Journal of Systems Architecture 56 (2010) 88–102

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

UML-based hardware/software co-design platform for dynamically partially reconfigurable network security systems

Chun-Hsian Huang, Pao-Ann Hsiung *, Jih-Sheng Shen

Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi-621, Taiwan, ROC

ARTICLE INFO

Article history:
Received 21 March 2009
Received in revised form 28 September 2009
Accepted 23 November 2009
Available online 3 December 2009

Keywords:
UML
Dynamically partially reconfigurable system
Hardware/software co-design
Network security embedded system

ABSTRACT

The dynamic partial reconfiguration technology of FPGA has made it possible to adapt system functionalities at run-time to changing environment conditions. However, this new dimension of dynamic hardware reconfigurability has rendered existing CAD tools and platforms incapable of efficiently exploring the design space. As a solution, we proposed a novel *UML-based hardware/software co-design platform* (UCoP) targeting at *dynamically partially reconfigurable network security systems* (DPRNSS). Computation-intensive network security functions, implemented as reconfigurable hardware functions, can be configured on-demand into a DPRNSS at run-time. Thus, UCoP not only supports dynamic adaptation to different environment conditions, but also increases hardware resource utilization. UCoP supports design space exploration for reconfigurable systems in three folds. Firstly, it provides reusable models of typical reconfigurable systems that can be customized according to user applications. Secondly, UCoP provides a partially reconfigurable hardware task template, using which users can focus on their hardware designs without going through the full partial reconfiguration flow. Thirdly, UCoP provides direct interactions between UML system models and real reconfigurable hardware modules, thus allowing accurate time measurements. Compared to the existing lower-bound and synthesis-based estimation methods, the accurate time measurements using UCoP at a high abstraction level can more efficiently reduce the system development efforts.

1. Introduction

Due to rapid technology breakthroughs, FPGAs devices, such as Xilinx Virtex II/II Pro, Virtex 4, and Virtex 5, can be partially reconfigured at run-time, which means that one part of the device can be reconfigured while other parts remain operational without being affected by reconfiguration. A hardware/software embedded system realized with such an FPGA device is called a *dynamically partially reconfigurable system* (DPRS), which enables more applications to be accelerated in hardware, and thus reduces the overall system execution time [21,22].

Our target applications focus on network security embedded systems, where the cryptographic and hash functions are usually computation-intensive, hard real-time, and non-adaptive to changing network conditions. They are usually designed as fixed ASICs, which makes the network security system unable to adapt dynamically to network security run-time requirements. An alternative is to use reconfigurable logics such as the FPGA. There are two reasons for implementing such functions using FPGA.

1. The regular arrays of logic cells on FPGAs are attractive for cryptographic and hash hardware designs because of its parameter-specific architecture. Many related researches [10–12,19] also showed that implementing computation-intensive cryptographic and hash hardware functions on the FPGAs can make a network security system more efficient by taking advantage of the specific architecture.
2. FPGA allows hardware functions to be dynamically reconfigured. Thus, a DPRS architecture is desired for a network security system that can dynamically adapt to changes.

Our target system is called a *dynamically partially reconfigurable network security systems* (DPRNSS), in which the computation-intensive cryptographic and hash designs, such as CRC, *Message-Digest algorithm 5* (MD5), RSA, *Data Encryption Standard* (DES), 3DES, and *Advanced Encryption Standard* (AES), are implemented as partially reconfigurable hardware functions for enhancing the system performance and flexibility. Here, the configured hardware functions running on the FPGA can be also called as hardware tasks. Compared to a network security embedded system, a DPRNSS contains not only software applications and hardware devices, but also reconfigurable hardware functions. Thus, the development of such a DPRS is much more complicated.

* Corresponding author. Tel.: +886 5 272 0411x33119; fax: +886 5 272 0859.
E-mail addresses: huang@cs.ccu.edu.tw (C.-H. Huang), hpa@computer.org (P.-A. Hsiung), sjs92@cs.ccu.edu.tw (J.-S. Shen).

1383-7621/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.sysarc.2009.11.007

To facilitate the DPRS design at a high abstraction level, the *Unified Modeling Language* (UML) [1], an industry de-facto standard, is used to model and develop DPRS [5,14,17,18]. Through system modeling, the functional interactions between the system and the applications can be easily described and analyzed. However, UML models of a DPRS are usually not customized, and thus designers must model their DPRS designs again when different applications are included. Furthermore, reconfigurable hardware functions are usually individually implemented at design-time without integrating with a unified interface. As a result, to incorporate hardware functions having different data interfaces with a DPRS at run-time becomes very difficult, which not only reduces system scalability but also increases development efforts.

For system requirement estimation, conventional UML models cannot accurately evaluate system characteristics, for example, usually the worst case execution time is used for estimating performance, instead of the actual execution time. The problem is that not only the simulation of such UML models of a DPRS is time consuming, but it might be even misleading, that is, an incorrect design could be verified by a simulator as being correct. Such inaccurate system validation further leads to much more iterations between the system modeling and implementation, thus delaying system development. To solve the above problems, a novel *UML-based hardware/software co-design platform* (UCoP)[1] is thus proposed, which contributes to the state-of-the-art in the following ways.

- UML models of UCoP are specified as reusable models, using which different user applications can be easily employed in UCoP to analyze the functional interactions among all system components. As a result, design time and efforts are significantly reduced.
- To increase the system scalability, a partially reconfigurable hardware task template is provided to integrate hardware functions having different data interfaces in UCoP. Designers can focus on their hardware designs without going through the full partial reconfiguration flow, which also reduces the system development efforts.
- The UML models of UCoP can be used to *directly* and accurately validate the system correctness and performance at a high-level phase instead of inaccurate time estimation using simulation. Through the *direct* interaction with the real hardware architecture, the number of iterations between the UML modeling and the system implementation is reduced more significantly until the final system is created.

This article is organized as follows. Section 2 discusses related research work and compares them with UCoP. The target embedded system is described in Section 3. The details of the partially reconfigurable hardware architecture and the UML modeling are illustrated in Section 4. Section 5 shows the implementation results of all dynamically partially reconfigurable hardware functions, and provides a comparison between the proposed platform and the related research. Finally, conclusions and future work are described in Section 6.

2. Related work

The reasons for adopting reconfiguration techniques in network security applications are described in this section. Further, existing contemporary UML-based methodologies for reconfigurable system design are also introduced and compared with the proposed UCoP.

Owing to the popular use of the network, the needs for data security and authentication are getting more and more important, in which cryptography plays a key role. However, cryptographic algorithms are usually computation-intensive, have hard real-time requirements, and are non-adaptive to changing network conditions. The algorithms also make different tradeoffs between security and complexity. To allow multiple tradeoffs and to adapt to changing network conditions at run-time, a data protective process needs a high-speed and flexible embedded system. Wollinger and Paar [19] illustrated the advantages of reconfigurable devices for cryptographic applications in embedded systems, including architecture efficiency, resource efficiency, throughput, and algorithm agility.

Besides much more efficient implementation of cryptographic applications on the reconfigurable devices, higher hardware resource utilization and system flexibility can be achieved using partial reconfiguration. Lagger et al. [10] proposed a self-reconfigurable pervasive platform for cryptographic applications. The authors compared a full-software design with a coprocessor design that had an FPGA device which could be partially configured with DES, 3DES, and *Route Coloniale 4* (RC4) hardware cores. Compared to the former, the performance of the latter was significantly enhanced due to dynamic reconfiguration techniques. In other related researches such as [11,12], the authors enhanced the performance of cryptographic hardware designs significantly by leveraging the advantages of reconfigurable FPGAs. These researches all demonstrated that reconfigurable FPGAs are very suitable for implementing cryptographic applications.

Besides low-level system implementation, high-level modeling is also used for system development. *Unified Modeling Language* (UML) [1], a de-facto standard language, is usually used to model system functional interactions not only in the field of software engineering, but also in embedded system development. In [5,17], the authors proposed a complete UML-based design methodology for reconfigurable architectures, which started with UML models and ended with final implementation and deployment. It included a model compiler to create executable applications from system-level specifications for reconfigurable architectures. In [14], a model-based approach for executable UML was proposed to close the gap between the system specification and its model-based execution on reconfigurable hardware. The UML specifications can be compiled to binary representations that were directly executed on their proposed abstract machine platform, which was implemented on a Virtex II FPGA. However, all these researches [5,14,17] focus only on the functional code generation and not design space exploration. As a result, system development usually needs several iterations between the UML modeling and the implementation phase until the final system is created.

Another UML-based design flow for dynamically reconfigurable systems was proposed in [18]. The design flow not only generated the system implementation from system-level specifications, but also proposed a hardware/software partitioning method that supported a software-oriented strategy and automatic hardware/software co-synthesis. The authors also established a reconfigurable system architecture to support their design flow. However, only fully reconfigurable architecture was supported, while the *partially* reconfigurable architecture was not considered.

The above UML-based design flows [5,14,17,18] only simulated the functional interactions between applications and a system, thus the physical design correctness of the system could be verified only after the UML models were synthesized into concrete system designs. Graf et al. [8] proposed a model-level debugger that could directly interact with the reconfigurable architecture. The debugger integrated the Matlab Stateflow models with its target system,

[1] UCoP was first proposed in [6] by us. In this work, the use of UCoP and more related experiments will be illustrated in details.

and consisted of a code generator for transforming the Stateflow models into HDL code. Through the JTAG cable, users can debug their configured system at the graphical model-level. However, the debugger did not provide support for real-time tracing of the functional interactions between applications and a system, thus their model execution must be suspended while reading the system state for debugging.

In this work, UML modeling is included in UCoP to analyze the system-level functional interactions between user applications and a DPRS. UML models of UCoP are classified into three categories, including *software application* models, *system management* models, and *hardware configuration* models. A DPRS only needs to be specified by customizing the three categories of reusable UML models, and high-level functional analysis can thus be performed in UCoP. UCoP further provides users with a partially reconfigurable hardware task template to integrate their hardware functions into a user-defined DPRS. As a result, users can focus on their hardware designs without going through the full partial reconfiguration flow. Through both the reusable UML models and the partially reconfigurable hardware task template in UCoP, the system scalability is significantly enhanced, while design time and efforts for a user can also be reduced.

In contrast to existing high-level time estimation methods [5,14,17,18] used in contemporary DPRS development, the UML models of UCoP *interacts directly* with the real hardware in a system architecture, thus enabling accurate time measurements. Users can accurately validate and analyze the correctness and performance of their new hardware designs in UCoP at a high design level before a system is actually implemented. Moreover, in contrast to the Matlab approach [8], UCoP supports real-time tracing without suspending model execution. Therefore, developing a DPRS in UCoP is more efficient and accurate because not only the time-consuming hardware/software co-simulation, such as using the SystemC language, can be avoided, but also real-time tracing can be provided for users to verify their DPRS. For our target DPRNSS, we leveraged the features and capabilities of reconfigurable devices to implement cryptographic applications in DPRNSS. Furthermore, the partial reconfiguration technique is adopted to achieve higher hardware resource utilization and system flexibility, while cryptographic and hash hardware functions can be dynamically reconfigured into DPRNSS on demand at runtime.

3. Dynamically partially reconfigurable network security system

Before introducing the UCoP design platform, we first describe our target DPRNSS architecture. In this section, the system requirements, the architecture design, the data flow, and the system features for DPRNSS will be illustrated in details.

3.1. System requirements

The proposed DPRNSS must be able to support for widely-used cryptographic functions, including RSA, DES, 3DES, and AES, for data encryption/decryption and four hash functions, including MD5, CRC32, CRC64, and CRC128, for data authentication. All eight functions must be dynamically reconfigurable , that is, configurable into DPRNSS at run-time to adapt to changing network environment conditions. The reconfigurable functions must be implemented as hardware in a Xilinx Virtex II FPGA with at most 15,000 slices, while their total amount of required logic resources must be less than 10,000 slices. Furthermore, the static power consumption of DPRNSS must be less than 600 mW. Furthermore, the network security status must be classified into different threat levels, and each cryptographic hardware function must be associated with a level of vulnerability that presents the maximum network security threat level that the function can endure for secure data transfers. DPRNSS must be able to monitor the current network security threat level and must be able to dynamically reconfigure a cryptographic hardware function into the system to meet the network security needs, that is, newly reconfigured hardware function has a level of vulnerability higher than the currently detected network security threat level.

3.2. Architecture design

To realize the system requirements, a DPRNSS design architecture as illustrated in Fig. 1 is created, which consists of five system devices, including a microprocessor, an FPGA, a hardware/software communication interface, a network interface, and an off-chip memory. The DPRNSS is implemented as a DPRS, which consists of several *Partially Reconfigurable Regions* (PRRs) in an FPGA, while the reconfigurable hardware functions, namely *Partially Reconfigurable Modules* (PRMs), are configured into PRRs. For the partial reconfiguration of cryptographic and hash hardware designs, two PRRs, namely `PRR1` and `PRR2`, are implemented on the FPGA. PRMs of cryptographic and hash hardware designs can be dynamically reconfigured into the PRRs. All partial bitstreams for cryptographic and hash hardware designs are stored in an off-chip memory. Four categories of software components, including attack monitor, configuration controller, system controller, and software application functions, run on the microprocessor. The attack monitor connects to the network interface device to monitor the network status for detecting the current network security threat level, and then for notifying the system controller. The configuration controller is responsible for dynamically reconfiguring the partial bitstreams into the FPGA, while the system controller manages all control and data transfers in the DPRNSS. The software application functions include real-time applications, such as multimedia and online interactive communication.

3.3. Data flow

In DPRNSS, the system controller receives encrypted data (cypher text) through the network interface, which are then sent through hardware/software communication interface to a hash hardware design on the FPGA for authenticating the data integrity. After authentication, the system controller sends the authenticated data to a software application for further processing. If authentication fails, no software application is invoked. Based on the negotiated encryption algorithm, the software application makes a request through the system controller for a corresponding decryption hardware. For transferring data to the network the above steps are reversed and the software application requests for encryption hardware functions instead. It is assumed that during any time period a cryptographic hardware design can guarantee the security of data transfers only for a fixed number of network attacks. When the current cryptographic hardware function cannot ensure the security of data transfers due to increased network attacks, the system controller starts negotiating with the other end-point system to determine whether it can support another more secure cryptographic algorithm. If the negotiation succeeds, the system controller requests the configuration controller to reconfigure the newly negotiated cryptographic hardware design into a PRR on the FPGA. After reconfiguration is done, the system controller transfers a complete message to the other end-point system to indicate that the newly negotiated cryptographic algorithm will be used for future data transfers.

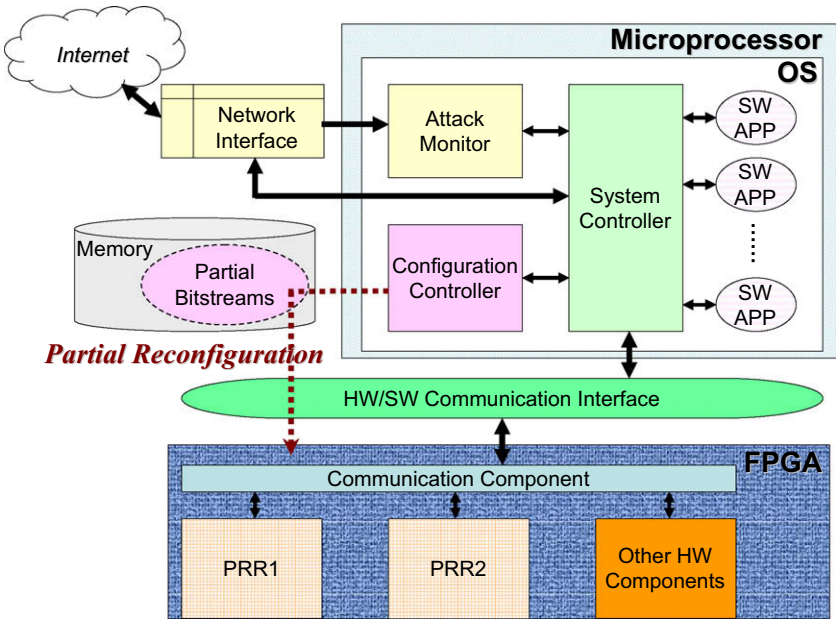

Fig. 1. Dynamically partially reconfigurable network security system.

3.4. System features

Compared to a traditional *Network Security Systems* (NSSs), DPRNSS can enable a greater number of cryptographic and hash hardware functions to be executed even though the total amount of required logic resources exceeds that available in the FPGA. Furthermore, DPRNSS can provide lower static power consumption and higher resource utilization, because cryptographic and hash hardware functions are configured on-demand into a DPRNSS, instead of integrating all hardware functions into a NSS at design time. The related experimental results will be given in Section 5.2. Furthermore, the capability for dynamically adapting hardware function, without switching off the system, is also very important for real-time online system environments, because it provides higher system flexibility compared to NSS. The attack monitor in DPRNSS can detect the current network security threat level, using which new cryptographic hardware functions can be dynamically reconfigured into the FPGA to support secure and efficient data transfers in the network.

4. UML-based HW/SW co-design platform

Though the resource utilization is higher and power consumption is lower in DPRNSS, the development of such a system is more complex than that of a traditional embedded system. Graphical functional modeling is often used to analyze the functional interactions among system components at a high abstraction level. For example, the existing UML-based simulation methods [5,14,17,18] verify system correctness and performance at the system level, where behaviors and the timing characteristics of a real system design are abstracted. As a result, the system correctness and performance cannot be guaranteed even after a DPRS has been verified using simulation, which usually leads to additional design iterations between the system modeling and implementation.

To bridge the widening gap between system models and design implementation, a novel UML-based hardware/software co-design platform (UCoP) is proposed for dynamically partially reconfigurable systems. UCoP achieves its goals in three ways. Firstly, a partially reconfigurable hardware architecture is included in UCoP. Secondly, a task template is provided to integrate different partially reconfigurable hardware functions. Thirdly, UCoP supports the high-level model verification under real physical constraints, as described in the following.

A novel feature in UCoP is the support for UML models to directly interact with real hardware designs that are configured at run-time into an FPGA. During the simulation of UML models, the software models can directly communicate with the hardware designs in FPGA, thus the gap between models and actual implementations is effectively bridged in UCoP. To realize UCoP, we integrated an FPGA platform-specific library into a UML modeling tool. The platform library consists of APIs for data access by hardware designs and for the FPGA configuration control. Users can invoke these platform APIs directly in their UML models, and thus the models can configure new hardware functions into the system and interact with them by sending/receiving data. As a result, hardware/software integration is easily achieved using UCoP. Further, a user can also mix models of different levels of abstraction during the design phase of a system. Thus, UCoP allows efficient simulation through hardware, accurate time measurements of hardware execution and configuration, and real-time debugging. The enhanced accuracy and efficiency in UCoP effectively shortens the overall design time by reducing the number of iterations between model refinement and design implementation. In the following subsections, the implementation details of UCoP will be described, including the partially reconfigurable hardware architecture, the proposed partially reconfigurable hardware task template, and the high-level UML modeling. Furthermore, DPRNSS will be taken as an example for introducing the use of UCoP.

4.1. Partially reconfigurable hardware architecture

To realize UCoP, we chose a reference board, as illustrated in Fig. 2, in which hardware functions run on the *Peripheral Component Interconnect* (PCI) board, an XtremeDSP Development Kit-II [13] from Nallatech, and the software functions run on the general-purpose processor in a personal computer. The PCI board contains three FPGA chips, including a `Clock FPGA` (Xilinx Virtex-II XC2V80) for clock configuration, an `Interface FPGA` (Xilinx Spartan-II) with pre-configured firmware for communicating over the PCI bus, and a `User FPGA` (Xilinx Virtex-II XC2V3000) for configuring the user

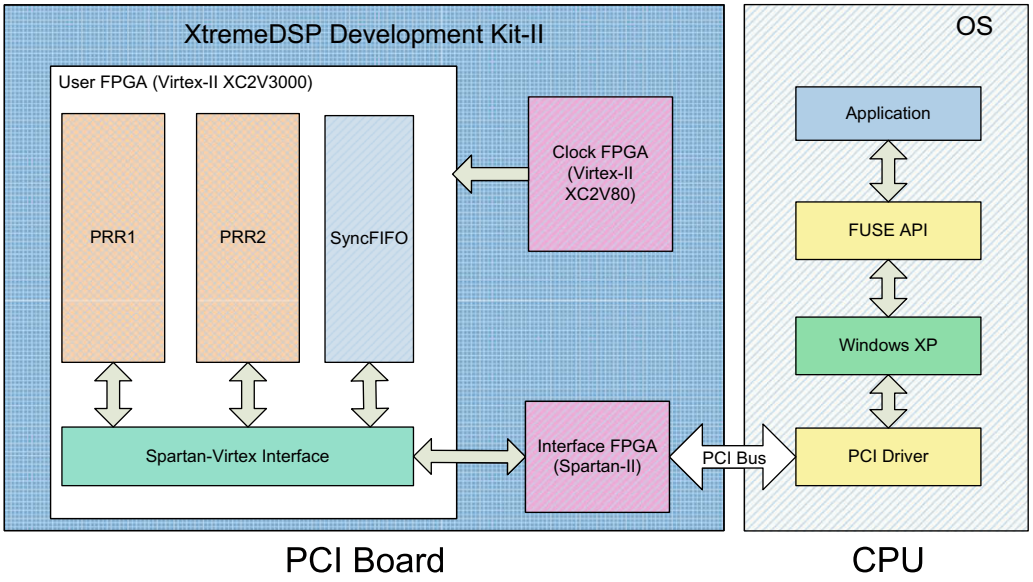

Fig. 2. Partially reconfigurable system architecture.

hardware designs. Besides the hardware part, the *Field Upgradable Systems Environment* (FUSE) APIs and the PCI driver are provided by the XtremeDSP Development Kit-II to facilitate FPGA reconfiguration and communication over the PCI bus.

In this work, the partially reconfigurable hardware architecture was implemented on the `User FPGA`. The dynamic area contains two PRRs in which the cryptographic and hash hardware designs can be (re)configured. The remaining area of the `User FPGA` is the static area, which mainly comprises a Spartan–Virtex interface to communicate with the `Interface FPGA` for transferring and receiving data, and a synchronous FIFO to access the data stored in the on-chip memory. By configuring the static full bitstream for the static area on the `User FPGA`, the dynamically partially reconfigurable hardware architecture can be initialized for communicating over the PCI bus, then the partial bitstreams for the two PRRs are configured as required by the DPRNSS described in Section 3. As shown in Fig. 1, for fitting different network security needs at run-time, the `Configuration Controller` can reconfigure the corresponding cryptographic or hash hardware designs, thus adapting the system functions to the changing network environments and threat level.

All data transfers in our current DPRNSS design are managed by the `System Controller`, which is executed as a software application. As a result, the processing results of a hash hardware design need to be transferred to the `System Controller`, and then back to the cryptographic hardware design. In the future, a *Reconfigurable Module Sequencer* (RMS) [15] design will be integrated into UCoP to support direct communication between cryptographic and hash hardware designs. The RMS can help manage data transfers between the hardware designs in two PRRs on the FPGA, without transferring the data to and from the `System Controller` in the OS. Thus, the communication overheads between the hardware designs in the two PRRs can be further reduced.

4.2. Partially reconfigurable hardware task template

Compared to the development of a traditional embedded system, that of a DPRS is more difficult due to its complicated partial reconfiguration design flow. However, the EA PR design flow is continuously repeated when new user-designed hardware functions are integrated into a DPRS. As a result, the interactive interface between user-designed hardware functions and the rest of the system must be unified. The rest of the system and user-designed hardware functions can be thus individually implemented, which reduces design time and efforts. Therefore, a unified interface is required for the development of a DPRS.

To ease the integration of user-designed hardware functions into the UCoP, a partially reconfigurable hardware task template is proposed, which connects the user functions with the system bus via the Spartan–Virtex interface as shown in Fig. 2. To use a newly developed cryptographic or hash hardware IP in DPRNSS, a designer has to simply integrate the new IP with the proposed hardware task template because the template provides a common communication interface between the IP and the rest of the system.

The partially reconfigurable hardware task template is as shown in Fig. 3. The word size required by all the cryptographic and hash hardware functions is 32 bits for all kinds of data transfers. Thus, the proposed template implements only 32-bit wide signals. It consists of eight 32-bit input data signals, one 32-bit input control signal, four 32-bit output data signals, and one 32-bit output control signal. To connect the reconfigurable part (the PRRs) with the static part (the Spartan–Virtex interface) in `User FPGA`, Xilinx bus macros [22] are inserted to allow correct communication and

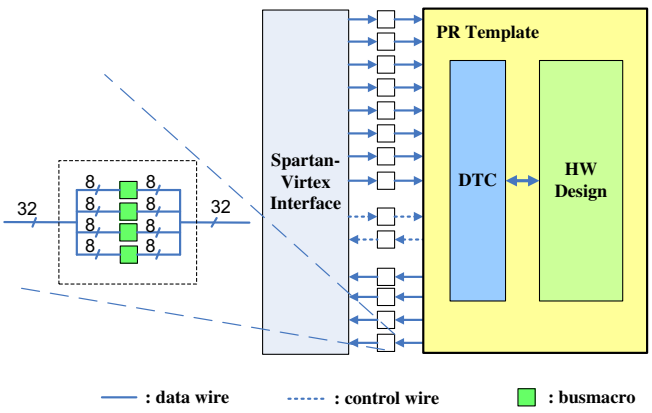

Fig. 3. Partially reconfigurable hardware task template.

connection. The template also contains an optional *Data Transformation Component* (DTC) for unpacking incoming data and packing outgoing data based on the I/O registers sizes in the cryptographic or hash hardware IPs.

For implementing the dynamically partially reconfigurable hardware architecture of DPRNSS, the *Early Access Partial Reconfiguration* (EA PR) flow [22] from Xilinx is adopted. Xilinx bus macros are inserted between each PRR and the static area. After generating the netlists for the static area and for all PRMs, a part of the EA PR flow is followed to generate a full bitstream for the static area and a partial bitstream for each PRM. Blank bitstreams are also generated for all PRRs, which can be used to reset the PRRs. As shown in Fig. 4, the PR implementation flow consists of four phases, namely budgeting, static logic implementation, PR block implementation, and assemble.

A DPRS hardware architecture consists of a static area and a reconfigurable area. In the EA PR flow, the design of the static area must follow the first two phases, namely budgeting and static logic implementation. The static area design can be reused across different applications, and is thus integrated into UCoP such that users can reuse it as required in different applications. As for as the design of new hardware functions is concerned, we need to perform only the last two phases of the PR implementation flow, namely PR block implementation and assemble phases. Corresponding partial bitstreams can thus be generated for each hardware function, without going through all the four phases. Furthermore, the necessary commands for generating partial bitstreams are integrated by UCoP into a script file. Thus, users only need to integrate their new cryptographic or hash hardware design with the proposed hardware task template, synthesize it, and run the script, without explicitly and manually going through the last two phases of the PR implementation flow step-by-step. Using UCoP, users inexperienced in the partial reconfiguration technique can still easily enhance their IP designs with the capability for partial reconfiguration and integrate them into a DPRS. UCoP thus significantly reduces design efforts and enhances system scalability.

4.3. High-level UML modeling

To design a user-specific DPRNSS, a designer must first model the required functions using UML. The basic DPRNSS architecture design and implementation are modeled using UML as described in Sections 3 and 4.1. The models can be classified into three categories, namely software application, hardware configuration, and system management. The software application models describe the software processes running on the microprocessor, while the hardware configuration models describe the bitstreams used for configuring hardware functions. The (re)configuration of all the bitstreams and control and data transfers in the DPRNSS are managed by the system management models. Thus, the system management models act as a bridge between the software application and hardware configuration models.

4.3.1. DPRNSS architecture modeling

The class diagram for a basic DPRNSS is illustrated in Fig. 5, where the `AttackMonitor` and `Application` classes are the software application models, the `PartialBitstream` and `NonprBitstream` classes are the hardware configuration models, and the `SystemController` and `Configure` classes are the system management models. The `AttackMonitor` and `Application` classes are responsible for monitoring the network status and for system applications, respectively. The `SystemContoller`

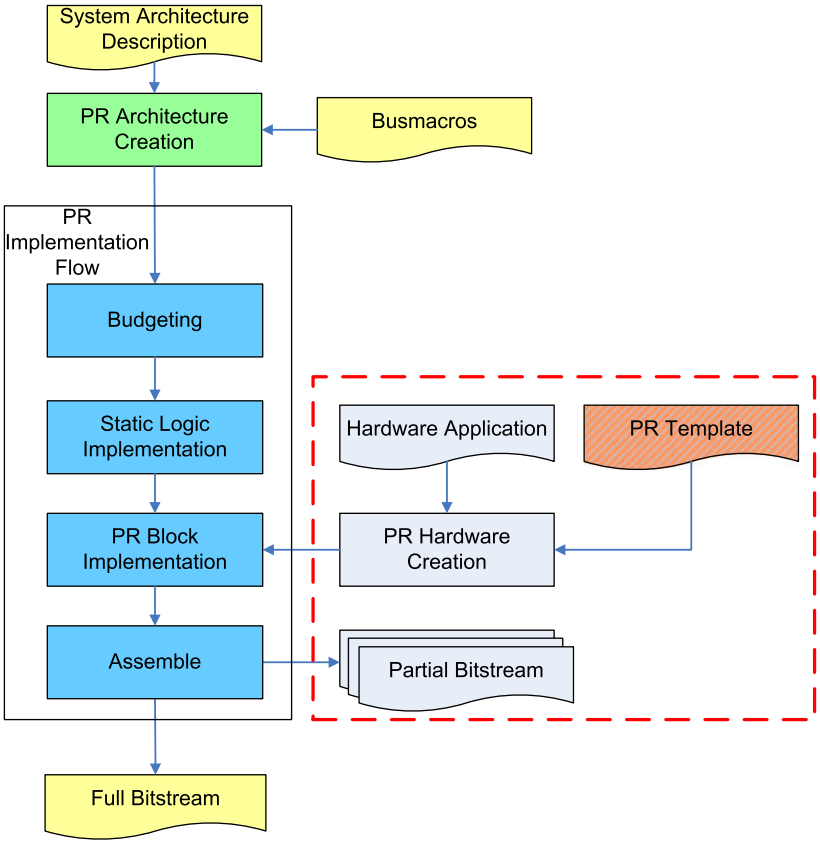

Fig. 4. PR hardware creation flow.

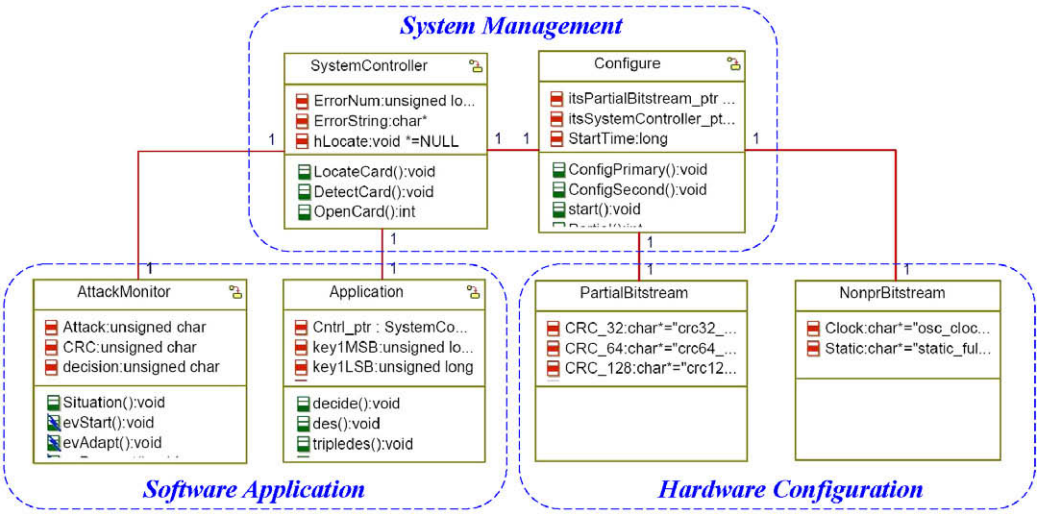

Fig. 5. Class diagram for DPRNSS.

class manages all system operations, and is thus individually associated with the `AttackMonitor`, `Application`, and `Configure` classes. The `PartialBitstream` class models the partial bitstream for each PRM, while the `NonprBitstream` class models the bitstream for the static part of a DPRNSS architecture design. The `Configure` class models the configuration of bitstreams, and is thus associated with the `PartialBitstream` and the `NonprBitstream` classes.

4.3.2. DPRNSS deployment modeling

The UML deployment diagram in Fig. 6 illustrates how the DPRNSS classes in Fig. 5 are mapped to the physical devices of the implementation platform in Fig. 2. The `Processor` node depends on the `SWApplication`, `AttackMonitor`, `Controller`, and `Configuration` components. The `User_FPGA` node depends on the `NonPRbitfile` and `PRbitfile` components, while the `Clock_FPGA` node depends on the `NonPRbitfile` component. The `SWApplication`, `AttackMonitor`, `Controller`, `Configuration`, `NonPRbitfile`, and `PRbitfile` components are implemented as the `Application`, `AttackMonitor`, `SystemController`, `Configure`, `NonprBitstream`, and `PartialBitstream` classes, respectively.

4.3.3. DPRNSS behavior modeling

Besides modeling the functional relationships and the physical device mapping for the DPRNSS using the class and deployment diagrams, respectively, UCoP uses the UML state machine diagrams to model the detailed operations for each system components. Fig. 7 illustrates the state machine diagram for the `SystemController` class, which is responsible for starting and initializing the DPRNSS. After the clock bitstream is configured into the `Clock FPGA`, the full bitstream and then the two partial bitstreams are configured into the `User FPGA`. When all configuration processes finish, the DPRNSS shows which cryptographic and hash hardware functions are configured in the two PRRs. Then, the `Monitor` state is used to receive the requests for dynamically partially reconfiguring the two PRRs according to the network security requirements from the `AttackMonitor` class. Figs. 11 and 12 in the Appendix illustrate the detailed operations for the other classes, except for the `NonprBitstream` and the `PartialBitstream` classes because they model passive components containing pointers to the bitstreams that are saved in memory.

4.3.4. DPRNSS validation framework

In this work, we adopt the Rhapsody tool for UML modeling and simulation [4] because it can automatically generate C, C++, Java,

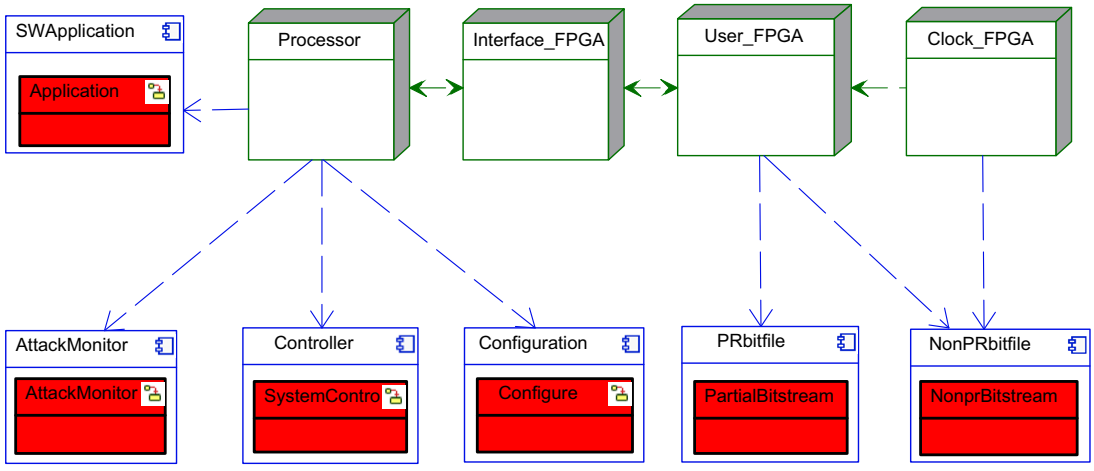

Fig. 6. Deployment diagram for DPRNSS.

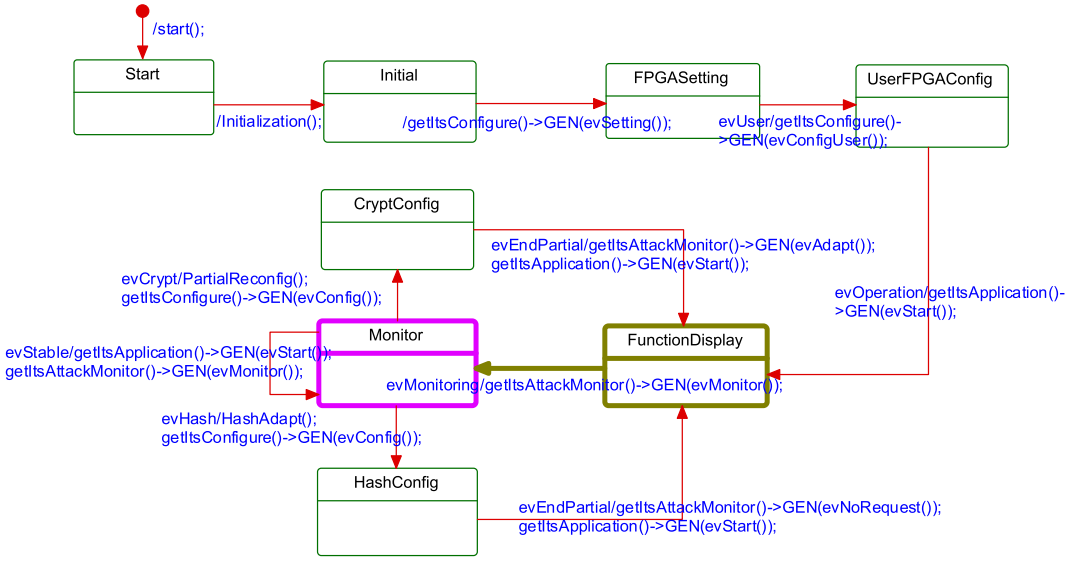

Fig. 7. State machine diagram for system controller class.

and Ada code from UML class and state machine models. The capability of simulating UML models in Rhapsody is very helpful in analyzing system functionalities and component model relationships. By using the specific action definition in Rhapsody, the transitions in different state machine diagrams can trigger each other, as long as, the classes are associated with each other. A transition could include a triggering event and at least one action. The event naming convention is *evName*. The actions are the function operations, the interaction with another state machine diagram, or both. Take the transition from the `Initial` state to the `ClockFPGAConfig` state of the `SystemController` class in Fig. 7 as an example. The action *getItsConfigure*() → *GEN*(*evConfigClock*) on this transition generates an `evConfigClock` event that triggers the transition from `Initial` state to the `ClockFPGAConfig` state in the `Configure` state machine diagram. After integrating the operations of the software application models and the system management models, we use the code generation capability of Rhapsody to generate a software execution framework for DPRNSS. Furthermore, we use sequence diagrams to analyze the functional interactions among the classes in temporal order. The dynamic adaption of security levels based on requests from the `AttackMonitor` class in DPRNSS is illustrated by the sequence diagram in Fig. 13, which shows how a partial reconfiguration is triggered upon request and the newly configured triple DES is used to do the next encryption. In the animation mode of Rhapsody, the state machine and sequence diagrams are simulated to validate the operations of DPRNSS. Thus, the functional interactions among the system components can be validated more intuitively, instead of only inspecting models. For example, the highlighted states in the state machine diagram at Fig. 7 indicate the current execution status.

Through the software framework generated by Rhapsody for the DPRNSS the system functionalities can be validated; however, the cryptographic and hash hardware designs are abstracted in the UML models, which means system correctness associated with physical constraints cannot be guaranteed. For example, the inaccurate time estimation methods for cryptographic and hash hardware designs could guarantee that data transfers with a specific *Quality of Service* (QoS) can be achieved; however, in reality it is not, which could cause a very serious problem, especially when hard real-time constraints are violated. Though the partial reconfiguration for cryptographic and hash hardware designs are requested from two different controllers, due to the architecture constraints, only one hardware function can be partially reconfigured into the FPGA at a time. The system correctness is hard to be guaranteed especially when the reconfiguration requests cannot be expected because they are created according to the current network status.

To provide accurate performance measurements and to meet physical constraints, the platform FUSE APIs and the PCI drivers provided by the XtremeDSP Development Kit-II must be integrated with the software framework of DPRNSS as shown in Fig. 8. Instead of doing this per application, UCoP integrates them directly into the code generator of Rhapsody, so that the gap between application models and system architecture constraints is effectively bridged. As far as application models are concerned, the operations for interacting with the hardware architecture must be modified to follow the control method required by the FUSE APIs and the PCI

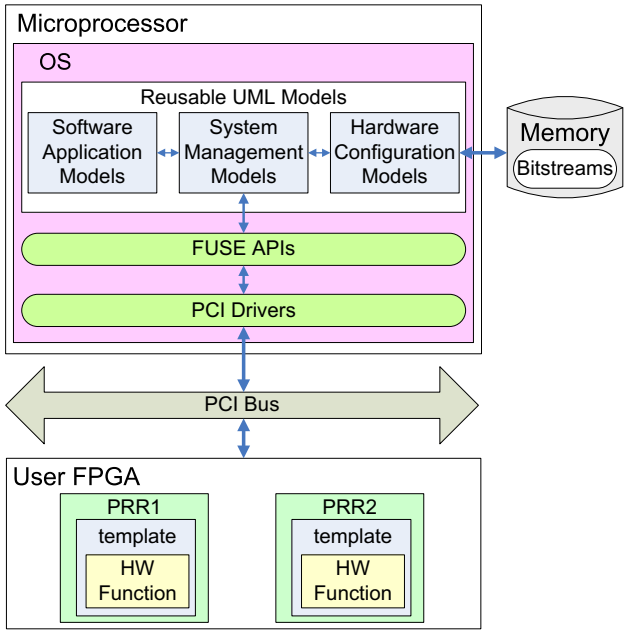

Fig. 8. UML-based HW/SW co-design platform.

drivers. Due to such an integration, through the animation mode of Rhapsody, the functional interactions among the three models and the real system hardware architecture can be, step by step, traced in the sequence diagrams and the state machine diagrams. The models can be rectified and then validated again efficiently. This *direct interaction* with a real system hardware architecture by the high-level application models allows users to analyze and validate the system performance and correctness at a high-level phase, which is very helpful, especially when the system is an adaptive one.

5. Experiments

The UCoP platform was implemented on the XtremeDSP Development Kit-II board connected to a host computer (Intel Core 2 1.86 GHz, 2GB RAM, Windows XP) over the PCI interface. The system hardware architecture of the DPRNSS was created in the Xilinx Virtex-II XC2V3000 FPGA (`User FPGA`), which contains 14,336 slices, 28,672 Flip-Flops, 28,672 LUTs, and 96 block RAMs. Four hash hardware designs, including three variants of CRC and an MD5, and four cryptographic hardware designs, including RSA, DES, triple DES, and AES, were used in the DPRNSS. The other two FPGAs, including `Clock FPGA` and `Interface FPGA`, are specific to the XtremeDSP development Kit-II board, and are used for the configuration of its clock and communication interface, respectively. Their measurements are not included in all experiments.

The DES implementation based on *Electronic Code Book* (ECB) cryptographic mode contained a regular structure that lends itself to pipelining and simple data manipulations to permit fast operations. The DES design was adopted in the triple DES design, where each round of DES in the triple DES implementation was pipelined in three stages. Both the DES and triple DES hardware designs were acquired from Xilinx [20]. The AES implementation followed the FIPS-197 document [3] and was based on ECB cryptographic mode. Its architecture was not pipelined and hence only able to perform multi encryption/decryption serially one after the other. A square-and-multiply algorithm was used in the RSA implementation, where the message value was squared for each bit of the exponent. The MD5 implementation did not include the block padding, and the 128 bits blocks padded needed be input in little endian mode. The AES, RSA, and MD5 hardware designs were acquired from OpenCores [2]. Three variants of the CRC hardware design were generated using the CRC tool developed by Easisc [7]. All the above hardware designs were implemented as partially reconfigurable hardware designs.

Different security requirements could be met by configuring the hash and cryptographic hardware designs dynamically into the DPRNSS. To make efficient use of hardware resources, the PRMs of similar sizes will be configured into the same PRR. In our current implementation, the three variants of CRC hardware designs can be individually (re)configured in `PRR1`, while the MD5 and the four cryptographic hardware designs can be individually (re)configured in `PRR2`. Due to the resource limitations of our current implementation, the MD5 hardware design and one required cryptographic hardware designs cannot be configured in the FPGA at the same time. Thus, the logic resources of the PRR2 are shared with the MD5 designs and four cryptographic hardware designs. To support a complete network security application having a hash and a cryptographic data processing, one method is to store the processing results of the MD5 hardware design in the synchronous FIFO component of `User FPGA`. After reconfiguring one required cryptographic hardware design into PRR2, the saved processing results in the synchronous FIFO component are then transferred to the reconfigured cryptographic hardware design for data encryption. Another alternative method is to realize the cryptographic algorithm as a software application, and thus the processing results of the MD5 hardware design can be transferred to the software cryptographic function for data encryption.

The following experiments will focus on demonstrating our contributions, including an analysis of the resource overhead for integrating the proposed hardware task template with each of the hash and cryptographic hardware designs, a comparison between conventional and dynamically partially reconfigurable network security systems, an accurate high-level validation for the DPRNSS by the direct interactions between UML models and the real system architecture. Furthermore, the proposed UCoP is not restricted to the implementation of network security systems. Rather, users may target a new application by adapting the software application models, integrating and synthesizing their hardware designs with the proposed hardware task template, and using our provided script file to generate the corresponding partial bitstreams.

5.1. Resource overhead analysis for PRMs

As shown in Table 1, the resource overheads incurred by the hardware task template for enhancing all the CRC hardware designs with the capability for partial reconfiguration are less than one percentage of the available FPGA resources.

The implementation results of the five hardware designs that can be configured in `PRR2` are shown in Table 2. For enhancing the five hardware designs with the capability for partial reconfiguration, the PRMs have an average overhead of around 0.6% of the available slices, 0.74% of the available Flip-Flops, and 0.08% of the available LUTs. An exception is that integrating the AES hardware design with the hardware task template require 0.3% lesser LUTs than the original hardware design due to synthesis compiler optimizations. The implementation results in Tables 1 and 2 show that

Table 1
Resource usage for each PRM in PRR1.

HW design		Slices		Flip-flops		LUTs	
		Used (#)	%	Used (#)	%	Used (#)	%
CRC32	Original	97	0.6	32	0.1	177	0.6
	PRM	183	1.2	224	0.7	323	1.1
	Overhead	86	0.6	192	0.6	146	0.5
CRC64	Original	166	1.1	32	0.1	297	1.0
	PRM	254	1.7	224	0.7	459	1.6
	Overhead	88	0.6	192	0.6	162	0.6
CRC128	Original	281	1.9	32	0.1	503	1.7
	PRM	345	2.4	288	1.0	637	2.2
	Overhead	64	0.5	256	0.9	134	0.5

%: the utility rate in terms of all available resources;
Overhead: template overheads in PRM compared to original design.

Table 2
Resource usage for each PRM in PRR2.

HW Design		Slices		Flip-Flops		LUTs	
		Used (#)	%	Used (#)	%	Used (#)	%
MD5	Original	1203	8.3	994	3.4	2227	7.7
	PRM	1333	9.2	1250	4.3	2228	7.7
	Overhead	130	0.9	256	0.9	1	<0.1
RSA	Original	503	3.5	463	1.6	905	3.1
	PRM	625	4.3	687	2.3	1002	3.4
	Overhead	122	0.8	224	0.7	97	0.3
DES	Original	3472	24.2	5302	18.4	5146	17.9
	PRM	3581	24.9	5502	19.1	5226	18.2
	Overhead	109	0.7	200	0.6	80	0.2
3DES	Original	3657	25.5	5571	19.4	5363	18.7
	PRM	3689	25.7	5635	19.6	5419	18.8
	Overhead	32	0.2	64	0.2	56	0.1
AES	Original	8268	57.6	1162	4.0	15,972	55.7
	PRM	8311	57.9	1442	5.0	15,892	55.4
	Overhead	43	0.3	280	1.0	−80	−0.3

%: the utility rate in terms of all available resources;
Overhead: template overheads in PRM compared to original design.

the overheads occurred by the hardware task template are almost negligible, which means the capability for dynamic reconfiguration can be achieved with negligible cost.

5.2. Comparing conventional and dynamically partially reconfigurable network security systems

We compare the conventional network security system with DPRNSS in terms of FPGA resource requirements and static power consumption, where both DPRNSS and NSS receive the same data from the network for processing. To support all the three CRCs, MD5, and the other four cryptographic functions, a conventional NSS requires all the eight functions to be implemented and integrated into the system design, whereas a DPRNSS can support all the eight functions by implementing only two different sized PRRs, namely a small `PRR1` and a large `PRR2`. As shown in Fig. 9, the small `PRR1` configured with CRC32 and the large `PRR2` configured with DES are highlighted for displaying the relative locations in the implementation results of DPRNSS. The overall resource usages for DPRNSS are given in Table 3, where the Spartan–Virtex interface also includes the resources for bus macros and global buffers.

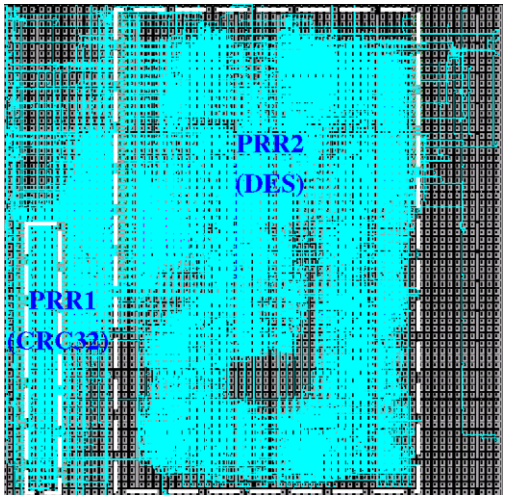

Fig. 9. Implementation result of DPRNSS.

Table 3
FPGA resource usage for UCoP.

HW design	Slices		Flip-Flops		LUTs		BRAMs	
	Used	%	Used	%	Used	%	Used	%
SyncFIFO	60	0.4	70	0.2	83	0.2	4	4.2
SVIface	577	4.0	515	1.7	964	3.3	0	0
PRR1	576	4.0	1152	4.0	1152	4.0	9	9.3
PRR2	8704	60.7	17,408	60.7	17,408	60.7	64	67
Total	9917	69.1	19,145	66.7	19,607	68.3	77	80.3

SyncFIFO: Synchronous FIFO; SVIface: Spartan–Virtex interface; BRAMs: Block RAMs;
%: the utility rate in terms of all available resources.

Note that, due to limited routing resources in the FPGA device, the FPGA resource utilization cannot be increased beyond 70%.

As shown in Table 4, a conventional network security system with all cryptographic and hash hardware designs needs 18,284 slices (127.5%), 14,173 Flip-Flops (49.4%), and 31,673 LUTs (11 0.4%) in terms of the Xilinx Virtex-II XC2V3000 FPGA, without including the switch circuits between all the hardware designs. This shows that all the hardware functions used in the DPRNSS cannot be accommodated in the NSS, when the NSS uses the same amount of logic resources of the XC2V3000 FPGA. However, the proposed DPRNSS needs at most 9293 slices (64.8%), 2315 Flip-Flops (8%), and 17,576 LUTs (61.3%) in terms of the Xilinx Virtex-II XC2V3000 FPGA, which presents the maximal resource usage by the PRMs of CRC128 and AES as given in Tables 1 and 2, respectively. This not only illustrates that DPRNSS can be implemented in the FPGA device, but it is also more efficient in resource usage than NSS.

In a conventional NSS, the non-executing hardware designs cause the problem of static power leakage. However, in a DPRNSS with two PRRs, only one cryptographic hardware design and one hash hardware design exist simultaneously. To perform power comparisons, we measured the static power consumption for each hardware design in the `User FPGA` using the Xilinx XPower tool. The results are given in Table 5 for all the eight hardware designs. Considering the worst case of using maximum power for each of the two PRRs, that is, CRC128 in `PRR1` and 3DES in `PRR2`, the DPRNSS requires 543.30 mW. However, a conventional NSS requires a total static power consumption of 765.87 mW. Thus, DPRNSS results in a power reduction of 29.1%. Furthermore, NSS and DPRNSS need additional time for hardware function switching and for partial reconfiguration, respectively, when they both support all the eight hardware functions. Here, the inter-arrival time of two network attacks must be lesser than the partial reconfiguration time of the required hardware function, and the security of data transfers through DPRNSS can be thus guaranteed. As a result, according to our measurement and estimations, a DPRNSS design can save 62.7% resources (slices) and 29.1% power consumption

Table 4
Resource comparisons between NSS and DPRNSS.

		Hash & Crypt	FIFO	SVI	Total	%
Slices (#)	NSS	17,647	60	577	18,284	127.5
	DPRNSS	8656	60	577	9293	64.8
Flip-Flops (#)	NSS	13,588	70	515	14,173	49.4
	DPRNSS	1730	70	515	2315	8
LUTs (#)	NSS	30,590	83	964	31,637	110.4
	DPRNSS	16,529	83	964	17,576	61.3

Hash & Crypt: the total amount of logic resources for hash and cryptographic hardware designs for NSS and the maximum ones for DPRNSS; FIFO: Synchronous FIFO; SVI: Spartan-Virtex interface;
Total: the total amount of required logic resources; %: the utility rate in terms of all available resources.

Table 5
Static power consumption.

Region	HW design	Power (mW)
PRR1	CRC32	1.77
	CRC64	2.02
	CRC128	2.11
PRR2	MD5	11.67
	RSA	7.42
	DES	108.54
	3DES	115.91
	AES	91.15
Static area		425.28

Table 7
Configuration time for each PRM.

Region	PRMs	Configuration time (ms)
PRR1	CRC32	93
	CRC64	94
	CRC128	94
PRR2	MD5	828
	RSA	766
	DES	859
	3DES	844
	AES	843

compared to NSS. Furthermore, although extra resource overheads incurred by the hardware task template need to be added in the hardware designs, new user-designed hardware functions can be easily integrated in the DPRNSS, that is, the functional scalability of the DPRNSS is higher than that of the NSS. Due to the dynamic adaptation of functionalities, DPRNSS is more flexible to meet changing network environments compared to NSS. Thus, UCoP easily allows such a DPRNSS design to be feasible and efficient. Table 6 gives some empirical comparisons between NSS and DPRNSS.

5.3. Direct interaction with hardware architecture at a high-level phase

Due to the integration of the platform APIs in UCoP, UML models can interact directly with actual hardware designs, and display the results of such run-time interactions in the state machines and sequence diagrams during simulation, instead of only printing the results on a remote terminal. Design errors can be easily detected in UCoP because the interaction results appear within the models, which is very useful for debugging. Thus, the time consuming code instrumentation method for debugging is thus avoided by UCoP.

In conventional UML-based system development methodologies [5,14,17,18], the system performance can only be estimated by simulating the UML models before the system is implemented. In contrast, the support for UML models to directly interact with the real system hardware architecture in UCoP makes system performance evaluation much more efficient and accurate. Efficiency is achieved similar to the hardware-accelerated simulation paradigm, here the partially reconfigurable hardware designs are not simulation models but actual hardware. Accuracy is achieved because the physical constraints in a DPRS are all taken into consideration in UCoP, unlike in commercial methods where most constraints are abstracted in the UML models.

5.3.1. Configuration time analysis

The configuration time for each partial bitstream in the DPRNSS is given in Table 7. We can observe that the configuration times for the PRMs configured in `PRR1` are approximately the same and that for the PRMs configured in `PRR2` are also approximately the same. Note that the reconfiguration time is directly proportionate to the bitstream size, which in turn is directly proportionate to the size of the PRR. A commonly used method to estimate configuration time is based on the size of the synthesized hardware design in terms of the FPGA resource usage, such as LUTs or slices. This estimation method gives accurate results only if the underlying model is 1-dimensional or 2-dimensional [9]. It does not work with the existing modular-design based method promoted by the Xilinx tools, such as PlanAhead. Using the estimation method in [9], the configuration time of the AES hardware design is estimated to be 16 times that of the RSA design because the FPGA resource usage (#LUTs) of the AES hardware design is 16 times that of the RSA hardware design as given in Table 2. However, AES and RSA hardware designs use the same PRR, thus their actual configuration times are similar. In DPRNSS, all cryptographic hardware designs are placed in the same PRR, so the sizes of their partial bitstreams are almost the same. Thus, the reconfiguration times for the cryptographic hardware designs in the `PRR2` are also the same. The synthesis-based estimation method [9] cannot guarantee the timing correctness and the performance of a system until the final system is created. In contrast, by using UCoP the real reconfiguration time is incurred so the users can more accurately analyze the system performance even before the final system is implemented.

Table 6
Comparisons between NSS and DPRNSS.

	Required resources	Static power consumption	Functional scalability	System flexibility
NSS	High	High	Low	Low
DPRNSS	Low	Low	High	High

5.3.2. Security analysis

In DPRNSS, the currently used cryptographic hardware design can be reconfigured into a new one at run-time based on the network security status. The four cryptographic hardware designs, namely RSA, DES, 3DES, and AES, are mapped into four security levels, from low to high, respectively. When the attack monitor detects that the number of network attacks in the previous time window is more than the number of attacks that the current cryptographic hardware design can be susceptible to in a constant time period, the DPRNSS sends a message to the remotely connected system for querying whether it can support another more secure cryptographic function of a higher level of security for future data decryptions. After the negotiation is successful, the DPRNSS starts to reconfigure its cryptographic hardware design into a more secure cryptographic design for future data encryptions.

Note that in a real-time environment, DPRNSS can still be susceptible to network attacks when the inter-arrival time of two network attacks is lesser than the configuration time of the required cryptographic hardware design. As shown in Table 7, the maximum configuration time is 859 ms, which means that if two consecutive network attacks do not occur within 859 ms, then DPRNSS can guarantee security by changing through dynamic reconfiguration.

In the future, the DPRNSS will be implemented on another FPGA device with more logic resources. If a request for partial reconfiguration is received, a required cryptographic hardware design can be pre-configured in another PRR, while the data is still being transferred to the original cryptographic hardware design. After the partial reconfiguration finishes, the original data transfers are then redirected to the new cryptographic hardware design for non-interrupted encryption.

5.3.3. Execution time analysis

Two different methods for the execution time estimation are adopted to show the advantage of the *direct interaction* with the real system hardware architecture in UCoP. In order to analyze the execution process for each cryptographic and hash hardware design, the execution time for each hardware design in the UCoP needs to be first defined. Given input data of D_{in}-bits, output data of D_{out}-bits, data size of D_{pci}-bits for each data transfer iteration over the PCI bus, data write and data read transfer time of δ_{wr} and δ_{rd} microseconds, respectively, for each iteration over the PCI bus, initialization time of T_{pci} microseconds for starting data transfer over the PCI bus, pure execution time of T_e microseconds for a hardware design in UCoP, the total latency is T_{total} as depicted in Equation (1).

$$T_{total} = T_{pci} + \left(\left\lceil \frac{D_{in}}{D_{pci}} \right\rceil \times \delta_{wr}\right) + T_e + \left(\left\lceil \frac{D_{out}}{D_{pci}} \right\rceil \times \delta_{rd}\right) \quad (1)$$

The measured total latency T_{total} includes not only the pure execution time T_e of a processing iteration for a hardware design, but also the time overheads of data transfers over the PCI bus. In [16], the authors measured the time necessary to transfer sequences of 32-bit values for obtaining the lower-bound on data transfers which is then used to estimate system performance. Using the lower-bound estimation method [16], the average time per register access for different numbers of registers over the PCI bus were measured as illustrated in Fig. 10, where *Write/Read* presents that the time taken per operation when writes and reads are interleaved. According to the experimental results in Fig. 10 and the lower-bound estimation method [16], the total latency for each hardware design can be estimated according to its number of register accesses. For example, as shown in Fig. 10, data encryption using the AES hardware design requires 17 register accesses, and the average time per register access is 7.13 μs. As a result, according to the lower-bound estimation method [16], we can estimate the total latency for data encryption using the AES hardware design as 121.21 μs. The total latencies for all the cryptographic and hash hardware designs using the lower-bound estimation method are listed in the fourth column of Table 8. Compared to the total latencies measured in UCoP, as shown in the fifth column of Table 8, the results of the lower-bound estimation method [16] is close to our measured results, but have inaccuracies ranging from −23.3% to 58.2%. This shows the importance of direct interaction with hardware in UCoP.

In UCoP, to obtain the pure execution time T_e for a hardware design in UCoP, we first measured the time interval between the start of data processing by a hardware design and the time when the *done* signal is received by the UML models of UCoP via the PCI bus. This shows that the measured time consists of the pure execution of a hardware design and the latency of a register access. To estimate and eliminate the access latency over the PCI bus, we further conducted extensive experiments for the measurements of the data read time δ_{rd} and the data write time δ_{wr} over the PCI bus as shown in Fig. 10, where we can observe that for larger number of register accesses the average time per register access gradually converges to a constant that is approximately 5700 ns. As a result, the pure execution time T_e for each hardware design in UCoP is the originally measured time exclusive of the access latency over the PCI bus. The pure execution time for all the cryptographic and hash hardware designs measured in UCoP is listed in Table 9.

Another method we proposed in [9] to estimate the hardware execution time considered the hardware simulation and the synthesis results, where the pure execution time for each hardware function was calculated by the number of clock cycles required for a processing iteration and the estimated frequency from its synthesis reports. Compared to the simulation and synthesis-based method, the method proposed in UCoP is more accurate because the execution time was obtained by actually measuring it. For example the method in [9] estimates the execution times for the 3DES decryption and for RSA encryption as 1129 and 9366 ns, respectively, which are both underestimated. Compared to the real execution time as shown in Table 9, the estimation method [9] results in a worst-case inaccuracy of −90.8%. Due to the inaccurate hardware execution time estimations, such as in [9,16], many more iterations between the UML modeling and the implementation phases are needed for the development of a DPRS. In UCoP, the direct interaction of UML models with the hardware architecture gives accurate measurements of the hardware executions, which shows the superiority of UCoP over existing estimation methods.

Fig. 10. Average time per register access.

Table 8
Total latency analysis for each PRM.

Region	HW design	Type	Estimated SF06 [16] (μs)	Measured Eq. (1) (μs)
PRR1	CRC32	Hash	35.78	22.61
	CRC64	Hash	39.42	28.06
	CRC128	Hash	52.10	39.41
PRR2	MD5	Hash	87.12	81.60
	RSA	Encrypt	41.86	53.99
		Decrypt	41.86	54.60
	DES	Encrypt	74.25	75.33
		Decrypt	74.25	74.60
	3DES	Encrypt	97.63	106.92
		Decrypt	97.63	107.10
	AES	Encrypt	121.21	122.01
		Decrypt	121.21	123.81

Table 9
Pure execution time for each PRM.

Region	HW design	Type	Exec time (ns)
PRR1	CRC32	Hash	6609
	CRC64	Hash	6815
	CRC128	Hash	6718
PRR2	MD5	Hash	6687
	RSA	Encrypt	18,090
		Decrypt	18,030
	DES	Encrypt	6078
		Decrypt	6082
	3DES	Encrypt	12,530
		Decrypt	12,323
	AES	Encrypt	6605
		Decrypt	6639

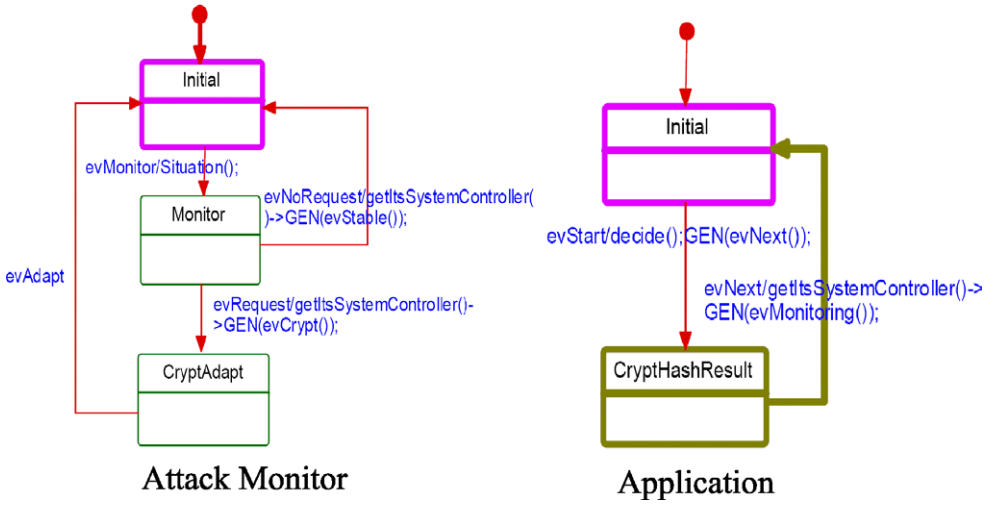

Fig. 11. State machine diagram for attack monitor and application classes.

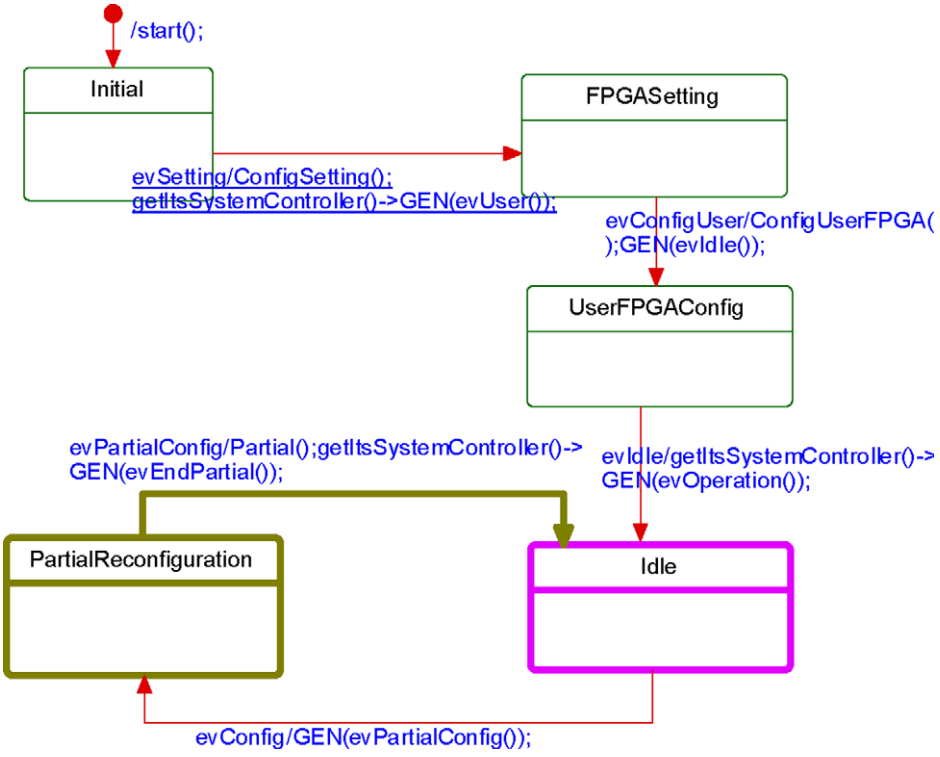

Fig. 12. State machine diagram for configure class.

6. Conclusions and future work

We proposed a *UML-based hardware/software co-design platform* (UCoP) for *dynamically partially reconfigurable network security systems* (DPRSNSS). Compared to the traditional network security embedded system, our proposed hardware architecture not only makes use of the advantages in implementing cryptographic hardware designs in the FPGA, but can also be partially reconfigured at run-time according to different security needs. A partially reconfigurable hardware template was provided for the users to easily integrate their cryptographic hardware designs into the proposed platform, which makes the platform more scalable. Furthermore, the UML models of the platform can *directly* interact with the real system architecture so that the users can more accurately estimate the power consumption, the configuration time and the execution time of each hardware design, and also the performance of the system at a high-level modeling phase, thus significantly reducing the system development efforts.

In the future, we will integrate a SystemC-based estimator for hardware/software design partitioning and task scheduling so that the system development on the proposed platform can be more complete and robust. Moreover, the extensions of a hardware design with the capability for dynamic swapping [9] will be integrated into our partially reconfigurable hardware architecture such that a higher system performance can be achieved.

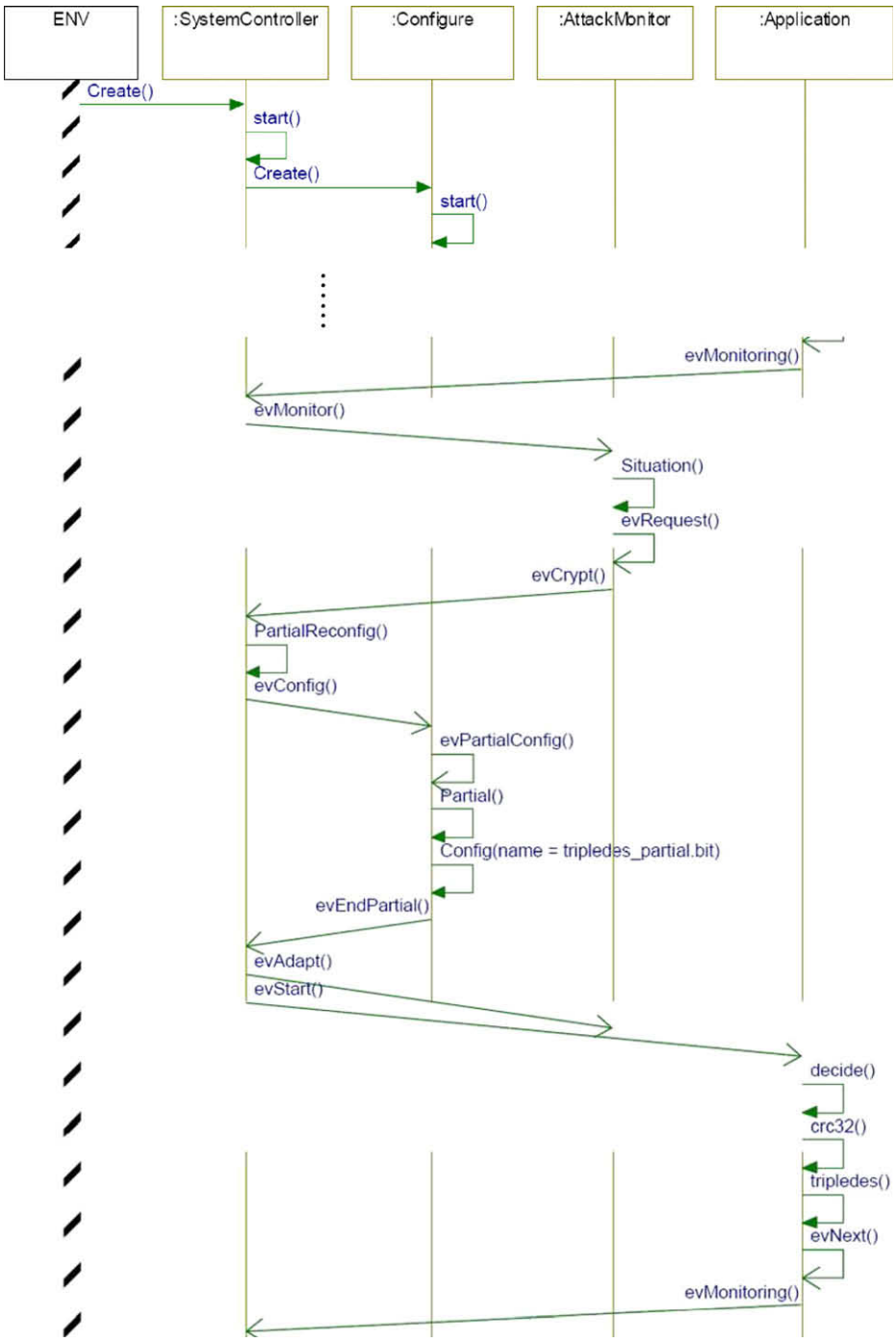

Fig. 13. A sequence diagram for the DPRNSS.

Appendix A. Other UML diagrams for DPRNSS

See Figs. 11–13.

References

[1] Object Management Group (OMG), Unified Modeling Language (UML). <http://www.uml.org>.
[2] OpenCores. <http://www.opencores.com/>.
[3] Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, November 2001.
[4] Rhapsody User Guide. Telelogic Inc. <http://www.telelogic.com>, 2004.
[5] T. Beierlein, D. Fröhlich, B. Steinbach, Model-driven compilation of UML-models for reconfigurable architectures, in: Proc. of the Second RTAS Workshop on Model-Driven Embedded Systems (MoDES04), May 2004.
[6] C.-H. Huang, P.-A. Hsiung, UML-based hardware/software co-design platform for dynamically partially reconfigurable network security systems, in: Proc. of the 13th IEEE Asia-Pacific Computer Systems Architecture Conference (ACSAC), August 2008, doi:10.1109/APCSAC.2008.4625436.
[7] Easics, CRC Tool. <http://www.easics.be/webtools/crctool>.
[8] P. Graf, M. Hübner, K.D. Müller-Glaser, J. Becker, A graphical model-level debugger for heterogenous reconfigurable architectures, in: Proc. of the 17th IEEE International Conference on Field Programmable Logic and Applications (FPL07), IEEE CS Press, 2007, pp. 722–725.
[9] C.-H. Huang, K.-J. Shih, C.-S. Lin, S.-S. Chang, P.-A. Hsiung, Dynamically swappable hardware design in partially reconfigurable systems, in: Proc. of the IEEE International Symposium on Circuits and Systems (ISCAS), IEEE Press, 2007, pp. 2742–2745.

[10] A. Lagger, A. Upegui, E. Sanchez, I. Gonzalez, Self-reconfigurable pervasive platform for cryptographic application, in: Proc. of the 16th IEEE International Conference on Field Programmable Logic and Applications (FPL06), IEEE CS Press, 2006, pp. 777–780.
[11] R. Laue, O. Kelm, S. Schipp, A. Shoufan, S.A. Huss, Compact AES-based architecture for symmetric encryption, hash function, and random number generation, in: Proc. of the 17th IEEE International Conference on Field Programmable Logic and Applications (FPL07), IEEE CS Press, 2007, pp. 480–484.
[12] N. Mentens, K. Sakiyama, L. Batina, I. Verbauwhede, B. Preneel, FPGA-oriented secure data path design: implementation of a public key coprocessor, in: Proc. of the 16th IEEE International Conference on Field Programmable Logic and Applications (FPL06), IEEE CS Press, 2006, pp. 133–138.
[13] Nallatech, XtremeDSP Development Kit-II User Guide (4) (2004).
[14] T. Schattkowsky, W. Mueller, A. Rettberg, A model-based approach for executable specifications on reconfigurable hardware, in: Proc. of the Conference on Design, Automation and Test in Europe (DATE05), IEEE CS Press, 2005, pp. 692–697.
[15] K.-J. Shih, C.-C. Hung, P.-A. Hsiung, Reconfigurable hardware module sequencer – a tradeoff between networked and data flow architectures, in: Proc. of the IEEE International Conference on Field-Programmable Technology (ICFPT'07), 2007, pp. 237–240.
[16] M.L. Silva, J.C. Ferreira, Support for partial run-time reconfiguration of platform FPGAs, Journal of Systems Architecture 52 (12) (2006) 709–726.
[17] B. Steinbach, C. Dorotska, D. Fröhlich, Automated hardware-synthesis of UML-models, in: Proc. of the Second International DAC Workshop, UML for SoC Design (UML-SOC'2005), Springer-Verlag, 2005.
[18] C.-H. Tseng, P.-A. Hsiung, A UML-based design flow and partitioning methodology for dynamically reconfigurable systems, in: Proc. of the Third International DAC Workshop, UML for SoC Design (UML-SOC'2006), Springer-Verlag, 2006.
[19] T. Wollinger, C. Paar, How secure are FPGAs in cryptographic applications, in: Proc. of the 13th IEEE International Conference on Field Programmable Logic and Applications (FPL03), Springer-Verlag, 2003, pp. 1–3.
[20] Xilinx, XAPP270 – High-Speed DES and Triple DES Encryptor/Decryptor, 2001. <http://www.xilinx.com>.
[21] Xilinx. XAPP290 – Two Flows for Partial Reconfiguration Module-Based or Difference-Based. <http://www.xilinx.com>, 2004.
[22] Xilinx. UG208 – Early Access Partial Reconfiguration User Guide. <http://www.xilinx.com>, 2006.

Chun-Hsian Huang received his B.S. degree in Information and Computer Education from National Tai-Tung University, TaiTung, Taiwan, ROC, in 2004. He is currently working toward his Ph.D. in the Department of Computer Science and Information Engineering at National Chung Cheng University, Chiayi, Taiwan, ROC. He is a teaching and research assistant in the Department of Computer Science and Information Engineering at National Chung Cheng University. His research interests include dynamically partially reconfigurable systems, UML-based hardware/software co-design methodology, hardware/software co-verification, and formal verification.

Pao-Ann Hsiung, Ph.D., received his B.S. in Mathematics and his Ph.D. in Electrical Engineering from the National Taiwan University, Taipei, Taiwan, ROC, in 1991 and 1996, respectively. From 1996 to 2000, he was a post-doctoral researcher at the Institute of Information Science, Academia Sinica, Taipei, Taiwan, ROC. From February 2001 to July 2002, he was an assistant professor and from August 2002 to July 2007 he was an associate professor in the Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan, ROC. Since August 2007, he has been a full professor. He was the recipient of the 2001 ACM Taipei Chapter Kuo-Ting Li Young Researcher for his significant contributions to design automation of electronic systems. He was also a recipient of the 2004 Young Scholar Research Award given by National Chung Cheng University to five young faculty members per year. He is a senior member of the IEEE, a senior member of the ACM, and a life member of the IICM. He has been included in several professional listings such as Marquis' Who's Who in the World, Marquis' Who's Who in Asia, Outstanding People of the 20th Century by International Biographical Centre, Cambridge, England, Rifacimento International's Admirable Asian Achievers (2006), Afro/Asian Who's Who, and Asia/Pacific Who's Who. He is an editorial board member of the International Journal of Embedded Systems (IJES), Interscience Publishers, USA; the International Journal of Multimedia and Ubiquitous Engineering (IJMUE), Science and Engineering Research Center (SERSC), USA; an associate editor of the Journal of Software Engineering (JSE), Academic Journals, Inc., USA; an editorial board member of the Open Software Engineering Journal (OSE), Bentham Science Publishers, Ltd., USA; an international editorial board member of the International Journal of Patterns (IJOP). He has been on the program committee of more than 50 international conferences. He served as session organizer and chair for PDPTA'99, and as workshop organizer and chair for RTC'99, DSVV'2000, and PDES'2005. He has published more than 160 papers in international journals and conferences. He has taken an active part in paper refereeing for international journals and conferences. His main research interests include reconfigurable computing and system design, multi-core programming, cognitive radio architecture, System-on-Chip (SoC) design and verification, embedded software synthesis and verification, real-time system design and verification, hardware-software co-design and co-verification, and component-based object-oriented application frameworks for real-time embedded systems.

Jih-Sheng Shen received his B.S. and his M.S. in Computer Science and Information Engineering from the I-Shou University and the National Chung Cheng University, Taiwan, ROC, in 2003 and 2004, respectively. His M.S. thesis was on the design and implementation of on-chip crossroad communication architectures for low power embedded systems. He is currently pursuing his Ph.D. in the Department of Computer Science and Information Engineering at the National Chung Cheng University, Taiwan, ROC. His research interests include the theories and the architectures of reconfigurable systems, machine learning strategies, Network-on-Chip (NoC) designs, encoding methods for minimizing crosstalk interferences and dynamic power consumption.