Model-Based Verification and Estimation Framework for Dynamically Partially Reconfigurable Systems

Chun-Hsian Huang, Jih-Sheng Shen, and Pao-Ann Hsiung†
Department of Computer Science and Information Engineering
National Chung Cheng University, Chiayi, Taiwan−621, ROC
huang@cs.ccu.edu.tw; sjs92@cs.ccu.edu.tw; †pahsiung@cs.ccu.edu.tw

***Abstract*—Compared to the verification of a conventional embedded system, that of a dynamically partially reconfigurable system is more complex and difficult because the requirements for partial reconfiguration changes with the time and environment. Thus, a complete and efficient verification mechanism is urgently required. Different from the non-exhaustive simulation-based verification used in most UML-based design flow for dynamically partially reconfigurable systems, we propose a model-based verification and estimation framework (MOVE) that covers the function-oriented platform independent verification and the physical-aware platform specific verification based on the model-driven architecture (MDA) design. User-specified UML models are first exhaustively verified using model checking and then integrated into a UML-based hardware/software co-design platform (UCoP) for physical-aware verification and estimation at a high abstraction level. Our experiments also demonstrate that the state-space-explosion problem does not occur when model checking reconfigurable systems using MOVE. Furthermore, we can have more accurate time measurements using UCoP which can be used to validate system correctness and performance at the system level compared to the existing synthesis-based and lower-bound estimation methods.**

***Index Terms*—UML, model checking, system verification, performance estimation, reconfigurable systems**

I. Introduction

FPGA 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 FPGA can be reconfigured, while other parts remain operational without being affected by reconfiguration. Using the partial reconfiguration technique, multiple combinations of hardware functions can be accommodated on an FPGA at different time points. 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 [17]. Due to the more flexible and scalable hardware capabilities, the development and validation of a DPRS becomes more complex than that of a common embedded system.

For the functional design of a DPRS, the *Unified Modeling Language* (UML) [1], an industry de-facto standard, has been used for modeling and development [14], [16]. Through system modeling, the functional interactions between the system and the applications can be easily described and analyzed. Most of these works are simulation-based and focus only on the functional code generation without any design-space exploration. However, the simulation-based methods cannot guarantee that all system behaviors are tested and corrected, which results in significantly more iterations for rectifying the system design. Further, most UML-based design methodologies use time estimates for simulating the functional interactions between applications and a system. As a result, the physical design correctness can be verified and estimated only after the UML models are synthesized into concrete system designs. To provide an efficient verification and accurate estimation mechanism, we propose a *MOdel-based Verification and Estimation framework* (MOVE) for dynamically partially reconfigurable systems as illustrated in Figure 1. Based on the *Model-Driven Architecture* (MDA) based UML design flow [1], MOVE supports both *Platform Independent Verification* (PIV) and *Platform Specific*

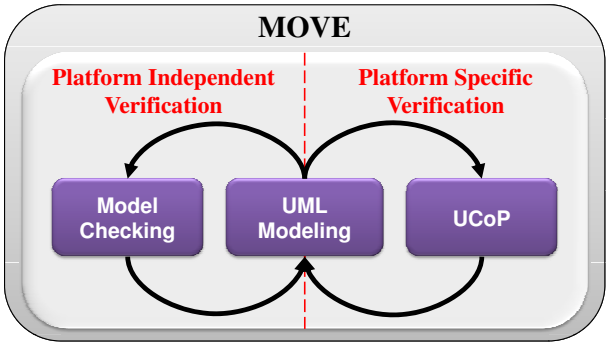


Fig. 1. Model-Based Verification and Estimation Framework for DPRS

Verification (PSV).

In PIV, we apply model checking [6] which allows exhaustive exploration of the DPRS state space to efficiently and completely verify the DPRS functionalities that change with time and environment. Thus, the problems of only being able to simulate partial system behaviors using the simulation-based methods, i.e non-exhaustive verification, can be solved. Furthermore, all reconfigurable hardware functions for the same partially reconfigurable region on an FPGA have the same interactive interfaces, which shows that the complexity of a DPRS architecture design is dependent on the number of partially reconfigurable regions and not on that of reconfigurable hardware functions. As a result, even though the number of reconfigurable hardware functions increases, the state-space explosion problem [1] [6] does not occur in the validation of a DPRS when model checking is used. In PSV, a *UML-based hardware/software Co-design Platform* (UCoP) [5], [10] is proposed to support physical-aware verification and estimation of reconfigurable systems. 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. As a result, designers can accurately validate system correctness and performance at a high abstraction level.

Different from most UML-based verification and estimation methods [14], [16], given the UML models of a DPRS, MOVE supports PIV through functional verification using model checking and PSV through physical-aware verification using UCoP. Due to the two-phase verification process, MOVE significantly reduces DPRS development efforts. The rest of the article is organized as follows: Section II discusses the related UML-based design methodologies for DPRS and the existing DPRS verification and estimation methods. Section III introduces the basic system components and the design constraints in a DPRS. The methodology adapted in the MOVE framework is described in Section IV, while the detailed design of MOVE is introduced in Section V. Section VI presents our experimental results and analysis. Finally, conclusions are described in Section VII.

[1]The state space of a system becomes very large, or even infinite. Thus, it is impossible to explore the entire system behaviors.

II. Related Work

Compared to a conventional embedded system, a DPRS manages not only traditional software applications and hardware devices, but also reconfigurable hardware functions. In a DPRS, the hardware/software functional combination can be dynamically adapted to fit run-time system requirements. As a result, the design of a DPRS is much more complicated than that of a conventional embedded system. To efficiently analyze system behaviors, some research works include high-level UML modeling for the system development. Beierlein et al. [16] proposed a complete UML-based design methodology for reconfigurable architectures, which started with UML models and ended with the final implementation and deployment. It included a model compiler for reconfigurable architectures to create executable applications from system-level specifications. Schattkowsky et al. [14] also proposed a model-based approach for executable UML 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. Similar to the development of a conventional embedded system, the simulation-based method is widely used for the DPRS verification, in which random patterns are used to test the system functional correctness.

Brito et al. [4] proposed a SystemC-based methodology for modeling and simulating a DPRS.

The system specifications were refined to the *Register Transfer Level* (RTL) SystemC for verification. The specific operations for dynamic partial reconfiguration were implemented in the SystemC kernel, and thus the behaviors in a DPRS were directly simulated using SystemC. However, the problems of only being able to simulate partial system behaviors, i.e non-exhaustive verification, still exists in the SystemC-based method [4], which can incur much more design iterations for rectifying and validating a system design. Instead of system verification using only simulation, an integrated design and verification methodology called Symbad [3] was proposed for reconfigurable systems. A reconfigurable system was first described using SystemC and then abstracted for formal verification. However, the SystemC descriptions used for model checking are not intuitively translated because they are based on different design points of view, and thus more design efforts are needed for model translation. Furthermore, the partial reconfiguration technology was not supported in their methodology.

Besides the functional verification of a DPRS, a commonly used synthesis-based estimation method [11] was proposed to evaluate the values of non-functional parameters of a DPRS. The configuration time was evaluated based on the size of the synthesized hardware function in terms of the FPGA resource usages, while the hardware execution time was evaluated based on the hardware simulation and the synthesis results. A lower-bound estimation method [15] was also proposed to evaluate the hardware execution time, where the time necessary to transfer sequences of 32-bit values was measured for obtaining the lower bound on data transfers and then used to estimate system performance. The estimated results using the synthesis-based and lower-bound methods were very close to the actual measured ones; however, the time under-estimation or over-estimation could still cause a very serious problem, especially when hard real-time constraints are violated.

In contrast to the existing UML-based methodologies [14], [16] for the DPRS verification using simulation, MOVE provides a more complete and efficient UML-based validation mechanism that supports platform independent verification (PIV) and platform specific verification (PSV). Instead of the non-exhaustive and time consuming simulation-based method, such as using SystemC [4], model checking is applied in PIV for providing exhaustive validation of a DPRS. Different from the complex translation between SystemC languages and the formal models in Symbad [3], in MOVE, the translation of model-based UML state machine diagrams into *Extended Timed Automata* (ETA), which are used for describing system behaviors in model checking, is more intuitive and straightforward. Compared to the inaccurate synthesis-based [11] and lower-bound [15] methods for DPRS time estimations, a UML-based hardware/software co-design platform (UCoP) is used in PSV for supporting the direct interactions between the UML models and a real DPRS hardware/software architecture, and thus accurate verification and estimations can be achieved at the system level.

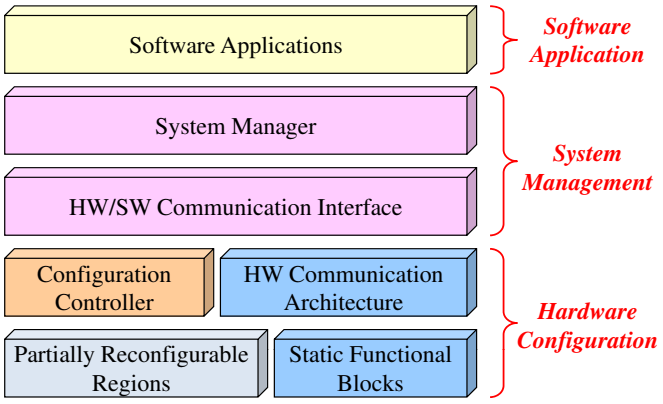


Fig. 2. DPRS Architecture

III. Dynamically Partially Reconfigurable System

Before introducing the *MOdel-based Verification and Estimation* (MOVE) framework, we first describe the design of a *Dynamically Partially Reconfigurable System* (DPRS), which is a hardware/software embedded system capable of reconfiguring new hardware functions into the system at run-time. According to the DPRS architecture design as shown in Figure 2, we can classify the DPRS components into three main categories, including hardware configuration, system management, and software application.

The hardware configuration category contains all the static and reconfigurable hardware components.

The reconfigurable area in an FPGA consists of several slots called *Partially Reconfigurable Regions* (PRRs), which can be reconfigured into different hardware functions at run-time. The static area in an FPGA includes the static functional blocks that cannot be reconfigured at run-time, a hardware communication architecture that connects all hardware components in an FPGA, and a configuration controller, such as *Internal Configuration Access Port* (ICAP) or SelectMap, that is embedded in the FPGA for configuring the partial bitstreams into PRRs. The system management components are responsible for managing the control and data transfers between hardware and software in a DPRS. It mainly includes a system manager and a hardware/software communication interface which includes the device drivers for hardware and system communication devices, such as *Peripheral Component Interconnect* (PCI). The software application category includes all user-specified application functions.

When we design a DPRS, there are mainly two physical constraints imposed by the partial reconfiguration technology as described in the following.

- **Mutual exclusion**: (1) only one hardware function can be configured at a time into a PRR; (2) only one PRR can be reconfigured at a time.
- **Invariable access**: the software applications cannot interact with a PRR when it is being reconfigured.

IV. Methodology in MOVE

A typical *Model-Driven Architecture* (MDA)-based UML design flow can be divided into the platform independent phase and the platform specific phase. In the platform independent phase, the UML models are used to analyze the functional interactions among system components without the platform-related features, such as the hardware configuration time and execution time. In the platform specific phase, the platform-related features are integrated into the UML models for synthesizing the final concrete system design. Similar to the conventional embedded system development, the simulation-based verification and estimation is usually used in the related UML-based DPRS methodologies [14], [16]. However, the simulation-based methods being non-exhaustive can verify only part of the functional behaviors in a system, with rough estimates of the time delays incurred by configuration and execution. As a result, much more design iterations between UML modeling and system implementation are required, which significantly increase the DPRS development time and efforts.

Based on the typical DPRS design as described in Section III, the UML models of MOVE are classified into three categories of *functional UML models* [10], namely hardware configuration models, system management models, and software application models. They are used to model the functional behaviors of a DPRS without the platform-related information, such as hardware configuration and execution time. A DPRS is first specified by customizing the three categories of *functional UML models*, high-level functional analysis can be thus performed in MOVE. To meet the characteristics of the MDA-based UML design flow, MOVE supports the two-phase verification process for a DPRS, including the function-oriented platform independent verification and the physical-aware platform specific verification, instead of system verification using only simulation. In the following subsections, the detailed information for each verification phase is illustrated.

A. Platform Independent Verification

The platform independent verification (PIV) focuses on verifying the functional interactions among DPRS components, without the platform-related information. In PIV, we adopt model checking [6] as the verification method for exhaustively validating the DPRS functional behaviors. However, the UML state machine diagrams cannot be accepted as system model input by most model checkers, which can accept only flat automata model. Thus, the state concurrency and hierarchy in the UML state machine diagrams must be transformed into semantically equivalent constructs in ETA. A flattening scheme [12] is applied in PIV as follows.

1) *Concurrency*: Each AND state in a UML state machine diagram is transformed into an equivalent set of concurrent ETA, and concurrency is thus preserved. Transitions entering or exiting an AND state must be synchronized so that all initial or history states in an AND state are entered simultaneously or all active states left simultaneously. In the

model translation process, a new initial mode was introduced into each component ETA and incoming and outgoing transitions could thus be synchronized.

2) *Hierarchy*: An OR state represents a lower hierarchy level in the UML state machine diagram. In the model translation process, this hierarchy is flattened out by embedding the lower hierarchy level of the UML state machine diagram into the higher hierarchy level. Thus, the incoming and outgoing transitions of an OR state must be preserved, so the hierarchy semantics was preserved through flattening.

Besides applying the flattening scheme for model translation, we classify the transitions in the UML state machine diagrams into two types, namely *reconfiguration* and *general*. Reconfiguration transitions are triggered due to the partial reconfiguration requirements, and *general* transitions are the other remaining transitions. To model real-time system behaviors, transitions in ETA have associated urgency types, including *lazy* and *eager*. *Lazy* transitions need not be taken even if their triggers are satisfied, while *eager* transitions are triggered as soon as possible. The model translation process specific for DPRS is described as follows.

- If the type of a transition in the UML state machine diagram is *reconfiguration*, the triggering condition of the corresponding transition in ETA is defined as *True* for direct triggering; otherwise, the triggering event of a transition in the UML state machine diagram is mapped to the triggering condition of the corresponding transition in ETA. By making the *reconfiguration* transitions non-deterministic, all possible functional combinations of a DPRS are verified.
- If the type of a transition in the UML state machine diagram is *reconfiguration*, the corresponding transition in ETA is associated with the *eager* urgency type, so that real-time reconfiguration is correctly modeled.

When all the state machine diagrams of *functional UML models* are translated into ETA, the global state graph obtained by merging all ETA is then model checking. The properties are specified using *Computation Tree Logic* (CTL) [7]. Model checking can show if a DPRS satisfies a CTL property or violates it by giving a counterexample. CTL properties, such as EF, AF, AG, AU, can all be defined [7], and the use of the CTL properties is briefly described as follows, where p and q are atomic observations.

- **Path qualifier**: A, for all paths; E, for some paths.
- **Temporal operators**: Xp, p holds next time; Fp, p eventually holds in the future; Gp, p always holds in the future; pUq, p holds until q holds.

In contrast to the conventional simulation-based verification, we apply model checking to exhaustively verify all possible system functional interactions among DPRS components. We extend the UML state machine diagrams and ETA for supporting the semantics of the DPRS design, and propose a model translation approach to transform the UML state machine diagrams into ETA. To efficiently verify the real-time features for a DPRS, MOVE leverages the urgency semantics of state transitions and the CTL properties.

B. Platform Specific Verification

The platform specific verification (PSV) focuses on verifying the system correctness and performance associated with the information of a target platform. In PSV, a UML-based hardware/software co-design platform (UCoP) [5] as shown in Figure 3 is proposed to support for UML models to directly interact with real hardware functions that are configured at run-time into an FPGA. The *functional UML models* are integrated with platform APIs, software executables, and hardware bitstreams into the *interactive UML models*. During the simulation of UML models, the *interactive UML models* can directly communicate with the hardware functions in FPGA, thus the gap between models and actual implementations is effectively bridged in UCoP. Therefore, the accurate time measurements can be performed and used at the system level, instead of only simulating the functional interactions between applications and a DPRS.

To ease the integration of user-designed hardware functions into the UCoP, a partially reconfigurable hardware task template (PR template) is proposed,

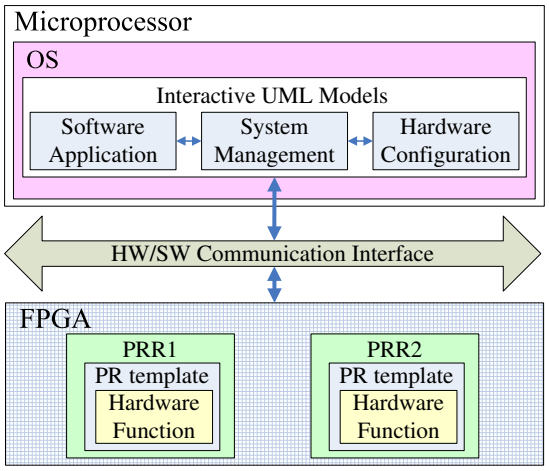


Fig. 3. UML-Based HW/SW Co-Design Platform

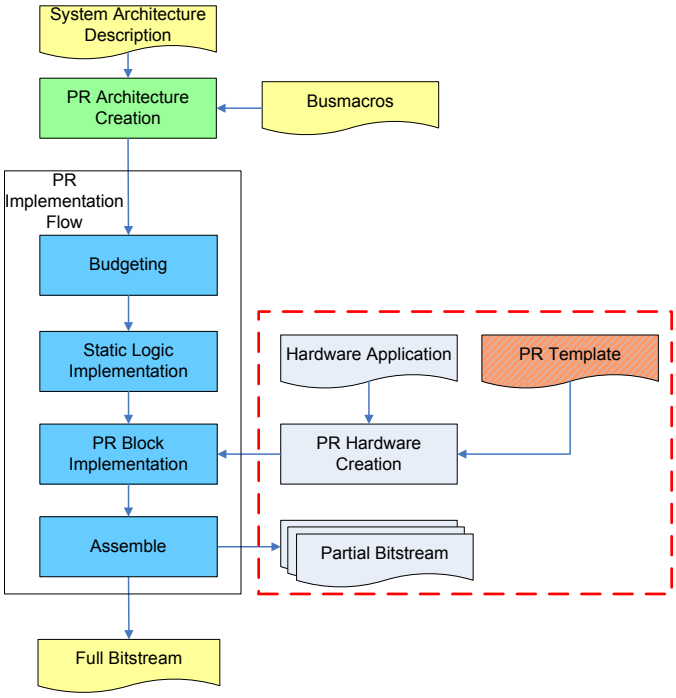


Fig. 4. PR Hardware Creation Flow

which connects the user functions with the hardware communication architecture. To use a newly developed hardware function in UCoP, a designer has to simply integrate the new hardware function with the PR template because it provides a *common* communication interface between the hardware function and the rest of the system. The PR template implements only 32-bit wide signals for all kinds of data transfers. 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. The PR 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 hardware functions.

For implementing the dynamically partially reconfigurable hardware architecture of DPRNSS, the *Early Access Partial Reconfiguration* (EA PR) flow [17] 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 Figure 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 hardware design with the PR 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 the PSV efforts.

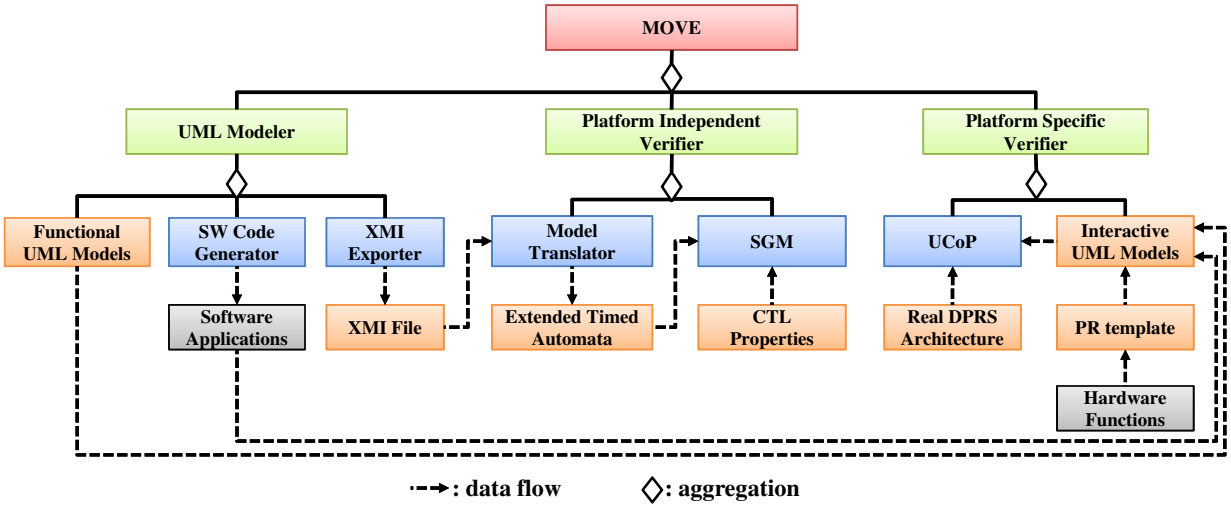


Fig. 5. Component View of MOVE

V. Model-Based Verification and Estimation Framework

The component-based architecture of MOVE, as illustrated in Figure 5, consists of three parts, namely a UML modeler, a platform independent verifier, and a platform specific verifier. Based on the three categories of *functional UML models*, including hardware configuration, system management, software application models, MOVE provides a basic framework for designers to model their DPRS design as shown in Figure 6.

The `SystemManager` class manages all control and data transfers in a DPRS, while `ConfigManager` class manages all FPGA configuration. The `Interactor` class is responsible for providing the interactive interface between software applications and hardware functions. The `ReconfigHW` and `StaticHW` classes are responsible for configuring the functions of PRRs and static area, respectively, in an FPGA. Besides modeling the functional relationships in a DPRS using the class diagram, MOVE also provides the basic frameworks for the state machine diagrams to model the detailed operations for each class except for `ReconfigHW` and `StaticHW` classes because they model passive components containing pointers to the bitstreams that are saved in memory. To translate the UML state machines into ETA, an XMI exporter is used to export the *functional UML models* in the *XML-based Metadata Interchange* (XMI) format.

MOVE integrates the *State Graph Manipulator* (SGM) [8], [9], which is continuingly developed and maintained by our laboratory, into the platform independent verifier. SGM supports CTL model checking, transition urgency types, and dead state checking. The platform independent verifier also includes a model translator to translate the UML models in the XMI file into the corresponding ETA for SGM. The physical architecture constraints for partial reconfiguration, including mutual exclusion and invariable access, as illustrated in Section III, and user-given properties are specified as CTL properties for model checking. When all properties are validated and all functional errors are corrected by analyzing the counterexamples, the *functional UML models* are then input to the platform specific verifier.

MOVE adopts UCoP, which can directly interact with the real DPRS architecture at a high abstraction level, as the platform specific verifier. UCoP was implemented on a reference board, that is, the XtremeDSP Development Kit-II [13] from Nallatech. The software applications run on a microprocessor. The *Field Upgradeable Systems Environment* (FUSE) APIs and the PCI driver are provided by the XtremeDSP Development Kit-II to facilitate the FPGA reconfiguration and communication over the PCI bus. Instead of doing this per application, UCoP integrates them directly into the software code gen-

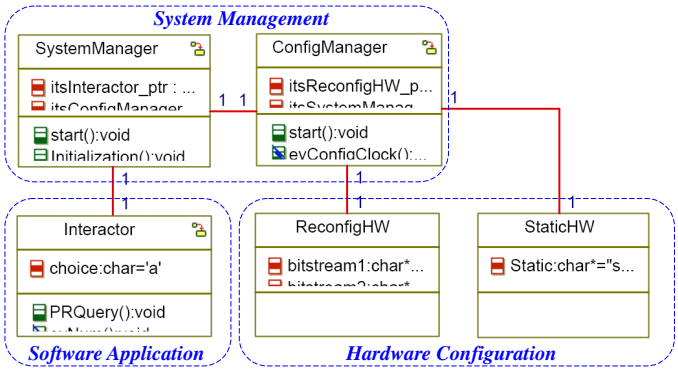


Fig. 6. Class Diagram Framework of MOVE

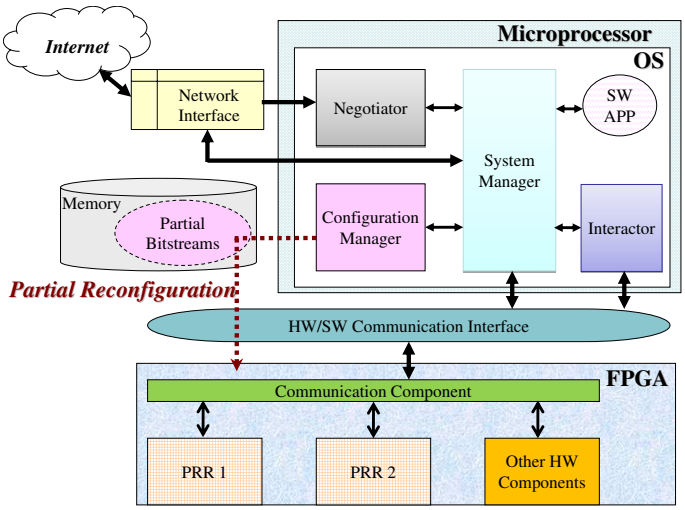


Fig. 7. Dynamically Partially Reconfigurable Network Security System

erator, so that the gap between application models and system architecture constraints is effectively bridged.

All reconfigurable hardware functions are integrated with the PR template and then the corresponding bitstreams are generated. The bitstreams are associated with the hardware configuration models, while software applications are implemented in the software application models. Furthermore, the FUSE APIs and the PCI driver are integrated into the system management models. Due to such an integration, the *interactive UML models* can directly interact with the real DPRS hardware architecture, and thus the accurate verification and estimation can be achieved at the system level.

VI. Experiments

To illustrate how MOVE can be applied to a real system, we use a *Dynamically Partially Reconfigurable Network Security System* (DPRNSS) as our example. As shown in Figure 7, DPRNSS consists of five system devices, including a microprocessor, an FPGA, a network interface, a hardware/software communication interface, and an off-chip memory. For the dynamically partial reconfiguration of cryptographic and hash hardware functions, some PRRs are implemented on the FPGA. All partial bitstreams for cryptographic and hash hardware designs are saved in the off-chip memory.

DPRNSS contains the basic controllers, including a configuration manager, a system manager, an interactor as described in the basic framework of MOVE of Figure 6, and a negotiator to negotiate with a receiver to use the same cryptographic and hash algorithm for data transfers in the network. The software application is a network multimedia application, which receives in real-time 128×128 pixel images from a camera. The received images are transferred to the cryptographic and hash hardware functions for data processing, and then sent to a receiver on the network. To validate DPRNSS using MOVE, the negotiator and the network multimedia application are modeled and then integrated with the software application models in MOVE.

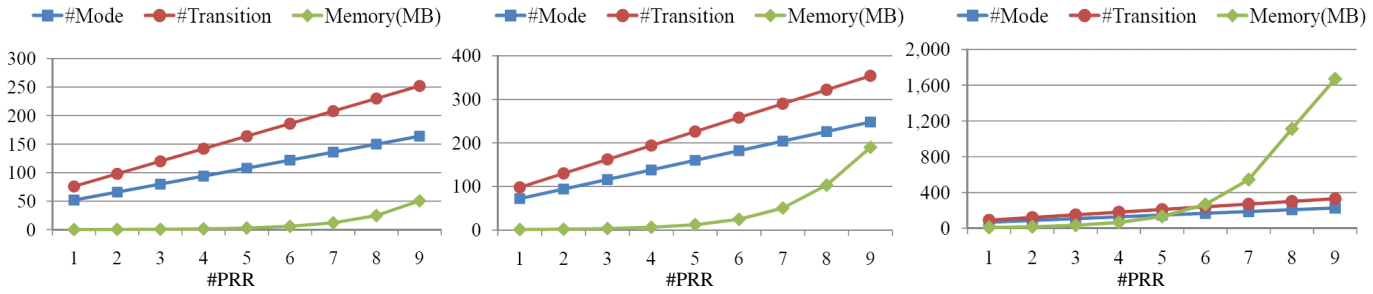
In this experiment, we adopt the Rhapsody modeling tool [2] that has the powerful capability for code generation in C, C++, Java, and Ada, as the UML modeler. Furthermore, we use the XMI toolkit in Rhapsody to export the *functional UML models* in the XMI format.

A. PIV using SGM

After successfully modeling DPRNSS by the *functional UML models*, the model translator is used to transform the UML state machine diagrams into ETA. Here, we use SGM, which runs on an Intel Pentium 4 CPU 3.00GHz with 8 GB RAM, to verify the following physical architecture constraints for partial reconfiguration described in Section III.

Mutual exclusion:

Only one hash or cryptographic hardware



(a) Merged timed automata without clock variables

(b) Microprocessor and FPGA have different frequencies

(c) Microprocessor, FPGA, and configuration controller have different frequencies

Fig. 8. Numbers of ETA Modes and Transitions and Memory Usage

function can be reconfigured at a time. The CTL properties are specified as follows:

"$AG(mode(SystemManager)=CryptConfig \rightarrow evHash=0)$"
"$AG(mode(SystemManager)=HashConfig \rightarrow evCrypt=0)$"

Invariable access:

While one hash or cryptographic hardware function is being reconfigured, the software applications cannot interact with it. The CTL property is specified as follows:

"$AG(mode(ConfigManager)=PartialReconfiguration \rightarrow !\{mode\ (SystemManager)=DataProcessing\})$"

We verify DPRNSS from 1 to 9 PRRs for configuring the cryptographic and hash hardware functions, which shows that DPRNSS becomes more and more flexible to adapt its functionalities at runtime. Figure 8 shows the numbers of ETA modes and transitions of the global DPRS timed automaton and the memory usage by SGM. We verify the functional interactions between DPRNSS components without clock variables as shown in Figure 8(a), where we can observe that the numbers of ETA modes and transitions increase linearly when the number of PRRs increases, and the memory usage increases significantly when the number of PRRs becomes more than 6.

Because the requirements for partial reconfiguration change over time and due to the environment conditions, the temporal feature is considered in our second experiment. The microprocessor and the FPGA have different frequencies in DPRNSS, and thus we assign two different clock variables to the software applications and the hardware functions as shown in Figure 8(b). We can observe that the memory usage increase exponentially when the number of PRRs increases. However, the numbers of ETA modes and transitions still increase linearly when the number of PRRs increases.

In a DPRS, the configuration controller usually has an independent frequency different from the microprocessor and the FPGA to configure the bitstreams. Thus, we assign the third clock variable to the operations for partial reconfiguration as shown in Figure 8(c), where we can observe that the exponential increase of the memory usage becomes more significant than that as shown in Figure 8(b). Though the memory usage increases exponentially when the numbers of clock variables and PRRs increase, the numbers of ETA modes and transitions still increase linearly. This is because of the inherent characteristics of reconfigurable systems, whose complexity is dependent on the number of PRRs, instead of functions. As a result, the state-space explosion problem [6] does not occur in the validation of such a DPRS using SGM. However, to validate such timing requirements, simulation needs much more exhaustive testbench or test vectors. This usually causes much more iterations for rectifying and validating a DPRS design, especially after the system has been implemented. Therefore, MOVE integrates SGM in PIV for providing efficient and exhaustive functional verification, instead of simulation-based methods, and our experiments

have demonstrated that the state-space explosion problem does not occur in MOVE.

B. PSV using UCoP

Two PRRs, namely PRR1 and PRR2, are implemented on a Xilinx Virtex-II XC2V3000 FPGA with 14,336 slices. All cryptographic and hash hardware functions are integrated with the PR template for generating the corresponding partial bitstreams, which are then incorporated with the hardware configuration models of MOVE. Here, the cryptographic hardware functions, including RSA, *Data Encryption Standard* (DES), 3DES, and *Advanced Encryption Standard* (AES), can be configured into PRR2, while the hash hardware functions, including CRC32, CRC64, and CRC128, can be configured into PRR1. As a result, through the animation mode of Rhapsody, the functional interactions among the *interactive UML models* and the real DPRS hardware architecture can be, step by step, dynamically traced in the sequence diagrams and the state machine diagrams. Thus, the accurate verification and estimation can be achieved at the high abstraction level.

1) Resource Usage and Configuration Time: As shown in Table I, the resource overheads incurred by the PR template for all the hardware functions are less than one percentage of the available FPGA resources, which are almost negligible. The configuration time for each hardware function is shown in the fifth column of Table I. We can observe that the configuration times for the hardware functions configured in PRR1 are approximately the same and that for the hardware functions 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.

Using the synthesis-based estimation method [11], the configuration time of the AES hardware function is estimated to be 16 times that of the RSA function because the FPGA resource usage of the AES hardware function is 16 times that of the RSA hardware function as given in Table I. However, it gives accurate results only if the underlying model is 1-dimensional or 2-dimensional [11]. It does not work with the existing modular-design based method promoted by the Xilinx tools, such as PlanAhead. In fact, the AES and RSA hardware functions use the same PRR, thus their actual configuration times are similar. The synthesis-based estimation method [11] cannot guarantee the timing correctness and the performance of a system until the final system is created. In contrast, using MOVE the real reconfiguration time is incurred so the users can more accurately analyze the system performance even before the final system is implemented.

TABLE I
CONFIGURATION TIME AND RESOURCE USAGE

Region	HW Function	Slice		Time (ms)
		Count	Overhead	
PRR1	CRC32	97 (0.6%)	86 (0.6%)	93
	CRC64	166 (1.1%)	88 (0.6%)	94
	CRC128	281 (1.9%)	64 (0.5%)	94
PRR2	RSA	503 (3.5%)	122 (0.8%)	766
	DES	3,472 (24.2%)	137 (0.9%)	859
	3DES	3,657 (25.5%)	33 (0.2%)	844
	AES	8,268 (57.6%)	43 (0.3%)	843

%: the utility rate in terms of all available slices; Overhead: PR template overheads compared to original hardware design

2) Execution Time Analysis: In this experiment, we focus on comparing the execution time estimation for each cryptographic and hash hardware functions using MOVE and that using the lower-bound [15] and the synthesis-based [11] estimation methods, without the stream buffering technology, to fit the requirements for real-time image transfer.

To analyze the execution process for each hardware function in DPRNSS, the execution time for each hardware function needs to be defined first. 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 function in MOVE, the total latency is T_{total}. As shown in Equation (1), the measured total latency includes not only the pure execution time (T_e) of a processing iteration for a hardware function, but also the time overheads of data transfers over the PCI bus. As shown in the fourth and fifth columns of Table II, the estimated total latencies for a 128×128 pixel image cryptographic and hash operation using the

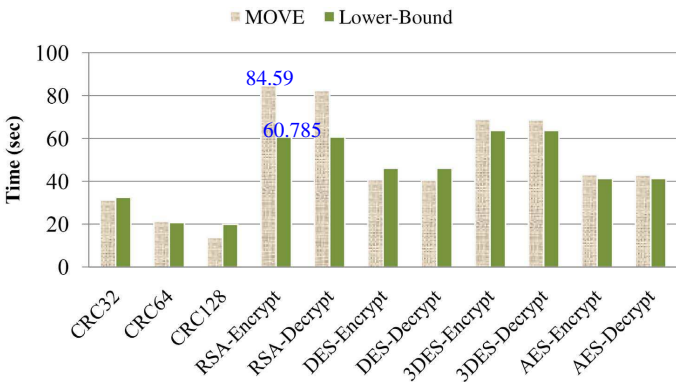


Fig. 9. Total Latencies using MOVE and Lower-bound Estimation

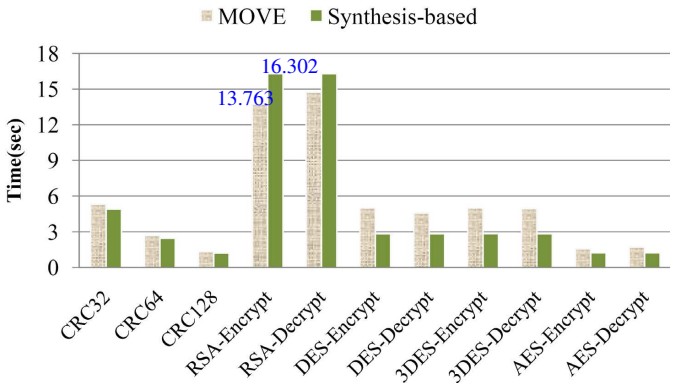


Fig. 10. Execution Time using MOVE and Synthesis-based Estimation

lower-bound estimation method [15] are compared with that using MOVE. The results of the lower-bound estimation method [15] have inaccuracies ranging from -28.1% (RSA encryption) to 47.5% (CRC128).

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

Figure 9 shows the total latencies in seconds for fifty 128×128 pixel image cryptographic and hash operation using MOVE and the lower-bound estimation method [15]. In a networked multimedia application using the RSA encryption with a *Quality of Service* (QoS) for 50 image frames per 80 seconds, the results based on the lower-bound estimation method indicates that the RSA encryption for 50 images needs 60.785 seconds, that is, the QoS can be achieved; however, as measured by MOVE (the RSA encryption for 50 images needs 84.59 seconds), the QoS cannot be achieved in reality.

To measure the pure execution time for a hardware function without considering data write and data read transfer time over the PCI bus, we can use Equation 2, which is modified from Equation 1. As shown in the sixth and seventh columns of Table II, the estimated execution time for a 128×128 pixel image cryptographic and hash operation using the synthesis-based estimation method are compared with that using MOVE. The results of the synthesis-based estimation method [11] have inaccuracies ranging from -43.4% (DES encryption) to 18.4% (RSA encryption).

$$T_e = T_{total} - (T_{pci} + (\lceil \frac{D_{in}}{D_{pci}} \rceil \times \delta_{wr}) + (\lceil \frac{D_{out}}{D_{pci}} \rceil \times \delta_{rd})) \quad (2)$$

Figure 10 shows the execution time in seconds for fifty 128×128 pixel image cryptographic and hash operation using MOVE and the synthesis-based estimation method [11]. In a networked multimedia application using the RSA encryption with a QoS for 50 image frames per 15 seconds, the design results based on the synthesis-based estimation method shows that the RSA encryption for 50 images needs 16.302 seconds, that is, the QoS cannot be achieved; however, in reality it can as measured in MOVE (the RSA encryption for 50 images needs 13.763 seconds). The time inaccuracy could cause a very serious problem, especially when hard real-time constraints are violated. Compared to the inaccurate lower-bound and synthesis-based estimation methods, MOVE provides the exact measured timing results. Our experiments also demonstrate that the use of MOVE for PSV is very helpful to designers for verifying and estimating system correctness and performance at the system level.

VII. Conclusions

Compared to the non-exhaustive and time-consuming simulation-based verification and estimation used in the related UML-based DPRS methodologies, based on the MDA-based UML design flow, both the function-oriented PIV and the physical-aware PSV are supported in MOVE. The user-specific *functional UML models* of a DPRS are

TABLE II
TIME COMPARISON FOR EACH HARDWARE FUNCTION

Region	Hardware Function	Type	Total Latency (ms)		Execution Time (ms)	
			Lower-bound	MOVE	Synthesis-based	MOVE
PRR1	CRC32	Hash	652.1	624.5	98.3	106.7
	CRC64	Hash	416.2	424.3	49.1	53.7
	CRC128	Hash	397.3	275.1	24.6	26.7
PRR2	RSA	Encrypt	1,215.7	1,691.8	326.0	275.3
		Decrypt	1,215.7	1,645.9	326.0	294.9
	DES	Encrypt	924.1	816.1	56.5	99.9
		Decrypt	924.1	811.4	56.5	91.8
	3DES	Encrypt	1,274.7	1,377.8	56.8	99.9
		Decrypt	1,274.7	1,372.6	56.5	99.1
	AES	Encrypt	826.6	863.2	25.0	31.5
		Decrypt	826.6	857.8	25.0	34.4

exhaustively verified using model checking, and are then integrated into the *interactive UML models* of UCoP for the accurate verification and estimation at a high abstraction level. Our experiments also demonstrate that the space-explosion problem does not occur in PIV and the accurate verification and estimation can be achieved in PSV compared to the existing synthesis-based and lower-bound estimation methods. Thus, MOVE can provide a very complete and efficient verification mechanism for users to validate their DPRS, which significantly reduces the DPRS development efforts.

VIII. ACKNOWLEDGMENTS

This work was partially supported by the research project NSC96-2221-E-194-065-MY2, National Science Council, Taiwan.

REFERENCES

[1] Object Management Group (OMG). http://www.omg.org.

[2] *Rhapsody User Guide*. Telelogic Inc, 2004. http://www.telelogic.com.

[3] M. Borgatti, A. Capello, U. Rossi, J. L. Lambert, I. Moussa, F. Fummi, and G. Pravadelli. An Integrated Design and Verification Methodology for Reconfigurable Multimedia Systems. In *Proceeding of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05)*, volume 3, pages 266–271. IEEE CS Press, March 2005.

[4] A. Brito, M. Kuhnle, M. Hubner, J. Becker, and E. U. K. Melcher. Modelling and Simulation of Dynamic and Partially Reconfigurable Systems using SystemC. In *Proceeding of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'07)*, pages 35–40, March 2007.

[5] C.-H. Huang and P.-A. Hsiung. UML-Based Hardware/Software Co-Design Platform for Dynamically Partially Reconfigurable Network Security Systems. In *Proceeding of the 13th IEEE Asia-Pacific Computer Systems Architecture Conference (ACSAC)*, August 2008. (doi:10.1109/APCSAC.2008.4625436).

[6] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. MIT Press, 1999.

[7] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic Model Checking for Real-Time Systems. In *Proceeding of IEEE International Conference Logics in Computer Science*, pages 394–406, 1992.

[8] P.-A. Hsiung, S.-W. Lin, Y.-R. Chen, C.-H. Huang, and W. Chu. Modeling and verification of real-time embedded systems with urgency. *Journal of Systems and Software*, 82(10):1627–1641, October 2009. (DOI: http://dx.doi.org/10.1016/j.jss.2009.03.013).

[9] P.-A. Hsiung, S.-W. Lin, Y.-R. Chen, C.-H. Huang, J.-J. Yeh, H.-Y. Sun, C.-S. Lin, and H.-W. Liao. Model checking timed systems with urgencies. In *Proceeding of the 4th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, volume 4218, pages 67–81, October 2006.

[10] C.-H. Huang and P.-A. Hsiung. On the Use of a UML-Based HW/SW Co-Design Platform for Reconfigurable Cryptographic Systems. In *Proceeding of the IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2221–2224. IEEE Press, May 2009.

[11] C.-H. Huang, K.-J. Shih, C.-S. Lin, S.-S. Chang, and P.-A. Hsiung. Dynamically Swappable Hardware Design in Partially Reconfigurable Systems. In *Proceeding of the IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2742–2745. IEEE Press, May 2007.

[12] L. Lavazza. *A Methodology for Formalizing Concepts Underlying the DESS Notation*. EUREKA-ITEA project, 2001.

[13] Nallatech. XtremeDSP Development Kit-II User Guide, Issue 4, 2004.

[14] T. Schattkowsky, W. Mueller, and A. Rettberg. A Model-Based Approach for Executable Specifications on Reconfigurable Hardware. In *Proceeding of the conference on Design, Automation and Test in Europe (DATE'05)*, pages 692–697. IEEE CS Press, March 2005.

[15] M. L. Silva and J. C. Ferreira. Support for Partial Run-Time Reconfiguration of Platform FPGAs. *Journal of Systems Architecture*, 52(12):709–726, December 2006.

[16] T. Beierlein, D. Fröhlich and B. Steinbach. Model-driven Compilation of UML-Models for Reconfigurable Architectures. In *Proceeding of the Second RTAS Workshop on Model-Driven Embedded Systems (MoDES'04)*, May 2004.

[17] Xilinx. UG208 – Early Access Partial Reconfiguration User Guide, 2006. http://www.xilinx.com.