

# ACCURATE AREA, TIME AND POWER MODELS FOR FPGA-BASED IMPLEMENTATIONS<sup>†</sup>

*Lanping Deng, Kanwaldeep Sobti, Yuanrui Zhang\*, Chaitali Chakrabarti*

Department of Electrical Engineering, Arizona State University

\*Department of Computer Science and Engineering, Pennsylvania State University  
email: {ldeng2, ksobti, chaitali}@asu.edu, yuz123@psu.edu

## ABSTRACT

This paper presents accurate area, time, power estimation models for implementations using FPGAs from the Xilinx Virtex-2Pro family [1]. These models are designed to facilitate efficient design space exploration in an automated algorithm-architecture codesign framework. Detailed models for estimating the number of slices, block RAMs and 18x18-bit multipliers for fixed point and floating point IP cores have been developed. These models are also utilized to develop power models that consider the effect of logic power, signal power, clock power and I/O power. Timing models have been developed to predict the latency of the fixed point and floating point IP cores. In all cases, the model coefficients have been derived by using curve fitting or regression analysis. The modeling error is quite small for single IP cores; the error for the area estimate, for instance, is on the average 0.95%. The error for fairly large examples such as floating point implementation of 8-point FFTs is also quite small; it is 1.87% for estimation of number of slices and 3.48% for estimation of power consumption. The proposed models have also been integrated into a hardware-software partitioning tool to facilitate design space exploration under area and time constraints.

**Keywords:** FPGA, estimators for area, time, power, IP core, regression analysis, design space exploration

## I. INTRODUCTION

Reconfigurable hardware, especially field programmable gate arrays (FPGA), are widely used for rapid prototyping of digital signal processing (DSP) systems. Since FPGA based architectures can be reconfigured according to user-specified design parameters, they are great candidates for achieving high performance at low cost. Recent FPGA platforms have millions of gates, up to 500MHz clock frequency, reasonably large on-chip memory and fast I/O interface. As a result, they provide an easy and cost-effective way to evaluate DSP algorithms from an implementation perspective. However, DSP algorithm developers still prefer to use high-level languages like C or MATLAB [2] to prototype and test their algorithms. One way of bridging the gap is to provide an easy mechanism for translating the high level algorithmic description onto an FPGA platform using parameterizable Intellectual Property (IP) cores [3], [4], [5], [6].

IP core based designs have greatly reduced the time and effort of hardware development. However, these designs cannot always satisfy the area-time-power constraints in a short design cycle. An example is the feedback-driven search algorithm for parameterized IP core based design for automatic design space exploration and optimization proposed in [7]. The main challenge here is the time for area, time and power evaluation. Existing commercial tools from Xilinx and Synopsys can report accurate resource numbers but only after going through the Synthesis, Placement and Routing (P&R) flow, which can take significant amount of time. For instance, a medium sized design with 50% slice utilization can take 30-40 minutes while a large design with 80% slice utilization can easily take an hour. Clearly, there is a need to develop fast area, time and power estimates.

There are several estimation tools for FPGA based implementations. High-level area and delay estimates are directly obtained from MATLAB descriptions in [8], [9]. Both of them extract register and functional unit usage of MATLAB descriptions to generate the estimates. [8] provides area models to estimate the maximum number of Configurable Logic Blocks (CLBs) and delay models that consider the operator delay. The technique in [9] is simulation-based, where a simulation trace containing register and functional unit usage is used to estimate the area and latency. Detailed area models have been developed in [10], [11]. [10] estimates FPGA resources by predicting the number of lookup tables (LUT) from the netlist.

<sup>†</sup> This paper is an extension of the ICASSP'08 paper "Accurate Models for Estimating Area and Power of FPGA Implementations".

It takes into account not only gate area and delay but also the wiring effects. But this approach requires that the design be synthesized to get the netlist and is more time consuming. The method in [11] develops area models for higher level design abstractions such as data-flow graph (DFG). It estimates the area based on the number of each type of operation and their average word-length. Unfortunately, the number of operations that are supported is quite limited and this method is applied to only regular algorithms in image processing.

Power models for FPGA implementation have been developed in [12], [13]. An RT-level power estimator which considers wire length capacitance and switching activity has been considered in [12]. These estimates are made even better in [13] by considering short-circuit power and leakage power. Comprehensive models for large parameterized IP generator based implementations have been presented in [14], [15]. While [14] presents function-level area and power models for fast Hadamard transform (FHT), [15] presents area models for fully customizable discrete Fourier transform (DFT). Thus both [14] and [15] are algorithm specific.

In this paper we provide fast and accurate estimates of area, time and power for any implementation based on Xilinx Virtex-2Pro FPGA. The models are integrated with the FANTOM tool [16], which is an algorithm-architecture co-design platform for generating fully pipelined hardware accelerators from user-provided MATLAB scripts. By adding the modeling aspect to the FANTOM tool, the system-level designer is provided with fairly accurate hardware-related estimates to guide hardware-software partitioning and even algorithm tuning. Also, our models have reduced the time for area and power estimation per design to several microseconds, thus making even exhaustive design space search feasible.

The proposed modeling work is quite general and is built for designs that utilize the parameterizable IP cores from Xilinx as well as custom-made kernels. Specifically, we develop area, time and power models for both fixed-point (FX) and floating-point (FP) IP cores for addition/subtraction, multiplication, square root and reciprocal operations from Xilinx, along with custom-made kernels such as 'round' and 'shift' operations. The models are obtained by curve fitting and linear regression. The area models of the IP cores are represented in terms of models for number of slices, number of block RAMs (BRAM) and number of 18x18-bit multipliers. In a large design that consists of multiple IP cores, the total number of slices is estimated by an empirical model that also takes into account slice utilization. The power consumption models are represented by those that are related to area such as logic power, clock and signal power, and those that are not related to area such as static power and I/O power. The timing models are represented by latency and expressed in number of clock cycles. Overall, the area, power and timing models are highly accurate. For instance, for designs with very high slice utilization, such as fully pipelined floating point implementation of 8-point FFTs, the error is only 1.87% for estimation of number of slices, 3.48% for estimation of power consumption and no-mismatch for time estimation. Finally, these models have been integrated into a hardware-software (HW-SW) partitioning method to support fast design space exploration.

The rest of the paper is as follows. Section II gives a brief introduction of the FANTOM tool and the Xilinx IP generator. Section III presents the detailed area and timing models for the IP cores, followed by models for system-level area and power consumption. Section IV validates the proposed models for lookup table based designs as well as FFT and DCT computations. Section V shows how these models can be used for efficient design space exploration. Section VI offers concluding remarks.

## II. EXISTING WORK

### II-A. FANTOM Tool

FANTOM is an algorithm architecture co-exploration platform for generating fully pipelined hardware accelerators from user-provided MATLAB descriptions [16]. Figure 1 shows the block diagram of the automated design flow in FANTOM. The input consists of a MATLAB description of the numerical algorithm, test data set, suitable optimization schemes, as well as constraints on accuracy, latency, area, and power. The output is an efficient FPGA based system in which the host computer supervises the FPGA based accelerator.

The first stage of the FANTOM flow is the code parser and analyzer (CPA) which transforms the input specifications into an intermediate representation known as ASG (Abstract Syntax Graph). The optimized ASG is then processed by the algorithm optimizer which applies optimization techniques related to table look up and data traversal. After these two stages, two different flows are adopted: one to generate hardware modules for the FPGA board and the other to generate software modules for the host computer. The hardware module implements the ASG operations using Xilinx IP cores and custom-made kernels, and downloads the hardware description language (HDL) code onto the FPGA platform. The software module generates the host program to supervise the operations in the FPGA platform. FANTOM provides an area-time-power estimation block inside the hardware module to facilitate early design space exploration. In the rest of the paper, we describe the area-time-power models and illustrate their use in design space exploration.

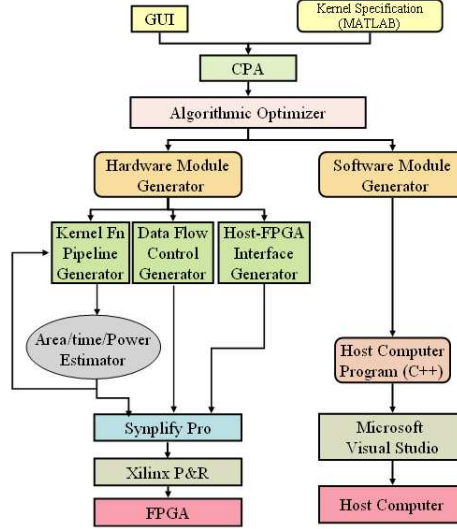


Fig. 1. Block Diagram of FANTOM Design Flow

## II-B. IP Core Implementations

The FANTOM tool makes extensive use of Intellectual Property (IP) cores provided by Xilinx [17]. Examples of Xilinx IP cores include arithmetic computation units (in both fixed point and floating point formats), industry standard UART controller and IIC (Inter-Integrated Circuit) interface. The IP cores are customized by (a) parameters that control the functionality, e.g., size of input and output vectors, precision of the input, output and intermediate data values, ordering of input and output vectors, and scaling scheme to avoid overflow, and (b) parameters that control the implementation choice, e.g., number of parallel modules, storage of intermediate data on block RAM (BRAM) macros or on distributed RAMs. The IP core generator produces synthesizable register transfer language (RTL)-level HDL descriptions of IP cores.

Table I lists the MATLAB operations and the corresponding IP cores. For example, the addition (+) operation is mapped to Xilinx LogiCORE Adder/Subtractor v7.0 [18]. In addition, there are custom-made kernels such as lookup table (LUT) based computation units, and functional units to implement the ‘round’ function and ‘shift’ function.

Table I. List of available components in IP core library.

Operation	Functional Unit
+/-	LogiCORE Adder/Subtractor v7.0
*	LogiCORE Multiplier v8.0
Reciprocal	LogiCORE Divider v1.0
Sqrt	LogiCORE Cordic v3.0
Floating Point Operator	LogiCORE Floating-Point Operators v3.0
Lookup table	Block RAMS
Round	Custom-made kernel
Shift	Custom-made kernel

## III. AREA, TIME, POWER MODELS

This section presents the proposed area, time and power models of Xilinx Virtex-2Pro FPGA based implementations. The IP cores that are currently supported are listed in Tables I, II and III.

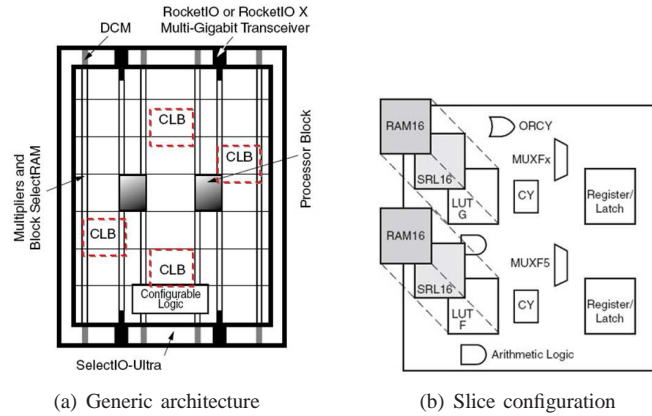
In order to develop the models, the IP cores are synthesized using Xilinx ISE 8.2i and Synplify Pro 8.6.2. In each case, at least 50 configurations with different number of bits are used. These configurations are synthesized followed by placement and routing, and curve fitting and non linear regression analysis [19] are used to derive the models. In addition, to develop the power models, ‘Xpower’ [20] is used, in which the clock frequency is set to 125MHz, activity factor to 12.5% and other parameters are set by default. For each IP core, the estimation error is given by:

$$abs. value\left(\frac{Synthesis\_result - Estimated\_result}{Synthesis\_result}\right)$$

### III-A. Area Models

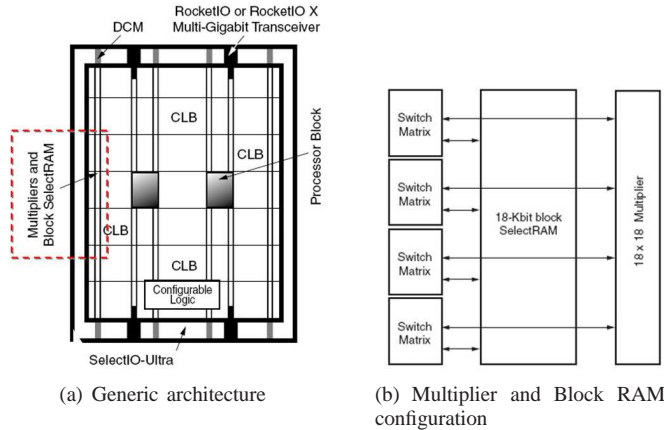
The models for estimating area of both fixed point and floating point IP cores are represented in terms of number of slices, number of block RAMs and number of  $18 \times 18$  multipliers.

The number of occupied slices is one of the most important metric for representing the area of a FPGA implementation. In Xilinx Virtex-2Pro family there may be as many as 44K slices. All the slices are packed in the Configurable Logic Blocks (CLB) and distributed all over the device, as shown in the red rectangles in Figure 2(a). Each slice consists of two function generators that can be configured as 4-input lookup tables, 16-bit shift registers or 16-bit distributed SelectRAM+ memory, and two storage elements that can be configured as D-flip flops or latches, carry logic, and arithmetic logic gates, as shown in Figure 2(b).



**Fig. 2.** (a) Distribution of CLBs in an FPGA architecture; (b) Virtex-2Pro FPGA slice configuration.

Virtex-2Pro FPGA also provides as many as 444 of 18K-bit Block RAM (BRAM) and  $18 \times 18$ -bit Multiplier (MULT18). Both BRAM and MULT18 are dedicated resources in the FPGA and their distribution is shown by the red rectangle region in Figure 3(a). A closer look shows that each Block RAM and  $18 \times 18$ -bit Multiplier is tied together with switch matrices, as shown in Figure 3(b), which means we can instantiate either the BRAM or the MULT18 but not both.



**Fig. 3.** (a) Distribution of BRAMs and Multipliers in an FPGA architecture; (b) Each Block RAM and  $18 \times 18$ -bit Multiplier is tied together with switch matrices.

The BRAM supports two types of configurations, the first type contains  $2K \times 9$ -bit,  $1K \times 18$ -bit and  $512 \times 36$ -bit configurations with access to all 18K-bit memory locations, and the second type contains  $16K \times 1$ -bit,  $8K \times 2$ -bit and  $4K \times 4$ -bit configurations with access to 16K-bit memory locations. In the actual implementations, multiple BRAM configurations can be grouped parallelly or sequentially to instantiate large memory.

In the rest of the paper, the following notations are used: *FX* for “fixed point format” with *I* for “integer bits” and *F* for “fraction bits”, and *FP* for “floating point format” with *E* for “exponent bits” and *M* for “mantissa bits”. For the Xilinx IP

cores, for fixed point,  $I$  and  $F$  can take on values from 0 to 64. For floating point, the range of bits for  $M$  depend on  $E$ . For instance, if  $E = 6$ , then  $M$  can be 4 to 29. In addition,  $in()$  and  $out()$  represent the number of input bits and output bits, respectively.

### III-A1. Fixed Point IP Cores

We have built models for IP cores and custom-made kernels operating in fixed point format. Table II lists the cores and the corresponding model parameters.

**Number of Slices:** Let  $Slice()$  denote the estimate of the number of slices in an IP core. We have derived expressions for estimating the number of slices for the cores listed in Table II. Some of these expressions are listed below.

$$Slice(add/sub\_FX) = 0.5 \cdot \max(in_1(I+F), in_2(I+F))$$

$$Slice(sqrt\_FX) = 0.56 \cdot [in(I) + 2.00 \cdot in(F)]^{1.8024} + 38.89$$

$$Slice(round\_FX) = 0.5 \cdot in(I)$$

$$Slice(recip\_FX) = 1.32 \cdot divisor(I+F) + 1.34 \cdot out(F) \cdot divisor(I+F)^{0.9397} + 3.55$$

The relationship between the number of slices and the input parameters is linear for Add/Sub, Round and Shift; the relationship is nonlinear for Square Root, Fixed-to-Float and Reciprocal. Note that the ‘shift’ operation is implemented using configurable flip-flops in a slice and does not consume any other resource. The minimum, maximum and average errors for the number of slices for fixed point IP cores are given in Table II. Note that the errors are very small for each individual IP core.

**Table II.** Fixed-point IP cores: Parameters and Errors in estimating the number of slices

FX cores	Model Parameters	Min.Err	Max.Err	Avg.Err
Add/Sub	$in_1(I,F), in_2(I,F)$	0.00%	0.00%	0.00%
Square Root	$in(I,F)$	0.00%	0.21%	0.10%
Round	$in(I)$	0.00%	0.00%	0.00%
Fixed-to-Float	$in(I,F)$	0.70%	4.88%	1.47%
Reciprocal	$divisor(I,F), out(F)$	0.87%	4.32%	1.58%
Shifter	$in(I,F), amount\ of\ shift$	0.00%	0.00%	0.00%

**Number of 18×18-bit Multipliers:** The 18×18-bit multiplier (MULT18) blocks on Virtex-2Pro FPGA are used automatically by the synthesis tool to implement multiplications. The MULT18 block supports two data input ports: 18-bit signed or 17-bit unsigned. In the FANTOM tool, we limit the configuration to be 17-bit unsigned. Then the number of MULT18 blocks to implement the product of  $in_1(I,F)$  and  $in_2(I,F)$  is given by

$$M_{18}(mult\_FX) = \lceil \frac{in_1(I+F)}{17} \rceil \cdot \lceil \frac{in_2(I+F)}{17} \rceil$$

**Number of Block RAMs:** The 18K-bit block SelectRAM+ (BRAM) blocks on Virtex-2Pro FPGA can be instantiated in the HDL descriptions to implement memory elements like lookup tables (LUT). In order to store a table with  $n$  entries and  $p$  bits per entry, a BRAM configuration with greater than  $n$  entries is selected and then multiple BRAMs of that type are utilized to accommodate the  $p$  bits per entry. For instance, if  $n = 1600$ ,  $p = 23$ , we choose the 2K× 9-b configuration and then utilize  $\lceil \frac{23}{9} \rceil = 3$  of such BRAMs.

### III-A2. Floating Point IP Cores

Models have been built for IP cores and custom-made kernels operating in floating point format. Table III lists the cores and the corresponding model parameters.

**Number of Slices:** We have derived expressions for estimating the number of slices for the floating point cores listed in Table III. Some of these expressions are listed below.

$$Slice(add\_FP) = 5.40 \cdot E + 11.06 \cdot M + 51.20$$

$$Slice(sub\_FP) = 6.35 \cdot E + 10.88 \cdot M + 47.50$$

$$Slice(sqrt\_FP) = 2.87 \cdot E + 1.02 \cdot M^{2.0496} + 75.67$$

$$Slice(recip\_FP) = 2.20 \cdot E + 3.94 \cdot M^{1.764} + 14.24$$

$$Slice(mult\_FP) = \begin{cases} 5.00 \cdot E + 2.67 \cdot M + 4.00 & \text{if } M \leq 17 \\ 3.66 \cdot E + 5.46 \cdot M + 24.82 & \text{if } M > 17 \end{cases}$$

The relationship between the number of slices and the parameters is linear for Add/Sub and Shift, is nonlinear for Reciprocal, Square Root and Float-to-Fixed, and is piecewise linear for Multiplier. The minimum, maximum and average errors for estimating the number of slices are given in Table III. All errors are fairly small.

**Table III.** Floating-point IP cores: Parameters and Errors for estimating the number of slices

FP cores	Model Parameters	Min.Err	Max.Err	Avg.Err
Add/Sub	$E, M$	0.00%	1.52%	0.74%
Square Root	$E, M$	0.26%	5.00%	3.24%
Float-to-Fixed	$out(I, F)$	1.03%	6.07%	1.77%
Reciprocal	$E, M$	0.71%	5.10%	1.33%
Shifter	$E, M, amount\ of\ shift$	0.00%	0.00%	0.00%

**Number of 18x18-bit Multipliers:** The function to calculate the number of MULT18 blocks needed for floating point multiplication is shown below. Here both the inputs have the same number of mantissa bits.

$$M_{18}(mult\_FP) = \lceil \frac{M}{17} \rceil \cdot \lceil \frac{M}{17} \rceil$$

**Number of Block RAMs:** The number of BRAMs in floating point design is estimated in the same way as in the fixed point design.

### III-B. Total Area Estimation

A typical design consists of multiple components, where each component consists of multiple IP cores, MULT18 blocks and BRAMs. The total number of BRAM modules ( $T_{Bram}$ ) and the total number of MULT18 blocks ( $T_{M18}$ ) can be obtained by adding the number of these modules in each of the components. However, the number of slices in a complete design cannot be simply estimated by adding the number of slices in each component. This is because during synthesis, placement and routing steps, the Xilinx software automatically optimizes the design and the optimization procedure is not transparent to the designer. For instance, in a small design, typically only half of the LUT and flip-flop resources available in a slice are utilized. As the design gets larger, more of the LUTs and flip-flops in a single slice get utilized and so the number of slices does not scale proportionately. Also the overhead due to routing increases and tends to occupy a significant number of slices, a factor which is not taken into account in the previous single-module models.

To correct this discrepancy, we have developed a simple empirical model based on evaluation of a large number of candidate designs. We calculated a scaling factor,  $\alpha$ , that is proportional to the number of slices in the design. The total number of slices,  $T_{slice}$  is then obtained by scaling the total number of slices by the factor  $\alpha$ .

$$T_{slice} = \alpha(utilization) \times \sum Slice()$$

Let  $S_{slice} = \sum Slice()$  be used to estimate the ‘slice utilization’ of the design. We approximate the relationship between the scaling factor  $\alpha$  and  $S_{slice}$  by a damped pendulum function [19] as follows.

$$\alpha(utilization) = A \cdot e^{-\beta \cdot S_{slice}} \cdot \cos(w \cdot (S_{slice} - \phi)) + \theta,$$

where  $A = 2.374$ ,  $\beta = 0.0067$ ,  $w = 0.468$ ,  $\phi = 266.59$ , and  $\theta = 1.128$ . This model gives fairly good results as shown in Section IV.

### III-C. Power Estimation

The power estimate of the FPGA implementation is given by the sum of static power and dynamic power. The static power is set by Xpower and depends on the specific FPGA family. The dynamic power is the sum of logic power, input/output power, signal power and clock power.

In order to derive the power models, we group the power components into two categories: (i) those that are not dependent on the area of a design, including static power and input/output power, and (ii) those that are proportional to the area of a design, including logic power, signal power and clock power. The power models for the components that are proportional to the area of the design are derived by performing nonlinear regression analysis on the area and power data of the IP cores.

### III-C1. Power Components not related to Area

**Static Power:** Xpower has a default static power consumption which is set to 204 mW for a single Virtex-2Pro-100 device in Xilinx ISE 8.2i.

**Input and Output Power:** When chip voltage, clock frequency and activity rate are fixed, the input power and the output power depends on the number of Input/Output Blocks (IOBs), which is proportional to the number of input/output pins in the design. The expressions for estimating the input and output power are listed below.

$$Power\_input = (input\_pins) \cdot 0.125 + 1$$

$$Power\_output = (output\_pins) \cdot 2.25$$

The average error of input and output power estimates is 5.09% and 2.13%, respectively.

### III-C2. Power Components related to Area

The majority of the dynamic power comes from logic power, clock power and signal power, which is greatly affected by the area of the actual implementation. We develop separate models for implementations that need block RAM and those that do not. In each case, we derive the model coefficients using regression analysis.

#### Implementations with BRAMs:

**Logic Power:** Logic power is a function of the number of slices, block RAMs and MULT18.

$$Power\_logic = \alpha_1 \cdot (T_{slice})^{c_1} + \alpha_2 \cdot (T_{M18})^{c_2} + \alpha_3 \cdot (T_{Bram})^{c_3} + c_4$$

where  $\alpha_1, \alpha_2, \alpha_3, c_1, c_2, c_3$  are constants listed in Table IV.

**Signal Power / Clock Power:** Signal power is proportional to the number and length of nets over which signal switching occurs. Clock power depends on the distribution of the clock nets, which depends on the chip area. We put these two components together since they are both dependent on the total area of the implementations

$$Power\_signal (clock) = \alpha_1 \cdot (c_1 \cdot T_{slice} + c_2 \cdot T_{M18} + T_{Bram})^{c_3} + c_4$$

where  $\alpha_1, c_1, c_2, c_3$  and  $c_4$  are constants listed in Table IV.

**Table IV.** Coefficients for power models of implementations with BRAM.

Power	$\alpha_1$	$\alpha_2$	$\alpha_3$	$c_1$	$c_2$	$c_3$	$c_4$
Logic	0.03	-1.28e-7	-5.93	1.18	4.21	0.83	-1.26
Signal	0.42	n/a	n/a	0.12	-0.42	1.09	9.46
Clock	0.18	n/a	n/a	0.20	1.24	1.06	47.47

#### Implementations without BRAMs:

In designs where no BRAMs are needed, the logic power, signal power and clock power can be estimated using similar formula. However, the actual value of the coefficients are quite different as shown in Table V.

$$Power\_logic (signal/clock) = (c_1 \cdot T_{slice} + T_{M18})^{c_2} + c_3$$

**Table V.** Coefficients for power models of implementations without BRAM

Power	$c_1$	$c_2$	$c_3$
Logic	0.07	0.96	-1.65
Signal	0.08	1.03	9.25
Clock	4.23	0.59	-18.73

### III-D. Timing Estimation

The timing model for each IP core is represented by the latency and expressed in terms of the number of clock cycles. Clearly, the latency depends on the bit-widths of the input and output operands. In floating point implementation, the latency value is based on the exponent ( $E$ ) and mantissa bits ( $M$ ), e.g. the 'Sqrt' operation with  $M$  mantissa bits has a latency of  $M + 4$  clock cycles. In fixed point implementation, the latency value is a function of the integer and fraction bits of the input and output operands. Table VI presents the timing model functions for some of the primitive hardware modules. These estimates are used by the FANTOM tool to generate timing information of the fully pipelined FPGA implementations.

**Table VI.** Latency estimation for each operation in TANOR.

Operation	Data Type	Model Parameters	Latency (cycle)
Add/Sub	FX	$in_1(I,F), in_2(I,F)$	$L = 1$
	FP	$E, M$	$L = \begin{cases} 9 & M \leq 4 \\ 10 & 4 < M \leq 13 \\ 11 & 13 < M \leq 28 \\ 12 & 28 < M \leq 61 \\ 13 & 61 < M \end{cases}$
Multiplier	FX	$in_1(I,F), in_2(I,F)$	$L = 3 + c_1 + c_2$ $c_i = \lfloor \frac{in_i(I+F)}{18} \rfloor, i = 1, 2$
	FP	$E, M$	$L = \begin{cases} 4 & M \leq 16 \\ 6 & 16 < M \leq 33 \\ 7 & 33 < M \leq 50 \\ 8 & 50 < M \leq 63 \end{cases}$
Sqrt	FX	$in(I,F), out(I,F)$	$L = 3 + c$ $c = out(F) + \frac{in(I)-1}{2}$
	FP	$E, M$	$L = M + 4$
Reciprocal	FX	$divisor(I,F), out(I,F)$	$L = \min(36, c)$ $c = 4 + divisor(F) + out(F)$
	FP	$E, M$	$L = M + 4$
Round	FX	$in(I,F), out(I,F)$	$L = 1$
	FP	$E, M$	
LUT	FX	$in(I,F), out(I,F)$	$L = 2$
	FP	$E, M$	

#### IV. MODEL VALIDATION

In this section, we compare the area and power estimates using our models and those obtained by actual synthesis followed by placement and routing for a representative set of examples. These examples include implementations that require BRAMs such as lookup table based function evaluations and implementations that require no BRAMs such as 1D Discrete Fourier Transform and 1D Discrete Cosine Transform. We do not present the timing results for these examples since all the implementations are all fully pipelined and there is no mismatch between estimates and synthesis results.

##### IV-A. Lookup Table based Implementations

We choose two functions that are important in scientific computations and are implemented using a combination of lookup tables and interpolation schemes [21]. The two functions are  $J_0(x)$  which is the Bessel function of the first kind with zero order, and  $e^{-x}$  which is the exponential function. We use FANTOM tool [16] to generate the HDL of these two functions when implemented in fixed point format. The configuration labeled  $(n, X, Y)$  corresponds to the case where  $n$  is the degree of Taylor polynomial [21],  $X$  is the maximum number of total bits and  $Y$  is the maximum number of fraction bits in fixed point implementations.

**Table VII.** Resource estimation results for  $J_0(x)$  and  $e^{-x}$  functions.

Config.	Slices			Total Power (mW)		
	Estim.	Synth.	Err.	Estim.	Synth.	Err.
$J_0(3,64,5)$	297	307	3.26%	397	369	7.80%
$J_0(3,64,10)$	418	441	5.22%	417	406	2.86%
$J_0(3,64,15)$	779	759	2.64%	508	482	5.50%
$J_0(3,64,20)$	1258	1113	12.99%	629	583	7.99%
$J_0(9,64,5)$	1143	1135	0.70%	586	590	0.66%
$J_0(9,64,10)$	1580	1844	14.30%	707	772	8.35%
$J_0(9,64,15)$	2057	2437	14.97%	845	945	10.09%
$J_0(9,64,20)$	4779	4315	10.57%	1589	1437	10.62%
$\exp(3,64,5)$	187	194	3.61%	368	339	8.50%
$\exp(3,64,10)$	291	273	6.58%	391	363	7.79%
$\exp(3,64,15)$	517	465	11.18%	445	424	5.00%
$\exp(3,64,20)$	805	722	11.45%	518	481	7.87%

The comparison results are shown in Table VII. For the 12 configurations shown here, the average error is 8.20% for the number of slices and 6.98% for the total power. The results of block RAM and 18x18-bit Multipliers are not shown because

there is no mismatch between the estimated and synthesized results.

#### IV-B. Transform Computations

In this section, we choose two algorithms which are widely used in digital signal processing: the 8-point FFT (FFT8) and the 8-point DCT (DCT8). The FFT8 is implemented in floating point; the notation  $(E, M)$  stands for  $E$  exponent bits and  $M$  mantissa bits. We also mapped two FFT8 implementations on a single FPGA to check the validity of our models for situations with extremely large slice utilization (99%). The DCT8 is implemented in fixed point; the notation  $(8, 16)$  means the input/output data are 8 bits and the internal variables are 16 bits.

**Table VIII.** Resource estimation results for FFT8 and DCT8.

Config.	Slices			Total Power (mW)		
	Estim.	Synth.	Err.	Estim.	Synth.	Err.
FFT(8,16)	16515	16808	1.74%	4835	4922	1.75%
FFT(8,18)	18112	18377	1.44%	5230	5244	0.26%
FFT(8,20)	19462	19486	0.12%	5554	5572	0.33%
FFT(8,23)	21480	21622	0.65%	6036	6040	0.06%
2 FFT(8,16)	33030	34219	3.47%	7838	7438	5.38%
2 FFT(8,18)	36225	37389	3.11%	8535	7971	6.98%
2 FFT(8,20)	38925	39680	1.89%	9092	8502	6.94%
2 FFT(8,23)	42961	44066	2.51%	9924	9353	6.01%
DCT(8,16)	431	421	2.38%	896	893	0.36%

Table VIII lists the estimated values, the actual values and the modeling errors for both the number of slices and the power consumption of the two transform examples. For the FFT8 and DCT8 configurations, the average error is 1.92% for the number of slices and 3.14% for the total power. Note that the error here is lower than the LUT based implementations presented in Table VII. This is because the estimate for the number of slices is a lot more accurate when the design is large and a significant portion of the FPGA slices is utilized. Since the estimate of the power consumption is closely related to the estimate of the number of slices, this translates to a fairly accurate estimate of the power consumption as well.

### V. APPLICATION IN DESIGN SPACE EXPLORATION

#### V-A. Exploration in Hardware Domain

We demonstrate the effectiveness of the estimation models in facilitating design space exploration of hardware-only FPGA implementations. We choose two-dimensional (2D) FFT as the representative example. The 2D FFT implementation is based on a 2D decomposition algorithm where the  $N \times N$  input data is decomposed into a mesh of size  $p \times p$ , where each element of the mesh is a block of size  $k \times k$ ,  $k = N/p$ . Butterfly operations are done among the blocks along the rows and along the columns, followed by local 2D FFT on each block. Details of the specific decomposition algorithm and the detailed architecture description can be found in [22].

In such a decomposition, the mesh size plays an important role. We need to decide whether a particular mesh size can fit in the target FPGA without actual synthesis. Table IX presents the component area and time values that we used for fast estimation. In this example, the data type is set to be 24 bits in fixed point format. The values of the standard 1D FFT IP core (available on Xilinx FPGA) are from actual synthesis results, and the values of the Add/Sub and Multiplier are from the area and timing models described in Section III. The  $\theta_n$  in Table IX stands for the initial latency of the FFT IP core, measured from the first input to the first output. For instance, for  $N = 128$ ,  $\theta_{128} = 91$  cycles. In scenarios where the inputs are streamed, the initial latency can be ignored.

**Table IX.** Area and Timing results for IP cores used in 2D FFT.

Component	Area			Time
	Slices	MULT18	BRAM	clock cycle
N-point FFT				
8	1591	12	0	$8 + \theta_8$
16	1959	12	0	$16 + \theta_{16}$
32	2984	24	0	$32 + \theta_{32}$
64	3274	24	4	$64 + \theta_{64}$
128	4229	36	7	$128 + \theta_{128}$
Add/Sub	12	0	0	1
Multiplier	94	4	0	5

**Table X.** Area and time estimates for  $128 \times 128$  2D FFT implementations.

Configuration	Result	Area			Time (cycles)	V2P-100 (Fit?)	V2P-50 (Fit?)
		Slices	BRAM	MULT18			
Row-Column	Synth.	18683	128	144	9025	Yes	Yes
	Estim.	18608	124	144	8374		
	Err.	-0.40%	-3.13%	0.0%	-7.21%		
$2 \times 2$ mesh (4 PEs)	Synth.	28400	146	288	6350	Yes	No
	Estim.	29608	136	288	6144		
	Err.	4.25%	-6.85%	0.0%	-3.24%		
$4 \times 4$ mesh (16 PEs)	Synth.	n.a.	n.a.	n.a.	n.a.	No	No
	Estim.	54720	112	576	3072		
$4 \times 4$ mesh <sup>†</sup> (8 PEs)	Synth.	32472	96	336	6280	Yes	No
	Estim.	30848	94	346	6144		
	Err.	-5.00%	-2.08%	4.76%	-2.17%		

Using these models, we estimate the area and time for different  $128 \times 128$  2D FFT implementations. The Row-Column algorithm based implementation uses 4 128-point FFT IP core for row- and column-wise operations. For 2D decomposition implementations, we choose  $2 \times 2$  and  $4 \times 4$  meshes. In  $2 \times 2$  mesh, the  $128 \times 128$  input data is divided into four  $64 \times 64$  blocks, each block is processed by a processing element (PE). Similarly, for  $4 \times 4$  mesh, the  $128 \times 128$  data is divided into 16 blocks, each of size  $32 \times 32$ . These are processed by either 16 PEs or 8 PEs (in  $4 \times 4^\dagger$  case). We consider two target FPGA devices, Virtex-2Pro-100 (V2P-100) and Virtex-2Pro-50 (V2P-50). V2P-100 device contains 44K slices, 444 BRAMs and MULT18s, and V2P-50 only contains half of the resources of V2P-100. The synthesized and estimated area/time results, as well as the estimation errors, are presented in Table X.

From Table X, we see that the area and timing estimation error is reasonably small. For the 3 configurations that can fit on V2P-100 (row 1, 2, 4), the average error is 3.21% for the number of slices, 4.02% for the number of BRAMs, 1.59% for the number of MULT18 and 4.20% for the total computation time. The estimated time is always less than the actual simulation results because the FFT IP core initializing time ( $\theta_n$  in Table IX) and the output shuffling time are not considered in the estimation. We also see that the Row-Column algorithm (row 1) requires lower resources but 30% more computation time.

Table X also shows that certain implementations cannot fit onto the FPGA platforms. For example, the estimates indicate that a  $2 \times 2$  mesh implementation is not viable if the target device is V2P-50 but is viable for V2P-100. The  $4 \times 4$  mesh implementation with 16 PEs is not viable even for a V2P-100 device. However if the  $4 \times 4$  mesh is implemented with 8 PEs, then it can be mapped to a V2P-100 device.

## V-B. Exploration in Hardware-Software Domain

In this section, we show how the hardware estimation models can also be used to generate better hardware-software (HW-SW) co-design solutions.

In the FANTOM automation flow shown in Figure 1, the code parser and analyzer (CPA) stage now employs an intermediate code representation, called *loop hierarchy tree (LHT)* [23], to capture the entire program structure. For every function, each loop header statement is represented by an internal node in an LHT, statements in a loop are placed in leaf nodes, and loop nesting relationships are captured by edges. Thus, LHT leaves contain the actual code fragments, and internal nodes merely capture the loop iteration counts. The functions are processed in a bottom-up order in the call graph. The leaf nodes can be processed by hardware or software. Branch and bound search is used to find the optimal solution that satisfies area requirement with minimum latency. Once a solution is found, hardware and software modules are generated automatically through the FANTOM flow.

The quality of partitioning depends on the accuracy of the estimation model. The hardware estimates are very accurate and obtained using the models described in Section III. The software estimates are based on best guesses for the different types of operations when implemented on the PowerPC embedded in the FPGA. Table XI lists the latency estimates of each type of operation in terms of cycles with the assumption that the FPGA has a frequency of 100 MHz and PowerPC CPU has a frequency of 300 MHz. The area of the FPGA based hardware is estimated in terms of number of slices and  $18 \times 18$ -bit multiplier (MULT18). The area of the software implementation is set to zero, since, apart from the PowerPC, it does not require any additional FPGA resources.

An example of the modeling directed hardware-software partitioning is demonstrated in Table XII. The target application is the gravitational N-body interaction problem [24], which simulates the evolution of a system of  $N$  interacting bodies. We assume that the cache hit rate is 95% and for a hit, the latency is 3 cycles. It can be seen that while the pure hardware

**Table XI.** Resources estimations for hardware-software partitioning.

Operation	Software		Hardware		
	Latency	Area	Latency	Slice	MULT18
Add/Sub	1	0	1	352	0
Multiplier	6	0	5	181	4
Divider	19	0	16	1027	0
Memory Access	80	0	1	n.a.	

solution (column 2) exceeds the hardware requirements, and the pure software solution (column 3) exceeds the timing requirements, the hardware-software co-design solution satisfies both the hardware and the timing constraints. Moreover, the partitioning process is very fast, e.g. for a program consisting of 30 – 40 loops, it takes around one minute to find the best hardware-software solution. Thus, this modeling directed hardware-software partitioning facilitates fast design space exploration in the FANTOM algorithm-architecture co-design platform.

**Table XII.** Comparison among hardware, software, and hardware-software co-design solutions for plain tiling.

Metrics	Hardware	Software	Hardware-Software	Constraints
Latency(cycles)	2E+06	2E+08	4E+06	1E+09
Slice(numbers)	70702	0	17196	44096
Multiplier(numbers)	1576	0	328	444

## VI. CONCLUSION

In this paper we have presented area and power estimation models for IP core based FPGA implementations. These models were developed to speed up the algorithm-architecture co-exploration for systems that have to meet area/power/accuracy requirements. The models consist of parameterized functions that estimate the hardware resource (number of slices, MULT18, BRAM) and latency for IP cores, and use of these functions to accurately estimate system-level area, time and power consumption. The models have been derived using curve fitting and non-linear regression methods. The average error for fairly large designs (FFT8) is only 1.87% for area estimation and 3.48% for power estimation. Further more, the time required to generate these estimates is of the order of microseconds, as compared to minutes or even hours for designs which undergo actual synthesis followed by placement and routing.

The modelling procedure used in the paper can be used for other FPGA families but with caution. If the components are complex such as DSP-48 in Virtex-4, the accuracy of the models for designs with low utilization is not as high. A more thorough understanding of when the parts of a complex component get instantiated is needed for developing high accuracy models.

## ACKNOWLEDGEMENTS

This work was supported by a grant from DARPA W911NF-05-1-0248. The authors would like to thank the members of the FANTOM project, Dr. J.S. Kim, P. Mangalagiri, Dr. V. Narayanan and Dr. M. Kandemir of Computer Science and Engineering Department, Pennsylvania State University, C.L. Yu of Electrical Engineering Department, Arizona State University, and Dr. N. Pitsianis and Dr. X. Sun of Computer Science Department, Duke University, for their valuable help.

## VII. REFERENCES

- [1] L. Deng, K. Sobti, and C. Chakrabarti, "Accurate models for estimating area and power of FPGA implementations," in *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing - (ICASSP)*, 2008, pp. 1417–1420.
- [2] MATLAB, "The MATLAB website," <http://www.mathworks.com>, 2007.
- [3] N.-E. Zergainoh, L. Tambour, P. Urard, and A. A. Jerraya, "Macrocell builder: IP-block-based design environment for high-throughput VLSI dedicated digital signal processing systems," *EURASIP Journal on Applied Signal Processing*, vol. 2006, no. 14, pp. 1–11, 2006.
- [4] Y. Atat and N. E. Zergainoh, "Simulink-based MPSoC design: new approach to bridge the gap between algorithm and architecture design," in *IEEE Computer Society Annual Symposium on VLSI*, 2007, pp. 9–14.
- [5] "System Generator for DSP," Xilinx Webpage, 2007, [http://www.xilinx.com/ise/optional\\_prod/system\\_generator.htm](http://www.xilinx.com/ise/optional_prod/system_generator.htm).
- [6] P. Banerjee and et al., "Overview of a compiler for synthesizing MATLAB programs onto FPGAs," *IEEE Transactions on VLSI systems*, vol. 12, pp. 312–323, 2004.
- [7] G. Nordin, P. Milder, J. Hoe, and M. Puschel, "Automatic generation of customized discrete Fourier transform IPs," in *Proceedings of the 42nd Design Automation Conference - (DAC)*, 2005, pp. 471–474.
- [8] A. Nayak, M. Haldar, A. Choudhary, and P. Banerjee, "Accurate area and delay estimators for FPGAs," in *Proceedings of the Design, Automation and Test conference in Europe - (DATE)*, 2002, pp. 862–869.

- [9] P. Bjur eus, M. Millberg, and A. Jantsch, "FPGA resource and timing estimation from Matlab execution traces," in *Proceedings of the International Symposium on Hardware/software Codesign - (CODES)*, 2002, pp. 31–36.
- [10] M. Xu and F. Kurdahi, "Area and timing estimation for lookup table based FPGAs," in *Proceedings of the European Conference on Design and Test - (EDTC)*, 1996, pp. 151–157.
- [11] R. Enzler, T. Jeger, D. Cottet, and G. Troster, "High-level area and performance estimation of hardware building blocks on FPGAs," in *Proceedings of International Workshop on Field-Programmable Logic and Applications - (FPL)*, 2000, pp. 525–534.
- [12] D. Chen, J. Cong, and Y. Fan, "Low-power high-level synthesis for FPGA architectures," in *Proceedings of International Symposium on Low Power Electronics and Design - (ISLPED)*, 2003, pp. 134–139.
- [13] K. K. W. Poon, A. Yan, and S. J. Wilton, "A flexible power model for FPGAs," in *Proceedings of International Conference on Field-Programmable Logic and Applications - (FPL)*, 2002, pp. 312–321.
- [14] A. Amira and S. Chandrasekaran, "Power modeling and efficient FPGA implementation of FHT for signal processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, pp. 286–295, March 2007.
- [15] P. A. Milder, M. Ahmad, J. C. Hoe, and M. Puschel, "Fast and accurate resource estimation of automatically generated custom DFT IP cores," in *Proceedings of the International Symposium on Field Programmable Gate Arrays*, 2006, pp. 211–220.
- [16] J. Kim and et al., "TANOR: A tool for accelerating N-body simulations on reconfigurable platform," in *Proceedings of International Conference On Field Programmable Logic And Applications - (FPL)*, 2007, pp. 68 – 73.
- [17] "Logicore," Xilinx Webpage, 2007, <http://www.xilinx.com/ipcenter/index.htm>.
- [18] LogiccoreAddSub, "The Xilinx webpage," [www.xilinx.com/ipcenter/catalog/logiccore/docs/addsub.pdf](http://www.xilinx.com/ipcenter/catalog/logiccore/docs/addsub.pdf), 2007.
- [19] P. H. Sherrod, "Nonlinear Regression Analysis Program," 2005, <http://www.nlreg.com/NLREG.pdf>.
- [20] Xpower, "The Xilinx Xpower webpage," [http://www.xilinx.com/products/design\\_tools/logic\\_design/verification/xpower.htm](http://www.xilinx.com/products/design_tools/logic_design/verification/xpower.htm), 2007.
- [21] K. Sobti and et al., "Efficient function evaluations with lookup tables for structured matrix operations," in *Proceedings of the IEEE Workshop on Signal Processing Systems - (SiPS)*, 2007, pp. 463 – 468.
- [22] J. Kim, "High performance signal processing on reconfigurable platforms," Ph.D. dissertation, Pennsylvania State University, State College, Pennsylvania, United States, 2008.
- [23] Y. Zhang and et al., "Compiler directed hierarchical hardware-software mapping for algorithm-architecture codesign," in *Grace Hopper Celebration of Women in Computing- (GHC)*, 2008, p. 36.
- [24] J. Makino, "The GRAPE project," *Computing in Science and Engineering*, vol. 8, no. 1, pp. 30–40, 2006.



**Lanping Deng** received his BS and MS degrees in Electrical Engineering from Tsinghua University, Beijing, China in 2003 and 2005, respectively. He is a PhD candidate in Electrical Engineering Department, Arizona State University, Tempe, AZ. His research interests include hardware-software codesign, FPGA-based accelerator design and EDA tool design.



**Kanwaldeep Sobti** received his MS degree in Electrical Engineering from Arizona State University, Tempe in 2007. He is a Senior Design Engineer with Advanced Micro Devices (AMD) in the field of structural testability and DFT techniques for AMD's next generation processor. His interests lie in microprocessor design, low power design, and CAD design, especially for ESL techniques.



**Yuanrui Zhang** received her BS and MS degrees in Computer Science from Peking University, Beijing, China in 2003 and 2006, respectively. She is working toward the PhD degree in Computer Science and Engineering Department, Pennsylvania State University, University Park. Her research interests include hardware-software mapping for algorithm-architecture codesign, helper thread and software data prefetching for chip multi-processors, and compiler optimization.



**Chaitali Chakrabarti** (SM'02) is a Professor with the Department of Electrical Engineering, Arizona State University, Tempe. Her research interests are in the areas of low power embedded systems design and VLSI architectures and algorithms for signal processing, image processing, and communications.