Floating-Point FPGA: Architecture and Modeling

Chun Hok Ho, *Student Member, IEEE*, Chi Wai Yu, Philip Leong, *Senior Member, IEEE*, Wayne Luk, *Fellow, IEEE*, and Steven J. E. Wilton, *Senior Member, IEEE*

***Abstract*—This paper presents an architecture for a reconfigurable device that is specifically optimized for floating-point applications. Fine-grained units are used for implementing control logic and bit-oriented operations, while parameterized 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 scheme is proposed to model embedded blocks using existing field-programmable gate array (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 computer-aided design vendors is then applied and static timing analysis can be used to estimate the performance of the FPGA with the embedded blocks. On selected floating-point benchmark circuits, our results indicate that the proposed architecture can achieve four times improvement in speed and 25 times reduction in area compared with a traditional FPGA device.**

***Index Terms*—Architecture, embedded blocks, field-programmable gate array (FPGA), floating point, modeling.**

I. Introduction

FIELD-programmable gate array (FPGA) technology has been widely adopted to speed up computationally intensive applications. Most current FPGA devices employ an island-style fine-grained architecture [1], with additional fixed-function heterogeneous blocks such as multipliers and block RAMs; these have been shown to have severe area penalties compared with application-specific integrated circuits (ASICs) [2]. In this paper, we propose an architecture for FPGAs that are optimized for floating-point applications. Such devices could be used for applications in DSP, control, high-performance computing, and other applications that have large dynamic range, convenience, and ease-of-verification compared with traditional fixed-point designs on conventional FPGAs.

The proposed floating-point FPGA (FPFPGA) architecture has both fine- and coarse-grained blocks, such usage of multiple granularity having advantages in speed, density, and power over more conventional heterogeneous FPGAs. The coarse-grained block is used to implement the datapath, while lookup table (LUT) based fine-grained resources are used for implementing state machines and bit level operations. In our architecture, the coarse-grained blocks have flexible, parameterized architectures that are synthesized from a hardware description language. This allows tuning of the parameters in a quantitative manner to achieve a good balance between area, performance, and flexibility.

One major issue when evaluating new architectures is determining how a fair comparison to existing commercial FPGA architectures can be made. The Versatile Place and Route (VPR) tool [1] is widely used in FPGA architecture research; however, the computer-aided design (CAD) algorithms used within are different from those of modern FPGAs, as is its underlying island-style FPGA architecture. As examples, VPR does not support retiming, nor does it support carry chains that are present in all major FPGA devices. To enable modeling of our FPFPGA and comparison with a standard island-style FPGA, we propose a methodology to evaluate an architecture based on an existing FPGA device. The key element of our methodology is to adopt virtual embedded blocks (VEBs), created from the reconfigurable fabric of an existing FPGA, to model the area, placement, and delay of the embedded blocks to be included in the FPGA fabric. Using this method, the impact of incorporating embedded elements on performance and area can be quickly evaluated, even if an actual implementation of the element is not available.

The key contributions of this paper are as follows.

1) A novel FPFPGA architecture combining fine-grained resources combined with design-time parameterizable coarse-grained units that are reconfigurable at runtime. To the best of our knowledge, this is the first time such a scheme has been proposed.
2) The VEB methodology that allows modeling of FPGA architectures with embedded blocks and comparisons with commercial FPGAs.
3) Experimental results over various applications for the FPFPGA device.

This paper is organized as follows. Section II describes related work and existing FPGA architectures. Section III describes the proposed FPFPGA architecture. An example mapping is presented in Section IV. Section V discusses the requirements and the associated design challenges of an FPFPGA compiler. The evaluation methodology, including a review of the VEB flow, is described in Section VI, and the evaluation is given in Section VII. Section VIII summarizes our research and discusses opportunities for future research.

Manuscript received April 16, 2008; revised September 12, 2008. First published April 14, 2009; current version published November 18, 2009. This work was supported in part by the U.K. Engineering and Physical Sciences Research Council (EPSRC) under Grant EP/C549481/1 and Grant EP/D060567/1 and by the Research Grants Council of the Hong Kong Special Administrative Region, China Earmarked Grant CUHK413707.

C. H. Ho, C. W. Yu, and W. Luk are with the Department of Computing, Imperial College, London SW7 2AZ, U.K. (e-mail: cho@doc.ic.ac.uk).

P. Leong is with the Department of Computing Science and Engineering, Chinese University of Hong Kong, Shatin N.T., Hong Kong.

S. J. E. Wilton is with the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC V6T 1Z4, Canada.

Digital Object Identifier 10.1109/TVLSI.2008.2006616

1063-8210/$26.00 © 2009 IEEE

II. Background

A. Related Work

FPGA architectures containing coarse-grained units have been reported in the literature. Compton and Hauck propose a domain-specific architecture that allows the generation of a reconfigurable fabric according to the needs of the application [3]. Ye and Rose suggest a coarse-grained architecture that employs bus-based connections, achieving a 14% area reduction for datapath circuits [4].

The study of embedded heterogeneous blocks for the acceleration of floating-point computations has been reported by Roesler and Nelson [5] as well as Beauchamp *et al.* [6]. Both studies conclude that employing heterogeneous blocks in a floating-point unit (FPU) can achieve area saving and increased clock rate over a fine-grained approach.

Leijten–Nowak and van Meerbergen [7] proposed mixed-level granularity logic blocks and compared their benefits with a standard island-style FPGA using the VPR tool [1]. Ye *et al.* [8] studied the effects of coarse-grained logic cells (LCs) and routing resources for datapath circuits, also using VPR. Kuon and Rose [2] reported the effectiveness of embedded elements in current FPGA devices by comparing such designs with the equivalent ASIC circuit in 90-nm process technology.

Beck modified VPR to explore the effects of introducing hard macros [9], while Beauchamp *et al.* augmented VPR to assess the impact of embedding FPUs in FPGAs [6]. We are not aware of studies concerning the effect of adding arbitrary embedded blocks to existing commercial FPGA devices, nor of methodologies to facilitate such studies.

In earlier work, we described the VEB technique for modeling heterogeneous blocks using commercial tools [10], domain-specific hybrid FPGAs [11], and a word-based synthesizable FPGA architecture [12]. This paper provides a unified view of these studies, describes the proposed FPGA architecture in greater detail, presents improved results through the use of a higher performance commercial floating-point core, introduces the mapping process for the FPFPGA, discusses the requirement of a hardware compiler dedicated to such FPFPGA device, and includes two new synthetic benchmark circuits in the study, one of which is twice the size of the largest circuit studied previously.

B. FPGA Architectures

An FPGA is typically constructed as an array of fine-grained or coarse-grained units. A typical fine-grained unit is a K-input LUT, where K typically ranges from 4 to 7, and can implement any K-input Boolean equation. We call this an LUT-based fabric. Several LUT-based cells can be joined in a hardwired manner to make a cluster. This greatly reduces area and routing resources within the fabric [13].

Heterogeneous functional blocks are found in commercial FPGA devices. For example, a Virtex II device has embedded fixed-function 18-bit multipliers, and a Xilinx Virtex 4 device has embedded DSP units with 18-bit multipliers and 48-bit accumulators. The flexibility of these blocks is limited and it is less common to build a digital system solely using these blocks. When the blocks are not used, they consume die area without adding to functionality.

FPGA fabric can have different levels of granularity. In general, a unit of smaller granularity has more flexibility, but can be less effective in speed, area, and power consumption. Fabrics with different granularity can coexist as evident in many commercial FPGA devices. Most importantly, the aforementioned examples illustrate that FPGA architectures are evolving to be more coarse-grained and application-specific. The proposed architecture in this paper follows this trend, focusing on floating-point computations.

III. FPFPGA Architecture

A. Requirements

Before we introduce the FPFPGA architecture, common characteristics of what we consider a reasonably large class of floating-point applications that 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 floating-point application circuits can be divided into control and datapath portions. The datapath typically contains floating-point operators such as adders, subtractors, and 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 optimized for floating-point computations, and for this reason, floating-point 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 [14].

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 unidirectional 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 aforementioned analysis, some basic requirements for FPFPGA architectures can be derived as follows.

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.

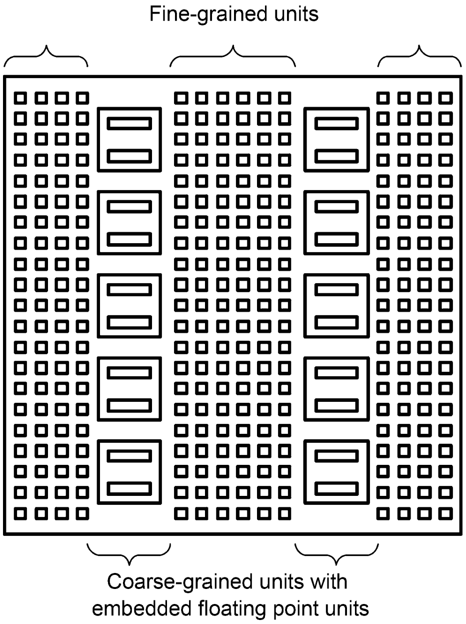


Fig. 1. Architecture of the FPFPGA.

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.

B. Architecture

Fig. 1 shows a top-level block diagram of our FPFPGA architecture. It employs an island-style fine-grained FPGA structure 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 multiplier, DSP block or block RAM) and fine-grained units in existing FPGA devices.

The coarse-grained logic architecture is optimized to implement the datapath portion of floating-point applications. The architecture of each block, inspired by previous work [4], [12], is shown in Fig. 2. Each block consists of a set of floating-point multipliers, adder/subtractors, and general-purpose bitblocks connected using a unidirectional bus-based interconnect architecture. Each of these blocks will be discussed in this section. To keep our discussion general, we have parameterized the architecture, as shown in Table I. There are D subblocks in each coarse-grained block. P of these D subblocks are floating-point multipliers, another P of them are floating-point adders, and the rest $(D-2P)$ are general-purpose wordblocks. Specific values of these parameters will be given in Section VI.

The core of each coarse-grained block contains P multiplier and P 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 D identical bitblocks, and is similar to our earlier published design [12]. 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 a register, can efficiently implement operations such as fixed-point addition and multiplexing. Like the multiplier and adder/subtractor blocks, wordblocks generate status flags such as MSB, LSB, carry out, overflow, and zero; these signals can be connected to the fine-grained units.

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 utilizing the feedback registers. The bus width of the coarse-grained units is 32 bits for the single-precision FPFPGA and 64 bits 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.

IV. Example Mapping

To illustrate how our architecture can be used to implement a datapath, we use the example of a floating-point matrix multiply. Fig. 3 illustrates the example datapath and the implementation of this datapath on our architecture. In this example, we assume an architecture in which the multiplication subblocks are located in the second and sixth subblocks within the architecture, and floating-point adder/subtractor units are located in the third and the seventh subblocks.

The datapath of this example application can be implemented using two coarse-grained blocks. The datapath produces the result of the equation $d0 \times d2 + d1 \times d3 + d4 \times d5$. The first coarse-grained unit performs two multiplications and one addition. The result $(r1)$ is forwarded to the next coarse-grained unit. The second coarse-grained unit performs one multiplication and one addition. However, as all multiplications start in the same clock cycle, the last addition cannot start until $r1$ is ready. In order to synchronize the arrival time of $r1$ and $d4 \times d5$, another floating-point adder (FA2) in the second coarse-grained block is instantiated as a first-input, first-output (FIFO) with the same latency as FA6 in CGU0. This demonstrates an alternate

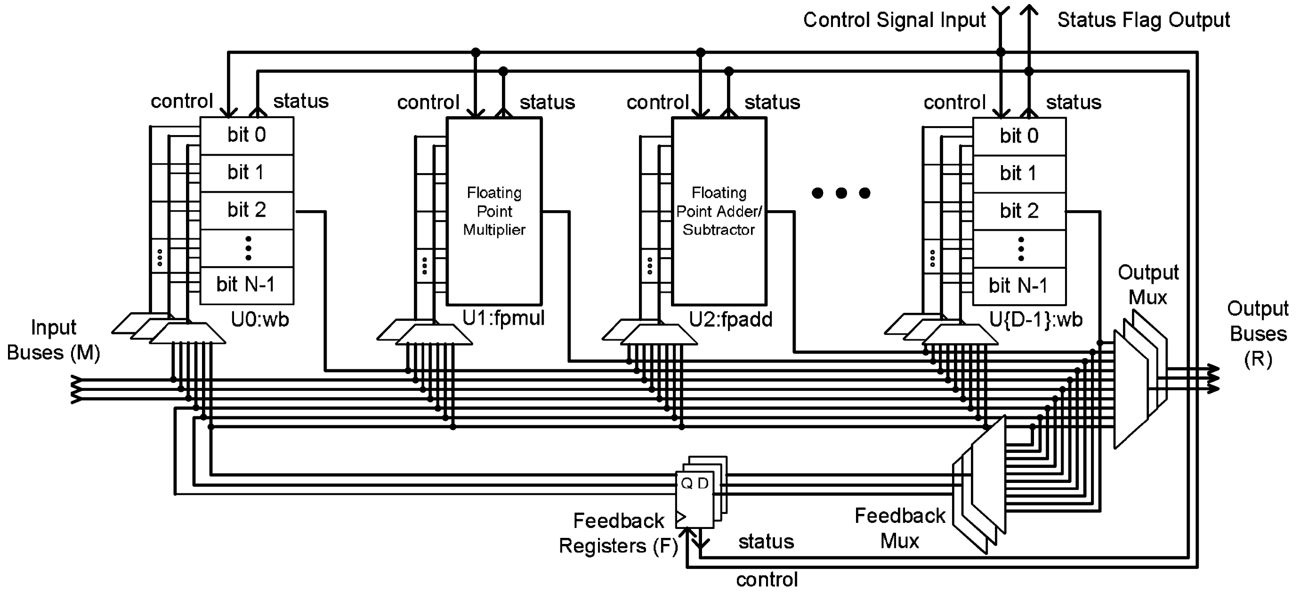
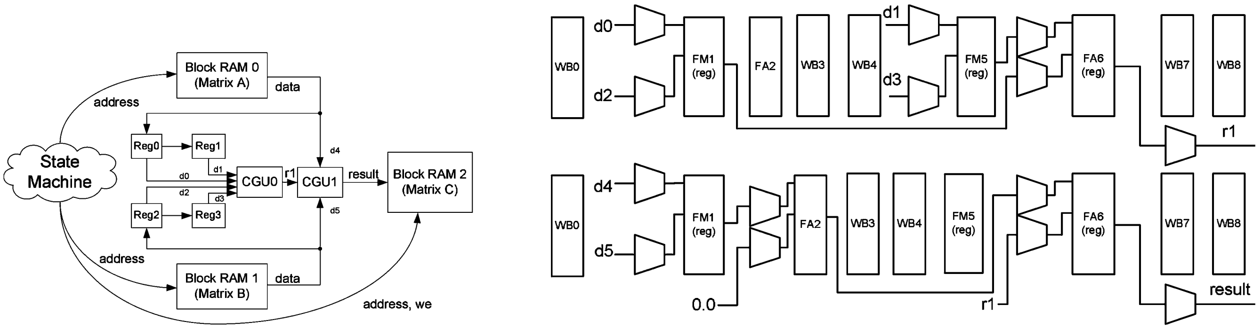


Fig. 2. Architecture of the coarse-grained unit.



(a) Fine-grained unit mapping.

(b) Coarse-grained unit mapping.

Fig. 3. Example mapping for matrix multiplication.

TABLE I
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	Number of Floating Point Adders and Multipliers

use of a coarse-grained unit. Finally, $r1$ and $d4 \times d5$ are added together and the state machine sends the result to the block RAM. All FPU subblocks have an enabled registered output to further pipeline the datapath.

V. FPFPGA COMPILATION

While traditional HDL design flow can be used in translating applications to our FPFPGA, the procedure is tedious and the designers have to fully understand the usage of the coarse-grained units in order to manually map the circuit effectively. A domain-specific hardware compiler, which can map a subset of a high-level language to the proposed architecture, is useful in developing applications on such as an FPFPGA. In addition, the hardware compiler is beneficial during the development of the FPFPGA itself since the compiler can be used to generate benchmark circuits. Although we have not implemented such a compiler, this section proposes the basic requirements of the compiler and discusses how some of the design challenges can be addressed.

The basic requirements of the FPFPGA compiler are as follows.

1) The compiler should contain a set of predefined built-in functions that represent the functionality in the coarse-grained unit. For example, the compiler can provide floating-point functions such as fadd(), fmul() (or even better, overloaded operators such as “+” or “*”) that associate with the floating operators in the coarse-grained unit. This feature allows application designers to infer the coarse-grained units easily.
2) It should have the ability to differentiate the control logic and the datapath. This feature would allow the technology mapper to handle the control logic and the datapath separately. Since the control logic can be efficiently implemented using the fine-grained logic, a standard hardware

compilation technique such as [15] can be used. The datapath, which is usually much more complicated, can be mapped to coarse-grained units whenever it is possible.

3) The compiler should contain a parametrizable technology mapper for the coarse-grained architecture. Since this is parametrized for design exploration, the technology mapper should map to devices with differing amounts of coarse-grained resources. For example, the technology mapper should be aware of the number of floating-point operator in a coarse-grained unit so it can fully utilize all the operators in a unit. This feature would allow FPGA designers to evaluate new architectures effectively by compiling benchmark circuits with modified architectural parameters.
4) The compiler should contain an intelligent resource allocation algorithm. It should be aware of the functionality of the coarse-grained unit and decide if the given operation is best implemented by coarse-grained units or fine-grained units. For example, if the compiler receives a "square root" instruction but there is no square root function in the coarse-grained units, the allocation algorithm can infer a square root operator using fine-grained unit instead.
5) Support is required for bitstream generation for coarse-grained units. Such a feature is necessary to determine the delay of a mapped coarse-grained unit.

Requirements 1, 4, and 5 have been studied in other contexts [16], and requirement 2 has been addressed in [17] in which the authors propose a compiler that can produce separate circuits for control logic and datapath for floating-point applications. Requirement 3 is new, and is specific for our architecture. One approach to creating this tool would be to develop a dedicated technology mapper for the coarse-grained units within the Trident framework [17]. A bitstream generator for coarse-grained units can be integrated into the framework as well. This is ongoing work.

VI. Modeling Methodology

In this section, we describe the methodology we use to model our architecture. We employ an experimental approach and use the concept of VEBs to model the embedded coarse-grained blocks. The following sections first describe the benchmark circuits we used, followed by a description of the VEB methodology.

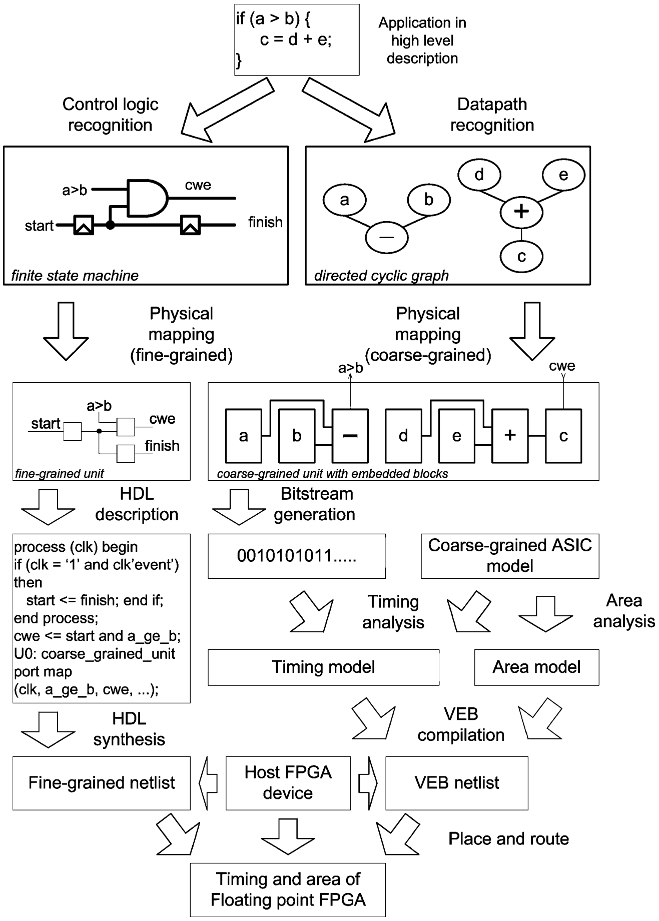
A. Benchmark Circuits

Eight benchmark circuits are used in this study, as shown in Table II. Five are computational kernels, one is a Monte Carlo simulation datapath, and two are synthetic circuits. All benchmark circuits involve single-precision floating operations. We choose these circuits since they are representative of the applications we envision being used on an FPFPGA. We note that the strong representation of simple floating-point kernels that map directly to the CGU favorably influences the overall density and performance metrics, so our results can be considered an upper bound. Dependencies, mapping, control, and interfacing are issues likely to degrade performance.

The *bfly* benchmark performs the computation $z = y + x * w$ where the inputs and output are complex numbers; this is commonly used within a fast Fourier transform computation. The *dscg* circuit is the datapath of a digital sine--cosine generator. The *fir* circuit is a four-tap finite-impulse response filter. The *mm3* circuit performs a 3×3 matrix multiplication. The *ode* circuit solves an ordinary differential equation. The *bgm* circuit computes Monte Carlo simulations of interest rate model derivatives priced under the Brace, Gatarek, and Musiela (BGM) framework [18]. All the word lengths of the aforementioned circuits are 32 bit.

TABLE II
BENCHMARK CIRCUITS

Circuit	# of Add/Sub	# of Mul	Domain	Nature
bfly	4	4	DSP	kernel
dscg	2	4	DSP	kernel
fir	3	4	DSP	kernel
mm3	2	3	Linear Algebra	kernel
ode	3	2	Linear Algebra	kernel
bgm	9	11	Finance	application
syn2	5	4	N/A	synthetic
syn7	25	25	N/A	synthetic



```
if (a > b) {
    c = d + e;
}
```

```
process (clk) begin
if (clk = '1' and clk'event')
then
  start <= finish; end if;
end process;
cwe <= start and a_ge_b;
U0: coarse_grained_unit
port map
(clk, a_ge_b, cwe, ...);
```

Fig. 4. Modeling flow overview.

In addition, a synthetic benchmark circuit generator based on [19] is used. The generator can produce floating-point circuits from a characterization file describing circuit and cluster statistics. Two synthetic benchmark circuits are produced. Circuit *syn2* contains five floating-point adders and four floating-point multipliers. Circuit *syn7* contains 25 floating-point adders and

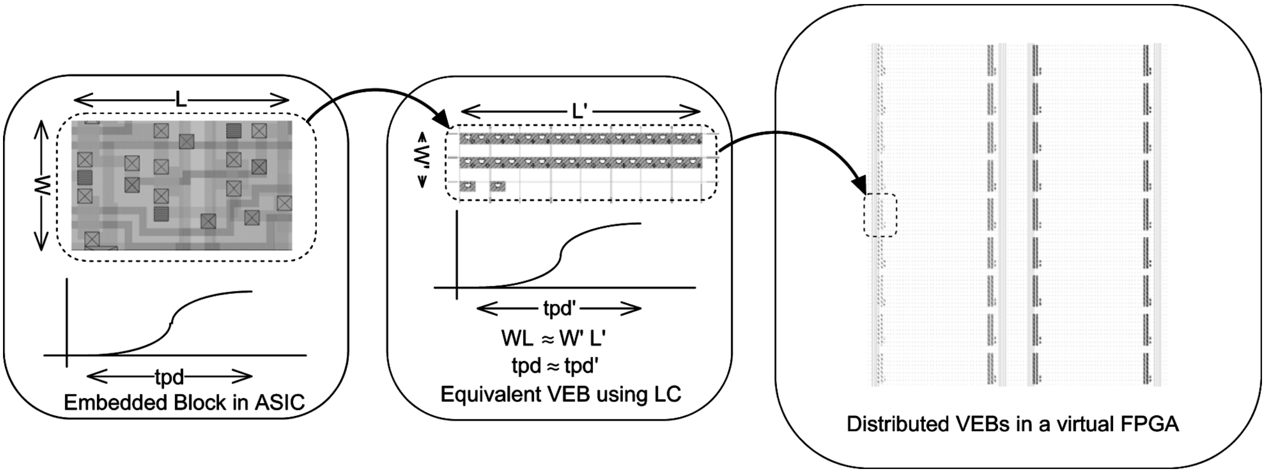


Fig. 5. Modeling coarse-grained unit in FPGAs using VEBs.

25 floating-point multipliers. The *syn7* circuit is considerably larger than the other benchmarks.

B. VEB Methodology

To model the mapping of our benchmark circuits on the architecture described in Section III, we employ the VEB methodology. This methodology allows us to quantify the impact of embedding our block into a modern FPGA using commercial CAD tool optimizations. This is in contrast to VPR-based methodologies that assume a bare-bone island-style FPGA (without carry chains and with a simplified routing architecture) and do not employ modern optimizations such as physical synthesis and retiming.

Fig. 4 illustrates the modeling 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. Since we do not yet have a complete implementation of a suitable compiler, we perform this step manually.

The datapath portion is then mapped to the embedded floating-point blocks (again, this is currently done manually). An example of this mapping was given in Section IV. 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 LUTs in the cases where no suitable embedded block is found or all have been used.

Unfortunately, this netlist cannot be implemented directly using commercial FPGA CAD tools, since the corresponding commercial FPGAs do not contain our floating-point embedded blocks. The basic strategy in our 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 Fig. 5.

To employ this methodology, area and delay models for the coarse-grained units are required. To estimate the area, we synthesize an ASIC description of each coarse-grained block using a comparable technology. For instance, 0.13-μm technology is used in synthesizing the ASIC block embedded in a Virtex II device, which, in turn, uses a 0.15-μm/0.12-μm process. Normalization to the feature size is then applied to obtain a more accurate area estimation. We employ a parameterized synthesizable IEEE 754 compliant floating-point library [20]. The library supports four rounding modes and denormalized numbers. A floating-point multiplier and floating-point adder are generated and synthesized using a regular standard cell library flow.

The area of the coarse-grained block is then translated into equivalent LC resources in the virtual FPGA. In order to make this translation, an estimate of the area of an LC in the FPGA is required, where an LC refers to a four-input LUT and an associated output register. The area estimation includes the associated routing resources and configuration bits. All area measures are normalized by dividing the actual area by the square of the feature size, making them independent of feature size. VEB utilization can then be computed as the normalized area of the coarse-grained unit divided by the normalized area of an LC. This value is in units of equivalent LCs, and the mapping enables modeling of coarse-grained units using existing FPGA resources. In addition, special consideration is given to the interface between the LCs and the VEB to ensure that the corresponding VEBs have sufficient I/O pins to connect to the routing resources. This can be verified by keeping track of the number of inputs and outputs that connect to the global routing resources in an LC. For example, if an LC only has two outputs, it is not possible to have a VEB with an area of four LCs that requires nine outputs. For such a case, the area is increased to five LCs.

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 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, we assume 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.

In our implementation, area matching is achieved by creating a dedicated scan chain using shift registers. A longer scan chain consumes more LC, and therefore, the VEB is larger. There are

TABLE III
NORMALIZATION ON THE AREA OF THE COARSE-GRAINED UNITS AGAINST A VIRTEX II LC

Fabric	Area (A) (μm^2)	Feature Size (L) (μm)	Normalised Area (A/L^2)	Area in LC	Input Pin	Output Pin
Virtex II LC	5,456	0.15	242,489	1	4(4)	2(2)
SP-CGU	498,847	0.13	30,203,964	122	157 (488)	162(244)
DP-CGU	1,025,624	0.13	60,687,797	251	285 (1004)	258(502)

SP and DP stand for single precision and double precision, respectively. CGU stands for coarse-grained unit. For the values shown in the second column (area), 15% overheads have already been applied on the coarse-grained units.

many options available to match the timing of a VEB. We utilize 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 LCs on the FPGA does not affect the timing of this circuit.

It should also be noted that the use of the carry and scan chains allows delay and area to be varied independently. Modeling wiring delays is more problematic, since the placement of the virtual FPGA must be similar to that of an FPGA with coarse-grained units to ensure that their routing is similar. This requires that: 1) the absolute location of VEBs matches the intended locations of real embedded blocks in the FPGA with coarse-grained units and 2) the design tools are able to assign instantiations of VEBs in the netlist to physical VEBs while minimizing routing delays.

The first requirement is addressed by locating VEBs at predefined absolute locations that matches the floorplan of the FPGA with coarse-grained units. To address requirement 2), the assignment of physical VEBs is currently made by a two-phase placement strategy that consists of unconstrained placement followed by manual placement. We first assume that the VEB can be placed anywhere on the virtual FPGA so that the place and route tools can determine the most suitable location for each VEB. Once the optimal VEB locations are known, a manual placement is applied to ensure that the placement of each VEB is aligned on dedicated columns while maintaining nearest displacement to the optimal location. We believe that this strategy can provide a reasonable placement as the location of each VEB is derived from the optimal placement.

There are inevitable differences between real implementations and the VEB emulated ones. In our previous work [10], we compared an actual embedded multiplier with one modeled using the VEB method. It was found that timing difference can be as large as 11% while the area is accurately determined. We believe such errors are acceptable for the first-order estimations desired. Once a suitable coarse-grained unit architecture is identified, a more in-depth analysis using lower level methods such as SPICE simulation can be performed to confirm the results.

To instantiate all the VEBs and connect all together, we describe the control logic and instantiate the VEBs explicitly and connect the signals between the fine-grained units and coarse-grained units. The design is then synthesized 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, a two-phase placement is used to locate near-optimal placement of VEBs along dedicated columns. We then use the vendor's place and route tool to obtain the final area and timing results. This represents the characterization of a circuit implemented on the FPFPGA with fine-grained units and routing resources exactly the same as the targeted FPGA.

It is important to note that timing information cannot be determined before programming the configuration bits. Otherwise, the tool reports the worst-case scenario where the longest combinational path from the first wordblock to the last wordblock is considered as critical path, and this is usually not the correct timing in most designs. To address this issue, the tool has to recognize the configuration of the coarse-grained units 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.

VII. RESULTS

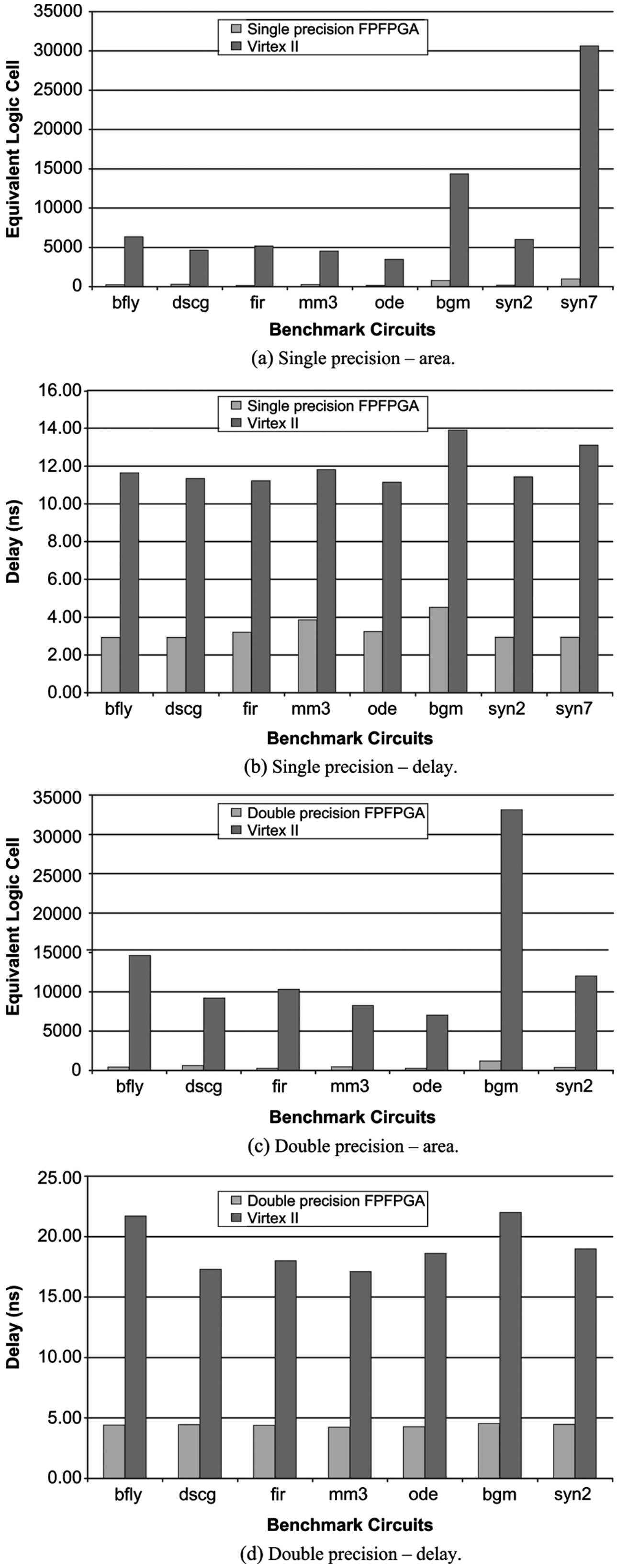
In this section, we present an evaluation of our architecture. The flow described in the previous section is employed.

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 each benchmark. A more in-depth analysis on how these parameters affect the application performance is ongoing 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 emphasize 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 ($P = 2$). We assume that the two floating-point multipliers in the coarse-grained unit are located at the second and the sixth subblocks. The two floating-point adders are located in the third and the seventh subblocks.

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-2006.06.

The physical die area and photomicrograph of a Virtex II device has been reported [21], and the normalization of the area of coarse-grained unit is estimated in Table III. From inspection of the die photograph, we estimate that 60% of the total die area is used for LCs.



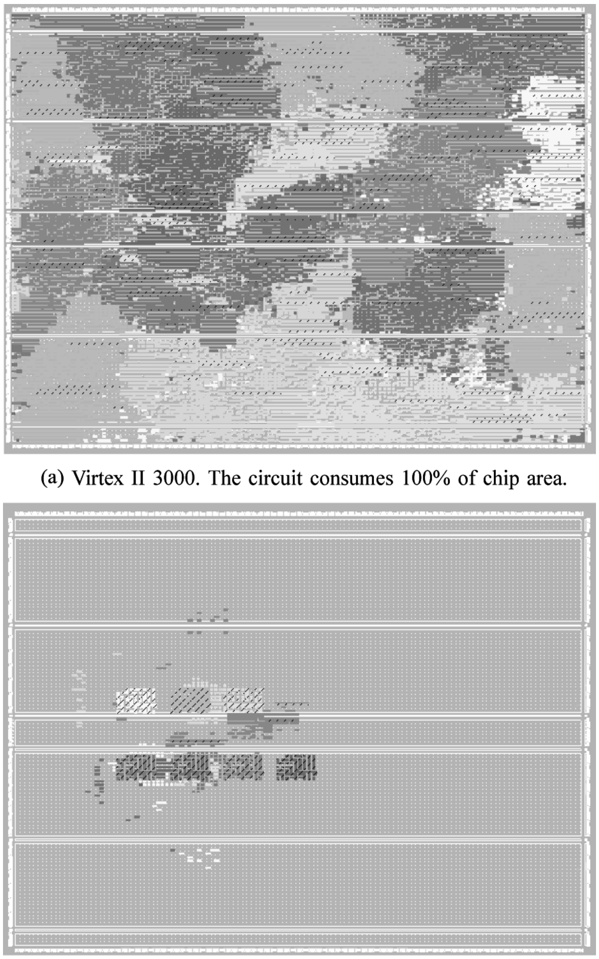
(a) Single precision – area.

(b) Single precision – delay.

(c) Double precision – area.

(d) Double precision – delay.

Fig. 6. Comparisons of FPFPGA and Xilinx Virtex II FPGA device.



(a) Virtex II 3000. The circuit consumes 100% of chip area.

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

Fig. 7. Floorplan of the single-precision *bgm* circuit on Virtex II FPGA and FPFPGA. Area is significantly reduced by introducing coarse-grained units.

This means that the area of a Virtex II LC is 5456 μm^2. This number is normalized against the feature size (0.15 μm). 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 433 780 μm^2. We further assume 15% overhead after place and route the design based on our experience [12]. The area values are normalized against the feature size (0.13 μm). The number of equivalent LC is obtained through the division of coarse-grained unit area by slice area. This shows that single-precision coarse-grained unit is equivalent to 122 LCs. Assuming each LC has two outputs, the VEB allows a maximum of 244 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 IV(a) and Fig. 6(a) and (b). A comparison between the floorplan of the Virtex II device and the floorplan of the FPFPGA on *bgm* circuit is illustrated in Fig. 7.

The FPU implementation on FPGA is based on the work in [22]. This implementation supports denormalized floating-point numbers that are required in the comparison with the FPFPGA. The FPU area for the XC2V3000 device [seventh column of Table IV(a)] 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

TABLE IV
FPFPGA Implementation Results

	Single precision FPFPGA					XC2V3000-6-FF1152				Reduction	
Circuit	number of CGU	CGU area (LC)	FGU area (LC)	Total Area (LC)	Delay (ns)	FPU area (LC)	Logic area (LC)	Total Area (LC)	Delay (ns)	Area (times)	Delay (times)
bfly	2	244 (0.9%)	212 (0.74%)	456 (1.6%)	2.92	11,678 (41%)	988 (3.4%)	12,666 (44%)	11.6	27.8	3.99
dscg	2	244 (0.9%)	352 (1.23%)	596 (2.1%)	2.92	8,838 (31%)	406 (1.4%)	9,244 (32%)	11.3	15.5	3.88
fir	2	244 (0.9%)	14 (0.05%)	258 (0.9%)	3.20	10,118 (35%)	218 (0.8%)	10,336 (36%)	11.2	40.1	3.51
mm3	2	244 (0.9%)	268 (0.93%)	512 (1.8%)	3.86	8,004 (28%)	1,010 (3.5%)	9,014 (31%)	11.8	17.6	3.06
ode	2	244 (0.9%)	38 (0.13%)	282 (1.0%)	3.24	6,658 (23%)	282 (1.0%)	6,942 (24%)	11.1	24.6	3.44
bgm	7	854 (3.0%)	646 (2.25%)	1,500 (5.2%)	4.52	27,856 (97%)	812 (2.8%)	28,668 (100%)	13.9	19.1	3.08
syn2	3	366 (1.3%)	0 (0.0%)	366 (1.3%)	2.93	11,966 (42%)	0 (0.0%)	11,966 (42%)	11.4	32.7	3.90
syn7*	16	1,952 (6.8%)	0 (0.0%)	1,952 (6.8%)	2.93	61,250 (214%)	0 (0.0%)	61,250 (214%)	13.1	31.4	4.47
Geometric Mean:										24.9	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 (LC)	FGU area (LC)	Total Area (LC)	Delay (ns)	FPU area (LC)	Logic area (LC)	Total Area (LC)	Delay (ns)	Area (times)	Delay (times)
bfly	2	504 (0.7%)	402 (0.74%)	906 (1.3%)	4.42	27,306 (40%)	1,926 (2.9%)	29,232 (43%)	21.7	32.3	4.91
dscg	2	504 (0.7%)	726 (1.07%)	1,230 (1.8%)	4.45	17,968 (27%)	404 (0.6%)	18,372 (27%)	17.3	14.9	3.89
fir	2	504 (0.7%)	12 (0.02%)	516 (0.8%)	4.38	20,290 (30%)	330 (0.5%)	20,620 (31%)	18.0	40.0	4.11
mm3	2	504 (0.7%)	458 (0.68%)	962 (1.4%)	4.25	15,058 (22%)	1,454 (2.2%)	16,512 (24%)	17.1	17.2	4.03
ode	2	504 (0.7%)	44 (0.07%)	548 (0.8%)	4.27	13,588 (20%)	478 (0.7%)	14,066 (21%)	18.6	25.7	4.35
bgm	7	1,764 (2.6%)	642 (0.95%)	2406 (1.0%)	4.55	65,836 (97%)	398 (0.6%)	66,234 (98%)	22.0	27.5	4.84
syn2	3	756 (1.1%)	0 (0%)	756 (1.1%)	4.47	24,032 (36%)	0 (0%)	24,032 (36%)	19.0	31.8	4.26
Geometric Mean:										25.7	4.33

(b) Double precision FPFPGA results. Circuit *syn7* is omitted since it cannot be fitted on any Virtex II FPGA device.

Values in the brackets indicate the percentages of LC used in corresponding FPGA device. CGU stands for coarse-grained unit and FGU stands for fine-grained unit.

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 FPUs that consume 214% of the total FPGA area. They can fit into 16 coarse-grained units, which constitute just 6.8% of the total FPGA area.

Similar experiments for double-precision floating-point applications have been conducted, and the results are reported in Table IV(b) and Fig. 6(c) and (d). 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 25 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 improvement possible if the architecture is well matched to the target applications. In essence, our architecture stands between ASIC and FPGA implementation. The authors in [2] 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 1.4 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 FPUs; thus, we would expect a ratio similar to that between normal FPGA and ASIC circuitry. Our results are consistent with [2] that suggests the ratio is between 3 and 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.

VIII. Conclusion

We propose an FPFPGA architecture that involves a combination of reconfigurable fine-grained and reconfigurable coarse-grained units optimized for floating-point computations. A parameterizable description is presented that 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. Using this approach, we show that the proposed FPFPGA enjoys improved speed and density over a conventional FPGA for floating-point intensive applications. The area can be reduced by 25 times and the frequency is increased by four times on average when comparing the proposed architecture with an existing commercial FPGA device. Current and future work includes developing automated design tools supporting facilities such as partitioning for coarse-grained units, and exploring further architectural customizations for a large number of domain-specific applications.

References

[1] *Architecture and CAD for Deep-Submicron FPGAs*, V. Betz, J. Rose, and A. Marquardt, Eds.. Norwell, MA: Kluwer, 1999.

[2] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 26, no. 2, pp. 203–215, Feb. 2007.

[3] K. Compton and S. Hauck, "Totem: Custom reconfigurable array generation," in *Proc. FCCM*, 2001, pp. 111–119.

[4] A. Ye and J. Rose, "Using bus-based connections to improve field-programmable gate-array density for implementing datapath circuits," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 5, pp. 462–473, May 2006.

[5] E. Roesler and B. Nelson, "Novel optimizations for hardware floating-point units in a modern FPGA architecture," in *Proc. FPL*, 2002, pp. 637–646.

[6] M. J. Beauchamp, S. Hauck, K. D. Underwood, and K. S. Hemmert, "Architectural modifications to enhance the floating-point performance of FPGAs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 2, pp. 177–187, Feb. 2008.

[7] K. Leijten-Nowak and J. L. van Meerbergen, "An FPGA architecture with enhanced datapath functionality," in *Proc. FPGA*, 2003, pp. 195–204.

[8] A. Ye, J. Rose, and D. Lewis, "Architecture of datapath-oriented coarsegrain logic and routing for FPGAs," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, 2003, pp. 61–64.

[9] L. Beck*, A Place-and-Route Tool for Heterogeneous FPGAs. Distributed Mentor Project Report*. Ithaca, NY: Cornell Univ. Press, 2004.

[10] C. Ho, P. Leong, W. Luk, S. Wilton, and S. Lopez-Buedo, "Virtual embedded blocks: A methodology for evaluating embedded elements in FPGAs," in *Proc. FCCM*, 2006, pp. 35–44.

[11] C. Ho, C. Yu, P. Leong, W. Luk, and S. Wilton, "Domain-specific FPGA: Architecture and floating point applications," in *Proc. FPL*, 2007, pp. 196–201.

[12] S. Wilton, C. Ho, P. Leong, W. Luk, and B. Quinton, "A synthesizable datapath-oriented embedded FPGA fabric," in *Proc. FPGA*, 2007, pp. 33–41.

[13] E. Ahmed and J. Rose, "The effect of LUT and cluster size on deep-submicron FPGA performance and density," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 12, no. 3, pp. 288–298, Mar. 2004.

[14] Xilinx, Inc., San Jose, CA, "Floating-point operator v3.0. Product specification," 2005.

[15] I. Page and W. Luk*, Compiling Occam Into FPGAs*. Oxford, U.K.: Abingdon EE&CS Books, 1991, pp. 271–283.

[16] Agility Design Solution, Inc., Palo Alto, CA, "Software product description for DK design suite Version 5.0," 2008.

[17] J. Tripp, M. Gokhale, and K. Peterson, "Trident: From high-level language to hardware circuitry," *Computer*, vol. 40, no. 3, pp. 28–37, Mar. 2007.

[18] G. Zhang, P. Leong, C. H. Ho, K. H. Tsoi, C. Cheung, D.-U. Lee, R. Cheung, and W. Luk, "Reconfigurable acceleration for Monte Carlo based financial simulation," in *Proc. ICFPT*, 2005, pp. 215–222.

[19] P. D. Kundarewich and J. Rose, "Synthetic circuit generation using clustering and iteration," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 23, no. 6, pp. 869–887, Jun. 2004.

[20] Synopsys, Inc., Mountain View, CA, "DesignWare building block IP, Datapath—Floating point overview," 2007.

[21] C. Yui, G. Swift, and C. Carmichael, "Single event upset susceptibility testing of the Xilinx Virtex II FPGA," presented at the Military Aerosp. Appl. Program. Logic Conf. (MAPLD), Laurel, MD, 2002.

[22] R. Usselmann, "Open floating point unit," 2005. [Online]. Available: http://www.opencores.org/project.cgi/web/fpu/overview

Chun Hok Ho (S'06) received the B.Eng. degree (Honors) in computer engineering and the M.Phil. degree in computer science and engineering from the Chinese University of Hong Kong, Hong Kong, in 2001 and 2003, respectively. He is currently pursuing the Ph.D. degree in the Custom Computing Group, Department of Computing, Imperial College, London, U.K.

His research interests include computer arithmetic, computer architecture, design automation, and optimization.

Mr. Ho was a recipient of several awards, such as Oversea Research Students Awards, and the Stamatis Vassiliadis Awards from International Conference on Field Programmable Logic and Applications in 2007 and 2008.

Chi Wai Yu received the B.Eng. degree from the Chinese University of Hong Kong, Hong Kong. He is currently pursuing the Ph.D. degree in Department of computing, Imperial College, London, U.K.

His research interest is in FPGA architecture.

Philip Leong (M'84–SM'02) received the B.Sc., B.E., and Ph.D. degrees from the University of Sydney, Sydney, Australia, in 1986, 1988, and 1993, respectively.

In 1993, he was a consultant to ST Microelectronics, Milan, Italy, where he worked on advanced flash memory-based integrated circuit design. From 1994-1997, he was a Lecturer with the University of Sydney. Since 1997, he has been with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong, where he is a Professor and the director of the Custom Computing Laboratory. He is also Visiting Professor at Imperial College, London, and the Chief Technology Consultant to Cluster Technology. His research interests include reconfigurable computing, signal processing, computer architecture, computer arithmetic and biologically inspired computing.

Dr. Leong was a recipient of the 2005 FPT Conference Best Paper as well as the 2007 and 2008 FPL conference Stamatis Vassiliadis Outstanding Paper Awards. He was program co-chair of the FPT and FPL conferences and is an Associate Editor for the *ACM Transactions on Reconfigurable Technology and Systems*. He is the author of more than 100 technical papers and 4 patents.

Wayne Luk (F'09) received the M.A., M.Sc., and D.Phil. degrees in engineering and computing science from the University of Oxford, Oxford, U.K.

He is a Professor of computer engineering with the Department of Computing, Imperial College London, London, U.K., and a Visiting Professor with Stanford University, Stanford, CA, and with Queen's University Belfast, Belfast, U.K. His research interests include theory and practice of customizing hardware and software for specific application domains, such as multimedia, communications, and finance. Much of his current work involves high-level compilation techniques and tools for parallel computers and embedded systems, particularly those containing reconfigurable devices such as field-programmable gate arrays.

Steven J. E. Wilton (S'86–M'97–SM'03) received the M.A.Sc. and Ph.D. degrees in electrical and computer engineering from the University of Toronto, Toronto, ON, Canada, in 1992 and 1997, respectively.

In 1997, he joined the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC, Canada, where he is now an Associate Professor. During 2003 and 2004, he was a Visiting Professor with the Department of Computing, Imperial College, London, U.K., and at the Interuniversity MicroElectronics Center (IMEC), Leuven, Belgium. He has also served as a consultant for Cypress Semiconductor and Altera Corporation. His research focuses on the architecture of FPGAs and the CAD tools that target these devices.

Dr. Wilton was a recipient of Best Paper Awards at the International Conference on Field-Programmable Technology in 2003, 2005, and 2007, and at the International Conference on Field-Programmable Logic and Applications in 2001, 2004, 2007, and 2008. In 1998, he won the Douglas Colton Medal for Research Excellence for his research into FPGA memory architectures. In 2005, he was the Program Chair for the ACM International Symposium on Field-Programmable Gate Arrays and the program co-chair for the International Conference on Field Programmable Logic and Applications. In 2007, he was the Program Co-Chair for the IEEE International Conference on Application-Specific Systems, Architectures, and Processors.