A Framework for Core-level Modeling and Design of Reconfigurable Computing Algorithms

Gongyu Wang, Greg Stitt, Herman Lam, Alan D. George
NSF Center for High-Performance Reconfigurable Computing (CHREC)
University of Florida, Gainesville, FL 32611
Email: {wangg, gstitt, hlam, george}@chrec.org

ABSTRACT
Reconfigurable computing (RC) is rapidly becoming a vital technology for many applications, from high-performance computing to embedded systems. The inherent advantages of custom-logic hardware devices, such as the FPGA, combined with the versatility of software-driven hardware configuration often boost performance while reducing power consumption. However, compared to software design tools, the relatively immature state of RC design tools significantly limits productivity and consequently limits widespread adoption of RC. Long and tedious design-translate-execute (DTE) processes for RC applications (e.g., using RTL through HDL) must be repeated in order to meet mission requirements. Novel methods for rapid virtual prototyping and performance prediction can reduce DTE repetitions by providing fast and accurate tradeoff analysis before the design stage. This paper presents a novel core-level modeling and design (CMD) framework for RC algorithms to support fast, accurate and early design-space exploration (DSE). The framework provides support for core-level modeling, performance prediction, and rapid bridging to design and translation. Core-level modeling enables detailed DSE without the need for coding. Performance prediction, such as maximum clock frequency, supports core-level DSE and can help system-level modeling and design tools to achieve more accurate system-level DSE. Finally, core-level models can be used to generate code templates and design constraints that feed translation tools and to rapidly obtain predicted performance.

Categories and Subject Descriptors
B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids; C.4 [**Performance of Systems**]: *Design studies, Modeling techniques, Performance attributes.*

General Terms
Algorithms, Performance, Design.

HPRCTA'09, November 15, 2009, Portland, Oregon

1. INTRODUCTION
Reconfigurable computing (RC) bridges the gap between hardware and software, often providing much higher performance than software and flexibility than hardware. By providing reconfigurable computational units with a programmable interconnect, RC devices such as field-programmable gate arrays (FPGAs) enable designers to create custom circuits for particular applications, often resulting in orders of magnitude speedup [2, 5, 22, 25] and improved energy efficiency [11, 13, 18].

Despite having many advantages, FPGA usage has been limited largely due to a requirement for hardware expertise and relatively immature design tools and methodologies. A contemporary RC application-development flow typically follows a design-translate-execute (DTE) methodology (shown in Fig. 1 as the last three steps) where the designer *designs* the application by specifying RTL functionality (through HDL), *translates* (or *implements*, we use them interchangeably in this paper) that functionality into a custom circuit (through synthesis and placement and routing (PAR)), and finally *executes* the resulting circuit for the purposes of verification or performance analysis. One significant problem with this approach is that, if the design needs to be changed, the designer must re-evaluate design decisions, modify the code, re-translate the code, and re-execute the new circuit. This process can require weeks or months of increased development time [12, 19]. Even if only a single line of code is changed, iterative PAR runs may take hours or even days.

To reduce the DTE iterations, previous works [20, 21] have proposed the use of *formulation* as a step preceding design, for early DSE. As shown in Fig. 1, formulation is the first step in the FDTE model of RC application development. During formulation, an RC application is abstractly modeled and, based on mission requirements (e.g., performance/power), predictions are made through simulation or analysis [14, 20, 21] to rapidly explore the potential design space for good candidate implementations or to rule out bad ones. It has been shown [16] that effort spent in formulation is expected to have significant impact on the overall productivity by reducing the number of DTE iterations since better strategic choices in formulation mean less frequent re-design.

One common approach of current formulation tools is to model algorithmic components (e.g., tasks) and architectural devices as black boxes, whose parameters (e.g., throughput, latency, area, clock frequency) are provided by designers via experimentation or

estimation. Existing formulation approaches [14, 20, 21] abstract away many low-level details leaving DSE with only high-level choices, such as topology of communication networks, addition/removal of components, modification of parameters of an entire algorithm/device, etc. Although these highly abstract approaches are appropriate for certain situations, an approach that also includes low-level details would enable more detailed and accurate DSE. Secondly, for widespread adoption of RC technology, it is not realistic to expect RC application developers (e.g., domain scientists) to have extensive knowledge of the hardware (e.g., FPGA) and be able to provide accurate estimation of all the algorithmic and architectural parameters needed by these formulation tools (e.g., the FPGA frequency required by the formulation tool, RAT, in [14]). Without accurate parameters, prediction fidelity of these early-DSE tools is limited. Even worse, if bad parameter values are assumed by the designer, formulation may increase the number of DTE iterations. Although designers can alternatively choose to implement particular tasks to gather accurate parameter values, implementation is time-consuming and there is the risk that the expensive implementation may eventually be ruled out during DSE. Finally, overly abstract models cannot be used to produce code templates and design constraints that are necessary to bridge system-level, model-based design to detailed design and implementation.

To address these problems, we propose a core-level modeling and design (CMD) framework for RC algorithms to support accurate and fast DSE and rapid bridging to design and translation. Core-level modeling enables more detailed DSE (in contrast to task-level modeling) without the need of coding (in contrast to RTL modeling) for algorithms on RC devices. CMD can be used to complement formulation tools (e.g., [14, 21]) to enable more accurate, system-level DSE by providing accurately predicted parameters (e.g., maximum clock frequency). Also, CMD can be used to complement RC design tools like System Generator [26] by providing efficient, early DSE. Moreover, CMD provides a core-level abstraction with enough details so that automated DSE, function simulation, verification, code generation, and fast code translation can be implemented as future add-ons.

The remainder of the paper is organized as follows. Section 2 presents related research. Section 3 provides an overview of the CMD framework, focusing on the details of two key components of the framework: core-level model construction and parameter prediction. Our initial work on fast code translation (a promising future direction) is briefly described as well in Section 3. In Section 4, we present case studies to validate the parameter prediction methods. Detailed DSE of FPGA algorithms and the methods to obtain predicted frequencies are demonstrated via proof-of-concept examples. Section 5 presents conclusions and future work.

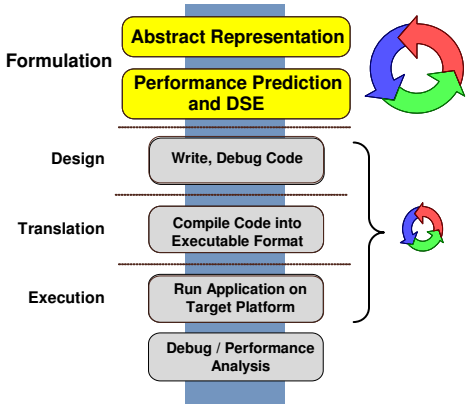


Fig. 1: FDTE Model

2. RELATED RESEARCH

In this paper, we focus on the primary challenges of the CMD framework: methodologies of core-level modeling and parameter prediction that can enable fast, accurate and early DSE.

Previous works on early DSE for RC applications [14, 20, 21] focused on system-level modeling and prediction. Those approaches generally modeled RC algorithms and components of large granularity (tasks) as black boxes with user-specified parameter values for system-level performance prediction. CMD models RC algorithms at the core level, which is more detailed than the task-level in [14, 20, 21], so that parameters of the tasks can be systematically and rapidly predicted.

Mohanty [15] proposed kernel-level modeling of FPGA algorithms, which is similar to CMD. Xilinx System Generator [26] shares the same abstract-modeling approach. A collection of modeling domains is described in [7]. CMD falls in the discrete-event and/or synchronous data flow domain. CMD improves upon these previous approaches by enabling clock-frequency prediction and early prediction-based DSE. Previous approaches were intended for simulation and design. Thus, instead of borrowing from those previous methods, we created a new modeling methodology. Future work will focus on integrating CMD with previous methods.

For parameter prediction, Strenski [24] provides a methodology to estimate the theoretical maximum Gflop/s for a given FPGA. However, this approach is generic to device and not specific to application or circuit. CMD models the algorithm of an RC application and predicts key parameters for the algorithm after its mapping to an RC device.

Two distinct methodologies to predict frequency and area for FPGA algorithms are presented in [17] and [6]. The former method works at the level of configurable logic blocks (CLBs), which is more fine-grained than CMD's core level. As a result, more detailed hardware knowledge is required to build the model and the prediction process is complicated and time-consuming. The latter method [6] works at the task level of FPGA, which is typically more coarse-grained than the core level in CMD. In this case, the accuracy for parameter prediction is limited due to greater abstraction. CMD models specific algorithm structure as well as primitive operations, but abstracts away the low-level architecture (e.g., CLB) information. In this manner, CMD can achieve prediction accuracy comparable to [17], with a prediction speed comparable to [6]. It should be noted that CMD is not a FPGA design framework (e.g., Xilinx ISE, Altera Quartus II, VPR [1]). CMD does not synthesize or place-and-route RTL components and hence does not require the detailed knowledge of FPGAs as do those design tools.

To the authors' knowledge, CMD is the first to perform RC algorithm modeling and FPGA frequency/area prediction at the core level.

3. CMD FRAMEWORK

The CMD framework is a design methodology and a set of tools that support core-level modeling, parameter prediction, code generation, and constraint generation, for the purpose of early DSE and fast implementation. A core-level model is comprised of functional cores and links that connect the cores. Functional cores can be any core primitives (as defined later), regardless of their complexity (e.g. adders, multipliers, FFT cores, etc.). Links that connect the cores represent the control/data signal paths. Shown in Fig. 2 is an overview of the framework and how it integrates within the FDTE model from Fig. 1.

Within the formulation stage, the CMD framework has two major steps: *core-level model construction* and *parameter prediction*. The inputs to core-level model construction are the *application algorithm* to be modeled (in the form of a task graph) and the characteristics of the target *RC platform*. Based on these two inputs, a designer performs core-level model construction using a CMD modeling tool. The core-level model is then used as input to *parameter prediction*. The details of core-level model construction are described in Section 3.1.

Parameter prediction takes in the core-level model and predicts values of key parameters, based on target device characteristics and optional designer inputs. DSE is performed iteratively, either manually or with automated techniques [23], between model construction and parameter prediction until design goals are met. DSE is also possible by iterating through the parameter prediction process. The parameter-prediction and DSE is detailed in Section 3.2.

DSE in the formulation stage is of little practical use if the results cannot be easily used in the detailed design and implementation. To bridge formulation into design, the CMD framework uses *code template generation*. *Constraints generation* is used to supply constraints to the translation stage. Code templates can be used to support automated or semi-automated generation of code, which combined with generated constraints can deliver an executable with frequency, area, latency (and other parameters) equivalent to those predicted during DSE. CMD can also help designers to work with legacy code by speeding up its implementation and optimizing its performance. Verification at the execution stage is performed using debug and performance analysis tools. In Section 3.3, generation of code templates, constraints, and corresponding improvements to the efficiency of *implementation* (i.e., translation) will be discussed.

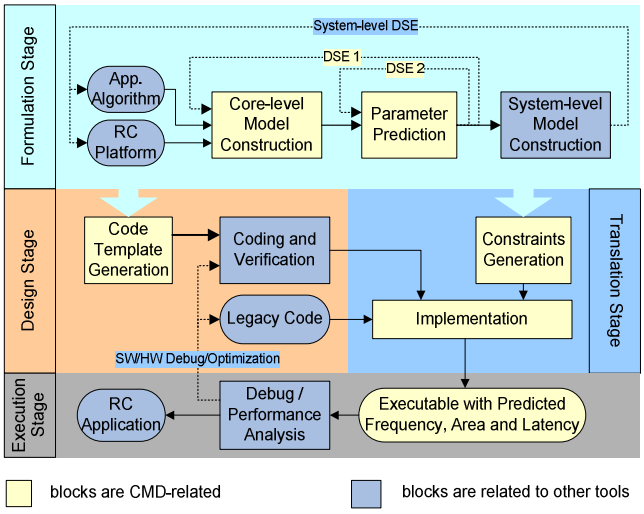
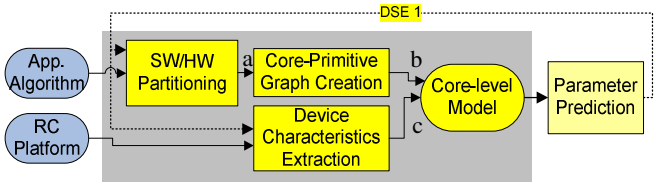


Fig. 2: CMD Framework and FDTE Model

(Dotted lines indicate iterative processes)

In the following subsections, we illustrate the CMD framework, focusing on the details of core-level model construction and parameter prediction. Note that some aspects of the framework described in this section are not yet implemented as automated tools but, for completeness of the CMD framework, they are conceptually illustrated. Also note that, although each section emphasizes FPGAs, CMD methods are also potentially applicable to other RC devices.

3.1 Core-level Model Construction



a: RC-device tasks b: core-primitive graph

c: device characteristics

Fig. 3: Core-level Model Construction

Fig. 3 illustrates the process of core-level model construction. The inputs are the application algorithm and the target RC platform. The first step of core-level model construction is SW/HW partitioning, which partitions out the tasks of the application to be implemented on RC devices, which we refer to as the RC-device tasks. Algorithms to fulfill the RC-device tasks are referred to as algorithms of RC-device tasks. CMD does not restrict the techniques used for SW/HW partitioning; it can use existing partitioning techniques (e.g., [3]) or can be specified by a designer. After partitioning, the designer models the algorithms of RC-device tasks through core-primitive graph creation. At the same time, a set of device characteristics (e.g., device infrastructure, basic operation characteristics) are extracted by designers for the target RC devices in the platform. These device characteristics are used to determine the parameter values of the core-primitive graph to produce a core-level model. These steps can potentially be performed automatically but the current CMD framework requires the designer to manually partition the algorithm and create the core-level model.

3.1.1 Core-primitive Graph

The core-primitive graph models the algorithms of RC-device tasks. Core primitives are the basic operations common to both RC devices and algorithms. Because core primitives are common

to RC devices, device vendors have often optimized these operations and provide documentation on their performance and area. Because core primitives represent basic operations that are common building blocks for algorithms, designers are able to use core primitives to intuitively describe their algorithm in a way similar to other high-level modeling tools (e.g., Simulink). For the same reason, designers can verify the functionality of their algorithms, unlike highly abstract formulation approaches.

For example, for a DSP algorithm on an FPGA, core primitives may consist of memory controllers, adders, multipliers, registers, and basic control. In many cases, a core primitive will correspond to a predefined IP core. CMD also allows for a core primitive to be defined in terms of other primitives (e.g., multiply-accumulate using an adder and multiplier primitive). Note that core primitives are not necessarily defined at one specific level of granularity. Designers can trade off prediction accuracy for reduced modeling time by selecting a level of granularity that is appropriate for a given situation. Core primitives are connected by directed links, which model data or control signals, to form a core-primitive graph. A simple example of core-primitive graph is shown in Fig. 4, which is explained in details in the next subsection.

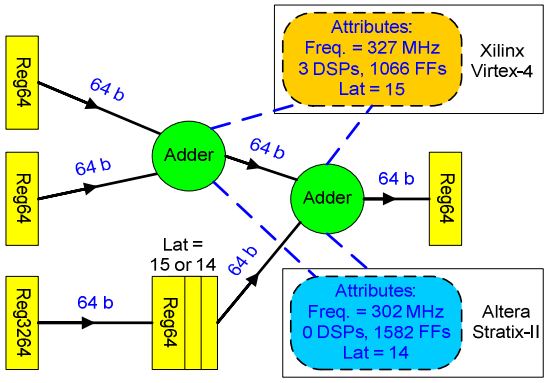


Fig. 4: Core-level Model of DPFP Summation

3.1.2 Core-level Model

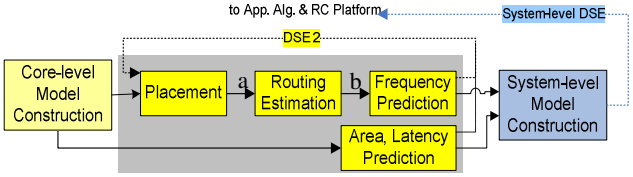
After modeling the RC-device tasks as a graph of core primitives, the designer then creates a core-level model that is parameterized by defining values for selected attributes (i.e., parameters) for each core primitive and link. In many cases, these values can simply be obtained from device datasheets, or can alternatively be determined via micro-benchmarking, which takes relatively short time because core primitives are often basic operations.

For FPGAs, CMD currently requires parameter definitions for maximum clock frequency, resource utilization (i.e., area, we use them interchangeably throughout the paper), and latency for each core primitive. Also, each link in the core-level model must have a specified bit-width. Additional attributes for core primitives can be introduced to enable new features for CMD.

CMD combines the attributes of core primitives and links with detailed characteristics of the targeted FPGA (e.g., logic cell layout, unit delay of routing channels) to form a complete core-level model, which is used by the parameter-prediction process. The attributes and detailed characteristics are both provided by the device characteristics extraction.

As an example, a core-level model for a pipelined double-precision floating-point (DPFP) summation on a Xilinx Virtex-4 LX100 or Altera Stratix-II FPGA is shown in Fig. 4. The model consists of two types of core primitives: adder and register. For each adder, the frequency, latency, and area (the number of digital-signal processing (DSP) slices and flip-flops (FF) used by the adder) are determined using the floating-point-operator datasheet [9], or provided by the Xilinx CORE Generator (or the Altera Megafunction Wizard), thus requiring little effort from a designer. All registers (labeled as Reg64 in Fig. 4, because the width of the registers is 64 bits) have latency of 1 clock cycle and they can be lumped in series to apply specific latency to signals.

3.2 Parameter Prediction



a: placed core-level model b: estimated routing delay

Fig. 5: CMD Parameter Prediction

CMD parameter prediction, shown in Fig. 5, uses the core-level model to estimate the values of key parameters for the entire algorithm of the RC-device tasks. The placement step first assigns physical locations of the core primitives on the target device. Based on the placement, CMD performs routing estimation to determine routing delays, which are used to perform frequency prediction of the algorithm. Other parameters such as area and latency are also determined. Although CMD can automate these steps using techniques described in the following sections, we currently evaluate the CMD framework by manually performing placement and routing estimation.

3.2.1 Placement

A placer in CMD can be implemented using any existing placement heuristic. As an example, simulated-annealing can be used to place core primitives by first determining the bounding boxes of all core primitives based on the resource utilization for each of them and the characteristics of the target device (e.g., resource geography). The shape of a bounding box is rectangular with changeable aspect ratio [4]. Next, the placer randomly tries different placements of all core primitives, using a cost function determined by different possible design goals.

If maximum frequency is the design goal, core primitives are placed close to each other using a cost function that minimizes the distances between linked core primitives. The distance between a pair of linked core primitives is defined as the orthogonal distance [8] between the centers of their bounding boxes (i.e., average-terminal distance). The cost function also considers routing congestion by scaling the average-terminal distance with the resource-utilization rate of the device. If the design goal is to fit all the core primitives into a certain bounding box, the placer uses

a cost function that minimizes the bounding-box size. Note that the basic element to be placed is a core primitive. Therefore, placement in CMD framework is more coarse-grained and will take much less time than placing a technology-mapped netlist.

Alternatively, designers with knowledge of the hardware architecture can manually place core primitives on the target device. In this way, if the resulting performance/area is not satisfactory, designers can come back and alter the placement (DSE 2, as shown in Fig. 2 and Fig. 5).

In this paper, maximum frequency is chosen as the design goal. For the case studies presented in Section 4, we use manual placement because detailed knowledge of commercial FPGAs is not available for us to implement the CMD placer and FPGA-vendor tools can aid manual placement of the core primitives.

In the example shown in Fig. 4, using the DSPs of the target FPGA (Virtex-4 LX100), we place the two adder core primitives over the FPGA's DSP columns. We then modify the adders' aspect ratios so that the distance between them is as short as possible. For Stratix-II S180, adders consume no DSP, which makes it easier to manipulate their location and distance.

3.2.2 Routing Estimation

Routing estimation determines routing delays for a specified placement. Given the knowledge of the target device's unit-wire delay and the previously determined average-terminal distances, CMD calculates the routing delay between each pair of linked core primitives. Moreover, based on the work of Feuer [8], routing congestion is considered in terms of total utilization rate of the device.

According to Feuer, the average internal-wire length of a circuit increases if more components are added to it. This relationship is given by Eq. 20 in [8]. The average internal-wire length is not needed for CMD routing estimation but the authors find that it is a good indicator of the routing congestion for a core-level model. Experiments have confirmed the equation's relation to routing congestion and calibrated it to scale the routing delays. The details of these experiments are outside the scope of this paper.

The assumption that the input/output ports of a core primitive are averaged at the center of its bounding box is made in the placement step and inherited by the routing-estimation step. CMD could eliminate the need for this assumption by modeling the locations of core primitives' input/output ports, which may involve trade-off study of quality and time and therefore is deferred to future work. Routing estimation for the example in Fig. 4 is presented in the next subsection.

3.2.3 Frequency Prediction

Frequency prediction estimates the clock frequency of the modeled algorithm. For our design goal, CMD initially predicts maximum frequency of the algorithm of RC-device tasks as the lowest frequency among all core primitives in the algorithm's core-level model. Then, CMD adds the routing delays between each pair of linked core primitives to their critical-path delays (inverse of their frequencies), resulting in adjusted frequencies of the core primitives. The routing delays are scaled according to the device utilization to account for routing congestion. Finally, CMD determines the predicted frequency as the lowest adjusted frequency among all core primitives.

For the example in Fig. 4, given the previously described placement, CMD estimates maximum frequencies of 327 MHz for the Virtex-4 LX100and 302MHz for the Stratix-II S180, which is the lowest frequency among all the core primitives for each FPGA. Although routing delays will often affect the maximum frequency, it has no effect in this case due to the lack of routing congestion resulting from low resource utilization. (under 10% of either Virtex-4 LX100 or Stratix-II S180). Congestion becomes significant for algorithms with higher device utilizations, as shown in the case studies in Section 4.

Currently, CMD assumes a single clock domain, but the method can be extended to handle multiple clock domains by performing the analysis of core primitives and routing delays in each domain.

3.2.4 Area, Latency Prediction

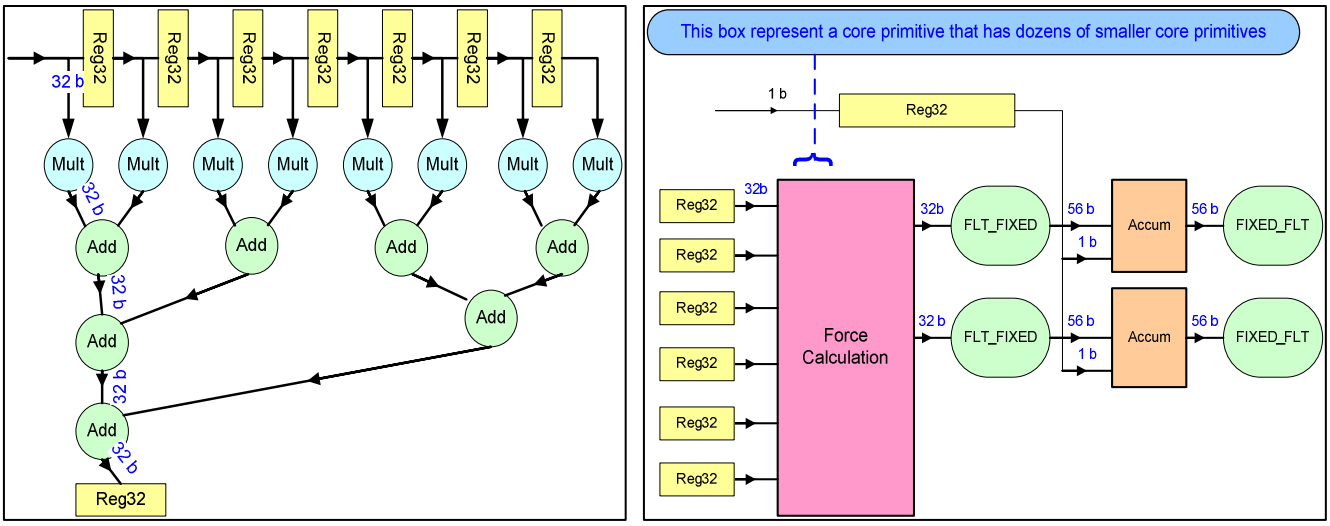
For latency prediction, latencies of core primitives along each path, from input ports to output ports, are summed to produce the latency of that path. Similarly, area prediction sums the resource utilizations of each core primitive in the core-level model to determine overall resource utilization, assuming core primitives do not share resources. Given accurate attribute values of core primitives, the accuracies of latency and area prediction can be very high. Although the flow in Fig. 5 shows only the prediction of typical parameters such as frequency, area and latency, CMD can be expanded to include other parameters and metrics.

3.3 Constraint Generation and Translation

As mentioned, DSE in the formulation stage is wasted if the results cannot be easily used in design and obtained in implementation. The CMD framework uses code-template generation and design-constraint generation to tunnel early DSE results into design and translation stages. Code template is generated from the core-level model of the algorithm of RC-device tasks to inherit design choices, such as the types of core primitives, the links between core primitives, etc., which have been determined during early DSE. However, code generation, especially automatic code generation, is a complicated problem and is deferred to future work. Constraint generation for FPGA is described in detail as follows.

For FPGAs, constraints generation creates frequency and area constraints that can help designers in translation stage by reducing the number of translation iterations normally required to manually explore different constraints. Area constraints can be determined from the placement of core primitives, which is shown via a proof-of-concept case study in Section 4.4.

The generated frequency constraint is determined by the predicted frequency plus some "slack" to account for improved performance from RTL PAR. Due to the uncertainty of the PAR process, this slack is impossible to predict. For example, a PAR tool may not find a solution for a frequency constraint of 250 MHz, but surprisingly may find a solution for a higher constraint of 260 MHz. For this reason, automatically exploring all possible



(a) (b)

Fig. 6: Core-level Models of Case Studies: (a) 8-tap SPFP-FIR Filter and (b) N-body Simulation

Table 1. Parameter Values for Selected Core Primitives in FIR filter and N-body Simulation Case Studies

Parameters	FIR filter				N-body Simulation			
	Mult		Add		FLT_FIXED		FIXED_FLT	
	Virtex-4	Stratix-II	Virtex-4	Stratix-II	Virtex-4	Stratix-II	Virtex-4	Stratix-II
Frequency (MHz)	397	400	378	375	305	275	321	275
DSP usage	4	0	0	0	0	0	0	0
FF usage	254	1125	588	902	289	379	227	345
Latency (cycles)	10	11	13	14	6	6	6	6

frequency constraints beyond the predicted one is vital to achieve the maximum frequency using certain FPGA PAR tools. Also, it saves time for designers to start searching from a predicted frequency rather than a speculated one.

CMD performs the frequency search using two possible methods. One method uses a binary search to reduce the total number of PAR iterations. The second method uses a cluster-based search that performs numerous PAR executions in parallel. For both techniques, the speedup compared to manual timing closure is dramatic. For the case study of N-body simulation, as described in Section 4, the time was reduced from days to less than an hour.

CMD can help designers that work with legacy code in three ways. Our cluster-based frequency search scripts can take in legacy code and produce better frequency in a relatively short period of time. Secondly, the core-level model of the legacy code can be created, possibly using automated commercial tools like Code2Graphics of Active-HDL. The resulting core-level model can then be used to not only generate constraints but also to optimize the performance via fast DSE. Finally, the core-level model represents a condensed yet precise means of documentation for the algorithm.

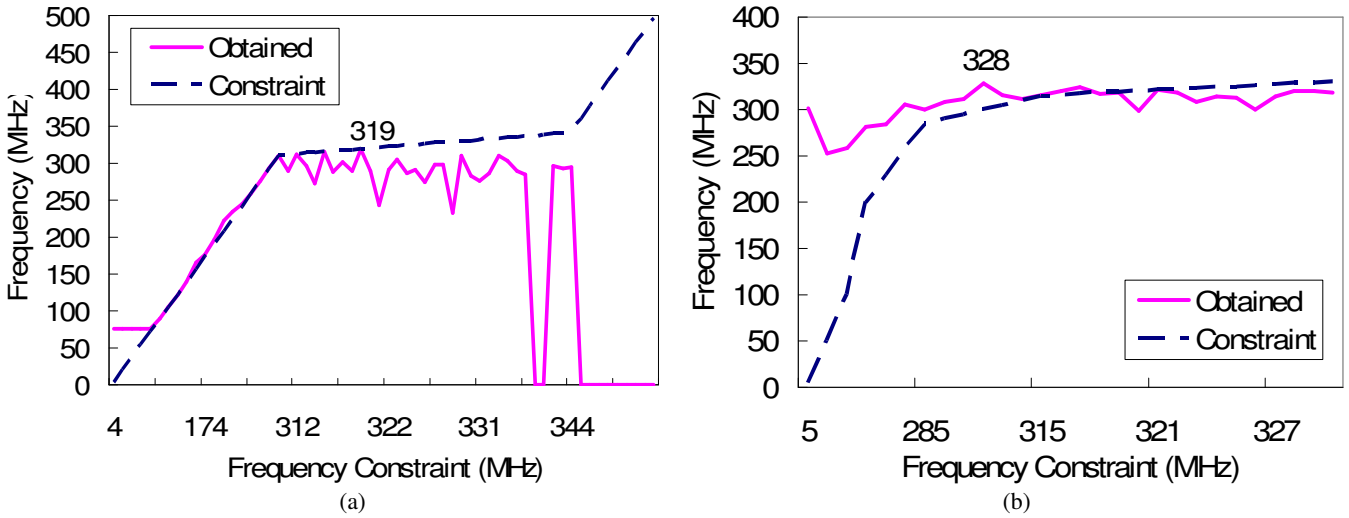
4. CASE STUDIES AND RESULTS

In this section, case studies are presented to showcase the accuracy of parameter prediction and productivity advantage of the CMD framework. The accuracy is verified through a comparison between predicted and verified frequencies. The productivity advantage is demonstrated in two scenarios: core-level DSE and fast code implementation. Note that additional productivity is gained from using a more abstract block diagram to design because block diagrams are more natural than low-level program code for many designers, especially those that are not experts of VHDL or Verilog.

To test our case studies, we use the Xilinx Virtex-4 LX100 and Altera Stratix-II S180 FPGAs as the target RC devices. ISE 9.2i and Quartus II 9.0 are used for synthesis and PAR. The settings of the tools are selected to maximize effort levels to achieve the highest possible clock frequency. To derive the routing delay of unit-length routing wire, the timing analyzers of both tools are used to empirically determine the average routing delay of a single switch matrix (0.5ns for Virtex-4 LX100 and 0.4ns for Stratix-II S180).

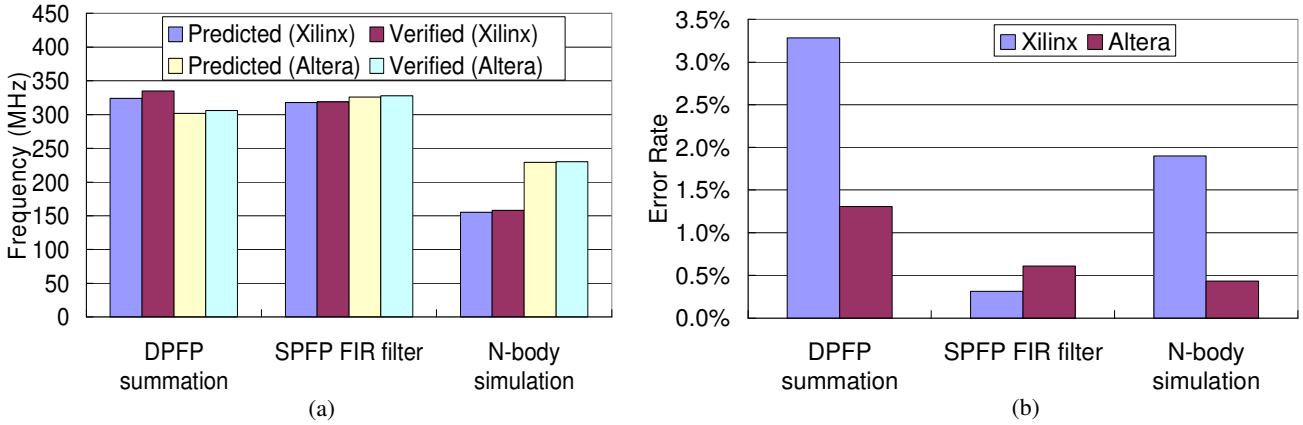
4.1 Core-level Models for the Case Studies

Three case studies were performed. The first case study (DPFP summation) uses the core-level model shown in Fig. 4. The second case study features an algorithm of an 8-tap single-precision floating-point (SPFP) FIR filter and the third case study features an algorithm of N-body simulation. The core-level models of the latter two case studies are shown in Fig. 6. The



(a) (b)

Fig. 7: Frequency Search Result of FIR-filter Case Study for (a) Virtex-4 LX100 and (b) Stratix-II S180



(a) (b)

Fig. 8: Comparison of Predicted Frequency against Verified Frequency by (a) Frequency and (b) Error Rate

attribute values of their core primitives are summarized in Table 1. For the sake of illustration, most core primitives in the core-level model of N-body simulation are combined into one high-level block (shown as *Force Calculation)* and attribute values of only two sample core primitives (FLT_FIXED converts the floating-point data to fixed point and FIXED_FLT converts the fixed-point data to floating point) are shown in Table 1.

These case studies were selected due to their nature of being floating-point DSP algorithms, whose clock frequencies are generally difficult to predict. In these case studies, the core-level models did not consider all the control logic, such as finite-state machines, which is left as future work. However, a finite-state machine can be modeled as one additional core. The N-body simulation has a fixed-point core primitive (Accum), whose characteristics were derived from benchmarking through its VHDL code. The VHDL code of Accum has 42 lines and the benchmarking took less than 5 minutes to derive a maximum frequency of 500MHz, which is the upper limit of both Virtex-4 LX100 and Stratix-II S180.

Although CMD can potentially create code from core-level models automatically, in the current CMD framework, we manually created the code for the case studies.

4.2 Verification of Parameter Prediction

To confirm the accuracy of CMD modeling and the parameter predictions for the algorithms of RC-device tasks, we compared the predicted frequencies (derived using manual placement) with the maximum frequency determined using the FPGA-vendor tools (through PAR on the target FPGAs). To determine the maximum frequency, we use the constraint-generation scripts described in Section 3.3 to further explore possible frequency constraints at increments of 1 MHz and mark the highest. Fig. 7 illustrates the frequency-constraint search for FIR filter. Other case studies are omitted for brevity. Note that a high frequency constraint (e.g. 319 MHz in Fig.7 (a)) can still be met even if a lower one (e.g. 316-318 MHz in Fig.7 (a)) is not.

The predicted and verified frequencies of all case studies are summarized in Fig. 8. It can be seen from the results that,

regardless of the algorithm complexity, CMD provides accurate frequency prediction, which lends itself to core-level DSE and also system-level DSE via integration with system-level tools. The worst error rate is 3.25% for DPFP summation.

For resource usage prediction, the results for all case studies are almost 100% accurate, even for the largest case study (N-body simulation) that utilizes nearly half of the FPGA. Note that for these examples, CMD can calculate the exact latency, and thus yields 100% accuracy. For prediction of both resource usage and latency, the accuracy is high due to the fact that corresponding attribute values of all primitives are accurate.

4.3 Core-level DSE

To demonstrate core-level DSE, we consider several possible DSP-usage combinations (maximum usage means 5 DSPs; full usage means 4 DSPs; medium usage means 1 DSP) of multipliers and adders for FIR filter on Virtex-4 LX100. The predicted and verified frequencies for each combination are summarized in Table 2. Note that the frequencies are correctly predicted by CMD for all but one combination—full usage of DSP for all multipliers and full usage of DSP for all adders (the grayed row in Table 2). The reason for this incorrect prediction is an inefficient manual placement, which illustrates that the performance from placement assignment can be limited by the designer's experiences.

Early DSE with CMD, as shown in Table 2, determines that the first combination (full usage of DSP for all multipliers and no usage of DSP for all adders) yields the best predicted frequency—317 MHz, which is acceptable when compared to the best verified frequency of the best combination (full for all multipliers and full for all adders)—333 MHz.

Considering the predicted frequencies in Table 2 take minutes to derive and the verified ones take days, it is a good tradeoff. In this way, CMD can increase the DSE productivity enormously. Also, the early DSE results can help the designer to determine if an FPGA design can meet external timing requirements very fast because neither coding nor PAR were needed.

DSP usage of floating-point cores of Altera FPGAs is usually not configurable. However, early DSE of other design options that are important to an application can be performed in a similar manner on Altera FPGAs. For example, the latency option in Altera's floating-point cores can be varied to achieve different maximum frequencies. We defer exploration of those options to future work.

Table 2. DSE for FIR Case Study

For all Mult	For all adder	Predicted	Verified
full	no	317	319
max	no	257	254
max	full	219	215
no	full	261	268
medium	full	279	280
full	full	246	333
no	no	262	271
medium	no	279	278

4.4 Frequency with Generated Constraints

As described in Section 3.3, the frequency constraint is generated by adding some "slack" to the predicted frequency. The area constraint is generated by translating our placement estimation or assignment to a format suitable for use of CAD tools. Fig. 9 shows an example of the area constraint, using the ISE area-constraint editor, for the FIR-filter case study on Virtex-4 LX100. The eight rectangles on the left of the figure are the bounding boxes of the eight multipliers in the core-level model of FIR filter. The other seven rectangles in the figure are the bounding boxes of the seven adders in the model. Each rectangle is linked to its adjacent ones. It can be seen that the choice of layout and shapes

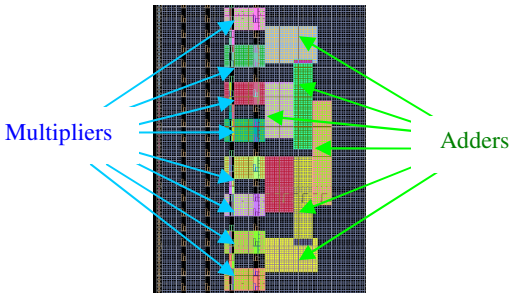
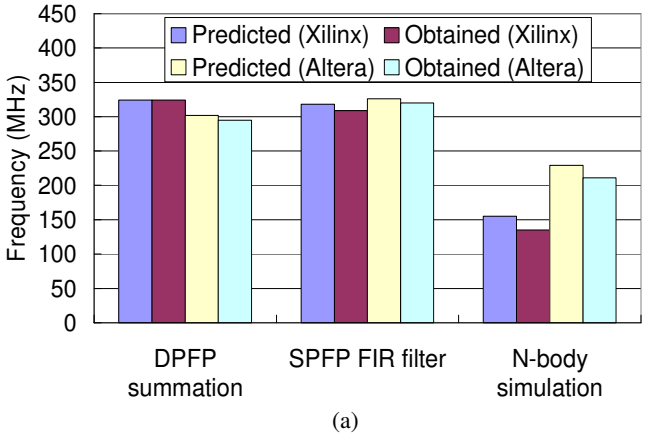
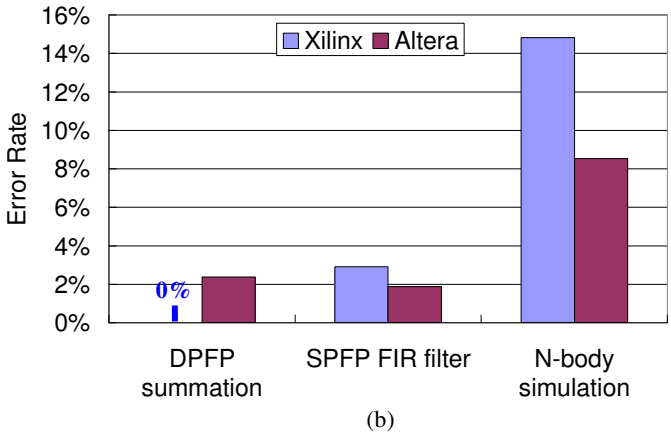


Fig. 9 Area Constraint for FIR Case Study



(a)



(b)

Fig. 10: Comparison of Predicted Frequency against Obtained Frequency by (a) Frequency and (b) Error Rate

of the rectangles makes the average-terminal distances as short as possible. The same approach should work for Altera FPGAs as well and it is an immediate future work of the authors.

Using the frequency constraint and area constraint generated by CMD, a reduced number of CAD tool iterations is required to obtain frequencies that are comparable to the predicted ones. The obtained frequencies for all case studies using this approach are shown in Fig. 10 (using both frequency and area constraints for Virtex-4 LX100 and using just frequency constraint for Stratix-II S180). Note that the results are derived by a single iteration of vendor tools for FIR filter and N-body simulation, which is to be compared with dozens of iterations when not using the generated constraints to derive the same resulting frequency.

Our automated scripts can be used to work with the generated frequency constraint to derive even better result than the obtained in a relatively short period of time. For example, the search for the FIR filter on a Virtex-4 LX100 took approximately 30 minutes to achieve 319 MHz (higher than the obtained 309MHz, as shown in Fig. 10), in contrast to several hours of using ISE serially to obtain about the same frequency (319MHz).

5. CONCLUSIONS

The usage of reconfigurable computing has been limited largely due to a common requirement of application designers for hardware expertise and the relatively immature nature of design tools and methodologies. A FDTE model of RC application development has been proposed [20, 21] in attempt to overcome this limitation and following this approach, several system-level formulation tools are available to help designers carry out early DSE. However, these tools assume that key parameters of algorithmic components and architectural devices in the system-level models are provided by designers, which greatly limits the wide use of formulation tools.

To address this vital problem, we created a core-level modeling and design (CMD) framework that support both fast, accurate DSE (of finer granularity than system-level tools) and rapid design and implementation (not available from system-level tools) for reconfigurable computing algorithms. This framework provides a core-level abstraction so that enough details are modeled to enable accurate parameter prediction, functional verification and fine-grained DSE. Also, the CMD framework produces code templates and design constraints, thus bridging formulated abstract models to concrete design and implementation.

Three case studies featuring algorithms of various complexities on two types of FPGAs have been performed to verify the proposed CMD framework and to demonstrate the usage scenarios. Frequency prediction for all case studies is very accurate with a maximum error rate of 3.25% and it takes minutes to predict frequency in contrast to hours and even days when using vendor tools. Thus, early DSE based on CMD framework can quickly determine good and bad designs before writing or implementing any code. Moreover, with generated constraints, the number of iterations of the translate-execution process is drastically reduced (from dozens to one for the case studies) to obtain performance that is close to the predicted, with 14% being the maximum error rate.

Several directions are identified to expand the CMD framework in future work. Importing more detailed device-architecture information into CMD can enable automatic core-primitive placement and routing (as described in Sections 3.2.1 and 3.2.2), which can further improve the productivity of designers in the formulation stage. Building virtual architecture models of RC devices (e.g. the architecture model used in VPR [1]) into CMD can enable formulation for device architectures to help FPGA device engineers to increase productivity and help FPGA application developers to choose amenable devices. Automatic code generation (as described in Section 3.3) from core-level models can provide an effective bridge from formulation into design.

6. ACKNOWLEDGMENTS

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

7. REFERENCES

[1] V. Betz , J. Rose, “VPR: A new packing, placement and routing tool for FPGA research,” Proc. of the 7th International Workshop on Field-Programmable Logic and Applications, London, UK, September 1997.

[2] K. Compton and S. Hauck, “Reconfigurable Computing: a Survey of System and Software,” ACM Computing Surveys, pp. 171-210, vol. 34, no. 2, June 2002.

[3] K. B. Chehida, and M. Auguin, “HW / SW partitioning approach for reconfigurable system design,” Proc. of the international Conference on Compilers, Architecture, and Synthesis For Embedded Systems, Grenoble, France, October 2002.

[4] C. Conger, A. Gordon-Ross, and A. George, "FPGA Design Framework for Partial Run-Time Reconfiguration," Proc. of 2008 International Conference on Engineering of Reconfigurable Systems and Algorithms, Las Vegas, Nevada, July 2008.

[5] A. Dollas, E. Sotiriades, and A. Emmanouelides, “Architecture and Design of GE1, a FCCM for Golomb Ruler Derivation,” Proc. of the IEEE Symposium on Field-Programmable Custom Computing Machines, Napa, California, April 1998.

[6] R. Enzler, T. Jeger, D. Cottet, and G. Troster, “High-Level Area and Performance Estimation of Hardware Building Blocks on FPGAs,” In R.W. Hartenstein and H. Gr¨unbacher (Eds.) FPL 2000, volume 1896, pages 525–534, Springer, 2000.

[7] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, Y. Xiong, "Taming Heterogeneity---the Ptolemy Approach," Proc. of the IEEE, volume.91(1), January 2003.

[8] M. Feuer, "Connectivity of Random Logic," IEEE Transactions on Computers, vol.C-31, no.1, pp.29-33, January 1982.

[9] Floating-Point Operator v4.0 Data Sheet, www.xilinx.com

[10] Floating-Point Megafunctions User Guide, www. altera.com

[11] J. Goodman, A. P. Chandrakasan, "An energy-efficient reconfigurable public-key cryptography processor," IEEE

Journal of Solid-State Circuits, vol.36, no.11, pp.1808-1820, Nov 2001.

[12] M. Haldar, A. Nayak, A. Choudhary, and P. Banerjee, “Parallel algorithms for FPGA placement” Proc. of the 10th Great Lakes Symposium on VLSI, Chicago, Illinois, March 2000.

[13] P. M. Heysters, G. K. Rauwerda, and G. J.M. Smit, “Implementation of a HiperLAN/2 receiver on the reconfigurable montium architecture,” Proc. of the 11th Reconfigurable Architectures Workshop, Santa Fe, USA, Apirl 2004.

[14] B. Holland, K. Nagarajan, C. Conger, A. Jacobs, and A. George, "RAT: A Methodology for Predicting Performance in Application Design Migration to FPGAs," Proc. of High-Performance Reconfigurable Computing Technologies & Applications Workshop at SC'07, Reno, NV, November 2007.

[15] S. Mohanty, and V. K. Prasanna, “A model-based extensible framework for efficient application design using FPGA,”ACM Trans. on Design Automation of Electronic Systems, April. 2007.

[16] S. Merchant, B. Holland, C. Reardon, A. George, H. Lam, G. Stitt, M. Smith, N. Alam, I. Gonzalez, E. El-Araby, P. Saha, T. El-Ghazawi, H. Simmler, "Strategic Challenges for Application Development Productivity in Reconfigurable Computing," Proc. of National Aerospace & Electronics Conference, Dayton, OH, July 2008.

[17] A. Nayak, M. Haldar, A. Choudhary, P. Banerjee, "Accurate area and delay estimators for FPGAs," Proc. of Design, Automation and Test in Europe Conference and Exhibition, April 2002.

[18] J. M. Rabaey, "Reconfigurable processing: the solution to low-power programmable DSP," IEEE International Conference on Acoustics, Speech, and Signal Processing, Munich, Germany, April 1997.

[19] J. Rose, and D. Hill, “Architectural and physical design challenges for one-million gate FPGAs and beyond,” Proc. of the ACM Fifth international Symposium on Field-Programmable Gate Arrays, Monterey, California, February 1997.

[20] C. Reardon, B. Holland, A. George, G. Stitt, H. Lam, "RCML: An Abstract Modeling Language for Design-Space Exploration in Reconfigurable Computing," submitted to IEEE Symposium on Application Specific Processors, San Francisco, California, July 2009.

[21] C. Reardon, E. Grobelny, A. George, and G. Wang, “A Simulation Framework for Rapid Analysis of Reconfigurable Computing Systems,” ACM Trans. on Reconfigurable Technologies and Systems (TRETS), to appear.

[22] E. Sotiriades, A. Dollas, and P. Athanas, “Hardware-Software Codesign and Parallel Implementation of a Golomb Ruler Derivation Engine,” Proc. of the 2000 IEEE Symposium on Field-Programmable Custom Computing Machines, Napa, California ,April 2000.

[23] B. So, M. Hall, and P. Diniz, “A compiler approach to fast hardware design-space exploration in FPGA-based systems,” Proc. of the 2002 ACM SIGPLAN Conference on Programming Language Design and Implementation, Berlin, Germany, June 2002.

[24] D. Strenski, “FPGA Floating Point Performance – a pencil and paper evaluation,” HPC Wire, January 2007, http://www.hpcwire.com/hpc/1195762.html

[25] M. Weinhardt, and W. Luk, “Pipeline Vectorization for Reconfigurable Systems,” Proc. of the 1999 IEEE Symposium on Field-Programmable Custom Computing Machines, Napa, California, Apirl 1999.

[26] Xilinx System Generator for DSP User Guides, Release 10.1.1, April, 2008. http://www.xilinx.com/ise/optional_prod/system_generator.htm