NetVP: A System-Level NETwork Virtual Platform for Network Accelerator Development

Chen-Chieh Wang, Sheng-Hsin Lo, Yao-Ning Liu, and Chung-Ho Chen
Institute of Computer and Communication Engineering
National Cheng Kung University, Tainan, Taiwan

***Abstract*—In this paper, we propose a Network Virtual Platform (NetVP) to develop and verify network accelerator like an IPsec processor. The NetVP provides on-line verification mechanism and is suitable for ESL top-down design flow, supporting developments of un-timed as well as timed models. System development using this NetVP is efficient and flexible since it allows the designer to explore design spaces such as the network bandwidth and system architecture easily.**

I. INTRODUCTION

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. In ESL design methodology, designers improve their design with top-down approach [1], as shown in TABLE I. A designer can implement the architecture abstract in Layer-3 first with a high-level language and later refine the design layer-by-layer. Finally, a pin accurate model written in hardware description language (HDL) is obtained.

Internet Protocol Security (IPsec) [2] is a protocol suite contains Authentication Header (AH) and Encapsulating Security Payload (ESP) protocols to provide confidentiality, authentication, and an optional anti-replay service for IP packet transmission. IPsec can be implemented in many locations and can be operated in transport or tunnel modes, working with different encryption and authentication algorithms [3]. Developing and verifying an IPsec processor, as shown in Fig. 1, is of great challenge.

Conventionally, to develop and verify a network-related hardware like an IPsec processor, one can verify it with off-line approach. Such approach is adopted in [4], correct input and output are logged in advance for comparison during HDL simulation. After the HDL model is built, it can be synthesized and loaded to an FPGA board for prototype verification. However, in this approach, the correctness, convenience and the cost of development are not favorable. Thus, we propose a Network Virtual Platform (NetVP) to provide a platform suitable for ESL top-down design flow and capable of on-line verification.

TABLE I. ESL COMMUNICATION ABSTRACTION LAYERS [1]

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

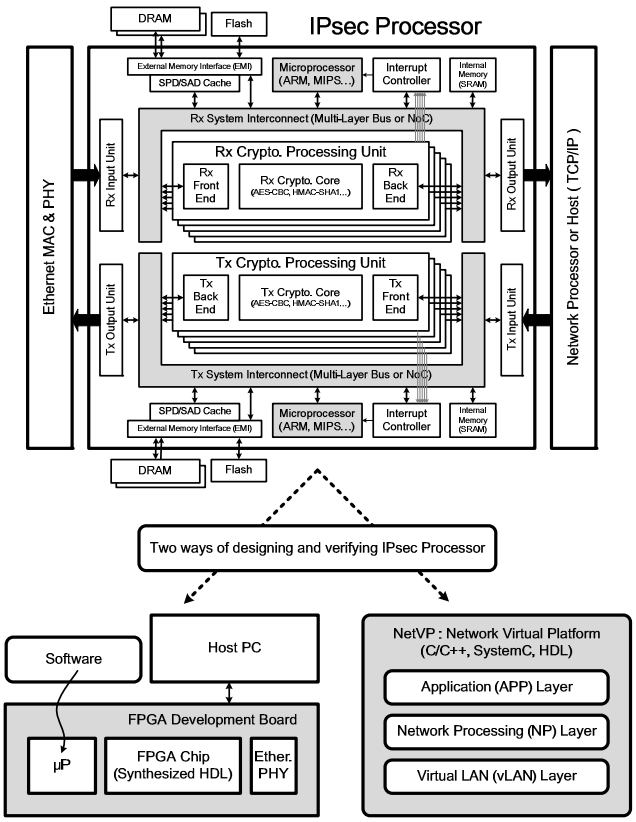

Figure 1. Design and verification methods of IPsec Processors

This paper is organized as follows. Section II describes the system architecture of the NetVP. Section III discusses the design and implementation issues. Finally, Section IV concludes this paper.

II. SYSTEM ARCHITECTURE

Fig. 2 shows the system architecture of the Network Virtual Platform (NetVP) that consists of three layers, including application (APP) layer, network processing (NP) layer, and virtual LAN (vLAN) layer. In the rest of this section, the function of each layer is introduced.

A. Connecting the NetVP with other end-systems

For simplicity in the verification of IPsec design, we need a mechanism which allows the NetVP to communicate with other end-systems, either another virtual platform or another physical computer. Therefore, we design a virtual LAN (vLAN) environment in the NetVP. Packets between virtual platforms are directly switched by the vLAN while the communication with another physical computer utilizes the raw socket connection built in the host Linux OS to transmit/receive the packets to/from the real network. Furthermore, we implement both un-timed model and timed model of the virtual MACs (vMAC) which help user applications to communicate with the vLAN. It will be introduced in detail later.

The network processing (NP) layer provides simple TCP/IP protocol stack in Linux user mode, and provides an *NP Socket API* for the application (APP) layer. The APP layer can build socket connection with the Linux OS of another physical computer through this *NP Socket API*.

B. Generating the test packets for IPsec verification

Unlike traditional off-line verification, the NetVP provides on-line verification mechanism by setting up socket connection with another physical computer installed with the Linux OS and generating the test packets. 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.

As shown in Fig. 2, we install the NetVP on Computer 1, and implement *Tx Test Packet Generator* and *Rx Test Packet Checker* applications on the APP layer. Computer 2 is set to be a NetVP_Remote with packet generator/checker installed as the counterparts of those in Computer 1. Connection is built between these two computers for test packets transportation and correctness verification.

We install the IPsec-Tools [5] on Computer 2, thus the NetVP_Remote can set the IPsec in the native version to build the golden testbench. If an IPsec processor implemented with the NetVP passes the on-line verification, it is clear that the functionality of this IPsec processor is identical to the native one in the Linux kernel.

As shown in the upper half of Fig. 2, in computer 2, we use alias command in Linux to build numerous virtual interfaces and assign corresponding IP addresses. We set IPsec configurations, like *Security Policy Database* (SPD) and *Security Association Database* (SAD), for each IP address, thus we are able to examine different IPsec configurations all together.

Since we can use Packet InterNet Groper (PING) command in Linux to generate and examine bidirectional

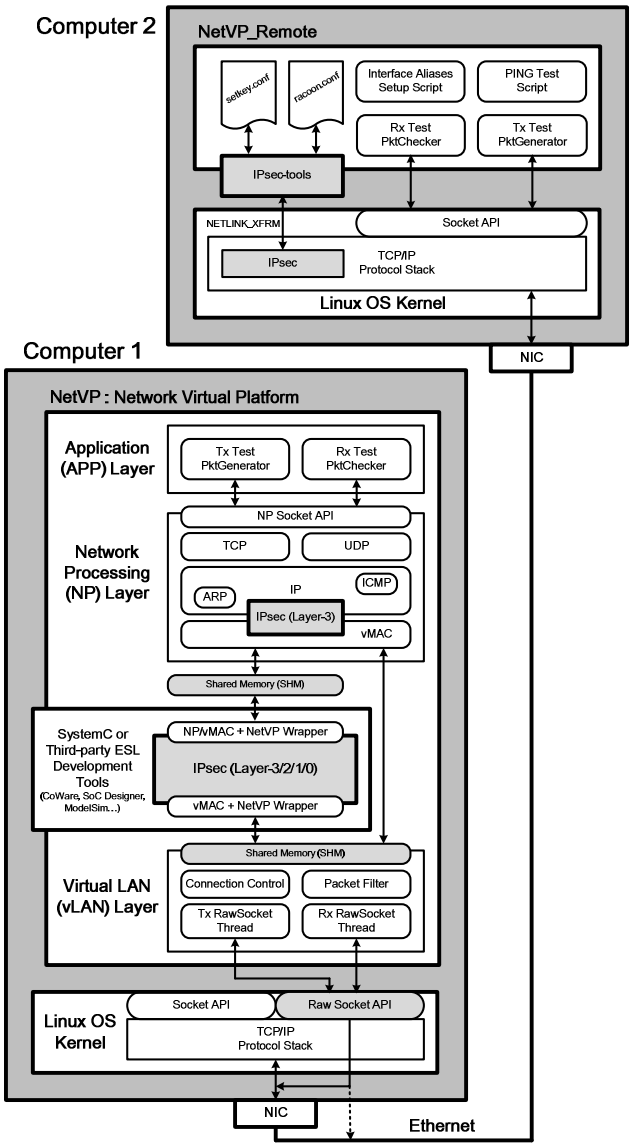

Figure 2. System architecture of the NetVP

communication and we are allowed to set packet size, packet count and source/destination IP address by ourselves, PING command is used to test an IPsec processor at the beginning of verification. However, for the sake of security, the packet sending rate of PING command is limited, thus it is not suitable for performance evaluation of IPsec processor. We build a pair of packet generator and packet checker in both APP layer and NetVP_Remote for throughput measurement.

C. Developing IPsec processor in the NetVP

According to the location where the IPsec is implemented, IPsec can be classified into three types: (1) *built-into-IP-stack*: It's a software solution and requires access to the IP source code. (2) *bump-in-the-stack* (BITS): IPsec can be implemented between the native IP and the network interface card (NIC). (3) *bump-in-the-wire* (BITW): IPsec can be implemented somewhere out of the NIC. That is, it may be built in a gateway and often is IP addressable. The NetVP provides

complete environment necessary to develop these three types of IPsec, and support ESL top-down design flow.

A designer can implement his or her un-timed (layer-3) model of IPsec on the network processing (NP) layer. This model is then connected to vLAN with untimed vMAC, and its correctness is examined with the on-line verification support provided by the NetVP.

In addition, a designer can also implement timed model of the IPsec with SystemC. With the *NetVP Wrapper*, the NetVP is able to be integrated with any ESL design tool supporting SystemC. If BITS version is to be built, the *NetVP Wrapper* in the timed model system directly communicates with the MAC driver in the NP layer. In the case of BITW, the wrapper can be turned into the vMAC peer-to-peer mode in which the wrapper is connected directly with the vMAC in the NP layer. A timed model of IPsec processor can communicate with the vLAN using a timed vMAC model. With SystemC-HDL co-simulation, a designer can rewrite these timed models with HDL and completely verify them with the NetVP on-line.

We can use the NetVP to verify an IPsec processor regardless of its type and timing accuracy. Besides, a packet monitor such as Wireshark [6] can be used for debugging.

III. Design and Implementation Issues

In this section, we introduce the design and implementation issues of each layer in the NetVP in detail.

A. vLAN layer and vMAC

The whole NetVP communicates with the real world via the vLAN layer. The bottom half of Fig. 3 shows the architecture of the vLAN layer. During IPsec development and verification, connection with the vLAN may be frequently rebuilt. In order to keep the designer from restarting the NetVP manually, the vLAN layer is designed using client-server architecture.

The vLAN server build threads of *TxRawSocket* for each enabled port, but all the ports share one common *RxRawSocket* thread. The thread in charge of connection control in the vLAN server will clean up all of the states inside the vLAN server whenever the vLAN is reconnected. The reset of raw socket is necessary to avoid the correctness of verification from being influenced by the packets remained in the buffer of raw socket.

To support the top-down design concept of ESL, we realized both timed model and un-timed model of vMAC. As shown in the upper half of Fig. 3, we use SystemC language to implement the cycle accurate timed model including full simulation of queue, buffer, DMA, MMIO, bus interface and interrupt. In addition, the *TxPHYif* and *RxPHYif* communicate with the vLAN server using shared memory mechanism. In a timed model, the designer is allowed to define the timing configuration of the vMAC, for example, 1Gbps or 10Gbps.

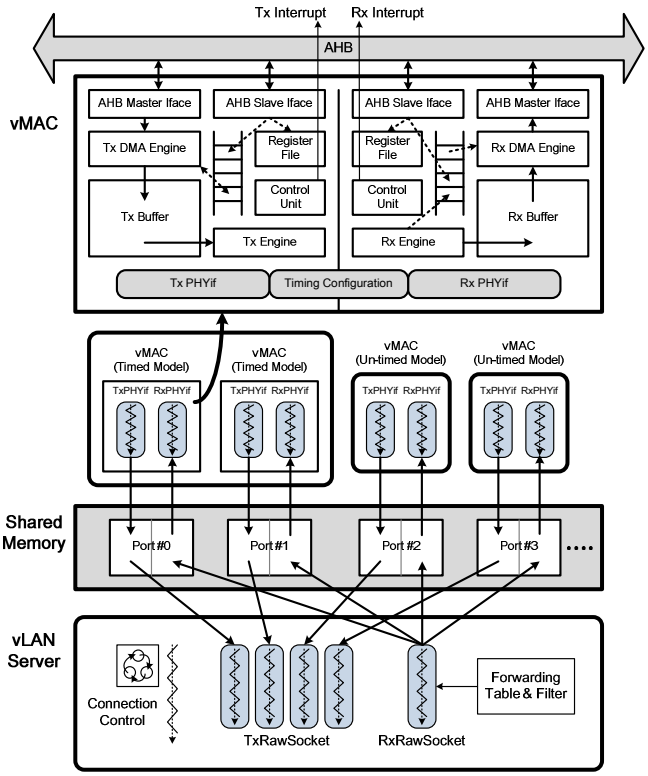

Figure 3. The architecture of the vLAN and vMAC

The operation of raw socket in Linux requires the privilege of the root user. Since vMAC and vLAN server exchange data using shared memory, it is still possible for a non-root user to use the vMAC to communicate with the vLAN server which is turned on by the root user. However, in order to allow a non-root user to control the forwarding table and filter inside of the vLAN server, we break the original vLAN server into *vLAN-Lite* and *vLAN-Bridge*. The vLAN-Lite is turned on by Root while the vLAN-Bridge is controlled by the user.

B. NP Layer

In the NP layer of the NetVP, we implement a simple TCP/IP protocol stack in Linux user mode, including TCP, UDP, IP, ARP and ICMP protocols, and provide an *NP Socket API* similar to the Socket API in Linux. The reason we build Linux user mode TCP/IP protocol stack in the NetVP instead of using the one in Linux kernel is that building a TCP/IP protocol of our own is really convenient for us since it allows us to modify the design or insert a new design into the system easily and efficiently. Moreover, the NP layer may be further improved to be a TCP/IP offload engine (TOE) in the future.

Many un-timed models of vMAC are integrated in the NP layer to simulate the environment containing multiple network interface cards (NIC). Also, the alias function is provided to allow each Ethernet interface to be mapped onto at most 255 virtual interfaces.

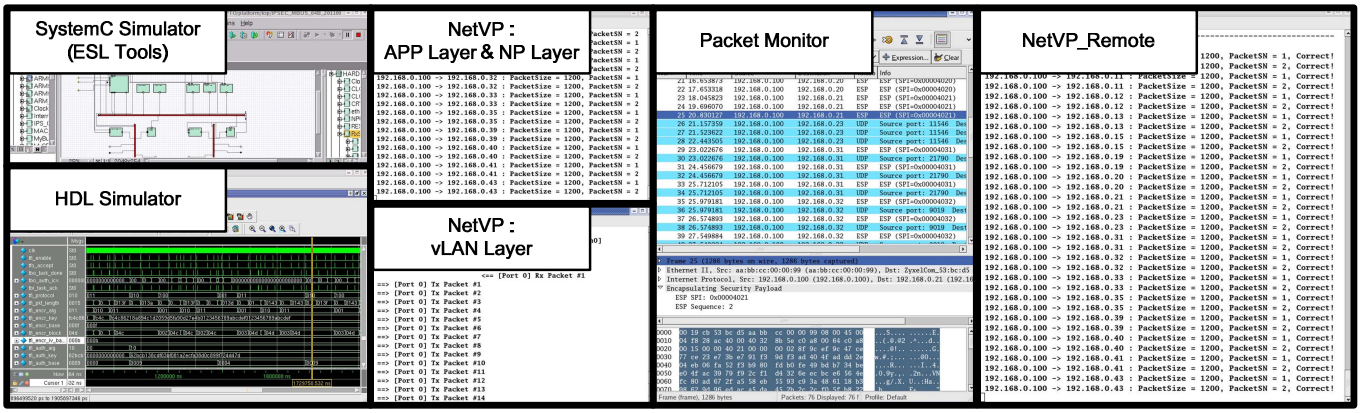

Figure 4. The snapshot of the NetVP development and verification platform

The NP layer is built in client-server architecture as well. In this way, no manual restart is needed. During the development of IPsec of BITS type, a designer can use C/C++ language to implement the un-timed model in the NP layer, and then refine it using SystemC model into a timed one. We can regard the vMAC in the NP layer as the server of the *NetVP Wrapper* which is used to connect the client of the *NetVP Wrapper* in the timed model.

C. *APP layer and NetVP_Remote*

The NetVP uses the APP layer and the NP layer to set up bidirectional testing connection with the NetVP_Remote. To evaluate the reception performance of IPsec, we can generate test packets at the NetVP_Remote side, and receive and examine the correctness of the test packets on the APP layer in the NetVP. However, considering the NetVP with timed models, the simulation speed of the whole NetVP is slowed down due to the low speed SystemC/HDL simulation. Thus there is a gap between the speed of NetVP and the one of NetVP_Remote. Such gap will lead to an overflow of the buffer in the raw socket. To prevent it from happening, an extra side-band control channel is built between the NP Layer of NetVP and the NetVP_Remote to provide a mechanism of flow control. More specifically, the NP layer of NetVP reports the number of received packets via this control channel; the NetVP_Remote will dynamically adjust the speed of packet transmission accordingly.

On the other hand, for IPsec transmission, the test packets are generated in the APP layer of NetVP, and are received and verified at the NetVP_Remote. After the initialization of the NetVP is completed, the APP layer waits for a READY packet sent from the NetVP_Remote before starting sending test packets, to ensure that the NetVP_Remote is well initialized and ready for verification of packets.

IV. CONCLUSIONS

We introduce a Network Virtual Platform (NetVP) for the development and verification of IPsec. In addition, it can also be applied to the development of other kinds of network accelerator, such as TCP/IP Offload Engine (TOE), Internet Small Computer System Interface (iSCSI), Network Attached Storage (NAS), network switch, etc. If full-system simulation is needed, the NetVP can be integrated with a virtual machine, as shown in our previous work [7]. Fig 4. shows the snapshot of the whole development and verification platform, including the NetVP, NetVP_Remote, SystemC simulator, HDL simulator and packet monitor.

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 verification mechanism. Moreover, unlike traditional FPGA verification process, the network bandwidth and architecture can be adjusted easily in timed vMAC and vLAN. This virtual platform is suitable for developing advancing ultra high speed network system even though the developers have no means to possess or obtain the required high speed components which are even not available in the market.

ACKNOWLEDGMENT

This work was supported in part by the National Science Council, Taiwan, under Grant NSC 100-2220-E-006-002.

REFERENCES

[1] A. Haverinen, M. Leclercq, N. Weyrich, and D. Wingard, "SystemC based SoC communication modeling for the OCP protocol," OCP-IP, October 14, 2002.

[2] S. Kent and K. Seo, "Security architecture for the Internet Protocol," IETF RFC4301, December 2005.

[3] V. Manral, "Cryptographic algorithm implementation requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH)," IETF RFC4835, April 2007.

[4] 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.

[5] "IPsec-Tools home page," http://ipsec-tools.sourceforge.net/, Available online.

[6] "Wireshark home page," http://www.wireshark.org/, Available online.

[7] 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.