Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2015, Article ID 784672, 20 pages
http://dx.doi.org/10.1155/2015/784672

Research Article

Core-Level Modeling and Frequency Prediction for DSP Applications on FPGAs

Gongyu Wang, Greg Stitt, Herman Lam, and Alan George

NSF Center for High-Performance Reconfigurable Computing (CHREC), Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611-6200, USA

Correspondence should be addressed to Gongyu Wang; wangg@chrec.org

Received 3 March 2015; Accepted 10 August 2015

Academic Editor: Michael Hübner

Field-programmable gate arrays (FPGAs) provide a promising technology that can improve performance of many high-performance computing and embedded applications. However, unlike software design tools, the relatively immature state of FPGA tools significantly limits productivity and consequently prevents widespread adoption of the technology. For example, the lengthy design-translate-execute (DTE) process often must be iterated to meet the application requirements. Previous works have enabled model-based, design-space exploration to reduce DTE iterations but are limited by a lack of accurate model-based prediction of key design parameters, the most important of which is clock frequency. In this paper, we present a core-level modeling and design (CMD) methodology that enables modeling of FPGA applications at an abstract level and yet produces accurate predictions of parameters such as clock frequency, resource utilization (i.e., area), and latency. We evaluate CMD's prediction methods using several high-performance DSP applications on various families of FPGAs and show an average clock-frequency prediction error of 3.6%, with a worst-case error of 20.4%, compared to the best of existing high-level prediction methods, 13.9% average error with 48.2% worst-case error. We also demonstrate how such prediction enables accurate design-space exploration without coding in a hardware-description language (HDL), significantly reducing the total design time.

1. Introduction

Field-programmable gate arrays (FPGAs) combine flexibility of software with performance of custom hardware, often resulting in orders of magnitude speedup [1–3] and improved energy efficiency [4–6]. Despite these advantages, FPGA usage has been limited largely due to a requirement for hardware expertise and relatively immature design flows. A typical FPGA design flow follows a design-translate-execute (DTE) methodology where the designer *designs* register-transfer level (RTL) functionality, *translates* that functionality into a circuit through synthesis, placement, and routing, and finally *executes* the resulting circuit for verification or performance analysis. One significant problem with this approach is that design-space exploration (DSE) must iterate over the lengthy DTE process, resulting in low productivity. To optimize a design, the designer must reevaluate design decisions, modify the code, retranslate the code, reexecute the new circuit, and iterate until constraints are met. These DTE iterations can require weeks or months of increased design time [7, 8], because even if only minor change is made to the code, placement and routing may take hours or even days.

As a solution to this problem, previous works [9–14] introduced techniques for model-based, design-space exploration before creating a functional design. As shown in Figure 1, we refer to such exploration as formulation. To perform formulation, a designer abstractly models an application and makes predictions (e.g., [15]) that enable rapid exploration of the design space. Instead of iterating over DTE in search of good design choices, a designer using DSE tools can iterate during formulation to identify the promising designs before performing DTE. By reducing DTE iterations, formulation has shown to significantly improve productivity [16].

Current formulation tools [12, 15] model algorithm implementation on FPGAs (i.e., FPGA applications) as a black box and make performance predictions based on key parameters of the model (e.g., clock frequency, area, and

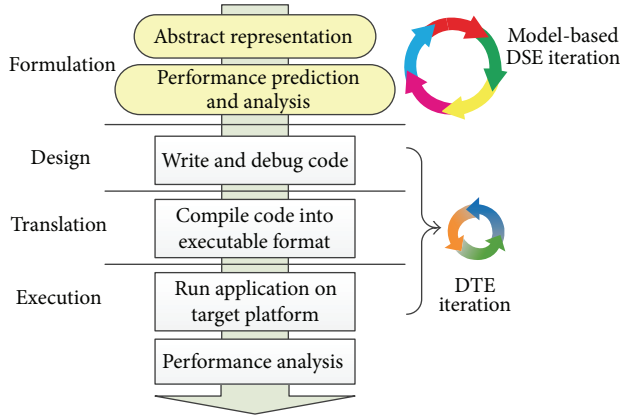


Figure 1: Formulation, design, translation, execution (FDTE) model for FPGA application development.

latency), which are often provided by designers via estimation or implementation. Although implementation can produce accurate parameter values, it is time-consuming due to lengthy placement and routing. Therefore, when using those formulation tools, application designers typically settle for crude parameter-estimation methods (e.g., use lowest frequency among components as estimated frequency of entire application). It is not uncommon that the crude estimations fail to differentiate design choices or even favor inferior designs over good ones. There is a need for better estimation methods of the parameters to enable more accurate DSE and more effective formulation.

To enable easier and more accurate parameter estimation for FPGA applications, we introduce a core-level modeling and design (CMD) methodology. By providing parameter-prediction methods for models created from RTL cores, CMD enables modeling of an FPGA application at an abstraction level that matches the required resources (i.e., core level), using accurate parameters for each core. With such models, CMD's prediction methods can produce accurate estimations of parameters such as clock frequency, resource utilization (i.e., area), and latency of an FPGA application, enabling core-level DSE that evaluates various design choices and identifies the favorable ones based on prediction results. Moreover, CMD provides the generation of code templates and vendor-tool frequency constraints to expedite design and translation. The CMD methods are not limited to manual usage. They can be automated and potentially integrated into any design tools, including high-level synthesis tools. We have created a prototype framework of tools that implements much of the CMD methodology.

A major research challenge of CMD is predicting the timing characteristics of FPGA applications based on the interconnection of coarse-grained cores. Although the timing parameters of each core can be determined from datasheets or micro-benchmarking, the interconnection of the cores can have a significant effect on the achievable clock frequency. A key contribution of this paper is a core-level prediction method of clock frequency, which considers the frequency of each individual core and the effects introduced by their placement and interconnection on an FPGA. Using several high-performance DSP applications, our evaluation shows that the average prediction error of CMD's clock-frequency prediction method is 3.6%, with the worst-case error of 20.4%. In comparison, the best of the existing high-level prediction methods shows an average error at 13.9%, with the worst-case error of 48.2%. Latency/area prediction methods are also presented and evaluated, which show 7% worst-case error. Note that control-oriented tasks are not considered by CMD because they are typically not on the critical paths of DSP applications.

To demonstrate that CMD's clock-frequency prediction enables more accurate DSE, we present examples of core-level DSE to show that it differentiates design choices and select the optimal design in cases where other high-level methods cannot. Moreover, we demonstrate the productivity benefits from using CMD for formulation and show significant development-time reduction for an example application.

In summary, the key contributions of this paper are listed as follows. A core-level modeling and design methodology was introduced that enables the modeling of FPGA applications at an abstract level and yet produces accurate predictions of key design parameters such as clock frequency, resource utilization (i.e., area), and latency. In particular, a high-level clock-frequency prediction method has been developed based on the core-level models of DSP applications on FPGAs. The accuracy of CMD's clock-frequency prediction method was evaluated using several typical DSP applications, demonstrating better accuracy than existing methods for high-level prediction. Thus, CMD methods enable accurate design-space exploration without having to code in a hardware-description language (HDL), significantly reducing the total design time.

The remainder of the paper is organized as follows. Section 2 presents related research. Section 3 presents an overview of CMD and the prototype framework. Section 4 discusses the details of the clock-frequency prediction method of CMD. In Section 5, we present case studies for the evaluation of the prediction methods and discuss examples of core-level DSE. Section 6 concludes the paper and discusses future work.

2. Background and Related Research

This paper is an extension of a previous work [17], in which we introduced the general core-level modeling and prediction methods, core-level DSE, and constraint generation. In addition to numerous refinements to the previous work, this paper includes the following extensions: (1) improved clock-frequency prediction method; (2) more thorough evaluation with additional case studies; (3) comparison studies with other high-level frequency prediction methods; and (4) demonstrative example of CMD's productivity benefits.

Current formulation tools, such as RAT [15] and RCML [11, 12], focused on more abstract system-level modeling and prediction. In those studies, FPGA algorithms and components are generally modeled as black boxes and clock frequencies are assumed for system-level performance prediction. CMD models FPGA algorithms at the core level,

which is generally more detailed than the task level in RAT and RCML, so that clock frequencies of the tasks can be systematically predicted. CMD could work in tandem with such system-level tools by providing accurate clock-frequency predictions. For example, RAT's performance-prediction accuracy heavily relies on accuracy of the specified clock frequency. CMD can help RAT to produce more accurate performance prediction by accurately predicting this frequency.

Mohanty and Prasanna [18] proposed kernel-level modeling of FPGA algorithms, which is similar to CMD, but their work focused on power-consumption prediction and did not provide clock-frequency prediction methods. Xilinx System Generator [19] shares the same abstract-modeling approach, but, again, clock-frequency prediction is not provided. A collection of modeling domains is defined and explored in Ptolemy [20]. Ptolemy focuses on application design using numerous models of computation, whereas CMD focuses on a model common to FPGA applications (i.e., data flow). Thus, CMD is unique in that it provides rapid model-based frequency prediction during formulation, without having to write any RTL code.

Strenski [21] provides a methodology to estimate the theoretical maximum Gflop/s for a given FPGA. However, that approach estimates theoretical performance independently of specific applications or circuits. CMD models algorithms and predicts key parameters after mapping to FPGA.

Two distinct methodologies to predict frequency and area for FPGA applications are presented in [22, 23]. The former works at the level of configurable logic blocks (CLBs), which is of finer grain than CMD's typical cores. Thus, more detailed hardware knowledge is required for modeling and the prediction process is more time-consuming. The latter works at the FPGA task level, which is generally more abstract than CMD cores. Although easy to specify, such abstraction provides less accurate parameter prediction. Also, the algorithm's structure is abstracted away to a simple set of primitive operations. CMD models specific algorithm structure as well as primitive operations but abstracts away the low-level device (e.g., CLB) information. In this manner, CMD can maintain similar prediction accuracy as [22] and approximate the prediction speed of [23]. Aside from [22, 23], there are many other tools and studies that discuss parameter prediction for frequency, area, latency, and power of FPGA applications. Those approaches base their prediction approaches either on low-level FPGA primitives [24–29] or on high-level statistical approximations [30–34]. CMD differs by providing a flexible level of abstraction that can potentially achieve the results of both approaches. However, CMD currently does not support prediction of power consumption, although such support could potentially be added.

A major contribution of this paper is a method for core-level routing prediction. There are many previous studies in this area, such as [35, 36]. CMD differs from those works in three aspects. The first is that the inputs to the models are of different granularity. For example, Das's approach [35] is based on technology-mapped netlists while CMD typically works at a higher level of abstraction, while also potentially supporting technology-mapped netlists. Thus, their analysis methods cannot be applied to CMD's model because many input parameters that their methods require are not available at CMD's higher level of abstraction. In fact, one goal of CMD is to hide low-level details. The second difference is that CMD targets different types of designers. Previous studies [35, 36] aim to help FPGA architects, whereas CMD helps FPGA application designers. Thus, previous approaches mainly consider exploration of potential FPGA architectures while CMD focuses mainly on commercial FPGA architectures. For example, FPGAs in Das's paper [35] are assumed to have no DSPs or BRAM blocks. The third difference is the use of Rent's rule [37]. Previous works such as [22, 35, 36] use Rent's rule on FPGAs with fixed lookup table (LUT) as their basic block, while two works [35, 36] assume that Rent's exponent is available as a parameter and the other [22] calculates Rent's exponent empirically. CMD uses Rent's rule at a higher level of abstraction, treating a flexible core as basic block and calculating Rent's exponent for specific applications. Moreover, in Das's paper [35], Rent's rule is used in its original form to predict the number of pins, while CMD uses it to predict average routing delay via Feuer's work [38]. Feuer's work is used in research (e.g., [39]) for wire length estimation. CMD differs from that research by conducting similar estimation on a higher abstraction level using K-means partitioning heuristic.

High-level synthesis (HLS) tools such as GAUT [40], Xilinx System Generator [19], and LabVIEW FPGA [41] improve FPGA productivity by automatically performing design-space exploration from high-level code. However, such exploration is restricted by various coding styles, often requiring designers to write an application in a specific way. As a result, designers using HLS still must perform repeated DTE iterations. Overall design time might be reduced by HLS but translation can still take hours or days. Therefore, formulation provides similar benefits even when using HLS. Furthermore, it is possible to integrate CMD's clock-frequency prediction with HLS tools to enable improved exploration.

3. Overview of CMD Methodology and Prototype Framework

The CMD methodology supports core-level modeling, parameter prediction, core-level DSE, and code-template/constraint generation. Figure 2 shows the CMD concept diagram integrated with the FDTE model from Figure 1.

Within the formulation stage, CMD has two major modules: *core-level model construction* and *parameter prediction*. The inputs to the former are the application algorithm and characteristics of the target FPGA platform, based on which a model is produced and then used by the latter to predict values of key design parameters (e.g., frequency, area, and latency). *Core-level DSE* (DSE 1 and DSE 2 in Figure 2) is enabled by evaluating various designs and selecting the ones with predicted parameters that are satisfactory for a design goal. The design goal can be singular (e.g., fastest clock speed, minimum area, or lowest latency) or combinative (e.g., fastest clock with an area budget). We pick maximum clock frequency as the example design goal in this paper, because it

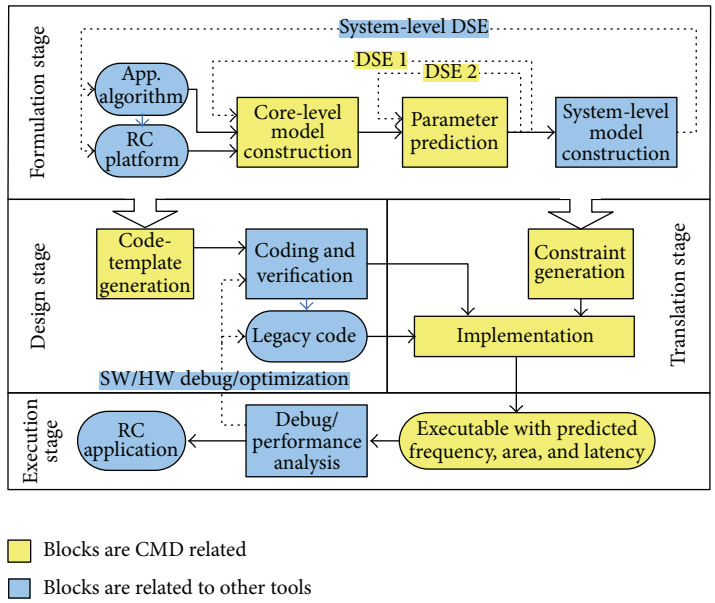


Figure 2: CMD methodology and FDTE model (dotted lines indicate iterative processes).

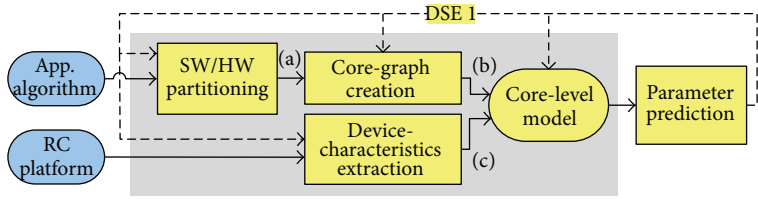


Figure 3: Core-level model construction with (a) FPGA tasks, (b) core graph, and (c) device characteristics.

is a common goal for high-performance DSP applications on FPGAs and core-level DSE for this goal is directly enabled by CMD's frequency prediction. Core-level DSE can be performed manually or with automated techniques [42].

To bridge formulation into design and translation, CMD provides *code-template generation* and *constraint generation*. The former generates skeleton HDL code in the design stage according to the core-level models. The latter provides constraints to synthesis, placement, and routing tools in the translation stage. Finally, the behavior and performance of the implemented design can be verified in the execution stage.

For proof of concept, we created a prototype framework of tools that implements and automates much of the CMD methodology. Although many of the individual modules of CMD are automated in the prototype framework, interaction between them currently requires manual effort, resulting in a semiautomatic approach.

3.1. Core-Level Model Construction. Figure 3 illustrates the core-level model construction module from Figure 2. The inputs to the module are the application algorithm and characteristics of the target FPGA platform. The first step of core-level model construction is SW/HW partitioning, which partitions out the tasks of algorithm to be implemented on FPGAs. We refer to them as FPGA tasks. CMD does not restrict techniques used for SW/HW partitioning; it can be an automated partitioning tool (e.g., [43]) or manual partitioning by designer. Note that design decisions are made during SW/HW partitioning. So, DSE (DSE 1 in Figure 3) could be conducted either manually by designers or automatically using DSE tools (based on some heuristics) to explore various partitions in the search for better parameter values (e.g., higher clock frequency) using parameter prediction. After partitioning, FPGA tasks are modeled through *core-graph creation*; and *device-characteristics extraction* (e.g., device infrastructure, basic-operation characteristics) is performed on target FPGA devices in the platform. From device characteristics, parameter values of the core graph are determined to produce a *core-level model*.

3.1.1. Core-Graph Creation and Device-Characteristics Extraction. The core-graph models the algorithms of FPGA tasks. Cores are basic elements of the core graph, which model algorithmic operations. Cores have inputs/outputs (I/O), which can be connected by directed *links* that model data or control flows. Cores and their links form the core graph. For example, a core graph for 3-input summation is shown in Figure 4(a). Cores can be defined at various levels of granularity. A designer can reduce modeling time by using more abstract cores (e.g., one 3-input summation core instead

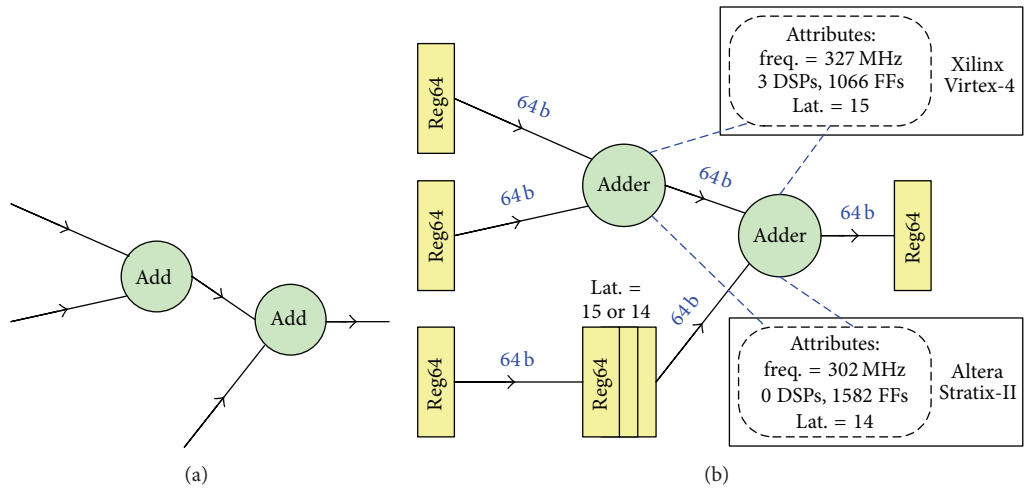


Figure 4: (a) Core graph and (b) core-level model for 3-input DPFP summation.

of two add cores) at the risk of reduced prediction accuracy. More abstract cores can also be defined by a core graph. Thus, a designer/tool can employ hierarchical modeling to manage large core graphs. Note that algorithmic design decisions (e.g., operation selection, parallelism, and algorithmic structuring) made by designers or tools are reflected in core-graph creation (e.g., abstraction level of cores, amounts of core replication, and selection of flat/hierarchical core graphs). So, as shown in Figure 3, a designer/tool could perform DSE (DSE 1) with core-graph creation.

Device-characteristics extraction involves core-instance discovery, datasheet lookup, and micro-benchmarking. *Core instances* are RTL entities with core-specified behavior and device-specific parameters (e.g., clock frequency, area, and latency/pipeline stages). Core instances can be any IP including combinational logic, DSP math functions, finite-state machines, and memories. Some core instances are provided by device vendors and thus have documentation on their achievable clock frequency, area, and latency for target FPGA devices. If not documented, the core instances are benchmarked to collect necessary parameter values. Either way, an abstraction level is formed with core instances as model elements, where a designer does not need to know the internals of core instances other than the key parameters. Note that design decisions (e.g., device selection, IP selection, and tool selection) are reflected in device-characteristics extraction (e.g., device selection, core-instance selection, and benchmarking-tool selection). Thus, as shown in Figure 3, a designer/tool could perform DSE with device-characteristics extraction.

3.1.2. Core-Level Model. As shown in Figure 3, after core-graph creation and device-characteristics extraction, a core-level model is created by a designer/tool that maps each core in the core graph to a core instance and assigns each link of the core with bit width based on I/O data types of the core instance. Design decisions are made by the designer or DSE tool to determine the core instances to which the cores are mapped. A common choice is to map cores to core instances provided by device vendors or other trustworthy sources. For example, an add core (shown in Figure 4(a)) is mapped to a pipelined DPFP adder (core instance) provided by Xilinx or Altera in order to create the core-level model shown in Figure 4(b). DSE (DSE 1 in Figure 3) can be conducted to explore mappings of add to different core instances (e.g., to pipelined adders of various latencies).

Figure 4(b) shows a core-level model for the 3-input summation example. The model consists of two types of core instances: adder and register (labeled as Reg64). On Xilinx Virtex-4 LX100 FPGA, frequency, latency, and area (the number of DSP slices and flip-flops (FF)) of the adder can be determined using the floating-point-operator datasheet [44] or Xilinx CORE Generator. Similarly, the parameter values of the adder on an Altera Stratix-II S180 FPGA can be found in [45]. Parameter values of the registers can also be found in the datasheets. To achieve maximum clock frequency, we decide to select the adder core instance with deepest pipelining, whose parameter values on both FPGAs are shown in Figure 4(b). No such option is available for Reg64 and, thus, for brevity, their parameter values are not shown in Figure 4(b).

The core graph and core-level model can potentially be created with any appropriate modeling or programming language. We leverage RCML [11] because it was created specifically for modeling FPGA applications. The RCML editor has been extended for editing core-level model and integrated into the CMD prototype framework. The editor is a drag-and-drop modeling environment with a graphical user interface similar to Simulink [46]. Using this tool, a designer starts with modeling the algorithm by graphically connecting cores via links to form a core graph. Next, the designer assigns each core and link with parameter values derived from device-characteristics extraction. Automatic DSE tools that map algorithm to core graph and assign parameter values could potentially be added to the prototype framework.

3.2. Parameter Prediction. Figure 5 shows the details of the parameter-prediction module from Figure 2, which uses the

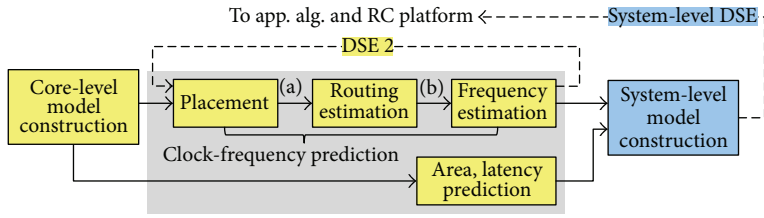


Figure 5: CMD parameter prediction with (a) placed core-level model and (b) estimated routing delay.

core-level model to estimate values of key parameters of an FPGA application, such as clock frequency, area, and latency. For *latency prediction*, latencies of core instances along each path, from input ports to output ports, are summed to produce the latency of that path. Similarly, *area prediction* sums the resource utilizations of each core instance in the core-level model to determine overall resource utilization. CMD models assume that all resource sharing occurs within a core instance as opposed to between instances. Given accurate parameter values of cores, the accuracies of latency and area prediction can be very high (to be shown in Section 5.4). We have automated area and latency prediction in the prototype framework.

Clock-frequency prediction is more challenging and thus is divided into three steps. Firstly, *placement estimation* assigns locations of the core instances on target FPGA based on a placement heuristic. Next, based on the estimated placement, CMD performs *routing estimation* to determine routing delays. Finally, the estimated routing delays are used to perform *frequency estimation* of the FPGA application. Clock-frequency prediction is a major contribution of this paper, the details of which are presented in Section 4.

Design decisions can be made by designer/tool during placement estimation via the selection and tuning of placement heuristic. Different placement heuristics work for different design goals. As mentioned, in this paper we pick the design goal of maximum clock frequency and hence a heuristic for shortest distance is selected and tuned, as discussed in Section 4. We note that placement estimation is also applicable to other design goals by selecting corresponding placement heuristics.

The clock-frequency prediction method can be automated and integrated in the prototype framework. However, because its automation requires significant effort, we manually perform the steps of the frequency prediction method for evaluation purpose and discuss how it can be automated in Section 4.

3.3. Code-Template Generation and Constraint Generation. As shown in Figure 2, besides core-level model construction and parameter prediction, CMD also includes the following: core-level DSE, code-template generation, and constraint generation. Design choices exposed by CMD for core-level DSE were discussed in the previous sections.

CMD can generate code template from the core-level model to inherit design decisions, such as types of cores and core-graph structure, as determined during core-level DSE. We can potentially integrate existing code-generation tools [47–49] into the CMD prototype framework but currently implemented a simple code-template generator within the CMD model editor for DSP applications.

Constraint generation creates frequency constraints that can reduce the number of translation iterations required to obtain maximum frequency. The generated frequency constraints are determined by the predicted frequency plus and minus some slack to account for the noise phenomena of the place-and-route process [50]. This phenomenon suggests that high frequency constraint can still be met if a circuit fails a lower constraint and more surprisingly low frequency constraint can sometimes produce faster circuit than higher ones. Thus, a range of frequency constraints must be tested in search of the one that produces the maximum frequency. The predicted frequency saves designers time to achieve maximum frequency because it provides a good starting point for the search and thus shortens the range of frequency constraints required to achieve the maximum frequency. Otherwise, designers would usually have to guess at a starting frequency constraint and search a much wider range. Note that frequency constraints are commonly used for timing-closure application development. The place-and-route process without frequency constraints usually produces circuits with modest frequency, which is not due to the noise phenomenon but tool configuration.

We perform the frequency search using two methods. One method performs exhaustive search over a range of frequency constraints with numerous place-and-route jobs executing in parallel on a high-performance computing cluster. The other method uses a binary search to reduce the total number of place-and-route iterations on a single computer. The binary search starts by selecting a range of low and high frequencies $[L, H]$ and running place-and-route with $(L + H)/2$ as the frequency constraint. After finding the resulting frequency R, the search continues with a range of $[R, H]$ if the original constraint was met and $[L, R]$ otherwise. These steps are repeated until the range is narrower than 5 MHz. Note that the binary search can be trapped at a local maximum to produce lower frequencies than the exhaustive search (an example will be shown and discussed in Section 5.3 over a distribution plot of constraints versus maximum frequency). However, using the CMD-generated constraints, binary search can quickly produce acceptable results and is useful when exhaustive search is not affordable.

Both frequency-search methods are automated and integrated in the prototype framework. They can both work with the generated frequency constraint and reduce the time needed to find the maximum frequency.

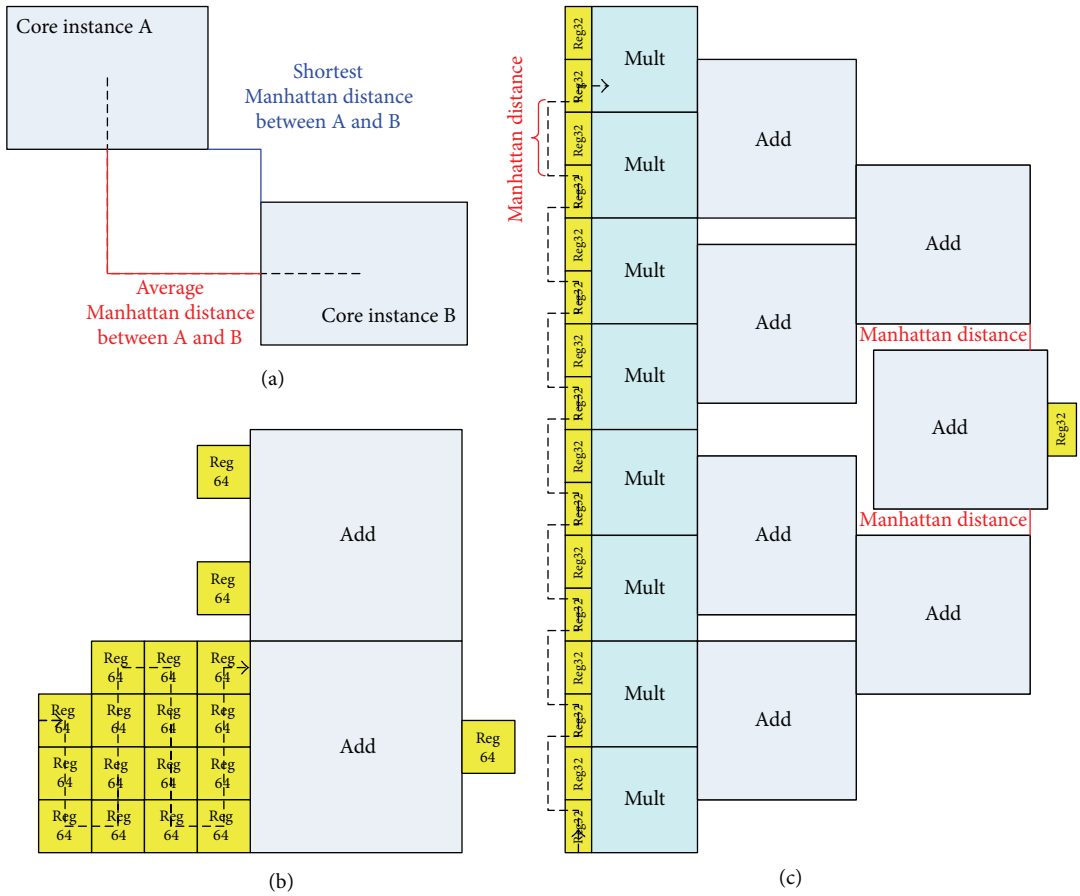


Figure 6: Placement estimation of core instances for the following: (a) generic case, (b) DPFP summation, and (c) 8-tap FIR filter.

4. Clock-Frequency Prediction Method of CMD

As noted in Section 3.2, clock-frequency prediction is the most challenging parameter-prediction method and it consists of three steps: placement estimation, routing estimation, and frequency estimation, as shown in Figure 5. In this section, we present details of each step.

4.1. Placement Estimation. The placer in CMD estimates vendor-tool placement of an application by placing each core instance on target FPGA. Although estimating exact placement is usually not feasible, core-level placement allows for quick estimation by only considering core instances as opposed to hundreds of thousands of technology-mapped units (e.g., LUTs and registers).

Placement estimation can potentially use any placement algorithm, but it currently uses a modified simulated-annealing heuristic based on Versatile Place and Route (VPR) [51]. This heuristic considers random core placements to minimize a cost function based on the bounding-box size of each net (i.e., core-instance link) and the capacities of routing channels within the bounding box. Because maximum frequency is the design goal, the heuristic places linked core instances close to each other to minimize the distances between them. The distance between a pair of linked core instances is defined as the Manhattan distance [38] between their bounding boxes, as shown in Figure 6(a). With the maximum frequency as the design goal, we use the shortest Manhattan distance between a pair of connected core instances instead of the average distance in the cost function. Note that though these two kinds of distance metric are both applicable in the cost function, shortest Manhattan distances are used for frequency estimation (to be explained in Section 4.3), as one of the factors that contribute to predicted frequency.

For the case studies presented in Section 5, we manually conduct placement estimation using vendor floor-planning tools, imitating the VPR algorithm, for the purpose of evaluation. Also, we assume that each core instance has a fixed rectangular shape with minimum length difference between its width and height. For core instances using only logic, the corresponding shape is a square. In other cases, a core instance is shaped by the layout of specialized units and cannot be a square (e.g., a core instance with DSP blocks that are laid out in a line). Changing aspect ratio may provide more accurate predictions. However, based on this evaluation methodology, we have observed that the VPR-style placement

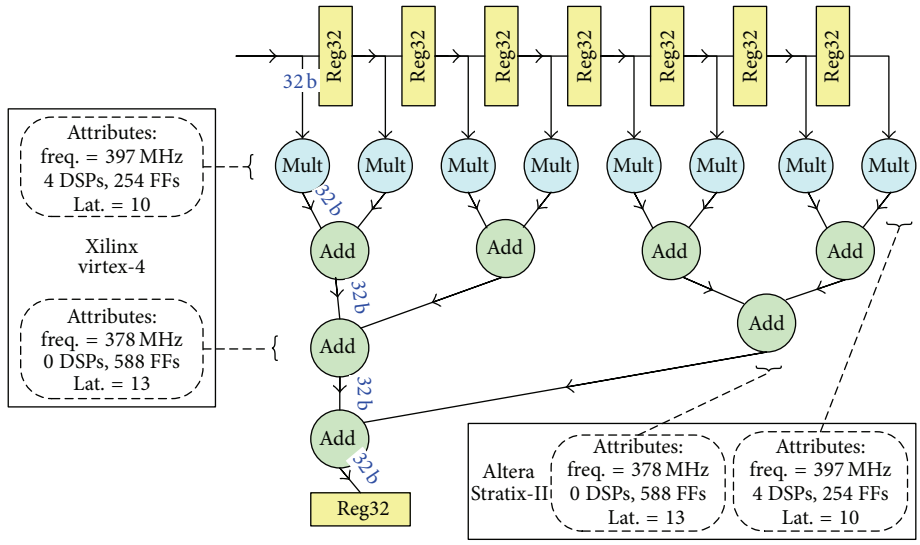


Figure 7: Core-level model of SPFP 8-tap FIR filter.

provides reasonably accurate clock-frequency predictions (results to be shown in Section 5).

For the DPFP summation example with Xilinx Virtex-4 LX100 device from Figure 4, where adders use DSP units of the FPGA, we place the two core instances of adders over the FPGA's DSP columns. For maximum frequency, the core instances are placed so that the distance between them is as short as possible. In this example, the distances between connected core instances are minimal due to placement shown in Figure 6(b). On the Stratix-II S180 device, adders consume no DSP units, which mean easier manipulation of their location and distance to maintain minimal distances between the core instances. However, it is not always possible to keep the distance between connected core instances to a minimum. Figure 6(c) shows a placement of the core-level model of an 8-tap single-precision floating-point (SPFP) FIR filter (its core-level model is shown in Figure 7) and this placement has nonzero Manhattan distances between several pairs of connected core instances. Note that relative sizes of the core shapes in Figure 6 are determined by resource utilization parameters of the core instances. Also note that multipliers in Figure 6(c) are arranged on top of the DSP columns of the target FPGA.

Our placement estimation approach is different from the placement algorithms of vendor tools. To compare results, we use the vendor's floor-plan tools to check vendor-tool placement of case-study applications against the core-level placement estimations. We have observed that core-level placement is similar to vendor-tool placement for small applications (DPFP summation and SPFP 8-tap FIR). For larger applications, it is hard to decide the similarities because low-level elements (e.g., LUTs) of core instances are sprinkled within a region due to low-level optimizations during technology mapping and low-level placement. However, our clock-frequency prediction is reasonably accurate for all

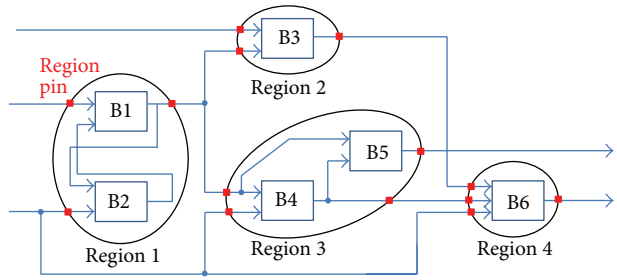


Figure 8: Example partition of a logic block diagram with 6 blocks (B1–B6).

case-study applications (to be shown in Section 5), which suggests that the presented placement estimation approach sufficiently mimics device-vendor tools. If vendor-tool placement changes significantly in future versions or for future FPGAs, then the placement heuristic used by CMD will need to change accordingly.

4.2. Routing Estimation. CMD routing estimation predicts routing delay for a specified placement estimation. In Section 2, we have noted that previous approaches [22, 23] to this problem assume certain abstraction levels and hence do not support CMD's flexible level of abstraction. Therefore, we leverage Rent's rule [37] and Feuer's [38] work to create a new approach to estimate routing delay and then clock frequency at the core level.

Given a connected logic block diagram and a partition of this diagram into *regions*, Rent's rule states that a relationship exists between P, the average number of pins per region, and B, the average number of blocks per region. An example is shown in Figure 8, in which a block represents a logic circuit, such as a logic gate, a storage element, register, and integrated

circuit chip. This relationship is generally represented by the following equation:

$$P = KB^p, \tag{1}$$

where K is the average number of pins per block and p is a small exponent that satisfies $0 < p < 1$. The value of p depends on the structure and partitioning of the block diagram. Given a partition of a block diagram, we can calculate P, K, and B and then by using (1) we can derive p. The example in Figure 8 has 4 regions of blocks, where region 1 has 2 blocks and 3 pins, region 2 has 1 block and 3 pins, region 3 has 2 blocks and 4 pins, and region 4 has 1 block and 4 pins. Thus, we can calculate that P equals $(3 + 3 + 4 + 4)/4$ and B equals $(2 + 1 + 2 + 1)/4$ for this example. Similarly, K is calculated to be $(3 + 3 + 3 + 3 + 3 + 4)/6$. Applying $P = 14/4$, $B = 6/4$, and $K = 19/6$ to (1) yields an instance of $p \approx 0.247$ for the given partition. However, to extract general Rent's exponent p, a calculation method that involves linear-regression fit in a log-log plot of P and B values at all partitioning levels is required [52].

Inserting derived Rent's exponent p into Feuer's work [38], we can predict average wire length, $\bar{r}$, internal to a circuit region using the following equation:

$$\bar{r} = \sqrt{2} \cdot \delta \cdot \frac{(2-\alpha)(5-\alpha)}{(3-\alpha)(4-\alpha)} \cdot \frac{C^{p-1/2}}{1+C^{p-1}}, \tag{2}$$

where $\alpha = 2 - 2p$ ($1/2 < p < 1$ as required in [38]), δ is the unit wire length, and C is the number of blocks in a given region. This equation is derived from several basic assumptions stated in [38]. These assumptions (e.g., circuit components are arranged in a uniform array), though appearing unrealistic for generic circuits, suit the scenario of FPGA circuits, which share common underlying structures (i.e., the FPGA fabric). Now we find the unit wire delay for an FPGA fabric, which is simply the delay between a pair of adjacent routing matrices. Note that the delay within such routing matrices is far greater than that of the connecting wires between them. Therefore, for FPGAs, we can replace unit wire length δ in (2) with the delay of two adjacent routing matrices Δ and derive the following equation that estimates the average routing delay of an FPGA circuit region:

$$\overline{\mathrm{RD}} = \sqrt{2} \cdot \Delta \cdot \frac{(2-\alpha)(5-\alpha)}{(3-\alpha)(4-\alpha)} \cdot \frac{C^{p-1/2}}{1+C^{p-1}}. \tag{3}$$

Similar to (2), this equation accounts for all routing delays internal to the region characterized by C (i.e., number of blocks in the region) and produces an average delay. If a wire has extra delay caused by routing congestion, the portion of the wire that is internal to the region is also considered in (3). Note that routing congestion is the result of the length of a routed wire being longer than the Manhattan distance between its endpoints. Also note that routing-matrix delay Δ can be derived from FPGA vendor tools. Refer to Section 5 for the measured values of Δ for the target FPGAs in the case studies.

Applying Rent's rule and (3) to a placed core-level model, we can predict the routing delay without resorting to actual routing. Our approach is summarized in Figure 9. Firstly, routing estimation partitions the placed core-level model into a number of regions. We initially consider the original K-means clustering algorithm [53] to create the desired partitioning algorithm, in which we use i as the cluster index (i-means). In addition to the cost function of i-means clustering, we also consider two cost functions for fast partition evaluation: min-cut and max-cut of number of links for a partitioning. After experimenting with the original K-means clustering, min-cut and max-cut, we have observed that though all three partitioning algorithms produce comparable results, K-means clustering with max-cut arrived at p larger than 1/2 with the least number of partitioning regions. Recalling that (3) only accounts for routing delay internal to a region, fewer regions mean that more routing congestion can be integrated into the prediction.

As shown in Figure 9, after partitioning the placed core-level model, routing estimation calculates an instance of Rent's exponent p based on the specific partitioning by plugging P, K, and B values of the partition into (1). As mentioned earlier, the rigorous method [52] to calculating p can be time-consuming. Thus, in order to improve the calculation speed, CMD's routing estimation uses a simplified method (as introduced in [39]) to find an acceptable approximation to actual Rent's exponent. Using this method, a certain number of partitioning samples (e.g., 9 samples in [39]) are collected including their values of p, $\log P$, and $\log B$. Once enough samples are collected, a linear-regression fit is conducted over the collected $\log P$ and $\log B$ values to derive the approximate p value. If this p value is not larger than 1/2, CMD's routing estimation goes back to partitioning the core-level model into more regions. If the resulting p is larger than 1/2, routing estimation moves on to calculate the routing delay $\overline{\mathrm{RD}}_i$ for each region i of the partitions by inserting p value and the number of core instances in the region (C_i value) into (3). $\overline{\mathrm{RD}}_i$ will then be used to predict frequency of the entire design. If p is not larger than 1/2 for every i ($i = 2, \ldots, N$) where N is the total number of core instances in the model, we assume that the average wire lengths in the regions are short and they have negligible routing delays for core-level prediction. This assumption is based on the work of Christie and Stroobandt [52]. They point out that the amount of optimization achieved in placement is reflected by Rent's exponent p and lower values of p correspond to a greater fraction of short interconnects.

Note that our partitioning algorithm is inspired by previous research [36, 39]. It is concluded in [36] that random-partitioning algorithms based on center-and-radius parameters produce statistically more accurate and stable Rent's exponent values. K-means clustering is a random-partitioning algorithm based on a center and a mean distance to the center. Moreover, our Rent's exponent calculation method is similar to the placement-based method in [39] but we experimented with smaller number of samples (j in Figure 9). In the extreme case, j can be set to 1 and Rent's exponent is approximated by p instance of a specific partitioning. Our case-study results (to be shown in Section 5) show that the approximation approach with j equal to 1 is reasonably accurate and also produces prediction results

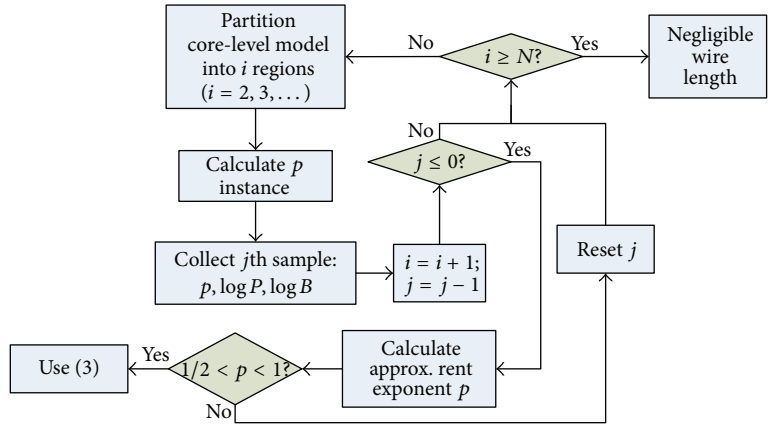


Figure 9: CMD routing estimation flow (N = total number of core instances).

faster because the linear-regression fit is skipped. However, it should be noted that users of CMD should consider using $j = 9$ or the general method (as in [52]) to calculate more accurate Rent's exponent if the prediction results start to show large errors. Finally, note that we conduct partitioning and Rent's exponent calculation based on the placed core-level model because Yang et al. [39] concluded that placement-based Rent's exponent produces more accurate wire length estimation than Rent's exponent based on netlist-partitioning.

To demonstrate the routing estimation method, consider the 8-tap FIR filter example shown in Figure 6(c). Our partitioning algorithm determines that the two-region partitioning is able to derive an acceptable approximation of Rent's exponent, with one region containing all multipliers and connected registers and the other region containing everything else. Following the rest of the estimation flow, routing delay internal to the former region is estimated to be 0.495 ns, and routing delay internal to the latter region is estimated to be 0.405 ns.

Now we explain how the same partition would not work for the same circuit on a Stratix-II S180 FPGA. Stratix-II S180 has 4 columns of DSPs and, as a result, the core instances are placed as shown in Figure 10. For the 2-region partition shown in Figure 10(a), approximate Rent's exponent is calculated to be 0.345, which is lower than 0.5. Our partitioning algorithm continues working until it finds the 3-region partition shown in Figure 10(b), which produces approximate Rent's exponent of 0.526. Using (3), the average routing delay internal to the region that contains all adders is estimated to be 0.288 ns.

Routing delay estimation does not account for the Manhattan distance across regions or the delay within core instances. To predict frequency of a core-level model, we need to consider all three factors, as discussed in the following section.

4.3. Frequency Estimation. Frequency estimation predicts the maximum clock frequency of the modeled FPGA application. CMD considers two types of delay: type-1 delay associated with the shortest Manhattan distances between all pairs of linked core instances and type-2 delay caused by regional routing. Type-1 delay is determined in the placement estimation stage. Manhattan distance can be measured by the number of routing matrices on the connecting nets. CMD uses this measurement combined with Δ to produce type-1 delay estimation. Type-2 delay is determined in routing estimation stage, as described in the previous section.

Type-1 delay is associated with a pair of linked core instances. If it is not zero, it could affect the frequency of both core instances. CMD calculates this effect by adding type-1 delay of a core-instance pair to the delay of a single pipeline stage (the inverse of pipeline frequency) of the slower core instance in the pair. Type-2 delay is associated with partitioning regions. Each region can contain many core instances. CMD calculates the resulting frequency by adding type-2 delay of a region to the delay of single pipeline stage of the slowest core instance within the region. Note that type-1 and type-2 delays are considered separately but not cumulatively. During the process of frequency estimation, CMD stores resulting frequencies from type-1 delays and the ones from type-2 delays in one sorted list from low to high. After all the computation is done, CMD uses the lowest frequency in the list as the predicted frequency. The reason why we use this approach is discussed after showing the following examples.

For the DPFP summation example in Figure 4(b), given the placement shown in Figure 6(b), CMD estimates maximum frequencies as 327 MHz for Virtex-4 LX100 and 310 MHz for Stratix-II S180. These are the same as the maximum frequencies of DPFP adders on both FPGAs because the models' type-1 and type-2 delays are zero.

For the 8-tap FIR filter example, we need to look at the two-target FPGAs separately. On Virtex-4 LX100, as shown in Figure 11, type-1 delay of the model is 0.5 ns between the registers (500 MHz) and two pairs of adders; type-2 delay of the left region is 0.495 ns and is 0.405 ns for the right region. The slowest core instance is the 378 MHz adder. Out of the three possible delay values associated with the adders, 0.5 ns

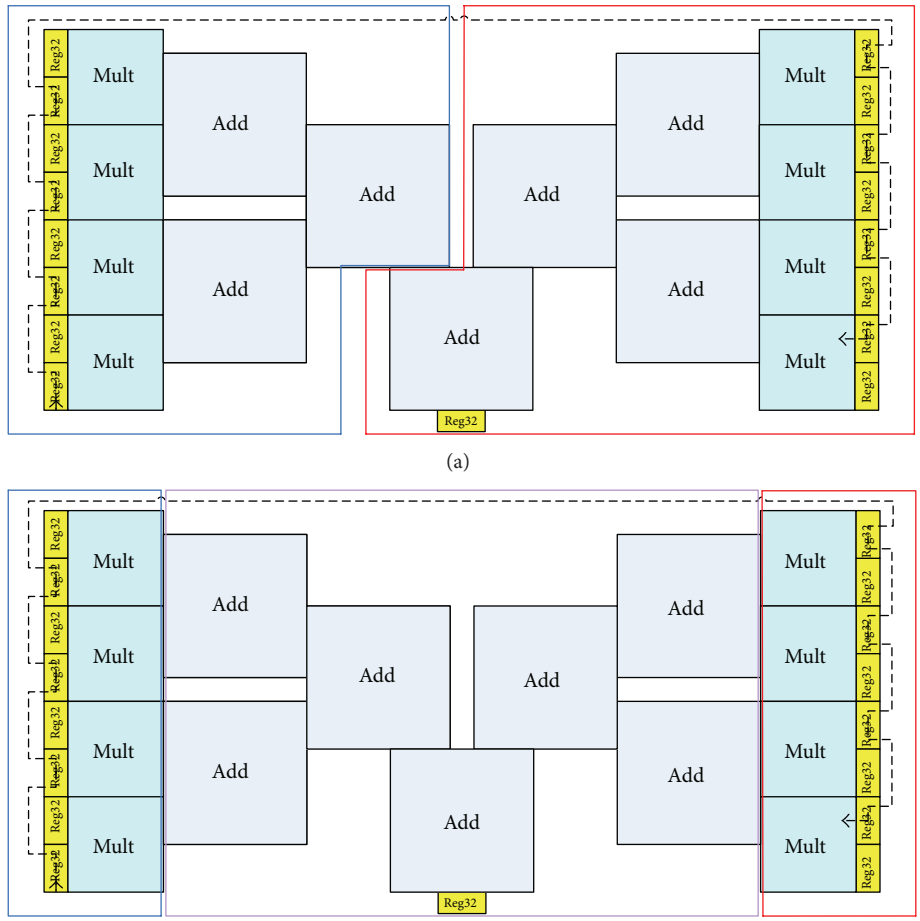


Figure 10: Placement and partitioning of 8-tap FIR filter on Stratix-II S180 into (a) two regions and (b) three regions.

produces the lowest resulting frequency. Therefore, 0.5 ns is added to the delay of the 378 MHz adder, and its frequency is adjusted to 317 MHz:

$$317\ \text{MHz} = \frac{1}{1000} \times \left(0.5\ \text{ns} + \frac{1000}{378}\ \text{MHz}\right), \quad (4)$$

which is the final predicted frequency. Similarly, on Stratix-II S180, type-1 delay of the model is 0.8 ns between the registers (450 MHz) and is 0 ns between all other pairs, given the placement in Figure 10(b). Note that although a link between the registers looks long, it is placed at long tracks available on the device and thus it only passes through two pairs of routing matrices. Type-2 delay is 0.288 ns for the region that contains all adders (350 MHz). Therefore, 0.288 ns is added to the delay of adder and its frequency is adjusted to 317 MHz:

$$317\ \text{MHz} = \frac{1}{1000} \times \left(0.288\ \text{ns} + \frac{1000}{350}\ \text{MHz}\right), \quad (5)$$

which is also the final predicted frequency of the FIR filter. Note that though a pair of registers has 0.8 ns delay in between, their adjusted frequency is higher than 317 MHz:

$$\begin{aligned} &\frac{1}{1000} \times \left(0.8\ \text{ns} + \frac{1000}{450}\ \text{MHz}\right) = 330\ \text{MHz} \\ &\quad > 317\ \text{MHz}. \end{aligned} \quad (6)$$

Two key choices are made in creating CMD's frequency prediction method. The first is that we chose the shortest Manhattan distance instead of the average distance to estimate type-1 delay, which may seem too optimistic. Our rationales are as follows: (1) it is realistic to assume that all I/O ports of a core instance are located near its block boundary; (2) it is possible that the critical path is of shortest Manhattan distance, among all paths that connect two core instances; (3) optimizations available at lower abstraction levels in vendor tools greatly overachieve predicted frequency from average-distance type-1 delays such that the average-distance estimation could be too pessimistic; and (4) the

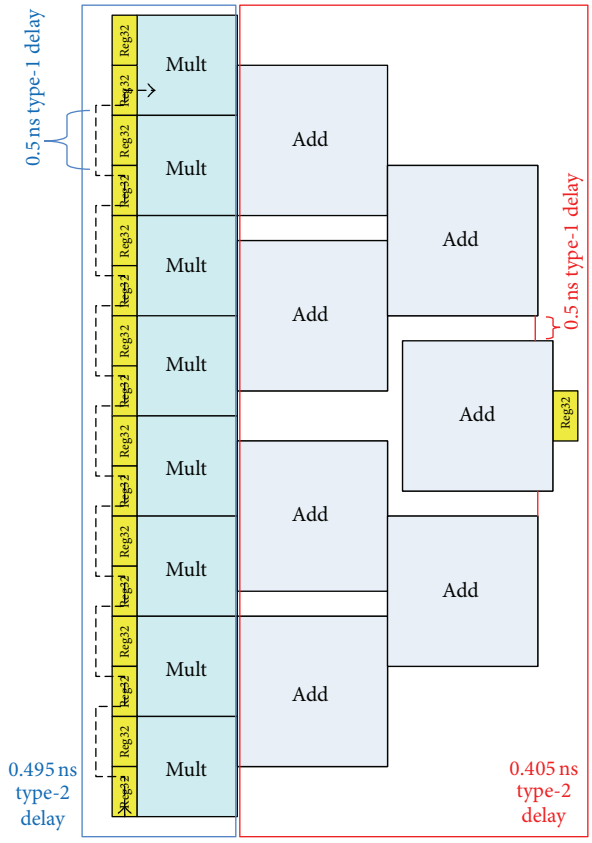


Figure 11: Type-1 and type-2 delays of 8-tap FIR filter on Virtex-4 LX100.

chances of being too optimistic by predicting from shortest-distance type-1 delay are reduced by consideration of type-2 delay.

The second key choice we made is to predict frequency from type-1 delay and type-2 delay separately and then pick the lowest predicted frequency as the final prediction. The reason for considering 2 types of delays separately is to avoid the weaknesses of either. Type-1 delay prediction does not account for routing congestion. Type-2 delay prediction can account for congestion but can only produce average routing delay within regions. By storing resulting frequencies from both types of prediction and choosing the lowest one, we aim to avoid bad prediction in the following situations: (1) all links among core instances are short except for one super-long link that is not caused by congestion; or (2) all core instances are placed close to each other but the routing congestion is severe. In the first situation, since almost all links are short and there is no congestion, the type-2 delay is short. However, the super-long link would in some cases make the entire circuit run slowly especially when it is on the critical path. If we were to use type-2 delay only, our prediction would be too optimistic. In the second situation, since all core instances are placed close to each other, their Manhattan distances are short and type-1 delays are negligible. However, it is possible that this tight placement incurs routing congestion, which is not accounted for by type-1 delay prediction. We would be too optimistic if we only considered type-1 delay in this case. According to our case studies (results to be shown in Section 5), the mixed approach using both type-1 and type-2 delays produces accurate predictions, despite the high abstraction levels of the core-level models.

Currently, CMD only handles synchronous circuits in a single clock domain. It can be extended to multiple clock domains by performing the estimation of placement and routing delays in each domain.

5. Case Studies, Results, and Discussion

In this section, case studies are presented to verify the accuracy of clock-frequency prediction. We also demonstrate how CMD's accurate frequency prediction facilitates core-level DSE through an example and discuss productivity benefits of using CMD for formulation and design.

5.1. Case-Study Devices and Tools. The target devices for case studies are DSP-rich FPGAs selected from Xilinx and Altera across three generations: Virtex-4 LX100, Virtex-6 LX130, and Virtex-7 VX330T from Xilinx and Stratix-II S180, Stratix-IV S40, and Stratix-V GSD3 from Altera. Note that the target devices from Xilinx are of speed grade 1 and the devices from Altera are of speed grade 3, because FPGAs of those speed grades are well documented. Also note that Virtex-4 and Stratix-II are old devices but they are still used for DSP applications in various fields (e.g., aerospace).

ISE and Quartus II tools are used for synthesis, placement, and routing for the verification case studies. The settings of the tools are selected to maximize the effort level to achieve the highest possible clock frequency, including global optimization, retiming, and the speed optimization strategy.

As discussed in Section 4.2, the delay of a pair of switch matrices for each case-study device is needed for routing estimation. Timing analyzers of both vendor tools are used to empirically determine this delay. The results are summarized in Table 1.

5.2. Case-Study Applications and Core-Level Models. The case-study applications are from the domain of high-performance DSP applications because of their growing popularity and common implementation on FPGAs. Evaluation of CMD in other application domains (e.g., control-intensive applications) is an interesting topic for future work as identified in Section 6.

Six case-study applications are selected: (1) the DPFP summation example with its core-level model shown in Figure 4(b); (2) the 8-tap SPFP FIR filter example with its model shown in Figure 7; (3) 4×4 matrix multiplication with its model shown in Figure 12(a); (4) an implementation of N-body simulation with model shown in Figure 12(b); (5) a 96-tap SPFP FIR filter that fills up the smallest targeted device; and (6) an infinite-impulse response (IIR) filter with SPFP data paths with model shown in Figure 12(c). Note that the 96-tap FIR is similar to the 8-tap FIR in structure, only with more cores. Also note that the IIR filter has an 8-tap feed-forward part (the same as the 8-tap FIR filter) and an 8-tap feedback part that features a long feedback data path.

Table 1: Switch matrix (SM) delay of select FPGA families.

	Virtex-4	Virtex-6	Virtex-7	Stratix-II	Stratix-IV	Stratix-V
SM delay	0.5 ns	0.35 ns	0.175 ns	0.4 ns	0.25 ns	0.175 ns

Table 2: Parameter values for selected core instances in case studies.

Parameters	Mult						Add						FLT_FIXED						FIXED_FLT					
	Xilinx			Altera			Xilinx			Altera			Xilinx			Altera			Xilinx			Altera		
Frequency (MHz)	397	429	504	400	431	605	378	450	652	375	499	714	354	503	498	240	410	639	482	550	600	445	462	653
DSP usage	4	2	2	4	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FF usage	254	183	36	1125	391	293	588	557	221	902	678	902	289	256	66	379	361	379	227	201	224	345	371	345
Latency (cycles)	10	8	8	11	11	11	13	12	12	14	14	14	6	6	6	6	6	6	7	7	7	6	6	6

First column of Xilinx is for Virtex-4 LX100, second is for Virtex-6 LX130, and third is for Virtex-7 VX330T.
First column of Altera is for Stratix-II S180, second is for Stratix-IV S40, and third is for Stratix-V GSD3.

These applications are selected for case studies for two reasons: (a) they contain common algorithmic structures that represent a wide range of DSP applications on FPGAs; and (b) they are floating-point applications that have lengthier place-and-route process than their combinational or fixed-point counterparts and hence can benefit more from CMD's clock-frequency prediction and core-level DSE.

In the core-level models of case-study applications, most core instances are provided by the vendors and thus their parameter values (e.g., maximum frequency, latency, and area) can be found in datasheets such as [44, 45]. The 8-tap FIR filter, matrix multiplication, IIR filter, and 96-tap FIR filter have multipliers and adders whose parameter values are shown in Table 2. In the core-level model of N-body simulation, for clarity of presentation, most core instances are combined into a high-level one (shown in Figure 12(b) as force calculation), except for three types: FLT_FIXED, FIXED_FLT, and Accum. FLT_FIXED converts floating-point data to fixed point and FIXED_FLT converts fixed point to floating point. Their parameter values are shown in Table 2. Accum is a fixed-point core instance, whose parameter values are derived from benchmarking its VHDL code for all target devices (e.g., maximum frequencies: 500 MHz for Virtex-4; 450 MHz for Stratix-II; 600 MHz for Virtex-6; 550 MHz for Stratix-IV; 625 MHz for Virtex-7; and 717 MHz for Stratix-V). Most core instances in the applications have registers at their I/O ports, which are not required but are often used for pipelined DSP applications on FPGAs. Thus, we choose the same style for modeling in order to achieve high prediction accuracy. Also, most core instances are highly optimized with retiming, register duplication, and so on, for the design goal of maximum frequency. CMD currently does not consider retiming within the cores, which means that register cores can be added/moved in between cores but not within cores.

Note that the case-study applications are written in VHDL by designers who are unfamiliar with CMD. Also, design decisions (such as selection of core instances) are made by the designers for maximum frequency based on their design experiences. Also note that although the case-study applications are based on VHDL, core instances can be IPs generated from HLS tools. Moreover, we focus on data paths and omit control logics (i.e., finite-state machines) in core-level models of the case-study applications, because the critical paths of these applications are the data paths. This is typical for high-performance DSP applications on FPGAs since the data paths are commonly much larger than the control logics due to replication for parallel high-throughput processing.

5.3. Verification of Clock-Frequency Prediction. To verify the accuracy of CMD's clock-frequency prediction method, for each case-study application, we compare CMD's predicted frequency against the maximum frequency of the VHDL implementation, which is determined using the exhaustive frequency-search tool described in Section 3.3 to thoroughly explore a wide range of possible frequency constraints at increments of 1 MHz. Note that this exhaustive search over a wide range of frequency constraints is only necessary for verifying frequency predictions and is not required by the prediction process. Figure 13 shows an example of the exhaustive frequency search for the 8-tap SPFP FIR filter (with constraint range [300 MHz, 330 MHz] zoomed). The necessity of searching every frequency constraint in a certain range is shown because a high frequency constraint (e.g., 319 MHz in Figure 13(a)) can still be met even if a lower one (e.g., 316–318 MHz in Figure 13(a)) is not, due to noise phenomena in FPGA routing [50]. Other case studies show similar results and hence are omitted for brevity. It is also possible to use Xilinx's Smart Explorer or Altera's Design-Space Explorer in tandem with our frequency-search tools for verification since they vary the tool settings instead of frequency constraints. However, our frequency-search tools are already configured to use the recommended tool settings to achieve the maximum frequency (as described

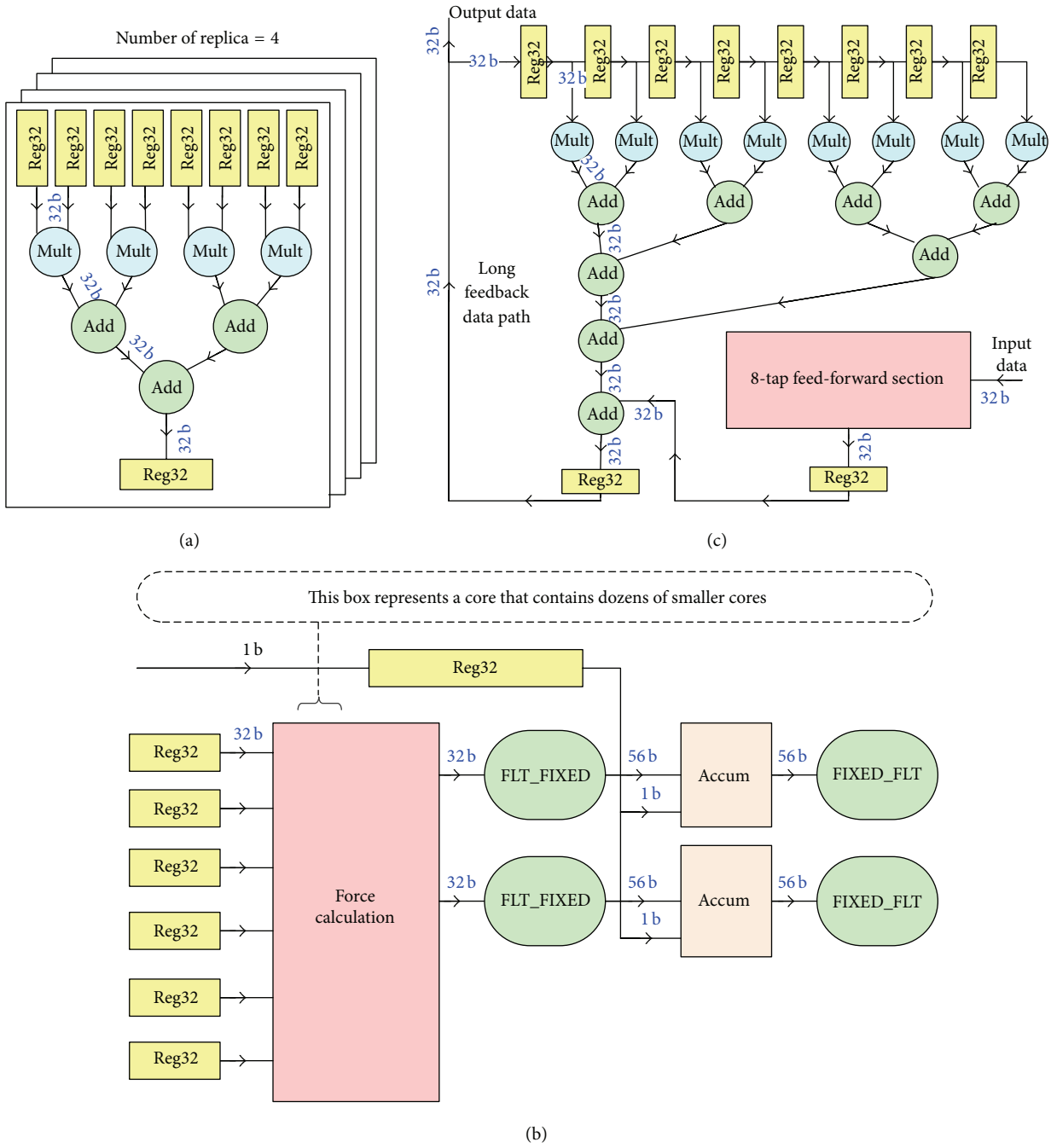


Figure 12: Core-level models of case-study applications: (a) 4 × 4 matrix multiply, (b) N-body simulation, and (c) 8-tap SPFP IIR filter.

in Section 5.1). Thus, we do not utilize Smart Explorer or Design-Space Explorer and only use the frequency-search tools for the verification case studies.

We also compare CMD's frequency prediction results with those of other high-level frequency prediction methods. Those methods, as we referenced in Section 1, are commonly used by application designers to produce crude estimates of achievable clock frequencies. The first method we compare uses the lowest frequency of all the core instances as the predicted frequency and we call this method *lowest core frequency* (LCF). The second method is the *adjusted lowest core frequency* (ALCF) that multiplies the LCF prediction by 0.75 and uses the result as the predicted frequency. Although LCF and ALCF are empirical methods commonly conducted manually by designers, we implemented them in the CMD prototype framework for faster access. The third method is the vendor-tools' *postsynthesis* frequency prediction. Note that postsynthesis prediction is not provided by Altera tools, so postsynthesis results for only Xilinx FPGAs are shown.

The verification and comparison results of all case-study applications are summarized in Figure 14. It can be seen from the results that regardless of the algorithm complexity or the

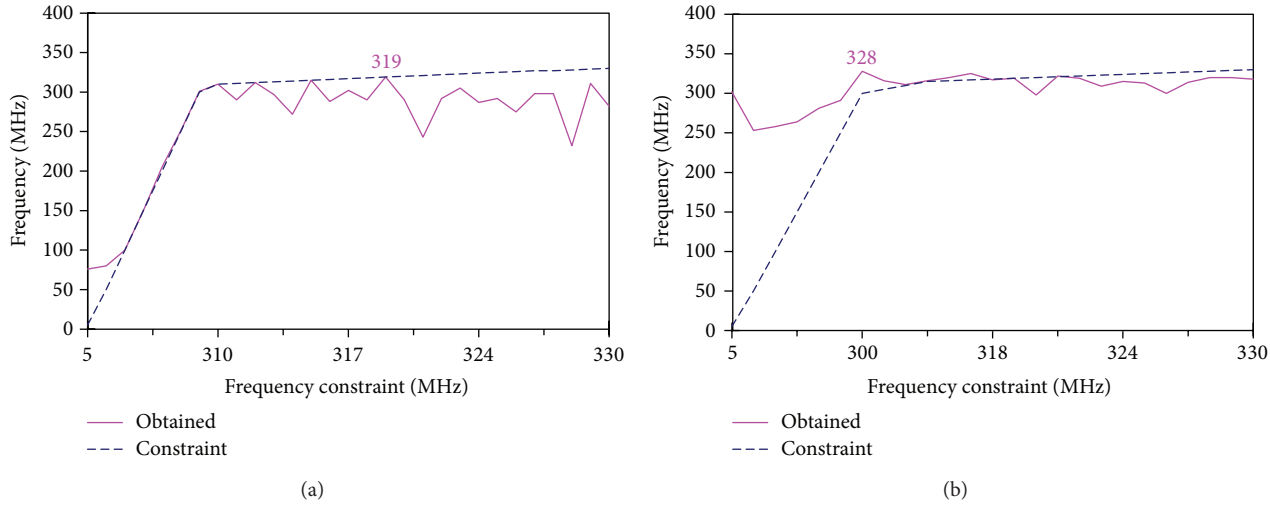


Figure 13: Frequency-constraint search of 8-tap FIR filter for (a) Virtex-4 LX100 and (b) Stratix-II S180, using ISE 12.1 and Quartus II 10.1 (zoomed in range of [300 MHz, 330 MHz]).

device family CMD provides consistently accurate frequency predictions compared with other high-level methods. In Figure 14(a), the worst error rate of CMD prediction is 9.1% for N-body simulation on Stratix-II S180. In Figure 14(b), the worst error rate is 20.4% for 96-tap FIR filter on Virtex-6 LX130. In Figure 14(c), the worst error rate is 7.4% for DPFP summation and FPFP matrix multiplication on Stratix-V GSD3. Across all case studies, the average prediction error rate of CMD is 3.6% and the maximum error rate is 20.4%; the average error rate of LCF is 15.6% with a maximum of 85.1%; the average error rate of ALCF is 17.7% with a maximum of 38.8%; the average error rate of postsynthesis is 13.9% with a maximum of 48.2%. Among all four prediction methods, CMD has the highest prediction accuracy.

In CMD's maximum error-rate case (20.4%), verified frequency is higher than predicted, which indicates that low-level optimizations are performed by vendor tools that were not captured by CMD (e.g., retiming). This difference is an expected limitation of CMD, since such optimizations are not available at CMD's abstraction level. However, we believe the limitation is acceptable since large errors were rare among all case studies (the second highest error rate is 9.1%). Plus, when such error occurs, CMD generally underestimates the maximum frequency, which we believe is still useful. Moreover, maximum error rate of 20.4% is not significantly higher than the 13% from [22], which relies on lower-level information of target device architecture than CMD. CMD is also significantly better than the 101% error from [23]. Note that CMD's prediction can also be optimistic because it is partly based on average routing delay. Finally, for the SPFP IIR filter case, CMD is able to accurately predict the maximum clock frequency for all target FPGAs. The predicted frequency is determined by the feedback data path, which is the critical path in this application. In general, DSP applications with feedback cycles are no problem for CMD because the cores are pipelined and feedback paths can be handled like any other paths.

Table 3: Resource utilization (area) prediction for N-body simulation.

	DSP	Verified DSP	FFs	Verified FFs
		Xilinx V4 LX100		
N-body	0	0	20325	18966
		Altera EP2 S180		
N-body	0	0	27121	26209

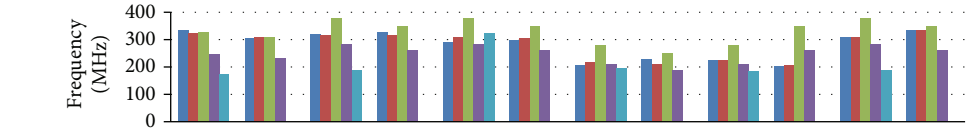
5.4. Accuracy of Area/Latency Prediction. To concisely show the accuracy of CMD's area prediction, we sample the results of one case study that has the largest prediction error (about 7%)—N-body simulation on a Virtex-4 LX100 FPGA. The results are shown in Table 3.

For latency prediction, because CMD can calculate the exact latencies for the case-study applications, the accuracy was 100%. It is because the applications have regular and static algorithmic structures (typical of DSP applications).

For both prediction methods, the accuracy is high because the parameter values of all core instances are accurately derived from datasheets or micro-benchmarking. We note that the accuracy of the latency/area prediction methods may vary for applications from other domains (e.g., control-intensive applications).

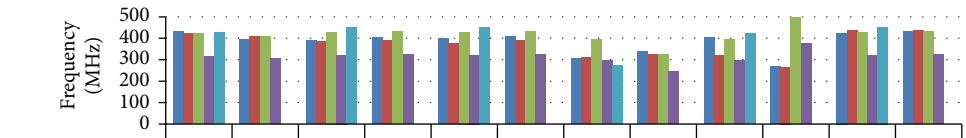
5.5. Core-Level DSE Enabled by Clock-Frequency Prediction. It is important to evaluate clock-frequency prediction in scenarios of DSE because prediction accuracy would not matter if the prediction results cannot help differentiate various design choices and select the optimal one among them. We demonstrate in this section that CMD's clock-frequency prediction enables more accurate core-level DSE than the other high-level frequency prediction methods.

To demonstrate core-level DSE, we consider an example design space for FIR filters: selection from several possible DSP usage choices for each multiplier and adder. We start



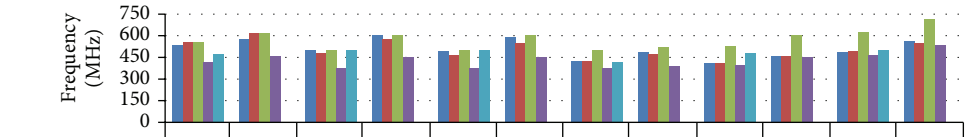
Prediction error rate	Xilinx DPFP sum	Altera DPFP sum	Xilinx 8-tap FIR	Altera 8-tap FIR	Xilinx Matrix mult	Altera Matrix mult	Xilinx N-body sim	Altera N-body sim	Xilinx 96-tap FIR	Altera 96-tap FIR	Xilinx 8-tap IIR	Altera 8-tap IIR
CMD	3.3%	1.3%	0.6%	3.4%	6.9%	1.7%	4.8%	9.1%	0.4%	1.5%	0.0%	0.3%
LCF	2.4%	1.3%	18.5%	6.7%	30.8%	17.1%	34.6%	9.1%	24.4%	73.3%	23.1%	4.8%
ALCF	26.8%	24.0%	11.1%	20.0%	1.9%	12.2%	1.0%	18.2%	6.7%	30.0%	7.7%	21.4%
Postsynthesis	48.2%	N/A	41.0%	N/A	11.2%	N/A	5.2%	N/A	18.3%	N/A	38.7%	N/A

(a) On Xilinx Virtex-4 LX100, Altera Stratix-II S180 using ISE 14.4, Quartus II 13.0sp1



Prediction error rate	Xilinx DPFP sum	Altera DPFP sum	Xilinx 8-tap FIR	Altera 8-tap FIR	Xilinx Matrix mult	Altera Matrix mult	Xilinx N-body sim	Altera N-body sim	Xilinx 96-tap FIR	Altera 96-tap FIR	Xilinx 8-tap IIR	Altera 8-tap IIR
CMD	2.1%	4.1%	1.8%	3.4%	5.5%	3.9%	1.6%	4.4%	20.4%	1.3%	2.8%	0.7%
LCF	2.1%	4.1%	10.0%	6.8%	7.5%	5.6%	29.9%	4.4%	1.7%	85.1%	1.2%	0.5%
ALCF	26.6%	21.9%	17.5%	19.9%	19.4%	20.8%	2.5%	28.3%	26.3%	38.8%	24.1%	25.3%
Postsynthesis	0.8%	N/A	15.8%	N/A	13.2%	N/A	9.6%	N/A	4.7%	N/A	6.7%	N/A

(b) On Xilinx Virtex-6 LX130, Altera Stratix-IV S40 using ISE 14.4, Quartus II 13.0sp1



Prediction error rate	Xilinx DPFP sum	Altera DPFP sum	Xilinx 8-tap FIR	Altera 8-tap FIR	Xilinx Matrix mult	Altera Matrix mult	Xilinx N-body sim	Altera N-body sim	Xilinx 96-tap FIR	Altera 96-tap FIR	Xilinx 8-tap IIR	Altera 8-tap IIR
CMD	4.1%	7.4%	4.4%	5.1%	5.5%	7.4%	0.5%	3.2%	0.2%	0.1%	1.4%	3.4%
LCF	4.1%	7.4%	0.0%	0.1%	2.9%	2.4%	18.0%	5.8%	27.4%	31.9%	29.4%	27.1%
ALCF	21.9%	19.5%	25.0%	24.9%	22.9%	23.2%	11.5%	20.6%	4.5%	1.1%	3.0%	4.7%
Postsynthesis	11.5%	N/A	0.1%	N/A	2.8%	N/A	2.0%	N/A	16.0%	N/A	4.2%	N/A

(c) On Xilinx Virtex-7 VX330T, Altera Stratix-V GSD3 using ISE 14.4, Quartus II 13.0sp1

Figure 14: Comparison of predicted frequency against verified frequency on various devices (error rates of CMD and compared high-level methods are summarized in tabular form).

with the 8-tap FIR filter on a Virtex-4 LX100 FPGA and evaluate eight key design choices as shown in Table 4(a). In this table, the first column shows DSP usage of all multipliers and the second column shows DSP usage of all adders, where *max*, *full*, *medium*, and *no* are code names in datasheets for DSP usage per core and actual numbers of used DSPs are listed after the comma. Each DSP in the table represents 18 × 18 DSP unit on FPGA. Table 4(a) also shows the predicted (using CMD and other high-level methods) and verified frequencies for each design choice. The results show

Table 4: Core-level DSE example for SPFP FIR filters (MHz).

(a) 8-tap SPFP FIR filter on Xilinx Virtex-4 LX100

For all mult	For all adder	Verified	Predicted			
			CMD	LCF	ALCF	Postsynthesis
Full, 4 DSPs	No, 0 DSPs	319	317	378	284	188
Max, 5 DSPs	No, 0 DSPs	254	257	378	284	188
Max, 5 DSPs	Full, 4 DSPs	215	219	390	293	188
No, 0 DSPs	Full, 4 DSPs	268	261	280	210	184
Medium, 1 DSP	Full, 4 DSPs	280	279	300	225	188
Full, 4 DSPs	Full, 4 DSPs	333	326	390	293	188
No, 0 DSPs	No, 0 DSPs	271	262	280	210	184
Medium, 1 DSP	No, 0 DSPs	278	279	300	225	188

(b) 48-tap SPFP FIR filter on Xilinx Virtex-4 LX100

For all mult	For all adder	Verified	Predicted			
			CMD	LCF	ALCF	Postsynthesis
Medium, 1 DSP	No, 0 DSPs	249	244	300	225	188
No, 0 DSPs	No, 0 DSPs	271	262	280	210	188

(c) 96-tap SPFP FIR filter on Altera Stratix-II S180

For all mult	For all adder	Verified	Predicted		
			CMD	LCF	ALCF
4 DSPs	0 DSPs	202	205	350	263
0 DSPs	0 DSPs	214	214	251	188

that CMD is able to differentiate all the design choices and select the one with the fastest clock, 6th design. LCF and ALCF are unable to differentiate 3rd and 6th designs. Postsynthesis is the least useful, showing almost identical predicted frequencies for the design choices.

Similar to the 8-tap FIR example, the example of 48-tap FIR filter on a Virtex-4 LX100 with two key design choices is shown in Table 4(b). Finally, the 96-tap FIR example with two key design choices is shown in Table 4(c), in which we use Stratix-II S180 instead of Virtex-4 LX100 because the on-chip DSP resource of Virtex-4 LX100 limits us to only one key design choice (no DSP for multiplier and adder). The results in Tables 4(b) and 4(c) demonstrate the advantage of CMD more clearly. For larger designs like 48-tap and 96-tap FIR, predictions of LCF and ALCF suggest the slower design; postsynthesis is indecisive. In contrast, CMD's accurate prediction is able to differentiate and select the fastest design in all cases.

Note that the example design space is much larger than the design choices that we evaluated. For example, the 8-tap FIR's example design space has 1320 different design choices because each multiplier or adder has several distinct DSP configurations and the number of configuration combinations quickly grows with 8 multipliers and 7 adders in the application. In this paper, we select eight typical design choices (varying DSP configurations for all multipliers or all adders) from the 1320 choices for the purpose of demonstration. Also note that the example design space is only a small corner of the entire design space for the simple 8-tap FIR, because of the combination explosion from design choices like latency, alternative IPs, and more (like the ones mentioned in Section 3.1). CMD can be applied if those design choices incur changes to the core-level model of the application (e.g., affecting key parameters of the core instances). However, rigorous DSE studies should be conducted on a case-by-case basis for specific applications. Also, automated placement estimation and clock-frequency prediction are needed for CMD to efficiently handle big design spaces of FPGA applications.

Core-level DSE enabled by CMD's accurate clock-frequency prediction contributes greatly to productivity improvement of FPGA application development. An anecdotal example of productivity gain from using CMD is discussed in the next section.

5.6. Productivity Improvements from CMD. The productivity improvements of using CMD are from three sources: (1) core-level DSE selects optimal design in the formulation stage; (2) code-template generation saves coding time in the design stage; and (3) the frequency constraint generation reduces the number of frequency constraints to search for the maximum frequency in the translation stage. The automated frequency-search tools in the prototype framework can work with generated frequency constraints to further reduce the time required to achieve the maximum frequency. For example, running a parallel exhaustive search on thirty servers over a range of constraints (e.g., ±15 MHz of the CMD-generated constraint) can achieve the maximum frequency, in comparable amount of time to a single run of vendor tools.

Table 5: Example of productivity gains from using CMD for 8-tap FIR filter.

	(a) 30-node cluster		(b) Single node	
Breakdown of dev. time	1	2	1	2
Formulation w/ CMD	N/A	1 hr	N/A	1 hr
Write impl. HDL code	8 hrs	6 hrs	8 hrs	6 hrs
Binary freq. search	40 hrs	N/A	40 hrs	3 hrs
Exhaustive freq. search	8 hrs	1 hr	N/A	N/A
Total design time	56 hrs	8 hrs	48 hrs	10 hrs
Result frequency	333 MHz	333 MHz	310 MHz	310 MHz

Column 1: traditional design method (no CMD or binary/exhaustive frequency-search methods).
Column 2: core-level DSE, code-template/constraints generation, and frequency-search tools.

To demonstrate the productivity improvements from using CMD's core-level DSE, code-template generation, constraint generation, and the frequency-search tools, we use the 8-tap SPFP FIR filter application as an example to compare design time of the CMD methodology against a traditional timing-closure design method, with maximum frequency as the design goal. The design time measures the period from formulation to obtaining the 8-tap SPFP FIR filter implementation with the highest achievable frequency. Coding time is estimated by the designer and frequency-search time is measured as runtime of our automated frequency-search tools. The design space is limited to the example design choices shown in Table 4(a).

The results are summarized in Table 5. Table 5(a) shows results of using 30-node cluster for the frequency-search tools. Table 5(b) shows the single-server results for comparison (hence no exhaustive search). Column 1 features our traditional design method; not using CMD for formulation, the designer wrote and debugged the VHDL code of the example application from scratch. After that, for each of the design choices, the designer modified the code and used binary search to obtain a resulting clock frequency. If a 30-node cluster is available, the exhaustive method is used to search 30 higher frequency constraints than the result frequency of binary search. Column 2 features a design method that uses CMD, including core-level DSE, code-template generation, and constraint generation.

Comparing the results between columns 1 and 2 in both Tables 5(a) and 5(b), we can see that the CMD methods achieve the same design as the traditional method in much shorter time. In particular, we observed over 7x (or 85%) design time reduction from using CMD with a 30-node cluster and 4x (or 75%) design time reduction with a single server. This improvement is comparable to HLS studies such as [54], which shows 5x (or 80%) reduction in design time for a stereo-matching application, but at the cost of slower clock frequency than the RTL design.

From the design time breakdown, we can see how much time each CMD method saved. Core-level DSE selected the optimal design during formulation in one hour and saved the time for design/translation of the other seven designs. That is why the exhaustive search time of CMD is 1/8th of that for the traditional method in Table 5(a) and the main reason why the binary search time is drastically reduced in Table 5(b). Code-template generation saved us two hours of coding time as shown in Tables 5(a) and 5(b). Constraint generation provides a good starting constraint for the frequency-search tools and thus as shown in Table 5(a) the binary search can be skipped and the same frequency as the traditional method is still achieved. As shown in the time breakdown, the most time saving comes from core-level DSE.

Note that the result frequency of 333 MHz is not achieved in Table 5(b). That is because the exhaustive search is not available and the binary search method is sensitive to the noise phenomenon of place-and-route process as discussed in Section 3.3. However, the design time reduction is still significant when compared to the traditional method on a single server. Also note that our example application is small and a single run of its placement-and-routing takes less than 1 hour. Thus, for large applications with lengthier placement-and-routing process, more time is likely to be saved by using CMD.

6. Conclusions and Future Work

FPGA usage has been limited largely due to the relatively immature nature of design tools. Previous studies have investigated formulation as a step before application development to overcome this limitation. Following this approach, several system-level formulation tools are available to help designers to conduct early DSE for FPGA applications. However, these tools assume that key parameters (e.g., clock frequency) of algorithmic components and architectural devices in the system-level models are provided by designers, which greatly limits wider use of these formulation tools.

To address this problem, we introduced a core-level modeling and design (CMD) methodology that enables the modeling of an FPGA application at an abstract level and features prediction methods that produce accurate estimates of key design parameters such as clock frequency. Through extensive case studies in the DSP application domain, it was shown that regardless of algorithm complexity or target device family CMD has the highest frequency prediction accuracy as compared to other high-level prediction methods—3.6% average prediction error versus 13.9% average error. More importantly, the case studies demonstrated that the higher accuracy allows CMD to better differentiate candidate designs and select the best one during core-level DSE. Finally, we demonstrated that core-level DSE and code-template/constraint generation can significantly reduce the total design time for an example application.

For future work, we identify several directions to expand the CMD methodology and the prototype framework. Firstly, importing architecture models of FPGAs (e.g., [51]) into CMD can help application designers to choose amenable devices. Secondly, leveraging existing RTL prediction research could reduce the need for micro-benchmarking cores. Thirdly, automated retiming within core-level model could be added to improve frequency prediction accuracy. Moreover, exploration and evaluation of larger FPGA circuits (e.g., from the VTR benchmark suite [55]) and additional

application domains (e.g., control-intensive applications) could help further validate CMD and identify additional usage of CMD. Finally, design goals other than maximum frequency could be explored and evaluated, such as maximum performance with an area constraint.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant nos. EEC-0642422 and IIP-1161022. The authors gratefully acknowledge tools and equipment provided by Xilinx and Altera that helped to make this work possible.

References

[1] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," *ACM Computing Surveys*, vol. 34, no. 2, pp. 171–210, 2002.

[2] T. El-Ghazawi, E. El-Araby, M. Huang, K. Gaj, V. Kindratenko, and D. Buell, "The promise of high-performance reconfigurable computing," *Computer*, vol. 41, no. 2, pp. 69–76, 2008.

[3] A. George, H. Lam, and G. Stitt, "Novo-G: at the forefront of scalable reconfigurable supercomputing," *Computing in Science and Engineering*, vol. 13, no. 1, Article ID 5678570, pp. 82–86, 2011.

[4] S. Choi, R. Scrofano, V. K. Prasanna, and J.-W. Jang, "Energy-efficient signal processing using FPGAs," in *Proceedings of the ACM/SIGDA 11th ACM International Symposium on Field Programmable Gate Arrays (FPGA '03)*, pp. 225–234, ACM, February 2003.

[5] J. Noguera and R. M. Badia, "Power-performance trade-offs for reconfigurable computing," in *Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '04)*, pp. 116–121, ACM, Stockholm, Sweden, September 2004.

[6] J. M. Rabaey, "Reconfigurable processing: the solution to low-power programmable DSP," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 275–278, Munich, Germany, April 1997.

[7] M. Haldar, A. Nayak, A. Choudhary, and P. Banerjee, "Parallel algorithms for FPGA placement," in *Proceedings of the 10th Great Lakes Symposium on VLSI (GLSVLSI '00)*, pp. 86–94, Chicago, Ill, USA, March 2000.

[8] J. Rose and D. Hill, "Architectural and physical design challenges for one-million gate FPGAs and beyond," in *Proceedings of the ACM 5th International Symposium on Field-Programmable Gate Arrays*, pp. 129–132, Monterey, Calif, USA, February 1997.

[9] H. Blume, H. Hubert, H. T. Feldkamper, and T. G. Noll, "Model-based exploration of the design space for heterogeneous systems on chip," in *Proceedings of the IEEE International Conference on Application- Specific Systems, Architectures, and Processors*, pp. 29–40, San Jose, Calif, USA, 2002.

[10] M. F. D. S. Oliveira, L. B. de Brisolara, L. Carro, and F. R. Wagner, "Early embedded software design space exploration using UML-based estimation," in *Proceedings of the 17th IEEE International Workshop on Rapid System Prototyping (RSP '06)*, pp. 24–32, IEEE, Chania, Greece, June 2006.

[11] C. Reardon, B. Holland, A. George, G. Stitt, and H. Lam, "RCML: an environment for estimation modeling of reconfigurable computing systems," *ACM Transactions on Embedded Computing Systems*, vol. 11, supplement 2, pp. 43:1–43:22, 2012.

[12] C. Reardon, E. Grobelny, A. D. George, and G. Wang, "A simulation framework for rapid analysis of reconfigurable computing systems," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 3, no. 4, article 25, 2010.

[13] K. Sigdel, M. Thompson, A. D. Pimentel, C. Galuzzi, and K. Bertelsy, "System-level runtime mapping exploration of reconfigurable architectures," in *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing*, Rome, Italy, May 2009.

[14] B. N. Uchevler, K. Svarstad, J. Kuper, and C. Baaij, "System-level modelling of dynamic reconfigurable designs using functional programming abstractions," in *Proceedings of the 14th International Symposium on Quality Electronic Design (ISQED '13)*, pp. 379–385, Santa Clara, Calif, USA, March 2013.

[15] B. Holland, K. Nagarajan, C. Conger, A. Jacobs, and A. D. George, "RAT: a methodology for predicting performance in application design migration to FPGAs," in *Proceedings of the 1st International Workshop on High-Performance Reconfigurable Computing Technology & Applications (HPRCTA '07)*, pp. 1–10, ACM, Reno, Nev, USA, November 2007.

[16] S. Merchant, B. Holland, C. Reardon et al., "Strategic challenges for application development productivity in reconfigurable computing," in *Proceedings of the IEEE National Aerospace and Electronics Conference (NAECON '08)*, pp. 209–218, IEEE, Dayton, Ohio, USA, July 2008.

[17] G. Wang, G. Stitt, H. Lam, and A. D. George, "A framework for core-level modeling and design of reconfigurable computing algorithms," in *Proceedings of the 3rd International Workshop on High-Performance Reconfigurable Computing Technology and Applications (HPRCTA '09)*, pp. 29–38, ACM, Portland, Ore, USA, November 2009.

[18] S. Mohanty and V. K. Prasanna, "A model-based extensible framework for efficient application design using FPGA," *ACM Transactions on Design Automation of Electronic Systems*, vol. 12, no. 2, Article ID 1230805, 2007.

[19] Xilinx System Generator for DSP User Guides, Release 10.1.1, 2008.

[20] J. Eker, J. W. Janneck, E. A. Lee et al., "Taming heterogeneity—the ptolemy approach," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 127–143, 2003.

[21] D. Strenski, "FPGA Floating Point Performance," 2007, http://www.hpcwire.com/2007/01/12/fpga_floating_point_performance/.

[22] A. Nayak, M. Haldar, A. Choudhary, and P. Banerjee, "Accurate area and delay estimators for FPGAs," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '02)*, pp. 862–869, IEEE, Paris, France, March 2002.

[23] R. Enzler, T. Jeger, D. Cottet, and G. Troster, "High-level area and performance estimation of hardware building blocks on FPGAs," in *Proceedings of the 10th International Conference on Field Programmable Logic and Its Applications (FPL '00), Villach, Austria, August 2000*, R. W. Hartenstein and H. Grünbacher, Eds., vol. 1896, pp. 525–534, Springer, 2000.

[24] M. B. Abdelhalim and S. E.-D. Habib, "Fast FPGA-based delay estimation for a novel hardware/software partitioning scheme,"

in *Proceedings of the 2nd international Design and Test Workshop (IDT '07)*, pp. 175–181, Cairo, Egypt, December 2007.

[25] R. J. Francis, J. Rose, and K. Chung, "Chortle: a technology mapping program for lookup table-based field programmable gate arrays," in *Proceedings of the 27th ACM/IEEE Design Automation Conference (DAC '90)*, pp. 613–619, Orlando, Fla, USA, June 1990.

[26] M. D. F. Schlag, P. K. Chan, and J. Kong, "Empirical evaluation of multilevel logic minimization tools for a field-programmable gate array technology," Tech. Rep., University of California, Santa Cruz, Santa Cruz, Calif, USA, 1991.

[27] XACT Development System, *Libraries Guide*, Xilinx, San Jose, Calif, USA, 1994.

[28] *XACT Xilinx Synopsys Interface FPGA User Guide*, Xilinx, San Jose, Calif, USA, 1995.

[29] L. Yan, T. Srikanthan, and N. Gang, "Area and delay estimation for FPGA implementation of coarse-grained reconfigurable architectures," in *Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '06)*, pp. 182–188, Ottawa, Canada, June 2006.

[30] P. Bjureus, M. Millberg, and A. Jantsch, "FPGA resource and timing estimation from Matlab execution traces," in *Proceedings of the 10th International Symposium on Hardware/Software Codesign (CODES '02)*, pp. 31–36, Estes Park, Colo, USA, May 2002.

[31] C. Brandolese, W. Fornaciari, and F. Salice, "An area estimation methodology for FPGA based designs at systemc-level," in *Proceedings of the 41st Design Automation Conference (DAC '04)*, pp. 129–132, San Diego, Calif, USA, June 2004.

[32] T. Jiang, X. Tang, and P. Banerjee, "Macro-models for high level area and power estimation on FPGAs," in *Proceedings of the 14th ACM Great Lakes symposium on VLSI (GLSVLSI '04)*, pp. 162–165, ACM, Boston, Mass, USA, April 2004.

[33] D. Kulkarni, W. A. Najjar, R. Rinker, and F. J. Kurdahi, "Compile-time area estimation for LUT-based FPGAs," *ACM Transactions on Design Automation of Electronic Systems*, vol. 11, no. 1, pp. 104–122, 2006.

[34] M. C. Lieu, S. K. Lam, and T. Srikanthan, "Rapid area-time estimation technique for porting C-based applications onto FPGA platforms," *Scalable Computing: Practice and Experience*, vol. 8, no. 4, pp. 359–371, 2007.

[35] J. Das, A. Lam, S. J. E. Wilton, P. H. W. Leong, and W. Luk, "An analytical model relating FPGA architecture to logic density and depth," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 12, pp. 2229–2242, 2011.

[36] J. Pistorius and M. Hutton, "Placement rent exponent calculation methods, temporal behaviour and FPGA architecture evaluation," in *Proceedings of the International Workshop on System Level Interconnect Prediction*, pp. 31–38, April 2003.

[37] B. S. Landman and R. L. Russo, "On a pin versus block relationship for partitions of logic graphs," *IEEE Transactions on Computers C*, vol. 20, no. 12, pp. 1469–1479, 1971.

[38] M. Feuer, "Connectivity of random logic," *IEEE Transactions on Computers*, vol. 31, no. 1, pp. 29–33, 1982.

[39] X. Yang, E. Bozorgzadeh, and M. Sarrafzadeh, "Wirelength estimation based on Rent exponents of partitioning and placement," in *Proceedings of the International Workshop on System-Level Interconnect Prediction (SLIP '01)*, pp. 25–31, April 2001.

[40] P. Coussy, C. Chavet, P. Bomel et al., "GAUT: a high-level synthesis tool for DSP applications," in *High-Level Synthesis: From Algorithm to Digital Circuit*, pp. 147–169, Springer, New York, NY, USA, 2008.

[41] LabVIEW FPGA Module, National Instrument, http://www.ni.com/fpga/.

[42] B. So, M. W. Hall, and P. C. Diniz, "A compiler approach to fast hardware design-space exploration in FPGA-based systems," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 165–176, Berlin, Germany, June 2002.

[43] K. B. Chehida and M. Auguin, "HW/SW partitioning approach for reconfigurable system design," in *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES '02)*, pp. 247–251, ACM, Grenoble, France, October 2002.

[44] Floating-Point Operator v4.0 Data Sheet, http://www.xilinx.com/.

[45] Floating-Point Megafunctions User Guide, http://www.altera.com/.

[46] Simulink User's Guide, http://www.mathworks.com/help/simulink/index.html.

[47] F. J. Budinsky, M. A. Finnie, J. M. Vlissides, and P. S. Yu, "Automatic code generation from design patterns," *IBM Systems Journal*, vol. 35, no. 2, pp. 151–171, 1996.

[48] J. Herrington, *Code Generation in Action*, Manning Publications, Greenwich, Conn, USA, 2003.

[49] P. Lee and M. Leone, "Optimizing ML with run-time code generation," *ACM SIGPLAN Notices*, vol. 31, no. 5, pp. 137–148, 1996.

[50] R. Y. Rubin and A. M. DeHon, "Timing-driven pathfinder pathology and remediation: quantifying and reducing delay noise in VPR-pathfinder," in *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '11)*, pp. 173–176, ACM, Monterey, Calif, USA, March 2011.

[51] V. Betz and J. Rose, "VPR: a new packing, placement and routing tool for FPGA research," in *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*, London, UK, September 1997.

[52] P. Christie and D. Stroobandt, "The interpretation and application of rent's rule," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 8, no. 6, pp. 639–648, 2000.

[53] J. Hartigan and M. Wong, "A k-means clustering algorithm," *Applied Statistics*, vol. 28, pp. 100–108, 1979.

[54] Y. Liang, K. Rupnow, Y. Li, D. Min, M. N. Do, and D. Chen, "High-level synthesis: productivity, performance, and software constraints," *Journal of Electrical and Computer Engineering*, vol. 2012, Article ID 649057, 14 pages, 2012.

[55] J. Rose, J. Luu, C. W. Yu et al., "The VTR project: architecture and CAD for FPGAs from verilog to routing," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '12)*, pp. 77–86, ACM, New York, NY, USA, February 2012.