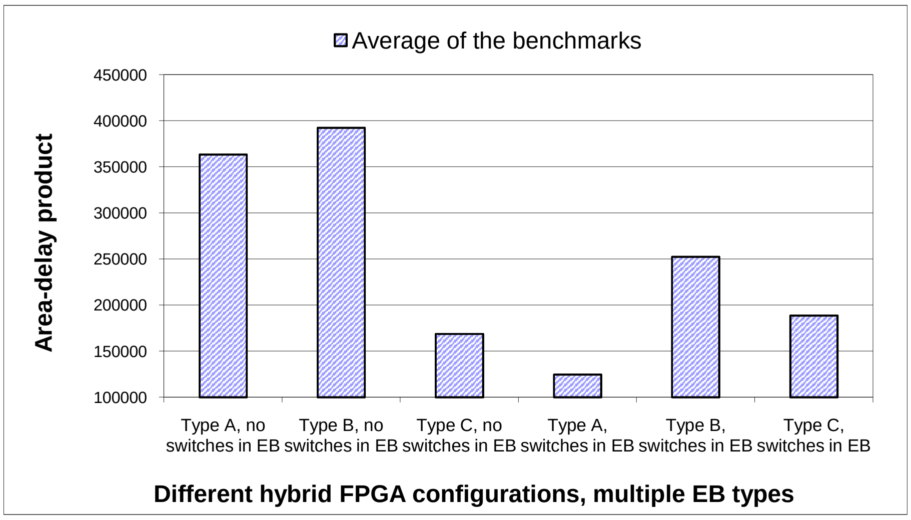


Figure 5.22: Area-delay product against various hybrid FPGA configurations, multiple EB types

5.6 Summary

Hybrid FPGAs require high routing resource demand. This is because the coarse-grained embedded blocks require high density connection to other elements. We explore four interconnection parameters in order to reduce routing area and net delay. First, we show the extra demand of routing requirements by examining channel width, segment length and netlength. Second, we examine the effect of the separation distance between EBs. Third, we study the trade-offs when we add switches on embedded blocks such that switching of routing direction is allowed inside the blocks. Finally, we add extra wires surrounding coarse-grained units to accommodate high density connection. We find that (a) the routing nets are mainly condensed near the edge of EBs, which occupy 44% of tracks usage, (b) both separating EBs and adding switches on EBs systems perform about 48.4% better than existing column based architecture in term of area and delay efficiency. (c) extra wires near the edge of EBs is efficiently improving the area of highly congested system by 34.9%, but cannot further improve the system optimised by separating EBs and adding switch on EBs.

We study the ways of improving the interface between coarse/fine-grained logic in the optimised hybrid FPGA. We find that the features of the interface of non-optimised hybrid FPGA include: (a) the EBs should be positioned close to each other in the middle of the chip to achieve the best area-delay product, (b) a square EB leads to the most efficient implementation, (c) smaller EBs should be located in column to avoid introducing extra routing demand to the large EBs. Based on the results in this chapter, we recommend which optimisation method should be used for which design goal in Table 5.2 and the interface for the optimised FPGA in Table 5.3.

Table 5.2: Recommendations of routing optimisation of hybrid FPGA on baseline architecture for different optimisation goals

	Optimisation method	
Goal	*Enough space to separate EBs or additional routing switches in EB are available*	*Not enough space to separate EBs and additional routing switches in EB are not available*
Speed	(a) D_{eb}=4 and no additional switches in EB	(c) extra wires near the edge of EB R_{extra}=3 and D_{extra}=2 or R_{extra}=3 and D_{extra}=3 (less efficient than method (a) and (b))
Area	(b) additional routing switches in EB and D_{eb}=1	method (c)
Balanced speed and area	optimisation method (a) or (b)	method (c)

Table 5.3: Recommendations of the interface for the optimised FPGA

	Recommended configuration for single EB type FPGA using method (b) in Table 5.2			
Goal	EB position	Pin location	EB aspect ratio	Channel width
Speed/area/balanced speed and area	positioned closely in the middle of the FPGA	arranged evenly on 4 sides of EB	1:1	1.2× of minimum channel width

	Recommended configuration for multiple EB types FPGA using method (b) in Table 5.2			
Goal	EB position	Pin location	EB aspect ratio	Channel width
Speed	small EBs are positioned between closely packed large EBs	arranged evenly on 4 sides of EB	1:1	1.2× of minimum channel width
Area/balanced speed and area	small EBs are arranged in column, large EBs are closely packed with D_{eb}=1	arranged evenly on 4 sides of EB	1:1	1.2× of minimum channel width

Chapter 6

Optimisation of Coarse-grained Floating Point Units

This chapter introduces a novel methodology to optimise coarse-grained floating point units (FPUs) in a hybrid FPGA. We employ common subgraph extraction to determine the number of floating point adders/subtractors (FAs), multipliers (FMs) and wordblocks (WBs) in the FPUs. We first study the area, speed and utilisation trade-offs of the selected FPU subgraphs in a set of floating point benchmark circuits. We then explore the impact of density and flexibility of FPUs on the system in terms of area, speed and routing resources. We derive an optimised coarse-grained FPU by considering both architectural issues and system-level issues.

6.1 Motivation

In a modern commercial FPGA device, different simple fixed point computation elements are connected together to achieve higher speed. An example for Xilinx Spartan-3A includes XtremeDSP DSP48A [140]. The DSP48A slices support many independent functions, including multiplier, multiplier-accumulator (MACC), pre-adder/subtractor followed by a multiply-accumulator, multiplier followed by an adder, wide bus multiplexers, magnitude comparator, or wide counter. The architecture also supports connecting multiple DSP48A slices to form wide math functions, DSP filters, and complex arithmetic without the use of the general FPGA fabric.

In our proposed FPU in Chapter 3, Section 3.1, Floating point adders/subtractors (FAs) and floating point multipliers (FMs) contain several basic functional elements such as barrel shifters, adders and multipliers. Wordblocks (WBs) are used for the bitwise operation of the floating point number such as comparison, shifting, latch and logical operation. Constructing hard circuit WBs, FAs and FMs, which are composed of basic functional elements, results in a more compact block with higher speed, but less flexibility. Optimising these hard circuits is essential. Higher density and speed FPUs can improve the overall area and delay of the floating point hybrid FPGA. We assess the impact of FPUs on hybrid FPGAs in terms of area, speed, routing resources and flexibility. The number and connection of WBs, FAs and FMs are determined by considering common subcircuits in a selected set of floating point benchmark circuits. Common subgraph extraction has been employed to find efficient arithmetic units over a set of benchmark circuits [57]. We adapt this method to study the combination of wordblocks, floating point adders/subtractors and multipliers. The FPUs constructed using common subgraph extraction do not include the feedback path in Chapter 3, Figure 3.2. Future work will involve the inclusion of feedback path in FPU using the common subgraph methodology.

6.2 Optimisation Parameters

In this section, we optimise the internal connection structure and the number of the WBs, FAs and FMs. If more WBs, FAs and FMs are inside an FPU (Figure 6.1(b)), greater area and speed improvement can be achieved, but the whole FPU is wasted if not used. In Figure 6.1(a), FPU1 with fewer WBs, FAs and FMs would waste fewer resources if not used, but we need a large amount of fine-grained elements to support WBs, FAs or FMs. Therefore, choosing a suitable number of WBs, FAs and FMs is important. We consider the FPU in the following parameters:

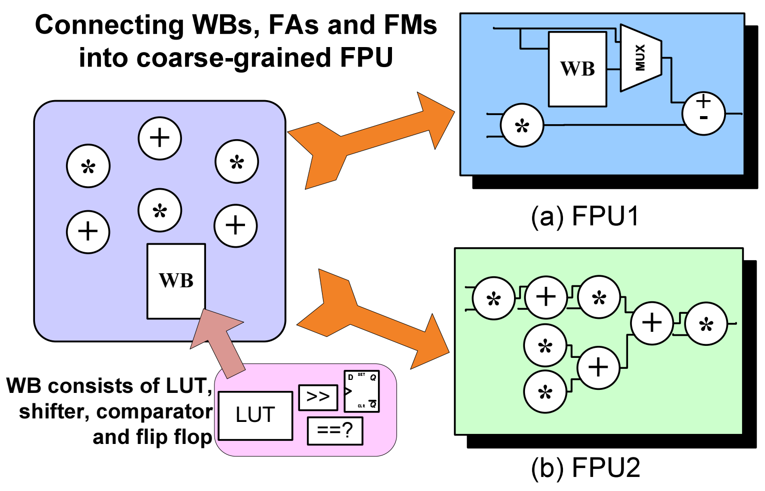

Connecting WBs, FAs and FMs into coarse-grained FPU
WB
MUX
WB
WB consists of LUT, shifter, comparator and flip flop
LUT
>>
==?
(a) FPU1
(b) FPU2

Figure 6.1: Connecting WBs, FAs and FMs into different coarse-grained FPUs

6.2.1 Internal Optimisation of FPU

The WBs, FAs and FMs in FPUs can be connected in different orders as shown in Figure 6.1. We consider the performance of individual FPUs by connecting such elements using commonly found connection patterns.

6.2.2 System-Level Optimisation

Based on the different FPU architectures from common subgraphs, we optimise the performance of the hybrid FPGAs by selecting the FPUs in the following ways:

- *Density of FPU*. An FPU with more computational elements achieves greater reduction since all elements can be closely packed. However, this may require more routing resources for the connection between the coarse-grained block and fine-grained block. Also the flexibility decreases, since it is difficult to reuse in another application.
- *Flexibility of FPU*. FPUs are wasted when not used. The FPUs can be reused across different applications. Therefore, embedding high flexibility can reduce the area waste for unused FPUs.

6.2.3 Optimisation by Merging FPUs

Inspired by the results of the system-level optimisation, fewer number of FPU types and smaller FPUs may lead to better placement for shorter delay. It is because different types of FPU cannot be swapped, the placement is not flexible. We try to reduce the number of FPU types by merging different FPUs into a larger FPU to increase placement flexibility. Figure 6.2 shows an example of merging *graph15* and *graph26* into a larger *graph15 + graph26*. We simply merge them into a single FPU without any optimisation, where only clock signal is shared. This merging scheme may have two possible outcomes:

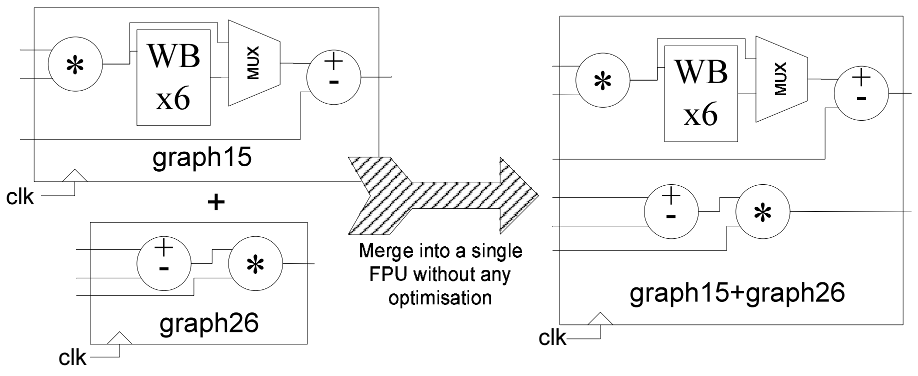


Figure 6.2: An example of merging *graph15* and *graph26* into a larger FPU

1. Better placement to reduce wirelength

In system-level optimisation, there are many types of FPUs to choose from in a hybrid FPGA. The position of various FPUs are fixed, and only the same FPU type can be swapped during the placement stage. This leads to inflexible placement and introduces long wire between FPUs as shown in Figure 6.3(a). Merging different FPU types into a larger FPU can achieve better placement and reduce the wirelength. Figure 6.3(b) shows an example of merging FPU1 and FPU3 to reduce the length of wire. FPU1_3 can be swapped to optimise the connections such as $Net1$ and $Net3$. $Net2$ becomes a short self-connected wire. Therefore, the delay of the circuit can be reduced in this case.

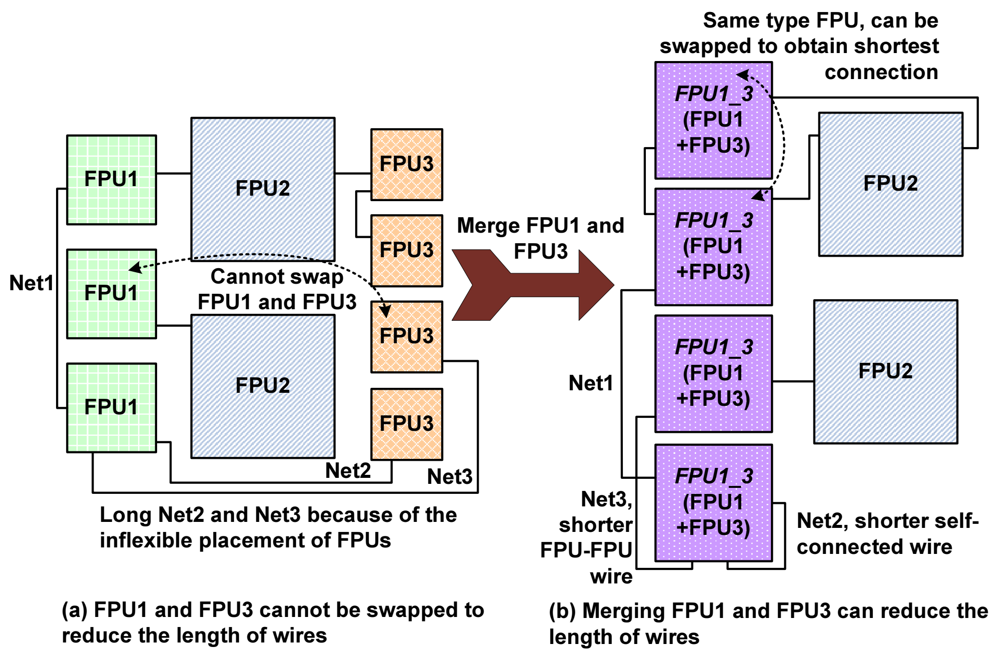

Same type FPU, can be swapped to obtain shortest connection
FPU1
FPU2
FPU3
FPU3
FPU1
Cannot swap FPU1 and FPU3
FPU3
Net1
FPU1
FPU2
FPU3
Net2
Net3
Long Net2 and Net3 because of the inflexible placement of FPUs
Merge FPU1 and FPU3
FPU1_3 (FPU1 +FPU3)
FPU2
Net1
Net3, shorter FPU-FPU wire
Net2, shorter self-connected wire
(a) FPU1 and FPU3 cannot be swapped to reduce the length of wires
(b) Merging FPU1 and FPU3 can reduce the length of wires

Figure 6.3: Merging different types of FPUs can obtain better placement and reduce wirelength

2. Larger FPU increases wirelength

However, a larger merged FPU can also increase the length of wire connection. Figure 6.4 shows an example of increasing wirelength. The original $Net1$ between CLB and FPU1 and $Net2$ between FPU1 and FPU2 are short (Figure 6.4(a)). In Figure 6.4(b), the width and height of the merged FPU1_2 is longer, which leads to longer nets and delay.

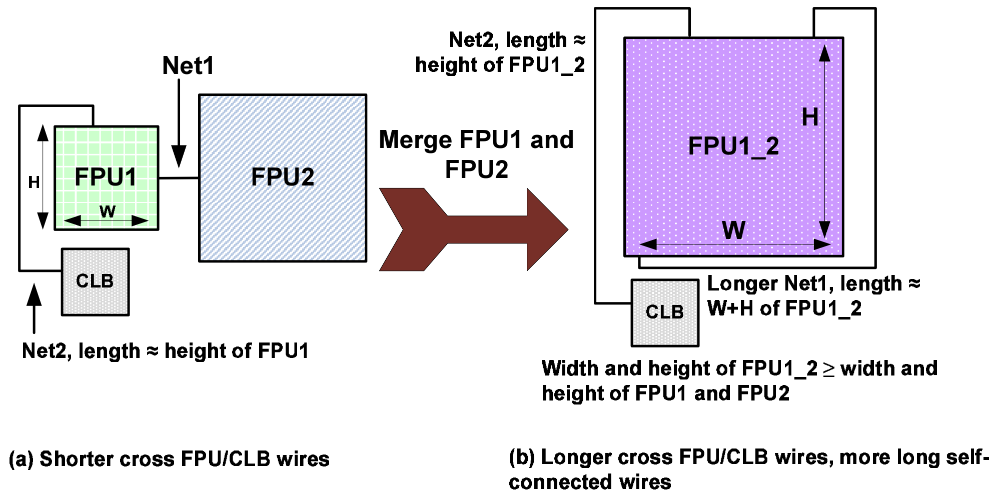

Net1
H
FPU1
W
FPU2
CLB
Net2, length ≈ height of FPU1
Merge FPU1 and FPU2
Net2, length ≈ height of FPU1_2
FPU1_2
H
W
CLB
Longer Net1, length ≈ W+H of FPU1_2
Width and height of FPU1_2 ≥ width and height of FPU1 and FPU2
(a) Shorter cross FPU/CLB wires
(b) Longer cross FPU/CLB wires, more long self-connected wires

Figure 6.4: Increase wirelength when merging FPUs

Since there are advantages and disadvantages of the proposed optimisation by merging FPUs, it is interesting to study the trade-offs of wirelength, area and speed in this optimisation scheme.

6.3 Methodology

We employ an empirical methodology to examine the speed and area of different coarse-grained FPUs. This section describes a common subgraph extraction methodology for floating point applications, and the tools, benchmarks and models that are used.

6.3.1 Common Subgraph Extraction

Floating point applications have common characteristics for floating point computations. A common subgraph in these applications represents functionality shared across the benchmark circuits. The subgraph can potentially be implemented as a hard EB to speed up the computation. Efficiency can usually be improved by combining similar FP operations into the same core, by common subgraph extraction [57]. We enhance this method for floating point application which supports FA, FM and WB extraction.

Figure 6.5 is an example of a common subgraph of two circuits. Figure 6.5(a) and (b) are part of *dscg* and *bfly* respectively. The common floating point operations can be extracted as a single unit as shown in Figure 6.5(c). The main differences between original common subgraph extraction [57] and enhanced version are:

- finding common floating point arithmetic components (floating point adder/subtractor and multiplier) instead of fixed point adder/subtractor and multiplier,
- identifying not only FF between two arithmetic components. We also identify binary operations such as AND, OR, XOR for WB. The binary operations and FF are consisted in a WB.

In the tool flow of common subgraph extraction as shown in Figure 6.6, floating point benchmark circuits are written in Verilog. Icarus Verilog [141] and ODIN [77] are used to parse and flatten the

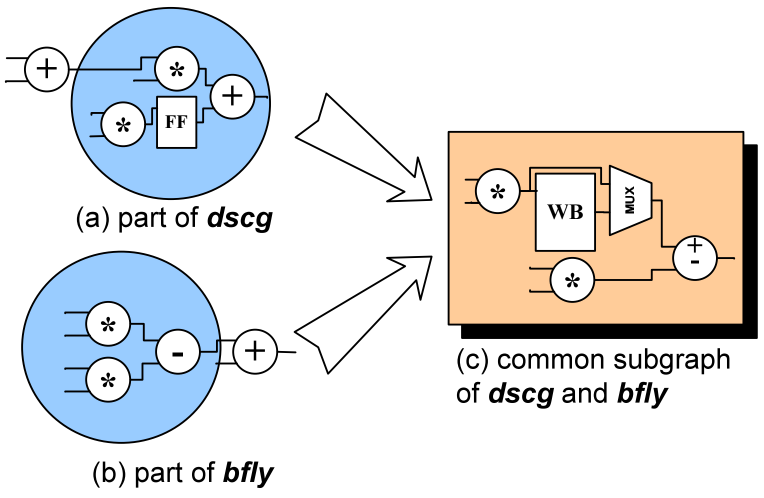


Figure 6.5: Common subgraph extraction for WB, FA and FM in FP applications

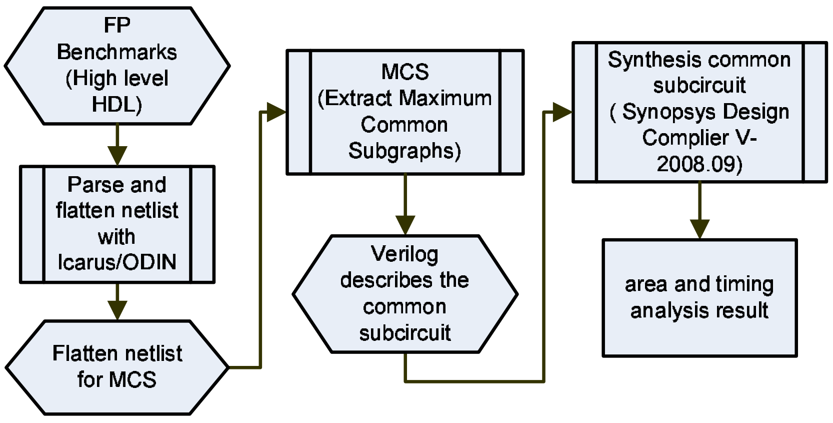

FP Benchmarks (High level HDL)
Parse and flatten netlist with Icarus/ODIN
Flatten netlist for MCS
MCS (Extract Maximum Common Subgraphs)
Verilog describes the common subcircuit
Synthesis common subcircuit (Synopsys Design Complier V-2008.09)
area and timing analysis result

Figure 6.6: Common subgraph extraction design flow

Verilog benchmark circuits. The flattened netlist is then fed into the program Maximum Common Subgraph (MCS) generation stage to extract the common subgraphs in these benchmark circuits.

The common subgraph extract algorithms in MCS are shown in Figure 6.7. Algorithm 1 identifies the common subgraph by traversing both graphs (graph of the benchmarks) to find two similar nodes (type T={FA or FM}). The new node is created as a seed node to grow the common subgraph structure. The graphs traversal add nodes to this common subgraph recursively. The *ISMATCH* function determines whether the visiting node satisfies our problem constraint to add to the subgraph. The constraint is whether any combination of components connecting to and from this visiting node is a combination of FA, FM or WB. Algorithm 2 (FAST_MATCH) describes the recursive algorithm for adding nodes to the common subgraph. All the ports of the node should be examined to match all the combination of inputs. The functions r_1(u') and r_2(u') return the nodes from benchmark netlists G_1 and G_2 respectively which have been mapped onto common subgraph node u'. $f^{-1}(u_1)$ returns the node of u_1 in G'. The algorithm terminates when all combinations of the seed nodes from the graphs have been visited.

The common subgraphs cover FP operations such as the example shown in Figure 6.5(c). With the connection information of WBs, FAs and FMs, we describe the coarse-grained FPU of common subcircuit in another Verilog file. The FPU, which consists of complex FA and FM circuits, is then synthesised by Synopsys Design Complier V-2008.09 with 0.13μm process. We obtain the area and delay of the this FPU, and use this information to evaluate the FPGA by VPH.

6.3.2 Evaluation Flow

After we have determined the FP coarse-grained blocks by common subgraph extraction, such blocks are interfaced to the fine-grained FPGA. We use the VPH work flow described in Chapter 3, Section 3.3.2 to explore this novel hybrid FPGA architecture. We use the common subgraph circuits as EBs in the high level HDL. We employ the eight benchmarks described in Chapter 2, Section 2.5.3 and the optimised routing architecture in Chapter 5 to evaluate the FPU generated using common subgraph extraction.

```
Input: $G_1$, $G_2$
1: $G'$ = empty
2: for all $v_i$ in $G_1$ do
3:     for all $v_j$ in $G_2$ do
4:         if $T(v_i) = T(v_j)$ then
5:             if ISMATCH($v_i$, $v_j$) then
6:                 add nodes to $G'$
7:                 FAST MATCH($G_1$, $G_2$, $G'$)
8:             end if
9:         end if
10:    end for
11: end for
```

Algorithm 1 : Common subgraph extraction top-level algorithm

```
Input: $G_1$, $G_2$, $G'$
1: for all nodes $u'$ in $G'$ do
2:     for all ports $i$ on node $r_1(u')$ do
3:         $u_1$ = node on port $i$ of $r_1(u')$
4:         for all ports $j$ on node $r_2(u')$ do
5:             $u_2$ = node on port $j$ of $r_2(u')$
6:             if ISMATCH($u_1$, $u_2$) then
7:                 add node to $G'$ on appropriate port of $u'$
8:                 FAST MATCH($G_1$, $G_2$, $G'$)
9:                 remove $f^{-1}(u_1)$ and its connections from $G'$
10:            end if
11:        end for
12:    end for
13: end for
```

Algorithm 2 FAST MATCH: Recursive algorithm to add nodes to a common subgraph

Figure 6.7: Common subgraph extraction algorithm in MCS

6.4 Results

In this section, the internal and system-level optimisation of coarse-grained floating point units are studied. The common subgraphs of the floating point benchmark circuits are examined to optimise the architecture of the FPU. After that we explore the area and delay impact of making the common subgraphs coarse-grained units in the hybrid FPGA. Finally, we further optimise the systems by merging different FPUs into a larger FPU. The default architecture parameters of the experiments conducted are: (1) additional 20% of minimum channel width to avoid congestion, (2) baseline architecture described in Chapter 3, Section 3.2, (3) optimised routing architecture - square EBs positioned in the center of FPGA, 4 sides I/O pin configuration of EB, switches in EB and D_{eb}=1 to facilitate routing flexibility which can achieve better area-delay product.

6.4.1 Internal Optimisation of FPU

We determine the common subgraphs of floating point arithmetic in the benchmark circuits. The hard FPU has a specific ordering of WBs, FAs and FMs so that it can be reused by different benchmark circuits. The common subgraphs are shown in Table 6.1, which are found by common subgraph extraction described in Section 6.3. Common subgraph extraction can enumerates all possible common subgraphs, but we only include subgraphs with two or more nodes. The hard FPU is obtained by 0.13μm process, however the CLB feature size modelled in VPH is 0.15μm. Therefore, the normalised area (Area/Feature size squared) is used to obtain the equivalent area in terms of CLBs. The performance of the various common subcircuits is examined.

Table 6.2 shows the occurrence in benchmarks, normalised area, delay, number of input/output and latency of each common subgraph found. It is obvious that more FAs and FMs embedded in an FPU can achieve higher area reduction since all the elements are compacted into a single unit. The average area of the FPUs is 1.9% less than that without clustering of WBs, FAs and FMs. But the delay after grouping is 5% more than pure FAs and FMs. The difference in the delay is because the timing report of single FA or FM does not account for the output to input delay when connecting them together. Therefore we should investigate the performance of the system rather than the single unit.

Table 6.1: The common subgraph structure occurred in benchmark circuits

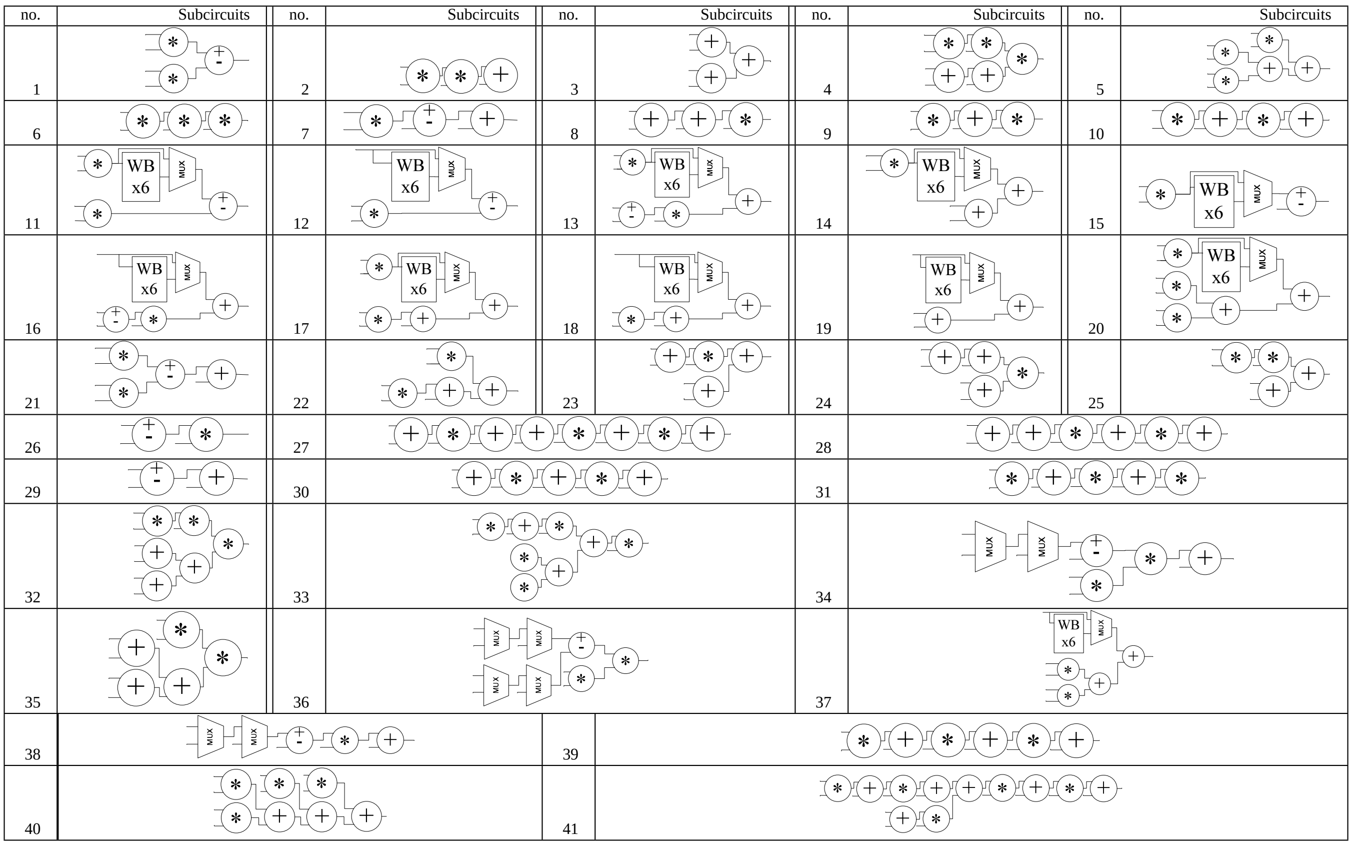


Table 6.2: Statistic of subgraphs.
(The feature size (L) of the coarse-grained units is 0.13μm. Virtex II CLB: area is 10,912μm^2, feature size is 0.15μm, normalised area is 485,013μm^2)

Graph	Occurrence in benchmarks								area(A)	normalised	area in	delay	no.	no.	latency
no.	dscg	bfly	fir	ode	mm3	bgm	syn2	syn7	(μm^2)	area(A/L^2)	CLB	(ns)	input	output	(cycles)
WB	No separate WB embedded								37,135	2,197,321	7.7	0.76	104	38	1
FA	4	4	3	3	2	9	5	25	75,681	4,478,179	15.7	2.88	67	39	6
FM	4	4	4	2	3	11	4	25	214,348	12,683,315	44.5	2.95	67	39	6
1	0	8	4	0	4	2	0	4	451,637	26,724,083	93.7	2.54	133	53	12
2	0	0	0	4	0	0	0	8	472,909	27,982,762	98.1	3.02	133	53	18
3	0	0	0	0	0	0	1	3	197,894	11,709,705	41.0	2.56	133	53	12
4	0	0	0	0	0	0	1	4	761,720	45,072,171	158.0	2.93	199	67	18
5	0	0	4	0	0	2	0	2	706,994	41,833,960	146.6	2.56	199	67	18
6	0	0	0	0	0	0	1	6	630,374	37,300,261	130.7	2.91	133	53	18
7	0	8	4	0	4	2	0	8	320,451	18,961,579	66.5	2.74	133	53	18
8	0	0	0	0	0	1	1	8	143,433	8,487,139	29.7	2.57	133	53	18
9	0	0	0	0	0	4	0	7	475,222	28,119,673	98.6	3.18	133	53	18
10	0	0	0	0	0	3	0	5	534,648	31,636,000	110.9	3.04	166	60	24
11	8	8	4	0	4	2	0	4	747,237	44,215,226	155.0	3.17	160	84	18
12	8	8	12	4	4	7	0	16	547,774	32,412,679	113.6	3.17	127	77	18
13	8	0	0	0	0	1	0	4	830,166	49,122,235	172.2	3.17	193	91	18
14	0	0	4	0	4	2	0	6	631,444	37,363,552	131.0	3.17	160	84	18
15	8	8	12	4	8	9	0	16	548,265	32,441,718	113.7	3.17	127	77	18
16	8	0	0	4	0	3	0	8	631,508	37,367,336	131.0	3.17	160	84	18
17	0	0	4	0	4	2	0	5	823,209	48,710,582	170.7	3.09	193	91	18
18	0	4	4	0	4	2	0	8	630,715	37,320,404	130.8	3.17	160	84	18
19	0	4	4	0	4	2	1	8	431,948	25,559,063	89.6	3.17	127	77	12
20	0	0	4	0	4	2	0	2	1,021,944	60,470,082	212.0	3.09	226	98	18
21	0	8	4	0	4	2	0	2	563,734	33,357,046	116.9	2.95	166	60	18
22	0	0	4	0	0	2	0	5	563,800	33,360,932	116.9	2.95	166	60	18
23	0	0	0	0	0	1	0	5	450,878	26,679,195	93.5	2.95	166	60	18
24	0	0	0	0	0	0	1	5	448,952	26,565,188	93.1	2.95	166	60	18
25	0	0	0	0	0	0	1	6	563,636	33,351,218	116.9	2.95	166	60	18
26	8	0	0	4	0	5	1	15	282,039	16,688,697	58.5	2.95	100	46	12
27	0	0	0	0	0	1	0	1	1,011,722	59,865,180	209.8	2.95	298	88	48
28	0	0	0	0	0	1	0	3	735,905	43,544,680	152.6	2.95	232	74	36
29	0	8	4	0	4	2	1	8	166,059	9,825,981	34.4	2.95	100	46	12
30	0	0	0	0	0	1	0	4	646,918	38,279,188	134.2	2.95	199	67	30
31	0	0	0	0	0	1	0	4	762,722	45,131,475	158.2	2.95	199	67	30
32	0	0	0	0	0	0	1	2	847,988	50,176,826	175.9	2.95	232	74	18
33	0	0	0	0	0	1	0	1	1,243,104	73,556,461	257.8	2.95	298	88	30
34	0	0	0	4	0	0	0	4	566,409	33,515,327	117.5	3.75	232	60	18
35	0	0	0	0	0	0	1	3	649,054	38,405,567	134.6	2.95	199	67	18
36	0	0	0	4	0	0	1	8	486,112	28,764,043	100.8	3.6	265	53	12
37	0	4	4	0	4	2	0	2	829,449	49,079,801	172.0	3.17	193	91	18
38	8	0	0	4	0	3	0	8	366,637	21,694,478	76.0	3.75	199	53	18
39	0	0	0	0	0	1	0	2	851,733	50,398,398	176.6	2.95	232	74	36
40	0	0	4	0	0	0	0	2	1,044,227	61,788,577	216.6	2.95	265	81	24
41	0	0	0	0	0	1	0	1	1,505,000	89,053,247	312.1	2.95	397	109	54

6.4.2 System-Level Optimisation

After we determine the common subcircuits, we evaluate the impact of these subcircuits to the systems. Based on the optimisation parameter in Section 6.2.2. The delay, area and routing resources of purely FA/FM system, and mixture of subcircuits are examined. In the purely FA/FM system, 25 x

Table 6.3: The utilisation rate of subcircuits in the three hybrid FPGAs

Hybrid FPGA	**1. Purely FA/FM FPGA**		**2. FPGA_12_15_26**					**3. FPGA_41_20_37_12_26**						
Graph no.	**FA**	**FM**	**12**	**15**	**26**	**FA**	**FM**	**41**	**20**	**37**	**12**	**26**	**FA**	**FM**
No. of subcircuits in FPGA	25	25	16	4	2	8	8	1	4	4	8	3	8	8
Benchmark	**Utilisation rate (%)**													
dscg	64	64	50	0	0	100	100	0	0	0	100	0	100	100
bfly	64	64	50	0	0	100	100	0	0	100	50	0	50	50
fir	48	64	75	0	0	0	50	0	100	0	50	0	0	0
ode	48	32	25	0	0	100	50	0	0	0	50	0	100	50
mm3	32	48	25	100	0	0	12.5	0	100	0	0	0	0	0
bgm	36	44	43.75	50	0	0	25	100	25	0	12.5	0	0	25
syn2	20	16	0	0	50	50	37.5	0	0	0	0	33.33	50	37.5
syn7	100	100	100	0	100	87.5	87.5	100	0	50	100	100	50	62.5

FA and 25 x **FM** are used in the hybrid FPGA.

1. Selection of Systems

We select two systems based on density and flexibility of FPU to compare to purely FA/FM system.

(a) Density of FPU: We select a set of common subcircuits with more FAs, FMs and WBs. For example, *graph41* has 11 nodes that are a combination of FAs and FMs, contains most computation elements among subgraphs in Table 6.2. Since the density and area reduction of this individual subcircuit is greatest compared to separate FA, FM and WB, this set of subcircuits can be the best choice to reduce the area of the hybrid FPGA. The selection scheme is shown in Figure 6.8 (a).

We choose 7 types of FPUs: ***graph41***, ***graph20***, ***graph37***, ***graph12***, ***graph26***, ***FM*** and ***FA*** as subcircuits to embed in the hybrid FPGA (*FPGA_41_20_37_12_26*).

(b) Flexibility of FPU: If we can reuse all the subcircuits for all applications, the area of the hybrid FPGA may be reduced. We select a set of subcircuits which have highest occurrence rate in the benchmark circuits from Table 6.2, based on the flow in Figure 6.8 (b). *graph12* has the highest occurrence rate (16 times) among all the subgraphs.

We finally choose 5 types of FPUs: ***graph12***, ***graph15***, ***graph26***, ***FM*** and ***FA*** to embed in the hybrid FPGA (*FPGA_12_15_26*).

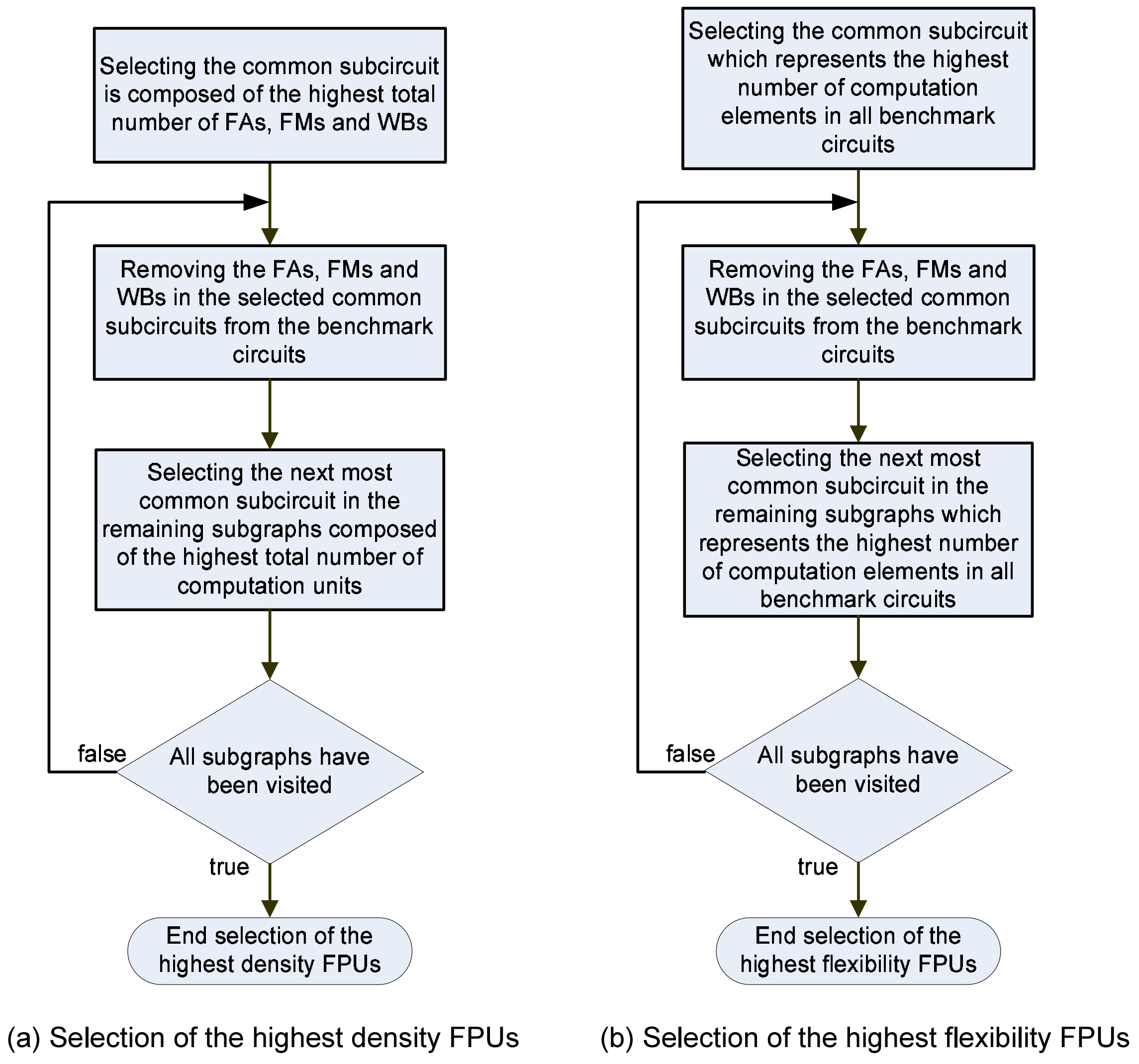

Selecting the common subcircuit is composed of the highest total number of FAs, FMs and WBs
Removing the FAs, FMs and WBs in the selected common subcircuits from the benchmark circuits
Selecting the next most common subcircuit in the remaining subgraphs composed of the highest total number of computation units
All subgraphs have been visited
false
true
End selection of the highest density FPUs
(a) Selection of the highest density FPUs
Selecting the common subcircuit which represents the highest number of computation elements in all benchmark circuits
Removing the FAs, FMs and WBs in the selected common subcircuits from the benchmark circuits
Selecting the next most common subcircuit in the remaining subgraphs which represents the highest number of computation elements in all benchmark circuits
All subgraphs have been visited
false
true
End selection of the highest flexibility FPUs
(b) Selection of the highest flexibility FPUs

Figure 6.8: The flow of the selection of (a) the highest density FPUs and (b) the highest flexibility FPUs in hybrid FPGAs

Discussion

From the three hybrid FPGAs we selected : (a) Purely FA/FM FPGA, (b) *FPGA_12_15_26* and (c) *FPGA_41_20_37_12_26*, we examine the area, delay and routing resources impact to the applications. The utilisation rate of the subcircuits in each hybrid FPGA for benchmark circuits is shown Table 6.3. The two selection methods are greedy, and do not consider any connections of the chosen subgraphs.

2. Fine-grained Area Impact

Figure 6.9 shows the average number of CLBs used in the eight applications with different types of graphs. If the area of FPU is not counted, *FPGA_12_15_26* reduces area by 14.4% and *FPGA_41_20_37_12_26* reduces area by 18% when compared to purely FA/FM hybrid FPGA's area in average. However, when the area of FPUs is counted in the total area, *FPGA_12_15_26* is 1.8% more than the purely FA/FM system in total area. Total area of *FPGA_41_20_37_12_26* is still 4.6% less than purely FA/FM system. The purely FA/FM system does not have wordblocks and as such it requires a lot of fine-grained CLBs as registers and logical operators. *FPGA_41_20_37_12_26* embeds high density subcircuits, which are not flexible to the benchmarks, some of the FPUs are thus wasted, as they are not used. However, the individual subcircuits obtain highest area reduction to overcome the waste of area when being used. Therefore, it is the best to reduce fine-grained area in hybrid FPGA. The density of *FPGA_12_15_26* is not high enough to overcome the waste of area, and results in higher total fine-grained area.

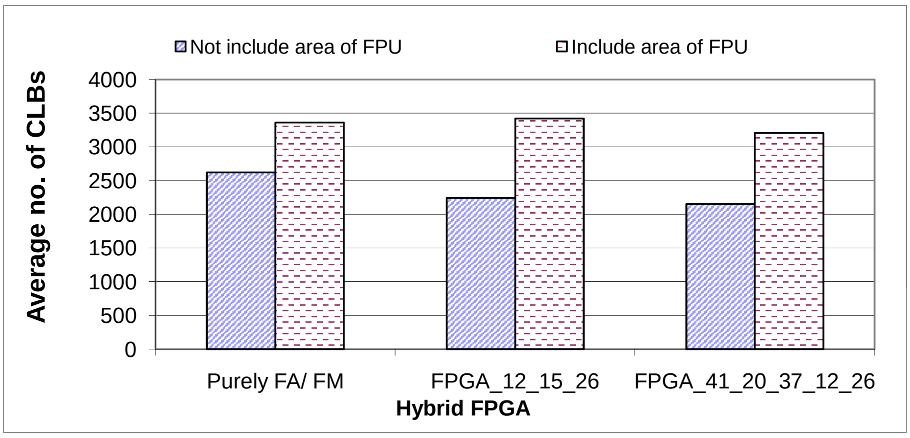


Figure 6.9: The number of CLBs used by different types of embedded FPUs

3. Delay Impact

Figure 6.10 shows purely FA/FM hybrid FPGA achieves highest speed. The delay of purely FA/FM FPGA is 20.1% and 23% less than *FPGA_12_15_26* and *FPGA_41_20_37_12_26* respectively. We discover that embedding more coarse-grained FPUs types causes a decrease in speed. Besides the different between the delay of individual FPUs. The critical path is dominated by the connection between two FPUs. This path can only be optimised by moving the FPUs close together. Since the various FPUs have different architectures, they cannot be swapped to get better placement. For example, we cannot swap *graph41* and *graph20*, but two *graph12* can be swapped. The purely FA/FM system has least types of subcircuits and the subcircuits are small. Therefore, due to the placement constraints of the different FPUs type, FPUs in a purely FA/FM system have more freedom to move around to achieve higher speed. This can be reflected by the wirelength, where the wirelength of purely FA/FM is 6.7% shorter than *FPGA_12_15_26* and 21% shorter than *FPGA_41_20_37_12_26*.

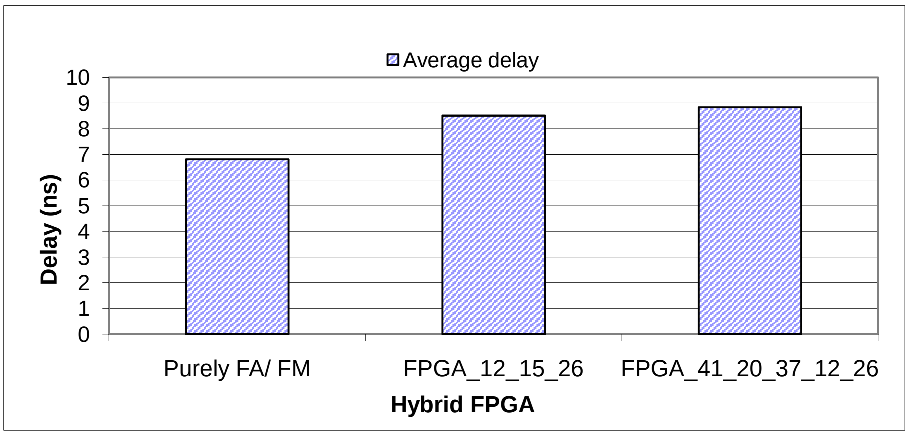


Figure 6.10: The delay of benchmark circuits by using different types of FPUs

4. Routing Resource Impact

On average, the total routing area of *FPGA_41_20_37_12_26* and *FPGA_12_15_26* are 27.9% and 23.5% less than the purely FA/FM FPGA as shown in Figure 6.11. We employ the optimised routing architecture, where switches are in EB and EBs are separated to minimise the congestion at the edge of EB. The congestion of all FPGAs should be similar. However, in *FPGA_41_20_37_12_26* and *FPGA_12_15_26*, most of the connections are in the FPUs, therefore they must use less routing resources for connection. *FPGA_41_20_37_12_26* is the most compact and consumes less total CLBs (including the area of FPUs). Therefore, it uses less total routing area.

From the above result, different mixtures of coarse-grained subcircuits can achieve different aspects of optimisation in hybrid FPGAs. As a result, we could use a suitable set of subcircuits to obtain a particular optimisation goal.

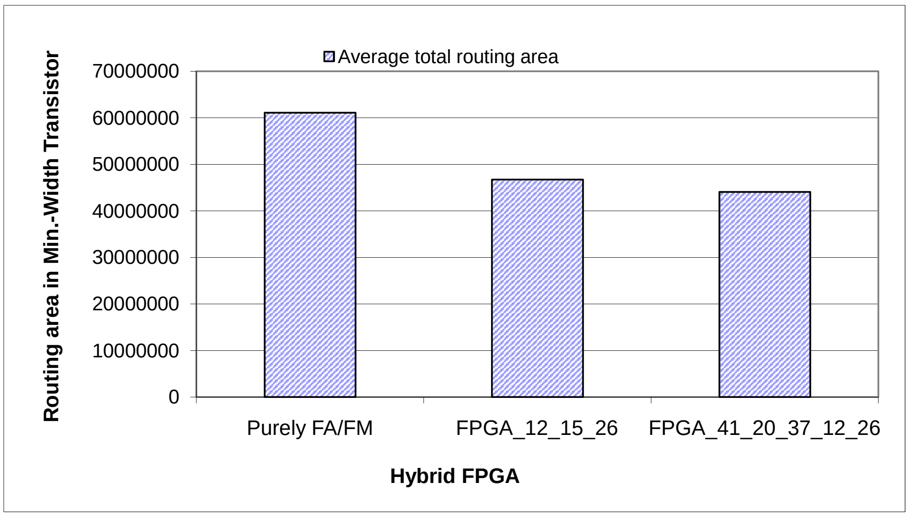


Figure 6.11: The average total routing area used different types of FPUs

5. Area-Delay Product Impact

After we examined the individual system's speed and area. We study the overall area-delay product of the systems in Figure 6.12. *FPGA_41_20_37_12_26* achieves the best area-delay product in the three systems. It is 2.2% and 6.4% better than *FPGA_12_15_26* and purely FA/FM FPGA respectively. *FPGA_41_20_37_12_26* is slower than the other systems, it consumes less routing resources. Overall, *FPGA_12_15_26* is the best for both speed and area.

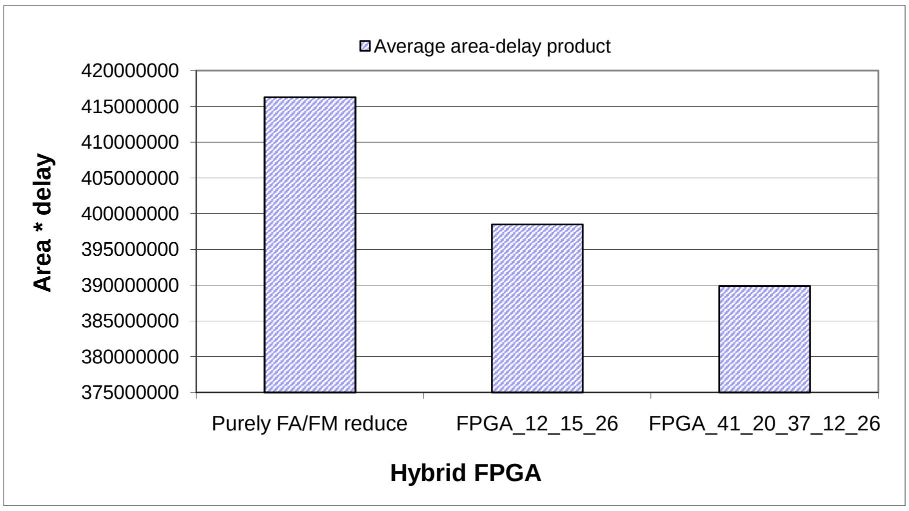


Figure 6.12: The area-delay product of different types of hybrid FPGA

6.4.3 Optimisation by merging FPUs

The various types of FPUs may cause ineffective placement and lead to longer wire as discussed in Section 6.2.3 and the result from Section 6.4.2. From the result of the last section, FA/FM FPGA is the fastest because the effective placement of FPUs and small FPUs. We propose merging different types of FPUs together to improve the placement flexibility (Figure 6.3). The merged FPU is larger and may result in longer net delay (Figure 6.4). Therefore, we examine the impact of the wirelength, area and delay of the three FPGAs obtained in Section 6.4.2 by merging their FPUs. The best merged FPUs scheme will be determined in this section.

1. Merging Scheme

We merge the FPUs with similar number of subcircuits in the FPGAs stated in Table 6.3 which minimises the waste of unused FPUs. For example, in *FPGA_41_20_37_12_26*, *graph37* occurs 4 times and *graph26* occurs 3 times. We merge them into a single FPU *graph26+graph37*. We apply different merge scheme: A, B, C, D and E, each of them has various combination of merged FPUs which is shown in Table 6.4. FPGA in scheme A contains smaller FPUs while FPGA in scheme E composes of larger merged FPUs. Table 6.5 shows the area, delay and the number of FPUs embedded in FPGAs. We examine the impact on area, delay and wirelength of the merged FPUs in the hybrid FPGAs based on the these FPU results.

Table 6.4: The different FPUs merged in the three FPGAs

Merge scheme	**Pure FA/FM FPGA**	**FPGA_12_15_26**	**FPGA_41_20_37_12_26**
A	(1)FA (2)FM	(1)graph12 (2)graph15 (3)graph26 (4)FA (5)FM	(1)graph12 (2)graph20 (3)graph26 (4)graph37 (5)graph41 (6)FA (7)FM
B	(1)FA+FM	(1)graph12 (2)graph15 (3)graph26 (4)FA+FM	(1)graph12 (2)graph20 (3)graph26 (4)graph37 (5)graph41 (6)FA+FM
C	(1)5*FA+5*FM	(1)graph12 (2)graph15+graph26 (3)FA+FM	(1)graph12 (2)graph20 (3)graph26+graph37 (4)graph41 (5)FA+FM
D	(1)7*FA+7*FM	(1)graph12+graph15+graph26+FA+FM	(1)graph12+graph20+graph26+graph37+graph41+FA+FM
E	(1)10*FA+10*FM	(1)4*graph12+graph15+graph26+2*FA+2*FM	(1)2*graph12+graph20+graph26+graph37+graph41+2*FA+2*FM

Table 6.5: The statistic of the merged FPUs in the three FPGAs

System: Purely FA/FM FPGA

Merged FPU type	Area in CLB	Delay (ns)	No. FPU	I/O density
FA+FM	56	2.95	25	13.43
5*FA+5*FM	289	2.95	5	29.46
7*FA+7*FM	400	2.95	4	35.04
10*FA+10*FM	576	2.95	3	41.70

System: FPGA_12_15_26

Merged FPU type	Area in CLB	Delay (ns)	No. FPU	I/O density
FA+FM	56	2.95	8	13.43
graph15+graph26	169	3.17	4	11.62
graph12+graph15+graph26+FA+FM	342	3.19	16	18.014
4*graph12+graph15+graph26+2*FA+2*FM	756	3.19	4	24.73

System: FPGA_41_20_37_12_26

Merged FPU type	Area in CLB	Delay (ns)	No. FPU	I/O density
FA+FM	56	2.95	8	13.43
graph26+graph37	225	3.17	4	12.47
graph12+graph20+graph41+graph37+graph41+FA+FM	930	3.19	8	24.11
2*graph12+graph20+graph26+graph37+graph41+2*FA+2*FM	1406	3.19	4	32.01

2. Delay and Wirelength Impact

Figure 6.13 shows there is a maximum delay reduction in the five merge schemes. The delay is the average place and route result of the eight benchmarks. Merge scheme A is the original FPGA without any merged FPUs. In scheme B, purely FA/FM hybrid FPGA reduces 3.1% and *FPGA_41_20_37_12_26* reduces 3.5% in delay, while *FPGA_12_15_26* achieves slightly amount of 1.6% delay reduction. Further increases in the size of merged FPUs increases the delay. The increment is due to the change of width (W) and height (H) of the merged FPUs. Table 6.6 shows the average wirelength and the maximum $W+H$ of the FPUs and merged FPUs. The wirelength is generally increasing while the FPUs are getting larger. When the $W+H$ of the merged FPUs are shorter than average wirelength, the speed can be improved. It is because of the both cross FPUs and self-connected wires are short as described in Figure 6.3. Once the $W+H$ of the merged FPUs exceed the average wirelength, the cross FPUs wire are short, but the self-connected wires are long as shown in Figure 6.4. The long self-connected wire becomes the critical path which leads to decrease in speed.

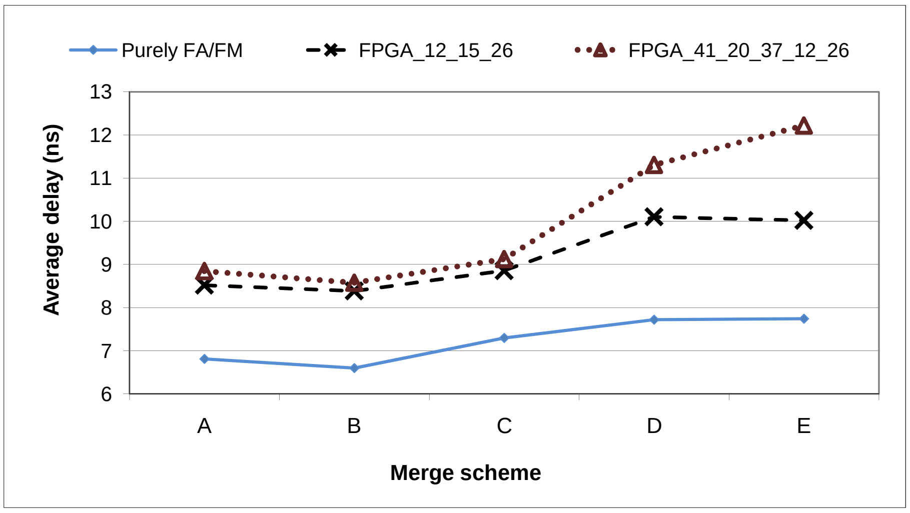


Figure 6.13: The delay in the three hybrid FPGAs by using different FPU merging methods

Table 6.6: The average wirelength of the three FPGAs in different merge schemes

	Purely FA/FM FPGA			FPGA_12_15_26			FPGA_41_20_37_12_26		
Merge Scheme	Average wirelength	Max. W+H of all FPUs	Max. W+H of merged FPUs	Average wirelength	Max. W+H of all FPUs	Max. W+H of merged FPUs	Average wirelength	Max. W+H of all FPUs	Max. W+H of merged FPUs
A	24.08	13	–	25.80	21	–	30.48	35	–
B	23.46	15	15	29.11	21	15	32.19	35	15
C	26.87	34	34	28.97	26	26	28.97	35	30
D	28.63	40	40	35.75	37	37	43.55	61	61
E	28.96	48	48	37.16	55	55	43.55	75	75

3. Area Impact

We examine the total routing area used in each merge scheme, the result is shown in Figure 6.14. There are at least 6.1%, 11.1% and 5.2% increment of routing area when merging FPUs into a larger FPU in purely FA/FM FPGA, *FPGA_12_15_26* and *FPGA_41_20_37_12_26* respectively. The larger and more compact FPU has higher I/O density as shown in Table 6.5. Originally, FA and FM have 7.7 and 12.6 I/O pins per CLB length respectively. The I/O density of merged FA+FM FPU is 13.43, which is 74% more than FA and 6.6% more than FM. In Chapter 5, we have showed that the higher I/O density FPU requires more routing resources. Therefore, the large FPUs in scheme B, C, D and E cause larger routing area.

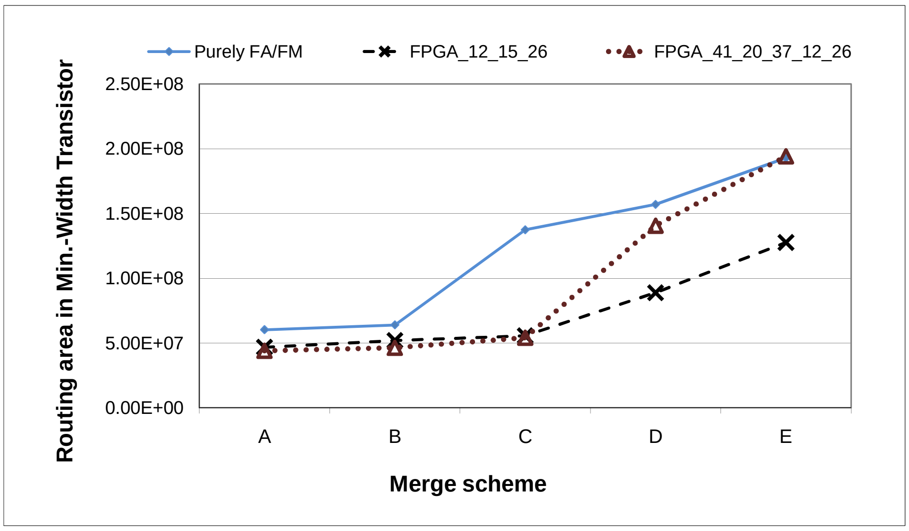


Figure 6.14: The routing area in the 3 hybrid FPGAs by using different FPU merging methods

4. Area-Delay Product Impact

Finally, we discuss the area-delay product to identify which scheme is the best for the overall performance. Figure 6.15 shows that the area-delay product of scheme A and B are similar in all the three systems. Scheme B has advantage in speed which can compensate the lost in routing area. Scheme A is opposite to scheme B, which has better area but slower. Therefore, they achieve similar performance. Scheme C, D and E include more compact FPU, the area gained in individual FPU cannot compensate the lost in speed and routing area as discussed before. As a result, the larger merged FPUs cause worse area-delay product.

From the above result, we could use a suitable merge scheme to obtain a particular optimisation goal.

6.5 Summary

This chapter illustrates the adoption of common subgraph extraction to determine optimised floating point coarse-grained blocks in hybrid FPGAs. Floating point circuits are often not efficiently implemented in fine-grained FPGA technology. This chapter has shown the impact of embedding different and multiple types of coarse-grained blocks on a floating point hybrid FPGA. We have found that, (a) the speed of the system is the highest for implementations involving only FAs and FMs, (b) higher

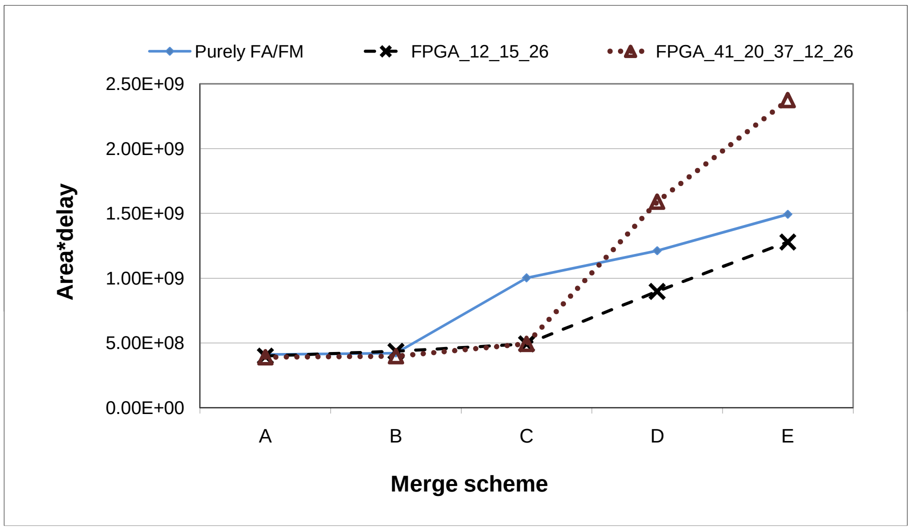


Figure 6.15: The area-delay product in the 3 hybrid FPGAs by using different FPU merging methods

density subgraphs produce greater reduction on the area of the system, and (c) they provide the best area-delay product, (d) merging of FPUs can improve the speed of the hybrid FPGAs, but results in lost of area. We can optimise specific parameters such as area, delay and balanced speed/area for the hybrid FPGA based on the above results as shown in Table 6.7.

Table 6.7: Recommendations of FPUs for different optimisation goals

Goal	Recommended optimisation of FPU
Speed	An FPU contains 1 FA and 1 FM
Area	Different types of FPUs which are the highest density common subgraphs
Balanced speed and area	Different types of FPUs which are the highest density common subgraphs

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The research in this thesis has contributed to domain-specific hybrid FPGA architecture. We propose a novel floating point hybrid FPGA architecture to optimise floating point computations for applications in science, medicine and finance. A floating point embedded block is included in the architecture. We develop an evaluation tool called VPH for the proposed domain-specific hybrid FPGA architecture. This tool enables evaluation of hybrid FPGAs without the limitation of physical constraints such as area of the chip, I/O pin position and position of the embedded blocks. The proposed architecture and VPH facilitate the study of interface between coarse-grained and fine-grained logics, routing architecture optimisation, and internal optimisation of embedded floating point units.

There are three main achievements. First, we explore the area, speed and routing resource impact of interface parameters: EB position, pin location, interconnect flexibility and EB aspect ratio for FPGAs with single and multiple EB types. Second, we study the routing demand of the large EBs and evaluate the effect of three routing optimisations to meet the extra routing requirement. Finally, we optimise the floating point embedded block by using common subgraph extraction and examine the area, speed and routing resource of the systems containing different optimised FPUs. The key results of the research are summarised in Table 7.1.

The results are important for hybrid FPGA architecture design. Various interface parameters and optimisation methods have different impact on speed, area and routing resources of hybrid FPGAs. We could use a suitable set of parameters to cover a particular optimisation goal. The goal can be optimising the FPGA architecture for either speed, area or a combination of speed and area.

7.2 Future Work

The limitations of the work presented in this thesis are summarised in Table 7.2. There could be improvements and further research based on the CAD tools and the optimisation architecture proposed in this thesis.

7.2.1 Other FPU Architectures

The proposed FPU in Chapter 3 adopts a reconfigurable bus-based architecture as embedded block in hybrid FPGAs to reduce area and delay. The FA and FM inside an FPU are fixed, so are not flexible. We can study more flexible FPUs in future work. For example, Chong and Parameswaran [55] have increased flexibility of an embedded FPU by providing one double-precision operation or two single-precision operations in parallel. This multi-mode embedded FPU improves 5.2 times in area and 5.8 times in delay over a set of benchmarks.

7.2.2 CAD Tools

VPH requires large memory to store the timing nodes for place and route of floating point benchmarks. It is because there are a lot of I/Os in an FPU and they require many routing tracks. Optimisation of memory usage should be done. Memory footprint reduction [81] for VPR can be employed for VPH.

The computation time of routing for coarse-grained blocks without internal switches is high. It is because changing of routing direction inside the blocks is not possible. The VPR placement and routing algorithms are both geometry-based which are not suitable for non-rectangular cores [80].

Table 7.1: Summary of the key results

Research Area	Key results
Architecture and modelling (Chapter 3)	• The proposed architecture can achieve 4 times improvement in speed and 17 times reduction in area on average when compared with traditional FPGA devices on the selected floating point benchmark circuits. • The hybrid FPGA evaluation tool VPH supports modern FPGA architecture. Flexible position and routing architecture for embedded blocks are enabled in the tool.
Interface between coarse/fine-grained elements (Chapter 4)	• EB position: They should be positioned (a) close near the center of the FPGA for better speed and (b) evenly in FPGA for better area. • EB pin: Their I/O pins should be arranged around all four sides of the FPU. • Channel width: the addition of tracks to minimum channel width has little impact on circuit speed. • EB aspect ratio: FPUs should have a square aspect ratio for speed. • Smaller embedded memory should be located between the large EBs in highly congested FPGA to reduce the consumption of routing resources and gain speed.
Routing optimisation (Chapter 5)	• The routing nets are mainly condensed near the edge of EBs, which occupy 44% of tracks usage. • Both the separation of EBs and adding switches on EBs systems perform about 48.4% better than existing column based architecture in term of area and delay efficiency. • Extra wires near the edge of EBs would improve the area of highly congested system by 34.9%, but cannot further improve the system optimised by separating EBs and adding switch on EBs. • Ways of improving the interface between coarse/fine-grained elements for optimised routing architecture are: (a) the EBs should be positioned close to each other in the middle of the chip to achieve the best area-delay product, (b) a square EB leads to the most efficient implementation (c) small embedded memory should be located in column to avoid introducing extra routing demand to the non-congested hybrid FPGA.
Optimisation of coarse-grained FPU (Chapter 6)	• Embedding more types and larger coarse-grained FPU in the system causes at most 23% increase in delay. • The routing area of the system can be reduced by 27.9% by embedding high density subgraphs as FPUs. • The high density subgraph is the best choice for achieving balanced speed and area FPGA. • Merging of FPUs can improve the speed of the hybrid FPGAs by at most 3.5%, but cannot improve the area-delay product.

Table 7.2: Summary of limitations of the works in this thesis

Research Area	Limitations
Architecture and modelling (Chapter 3)	• The proposed connections inside FPU may not be the best way to construct an FPU (address in Section 7.2.1). • VPH is slow to route large number of FPU with high density of routing channel (address in Section 7.2.2 and Section 7.2.3). • Routing algorithm is not optimised for hybrid FPGA with coarse-grained blocks which do not have switches inside (address in Section 7.2.2). • Power estimation is currently not available (address in Section 7.2.2).
Interface between coarse/fine-grained elements (Chapter 4)	• The methodology only covers FPU in the floating point FPGA (address in Section 7.2.4).
Routing optimisation (Chapter 5)	• The proposed methods are only based on the fine-grained routing architecture, which may not be effective for the hybrid FPGA (address in Section 7.2.5).
Optimisation of coarse-grained FPU (Chapter 6)	• No technology mapping tool for common subcircuits in the floating point hybrid FPGA (address in Section 7.2.2). • There is no feedback path in the common subcircuits, which likes the datapath described in Chapter 3 (address in Section 7.2.6).
Others	• Power optimisation of hybrid FPGAs is not considered (address in Section 7.2.7). • Impact of process variation on embedded blocks is not studied (address in Section 7.2.8).

In [80], the placement and routing algorithms are modified for totally unroutable embedded blocks which are different to our blocks. We can employ the algorithms in [80] to facilitate the routing time of our proposed hybrid FPGA.

Power modelling is not supported by VPH. We need to develop a power model in VPH to provide power evaluation of hybrid FPGAs. The power model can be based on the work from the University of British Columbia [85] to estimate the power consumption of hybrid FPGAs.

There is lack of synthesis and mapping tool for packing the floating point operators into FPUs and mapping the proposed FPU into a target architecture. We can employ the open source synthesis tool Odin [77] to support packing and mapping of the proposed FPU for VPH.

7.2.3 Analytical Modelling for Delay, Area and Power of Hybrid FPGA

As introduced in Chapter 2, Section 2.3.4, analytical models can be used for quickly searching the design space by estimating the effects of different architecture parameters. There are limited models for domain-specific hybrid FPGA architectures. Smith et al. [90] have modelled wirelength for heterogeneous FPGAs. Analytical models for delay, area and power of hybrid FPGAs would also be useful. As an example of using the model, we could try to find out the optimum number and combination of coarse-grained elements in a hybrid FPGA for a specific domain.

7.2.4 Other Application Domains

We only consider floating point application domain in this thesis. We can employ the interface methodology, routing and coarse-grained block optimisation to produce other domain-specific hybrid FPGAs. One possible domain is in network processing. Networking applications include packet encoding, decoding, filtering and flow monitoring. Implementing a flow monitor requires a large number of resources (about 2000 slices and 800 Block RAM) in an FPGA [142]. Embedding ASIC cores of network flow monitor in FPGAs can reduce the area consumed by the flow monitor and may even increase the maximum packet flow. This could help the development of the flow monitoring architecture to provide quality of service (QoS) and denial of service (DoS) detection.

Medical imaging is another domain which is not efficiently implemented in FPGAs. Accelerating texture analysis for prostate cancer classification in an FPGA consumes 77% of slices and 76% of BRAM in FPGA [143]. Computing texture feature from Grey Level Co-occurrence Matrix (GLCM) is very time and area expensive. GLCM is commonly used in medical imaging. Embedding GLCM ASIC cores in FPGAs can improve the area and speed for medical applications.

7.2.5 Routing for Hybrid FPGA

The routing architecture examined in this thesis is based on existing fine-grained routing architecture which may not be optimised for domain-specific FPGAs.

In [59], a coarse-grained architecture with multibit bus-based connections has been proposed. We can employ this connection method to further optimise our routing architecture. In addition, we can explore the trade-offs of different switching methods for hybrid FPGAs rather than the fine-grained switch box. An example is crossbar interconnection [144], which provides higher bandwidth and simpler design, but it is less area-efficient. We can study the speed and area trade-offs of the crossbar interconnection for hybrid FPGAs by examining which part of the FPGA should use the crossbar and the ratio of the crossbar to the fine-grained routing resources.

In addition, VPR5.0 [82] includes single driver routing which is commonly use in modern commercial devices. We should adopt this new routing feature of VPR5.0 in VPH.

7.2.6 Feedback path for common subgraph coarse-grained FPU

We adopt common subgraph extraction to optimise the coarse-grained FPU in Chapter 6. The common subcircuits include direct connection between FP adders/subtractors, multipliers and wordblocks with simple multiplexers. They do not make use of the feedback path in the FPU proposed in Chapter 3, Section 3.1.2 (Figure 3.2). The feedback path enables the FPU to represent several common subcircuits as shown in Figure 7.1, at the expense of is using more internal wire connections and feedback multiplexers. It is interesting to study the performance trade-offs of the whole system with and without feedback paths. In order to make a comparison, we need to find the best combination of FP adders/subtractors, FP multipliers and parameters for the FPU with feedback paths (Table 3.1 in Section 3.1.2) to represent all the common subcircuits found in common subgraph extraction (Figure 7.2). Geometric Programming [145] is a technique widely used in digital circuit optimisation, which may have a chance to solve this problem.

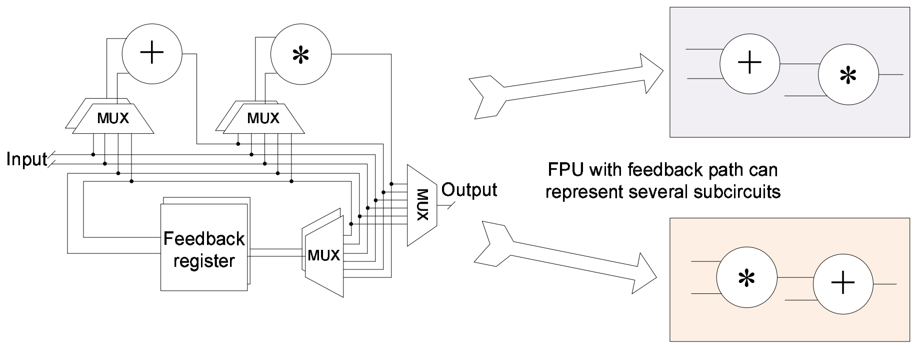


Figure 7.1: An example of representation of an FPU with feedback path

7.2.7 Optimisation of Power Consumption

Altera Stratix devices employ dual voltage technique to reduce power consumption [42]. They have different regions for high speed (high power) processing and low speed (low power) processing in the fine-grained FPGA. Based on this idea, we can further explore the number and the type of coarse-grained elements that should be embedded in high power or low power regions in a hybrid FPGA to achieve better performance (Figure 7.3). Enhanced CAD tools which optimise critical paths for speed or power in an FPGA containing different power regions and embedded blocks are required.

7.2.8 Process Variation

In deep sub-micron technology, there is significant reduction in feature size of transistors. The variation in process, voltage and temperature (PVT [146]) needs to be considered in designing high performance devices, since such devices have to work under a range of parameter values. The variation

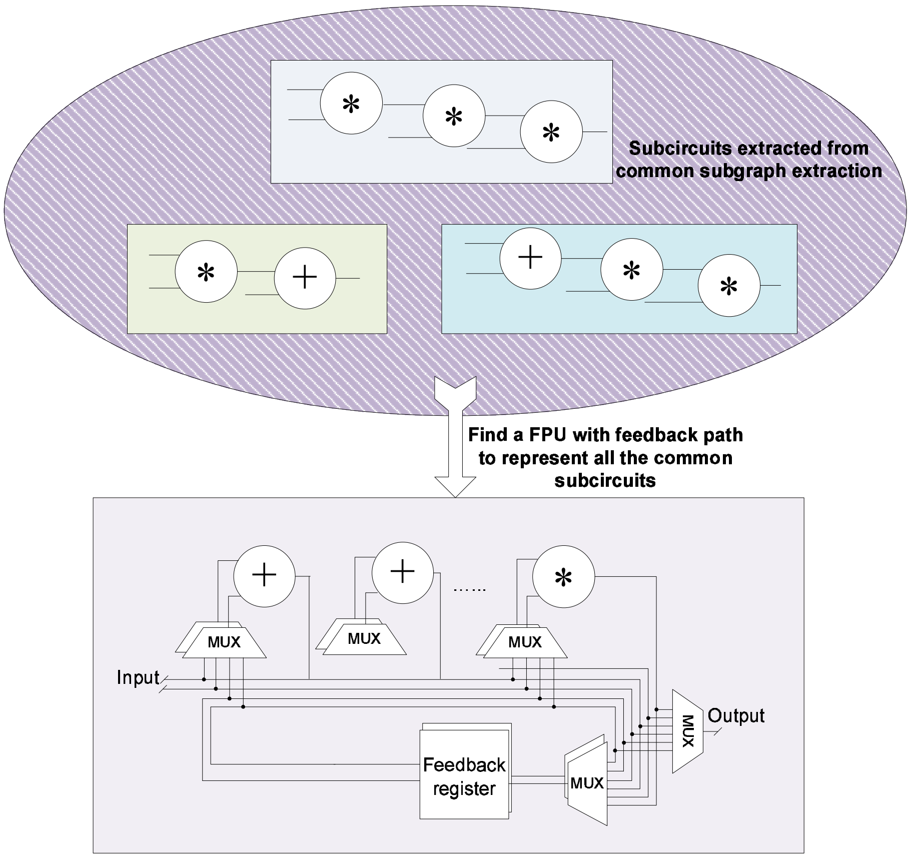

Subcircuits extracted from common subgraph extraction
Find a FPU with feedback path to represent all the common subcircuits
MUX
MUX
MUX
Input
Output
MUX
Feedback register
MUX

Figure 7.2: Merging all common subcircuits to an FPU with feedback path

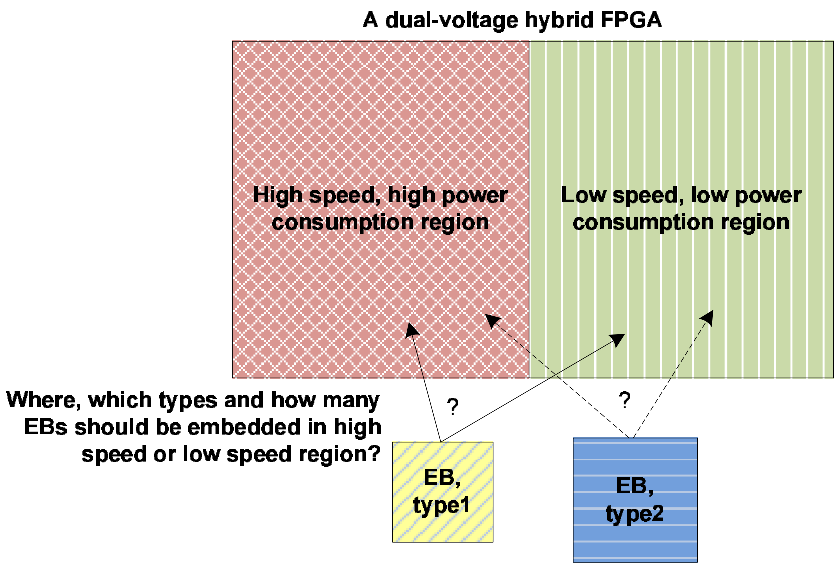

A dual-voltage hybrid FPGA
High speed, high power consumption region
Low speed, low power consumption region
Where, which types and how many EBs should be embedded in high speed or low speed region?
?
?
EB, type1
EB, type2

Figure 7.3: An illustration of the difficulties in embedding EBs in dual-voltage hybrid FPGA

is due to many factors including processing temperatures, equipment properties and wafer polishing. Smaller feature size makes the manufacturing process more difficult to be controlled.

There are studies of process variation in FPGAs [147,148] but the effect of embedding coarse-grained elements is not included. The timing yield optimisation of hybrid FPGA architectures would be an interesting topic to be considered in future work.

Bibliography

[1] I. Kuon , R. Tessier and J. Rose. FPGA Architecture: Survey and Challenges. *Foundations and Trends in Electronic Design Automation,* 2(2):135–253, 2008.

[2] Altera Corp. Stratix III Device Handbook, Vol.1. 2006.

[3] Xilinx Inc. Virtex-5 Family Overview - LX, LXT, and SXT Platforms. 2007.

[4] I. Kuon and J. Rose. Measuring the Gap between FPGAs and ASICs. *IEEE Transactions on Computer-Aided Design (CAD) of Integrated Circuits and Systems,* 26(2):203–215, 2007.

[5] S.J.E. Wilton. Architecture and Algorithms for Field-Programmable Gate Arrays with Embedded Memory. *PhD dissertation, University of Toronto, 1997.*

[6] Actel, ProASIC3 Flash Family FPGAs Datasheet: Device Architecture, 2007.

[7] G.L. Zhang and P.H.W. Leong and C.H. Ho and K.H. Tsoi and C.C.C. Cheung, D. Lee, R.C.C. Cheung and W. Luk. Reconfigurable Acceleration for Monte Carlo Based Financial Simulation. In *Proc. International Conference on Field-Programmable Technology (FPT),* pages 215–222, 2005.

[8] J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone and J.C. Phillips. GPU Computing. *Proceedings of the IEEE,* 96(5):879–899, May 2008.

[9] M. Zechner and M. Granitzer. Accelerating K-Means on the Graphics Processor via CUDA. In *Proc. International Conference on Intensive Applications and Services (INTENSIVE),* pages 7–15, 2009.

[10] S. Che, J. Li, J.W. Sheaffer, K. Skadron and J. Lach. Accelerating Compute-Intensive Applications with GPUs and FPGAs. In *Proc. Symposium on Application Specific Processors (SASP)*, pages 101–107, 2008.

[11] D.B. Thomas, L.W. Howes and W. Luk. A comparison of CPUs, GPUs, FPGAs, and Massively Parallel Processor Arrays for Random Number Generation. In *Proc. International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 63–72, 2009.

[12] C.H. Ho, C.W. Yu, P.H.W. Leong, W. Luk and S.J.E. Wilton. Floating-Point FPGA: Architecture and Modeling. *IEEE Transactions on Very-Large Scale Integration (VLSI) Systems*, 17(2):1709–1718, Dec 2009.

[13] C.W. Yu, A.M. Smith, W. Luk, P.H.W. Leong, S.J.E. Wilton. Optimizing Coarse-grained Units in Floating Point Hybrid FPGA. In *Proc. International Conference on Field-Programmable Technology (FPT)*, pages 57–64, 2008.

[14] C.W. Yu, J. Lamoureux, S.J.E. Wilton, P.H.W. Leong and W. Luk. The Coarse-Grained/Fine-Grained Logic Interface with Embedded Floating-Point Arithmetic Units. *International Journal of Reconfigurable Computing*, 2008, Article ID 736203, 10 pages, 2008.

[15] C.W. Yu, W. Luk, S.J.E. Wilton, P.H.W. Leong. Routing Optimization for Hybrid FPGAs. In *Proc. International Conference on Field-Programmable Technology (FPT)*, pages 419–422, 2009.

[16] V. Betz and J. Rose. VPR: A New Packing, Placement and Routing Tool for FPGA Research. In *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, pages 213–222, 1997.

[17] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W.M. Fang and J. Rose. VPR 5.0: FPGA CAD and Architecture Exploration Tools with Single-driver Routing, Heterogeneity and Process Scaling. In *Proc. International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 133–142, 2009.

[18] J. Rose. Hard vs. Soft: The Central Question of Pre-fabricated Silicon. In *Proc. International Symposium on Multiple-Valued Logic (ISMVL)*, pages 2–5, 2004.

[19] C.W. Yu. A Tool for Exploring Hybrid FPGAs. In *Proc. International Conference on Field Programmable Logic and Applications (FPL), PhD Forum*, pages 509–510, 2007.

[20] http://www.doc.ic.ac.uk/ cyu/VPH.

[21] V. Betz and J. Rose. Effect of the Prefabricated Routing Track Distribution on FPGA Area-Efficiency. *IEEE Transactions on Very-Large Scale Integration (VLSI) Systems*, 6(3):445–456, 1998.

[22] V. Betz and J. Rose. FPGA Routing Architecture: Segmentation and Buffering to Optimize Speed and Density. In *Proc. International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 59–68, 1999.

[23] V. Betz, J. Rose and A. Marquardt. Architecture and CAD for Deep-Submicron FPGAs. *Kluwer Academic Publishers*, 1999.

[24] D. Lewis et al. The Stratix Routing and Logic Architecture. In *Proc. International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 12–20, 2003.

[25] E. Ahmed and J. Rose. The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and Density. *IEEE Transactions on Very-Large Scale Integration (VLSI) Systems*, 12(3):288–298, Mar 2004.

[26] I. Kuon and J. Rose. Area and Delay Trade-offs in the Circuit and Architecture Design of FPGAs. In *Proc. International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 149–158, 2008.

[27] D. Lewis et al. The Stratix II Logic and Routing Architecture. In *Proc. International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 14–20, 2005.

[28] Altera Corp. Stratix III FPGAs vs. Xilinx Virtex-5 Devices: Architecture and Performance Comparison. 2007.

[29] S. Brown, R. Francis, J. Rose and Z. Vranesic. Field-Programmable Gate Arrays. *Kluwer Academic Publishers*, 1992.

[30] N. Weste and K. Eshraghian. *Principles of CMOS VLSI Design*. Addison-Wesley, second edition, 1993.

[31] J. Rose and S. Brown. Flexibility of Interconnection Structures for Field-Programmable Gate Arrays. *IEEE Journal of Solid-State Circuits*, 26(3):277–282, 1983.

[32] V. Betz. Architecture and CAD for Speed and Area Optimization of FPGAs. *Ph.D. Dissertation, University of Toronto, 1998.*

[33] V. Betz and J. Rose. Circuit Design, Transistor Sizing and Wire Layout of FPGA Interconnect. In *Proc. Custom Integrated Circuits*, pages 171–174, 1999.

[34] Y.W. Chang, D.F. Wong and C.K. Wong. Universal Switch Modules for FPGA Design. *ACM Transactions on Design Automation of Electronic Systems*, 1(1):80–101, 1996.

[35] V. Chandra and H. Schmit. Simultaneous Optimization of Driving Buffer and Routing Switch Sizes in an FPGA using an Iso-Area Approach. In *Proc. IEEE Computer Society Annual Symposium on VLSI*, pages 28–33, 2002.

[36] G. Lemieux, E. Lee, M. Tom and A. Yu. Directional and single-driver wires in FPGA interconnect. In *Proc. International Conference on Field-Programmable Technology (FPT)*, pages 41–48, 2004.

[37] L. Zhou, C.C. Cheung, and Y.L. Wu. What if Merging Connection and Switch Boxes — an Experimental Revisit on FPGA Architectures. In *Proc. International Conference on Communications, Circuits and Systems (ICCCAS)*, pages 1295–1299, 2004.

[38] Xilinx Inc. *Power Consumption in 65 nm FPGAs*. White Paper, 2006.

[39] Altera Corp. Stratix III Programmable Power. 2006.

[40] F. Li, Y. Lin, L. He and J. Cong. Low-power FPGA using Pre-defined Dual-Vdd/Dual-Vt Fabrics. In *Proc. International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 42–50, 2004.

[41] F. Li, Y. Lin and L. He. FPGA Power Reduction using Configurable Dual-Vdd. In *Proc. Design Automation Conference (DAC)*, pages 735–740, 2004.

[42] D. Lewis et al. Architectural Enhancements in Stratix-III^{TM} and Stratix-IV^{TM}. In *Proc. International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 33–42, 2009.

[43] L. Shang, A.S. Kaviani and K. Bathala. Dynamic power consumption in Virtex^{TM}-II FPGA family. In *Proc. International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 157–164, 2002.

[44] V. George. Low Energy Field-Programmable Gate Array. *PhD dissertation, University of California, Berkeley, 2000.*

[45] R. Krishnan M. Meijer and M. Bennebroek. Energy-Efficient FPGA Interconnect Design. In *Proc. the conference on Design, automation and test in Europe*, pages 42–47, 2006.

[46] M. Lin and A.E. Gamal. A Routing Fabric for Monolithically Stacked 3D-FPGA. In *Proc. International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 3–12, 2007.

[47] A. Kaviani and S. Brown. Hybrid FPGA architecture. In *Proc. International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 3–9, 1996.

[48] R. Hartenstein. Coarse Grain Reconfigurable Architecture (Embedded Tutorial). In *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 564–570, 2001.

[49] T.J. Todman, G.A. Constantinides, S.J.E. Wilton, O. Mencer, W. Luk, and P.Y.K. Cheung. Reconfigurable computing: architectures and design methods. *IEE Proceedings- Computers and Digital Techniques*, 152(2):193–207, 2005.

[50] P. Jamieson and J. Rose. Enhancing the Area-Efficiency of FPGAs with Hard Circuits Using Shadow Clusters. In *Proc. International Conference on Field-Programmable Technology (FPT)*, pages 1–8, 2006.

[51] S. Shukla, N.W. Bergmann and J. Becker. QUKU: A Coarse Grained Paradigm for FPGA. In *Proc. Dagstuhl Seminar 06141*, 2006.

[52] G. Govindu, L. Zhuo, S. Choi, and V. Prasanna. Analysis of High-performance Floating-point Arithmetic on FPGAs. In *Proc. Parallel and Distributed Processing Symposium*, pages 149–156, 2004.

[53] E. Roesler and B. Nelson. Novel Optimizations for Hardware Floating-Point Units in a Modern FPGA Architecture . In *Proc. International Conference on Field Programmable Logic and Applications (FPL), LNCS*, volume 2438, pages 323–345. Springer, 2002.

[54] M.J. Beauchamp, S. Hauck, K.D. Underwood and K.S. Hemmert. Architectural Modifications to Enhance the Floating-Point Performance of FPGAs. *IEEE Transactions on Very-Large Scale Integration (VLSI) Systems*, 16(2):177–187, Feb. 2008.

[55] Y.J. Chong and S. Parameswaran. Flexible Multi-Mode Embedded Floating-Point Unit for Field Programmable Gate Arrays. In *Proc. International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 171–180, 2009.

[56] S.J.E. Wilton, C.H. Ho, B. Quinton, P.H.W. Leong and W. Luk. A Synthesizable Datapath-Oriented Embedded FPGA Fabric for Silicon Debug Applications. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 1(1):1–25, 2008.

[57] A.M. Smith, G.A. Constantinides and P.Y.K. Cheung. Fused-Arithmetic Unit Generation for Reconfigurable Devices using Common Subgraph Extraction. In *Proc. International Conference on Field-Programmable Technology (FPT)*, pages 105–112, 2007.

[58] S.J.E. Wilton, J. Rose, and Z.G. Vranesic. The Memory/Logic Interface in FPGAs with Large Embedded Memory Arrays. *IEEE Transactions on Very-Large Scale Integration (VLSI) Systems*, 7(1), 1999.

[59] A. Ye and J. Rose. Using Bus-Based Connections to Improve Field-Programmable Gate-Array Density for Implementing Datapath Circuits. *IEEE Transactions on Very-Large Scale Integration (VLSI) Systems*, 14(5):462–473, 2006.

[60] T.S.T. Mak, P. Sedcole, P.Y.K. Cheung and W. Luk. On-FPGA Communication Architectures and Design Factors. In *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8, 2006.

[61] S.Y.L. Chin, C.S.P. Lee and S.J.E. Wilton. On the power dissipation of embedded memory blocks used to implement logic in field-programmable gate arrays. *International Journal of Reconfigurable Computing*, 2008:1–13, 2008.

[62] C.H. Ho, P.H.W. Leong, W. Luk, W. and S.J.E. Wilton. Rapid estimation of power consumption for hybrid FPGAs. In *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, pages 227–232, 2008.

[63] D. Chen, J. Cong and P. Pan. FPGA Design Automation: A Survey. *Foundations and Trends in Electronic Design Automation*, 1(3):139–169, 2006.

[64] Celoxica. http://www.celoxica.com.

[65] J.G.F. Coutinho and W. Luk. Source-directed Transformations for hardware Compilation. In *Proc. International Conference on Field-Programmable Technology (FPT)*, pages 278–285, 2003.

[66] M. Gokhale, J. Stone, J. Arnold and M. Kalinowski. Stream-oriented FPGA computing in the Streams-C high level language . In *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 49–56, 2000.

[67] S. Gupta, N. Dutt, R. Gupta and A. Nicolau. SPARK: a high-level synthesis framework for applying parallelizing compiler transformations. In *Proc. International Conference on VLSI Design*, pages 461–466, 2003.

[68] O. Mencer, D.J. Pearce and L.W. Howes and W. Luk. Design Space Exploration with A Stream Compiler. In *Proc. International Conference on Field-Programmable Technology (FPT)*, pages 270–277, 2003.

[69] M. Weinhardt and W. Luk. Pipeline Vectorization. *IEEE Transactions on Computer-Aided Design (CAD) of Integrated Circuits and Systems*, 20(2):234–248, Feb 2001.

[70] T. Todman, J.G. Coutinho and W. Luk. Customisable Hardware Compilation. *Journal of Supercomputer*, 32(2):119–137, 2005.

[71] Xilinx Inc. ISE Design Suite Software Manuals and Help - PDF Collection, UG681 (v 11.2), June 24, 2009.

[72] Altera Inc. Quartus II Handbook Version 9.0, March, 2009.

[73] A. Chattopadhyay, Z. Rakosi, K. Karuri, D. Kammler, R. Leupers, G. Ascheid and H. Meyr. Pre- and Post-Fabrication Architecture Exploration for Partially Reconfigurable VLIW Processors. In *Proc. International Workshop on Rapid System Prototyping*, pages 189–194, 2007.

[74] A. Chattopadhyay, W. Ahmed, K. Karari, D. Kammler, R. Leupers, G. Ascheid and H. Meyr. Design Space Exploration of Partially Re-configurable Embedded Processors. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1–6, 2007.

[75] S. Kirkpatrick, C.D. Gelatt, Jr. and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

[76] E.W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1(1):269–271, December 1959.

[77] P. Jamieson and J. Rose. A verilog RTL Synthesis Tool for Heterogeneous FPGAs. In *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, pages 305–310, 2005.

[78] M. Tom and G. Lemieux. Logic block clustering of large designs for channel-width constrained FPGAs. In *Proc. Design Automation Conference (DAC)*, pages 726–731, 2005.

[79] C.L. Zhou, W.C. Tang and Y.L. Wu. Fast Placement-Intact Logic Perturbation Targeting for FPGA Performance Improvement. In *Proc. Southern Programmable Logic Conference (SPL)*, pages 63–68, 2007.

[80] T. Wong and S. Wilton. Placement and routing for non-rectangular embedded programmable logic cores in SoC design. In *Proc. International Conference on Field-Programmable Technology (FPT)*, pages 65 –72, 2004.

[81] S.Y.L. Chin, S.J.E. Wilton. Memory Footprint Reduction for FPGA Routing Algorithms. In *Proc. International Conference on Field-Programmable Technology (FPT)*, pages 1–8, 2007.

[82] *VPR and T-VPack 5.0.* `http://www.eecg.toronto.edu/vpr/`.

[83] J. Lamoureux, J. and S.J.E. Wilton. On the interaction between power-aware FPGA CAD Algorithms. In *Proc. International Conference on Computer Aided Design*, pages 701–708, 2003.

[84] K.K.W. Poon, S.J.E. Wilton and A. Yan. A Detailed Power Model for Field-Programmable Gate Arrays. *ACM Transactions on Design Automation of Electronic System*, 10(2):279–302, 2005.

[85] `http://www.ece.ubc.ca/~stevew/powermodel/power_intro.html`.

[86] J. Lamoureux and S.J.E. Wilton. Activity Estimation for Field-Programmable Gate Arrays. In *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8, 2006.

[87] C.H. Ho, P.H.W. Leong, W. Luk, S.J.E. Wilton and S. Lopez-Buedo. Virtual Embedded Blocks: A Methodology for Evaluating Embedded Elements in FPGAs. In *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 35–44, 2006.

[88] W.M. Fang and J. Rose. Modeling routing demand for early-stage FPGA architecture development. In *Proc. International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 139–148, 2008.

[89] A. Lam, S.J.E. Wilton, P.H.W. Leong and W. Luk. An Analytical Model Describing the Relationships between Logic Architecture and FPGA Density. In *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, pages 221–226, 2008.

[90] A.M. Smith, S.J.E. Wilton and J. Das. Wirelength Modeling for Homogeneous and Heterogeneous FPGA Architectural Development. In *Proc. International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2009.

[91] H.L. Yu, Y.H. Chan and P.H.W. Leong. FPGA interconnect design using logical effort. In *Proc. International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 257–257, 2008.

[92] J. Das, S.J.E. Wilton, P.H.W. Leong and W. Luk. Modeling Post-techmapping and Post-clustering FPGA Circuit Depth. In *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, pages 205–211, 2009.

[93] C.W. Yu, K.H. Kwong, K.H. Lee and P.H.W. Leong. A Smith-Waterman Systolic Cell. . *In Patrick Lysaght and Wolfgang Rosenstiel, editors, New Algorithms, Architectures and Applications for Reconfigurable Computing*, pages 291–300, Springer 2003.

[94] K.H. Leung, K.W. Ma, W.K. Wong and P.H.W. Leong. Implementation of a Microcoded Elliptic Curve Cryptographic Processor. In *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 68–76, 2000.

[95] K.K. Ting, S.C.L. Yuen, K.H. Lee and P.H.W. Leong. An FPGA Based SHA-256 Processor. In *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, pages 577–585, 2002.

[96] K.H. Tsoi, K.H. Leung and P.H.W Leong. High Performance Physical Random Number Generator. *Computers and Digital Techniques, IET*, 1(4):349–352, July 2007.

[97] P.H.W. Leong and C.K. Chung. A FPGA Based Runtime Configurable Clause Evaluator for SAT Problems. *Electronics Letters*, 35(19):1618–1619, 1999.

[98] P.H.W Leong, C.W. Sham, W.C. Wong, H.Y. Wong, W.S. Yuen and M.P. Leong. A Bitstream Reconfigurable FPGA Implementation of the WSAT algorithm. *IEEE Transactions on Very-Large Scale Integration (VLSI) Systems*, 9(1):197–201, Feb 2001.

[99] C.K. Wong and P.H.W. Leong. An FPGA-Based Electronic Cochlea with Dual Fixed-Point Arithmetic. In *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–6, 2006.

[100] K.H. Tsoi, D. Rueckert, C.H. Ho and W. Luk. Reconfigurable Acceleration of 3D Image Registration. In *Proc. Southern Programmable Logic Conference (SPL)*, pages 95–100, 2009.

[101] D.B. Thomas, J.A. Bower and W. Luk. Automatic Generation and Optimisation of Reconfigurable Financial Monte-Carlo Simulations. In *Proc. IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 168–173, 2007.

[102] K.H. Tsoi, C.H. Ho, H.C. Yeung and P.H.W. Leong. An Arithmetic Library and Its Application to the N-body Problem. In *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 68–78, 2004.

[103] L. Musa. FPGAS in high energy physics experiments at CERN. In *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, pages 2–2, 2008.

[104] IEEE Task P754. A Proposed Standard for Binary Floating-Point Arithmetic. *IEEE Computer*, 14(12):51–62, 1981.

[105] New York ANSI/IEEE. IEEE Standard for Binary Floating-Point Arithmetic. Technical report, The Insittution of Electrical and Electronics Engineerings, Inc, 1985. IEEE Std 754-1985.

[106] S.F. Oberman, H. Al-Twaijry and M.J. Flynn. The SNAP project: design of floating point arithmetic units. In *Proc. IEEE Symposium on Computer Arithmetic*, pages 156–165, 1997.

[107] J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach*. 2003.

[108] R. Usselmann. *Floating Point Unit*. http://www.opencores.com/project,fpu, 2005.

[109] D. Lundgren. *double_fpu_verilog*. http://www.opencores.org/project,double_fpu, 2009.

[110] A.A. Gaffar, W. Luk, P.Y.K. Cheung, N. Shirazi and J. Hwang. Automating Customisation of Floating-Point Designs. In *Proc. International Conference on Field Programmable Logic and Applications (FPL), LNCS*, volume 2438, pages 241–268. Springer, 2002.

[111] A.A. Gaffar, O. Mencer, W. Luk, P.Y.K. Cheung and N. Shirazi. Floating-point Bitwidth Analysis via Automatic Differentiation. In *Proc. International Conference on Field-Programmable Technology (FPT)*, pages 158–165, 2002.

[112] A.A. Gaffar, O. Mencer and W. Luk. Unifying Bit-width Optimisation for Fixed-point and Floating-point Designs. In *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 79–88, 2004.

[113] H. Fu, W. Osborne, R.G. Clapp, O. Mencer and W. Luk. Accelerating Seismic Computations Using Customized Number Representations on FPGAs. *EURASIP Journal on Embedded Systems*, 2009.

[114] H. Suzuki, H. Morinaka, H. Makino, Y. Nakase, K. Mashiko and T. Sumi. Leading-Zero Anticipatory Logic for High-Speed Floating Point Addition . *IEEE JOURNAL OF SOLID-STATE CIRCUITS*, 31(8), 1996.

[115] P.M. Farmwald. *On the design of high performance digital arithmetic units*. PhD thesis, Stanford, CA, USA, 1981.

[116] J. Liang, R. Tessier and O. Mencer. Floating Point Unit Generation and Evaluation for FPGAs. In *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 185–194, 2003.

[117] Xilinx Inc. *Floating-Point Operator v5.0*. Product Specification, 2009.

[118] C.H. Ho. Automatic Synthesis and Optimization of Floating Point Hardware. *MPhil dissertation, The Chinese University of Hong Kong, 2003.*

[119] K. Turkington, K. Masselos, G.A. Constantinides and P.H.W. Leong. FPGA Based Acceleration of the Linpack Benchmark: A High Level Code Transformation Approach. In *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–6, 2006.

[120] K.D. Underwood and K.S. Hemmert. Closing the Gap: CPU and FPGA Trends in Sustainable Floating-Point BLAS Performance. In *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 219–228, 2004.

[121] Standard performance evaluation corporation. *http://www.spec.org/cpu2006/*. 2006.

[122] S. Yang. Logic Synthesis and Optimization Benchmarks. *Tech Report, Microelectronics Center of North Carolina*, 1991.

[123] T.C.P. Chau, S.M.H. Ho, P.H.W. Leong, P. Zipf and M. Glesner. Generation of Synthetic Floating-Point Benchmark Circuits. In *Proc. IEEE International Symposium on Parallel&Distributed Processing (IPDPS)*, pages 1–9, 2009.

[124] P. D. Kundarewich and J. Rose. Synthetic circuit generation using clustering and iteration. *IEEE Transactions on Computer-Aided Design (CAD) of Integrated Circuits and Systems*, 23(6):869–887, 2004.

[125] S.K. Mitra. *Digital Signal Processing A Computer-Based Approach International Editions 1998*, pages 339–416. McGraw-Hill, 1998.

[126] E.I. Garcia, R. Cumplido and M. Arias. Pipelined CORDIC Design on FPGA for a Digital Sine and Cosine Waves Generator. In *Proc. International Conference on Electrical and Electronics Engineering*, pages 1–4, 2006.

[127] Intel Corp. Introducing the 45nm Next-Generation Intel®Core^{TM} Microarchitecture. *White Paper*, 2007.

[128] NVIDIA Corp. *Quick Guide to NVIDIA GeForce Desktop Graphics Processors*. 2008.

[129] E. Kilgariff and R. Fernando. The GeForce 6 series GPU architecture. In *ACM SIGGRAPH 2005 Courses*, page 29, 2005.

[130] J. Michalakes and M. Vachharajani. GPU Acceleration of Numerical Weather Prediction. In *Proc. IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1–7, 2008.

[131] NVIDIA Corp. *NVIDIA CUDA^{TM} Programming Guide*. 2009.

[132] K. Underwood. FPGAs vs. CPUs: Trends in Peak Floating-Point Performance. In *Proc. International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 171–180, 2004.

[133] Synopsys, Inc. *DesignWare Building Block IP, Datapath – Floating Point Overview*. December 2007.

[134] C. Yui, G. Swift and C. Carmichael. Single Event Upset Susceptibility Testing of the Xilinx Virtex II FPGA. In *Proc. Military and Aerospace Applications of Programmable Logic Conference (MAPLD)*, vol. IV, 2002.

[135] Xilinx Inc. Virtex-II Platform FPGAs: Complete Data Sheet. March 2005(v3.4).

[136] *Predictive Technology Model (PTM)*. `http://www.eas.asu.edu/~ptm/`.

[137] E. Sentovich et al. SIS: A System for Sequential Circuit Analysis. *Tech Report No. UCB/ERL M92/41, University of California, Berkley,1992*.

[138] J. Rubinstein, P. Penfield and M. Horowitz. Signal Delay in RC Tree Networks. *IEEE Transactions on Computer-Aided Design (CAD) of Integrated Circuits and Systems*, 2(3):202–211, 1983.

[139] H. Schmit and V. Chandra. FPGA Switch Block Layout and Evaluation. In *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 11–18, 2002.

[140] User Guide of XtremeDSP DSP48A for Spartan-3A DSP FPGAs (UG431). July 2008(v1.3).

[141] ICARUS, Verilog http://www.icarus.com/eda/verilog.

[142] S. Yusuf, W. Luk, M. Sloman, N. Dulay, E.C. Lupu and G. Brown. Reconfigurable Architecture for Network Flow Analysis. *IEEE Transactions on Very-Large Scale Integration (VLSI) Systems*, 16(1):57–65, Jan. 2008.

[143] M.A. Tahir, A. Bouridane and F. Kurugollu. An FPGA Based Coprocessor for GLCM and Haralick Texture Features and their Application in Prostate Cancer Classification. *Analog Integraded Circuits Signal Processing*, 43(2):205–215, 2005.

[144] H. Fan and Y.L. Wu. Crossbar based design schemes for switch boxes and programmable interconnection networks. In *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC)*, volume 2, pages 910–915, Jan. 2005.

[145] S. Boyd, S.-J. Kim, L. Vandenberghe and A. Hassibi. A Tutorial on Geometric Programming. *Optimization and Engineering*, 8(1):67–127, 2007.

[146] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi and V. De. Parameter Variations and Impact on Circuits and Microarchitecture. In *Proc. Design Automation Conference (DAC)*, pages 338 – 342, 2003.

[147] J.S.J. Wong, P. Sedcole and P.Y.K. Cheung. Self-characterization of Combinatorial Circuit Delays in FPGAs. In *Proc. International Conference on Field-Programmable Technology (FPT)*, pages 17–23, 2007.

[148] P. Sedcole, J.S.J. Wong and P.Y.K. Cheung. Characterisation of FPGA Clock Variability. In *Proc. IEEE International Symposium on VLSI*, pages 322–328, 2008.