PAPER

A System-Level Network Virtual Platform for IPsec Processor Development

Chen-Chieh WANG[†a)], *Student Member and* **Chung-Ho CHEN**[†b)], *Nonmember*

SUMMARY Developing a complex network accelerator like an IPsec processor is a great challenge. To this end, we propose a Network Virtual Platform (NetVP) that consists of one or more virtual host (vHOST) modules and virtual local area network (vLAN) modules to support electronic system level (ESL) top-down design flow as well as provide the on-line verification throughout the entire development process. The on-line verification capability of NetVP enables the designed target to communicate with a real network for system validation. For ESL top-down design flow, we also propose untimed and timed interfaces to support hardware/software co-simulation. In addition, the NetVP can be used in conjunction with any ESL development platform through the untimed/timed interface. System development that uses this NetVP is efficient and flexible since it allows the designer to explore design spaces such as the network bandwidth and system architecture easily. The NetVP can also be applied to the development of other kinds of network accelerators.

key words: *ESL, IPsec, network simulator, on-line verification, virtual platform*

1. Introduction

Internet Protocol Security (IPsec) [1] is a protocol suite using Authentication Header (AH) and Encapsulation Security Payload (ESP) protocols to provide confidentiality, authentication and an optional anti-replay service for IP packet transmission. Unlike other security protocols such as SSH, SSL/TLS, and WPA, IPsec resides in the network layer to provide universal and transparent operations. This means that the applications need not to be specifically designed for using IPsec. IPsec can be implemented in a host or a gateway system and can be operated in either the transport mode or the tunnel mode, using various encryption and authentication algorithms [2]. In IPv6, IPsec is defined as one of the mandatory functions. It is also widely used to set up Virtual Private Networks (VPNs).

Which one of the three types of IPsec to use depends on where it is deployed. The first type is the integrated-into-the-stack (IITS) in which IPsec is integrated into the IP protocol stack. Implementation of this type usually resorts to a software solution and requires access to the IP source code. The second type is the bump-in-the-stack (BITS), where IPsec is assigned to a separate architectural layer between the native IP and the data link layer. The advantage of the BITS architecture is that IPsec can be retrofitted to any IP device without modifying the IP protocol stack. The third type of IPsec deployment is the bump-in-the-wire (BITW) method in which IPsec can be implemented somewhere out of the host system. That is, it may be built in a stand-alone hardware device like a security gateway and often it is IP addressable. In addition, there are many kinds of configuration in IPsec such as protocols, operation modes, cryptographic algorithms, keys, etc. Consequently, developing and verifying an IPsec processor brings up a great challenge.

Figure 1 (a) shows a traditional IC design flow used to develop and verify a network-related hardware like an IPsec processor. In this case, a high-level functional model may be built to verify the correctness of the algorithms in the system as well as to provide an environment for software/hardware co-design. Thereafter, a designer can utilize a hardware description language (HDL), such as Verilog or VHDL, to design the hardware modules and to integrate these modules into a system-on-a-chip (SoC) platform. In HDL development, a designer can verify a particular module with an off-line approach which requires a golden model developed in advance to generate the correct input and output data for comparisons during HDL simulation. Such an approach is adopted in [3] to verify the IPsec processor. Due to the limit of slow HDL simulation, software design and verification may be delayed until the hardware design has been completed and loaded into an FPGA board for on-line prototyping verification. However, using this approach to develop a complex system may suffer from the poor correctness in the early stage and as a result, a higher cost in the development effort.

In order to shorten the development time, the IC design methodology has evolved over time from the early register transfer level (RTL) design, through the system-on-a-chip (SoC) design, to the current electronic system level (ESL) design [4]. The ESL design methodology, as shown in Fig. 1 (b), aims at modeling the behavior of the entire system using abstract modeling languages such as SystemC [5], [6], and introduces new concepts such as Transaction Level Modeling (TLM) [7] and Event Driven Modeling to provide a fast simulation environment in which software and hardware can be designed and verified simultaneously. In ESL design methodology, a designer can refine his or her design with a top-down approach [8], as shown in Table 1, from layer-3 untimed model, through approximately timed model, to cycle-accurate timed model. Finally, a pin accurate model written in HDL or synthesized SystemC can be obtained.

Manuscript received June 22, 2012.
Manuscript revised December 22, 2012.

[†]The authors are with the Institute of Computer and Communication Engineering, National Cheng Kung University, Taiwan.
a) E-mail: ccwang@mail.ee.ncku.edu.tw
b) E-mail: chchen@mail.ncku.edu.tw

DOI: 10.1587/transinf.E96.D.1095

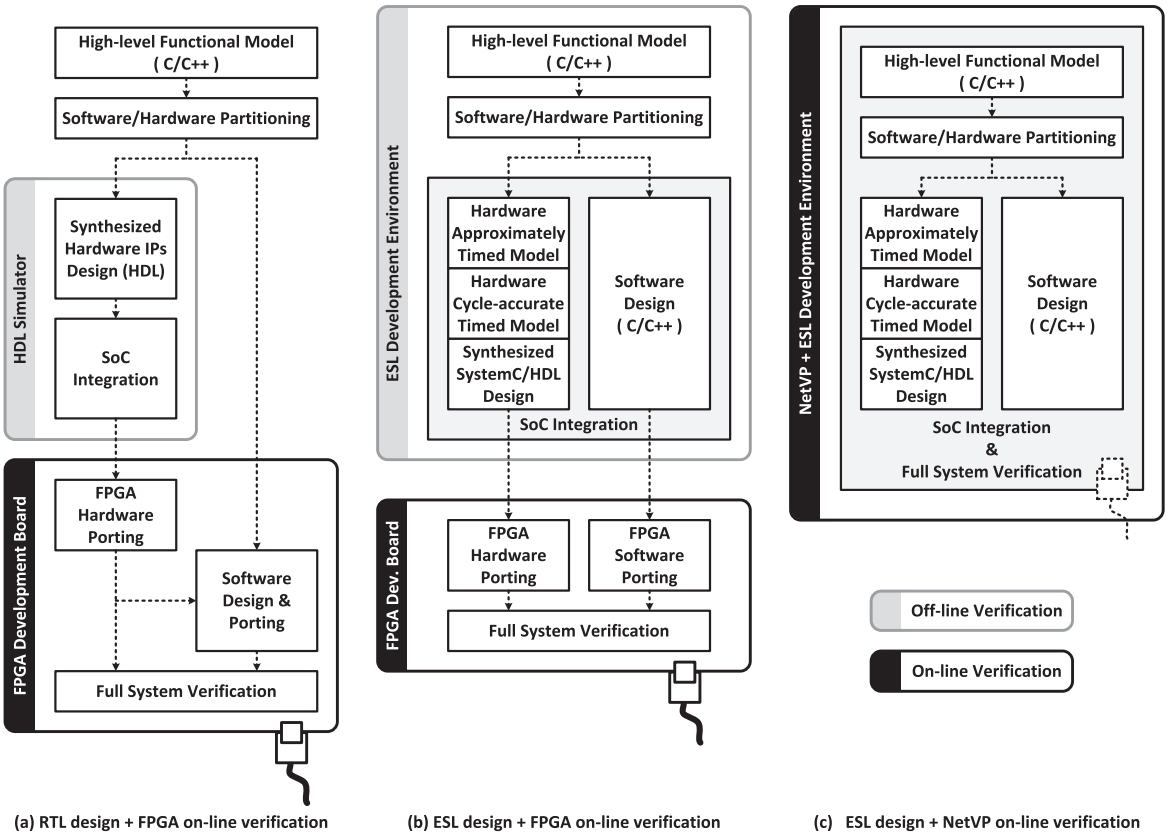


Fig. 1 Comparison of different design and verification flows for network devices.

Table 1 Abstraction layers in ESL top-down design flow.

Abstraction layers	Abstraction removes	Protocol timing	Interface
Layer-3: Message layer	Resource sharing, time	None	Function call
Layer-2: Transaction layer	Clock, protocols	Approximate	Function call
Layer-1: Transfer layer	Wires, registers	Cycle accurate	Function call
Layer-0: RTL layer	Gates, gate/wire delays	Cycle accurate	Signals

For hardware/software co-development, most commercial ESL development tools, such as Platform Architect [9] and SoC Designer [10], provide rich system modules such as CPU, on-chip bus, and memory models that can help users to build their SoC platform quickly and conveniently. However, these development tools have no on-line verification capability for network-related hardware design before the design loaded into an FPGA board. In contrast, most current network simulators, such as ns-3 [11], NetSim [12], OMNeT++ [13], and OPNET [14], provide a virtual network environment in which users can model various kinds of network topology and framework to evaluate their network protocols and algorithms; to the best of our knowledge, these network simulators do not provide the ESL development environment for network-related hardware design. In this paper, we propose a Network Virtual Platform (NetVP) which is able not only to support ESL top-down design flow for network-related hardware development but also to provide the on-line live system verification capability throughout the entire development process, as shown in Fig. 1 (c). The NetVP can connect the system under development to an outside system via a physical network line and form a network golden test bench to improve the efficiency of verification.

This paper is organized as follows. Section 2 presents the system architecture of the Network Virtual Platform (NetVP) and illustrates its design and implementation issues. Section 3 describes how to develop a complex IPsec processor with the NetVP. Section 4 discusses the performance evaluation of the NetVP. Finally, Sect. 5 concludes this paper.

2. Network Virtual Platform (NetVP)

Figure 2 shows the system architecture of the Network Virtual Platform (NetVP) that consists of two virtual host (vHOST) modules and one virtual local area network (vLAN) module. In addition, the NetVP can communicate with the testbench module, i.e., the NetVP_Costar installed on another computer and form an on-line real system verification framework. The vHOST provides a hardware/software co-development environment for the system

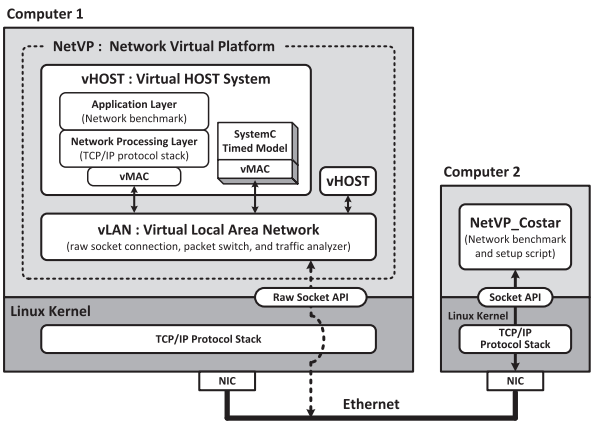


Fig. 2 System framework of the Network Virtual Platform (NetVP).

design of network-related hardware devices and employs an Application (APP) layer and a Network Processing (NP) layer to provide the function of network benchmark and TCP/IP protocol stack, respectively. Using a semi-hosting approach to design a vHOST system by ourselves instead of employing a complex virtual machine, such as QEMU [15] or Simics [16], can increase the simulation performance of the entire system. In addition, to build a Linux user mode TCP/IP protocol stack in the vHOST instead of using the one in the Linux kernel is more convenient since it allows us to modify the design or insert a new design into the system easily and efficiently.

In order to support on-line full system verification, we need a mechanism that allows a vHOST to communicate with another end-system, either a vHOST or a physical computer. Thus, we design a virtual LAN (vLAN) environment in the NetVP to switch packets between different vHOSTs as well as to connect its internal virtual network with an outside real-world network through the Raw Socket API of Linux kernel when needed. In addition, we develop a NetVP_Costar module, as shown in the right hand side of Fig. 2, which can establish network socket connections with a vHOST by using the standard TCP/IP protocol stack in the Linux kernel and build a golden network testbench for on-line real system verification. Furthermore, we design both the untimed model and timed model of the virtual MAC (vMAC), which can help users to connect their vHOST with the vLAN. In the rest of this section, the design issues of each module are introduced in detail.

2.1 Virtual Network Environment (vLAN and vMAC)

The virtual LAN (vLAN) module provides two operation modes: local mode and global mode. The local mode directly switches the packets between vHOSTs while the global mode allows a vHOST to communicate with another physical computer. During the phase of IPsec development and verification, the vHOST module which provides a hardware/software co-development environment may be recompiled and restarted frequently. This means that the connection between the vMAC and vLAN also needs to be

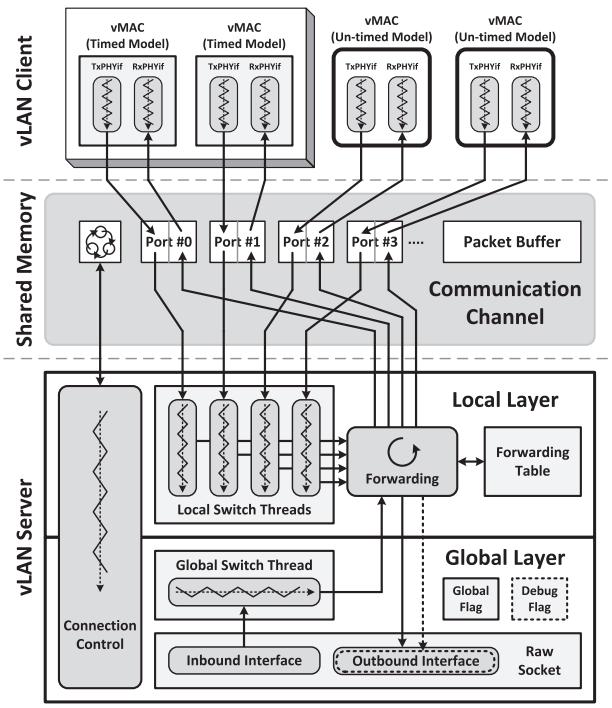


Fig. 3 Block diagram of the virtual LAN (vLAN) module.

re-established frequently. In order to keep the designer from restarting the vLAN manually, the vLAN uses the client-server architecture, as shown in Fig. 3, to support the hot-plug feature. The vLAN module applies the shared memory mechanism to create a communication channel that allows a vLAN client, such as a vMAC, to communicate with the vLAN server.

The vLAN server, as shown in the bottom half of Fig. 3, consists of a Local layer and a Global layer. The Local layer creates a virtual network environment to switch all packets between each enabled port that is connected with a vMAC; in the architecture view, it is similar to a network switch device. The Local layer builds *Local_Switch_Threads* to forward packets based on the content of the forwarding table which has recorded the MAC addresses associated with each enabled port. If the vLAN global mode is enabled, the Global layer creates a raw socket connection built in the host Linux operating system to exchange the raw packets with an outside real network. The broadcast packets and the packets which do not find the destination in the Local layer will be delivered to an outside real network through the outbound interface of the raw socket. In addition, the Global layer builds a *Global_Switch_Thread* to forward the packets received from the inbound interface of the raw socket.

Using the Raw Socket API of Linux kernel to connect the vLAN with a real network not only creates an on-line real system verification environment but also provides a debugging environment in which a user can employ a packet analyzer, such as Wireshark [17], to trace the packet traffic of the real network. In the vLAN local mode, however, a third-party packet analyzer is not able to capture the packets in our vLAN directly. Thus, we also use the raw socket

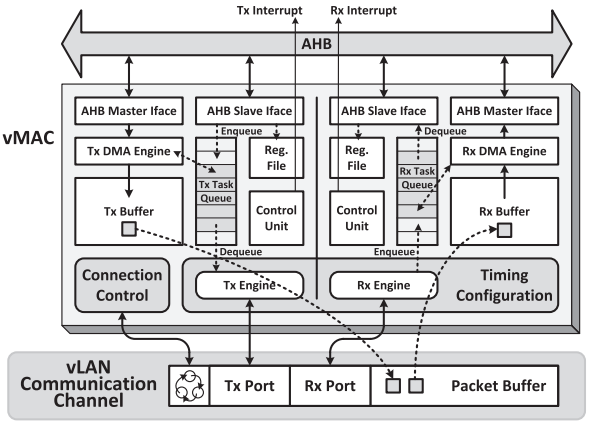


Fig. 4 Block diagram of the virtual MAC (vMAC) timed model.

of the Global layer to provide a debugging channel that is represented by the dotted lines in Fig. 3. This debugging channel only uses the outbound interface of the raw socket to deliver all of the packets in the vLAN to an outside real network. The operation of packet switching in the vLAN uses a copy-by-reference technique to remove unnecessary data copy. More specifically, it only exchanges the packet pointers among the ports that are associated with vMACs and the actual data copy of entire packet will occur when a packet arrives or leaves the vLAN.

In the vLAN server, the thread in charge of the connection control continuously monitors the state of each port in the communication channel, and it will build all the necessary threads whenever any one of the ports is connected with a vMAC. Due to security reason, the operation of the raw socket in the Linux system requires the privilege of the root user. Since vMAC and vLAN are designed to be client-server architecture that exchanges data using shared memory mechanism, it is still possible for a non-root user to employ the vMAC to communicate with the vLAN server which is turned on by the root user.

In order to support the top-down design concept of ESL, we not only build an untimed model of vMAC but also design a SystemC cycle accurate timed model of the vMAC including full simulation of queue, buffer, DMA, MMIO, bus interface, and interrupt. Like a real-world network interface card (NIC), the vMAC module can help users to connect their vHOST with the vLAN server through the vLAN communication channel. Figure 4 shows the architecture of the vMAC timed model. It is designed to be a full-duplex communication system; the receiving and transmitting subsystems have their dedicated bus interface, packet buffer, and processing units. The Ethernet PHY chip and the Media Independent Interface (MII), which connect a MAC-block to a PHY chip, are not modeled in our virtual platform in order to speed up simulation; they are directly replaced by the interface designed for the vLAN communication channel.

In the vMAC timed model, the *Connection_Control* thread employs a handshaking protocol to build the vLAN communication channel in which the *Tx_Engine* and *Rx_Engine* threads can exchange packets between vMAC and vLAN server. In the receiving subsystem, the *Rx_Engine* moves the receiving packets into the *Rx_Buffer* as well as generates the packet descriptors associated with these packets and puts them into the *Rx_Task_Queue*. The packets which are ready in the *Rx_Buffer* will be moved to outside global buffer by the *Rx_DMA_Engine* through the bus master interface. In addition, the vMAC provides memory-mapping I/O (MMIO) and interrupt interfaces to support both of the polling and interrupt communication mechanisms for its software driver.

The transmitting subsystem is similar to the receiving subsystem; the only difference is that the data moving is in the opposite direction. Moreover, the designer can also define the timing configuration of the vMAC timed model to support different bandwidth options, for example, 1 Gbps or 10 Gbps; the *Tx_Engine* and *Rx_Engine* threads in vMAC will use this timing configuration to generate the time delay for the SystemC simulation kernel.

2.2 Virtual Host Environment (vHOST)

For on-line full system verification, we design a virtual host environment (vHOST), as shown in the right hand side of Fig. 5. In order to allow the vHOST to communicate with other end-systems, the vHOST employs an Application (APP) layer and a Network Processing (NP) layer to provide the function of network benchmark and TCP/IP protocol stack, respectively. Moreover, we design an untimed and a timed (UT-T) interface that can connect the untimed vHOST with other SystemC timed models, as shown in the left hand side of Fig. 5, to form a hardware/software co-development environment where users can apply the ESL top-down design flow to develop their target system.

In the NP layer of vHOST, we design a simple TCP/IP protocol stack in Linux user mode, including TCP, UDP, IP, ARP, and ICMP protocols, and also provide an NP Socket API similar to the Socket API in Linux. The APP layer can build socket connections with another end-system, either a vHOST or a physical computer, through this NP Socket API. The TCP/IP protocol stack in the vHOST is designed with zero-copy architecture to remove unnecessary packet data copy. In addition, many vMAC untimed modules can be integrated in the NP layer to simulate the environment containing multiple network interface cards (NICs). Also, the IP aliasing function is provided to allow each NIC to be mapped onto at most 255 virtual interfaces.

In the APP layer, we build an *APP_Control_Thread* to manage user applications; it provides a user interface similar to a shell in Linux that can allow users to start, abort, and terminate their applications. In fact, all of the user applications in APP layer will be forked from the *APP_Control_Thread*. In order to establish the bidirectional testing connections between the vHOST and the NetVP_Costar for on-line real system verification, we design *Tx_Test_PktGenerator* and *Rx_Test_PktChecker* applications in the APP layer and their counterparts in the NetVP_Costar module. For reception

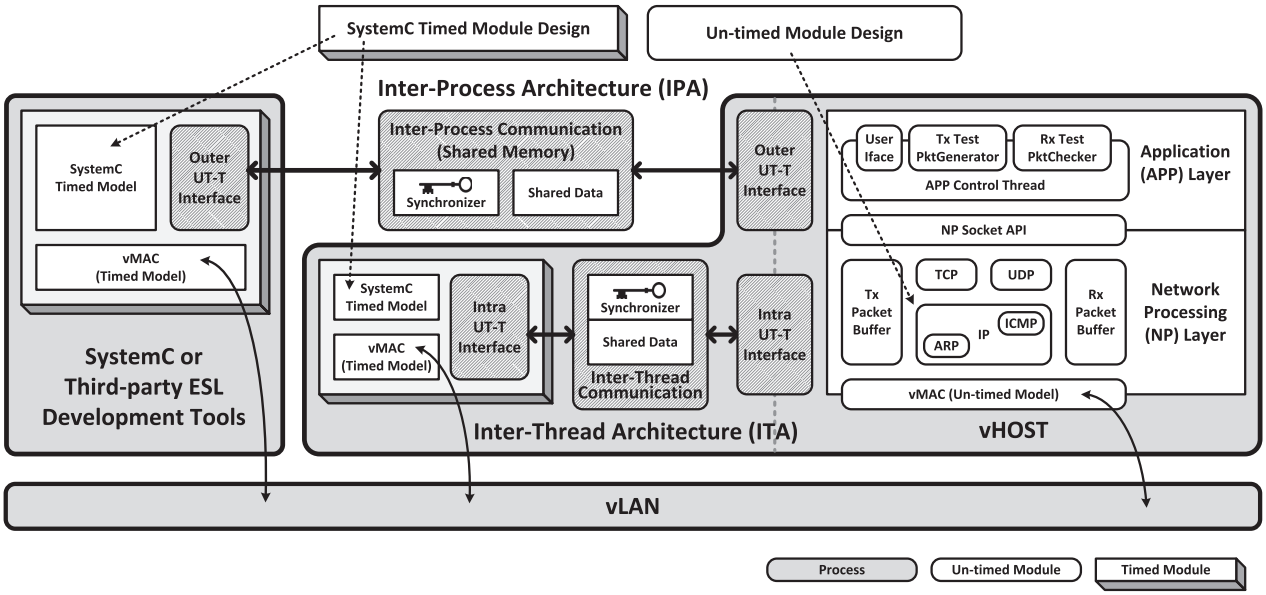


Fig. 5 Block diagram of the virtual host (vHOST) with the untimed/timed (UT-T) interface.

verification, the NetVP_Costar can generate the test packets which then are received and examined by the APP layer of vHOST.

However, considering the vHOST with timed models, the simulation speed of the whole NetVP is slowed down due to the low speed SystemC/HDL simulation. As a result, the gap between the speed of NetVP_Costar and the speed of vHOST will lead to an overflow of the raw socket buffer in vLAN module during on-line verification. To prevent this from happening, we design static and dynamic flow control mechanisms in our system. The static flow control mechanism allows users to define the throughput of the packet generating in the NetVP_Costar. In contrast, the dynamic flow control mechanism builds an extra side-band control channel between the NP layer of vHOST and the NetVP_Costar; the NP layer reports the number of received packets via this control channel, and the NetVP_Costar will dynamically adjust the speed of packet transmission accordingly. On the other hand, for transmission verification, the test packets are generated in the APP layer of vHOST, and are received and verified at the NetVP_Costar. After the initialization of the vHOST is completed, the APP layer waits for a READY packet sent from the NetVP_Costar before starting sending test packets, to ensure that the NetVP_Costar is well initialized and ready for packet validation.

Both of the APP layer and NP layer designed in C/C++ language are untimed models; it can allow users to insert a new untimed module into the system easily as well as to provide an on-line verification environment for this new untimed module. In order to support ESL top-down design flow, we design two kinds of architecture, Inter-Thread Architecture (ITA) and Inter-Process Architecture (IPA), to connect the untimed vHOST with other SystemC timed models to form a hardware/software co-development environment, as shown in Fig. 5. The gray block, rounded rectangle, and three-dimensional rectangle in Fig. 5 represent the process, untimed module, and timed module in our system, respectively.

In the Inter-Thread Architecture (ITA), the untimed model and timed model are integrated into a single process and simulated in different threads which can be synchronized with the Inter-Thread Communication (ITC) methods, such as the POSIX thread synchronization methods in Linux. Thus, we design a pair of Intra-UT-T (UnTimed-Timed) interfaces to build a synchronization channel between the untimed model and the timed model. A designer can develop his or her timed modules in the SystemC environment and employ the Intra-UT-T interface to communicate with other untimed threads. In contrast, the Inter-Process Architecture (IPA) simulates the untimed model and timed model in different processes and synchronizes them with the Inter-Process Communication (IPC) methods, such as the System V IPC in Linux. Thus, we also design a pair of Outer-UT-T interfaces and apply the shared memory mechanism to build a synchronization channel between the untimed model and the timed model.

The ITA is a simple and efficient architecture to integrate the untimed model and timed model together; however, if we want to employ a third-party ESL development tool, which have rich system modules, to build a SoC platform quickly and conveniently, the IPA is used to integrate our vHOST with other tools. In other words, our NetVP can be used in conjunction with any ESL development platform conveniently.

3. IPsec Processor Development

Having discussed the architecture of Network Virtual Platform (NetVP), we are now ready to move on the application of NetVP. System development that uses the NetVP is

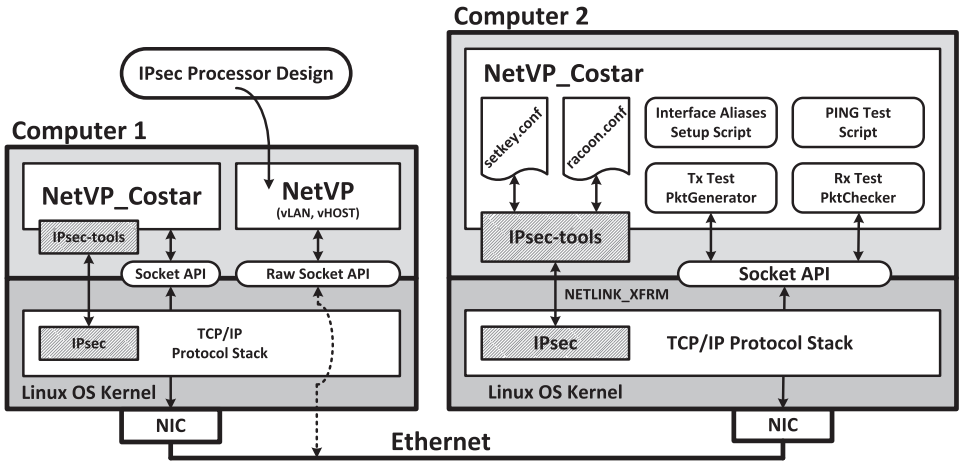


Fig. 6 System framework of the IPsec golden testbench.

efficient and flexible since it allows the designer to explore design spaces such as the network bandwidth and system architecture. In this section, we present how to develop a complex IPsec processor with the NetVP and in this paper, we will focus on the verification of IPsec processor design in the NetVP during different phases of the ESL top-down design flow.

3.1 IPsec Golden Testbench

First of all, we need a golden testbench that can allow us to verify the correctness of our IPsec design during later development process. Since IPsec support has been a part of the Linux kernel 2.6 itself, we can take the advantage of this and use the native software-based IPsec in Linux kernel as an IPsec golden model. This model is used in conjunction with the NetVP and NetVP_Costar modules to form a golden testbench for on-line verification, as shown in Fig. 6. Before developing an IPsec processor in the NetVP, we first employ a pair of the NetVP_Costar modules to verify that the native IPsec in Linux kernel is working well between the two computers which have installed the Linux operating system and connected with each other through the Ethernet network directly. For complex IPsec configurations, using IPsec-tools [18] can help us to conveniently setup the Security Policy Database (SPD) and Security Association Database (SAD) through the NETLINK_XFRM interface in Linux. It is possible to verify all of the IPsec configurations between the two computers by using the IP aliasing function in Linux to associate more than one IP address to a network interface. Table 2 shows the IPsec configuration of our network testbench in the NetVP_Costar. For simplicity, only the combinations of AES-CBC and HMAC-SHA1 algorithms have been shown in this table. Designers can study and trace the packet format of IPsec in this golden testbench with a packet analyzer, such as Wireshark [17].

Since we can utilize the Packet InterNet Groper (PING) command in Linux to generate and examine bidirectional communication and we can manually set the packet size, packet count, and source/destination IP address, the PING command is used to test our IPsec design at the beginning of verification. However, due to the limited packet sending rate of the PING command, it is not suitable for performance evaluation of the target IPsec processor. Thus, we develop a pair of packet generator and packet checker in both the NetVP_Costar and the APP layer of vHOST for bidirectional throughput measurement. The NetVP_Costar module provides all the requirements for the verification environment mentioned above, including the packet generator/checker as well as the script files for IP aliasing function, IPsec-tools, and PING test.

Table 2 IPsec configuration for network testbench.

IP Address	IPsec Protocol-Mode	Encr. Algorithm	Auth. Algorithm
192.168.0.10	Bypass	-	-
192.168.0.11	ESP - Transport	AES128-CBC	-
192.168.0.12	ESP - Transport	AES192-CBC	-
192.168.0.13	ESP - Transport	AES256-CBC	-
192.168.0.14	ESP - Transport	-	HMAC-SHA1
192.168.0.15	ESP - Transport	AES128-CBC	HMAC-SHA1
192.168.0.16	ESP - Transport	AES192-CBC	HMAC-SHA1
192.168.0.17	ESP - Transport	AES256-CBC	HMAC-SHA1
192.168.0.18	AH - Transport	-	HMAC-SHA1
192.168.0.19	ESP - Tunnel	AES128-CBC	-
192.168.0.20	ESP - Tunnel	AES192-CBC	-
192.168.0.21	ESP - Tunnel	AES256-CBC	-
192.168.0.22	ESP - Tunnel	-	HMAC-SHA1
192.168.0.23	ESP - Tunnel	AES128-CBC	HMAC-SHA1
192.168.0.24	ESP - Tunnel	AES192-CBC	HMAC-SHA1
192.168.0.25	ESP - Tunnel	AES256-CBC	HMAC-SHA1
192.168.0.26	AH - Tunnel	-	HMAC-SHA1

3.2 IPsec Untimed Functional Model

After the golden testbench has been built, an IPsec functional model can be realized and verified in the NetVP. Thus, we use C/C++ language to design an untimed functional model of the IPsec protocol suite, including inbound/outbound packet processing, SPD/SAD database querying, and cryptographic algorithm computations. Since

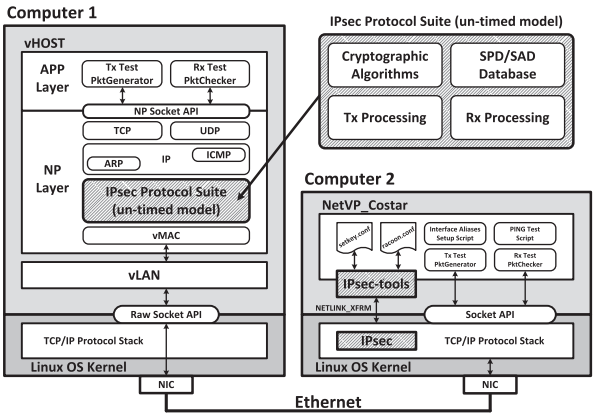


Fig. 7 On-line verification environment for IPsec un-timed functional model.

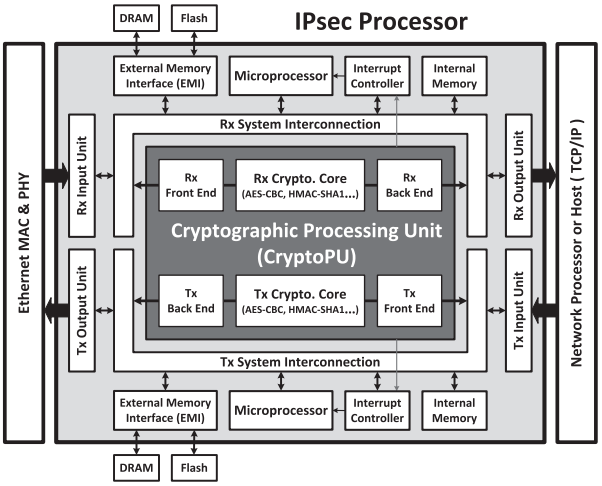


Fig. 8 System architecture of the IPsec processor with the cryptographic processing unit (CryptoPU).

the TCP/IP protocol stack of the NP layer in vHOST operates in the Linux user mode, this IPsec functional model can be easily integrated into the NP layer and connected to the vLAN with the untimed vMAC module, as shown in Fig. 7, without modifying and re-compiling the Linux kernel of the host machine.

During the IPsec development, a designer may modify and re-compile the design modules in the vHOST frequently. In other words, the communication channel between the vHOST and the vLAN must be rebuilt every time for on-line verification. Thanks to the client-server architecture between the vMAC and vLAN, a designer can just focus on the IPsec design while the vMAC can automatically build a communication channel with vLAN by itself. Figure 7 also shows the on-line verification environment for IPsec untimed functional model. Unlike traditional off-line verification, the NetVP provides on-line verification capability by setting up socket connection with the NetVP_Costar module installed on another physical computer. Bidirectional socket connection can be configured with different security policies and security associations in IPsec, so we can develop and verify the inbound and outbound packet processing of IPsec individually. A designer can say for sure that his or her IPsec design is correct as long as it can communicate with another computer with the native version of IPsec in a Linux kernel.

3.3 IPsec Timed Model

After a working IPsec untimed functional model is obtained, we evaluate the requirements of system performance and make decision on software/hardware partitioning. The encryption and authentication algorithms used in IPsec are computationally intensive yet regular. Therefore, implementing an ASIC-based cryptographic processing accelerator like our previous work [19], as shown in Fig. 8, is an effective scheme to improve the performance of an IPsec processor. The other part of IPsec, including protocol header parsing as well as the SPD/SAD database querying and management, is more appropriate to be realized in software.

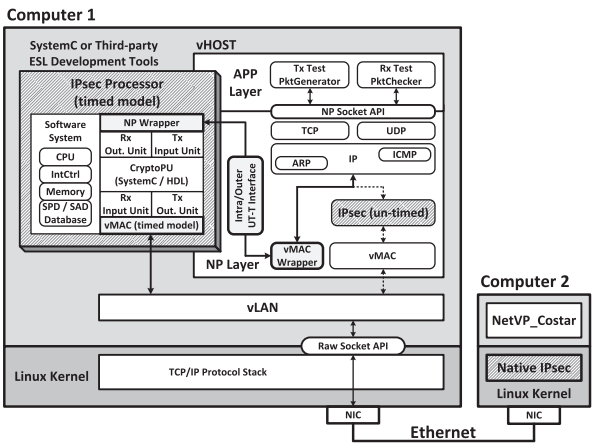
The NetVP aims at providing an SoC hardware/software co-development environment in which a designer can design his or her network-related device efficiently. Through the Intra/Outer UT-T interfaces, designers can move any cryptographic algorithms, hardware modules, as well as whole IPsec processor from the functional model to SystemC timed model. Following the ESL top-down design flow, these hardware modules can be refined from untimed model, through approximately timed model, to cycle-accurate timed model. The communication between these hardware modules can be designed with Transaction Level Modeling (TLM) to speed up the simulation time. According to different time accuracy requirements, the software part of IPsec processor can be performed in an Instruction-Set-Simulator (ISS) or cycle-accurate microprocessor model. Finally, a pin accurate model written in Synthesized SystemC or hardware description language (HDL) is obtained.

SystemC is a set of C++ libraries to provide an event-driven simulation kernel in C++ for hardware modeling. In SystemC, designers can use event-trigger mechanism and *wait()* function to arrange the timing of each hardware module as well as use the *sc_time* data structure and *sc_time_stamp()* method to manipulate the timing information at run-time. Thus, designers can apply this approach to evaluate the performance of each sub-module and the performance of the whole IPsec processor in the NetVP environment.

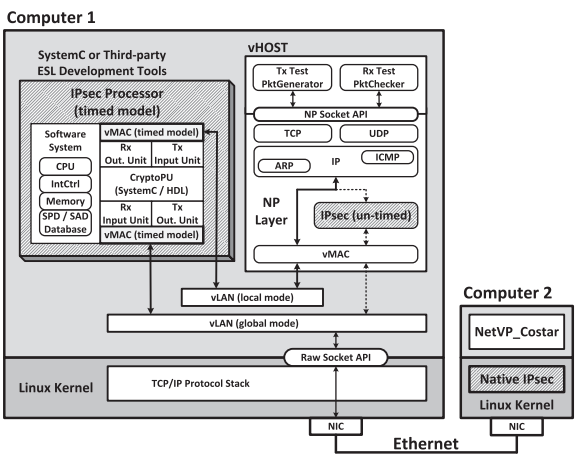
During the IPsec timed module development, we can still use the NetVP in conjunction with the NetVP_Costar to perform on-line verification. The NetVP can be configured to support different types of IPsec processor, such as bump-in-the-stack (BITS) or bump-in-the-wire (BITW). Figure 9 (a) shows the on-line verification environment for the BITS architecture of an IPsec processor. The IPsec processor is simulated in timed model and through the NP wrapper and vMAC wrapper to communicate with the NP

layer in the vHOST. On the other hand, the IPsec processor can employ a vMAC timed model to communicate with a real network through the vLAN module. With SystemC-HDL co-simulation, a designer can rewrite these timed modules in HDL and completely verify them with the NetVP on-line. Figure 9 (b) shows the on-line verification environment for the BITW architecture of an IPsec processor. In the case of BITW, two vLAN modules are deployed to switch packets; one is configured in local mode to deliver packets between the IPsec processor and the vHOST while the other one is configured in global mode to deliver packets between the IPsec processor and the real network environment.

We can use the NetVP to verify an IPsec processor design regardless of its type and timing accuracy as well as employ the packet analyzer to trace the packet traffic for debugging. Moreover, if we encounter any problems during the timed model development, we can also deliver packets to both the untimed model and timed model, and utilize the result of untimed model to debug the counterparts in the timed model.



(a) bump-in-the-stack (BITS)



(b) bump-in-the-wire (BITW)

Fig. 9 On-line verification environment for two types of IPsec processor. (a) bump-in-the-stack (BITS). (b) bump-in-the-wire (BITW).

4. Performance Evaluation

The Network Virtual Platform (NetVP) provides an ESL development environment for network-related hardware design; therefore, a designer can evaluate the performance of their target design by observing the simulated time model inside the virtual platform. In this section, we focus on the simulation performance of the NetVP itself in different IPsec development phases, as shown in Table 3. The simulation performance of the NetVP is a combination of many factors: the host system, the C++ compiler, the SystemC simulator, and the model being simulated. The simulation performance may not affect the correctness of the design verification in the NetVP, but it determines the scale and how many details of the target system and testbench can be modeled in the NetVP within acceptable time.

4.1 Experimental Environment

The on-line verification environments for different IPsec development phases, which have been discussed in Sect. 3, also can be used to estimate the simulation performance of the NetVP itself. The simulation host has an Intel Core i7 920 quad-core 2.6 GHz CPU with 4 GB main memory, and runs Linux of the kernel version 2.6. Two simulation hosts are connected through a 1 Gbps Ethernet environment to support on-line verification and evaluation. The packet generator in the NetVP_Costar or in the APP layer of the vHOST generates a number of UDP packets to estimate the simulation performance of the entire virtual platform.

In phase 2, 3, and 4, the security association of the IPsec is configured to use following features: the ESP protocol in the transport mode, the AES128-CBC encryption algorithm, and the HMAC-SHA1 authentication algorithm. In phase 3, the SystemC cycle-accurate model of the CryptoPU is integrated into the NetVP with the Inter-Thread Architecture (ITA). In phase 4, the IPsec processor modeled in Platform Architect [9] is integrated into the NetVP with the Inter-Process Architecture (IPA), and the CryptoPU

Table 3 IPsec top-down development phases in the NetVP.

Phases	Description
Phase 1	Only the NetVP is simulated without any IPsec module.
Phase 2	An IPsec untimed functional model has been integrated into the NetVP, as shown in Fig. 7.
Phase 3	After the software/hardware partitioning, a cryptographic processing unit (CryptoPU), as shown in Fig. 8, has been designed in SystemC cycle-accurate model. The other part of IPsec is the same as the one in phase 2, untimed functional model.
Phase 4	An IPsec processor with BITS architecture, as shown in Fig. 9 (a), has been designed as a complete SoC platform. The software part of IPsec processor has been ported into this SoC platform and executed by a microprocessor; the CryptoPU module has been rewritten in Verilog HDL. The whole IPsec processor has been modeled in the Platform Architect [9], an ESL development tool, with SystemC-HDL co-simulation and embedded in the NetVP.

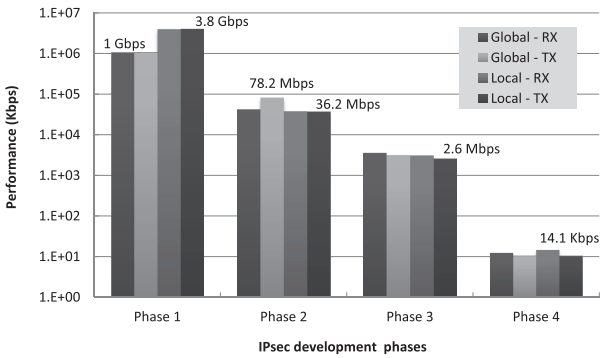


Fig. 10 Simulation performance of the NetVP in different IPsec development phases.

module is rewritten in Verilog HDL with about 207 K gates in 0.13 μm CMOS technology for SystemC-HDL co-simulation. The operation modes of the vLAN and amount of bidirectional communication are estimated during each development phase. More specifically, the vLAN global mode is used to test the communication between the NetVP and NetVP_Costar while the vLAN local mode is used to test the communication between two vHOST modules in the NetVP. In the case of vLAN local mode for phase 3 and 4, one vHOST module is modeled with the IPsec timed model while the other still remains in untimed model to speed up the simulation.

The simulation performance is measured in the equivalent bandwidth (bit-per-second), indicating the packet processing capability of the entire virtual platform. It is calculated as follows:

$$P_{BW} = \frac{\sum(B_i + B_e) \times 8}{\Delta T} \quad (1)$$

where ΔT is a period of time under steady state after entering simulation, B_i is the IP/IPsec packet size in bytes for the packets processed by the NetVP within the period, and B_e is the Ethernet overhead in bytes. The Ethernet overhead consists of the preamble (7 B), the SFD (1 B), the header (14 B), the trailer (4 B), and the interframe gap (12 B). In our simulation, the ΔT is set to 600 seconds, and the IP/IPsec packet size is set to the maximum packet size under the restriction of the 1500-byte Ethernet MTU.

4.2 Experimental Results

Figure 10 shows the simulation results in different IPsec development phases with respect to different operation modes of the vLAN. Obviously, the simulation performance decreases drastically as the model accuracy of the IPsec processor increases. In phase 1, the performance of the vLAN global mode is consistent with the bandwidth of real 1 Gbps network interface. On the other hand, the performance of the vLAN local mode which is not limited by the real network interface is up to 3.8 Gbps.

In phase 2, the IPsec untimed functional model has been integrated into the NetVP. The decryption of the AES-CBC algorithm for inbound packets needs more computation power than the encryption for outbound packets. Therefore, the performance of Global-TX is higher than Global-RX. In the case of the Local-TX and Local-RX, the simulation host has to execute both of the encryption and decryption of the AES-CBC algorithm for the two vHOST modules respectively; therefore, the performance of Local-TX and Local-RX is limited by the slower vHOST performing the decryption due to local loop-back. In phase 3 and 4, the cryptographic algorithms used in IPsec have been realized to a cycle-accurate unit, i.e. the CryptoPU, to improve the performance of the IPsec processor. In the CryptoPU, the processing of outbound packets needs more execution cycles than inbound packets; therefore, the simulation performance of TX is slightly lower than RX.

Before the HDL simulation in phase 4, the hardware design of IPsec processor can be designed in SystemC approximately or cycle-accurate timed models in phase 3 to get the better simulation performance during architecture exploration. The simulation performance of the entire virtual platform is limited by the SystemC and/or HDL simulation. For SystemC simulation, utilizing various parallel simulation techniques [20], [21] on symmetric multiprocessing (SMP) machines can further improve the SystemC simulation performance. For slow HDL simulation, the NetVP can be integrated with an FPGA board to create a hardware/software co-verification environment [22] that uses faster FPGA emulation to replace the HDL simulation.

5. Conclusion

We introduce a Network Virtual Platform (NetVP) for the development and verification of IPsec processor. In the NetVP, users not only can develop their products using the ESL top-down design flow, but also verify their design easily with the on-line real system verification mechanism. Moreover, unlike traditional FPGA verification process, the network bandwidth and architecture can be adjusted simply by using the timed models of vMAC and vLAN. This virtual platform is suitable for developing advanced ultra high speed network system when the developers have no means to possess or obtain the required high speed components which are yet not available in the market.

In addition, the NetVP can also be applied to the development of other kinds of network accelerators, such as TCP/IP Offload Engine (TOE), Internet Small Computer System Interface (iSCSI), Network Attached Storage (NAS), network router, network switch, etc. If full-system simulation is needed, the NetVP can be integrated with a virtual machine, as shown in our previous work [23]. Moreover, the NetVP can be used in conjunction with any third-party ESL development tool through the Outer-UT-T interface.

Acknowledgments

This work was supported in part by the National Science

Council, Taiwan, under Grant NSC 101-2220-E-006-002.

References

[1] S. Kent and K. Seo, "Security architecture for the internet protocol," IETF RFC4301, Dec. 2005.
[2] V. Manral, "Cryptographic algorithm implementation requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH)," IETF RFC4835, April 2007.
[3] M.-Y. Wang and C.-W. Wu, "A mesh-structured scalable IPsec processor," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.18, no.5, pp.725–731, May 2010.
[4] B. Bailey, G. Martin, and A. Piziali, ESL design and verification: A prescription for electronic system level methodology, Morgan Kaufmann/Elsevier, 2007.
[5] Open SystemC Initiative, "IEEE Std 1666-2011: SystemC language reference manual," IEEE Computer Society, Sept. 2011.
[6] Accellera Systems Initiative, http://www.accellera.org/
[7] L. Cai and D. Gajski, "Transaction level modeling: An overview," Proc. Int. Conf. on HW/SW Codesign and Syst. Synth., pp.19–24, Oct. 2003.
[8] A. Haverinen, M. Leclercq, N. Weyrich, and D. Wingard, "SystemC based SoC communication modeling for the OCP protocol," OCP-IP, Oct. 2002.
[9] Platform Architect, Synopsys Inc., http://www.synopsys.com/
[10] SoC Designer, Carbon Design Systems Inc., http://www.carbondesignsystems.com/
[11] T.R. Henderson, S. Roy, S. Floyd, and G.F. Riley, "ns-3 project goals," Proc. 2006 Workshop on ns-2: The IP Network Simulator, p.13, Oct. 2006.
[12] NetSim, Tetcos Inc., http://www.tetcos.com/
[13] A. Varga, "The OMNeT++ discrete event simulation system," Proc. European Simul. Multiconf. (ESM), pp.319–324, June 2001.
[14] X. Chang, "Network simulations with OPNET," Proc. Winter Simul. Conf., pp.307–314, Dec. 1999.
[15] QEMU, open source procesor emulator, http://www.qemu.org/
[16] Simics, Wind River Systems Inc, http://www.windriver.com/products/simics/
[17] Wireshark, open source packet analyzer, http://www.wireshark.org/
[18] IPsec-Tools home page, http://ipsec-tools.sourceforge.net/
[19] C.-C. Wang and C.-H. Chen, "An optimized cryptographic processing unit for IPsec processors," Int. Tech. Conf. on Circuits/Syst., Comput. and Commun., pp.899–902, June 2011.
[20] P. Ezudheen, P. Chandran, J. Chandra, B.P. Simon, and D. Ravi, "Parallelizing SystemC kernel for fast hardware simulation on SMP machines," Principles of Advan. and Distri. Simul., pp.80–87, June 2009.
[21] D. Yun, S. Kim, and S. Ha, "A parallel simulation technique for multicore embedded systems and its performance analysis," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.31, no.1, pp.121–131, Jan. 2012.
[22] Y. Nakamura, K. Hosokawa, I. Kuroda, K. Yoshikawa, and T. Yoshimura, "A fast hardware/software co-verification method for system-on-a-chip by using a C/C++ simulator and FPGA emulator with shared register communication," Proc. Design Automation Conf., pp.299–304, June 2004.
[23] C.-C. Wang, R.-P. Wong, J.-W. Lin, and C.-H. Chen, "System-level development and verification framework for high-performance system accelerator," IEEE Int. Symp. on VLSI Design, Automation and Test, April 2009.

Chen-Chieh Wang received a B.S. degree in electrical engineering with a minor in computer science from the Feng-Chia University, Taiwan, and an M.S. degree in computer and communication engineering from the National Cheng-Kung University, Taiwan, in 2003 and 2005, respectively. He is currently pursuing the Ph.D. degree in the Institute of Computer and Communication Engineering, National Cheng-Kung University, Taiwan. His research interests include advanced computer architecture, network security, SoC integration, and ESL design.

Chung-Ho Chen received the M.S.E.E. degree in electrical engineering from the University of Missouri-Rolla, Rolla, in 1989 and the Ph.D. degree in electrical engineering from the University of Washington, Seattle, in 1993. In 1993, he was with the Department of Electronic Engineering, National Yunlin University of Science and Technology. In 1999, he joined the Department of Electrical Engineering, National Cheng-Kung University, Tainan City, Taiwan, where he is currently a Professor. His research areas include advanced computer architecture, graphics processing, and high-speed ESL simulation systems. Prof. Chen was a recipient of the 2009 Outstanding Teaching Award from the National Cheng-Kung University. He was the Technical Program Chair of the 2002 VLSI Design/CAD Symposium held in Taiwan. He was the IEEE circuits and systems society Tainan chapter chair for the years of 2011–2012.