Modelling Reconfigurable Systems in Event Driven Simulation

Kuen Hung Tsoi, Tobias Becker and Wayne Luk
Department of Computing, Imperial College London
{khtsoi, tbecker, wl}@doc.ic.ac.uk

ABSTRACT

Reconfigurable platforms allow hardware developers to customise their designs for specific applications. However, their adoption involves challenges in understanding and estimating the impact of various design parameters and approaches. This paper proposes a unified framework to model behaviour of reconfigurable systems using an event driven simulation approach. This provides an abstract yet informative method to capture both analytical relationships and empirical parameters of reconfigurable systems. It can be used to help making design decisions or verifying analytical models. We apply this approach to three models of reconfigurable applications to estimate the communication efficiency of networked clusters, and the performance and energy efficiency of runtime reconfigurable designs for software-defined radio and for option pricing in finance. The results show that, through this simulation framework, we can verify the accuracy of analytical models and also obtain practical information that is not provided by analytical models.

1. INTRODUCTION

Reconfigurable computing platforms usually employ Field Programmable Gate Array (FPGA) devices as the main processing elements. With high flexibility for mapping applications to reconfigurable fabric, FPGA platforms have become a key part in many applications ranging from battery powered embedded systems to throughput oriented accelerators in data centres. The flexibility and efficiency of FPGA systems come with the high cost in system development. It usually takes much longer time to develop when compared to the software approach. Hence, careful planning in early stages of the design flow is essential to reduce development cost for reconfigurable platforms.

For simple designs, the expected performance and the best design approach can be obtained based on developers' experience. When the design complexity grows beyond the scope of manual analysis, abstract models are usually used to facilitate these tasks. A model includes key parameters from both the algorithm and the platform as well as the mathematical relationship between them. Once the model is constructed, the behaviour of the system can be studied before detailed implementation.

Abstraction creates both benefits and drawbacks in modelling. It helps simplifying the system for a better understanding. However, details affecting realistic system behaviour may be omitted. More importantly, most models are based on static analysis of the system and can only reflect the system behaviour under certain assumptions. For example, the Xilinx XPower Analyser [1] requires information about the toggling frequency of the registers. In reality, this information is highly dependent on input data and may be changed over time. Pure analytical models may predict results under pre-defined assumptions but can deviate from specific scenarios.

It is important to have a way to verify and calibrate the model for a reconfigurable platform before we make critical design decisions based on that model. In this work, we present a method involving a general event driven simulation framework to complement the use of abstract models in FPGA-based development. The challenge is that the models are created for radically different purposes and represent different attributes of FPGA designs. It is difficult to develop a single simulation program which can be applied to all models directly. We address this problem by introducing customised libraries in conjunction with a generic simulation framework. With practical examples showing how a model or an actual system can be simulated in this framework, we believe the proposed approach can be useful in FPGA-based reconfigurable application development.

The focus of this work is on the flexibility and generalisation of the framework instead of optimising for specific models or applications. The novel aspects of our approach include:

- A generic event driven simulation framework for modelling reconfigurable platforms. The simulation adopts behavioural modelling, platform specific parameters and practical data sets from the target application. It provides realistic information about energy consumption, communication, and computation performance.
- A systematic approach to create libraries capturing reconfigurability of the target platforms for different types of applications. These libraries are application specific and can be extended independently from the simulator architecture. Also, fast design exploration is facilitated using the different sets of libraries.
- The proposed approach is applied to three case studies to demonstrate its effectiveness. It is used to determine the key networking parameters in an FPGA cluster application, verify the power model for a reconfigurable filter in software-defined radio, and estimate the performance of an option pricing application.

This work was presented in part at the Third International Workshop on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2012), Naha, Okinawa, Japan, May 31-June 1, 2012.

2. RELATED WORK

There are various vendor provided tools for power estimation of FPGA design. Besides the Xilinx XPower Estimator [1] mentioned before, more detail and realistic tools, such as the Altera PowerPlay [2], make use of the recorded activities of the during logic simulation. Although these tools provide more accurate information than the static models, the required activity records can only be obtained when the application logic is developed. It is difficult to apply this approach in design space exploring in early design stages.

Researchers have been using domain specific models to estimate merits of reconfigurable computing platforms. In 2005, a multi-level power modelling framework was proposed for the VPR tool [3]. The framework models both dynamic and static power characteristics of the connections, lookup table clusters and transitional glitches. In 2010, a method was proposed to optimise the degree of parallelism based on effects of execution energy and reconfiguration energy in an FPGA [4]. This work achieved significant energy reduction in filter designs by run-time reconfiguration of coefficient-optimised modules. Performance for specific types of applications can also be modelled [5, 6]. Given the benefits and achievements from these studies, it is difficult to apply the same approach to a wider range of applications due to the lack of a generic and systematic framework.

There are also simulation tools and abstract high-level models. Logic simulators, such as ModelSim and VCS, can provide cycle accurate performance merits at the cost of fully implemented design and long simulation time. SystemC [7, 8] is a very common simulation framework for hardware development. The SystemC framework is good at system level modelling but involves a complex software environment which is too complicate in many cases. There are also well developed simulators for the networking domain [9, 10]. These simulators are often focusing on the application level can have limited extendibility for capturing the reconfigurable nature of the hardware platform.

In 2008, a dynamic module library [11] for system level modelling and simulation is proposed. Based on an extension of the SystemC environment, the library can emulate a dynamic reconfigurable system by creating and eliminating functional objects while the simulation is running continuously. This approach focuses on the interface and behaviour of dynamic reconfiguration at functional level. Information such as power and performance are not included in the proposed extension.

3. SIMULATION FRAMEWORK

In this work, we adopt a generic event driven simulation framework to perform various tasks from model verification to design space exploration. To facilitate the simulation process, we develop our own simulation software. The design philosophy of the simulator is to provide sufficient functionalities for target applications while restricting the complexity of the software.

3.1 Overview

Figure 1 shows a simplified architecture of the proposed simulator. There are three major components in the system: the System Design Manager (SDM), the Sequencer (SEQ), and the Publish/Subscribe Module (PSM). A design is described by a user in XML form, and this description contains application specific elements of the simulation such as the

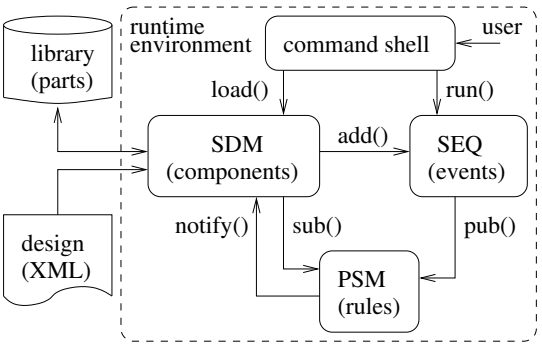

Figure 1: Simulation software architecture.

Table 1: Simulator architecture summary.

	runtime env.	library	design
format	executable	binary object	XML
origin	framework	model designer	user
update	static	per model type	per application
function	simulation	primitive blocks	system layout

behaviour of its components, the platform architecture, and the design inputs. The library contains descriptions of reconfigurable elements which are available at run time. When the simulator starts, it instructs the SDM to load the system design from a user provided design file (in XML format). While loading the design file, the SDM automatically locates and uses the required parts from the libraries. It then simulates the execution of the system as a sequence of events. The SEQ module registers and schedules events in a chronological order. When instructed by user, the SEQ module will fire the registered events up to a specific time interval. For each fired event, the PSM triggers the related actions which may lead to new events being added to the SEQ.

The runtime environment (dotted line region in Figure 1) and the library are developed and distributed in binary executable format. The design file is read and parsed by the simulator dynamically. User can rapidly explore different system parameters or configurations by modifying the XML file without re-compilation. Table 1 summarises the functions and contents of these parts in the simulation process.

This simulator architecture features several advantages. First, the runtime modules of the simulator are designed to be generic so that they can be reused for a variety of application without modification. Second, the isolation between runtime environment, library and design enable rapid system exploration without the knowledge of underlying simulation mechanism. Third, no programming language extension is required since the event-based simulation details is encapsulated in the three main modules.

3.2 Sequencer and Public/Subscriber Model

In our simulation framework, an *event* is defined as a combination of source, type and scheduled time of the event. For example, an event is created for a reconfigurable multiplier finishing the assigned operations after $120ns$ after the data valid event. The SEQ module maintains an internal data structure for all the previously added events. Events in the SEQ module are grouped and sorted chronologically in a list according to their scheduled time.

The SEQ module processes events by *firing* them. An event is fired when its contents are sent to a specific receiver, the PSM in this case, and has itself removed from the scheduled

event list. Users can instruct the SEQ module to run for a given time interval through the simulator command shell. Once the SEQ received the *run()* command, it fires the scheduled events and advances the simulation time after all events scheduled at the current simulation time are fired. The process is repeated until the current time passes the specific time interval. New events can be added during simulation, and the SEQ module is programmed to handle this situation.

Events are sent to the PSM through the *publication()* API. The PSM stores a collection of *rules* based on the descriptions in the design file. A rule is defined as the combination of expected event sources, expected event type and all the targets which will respond to the event. The SDM adds, through the *subscription()* API, new rules to the PSM following the specifications in the system design file. When the PSM receives a fired event, the corresponding rules are located and the targets in rules are notified by the *notification()* API of the SDM.

3.3 System Design Manager

The main functions of the SDM module are: to parse the design file; to create and configure the components; and to declare interaction between the components. The usage of a system design file is explained in the following subsection.

A *component* represents a self-contained behavioural block in the system being modelled. It consists a private collection parameters and the callback functions for manipulating these parameters. In the current implementation, 4 callback functions are included: *cleanup*, *configure*, *reference* and *notify*. The configure function allows users to change the parameters of a component at run time. The reference function returns pointers to the component's internal parameters by symbolic lookup. The notify function is called when the component received notification from the PSM. These functions facilitates the interaction between components and the control from users. The SDM keeps track of the creation and deletion of all the components during simulation. With the help from callback functions and symbolic references, the SDM serves as a proxy between the actual implementation of the component and the simulator runtime modules.

The SDM initialises the rule table in PSM following the specifications in the design file. The components interact by scheduling new events and responding to notifications according to this rule table. After loading the design file, the SDM automatically creates a *startup* component which adds the first event in the simulation at time zero. All other components requiring a reset or initialization process will be notified by this event.

3.4 System design entry

The model for the reconfigurable platform is captured in the simulation library. The library contains *parts* which can be instantiated by the SDM as components. A *part* is defined as a combination of a specific data structure and the implementation of the feedback functions. The data structure is the format of the component's private parameters which are known by the specific part only. To set up simulation, the components have to be created and configured by the user in the XML design file. A simplified example of a design file in a digital circuit application is listed below.

Listing 1: Simplified Design File

```
1  ...
2    <name>circuit</name>
3    <scale>0.001</scale>
4    <library>digital</library>
5  <!-- component declaration -->
6    <component>
7      <name>U0_and</name>
8      <part>and_gate</part>
9    </component>
10
11   <component>
12     <name>A</name>
13     <part>active_source</part>
14   </component>
15 ...
16 <!-- component configuration -->
17   <configuration>
18     <component>A</component>
19     <url>sim_data_a.txt</url>
20   </configuration>
21 ...
22 <!-- component connection -->
23   <connection>
24     <src>A</src>
25     <src_port>dat</src_port>
26     <dst>and</dst>
27     <dst_port>I0</dst_port>
28   </connection>
29 ...
30 <!-- component interaction -->
31   <interaction>
32     <component>and</component>
33     <event> <src>A</src> <msg>1</msg> </event>
34     <event> <src>B</src> <msg>1</msg> </event>
35   </interaction>
36 ...
```

The design file contains five parts. The first part is the preamble which includes the design name, version, simulation time resolution and library information. The SDM can load multiple libraries depending on the context in this part.

The second part (line 5–14) is for component declarations. Each component is declared as a pair of component name and part name. Component name can be used as a symbolic reference in the following parts of the design file. Part name is used to locate the parts in the loaded library so that the SDM can create a corresponding component. The simulator assigns an unique numerical *ID* to each created component for internal reference.

The third part (line 16–20) describes component configurations. The parameter names and values are given for specific components. Users have to consult the library implementation for available parameters and the acceptable input ranges. In the example, a file name is assigned for the *url* parameter of the active input source component 'A'. Component parameters can be configured either statically during design file parsing or dynamically during simulation.

The fourth part (line 22–28) describes data communication between components. For easier data coherency, a component retrieves its input parameter values by referencing an external data structure instead of keeping a local copy. The connection is made by first getting the reference of the source and then assigning it to the destination as a regular parameter value. In the example, the reference of parameter `dat` from component `A` is assigned to parameter `I0` of component `U0_and`. The actual value of `dat` will be used when `I0` is involved in computation within `and_gate`.

The fifth part (line 30–35) describes interaction between components. The interaction rules are defined as a component followed by a list of events it will respond to. The numerical event type is defined in the library and must be unambiguous within a part. The connection and the interaction together enable a component to collect the environment

status during simulation.

4. MAPPING OF ABSTRACT MODELS

4.1 Creating model library

A library is a collection of multiple parts. Each part has its own data structure and defined reactions to the changes of system status such as incoming events. The first step in developing a library is to decompose the model into parts. In the digital circuit example, logic gates and storage elements are mapped to individual parts directly. It is possible to have complicated high level parts including more model information to improve run time of the simulator. For example, when simulating a network model, a complete network node can be mapped to a single part instead of separated parts for queues and routers. On the other hand, mapping a complete model into a single part is effectively equivalent to creating dedicated simulator for that model. Without utilising the functions of the simulator runtime environment, this may restrict future changes of the model and limit design exploration opportunities. So the challenge is to find a balanced granularity when mapping models to library parts.

The second step is to define the parameters of each part. These parameters can be obtained from the abstract model itself. They may also be extracted from the reconfigurable platform to present a empirical environment during simulation. Both external data sources and internal status of the part, as discussed in the previous section, can be included in the parameter set. The value of a parameter can be changed by loading external simulation vectors, by referencing the output values from other components, or by recomputing the referenced input values. The parameter values can also be swept through user commands during simulation for rapid design space exploration.

The third step is to define the event type and the corresponding reactions for each possible event. An acceptable event of a part can be generated either internally or externally. Internal events are usually used to implement a finite state machine. External events are usually used to notify the changes of values of the input parameters. Combining the event type and the source component ID, the reference to the changed input parameter can be correctly identified. An abstract model usually captures system behaviour in mathematical expressions. In the proposed framework, these expressions are implemented as response to the incoming events.

4.2 Simulation in reconfigurable design flow

The library can be developed and used in conjunction with the abstract model. The main purpose for simulation is to verify the model and reveal deviations introduced by abstraction. Once the system parameters and their relationships are identified, model libraries can be constructed as described in the previous subsection. The remaining question is how we can actually make use of the simulation.

The proposed simulation approach is different from manual analytical treatment. Manual analytical treatment often involves a static set of parameters for calculating system merits based on mathematical expressions. In the simulation approach, the inputs include not only static parameter values but also dynamic changes in both system configuration and application data. When applying application data, either directly or after characterisation, the output of the simulation approach will provide a more realistic estimation of system behaviour. This can be used to evaluate the accuracy of the abstract model. When sweeping the system configurations at run time, our simulation approach generates multiple results of the same merit. This can be used to refine the parameter settings of the current model.

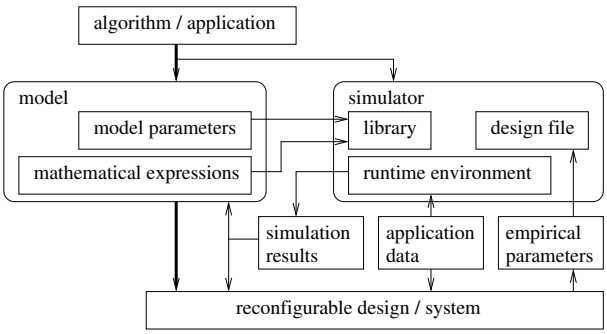

Figure 2: Simulation in design flow.

Figure 2 illustrates the use of this simulation framework in the design flow for reconfigurable platforms. Having the simulation approach in parallel with the modelling stage, better design decisions can be made in the early stage of the development flow. The thin arrows represent the new information flow when applying the simulation approach. It shows that previously unused information such as detailed empirical parameters and application data sets are now utilised in the simulation. The results from simulation can be used to improve the accuracy of the analytical models and the performance of the final reconfigurable designs.

To study different behaviour in the system, models are usually created individually with a specific set of parameters and mathematical expressions. It is difficult to combine these individually constructed models to give a more comprehensive understanding of the system. With the help of the simulation approach, we can study different system merits at the same time by sharing the same events and status in the system.

Our simulator allows inspection of intermediate states of the system during simulation. Although manual analysis can be used in estimating peak system performance, it is difficult to study the origins of timing critical events. The ability to pause at any time during simulation based on user commands or pre-defined triggers can show developers when and what happens at critical points.

5. APPLICATION CASE STUDIES

To demonstrate the proposed simulation approach, we apply the simulator to three existing applications with abstract models. In each case, the method of mapping the model to the simulation framework is presented as well as the results from the simulator. We also explain how the simulation results improve system-level designs.

5.1 Case I: Network communication model

Efficient communication between nodes is critical for achieving high performance in a computer cluster. Based on a dedicated inter-accelerator network, the work in [12] enhances such communication with advanced networking functions such as broadcasting and priority routing. It is important to have an estimation of the expected performance of an inter-FPGA network that incorporates these new features.

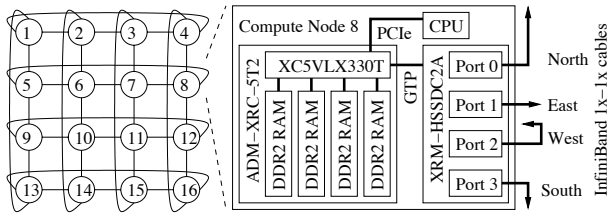

Figure 3: A 2D torus network utilising point-to-point FPGA serial communication.

The model is designed for a 2D torus network as shown in Figure 3. Equation 1 is used to model the one-to-all (ota) communication performance:

$$T_{ota} = \lceil d/p \rceil \times (T'_L + T_a) + (\hat{k} - 1) \times T'_L + T_a \qquad (1)$$

where T_{ota} is the link transmission delay for a packet, T'_L is the link latency, T_a is the routing latency, d is the data size in bits, p is the packet size and $\hat{k}$ is the maximum link length. This is based on the assumption of no congestion point in the network and the arrival rates from all ports are the same. It is difficult to estimate the all-to-all performance using theoretical analysis.

For this system, we create a library for modelling FPGA wired network efficiency. We decompose the model into two parts: the node and the communication topology. The customised link level protocol is implemented as a state machine inside a node. It includes system parameters such as line bandwidth, packet processing time, queue size and packet ID counter. In a theoretical model, the time to process a packet in the FPGA is usually omitted for simplicity. This parameter is supported by our simulation library.

The original model [12] adopts fixed packet size, constant packet generation rate, and maximum link length for a worst case scenario study. In the simulation, a random number generator is used for producing realistic values for simulation parameters. The generation rate, size and link length are all generated randomly for the node during simulation.

The connection section in the design file creates the 2D torus connections (Figure 3). The communication topology part in the library can be configured dynamically to change the size and shape of the torus. Since the routing logic is based on the size and shape of the 2D torus network, the topology part also implements callback function for the node to route incoming packets.

The simulator prints out messages of creation, routing and processing for each packet during simulation. This transcript of packet life cycle provides a detailed view of the simulated network status before FPGA implementation. New information can be obtained from simulation: we observe that the queue size should be increased linearly with the maximum link length of the network to avoid buffer overflow in a node. The information can be used when designing the actual hardware for efficient area utilization.

5.2 Case II: Energy efficiency model

Energy efficiency is another important issue in reconfigurable computing. A method of optimising parallelism for energy efficiency is proposed in [4]. The idea is to leverage the benefits of reconfigurability by scaling the number of parallel cores such that total energy consumption is minimised by trading off reconfiguration improvements and overheads. The total energy, for both computation and configuration, is modelled in Equation 2.

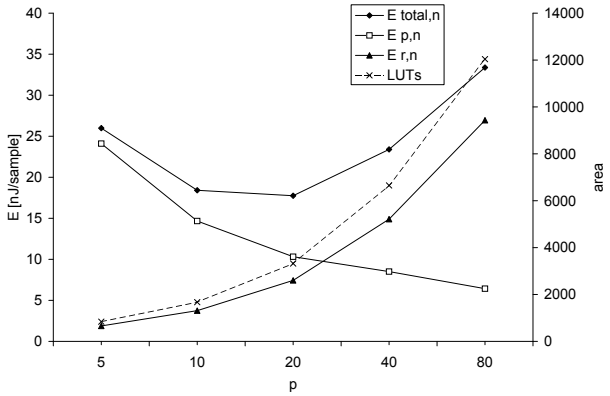

Figure 4: Energy efficiency and area of an FIR filter for different parallelism using Xilinx CoreGen.

$$E_{total} = P_{c,e} \cdot t_{p,e} \cdot s \cdot n + P_o \cdot \frac{t_{p,e} \cdot s \cdot n}{p} + P_r \cdot t_{r,e} \cdot p \qquad (2)$$

where $P_{c,e}$ is the computation power consumption per processing element, $t_{p,e}$ is the processing time of a processing element (PE), s is the iteration steps in the algorithm, n is the data item count to be processed between reconfigurations, p is the number of parallel processing elements, P_o is the constant power overhead, P_r is the reconfiguration power and $t_{r,e}$ the reconfiguration time for one PE.

This model captures system energy consumption as a function of parallelism, logic area and computation time. The larger the parallelism, the more logic cores are employed and the shorter the computation time is. The energy used for configuration is increased but the energy overhead in computation is reduced. The model is used to find the optimal degree of parallelism for the total energy cost as shown in Figure 4 for an 80-tap FIR filter which can process 10,000 16-bit samples between parameter updates.

In this work, the model is captured in the library as two parts: the PE part and the device part. The PE part can accept parameters such as processing time and energy for computation and configuration. This enables us to simulate systems where heterogeneous PEs are available since the processing elements can have different energy foot prints. The device part maintains the total area resources of the FPGA target. Each PE reconfiguration involves checking resource availability so that reconfiguration takes place only if resource constraints are met. The energy overhead is also captured in this device part.

The simulation starts by creating a sufficiently large number of PE components and a single device component. The user input file is a list of configurations including data size and parallelism. PE components are activated according to this configuration by first entering the configuration stage and then entering the computing stage. Each PE component records its own configuration and computing power according to the parameters and the simulation time. The device component keeps track of all the activities of the PE components and records the overhead energy. Once all PE components exit their computing state, the energy of all components are reported. By doing so, the user can rapidly explore different design parameters and parallelism trade-offs by revising the configuration data file. Our simulation results match experimental results from [4].

5.3 Case III: Performance model

This case study is similar to the previous one but optimising for performance under area constraints. Some FPGA devices allow the user to reconfigure part of its contents at run time while the rest of the FPGA device continues functioning. A run-time reconfiguration scheme is proposed in [13] to utilise specialised operators for better area efficiency. This approach is applied to a financial option pricing application where multiplication and addition are the essential operators in the computation.

The scheme is based on the fact that one of the inputs to the multipliers and the adders are coefficients for specific options. These coefficients are relatively static during the pricing computation. Since a constant coefficient operator has significant area reduction when compared with a general full-range version, it is possible to have a performance gain by reconfiguring the FPGA device with a large number of specialised operators at run time. However, this approach suffers from two main drawbacks. First, the time for reconfiguration is proportional to the area being reconfigured. This can be a significant overhead if the coefficients are changed continuously and frequently. Second, the number of available specialised operators are limited by the development time and storage requirements. To support coefficients without a corresponding specialised operator, a general full-range operator must be used.

The original analytical model [13] involves application and device constraints and system overhead. In contrast, our approach models the system as a scheduler. The model library has two parts: the processing element (PE) part and the device part. The PE part is similar to the previous case study except that it does not record any system merit. The device part is different since it is used as a scheduler in this case. The device part is associated with an external resource file which lists all the supported coefficients. The user input is a stream of coefficients from real-world applications.

When simulation starts, a large number of PE components and a single device component are created. The device component reads in the user supplied coefficient stream and looks up into the supported list. If the coefficient set is supported, the device component configures and notifies one idle PE component and reduces the internally recorded available area resources. If the coefficient is not supported from the pre-implemented specialised PE, a generic PE using all full-range input operators is configured to be the idle PE component. After all inputs in the coefficient stream are processed, the final simulation time is reported as the result of the simulation.

The original analytical model can only estimate system performance under a specific coefficient set. By applying our simulation approach, we can estimate system performance for a continuous input stream which bears close resemblance to a real world situation. Also, since the computation time and the reconfiguration time are different for specialised and generic PE components, the system will have PE components in different states after a period of simulation time. This is inspired by real hardware implementations where the configuration and computation of the PEs are not synchronised. In this case study, the simulator provides results based on a dynamic environment and information which is previously unavailable from the static model.

Our simulation approach enables us to study a new run-time reconfiguration scheme for improved performance. By observing the simulation process, we notice that there is a situation that the same specialised component is loaded multiple times. This inspires us to consider a scheduler which processes a window of coefficient sets instead of a single set each time. By re-ordering and grouping the inputs, we can expect further improved performance of the application.

6. CONCLUSION

This paper presents a novel event driven simulation framework targeting reconfigurable computing platforms. The steps for mapping reconfigurable models to simulation libraries are also discussed. After applying this simulation approach to three applications, we demonstrate that it can reveal potential system bottlenecks, help finding optimised values for system parameters, evaluate the accuracy of abstract models, and provide previously unavailable information. Future work includes refining our techniques and tools, and extending them for new applications.

Acknowledgment

The support from Imperial College London Research Excellence Award, UK Engineering and Physical Sciences Research Council, Alpha Data and Xilinx is gratefully acknowledged. The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreements number 248976, 257906 and 287804.

7. REFERENCES

[1] *XPower Estimator User Guide*, Xilinx, 2011.

[2] *Quartus II Handbook Version 11.1 Volume 3: Verification*, Altera, 2011.

[3] F. Li *et al.*, "Power modeling and characteristics of field programmable gate arrays," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 11, pp. 1712–1724, Nov. 2005.

[4] T. Becker, W. Luk, and P. Cheung, "Energy-aware optimisation for run-time reconfiguration," in *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2010, pp. 55–62.

[5] C. Lin, H. So, and P. Leong, "A model for matrix multiplication performance on FPGAs," in *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, 2011, pp. 305–310.

[6] M. Mücke, B. Lesser, and W. Gansterer, "Peak performance model for a custom precision floating-point dot product on FPGAs," in *In Proc. Parallel Processing Workshops (Euro-Par)*, 2011, pp. 399–406.

[7] T. Grtker *et al.*, *System Design with SystemC*, 2002.

[8] A. Schallenberg *et al.*, "OSSS+R: modelling and simulating self-reconfigurable systems," in *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, 2006, pp. 1–6.

[9] K. Fall and K. Varadhan, *The ns Manual*, 2011.

[10] *OMNeT++ User Manual version 4.2*, Andras Varga and OpenSim Ltd., 2011.

[11] K. Asano *et al.*, "Dynamic module library for system level modeling and simulation of dynamically reconfigurable systems," *Journal of Computers*, vol. 3, no. 2, pp. 55–62, Feb. 2008.

[12] S. Denholm, K. Tsoi, P. Pietzuch, and W. Luk, "Efficient communication for FPGA clusters," in *Proc. International Symposium on Applied Reconfigurable Computing*, 2012.

[13] T. Becker, Q. Jin, W. Luk, and S. Weston, "Dynamic constant reconfiguration for explicit finite difference option pricing," in *Proc. International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, 2011, pp. 176–181.