Architecture-level Simulations with Rapid Power Estimations for Security Processors with Multiple Power Domains *

Chung-Wen Huang Yung-Chia Lin Yi-Ping You Jenq-Kuen Lee Ting-Ting Hwang

Department of CS, The National Tsing Hua University, Hsinchu 300, Taiwan
{cwhuang,yclin,ypyou}@pllab.cs.nthu.edu.tw, {jklee,tingting}@cs.nthu.edu.tw

Abstract— The power dissipation is the concern for SoC designs and embedded systems to extend battery life. Techniques like dynamic voltage scaling (DVS), power gating (PG), and multiple domain partitioning help provide mechanisms to reduce dynamic and static powers. Based on those techniques, the control system can be system software or hardware monitors. In ether cases, energy estimations in the architecture level are needed to facilitate the design space explorations of architecture and software scheduling designs for energy reductions. In this paper, we propose an architecture-level simulation environment for security processors with power estimation. The simulation environment includes a transaction-level modeling (TLM) simulator implemented in SystemC with multiple power domains, an analytical model, a workload generator, power parameter banks, versatile outputs, and succinct GUIs. Architecture developers can use it to evaluate different architecture configurations and retrieve performance results and power estimations. System software developers can use these tools for experiments in devising power-aware scheduling methods on security processors for power dissipation reduction.

I. Introduction

The low power issues reside in SOC designs and embedded systems either with the plug power supply or with the battery power supply. To reduce energy consumption and keep the performance improving, the researchers develop kinds of power saving mechanisms. Engineers in the integrated circuit process continue improving the semiconductor material, reducing the die size, lowering the supply voltage, and providing the standard cell library with multiple supply voltages or threshold voltages. The EDA companies and IC design houses are working with clock gating, power gating, multiple frequencies, and multiple power domain partitions. The techniques of multiple power domain is popular seen at the board level design where the supply voltages and frequencies are different among the CPU, memory, PCI bus, IDE, SCSI, and other key components. Recently, there are some companies [2, 9] implementing this techniques in their SoC platforms to improving the energy saving. For the system software, we propose control algorithms with lower energy consumption to analyze the efficient usage of operations. Also the power-aware compiler techniques are evaluated to be useful as the low power scheduling, instruction replacement, data/instruction locality, and power control instructions.

As energy estimations in the architecture level are needed to facilitate the design space explorations of architecture and software scheduling designs for energy reductions, we illustrate how to give such an estimation in the architecture level with a case study. A configurable security processor (**SP**) is used in this paper to illustrate the architecture level simulation and to estimate the energy consumption with different architecture configurations. **SP**s are designed to accelerate the computation-intensive algorithms like data encryption, user authentication, hash function, and the data compression. To guard data communications against malicious hackers, various network devices, like broadband access devices, and remote storage servers integrate **SP**s as accelerators. Especially in the wireless transmission where data are exposed in public domains, **SP**s are needed to ensure safety communications.

The security processor we presented is a SOC platform which has a DMA transfer module, a controller, an internal bus, several channels, and versatile crypto-engines. Our simulators are grown with developing the security processor. To evaluate the performance and perform design space exploration, the simulators are written in SystemC. SystemC can models the HW/SW co-design at the (untimed) functional level, transaction level [13], and the pin level (RTL level). The transaction level is suitable for platform design and the software/architecture verification. Therefore, in the simulator we model the bus and controller at cycle accurate transaction level and the crypto-engines at timed transaction level. Under this scheme, it is apt to add new cryptographic algorithms as the crypto-engines. Moreover, with the hardware development, the coefficients of performance and power are updated.

The remainder of this paper is organized as follows. Section II introduces the hardware architecture of security processor and basic functions of individual crypto-engines. Section III describes the implementations of simulator with multiple voltage domain and configurable functions. The simulation

*The work was supported in part by NSC-93-2213-E-007-025, NSC-93-2220-E-007-020, NSC-93-2220-E-007-019, MOE research excellent project under grant no. NSC93-2752-E-007-004-PAE, MOEA research project under grant no. 93-EC-17-A-03-S1-0002, no. 94-EC-17-A-01-S1-034 and an Intel research grant.

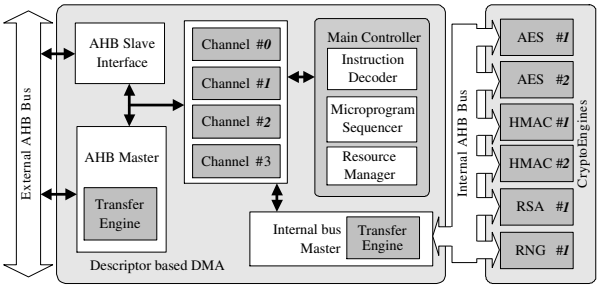

Fig. 1. Scalable architecture of security processor

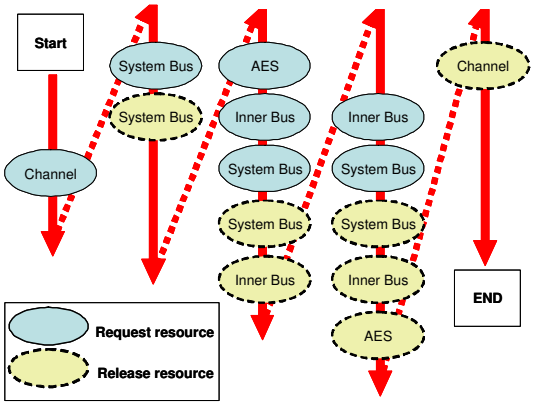

Fig. 2. The AES operation on descriptor-based security processor

tool flow and the experiments are presented in Section IV. Finally, the conclusion is given in Section V.

II. Architecture of the Security Processor

The main feature of our security processor is the scalable architecture. To achieve this feature, the internal buses are constructed inside this security processor. Therefore, versatile crypto-engines can be integrated into the **SP** by adopting the compatible bus interface wrappers. The other configurable parameters include the number of external buses, transfer engines, channels, and the internal buses. In the **SP**, a descriptor-base DMA controller is implemented to interpret the descriptors and manipulate the crypto-engines to perform proper cryptographic operations. The processing flow and cryptographic operations are handled by descriptors to reduce the control signals from the main processor. The descriptor is a data structure which contains the type of en-/de-cryption functions, the encryption key, the length of data, and pointers that indicate the data addresses. The descriptor also has a pointer to the next descriptor, so the DMA module could utilize the link list of descriptors to gather data without much overhead.

Fig. 1 illustrates a security processor design with one external bus, one descriptor-base DMA, four channels, one internal bus, two AES engines, two HMAC engines, one RSA engine, and one random number generator (RNG) engine. The main controller has a slave interface of external bus which accepts the control signals and returns the operation feedback via the interrupt port. In the main controller, there is an instruction decoder module and microprogram sequencer in charge of the descriptor decoding, and the resource allocation module distributes the resources as the descriptor demands. The DMA module integrates master interfaces of external bus with the channels and the transfer engines. Each channel stores the header of its processing descriptor. Transfer engines pass the data to dedicated crypto engines via the internal bus. The internal buses are designed to support multiple layers for high speed data transmission. Because the execution time of the crypto engine may be varied, the crypto engine will signal the main controller when the operations are done. The AES engine [14] supports the standard AES encryption and decryption with 128-, 192- and 256-bit keys, with both ECB and CBC modes. The RSA engine [15] performs the modular multiplication based on an enhanced word-based Montgomery algorithm, supporting scalable keys of the length up to 1024 bits. The HMAC [12] engine proposes an area-efficient integrated SHA-1/MD5 core. The RNG engine is a pseudo random number generator that consists of five linear feed-back shift registers with different structures and data scrambling in the output.

Although the original bus interfaces of our **SP** architecture are dedicated for accessing memory, they could be modified as network entrance interfaces. Together with package processing modules and physical MACs, the **SP**s can process data directly from the network. The similar architectures can be seen from Motorola MPC180/190 [6], Broadcom BCM5823, and Hifn 7954 series [7], except one thing that these **SP**s use wired connections instead of the internal buses.

To illustrate the kernel operations of **SP**s, Fig. 2 shows an AES operation in the architecture. At first, the user program calls the encryption libraries which pack the processing data as descriptors and activate the **SP**. When the controller of **SP** is aware of a start signal, it retrieves the memory address of descriptors and makes a channel ready to receive the descriptor information. At the "Setup channel data" phase, the main controller continually arranges the transfer engine and the master interface of external bus to get the descriptor information into the channel.

Next, in the "AES operation" phase, after sequentially requesting the AES cryptographic engine, inner bus, transfer engine, and master interface of external bus, the transfer engine will read data from memory and fill in the buffer into the AES cryptographic engine. Once data transmission is done, the main controller releases the requested resources except the AES crypto engine. When the operation of AES engine is done, it will send a signal to the main controller. As the main controller is aware of this state, it enters the "Store result" phase. It will request the inner bus, transfer engine and external bus master interface for storing the output data. The main controller then releases all the resources. The steps and phases in Fig. 2 can also be interleaved.

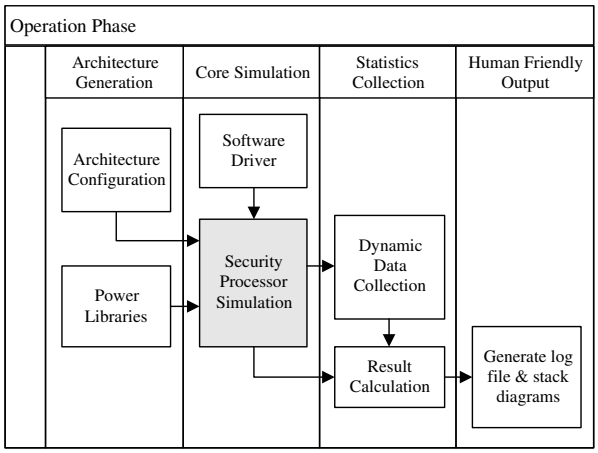

Fig. 3. Security processor simulation platform

III. Security Processor Simulation Methodology

This section describes the simulation of the configurable security processor presented in the Section II. We will introduce the full view of the simulation platform and the detailed modeling of the **SP**. After illustrating the performance simulation, we show a rapid power estimation methodology of a refined architecture of the **SP** with multiple voltage/frequency domains. We use SystemC as the design language and construct the architecture and operations of the **SP**.

In Fig. 3, the simulation platform was separated into "Architecture Generation", "Core Simulation", "Statistics Collection", and the "Human Friendly Output", according to the operation phase. In the "Architecture Generation" phase, there are two functions including the architecture configuration and the power library loading. The configuration support helps generate a configuration file consisting of the processing ability, the frequency level, the operating voltage, and the amount of each component. The library consists of the memory capacitance per unit, the average capacitance of each component, the operating voltages, and the frequency ratios in different voltages. The software driver and **SP** simulation are parts of "Core Simulation" phase. The software driver simulates the functions among the CPU, the **SP**, and the external memory. It fills the memory with encryption data from bench files. The bench files consist of the **SP** descriptors and the arrival time of descriptors. When the data arrival is catched in the simulation, software driver sends signals to notify the **SP** that encryption data is ready. When the **SP** simulation starts, the dynamic data collection continues to collect the information of the **SP** simulator. If the designers need to know the detailed of operations, the dynamic data collection can be used to generate the value change dump (.VCD) file which can be viewed in waveform tools as shown in Fig. 4. We have a sample code for dumping out the resource allocation table shown below.

```
if(vcd_on)
{
  char stbuf[20];
  sc_string NAME = "trace_file";
  sprintf(stbuf, "(%d)", (int)order);
  // trace file creation
  sc_trace_file *tf
        = sc_create_vcd_trace_file (NAME + stbuf);
  // External Signals
  sc_trace(tf,  clock, "clock(0)");
  NAME = "valid_CH";
  for (int i = 0; i< num_CH; i++)
  {
    sprintf(stbuf, "(%d)", i);
    sc_trace(tf, simulator.cp->cont->valid_CH[i],
                                        NAME + stbuf);
  }
  ...
}
```

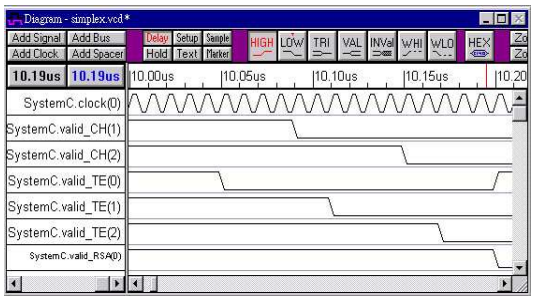

Fig. 4. A simplex waveform view from WaveFormer Pro

Then the simulation result of the performance and the energy consumption will become an input of a log file to a graphic generator at the "Human Friendly Output" phase. The organization of the **SP** simulator is shown at Fig. 5. Under the simulator, we have four modules: the crypto processor, external buses, external memory, and the clock generator. The clock generator distributes every module with user assigned frequencies. The external memory is connected with the crypto processor with the external buses. The accessing ports of external memory and the number of external buses are configurable. The external bus and memory operations are simulated in functional equivalence with specified delays of the read/write operations. The follow is the sample code of the read operation of the memory. The *memory_rwait*() pauses the operation in a selected delay and counts the reading times.

```
int M_MODULE::read()
{
        if(in_d_address == NULL)
        {
        cerr<<"READ address error!";
        exit(1);
        }
        memory_rwait(1);
        length.write(in_d_address->datal[pair_num]);
        memory_rwait(in_d_address->datal[pair_num]);
        data.write(in_d_address->datap[pair_num]);
        return 0;
};
```

The crypto processor consists of crypto modules, DMA transfer engines, channels, a controller module, and internal buses. We set the controller, DMA transfer engines, channels, and the internal buses at the same operating frequency and voltage. The frequency and voltage of the crypto modules are configurable. The internal buses are configurable and simulated as simplified AMBA AHB buses [1] without the BURST and

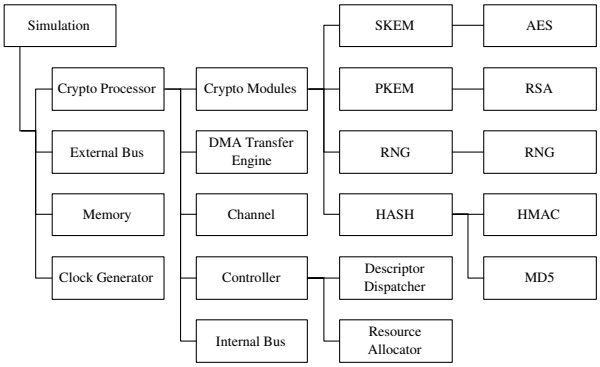

Fig. 5. Organization graph of the **SP** simulator

SPLIT mode. The arbitrator of internal buses control the traffic between transfer engines and the crypto modules. The **SP** has several DMA transfer engines that move descriptors from external memory and save them in the channels. If a descriptor indicates the specified crypto module, the controller will put the request to the waiting queue of a specified crypto module. Otherwise the controller will pick a free crypto module from the resource table.

We classify the crpto modules into the secret key encryption module (SKEM), public key encryption module (PKEM), the random number generation module (RNG), and the hash function module (HASH). They are functions simulated as the level shifters [2,5] that upgrade or downgrade the voltage level among the crypto engines and other modules. In addition, they provide the local frequency assignments to each crypto engine.

AES Cryptographic Module	
Connection Ports	**Functions**

//ports to clock generator
sc_in_clk aes_clk;
//ports to internal bus
sc_out < *sc_uint* < 32 >> ∗ HRDATA;
sc_inout < *bool* > ∗ HREADY;
sc_in < *sc_uint* < 32 > ∗ HADDR_s;
sc_in < *bool* > ∗ HWRITE_s;
sc_in < *sc_uint* < 32 > ∗ > ∗ HWDATA_s;
//ports to controller
sc_inout < *bool* > ∗ AES_start_setup;
sc_in < *sc_uint* < 32 >> ∗ AES_setup;
sc_out < *sc_uint* < 32 >> CM_interrupt;
sc_inout < *bool* > CM_INT_start;
sc_in < *sc_uint* < 32 >> CM_INT_owner;
	//initial AES
	void setup();
	//transfer the data in_out to internal bus
	void data_inout();
	//perform AES operations
	void do_aes();
	//calculate the energy consumption
	void sum_energy();
	//record the hamming distance of data
	void bus_ham(*sc_uint* < 32 > *data*);

TABLE I
THE CONNECTION PORTS AND FUNCTIONS OF AES MODULES

We have the connection ports and the implemented functions of AES crypto engine shown in the Table. I. The port connected to the SKEM is the local clock of AES operation. The ports to the internal buses provide data transitions under 32-bit width AHB bus. The rest of the ports are connected to the controller and signal the AES to start or interrupt the controller to gain encryption results. The *setup*() function identifies the type of en-/de-cryption, the key size, and the operation mode of AES algorithm. We also have a delay table in this function that support operation voltages at 0V, 1V, 1.2V, 1.5V, 1.8V, and 1.98V.

```
   double wait_time[6][6]
   = {
//Voltage:  0,  1,1.2,1.5,1.8,1.98
          {0, 15, 20, 35, 63, 100},    // <-0
          {0,  0,  5, 25, 50,  80},    // <-1
          {0,  3,  0, 10, 30,  60},    // <-1.2
          {0,  8,  5,  0, 16,  45},    // <-1.5
          {0, 17, 12,  8,  0,  30},    // <-1.8
          {0, 28, 20, 15,  8,   0} };  // <-1.98
```

According to the table, the *setup*() function will pause in a chosen time to simulate the delay of voltage scaling and power gating. The *data_inout*() simulates the AHB wrapper that supports the data transmission, and we have the *bus_ham*(*sc_uint* < 32 > *data*) function to record the hamming distance of data transmission. The *do_aes*() performs the AES operations in the timed transaction level model and the *sum_energy*() calculates the energy during the AES operations according to power libraries. The following code segment describes behavior for the AES module to receive encryption key from the internal bus to the registers. The energy consumption is recorded at the same time.

```
for(int i = 0; i < length; i++)
{
    key[4*i+0] = HWDATA_s.read()[i].range(31,24);
    key[4*i+1] = HWDATA_s.read()[i].range(23,16);
    key[4*i+2] = HWDATA_s.read()[i].range(15,8);
    key[4*i+3] = HWDATA_s.read()[i].range(7,0);
    bus_ham(HWDATA_s.read()->range(31,0));
    wait();
    sum_energy();
}
```

The ideas of the *bus_ham*(*sc_uint* < 32 > *data*) and *sum_energy*() functions are according to the energy equation. For the high level power estimation of chip components, the estimation result is data sensitive and scalable with the component size which influences the capacitance. The basic energy consumption equation is $E = C \times V_{dd}^2$ for a full charge/discharge of a component, where E is the energy, C is the capacitance, and V_{dd} is the supply voltage. The modified equation with the data sensitive is described below:

$$Energy = \alpha \times C \times V_{dd}^2$$

where α is the switching activity. To gain the switching activities, we have *bus_ham*(*sc_uint* < 32 > *data*) function to monitor the data variations. When the **SP** simulation starts, the functions record the difference of data values in every cycle. The capacitance of each component is built in the power libraries and accessed by the *sum_energy*() function. The capacitance information is from the synthesis result of the **SP** design based on UMC 0.18 CMOS process with 1.8V operating voltage. After we have the energy summation, the power can be obtained from the equation:

$$Power = \alpha \times C \times V_{dd}^2 \times f = Energy \times f = {Energy}/{Time}$$

where the *Power* is from *Energy* plus f(frequency) and the frequency is the inverse of execution time. Therefore, we have the energy amount and can report the average power of each

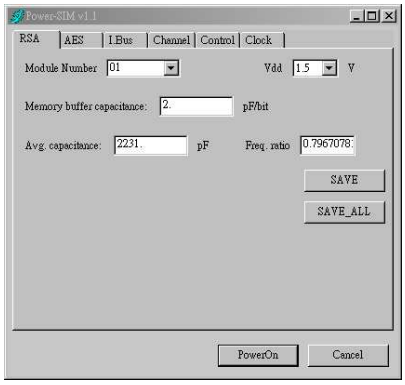

Fig. 6. Power library tool support read and modification

component. To make it easily to modify the power libraries, we build a library construction tool in Fig. 6. The library tool shows libraries in a succinct GUI. When the user fill the main power information, it will automatically generate the rest power information in the back ground. The frequency is related to the V_{dd} according to the equation.

$$Frequency = k \times (V_{dd} - V_{th})^2 / V_{dd}$$

where the V_t is the threshold voltage. To calculate the relative frequency, we estimate the utmost frequency of crypto modules in the original hardware design at the V_{dd} = 1.8 V. With the V_{th} information from CMOS libraries, the coefficient k is obtained. Therefore, if the operating voltage changes during the simulation, we can have the specified module run in an adaptive frequency.

IV. Experiment Results

The **SP** simulation platform can be used in various ways for architecture and scheduling policy designs on both performance and power. We illustrate such usages with two experiments, one on the design space exploration (DSE) toolkits to point out the bottleneck of an architecture design, and one on using the power controls of multiple power domain. The **SP** simulation platform gives estimations for different designs.

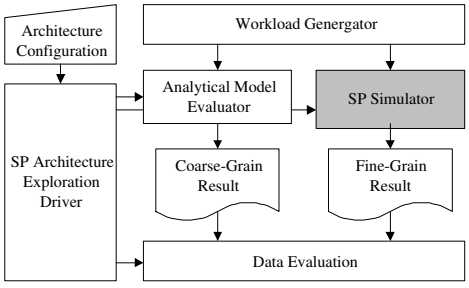

Fig. 7. Design space exploration toolkit

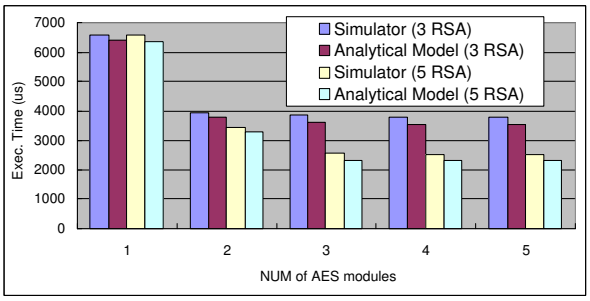

Fig. 8. Performance simulation results of multiple RSA and AES modules

A. Experiment 1: DSE toolkits

We have built a toolkit based on our **SP** simulation platform. It integrates the simulator and the analytical model evaluator into the unified design space exploration interface as shown in Fig. 7. The analytical model [10, 11] of the **SP** was developed as a simple and appropriate solution for rapid illustration of architectural behavior in the distributed parallel processing design of SOC, which is extended based on super-computer behavior modelings [3, 4, 8]. The architecture configuration describes the architecture parameters accepted by the **SP** architecture exploration driver and used for both the analytical model and the **SP** simulator. The workload generator provides the statistics of workloads to the analytical model evaluator and descriptor data to the **SP** simulator. The descriptor data resource can be automatically generated by the workload generator itself or by measures of real applications. The data evaluation tool in the DSE toolkits performs the comparison between the coarse grain result from the analytical model and fine grain result from the **SP** simulator.

To evaluate the architecture design, we set up both the **SP** simulator and the analytical model with the same parameters of hardware configurations. The input workload for the **SP** simulator is measured by SSH activities on a real server and then condensed to emulate higher workload in 1000 μs. In the analytical model, we use the averages of the same input workload. The working frequency of **SP** and external bus is simulated at 133MHz. Fig. 8 shows the execution results for a configuration of the **SP** architecture in 1 internal bus, 2 transfer engines and 10 channels. We have the 3 RSA module configuration and the 5 RSA module configuration. The number of AES modules is increasing with the extending of X-axis. The Y-axis is the simulation time according to the **SP** clock generator. The internal AHB bus has the same frequency to the external bus. The RSA modules operate in 100MHz and each of them provides 3.17Mbps processing rate. The AES modules operate in 100MHz and each of them supports up to 706Mbps processing rate. By profiling the operations of a SSH server, the raw data workload is 3.2Gbps, and the AES encryption data request rate and the RSA demands are 100:1. With the result in Fig. 8, we realize in the configuration (AES module = 1) that the workload costs over 6800 μs to be finished. When the number of AES increases to 2, the time is decreased to be less than 4000 μs. It's because the lacks of AES mod-

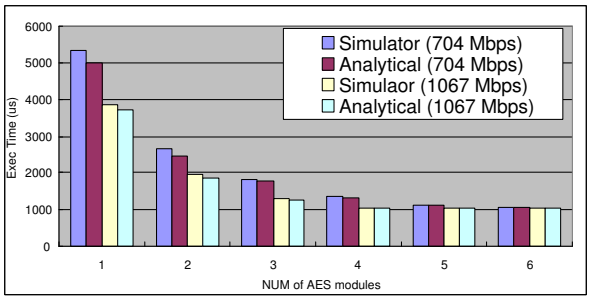

Fig. 9. Performance simulation results of different frequency AES modules

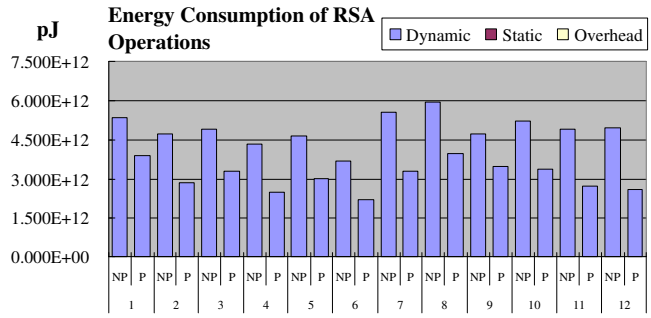

Fig. 10. Energy simulation result of RSA modules

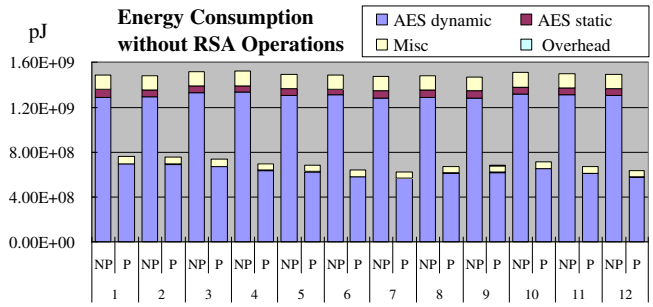

Fig. 11. Energy simulation result of **SP** except RSA modules

ules presents performance bottlenecks in the first case. Next, the performance bottleneck is again unveiled by the comparison of configurations in 3 and 5 RSA modules between 2 AES case and 3 AES case. In the 3 AES configuration, the simulation time is around 4000 μs in 3 RSA models configuration, but the simulation time is reduced to 2500 μs in 5 RSA models configuration. AES module dominates performance with this configuration. Finally, when the AES modules increase from 3 to 5 there is no further performance gained in Fig. 8. The performance bottleneck at this stage belongs to other resources such as the transfer engines, bus transfer rate, and controller overheads.

We also have experiments in considering the effects of the operating frequency and the number of modules. We set a configuration that the system is operated at 100Mhz and the AES modules are operated in 100MHz(704Mbps) or 150MHz(1067Mbps). The workload is 2.8Gbps with only AES encryption data. Fig. 9 shows that one 1067Mbps AES module assembled in the **SP** can improve around 28% of one 706Mbps AES module. When the number of AES module increases, the performance gap is narrowed. When the AES module module increases to 6, the simulation time approximates to 1000μs.

B. Experiment 2: Power-aware Scheduling

In this experiment, we address the variable voltage controls of the **SP** simulation platform. To utilize the power control features, we use a scheduling algorithm [11] to re-organize the descriptor sequence and insert the power controls in the descriptors. The power controls include the voltage scaling and the power gating. The power gating denotes the specified modules to clamp operating voltage V_{dd} and the voltage scaling sets the operating voltage to the specified module.

We implement a randomized security task generator to generate benchmark descriptor files for the simulator. The generator can generate the simulated OS-level de-/en-cryption jobs and each job has randomized operation type, randomized data size, randomized keys and content, randomized arrival time, and randomized deadline, according to a settable configuration of job arrival distribution type, job number, job density, ratio of distinct operation types, job size variance, and job deadline variance. Each generated job is then converted by the generator to numerous descriptors executable by the simulator. The generated benchmarks consist of 12 sets files with different task generator configurations listed in Table II. They are mainly divided into three types of arrival distributions. Each distribution type has three sets with different task slackness, which are dependent on job density and job deadline range: the first set has the high density and short deadline; the second has the high density and long deadline; the third has the low density and long deadline. We have generated 100 distinct descriptor files for each set and computed their average energy consumptions from the results of the simulator.

The average energy consumptions of **SP** are separated into RSA modules (Fig. 10) and the other components (Fig. 11). The *NP* means that descriptors proceed without power scheduling and the *P* means with the power scheduling support. The *static* means the energy consumption from the the static power and the *dynamic* means the means the energy consumption when the components are executing. The *Misc* is the energy consumption of the channels, controller, and the internal bus. The *Overhead* is the delay energy consumption due to the voltage scaling time and the voltage clamping time. The experiments results show that the average energy reduction is up to 37% achieved by the power-aware scheduling scheme. Again our energy simulator in the architecture level provides a tool for evaluating different voltage scheduling policies.

V. Conclusion

The architecture simulation platform provide the insight of the performance analysis and power estimation of the security processor. It provides the infrastructure of the hardware

Set	1	2	3	4	5	6	7	8	9	10	11	12
Distribution	uniform				normal				exponential			
Job number	300											
Jobs/Time(ms)	1500	375	1500	375	1500	375	1500	375	1500	375	1500	375
AES:RSA	60:1											
Max data size (bytes)	1280											
Max AES deadline (ms)	3072		3430		3072		3430		3072		3430	
Max RSA deadline (ms)	13312		15872		13312		15872		13312		15872	
Dynamic energy reduction (%)	26.85	39.45	32.36	42.42	34.61	40.50	40.30	33.03	26.16	35.10	44.17	47.23
Leakage energy reduction (%)	82.91	81.68	83.09	82.53	83.92	82.41	83.17	80.91	80.87	82.84	82.82	82.27
Total energy reduction (%)	26.85	39.45	32.37	42.42	34.61	40.50	40.30	33.03	26.16	35.10	44.17	47.23

TABLE II
BENCHMARK SETTINGS AND RESULTS

design flow that supports the design space exploration. The simulation platform is is to be modified and can generate the simulation result in a reasonable time, that helps the designers to do experiments on the **SP**. We will continue making efforts to improve the simulation details and make the **SP** simulator to a more generic simulation platform.

REFERENCES

[1] ARM. *AMBA. Specification (Rev 2.0)*, 1999.

[2] ARM. *Intelligent Energy Controller Technical Overview*, Aug 2004.

[3] D. Atapattu and D. Gannon. Building analytical models into an interactive performance prediction tool. In *Proceedings of ACM Supercomputing 89*, pages 512–530, November 1989.

[4] F. Bodin, D. Windheiser, W. Jalby, D. Atapattu, M. Lee, and D. Gannon. Performance evaluation and prediction for parallel algorithms on the bbn gp1000. In *Proc. of the 4th ACM International Conference on Supercomputing*, pages 401–403, June 1990.

[5] J.-M. Chang and M. Pedram. Energy minimization using multiple supply voltages. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 5(4), Dec 1997.

[6] N. Gammage and G. Waters. *Securing the Smart Network with Motorola Security Processors*, March 2003.

[7] Hifn. *7954 security processor Data Sheet*, December 2003.

[8] K. Hwang and F. Briggs. *Computer Architecture and Parallel Processing*. Mc Graw-Hill, 1984.

[9] INTEL. *Intel PXA27x Processor Family Developers Manual*, Oct 2004.

[10] Y.-C. Lin, C.-W. Huang, and J.-K. Lee. System-level design space exploration for security processor prototyping in analytical approaches. In *Asia and South Pacific Design Automation Conf. (ASP-DAC)*, pages 376–380, Jan 2005.

[11] Y.-C. Lin, Y.-P. You, C.-W. Huang, J.-K. Lee, W.-K. Shih, and T.-T. Hwang. Power-aware scheduling for parallel security processors with analytical models. In *Languages and Compilers for Parallel Computing (LCPC)*, Sep 2004.

[12] M.-Y.Wang, C.-P. Su, C.-T. Huang, and C.-W.Wu. An hmac processor with integrated sha-1 and md5 algorithms. In *Asia and South Pacific Design Automation Conf. (ASP-DAC)*, pages 456–458, Jan 2004.

[13] S. Pasricha. Transaction level modeling of soc with systemc 2.0. Technical report, Synopsys Users Group Conference, 2002.

[14] C.-P. Su, T.-F. Lin, C.-T. Huang, and C.-W. Wu. A high-throughput low cost aes processor. *IEEE Communications Magazine*, 41(12):86–91, Dec 2003.

[15] C.-H. Wang, C.-P. Su, C.-T. Huang, and C.-W. Wu. A word-based rsa crypto-processor with enhanced pipeline performance. In *4th IEEE Asia-Pacific Conf. ASIC*, Aug 2004.