Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2008, Article ID 736203, 10 pages
doi:10.1155/2008/736203

Research Article

The Coarse-Grained/Fine-Grained Logic Interface in FPGAs with Embedded Floating-Point Arithmetic Units

Chi Wai Yu,[1] Julien Lamoureux,[2] Steven J. E. Wilton,[2] Philip H. W. Leong,[3] and Wayne Luk[1]

[1] *Department of Computing, Imperial College London, London SW7 2AZ, UK*
[2] *Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, British Columbia, Canada V6T1Z4*
[3] *Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong*

Correspondence should be addressed to Chi Wai Yu, cyu@doc.ic.ac.uk

Received 7 July 2008; Accepted 30 October 2008

Recommended by Gustavo Sutter

This paper 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 interconnect between embedded floating point units (FPUs) and the fine-grained logic fabric in FPGAs. It also studies this interface in FPGAs which contain both FPUs and embedded memories. The results show that (1) FPUs should have a square aspect ratio; (2) they should be positioned near the center of the FPGA; (3) their I/O pins should be arranged around all four sides of the FPU; (4) embedded memory should be located between the FPUs; and (5) connecting higher I/O density coarse-grained blocks increases the demand for routing resources. The hybrid FPGAs with embedded memory required 12% wider channels than the case where embedded memory is not used.

1. Introduction

Significant improvements in the performance, logic density, and power efficiency of field-programmable gate arrays (FPGAs) have made them useful for implementing nearly any type of digital application. In early FPGAs, significant improvements were made by optimizing the fine-grained programmable logic and routing architecture of the FPGA. Today, further improvements are being made by embedding coarse-grained elements such as memories, multipliers, and processors within the fine-grained programmable fabric of the FPGA. Coarse-grained elements can implement a specific function more efficiently than fine-grained programmable logic. However, since they are not as flexible, they only benefit applications which utilize them. This limits the types of embedded blocks which are commercially viable in general-purpose FPGAs to very common circuit elements such as memories, adders, and multipliers. For domain-specific FPGAs, however, additional embedded blocks may make sense. For example, an FPGA that is built specifically to implement applications containing a significant amount of floating point computation would benefit from embedded floating point units. This was explored in [1], in which a domain-specific FPGA that incorporates coarse-grained floating point units (FPUs) was described. The results in [1] show that the embedded floating point units lead to an 18 times density improvement for a set of floating point datapath circuits.

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 paper, we examine this interface. We focus on architectural issues, such as the location of the embedded elements, and the interconnect between the embedded elements and the fine-grained fabric. Our approach is presented in the context of the embedded floating point blocks described in [1]. Specifically, the key contributions of this paper are the following:

(i) a set of parameters that describes the interface between coarse-grained and fine-grained programmable logic in FPGAs;

(ii) an empirical framework to model the impact of coarse-grained architectural parameters in terms of performance, density, and power consumption;

(iii) 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;

(iv) 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.

Although this study focuses on FPGAs with embedded FPUs, its findings may be applicable to other types of embedded computational blocks.

A preliminary version of this work was presented in [2]. This paper further expands the study by considering hybrid FPGAs with more than one type of coarse-grained block. This is important because the different coarse-grained blocks have different I/O density, area, and speed. The connection of those blocks should affect the performance and routing resources required in the hybrid FPGA.

This paper is organized as follows. Section 2 describes related work. Section 3 illustrates the interface between coarse- and fine-grained logic and presents corresponding parameters to describe this interface. Section 4 then presents the empirical framework used to evaluate different interface schemes. Finally, Section 5 presents our results and analysis, and Section 6 summarizes our conclusions.

2. Background

Conventional island-style FPGAs consist mainly of a fine-grained programmable fabric that is made up of configurable logic blocks (CLBs), programmable routing resources, and programmable I/Os. The CLBs consist of one or more k-input lookup tables (k-LUT) and fast local interconnect. Each k-LUT can implement any single output function with k inputs or less. The routing resources implement the interconnect between the CLBs and the I/Os.

A significant number of studies have focused on optimizing this type of FPGA architecture to minimize area, critical-path delay, and power consumption. As an example, the study described in [3] compares different aspects of segmented routing architectures, such as wirelength distribution, switch block implementation, and connection block flexibility, with the goal of creating a fast and area-efficient general-purpose FPGA architecture.

More recent work has focused on adding coarse-grained blocks within the fine-grained fabric. Examples of this include embedded arithmetic multipliers [4, 5] and embedded processors [5]. Coarse-grained blocks improve area and delay since they can implement specific functions more efficiently than the fine-grained logic [6]. On the other hand, coarse-grained blocks waste area when they are not used by an application. FPGAs vendors must consider this tradeoff to determine the type and number of coarse-grained blocks that should be embedded within their devices.

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. A number of recent approaches have been proposed in the literature. In [7], a coarse-grained architecture with bus-based interconnect has been shown to reduce area for datapath circuits. In [8], a tool that generates a domain-specific reconfigurable fabric that is tailored to a specified set of application has been proposed. In [9], the QUKU architecture which merges coarse-grained reconfigurable processing element array and FPGA architectures has been described. This two-level reconfigurable architecture provides active support for fast and efficient dynamic reconfiguration. Enzler et al. [10] has proposed a framework for the cycle-accurate performance evaluation of hybrid reconfigurable processors on the system level, which is based on data-streaming applications. In [1], a domain-specific hybrid FPGA architecture that targets floating point arithmetic applications by incorporating floating point units within a fine-grained programmable fabric has been presented; this architecture is shown to be 18 times more area-efficient than a purely fine-grained architecture for floating point arithmetic applications.

One of the key parts of an FPGA with embedded coarse-grained blocks is the routing structure between the embedded blocks and the fine-grained logic resources. If the coarse-grained/fine-grained interface is not flexible enough, many applications will be unroutable. On the other hand, if the interface is overly flexible, the routing resources will be slower and consume more area than is necessary. Although a number of studies have proposed new coarse-grained blocks and hybrid FPGA architectures, few have examined the interface between the coarse-grained blocks and fine-grained fabric in significant detail. In [11], the local routing resources that connect CLBs to the FPGA routing resource are shared with the embedded blocks to minimize 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, for embedded blocks which has higher I/O pin densities than the existing routing resources are not sufficient. In [12], the interface between embedded memory blocks and

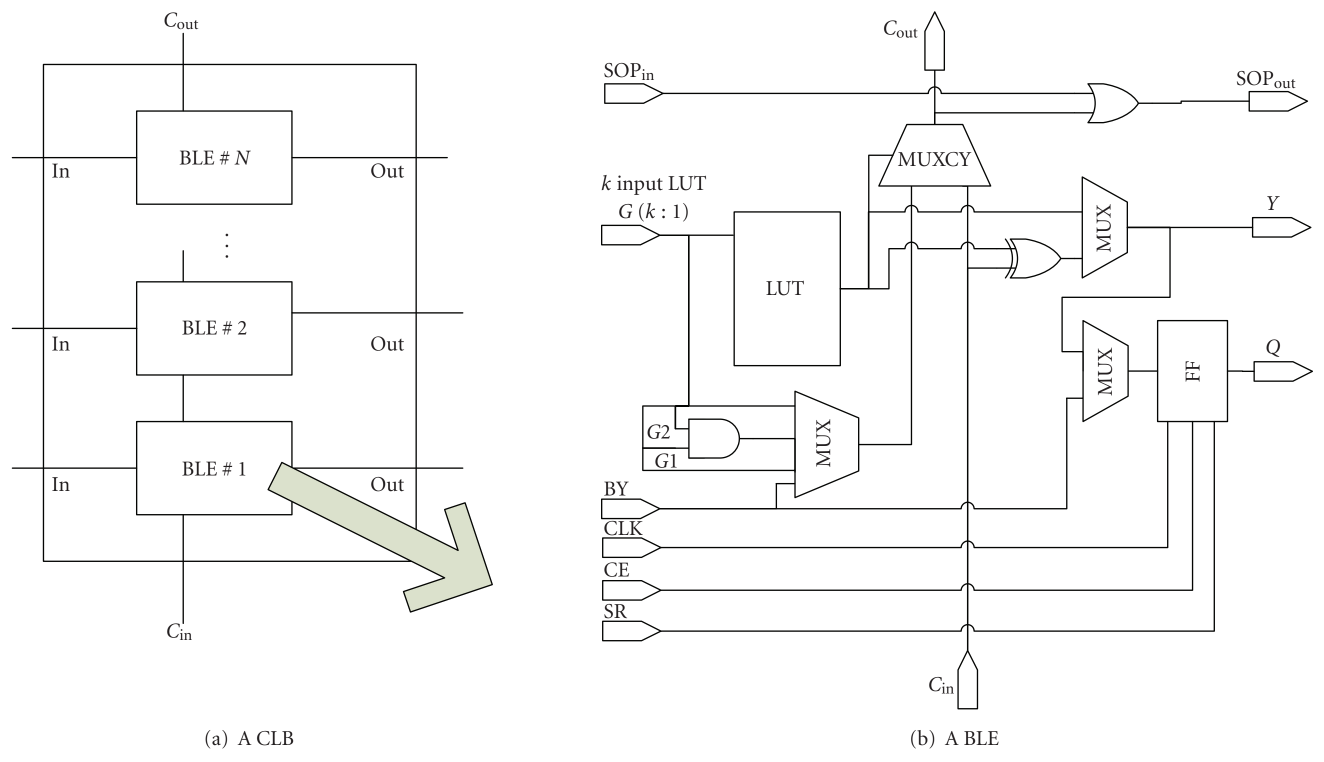


Figure 1: A configurable logic block and the basic logic element inside.

fine-grained programmable logic is examined. Memories are quite different from computation blocks, and so we expect that the interface presented in [12] would not be suitable for our problem.

3. Coarse/Fine-Grained Interface

In this section, we describe the architecture of the blocks used in this work. We first present our assumptions regarding the fine and coarse-grained logic and then give a description of a generic interface architecture with parameters that cover the space of architectures considered.

3.1. Fine-Grained FPGA Assumptions

We assume that the fine-grained resources in the FPGA consist of a grid of identical configurable logic blocks (CLBs), each containing N basic logic elements (BLEs). Each BLE contains a k-LUT and flip flop. The CLB also contains support for carry chains, shift registers, internal multiplexers, and XOR gates. Figure 1 shows the CLB and BLE modelled.

The CLBs are connected using horizontal and vertical channels, as described in [3]. Each channel contains W parallel routing tracks of length 1 and is connected to neighbouring CLBs using a connection block, and intersecting channels using a switch block. We use the subset switch block (also known as disjoint) with $Fc_{\text{switch}} = W$, $Fs = 3$, $Fc_{\text{output}} = 1$, $Fc_{\text{input}} = 1$, and $Fc_{\text{pad}} = 1$ [3].

3.2. Coarse-Grained Block Assumptions

In general, FPGA-based floating point application circuits can be divided into control and datapath circuits. The datapath occupies most of the area in the form of FPUs. The required processing mainly consists of addition, subtraction, and multiplication. We adopt the coarse-grained floating point blocks described in [1]. The datapath circuit is implemented in this floating point block. The floating point multiplier block is a fixed-function block. The floating point adder block can be configured for either floating point addition or subtraction. Each block has a reconfigurable registered output and associated control input and status output signals. A wordblock contains N identical bitblocks. Bitblock contains two 4-input LUTs and a reconfigurable output register. 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 a register, can efficiently implement operations such as addition and multiplexing.

In our assumption, each coarse-grained block contains two double precision floating point adders, two double precision floating point multipliers, and five wordblocks which can efficiently implement operations such as addition and multiplexing as shown in Figure 2.

In addition to FPUs, we also consider embedded memories. Specially, we consider block-selected RAMs (BRAMs) as described in [13]. The details of floating point blocks and BRAMs are shown in Table 1.

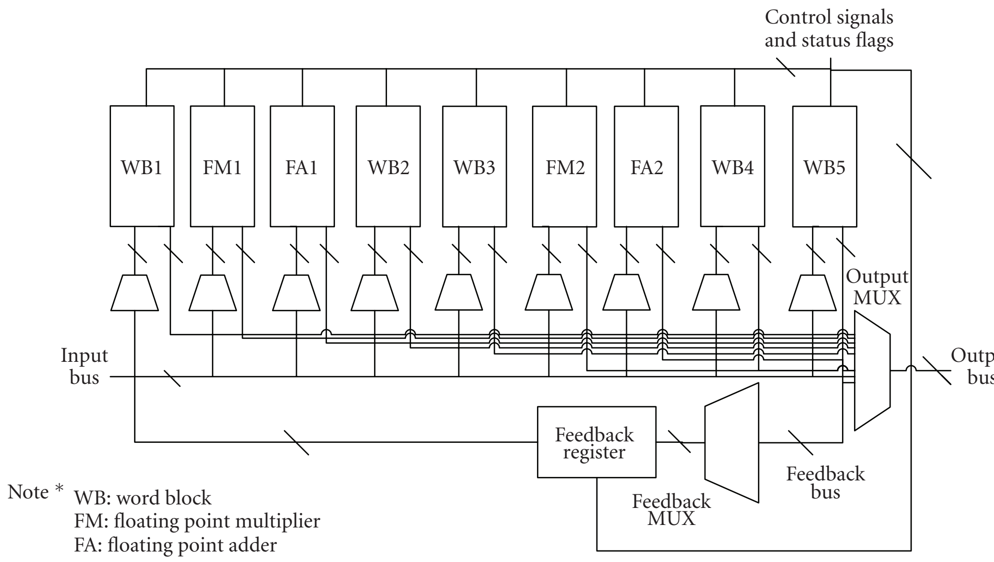


FIGURE 2: Coarse-grained unit modelled in this paper.

TABLE 1: Statistic of the coarse-grained blocks used.

	BRAM	Floating point unit
Number of input	90	286
Number of output	64	258
Area (no. of CLB)	8	182
Delay (nanoseconds)	2.1	9.2

3.3. Coarse-Grained Interface

Based on our detailed area model, we estimate that our embedded FPU consumes roughly the same amount of area as 182 tiles. Each tile represents a CLB and its associated interconnect, buffer, and configuration bit. To embed an FPU, we remove a 13 × 14 grid of CLBs, and replace them with a single EB. Figure 3 shows an example of replacing 3×3 grid of CLBs by a single embedded block (EB). We assume that the EB pins connect to the routing architecture through connection blocks, similar to those used for CLBs. Although other connection patterns are possible (see, e.g., [12]), this pattern allows us to minimize the number of changes to the existing FPGA routing architecture, so that we can leverage the significant amount of previous work on FPGA routing structures. We also assume that the gridded routing fabric extends over the embedded block, as shown in Figure 3. Given the large number of metal layers available in modern CMOS processes, it is reasonable that tracks can easily be placed on top of the embedded blocks. In Figure 3, the four switch blocks required at the interface of the horizontal and vertical channels must coexist with the embedded block; the embedded block, which takes the same area as nine CLBs, includes the area of these four switch blocks. Although it would be possible to consider architectures in which the grid is "broken" [14], it would require changes to the detailed routing architecture. In addition to FPUs, the memories are embedded in the hybrid FPGA under the same assumption. However, the area and the delay of the memories are different to FPUs, which is shown in Table 1. The size of BRAM is 2×4 tiles.

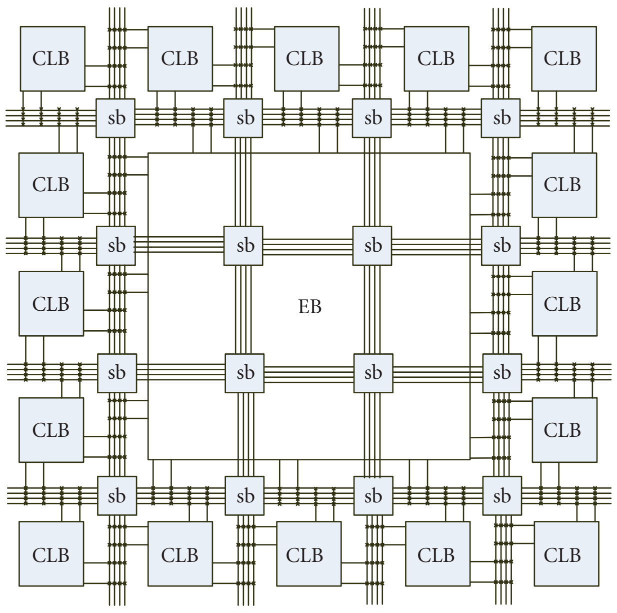
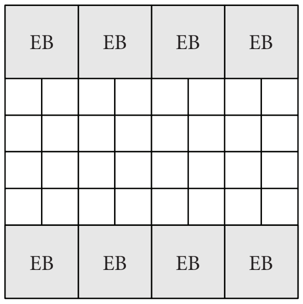


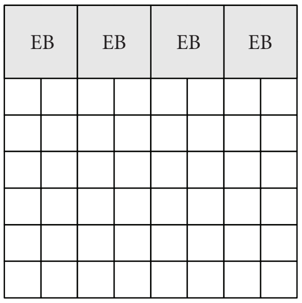
FIGURE 3: Connection between coarse- and fine-grained units through switch box (sb).

3.4. Interface Parameters—Single EB Type

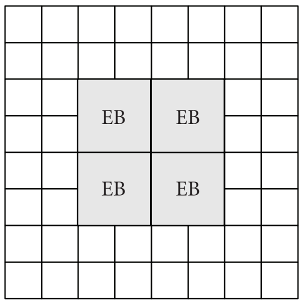
In this paper, we consider a range of interface architectures. First, we explore the single EB-type hybrid FPGA. To describe the space of single EB-type architectures that we consider, we define the following parameters.



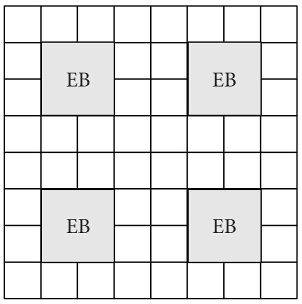
(a) Type 1: EBs are on the top and bottom of CLBs



(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: Various positions of the EBs relative to the fine-grained CLBs.

(1) Single EB Position

The embedded blocks can be placed in various places within the FPGA. In this paper, we consider the positions as shown in Figure 4.

(2) Single EB Pin Location

Figure 5 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), and (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).

(3) Single EB Channel Width

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) Single EB Shape

Several layouts of each embedded block are possible. We consider various aspect ratios.

3.5. Interface Parameters: Multiple EB Types

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 architecture may be different from the single type EB FPGA. Therefore, we investigate the following parameters for the hybrid FPGAs with two types of embedded blocks.

(1) Multiple EB Position

We arrange the different EB types in various ways as shown in Figure 6. 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.

(2) Multiple EB Channel Width

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, more wires are 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. Methodology

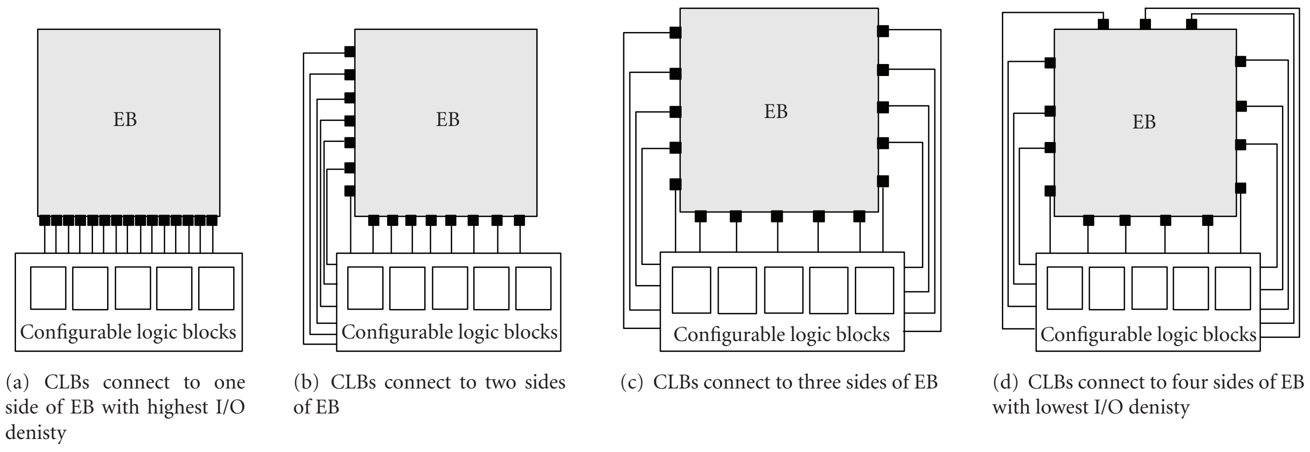
We employ an empirical methodology to examine the impact of the interface parameters described in the previous section. This section describes the benchmark circuits, the CAD tools, and the model that are used.

4.1. Domain-Specific Benchmark Circuits

First we use six double precision floating point benchmark circuits [15] with only one kind of coarse-grained embedded block. They are (1) *bfly*: the basic component of fast Fourier transform: $z = y + x*w$ using complex numbers; (2) *dscg*: a digital sine-cosine generator; (3) *fir4*: a 4-tap finite impulse response filter; (4) *mm3*: a 3×3 matrix multiplication circuit, (5) *ode*: an ordinary differential equation solver; (6) *bgm*: a datapath to compute Monte Carlo simulations of interest rate model derivatives priced under the Brace-Gątarek-Musiela (BGM) framework.

These benchmarks are chosen since they each involve a significant amount of floating point computation. Since *bfly*, *dscg*, *fir4*, *ode*, and *mm3* contain a small number of fine-grained units, each core is replicated four times and are connected together. For example, a *dscg* benchmark contains four *dscg* cores connected together. All circuits use a single global clock. The amount of FPUs and CLBs used for each benchmark circuit is shown in Table 2.

For the experiment involving embedded memories, we add BRAMs to the benchmark circuits. It is more realistic



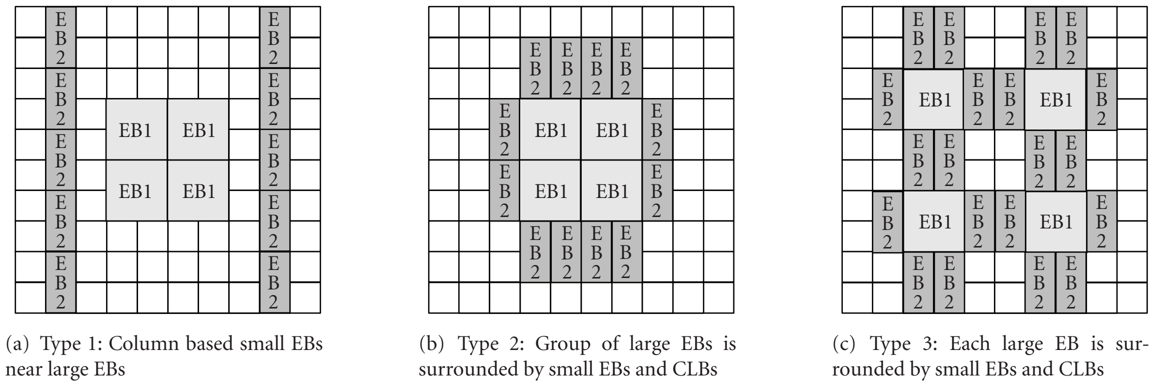
(a) CLBs connect to one side of EB with highest I/O denisty

(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 denisty

Figure 5: Different pin positions in EB.



(a) Type 1: Column based small EBs near large EBs

(b) Type 2: Group of large EBs is surrounded by small EBs and CLBs

(c) Type 3: Each large EB is surrounded by small EBs and CLBs

Figure 6: Various positions of the multiple EBs.

to store the input and output data of the applications in internal BRAMs rather than store the data in of-chip memories. The BRAM data lines are connected to primary input/output of the benchmark circuits. The BRAM address lines are connected to counters which are also added to the benchmark circuits. The adders do not affect the performance because they are implemented using fast carry chains, the delay of which is small compared to the delay of 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 2.

4.2. VPH: Versatile Place and Route for Hybrid FPGAs

We use the evaluation tool VPH to explore our architectures. VPH is a modified version of the VPR tool, with support for embedded blocks, complex logic blocks, carry chains, and constraint files [16]. The tool is available at [17]. In the VPH design flow, shown in Figure 7, applications and coarse-grained elements are written in a high-level hardware description language (VHDL) and synthesized to a mapped

Table 2: Amount of FPU, CLB, and also BRAM used in each benchmark circuit.

Benchmarks	*bgm*	*dscg*	*bfly*	*ode*	*mm3*	*fir4*
No. of CLB	6433	649	884	430	876	282
No. of FPU	7	8	8	8	8	8
No. of BRAM	18	22	40	25	12	12

library netlist in VHDL format using Synplicity's Synplify Premier 8.5 tool. The library netlist contains the usage and connection of simple units such as registers, LUTs, internal multiplexors, and internal inverters. The basic logic block packing tool, VPHpack, packs these units into basic logic elements (BLEs). VPHpack clusters BLEs into CLBs.

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 Synopsys Design Compiler V-2004.06. 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,

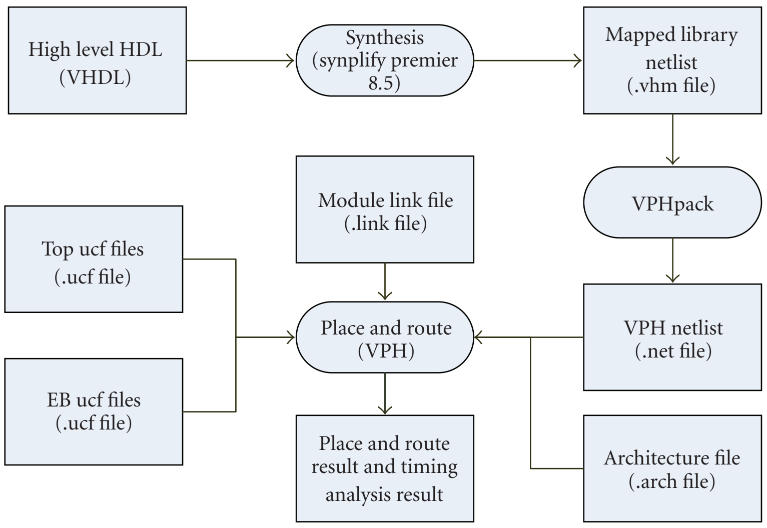

High level HDL (VHDL)
Synthesis (synplify premier 8.5)
Mapped library netlist (.vhm file)
VPHpack
Module link file (.link file)
Top ucf files (.ucf file)
Place and route (VPH)
VPH netlist (.net file)
EB ucf files (.ucf file)
Place and route result and timing analysis result
Architecture file (.arch file)

Figure 7: Design flow of exploration using VPH.

and timing analysis to produce area and delay estimates for each benchmark circuit.

5. Results and Discussion—Single EB Type

In this section, the impact of the interface parameters in Section 3 on hybrid FPGAs is studied. In the experiments conducted, the default architecture parameters are (1) CLB with 2× 4-LUTs, (2) type 3 EB position (Figure 4) as it gives the best performance for the first experiment, (3) channel width 80; since the maximum I/O density of the EBs is 42 pins per slice width, we choose 80 to be the channel width to facilitate routing, (4) size of the floating point unit is 13 × 14 CLBs. We use higher routing effort than our preliminary version of work in [2]; therefore, the experiments result can achieve higher speed than our previous work.

5.1. Single EB Position Results

We first examine how the position of the EBs affects the overall performance of the device. As shown in Figure 4, 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.

Figure 8 shows the results for each of the positioning strategies described in Figure 4. 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 most 14.4% in speed improvement compared to other positioning strategies. The critical path of our circuits tends to include nets that connect multiple EBs; thus placing the EBs close to each other is beneficial.

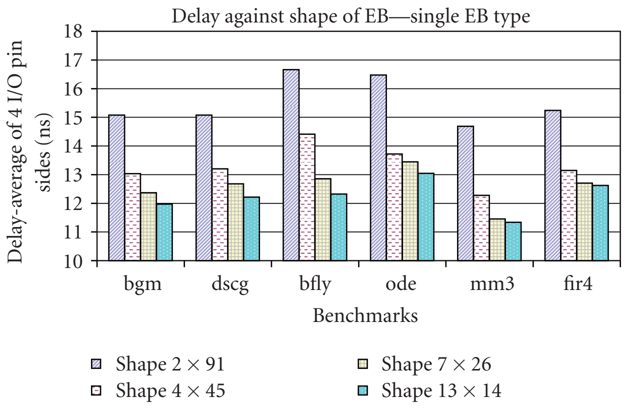


Figure 8: Delay against various EBs positions for single EB type FPGA, as defined in Figure 4.

5.2. Single EB Pin Location Results

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.

The results are shown in Tables 3 and 4. The critical path of the circuit is slightly smaller if all pins are placed on a single side of the embedded block. In several of our benchmarks, the critical path includes a path from one EB, through a register in a CLB, into another EB. These connections are shorter if the pins are close together. On the other hand, Table 4 shows that the routing demand in each channel can be reduced by distributing the pins evenly around each EB. Compared to the configuration in which all pins are on one side of the block, evenly distributing the pins

Table 3: Critical path delay in ns for different EB's I/O positions as shown in Figure 5 for single EB-type FPGA. The percentage shows the deviation from the 1 side result.

Circuits	1 side	2 sides	3 sides	4 sides
	(42 wires/clb)	(21 wires/clb)	(14 wires/clb)	(11 wires/clb)
bgm	12.01 (0%)	11.92 (−0.7%)	11.93 (−0.7%)	12.03 (0.2%)
dscg	12.12 (0%)	12.34 (1.8%)	12.13 (0.1%)	12.28 (1.3%)
bfly	12.42 (0%)	12.38 (−0.3%)	12.30 (−1.0%)	12.19 (−1.9%)
ode	13.02 (0%)	13.30 (2.2%)	12.79 (−1.8%)	13.06 (0.3%)
mm3	11.22 (0%)	11.53 (2.8%)	11.31 (0.8%)	11.29 (0.6%)
fir4	12.63 (0%)	12.79 (1.3%)	12.41 (−1.7%)	12.67 (0.3%)
Average	12.23 (0%)	12.37 (1.1%)	12.14 (−1.9%)	12.25 (0.9%)

Table 4: Minimum channel width (number of wires) for different I/O positions for single EB-type FPGA as shown in Figure 5. The percentage shows the deviation from the 1 side result.

Circuits	1 side	2 sides	3 sides	4 sides
bgm	44 (0%)	44 (0%)	30 (−32%)	27 (−39%)
dscg	43 (0%)	44 (2%)	30 (−30%)	33 (−23%)
bfly	44 (0%)	44 (0%)	38 (−14%)	37 (−16%)
ode	43 (0%)	44 (2%)	35 (−19%)	33 (−23%)
mm3	45 (0%)	45 (0%)	29 (−36%)	30 (−33%)
fir4	42 (0%)	44 (5%)	32 (−24%)	29 (−31%)
Average	43.5 (0%)	44.2 (1.6%)	32.3 (−26%)	31.5 (−28%)

reduces the channel width by 39%. We conclude that this is the best choice.

5.3. Single EB 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 an increased delay.

The results in Figure 9 show the effect of EB to CLB channel width on delay. For routable circuits, rather surprisingly, the average variation is less than 3%. We believe that this is due to critical paths being routed efficiently, so once the circuit is routable, channel width does not affect delay.

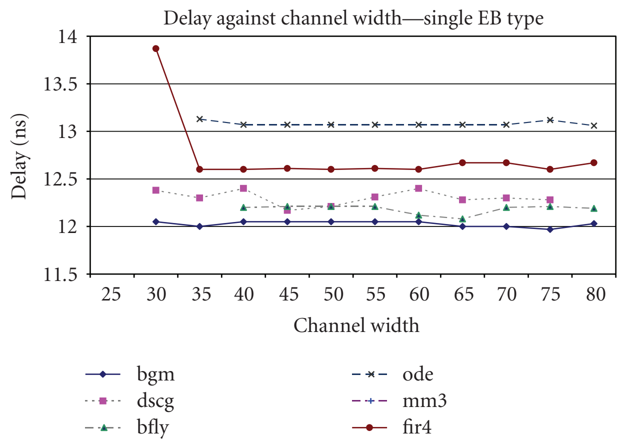


Figure 9: Delay against channel width for single EB-type FPGA.

5.4. Single 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 a change in the delay of the nets connecting these pins.

We modify the shape of the EBs from rectangular (2×91) to square (13×14); the width and height are counted in the number of CLBs. The results in Figure 10 show that square EBs are the most efficient for all applications and result in a 14.4% speed improvement compared to the 2×91 shape. Square EBs lead to a better worst-case delay between the EBs, shortening the critical path in our benchmark circuits.

6. Results and Discussion: Multiple EB Types

After finding the best parameters for the single EB-type case, 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 this system. The size of BRAM is 2×4 CLBs in these experiments.

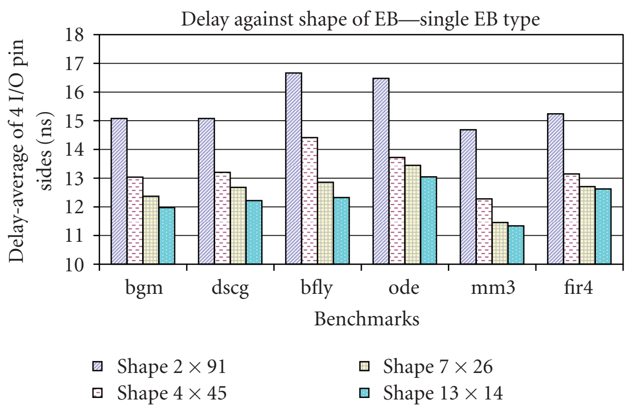


Figure 10: Delay against various EBs' shape for single EB-type FPGA.

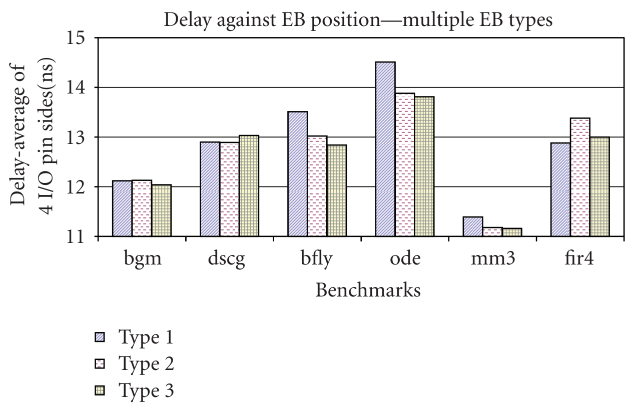


Figure 11: Delay against various EBs positions for multiple EB types FPGA, as defined in Figure 6.

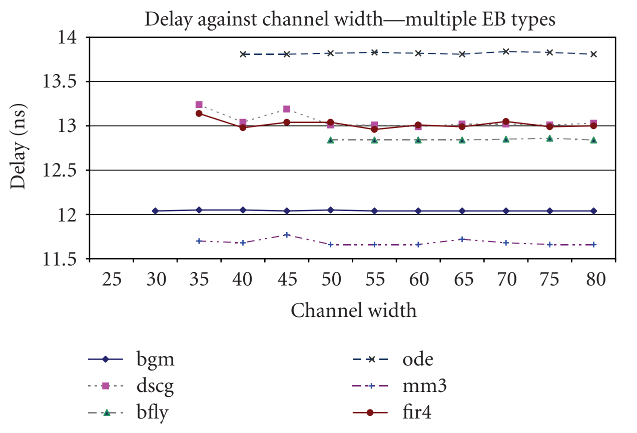


Figure 12: Delay against channel width for multiple EB types FPGA.

Table 5: Minimum channel width of multiple EB types FPGA.

Circuits	Minimum channel width	Channel width increase (%) (I/O pos.: 4 sides, type 3 in Figure 4)
bgm	27	0
dscg	35	6.06
bfly	50	35.14
ode	36	9.09
mm3	32	6.66
fir4	33	13.79
Average	35.5	11.79

6.1. Multiple EB Position Results

We first explore the effect of the BRAMs position on the floating point hybrid FPGA. Figure 11 shows the best location between floating point units which corresponds to type 3 in Figure 6. This configuration performs better than the traditional column-based BRAM (type 1 in Figure 6) used in Xilinx devices because the connections between floating point units and BRAMs are reduced in this case. In a similar way shown in Section 5.1, placing the embedded blocks closer together reduces the length of the bus-based connections between the embedded blocks which improves performance and reduces congestion.

6.2. Multiple EB Interconnect Flexibility

Finally, we investigate routing resources for the multiple EB system. Figure 12 shows delay for different channel widths. The channel width is observed to be nearly constant which is similar to the case discussed in Section 5.3. Table 5 shows the increase in channel width required when embedded BRAM is introduced. Since both FPUs and BRAMs have large I/O requirements, an increase in channel width of 12% over the case without embedded BRAM is required.

7. Conclusion

This paper 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 FPGA. We find that (a) the EBs should be positioned close to each other in the middle of the chip, (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 implementations (e) smaller EBs should be located between large EB to achieve higher speed, and (f) embedding higher I/O density EB types leads to more routing resources being consumed. Although our results are specific to the architecture studied, we believe that they can be applied to FPGAs containing other types of embedded blocks. Current and future work includes extending our methodology to cover other embedded blocks for different domain-specific applications.

Acknowledgment

The support of UK Engineering and Physical Sciences Research Council (EP/D060567/1 and EP/C549481/1), Agility Celoxica and Xilinx are gratefully acknowledged.

References

[1] 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 *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '07)*, pp. 196–201, Amsterdam, The Netherlands, August 2007.

[2] C. W. Yu, J. Lamoureux, S. J. E. Wilton, P. H. W. Leong, and W. Luk, "The coarse-grained/fine-grained logic interface in FPGAs with embedded floating-point arithmetic units," in *Proceedings of the 4th Southern Conference on Programmable Logic (SPL '08)*, pp. 63–68, San Carlos de Bariloche, Argentina, March 2008.

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

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

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

[6] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.

[7] A. Ye, J. Rose, and D. Lewis, "Architecture of datapath-oriented coarse-grain logic and routing for FPGAs," in *Proceedings of IEEE Custom Integrated Circuits Conference (CICC '03)*, pp. 61–64, San Jose, Calif, USA, September 2003.

[8] K. Compton and S. Hauck, "Totem: custom reconfigurable array generation," in *Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '01)*, pp. 111–119, Rohnert Park, Calif, USA, April-May 2001.

[9] S. Shukla, N. W. Bergmann, and J. Becker, "QUKU: a coarse grained paradigm for FPGA," in *Proceedings of the Dagstuhl Seminar 06141*, Dagstuhl, Germany, April 2006.

[10] R. Enzler, C. Plessl, and M. Platzner, "System-level performance evaluation of reconfigurable processors," *Microprocessors and Microsystems*, vol. 29, no. 2-3, pp. 63–73, 2005.

[11] P. Jamieson and J. Rose, "Enhancing the area-efficiency of FPGAs with hard circuits using shadow clusters," in *Proceedings of IEEE International Conference on Field Programmable Technology (FPT '06)*, pp. 1–8, Bangkok, Thailand, December 2006.

[12] 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 Systems*, vol. 7, no. 1, pp. 80–91, 1999.

[13] "Virtex-II Plaform FPGAs: Complete Data Sheet," (v3.4), March 2005, http://direct.xilinx.com/bvdocs/publications/ds031.pdf.

[14] T. Wong and S. J. E. Wilton, "Placement and routing for non-rectangular embedded programmable logic cores in SoC design," in *Proceedings of IEEE International Conference on Field-Programmable Technology (FPT '04)*, pp. 65–72, Brisbane, Australia, December 2004.

[15] 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 *Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '06)*, pp. 35–44, Napa, Calif, USA, April 2006.

[16] C. W. Yu, "A tool for exploring hybrid FPGAs," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '07)*, pp. 509–510, Amsterdam, The Netherlands, August 2007.

[17] http://www.doc.ic.ac.uk/~cyu/vph.zip.