A Multi-Processor NoC Platform Applied on the 802.11i TKIP Cryptosystem

Jung-Ho Lee, Sung-Rok Yoon, Kwang-Eui Pyun, and Sin-Chong Park
Information and Communications University
119, Munjiro, Yuseong-gu, Daejeon, 305-732, Korea

Abstract— **Since 2001, there have been a myriad of papers on systematic analysis of Multi-Processor System on Chip (MPSoC) and Network on Chip (NoC). Nevertheless, we only have a few of their practical application. Till now, main interest of researchers has been to adapt NoC to the communication intensive multimedia system like H.263. However, this paper attempts to expand the domain of NoC platform to one of the wireless security algorithms (TKIP), because its inter-component transaction pattern shows considerable characteristic for NoC. This paper consists of the explanation on operational sequence of the algorithm in chosen architecture and the brief illustration of important composing NoC blocks (Network Interface, Router).**

I. Introduction

TKIP is the one of three algorithms, defined in 802.11i enhanced security standard [1]. We have chosen TKIP cryptosystem, because it is the most viable solution for MPSoC system, in that it can satisfy both 802.11n minimum throughput requirement (76 Mbps) and reasonable robustness against known attacks. WEP is the one we should choose, if we consider performance only. However, WEP has already exposed too wide crevice against malicious intrusion, such as fragmentation attack and weak IV (Initialization Vector) attack [2].

In 0.13μ CMOS process, typical ARM7 core occupies $0.24mm^2$ area [3]. Considering this small area, integrating 10 ARM cores on a chip seems to be reasonable. In the future at hand, that sort of on-chip core grid processor will come into the market as a facility for general applications. Motivated by this expectation, in this paper, we analyze TKIP security algorithm on a general purpose MPSoC platform.

The paper is organized as follows: Section II discusses about the way how to divide a workload to many processors to meet the throughput requirement. Section III explains the applied NoC architecture. Experimental results are shown in Section IV. Finally, Section V concludes this paper.

II. Concurrent Operation of The TKIP components

As shown in Fig. 1, TKIP cryptosystem is composed of three major components: RC4, Michael, and CRC-32. RC4 is the crux of the cryptographic algorithm, which grants confidentiality to the payload. RC4 accompanies a key mixing algorithm that protects the system from replay attack. And the second one - Michael - is to prevent the system from forgery attack by hijacker. CRC-32 is for generating ICV (Integrity Check Value), which checks sanity of the transmitted payload.

TABLE I
Execution cycles for processing 2304 Bytes of MSDU on a single 100Mhz ARM processor

	# of Execution Cycles
RC4	75963(RC4)+5203(KeyGen)
Michael	63040
CRC-32	32969

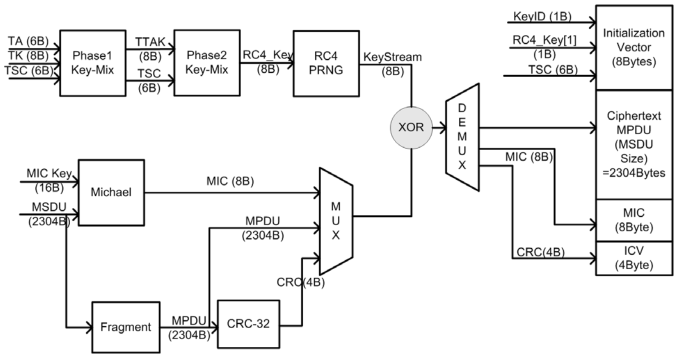

Fig. 1. TKIP cryptosystem overview

To determine the degree of concurrency, we first calculate maximum allowable number of cycles for processing 2304 Bytes of MSDU (MAC Service Data Unit).

$$Max.cy. = \frac{2304Bytes \times 8bits/Byte}{10 \times 10^{-9}sec \times 76 \times 10^{6}bps} = 24253\,cy. \quad (1)$$

Table I is showing the single CPU execution result of each component. Based on the observation, the equation (1) tells that the system ought to complete all operations on a MSDU packet (2304 Byte) every 24253 cycles ($243\mu s$ in 100Mhz CPU). RC4, Michael, and CRC-32 should be divided into 4, 3, and 2, respectively, to meet the throughput goal.

A. Chronological Relationship Among The Components

We can expect strict parallelism if and only if a problem is divided into several pieces of independent part. Unfortunately, vast amount of cryptographic algorithm are fully sequential, and cannot be broken into independent parts. To make it worse, Michael and RC4 have a loop, in which result of previous loop is fed as an input to the consecutive loop, forming dependent chain. Therefore, we have no other choice than to juxtapose

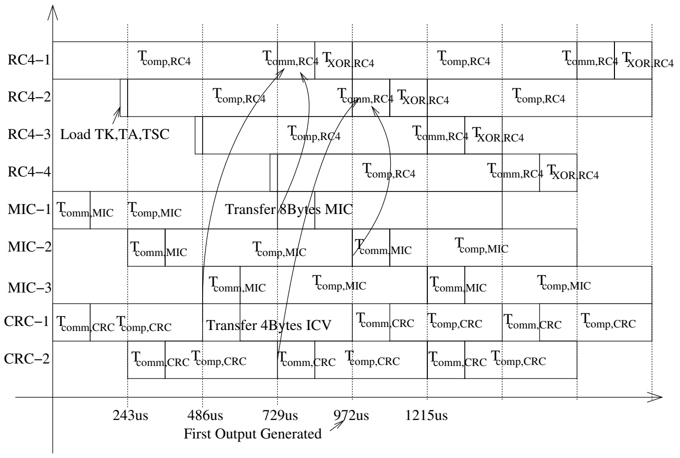

Fig. 2. Synchronization timing of TKIP components

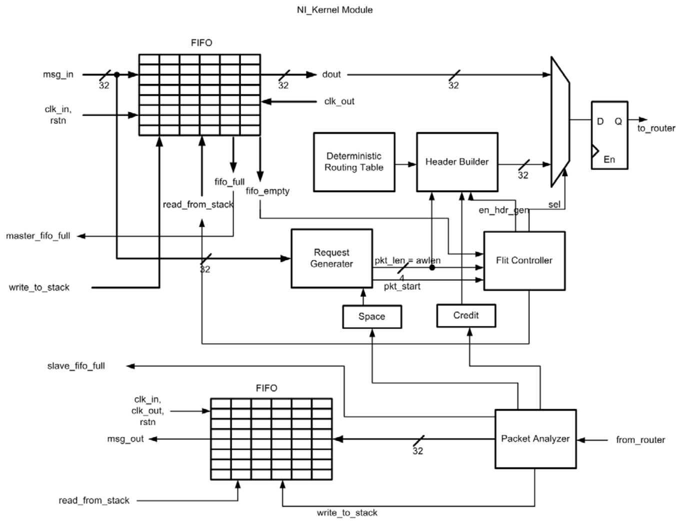

Fig. 3. NI kernel block diagram

each component in a pipelining style. Fig. 2 illustrates timing sequences how 9 CPUs are pipelined to achieve 76Mbps throughput. Fig. 2 shows that new MSDU is fed into an idle block every $243\mu s$, which is the maximum allowable time to meet the throughput requirement (76Mbps). Here, number of CPUs, allocated for each operations (RC4, MIC, and CRC), is determined so that at least one resource of each component can be released every $243\mu s$. Due to pipelining delay, we should wait until $972\mu s$ for this system to yield the first output. Starting from $972\mu s$, en/decapsulated ouput packet, including MIC and ICV, is generated every $243\mu s$. In the pipelining, we can observe that the sequence of the set of cooperating components for processing a MSDU, are repeated every 12 intervals. For instance, RC4-1, MIC-1, and CRC-1 blocks work together to process the first MSDU, which arrives at the first interval ($0 \sim 243\mu s$). Then, RC4-2, MIC-2, and CRC-2 blocks collaborate to encapsulate the MSDU, arriving at the second time interval ($244 \sim 486\mu s$). And after 12th time interval ($2916 \sim 3159\mu s$), RC4-1, MIC-1, and CRC-1 blocks are selected for processing an incoming MSDU. This means that the number of the cooperating set is finite, and each block has predetermined working partner.

B. Concurrent Calculation of RC4 PRNG

We have 4 RC4 blocks. The role of RC4 block is to calculate RC4 PRNs, based on the key generated from TK (Temporal Key), TA (Transmitter Address), TSC (TKIP Sequence Counter). Note that the block separates two steps: generating PRNs and XORing with data (MSDU, MIC, and ICV). There are two reasons behind this scheme. Firstly, we can save CPU cycle by fully utilizing burst memory access. It is easy to think of reading $16 \times 4Bytes$ (bus bandwidth = 32bits = 4Bytes) at a time gives us benefit, than reading$1Byte$ in every loop. Secondly, final XOR operation can be performed in SIMD (Single Instruction Multiple Data) manner ($1Byte\, XOR \times 4$), because payload byte stream has no dependence, and the RPNs are readily computed. When we use 32bit XOR operation we can make the number of loops $\frac{1}{4}$of original one.

C. Concurrent Calculation of Michael MIC

As mentioned above, sequential partitioning of Michael is algorithmically infeasible. Therefore, we have to juxtaposes 3 MIC block units and feed them the workloads every $243\mu s$. In realization, Michael block reads 16bits (2Bytes) MIC and 2304 Bytes MSDU key from shared memory, and produces write back into the final result into the shared memory, so that RC4 can read them and use them.

D. Concurrent Calculation of CRC-32 Block

CRC (Cyclic Redundancy Code) itself is parallelizable, because it is based on division in $GF(2^{32})$. Nevertheless, we don’t need to take the benefit of this art, because input packet stream of other cooperating components doesnŠt split up into several parts. For this reason, we just allocate two processor cores for independent CRC operation, and feed each of them with a MSDU payload.

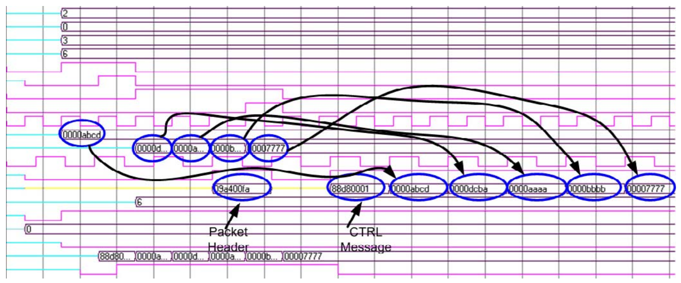

Fig. 4. Packetization delay of NI, when CPU clock is 100MHz and NoC clock is 75MHz.

III. Interconnection Architecture of the Proposed System

NoC is proposed as a design space solution for increasing cost of synchronization between cores, originated multi-GHz core clock against worsening RC delay and capacitive coupling between wires [4].

Our system is interwoven with the NoC fabrics, forming 3x3 torus topology shown in Fig. 7. In this topology, any pair of two components is separated by more than two hops from each other.

A. Router and Routing

Our system adopts deterministic routing model, compensating for the two limitations: stringent latency requirement, and traffic predictability [5]. TKIP system doesn't have possibility of waiting for reordering or retransmission of packet stream, and the traffic pattern is regular and fully predictable. Beside performance and simplicity benefit, deterministic routing has livelock-free property, while it still suffers from deadlock problem. In our design, deadlock problem is resolved by using the virtual channel, which shares a physical channel by several logically separated channel [6]. As shown in Fig. 7, our router has two destination differentiated buffers sharing a pair of up/down transmission line.

As for forwarding strategy, we mobilize wormhole method. As its name implies, if header filt opens the circuit, the subsequent body flit (of the worm) follows the path without any intervention of complex control algorithm. Thus, although the nature of wormhole routing is packet switching, it is almost as efficient as the circuit switching. With wormhole routing, the packet reaches to its destination node in $T = \left((n-1) + \frac{L}{W}\right) T_c$,where $T_c$is the channel cycle time, L is the packet length, n is the number of links and W is the channel width. For instance, in Fig. 7, It takes $\left((3-1) + \frac{2304\times 8}{32}\right) \times 10ns = 11.52\mu s$ for 2304 Bytes MSDU leaving from NI (Network Interface) shared memory node and arrives at the NI of CPU2, when the cycle time is 10ns channel width is 32bits.

B. NI (Network Interface)

Network Interface decouples computation and communication workload by translating the language CPU can understand (AMBA/AXI signals), into the language of router (header flit followed by packet body flits), and vice versa. Among several researchs on NI, Radulescu et al.'s work [7] is robust and practical enough to connect commercial cores with AMBA/AXI interface. Basically, our NI design depicted in Fig. 3 followed their design criteria. The NI provides extensive set of services including multicast, narrowcast, with QoS (Quality of Service) guaranteed. But their NI is too heavy at current position, because they have complex QoS scheduler and corresponding queues. But usual multiprocessor application, including TKIP, doesn't need service differentiation, because all traffics have the same significance. Thus we decide to exclude QoS scheduling logic and reduce the packetization delay of NI, instead of providing delicate QoS control.

Our NI kernel is depicted in Fig. 3. It operates as follows: Message is forwarded from FIFO to request generator, on its arrival. Then the header builder generates routing path, by referring to the deterministic routing table, and forms header flit. Then the header is forwarded into the NoC domain. Fig. 4 shows a waveform of Verilog simulation of our NI. Simulation parameters are as follows-ARM core clock: 100MHz, NoC clock: 75MHz, Burst length: 4, flit width: 32bits, BUS width: 32bits, Burst mode: INCREMENT. This waveform says that the system delays of packet header and body flit generation processes are 40ns and 70ns, respectively. Once the header is fed into the NoC router, the routing path is open, thus subsequent flits follow the path, without any delay. Our NI also supports multicast operation, in which data packet is flooded into multiple paths simultaneously. This multicast functionality presents the greatest advantage over conventional BUS architecture to the architecture; BUS cannot avoid bottleneck because only one master (CPU) can access the data source.

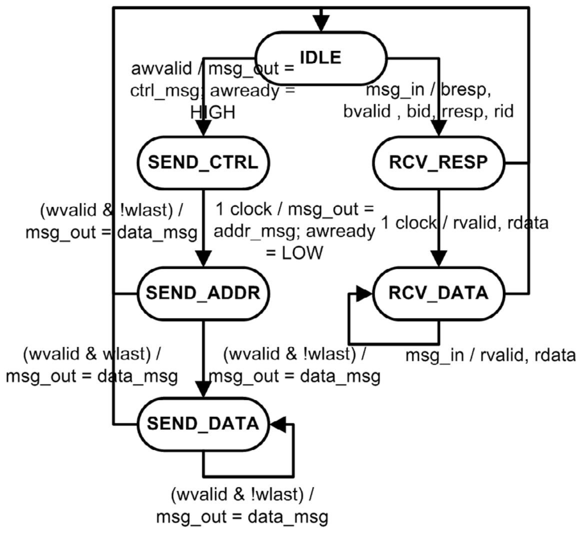
IDLE
awvalid / msg_out = ctrl_msg; awready = HIGH
msg_in / bresp, bvalid , bid, rresp, rid
SEND_CTRL
RCV_RESP
(wvalid & !wlast) / msg_out = data_msg
1 clock / msg_out = addr_msg; awready = LOW
1 clock / rvalid, rdata
SEND_ADDR
RCV_DATA
(wvalid & wlast) / msg_out = data_msg
(wvalid & !wlast) / msg_out = data_msg
msg_in / rvalid, rdata
SEND_DATA
(wvalid & !wlast) / msg_out = data_msg

Fig. 5. Mealy FSM of NI Shell

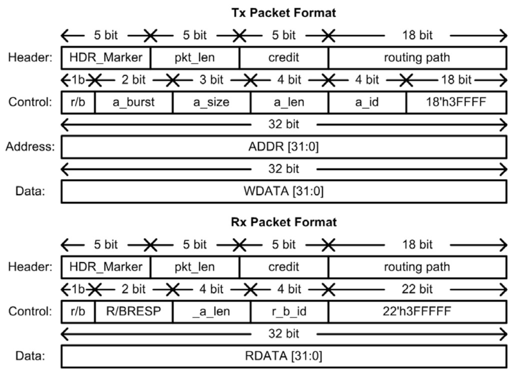

Fig. 6. Flit format defined in our system

1) Interfacing NI with AXI Ports: AXI protocol has 5 distinct channels, as listed here: read/write address channel, read/write data channel, and response channel. Address channels are composed of 32bit address data and associated control signals, and data channels are composed of 32bit payload and associated control signals. On the one hand, read operation is performed by sending address with associated control signals.

On the other hand, write operation is done by sending initial address with associated control signals, followed by data burst, whose length specified from **awlen** of control signals. The **awlen** control signal declared at the beginning of the transaction, makes the NI design more efficient, because the system can predict the number of flits to transmit before all data is stacked into the NI. And every slave component attach to each NI also requires burst size (**awsize**), burst length (**awlen**), and operation ID (identifier for data interleaving, **awid**). Hence the NI packetizes all of these control signals, as shown in Fig. 6.

For a response message (Fig. 6), we've defined a bit(r/b) to indicate if the message corresponds read data or to a write response. Next two bits contain **R/BRESP** signal which indicates the result status of read or write transaction. According to response packet from slave, master determines whether retransmission is required or not.

Fig. 5 shows you a mealy machine for operating NI shell. On receipt of reset signal the state is initialized to IDLE. In IDLE state, if control and address signals are asserted on the channel, NI shell snatches them and generates messages described in Fig. 6. After shedding all data messages followed by control message, the state machine returns to IDLE state stimulated by **wlast** signal. As for response phase, it is a completely reverse process of message generation; based on incoming message it makes corresponding response signal or forwards data flits.

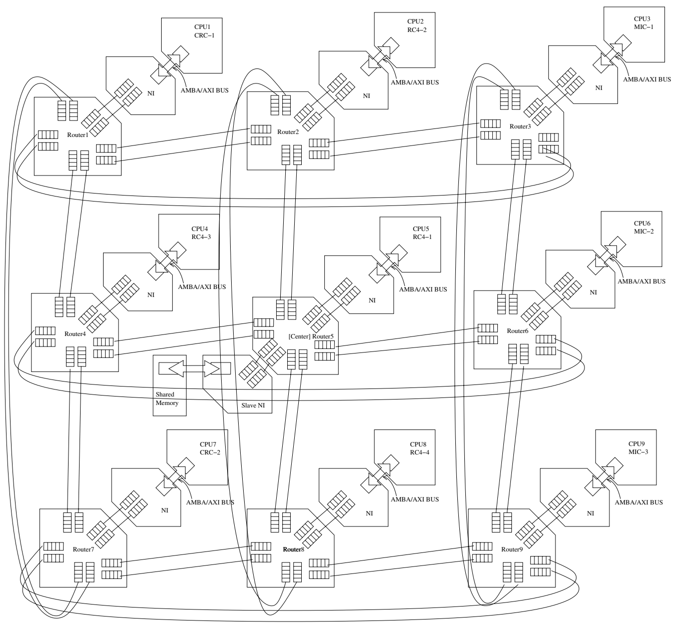

Fig. 7. TKIP cryptosystem mapped on the 3x3 torus MPSoC platform interconnected by on-chip network

IV. The Evaluation of Communication Architecture

The key of the communication performance in TKIP system is determined by the capability of the concurrent MSDU transmission. This concurrency is given by distributed channels of NoC connecting various components, as opposed to the shared channel in BUS. Before designing a RTL circuit, we made an exploration on these communication architectures with SystemC, to know them inside out. In this experiment, all the components - CPUs, NIs, and Routers - operate at the same frequency (100Mhz). The overall architecture is illustrated in Fig. 7.

Three cooperating components (RC4, MIC, and CRC32) can read MSDU from shared memory, on the fly, because the NI supports for multicast operation. For instance, at the first interval ($0 \sim 243\mu s$) in Fig. 2, MSDU flits are forwarded toward RC4-1, MIC-1, and CRC-1, simultaneously. This is possible, because the RC4-1 block imposes multicast read request on the shared memory, instead of point-to-point read command. In the realization, it takes 100ns for the read request to go from RC4-1 node to NI of shared memory, passing through router1, router4, and router5. And it takes 190ns for RC4-1 node to receive first response from the time at which first read instruction is fetched in RC4-1 CPU. And the total collapsed time was 350ns, when the transmission of first $16 Bursts \times 4 Bytes = 64 Bytes$ is completed. For the complete transmission of the whole $2304 Bytes$ MSDU, 36 read instructions should be executed, because the maximum burst length of AXI protocol is 16. Therefore, every $243\mu s$, 36 reads operation is done in $10.15\mu s$.

V. Conclusions

This paper explores an example of cryptographic application on a Multi-Processor NoC system. The platform seeks to meet the throughput requirement of 802.11n wireless LAN. To achieve the goal, firstly, the load of each component is balanced evenly, according to the performance measurement on a single ARM core. Then, the 9 components are interconnected with NoC fabric to optimize the communication cost and resolve the synchronization problem. As a result of efficient communication architecture, which is equipped with multicast operation, three components can read a MSDU in $10.15\mu s$, as opposed to $30.29\mu s$ in the case without multicast operation.

References

[1] *Amendment 6: Medium Access Control (MAC) Security Enhacements*, IEEE Std. 802.11i, 2004.

[2] M. H. Andrea B. and J. L., "The Final Nail in WEP's Coffin," in *Proc. of the 2006 IEEE Symposium on Security and Privacy (S&P06)*, 2006.

[3] Diamond Standard Processors - Beats the Competition Every Way. [Online]. Available: http://www.tensilica.com/diamond/di-competition.htm

[4] L. Benini and G. D. Micheli, "Powering networks on chips," in *Proc. ISSS*, 2001.

[5] J. Hu and R. Marculescu, "Energy- and Performance- Aware Mapping for Regular NoC Architecture," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, pp. 551–562, Apr. 2005.

[6] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconection networks." *IEEE Trans. Comput.*, vol. 36, pp. 547–553, May 1987.

[7] A. R. et al., "An Efficient On-Chip NI Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Configuration." *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, pp. 4–17, Jan. 2005.