

Architecture Design of Computing Intensive SoCs*

YUE Yao (岳 耀), ZHANG Chunming (张春明), WANG Haixin (王海欣),
BAI Guoqiang (白国强), CHEN Hongyi (陈弘毅)**

Institute of Microelectronics, Tsinghua University, Beijing 100084, China

Abstract: Most existing system-on-chip (SoC) architectures are for microprocessor-centric designs. They are not suitable for computing intensive SoCs, which have their own configurability, extendibility, performance, and data exchange characteristics. This paper analyzes these characteristics and gives design principles for computing intensive SoCs. Three architectures suitable for different situations are compared with selection criteria given. The architectural design of a high performance network security accelerator (HPNSA) is used to elaborate on the design techniques to fully exploit the performance potential of the architectures. A behavior-level simulation system is implemented with the C++ programming language to evaluate the HPNSA performance and to obtain the optimum system design parameters. Simulations show that this architecture provides high performance data transfer.

Key words: architecture design; coprocessor; security accelerator; behavior-level simulation; system-on-chip (SoC)

Introduction

As information technology develops, the computing demands in various fields are rapidly increasing such as information security, multimedia encoding, and digital signal processing. Thus, coprocessors are often used to reduce the computing burden on the basic microprocessor circuitry to allow it to work at optimum speed^[1]. Since the required processing on coprocessors could often be reduced to a combination of several certain algorithms, the system-on-chip (SoC) design is a suitable design method for implementing these coprocessors because it can utilize different intelligent property modules to realize these algorithms and, therefore, increase the design efficiency^[2].

The architectural design is very important for SoCs because it greatly influences the performance, configurability, and extendibility of the systems. However, many existing SoC architectures are for microprocessor-centric SoCs^[3,4]. They are not very suitable for computing intensive SoCs because computing intelligent property modules lack the scheduling and control ability of microprocessors. This paper analyzes the characteristics of computing intensive SoCs and presents three SoC architectures. An implementation scheme that achieves seamless data processing is introduced in the architectural design of a high performance network security accelerator (HPNSA)^[5]. A behavior-level simulation is implemented with the C++ programming language to assess the HPNSA architecture.

1 Analysis of Computing Intensive SoCs

1.1 General analysis

Good reconfigurability and extendibility are crucial

Received: 2008-04-03; revised: 2009-03-16

* Supported by the National Natural Science Foundation of China (No. 60576027) and by the National High-Tech Research and Development (863) Program of China (No. 2006AA01Z415)

** To whom correspondence should be addressed.

E-mail: chy-ime@tsinghua.edu.cn; Tel: 86-10-62779252

requirements for computing intensive SoC architectures. To achieve these objectives, the algorithm engines (AEs), the intelligent property modules in the computing intensive SoCs, should concentrate on data processing and have the simplest system interface so that they can easily be incorporated into the SoCs. However, this does not mean that AEs should only have a very simple function. Many AEs do very complex processing in multiple modes, but the AEs should not control the data transfer and task scheduling for the following four reasons.

(1) Unlike microprocessors, AEs are usually application specific integrated circuits (ASIC) to perform certain algorithms and they lack control and scheduling capabilities.

(2) Intelligent property modules are designed to accommodate various applications and they lack the global view of the entire working environment.

(3) SoCs need a unified interface to other systems but intelligent property modules are not necessarily designed with the same interface.

(4) SoC should have the maximum control over the intelligent property modules and schedule them freely to best complete the required tasks and to realize new functions.

On the other hand, the SoC framework should specialize in data transfer, task scheduling, and logic control. Since task scheduling and logic control are closely related, the SoC framework is usually divided into the data transfer module (DTM) and the central allocator and controller (CAC). The DTM is responsible for reading input data and configuration information, sending out results, and exchanging intermediate data between intelligent property modules, while the CAC is in charge of task scheduling, resource allocation, and logic control of the intelligent property modules and DTM.

1.2 Data transfer module

The most popular schemes for the DTM design use hierarchical levels of buses^[3,4,6,7] because (1) Intelligent property could be easily connected to on-chip buses with simple interfaces; (2) High performance modules and slow modules can use different data transfer buses; and (3) Hierarchical level of buses can separate the intelligent property modules from the external environment and allow the intelligent property

modules to concentrate on the data processing.

Although on-chip buses have lower efficiencies than hardware interconnections, the buses are adequate for most computing intensive SoCs because these SoCs usually communicate with other systems using print circuits board (PCB) buses, such as VME^[8], PCI-X^[9], and PCI Express^[10], which are much slower than on-chip buses. The PCB bus, rather than the on-chip bus, is then the bottleneck for computing intensive SoCs, as long as the data exchange between the intelligent property modules is not too frequent. Although heavy data exchange occurs in some cases, it often results from improper blocking of the intelligent property modules.

For computing intensive SoCs, low performance intelligent property modules rarely exist since the AEs are usually ASIC modules. Therefore, low performance buses that specialize in slow data transfer such as CoreConnect^[6] and AMBA^[7] are not included. Therefore, the following two hierarchies of buses are usually enough: inter bus for data transfer between the DTM and other systems, and inner bus for data transfer between the DTM and intelligent property modules. Since inter bus must match the PCB bus, little optimization can be done, but the inner bus allows free choice. Common choices for this bus are unidirectional buses, bidirectional buses, crossbar switch buses, and segmented buses^[3,4]. The latter two buses provide efficient data exchange between intelligent property modules, but they are not suitable for computing intensive SoCs because they require the intelligent property modules to take charge of the intermediate data transfer. Thus, the remaining problem is how to choose between a unidirectional bus and a bidirectional bus. Two unidirectional buses would lead to less bus conflicts than a single bidirectional bus. To do a simple quantitative comparison between the two schemes, define the stochastic processes $X_1(t)$ and $X_2(t)$ for unidirectional buses as follows:

$$X_1(t) = \begin{cases} 1, & \text{if input bus is busy;} \\ 0, & \text{if input bus is free,} \end{cases}$$

$$X_2(t) = \begin{cases} 1, & \text{if output bus is busy;} \\ 0, & \text{if output bus is free.} \end{cases}$$

Further assume that $X_1(t)$ and $X_2(t)$ are two independent stationary processes satisfying

$$E\{X_1(t)\} = \eta_1, \quad E\{X_2(t)\} = \eta_2,$$

where η_1 and η_2 denote the average occupancy rates of the two unidirectional buses. Then

$$\langle E\{X_1(t)X_2(t)\} \rangle = \langle E\{X_1(t)\}E\{X_2(t)\} \rangle = \eta_1\eta_2.$$

$\langle E\{X_1(t)X_2(t)\} \rangle$ represents the average possibility that both of the two unidirectional buses are busy, or the average additional bus contention rate that will be introduced by replacing the two unidirectional buses with a bidirectional bus. Assume that all three buses have the same width, w , and the same transfer rate, r , then the performance ratio between the bidirectional bus and the unidirectional buses is equal to

$$\frac{P_{bi}}{P_{uni}} = \frac{\eta_1wr + (\eta_2 - \eta_1\eta_2)wr}{\eta_1wr + \eta_2wr} = 1 - \frac{\eta_1\eta_2}{\eta_1 + \eta_2}.$$

This ratio decreases as η_1 or η_2 increases while P_{bi} approaches P_{uni} when η_1 or η_2 approaches 0 and it is only 1/2 of P_{uni} when $\eta_1 = \eta_2 = 1$. The above equation provides a criterion for choosing what type of bus should

be used once the required performance is determined.

Three possible architectures shown in Fig. 1 are for computing intensive SoCs with two levels of hierarchical buses. In these architectures, the DMAs are responsible for caching data between the two levels of buses. In Fig. 1a, all the DMAs are bidirectional because both the inner bus and the inter bus are bidirectional. With unidirectional inner buses, the DMAs can either be bidirectional or unidirectional as in Figs. 1b and 1c. If bidirectional DMAs are used, they could also be responsible for exchanging data between different intelligent property modules. An alternative method suitable for all three designs uses on-chip SRAM connected to the inter bus to store the intermediate results. Both approaches have their own strengths and weaknesses.

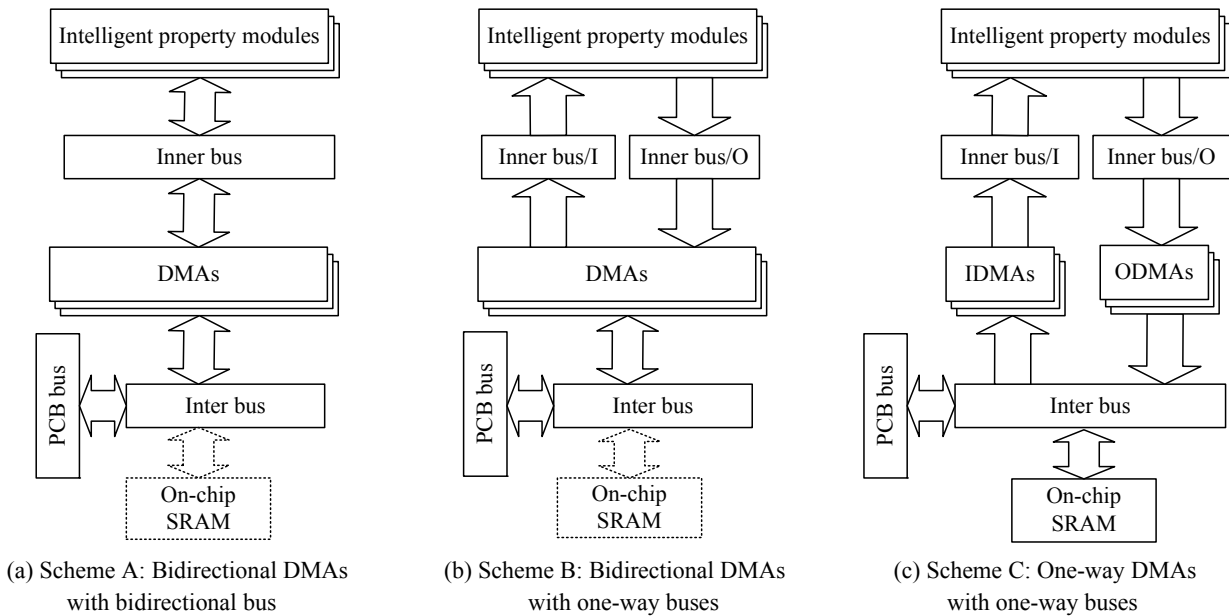


Fig. 1 Architectures for computing intensive SoC

Approach 1 Bidirectional DMAs to transfer data between intelligent property modules are more efficient and result in less contention for the inter bus. However in many cases, the processing rates of the different intelligent property modules do not match, so intermediate results may spend a long time in the DMAs which may obstruct other data that needs to be transferred immediately. To alleviate this side effect, more DMAs are required. Furthermore, this approach lacks flexibility for the following two reasons. First, each intermediate result will occupy one DMA regardless of its length, which wastes memory. Second, the input data of one intelligent property module might be

the results of several other intelligent property modules. If the DMA is responsible for this data exchange, it must guarantee the right storage order of these results. Although this function can be implemented, this will lead to a complex DMA structure and the DMA must be responsible for part of the control and scheduling. Therefore, this approach is most suitable when the intelligent property modules have similar processing rates and only simple intermediate data exchanges.

Approach 2 Using on-chip SRAM to store intermediate results is less efficient because additional data transfer is required on the inter bus. However, no data obstruction can occur in the DMAs and the DMAs

structures are relatively simple because they need not do the control and scheduling. In addition, using on-chip SRAM to store intermediate data is more flexible. Thus, when the processing rates of the intelligent property modules differ greatly or complex intermediate data transfer is required, this approach is more appropriate.

If Approach 2 is used, Figs. 1b and 1c differ only in the DMA design. Bidirectional DMAs in scheme B of Fig. 1 are more reusable, but with more complex structures. In contrast, scheme C needs more DMAs, but it simplifies the structures of the inner bus, inter bus, and DMAs. Therefore, scheme C will shorten the design period and will not necessarily result in a larger area.

When intermediate data exchanges are very frequent, specialized buses and DMAs can be introduced to increase performance, at the expense of a larger area and higher power dissipation.

1.3 Central allocator and controller

One responsibility of the CAC is task scheduling and resource allocation. However, this is a network processor (NP)-hard problem in the strong sense^[11]. The simplest solution is a first-come-first-served, which is reasonable when tasks arrive in serial. Gang scheduling has been used to improve this method^[12].

The other responsibility of the CAC is to control the intelligent property modules and DTM function and timing. Proper design of control logic is critical to fully exploit the potential of the DTM. To achieve seamless data processing, once an intelligent property module completes its processing, its next input data should be already in a DMA while another DMA should be ready to send out the result. A control policy to satisfy this requirement is introduced in the HPNSA architectural design.

The CAC can be implemented as either an ASIC or an embedded microprocessing unit (MPU) based on the performance and programmability required by the system specification.

2 HPNSA Architectural Design

2.1 Introduction to the HPNSA

Information security is playing an increasingly important role in protecting data on the Internet. The NP^[13],

as the fifth generation router, is not adequate for data encryption and decryption because the computing load is very heavy^[14]. Therefore, an HPNSA is often used with an NP for specialized data encryption and decryption, especially to better support the socket secure layer (SSL) and IP security (IPSec) protocols^[15,16].

A typical HPNSA should be able to perform block cipher algorithms, public key algorithms, hash functions, as well as random number generation. It should also have the capability to execute several algorithms in a chain which is common in the SSL and IPSec protocols^[5]. One such example is to use the triple data encryption standard (triple-DES) algorithm to encrypt the input data and then to sign the result using the SHA-1 algorithm.

2.2 Overall design

The design objectives of this accelerator are^[5] (1) using 0.18 μm CMOS digital technology; (2) using the PCI-X bus (133 MHz, 64 bits) interface^[9]; (3) using 200 MHz clock frequency except for the PCI-X interface; (4) implementing block cipher algorithms including DES, triple-DES, and AES; (5) implementing public key algorithms including ECC and RSA; (6) implementing hash algorithms including SHA-1 and SCH; (7) implementing random number generation (RNG); (8) executing the algorithms in a chain to better support the SSL and IPSec protocols; and (9) having high performance, good configurability, and extensibility.

Since the processing rate of public key algorithms is much slower than that of block cipher algorithms and hash algorithms, using on-chip SRAM to store intermediate results is more reasonable as has been analyzed in Section 1. The performance ratio between a bidirectional bus and unidirectional buses can be calculated using

$$\frac{P_{\text{bi}}}{P_{\text{uni}}} = 1 - \frac{\eta_1 \eta_2}{\eta_1 + \eta_2} \geq 1 - \frac{(\eta_1 + \eta_2)^2}{4(\eta_1 + \eta_2)} = 1 - \frac{\eta_1 + \eta_2}{4}.$$

Since on-chip SRAM is used to store intermediate results, all the data transferred by the inner bus must also be transferred by the inter bus. Therefore,

$$(\eta_1 + \eta_2) \times 200 \leq 1 \times 133, \quad \eta_1 + \eta_2 \leq 0.665,$$

$$\frac{P_{\text{bi}}}{P_{\text{uni}}} \geq 1 - \frac{0.665}{4} = 0.83375.$$

Although using a bidirectional bus reduces the inner bus performance by only 16.625%, unidirectional buses

are used because not only high performance is the primary design goal, but also one unidirectional intelligent property bus suitable for this system^[17] is available. To simplify the DTM design, the architecture in Fig. 1c is used in this design.

The simplified architecture is illustrated in Fig. 2 with emphasis on the data transfer. When the NP needs the HPNSA for data encryption or decryption, it sends a descriptor to the CAC that describes the task details, such as which algorithm or algorithms to use, the configuration memory address, the input data, and the result. If this task requires the HPNSA to execute several algorithms in a chain, the CAC divides this task into several “middle-level tasks (MT)”, each responsible for one algorithm. The AEs required by the MTs are allocated by the CAC. If an MT need not process data in succession, such as the DES algorithm in the electronic code book (ECB) mode, the MT is allowed to occupy more than one AE at the same time. The intermediate result generated by the first MT will be used by successive MT for the same task stored in the on-chip SRAM. Since one MT may execute an algorithm with very long input data, the MT is further divided into sub tasks to guarantee fair use of the bus among all the MTs. Sub tasks fall into two types, i.e., sub configure task (SCT) to transfer the configuration and sub data task (SDT) to transfer and process the data. In the final design, each SDT is no longer than 512 bytes.

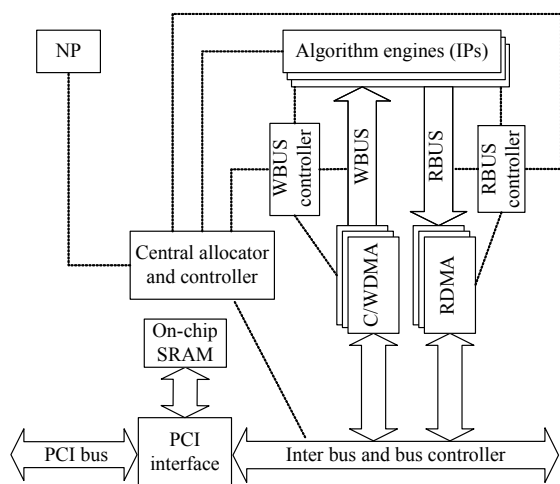


Fig. 2 Simplified architecture schematic diagram

2.3 SCT

After an MT occupies a certain AE, the CAC sends out one SCT to transfer the configuration, including the

algorithm mode, the key to be used, and some algorithm related parameters, to the allocated AE. The algorithm mode is described in the descriptor, while other configuration is in off-chip memory. This process executes as follows. The CAC writes the algorithm mode into the allocated configuration transfer DMA (CDMA) and tells the inter bus controller to transfer the other configuration information to this CDMA from the off-chip SRAM. Meanwhile, the CAC tells the write bus (WBUS) controller which AE is to receive the configuration information. The allocated CDMA requests the WBUS as soon as the configuration arrives. The WBUS controller only grants the request after it is notified by the corresponding AE that the previous execution, if exists, has completed. The AE uses some time to configure itself after the configuration arrives and then notifies the WBUS controller that it is ready to read input data, which ends the SCT process. This process is for SCTs that configure only one AE, but since one MT can simultaneously occupy multiple AEs, one SCT could also configure more than one AE. In this case, WBUS controller notifies all the AEs allocated to the MT to receive the configuration.

2.4 SDT

The SDT is responsible for transferring input data from the memory into the AEs and transferring the results to the memory. Here, memory stands for either on-chip SRAM or off-chip memory. The SDT can be subdivided into the input phase, processing phase, and output phase. The SDT input phase is much the same as for the SCT with only the following differences.

(1) The SDT input phase uses a write input DMA (WDMA) rather than a CDMA. The WDMA need not read any data from the CAC as the CDMA does. The WDMA also has a lower priority than the CDMA for both the inter bus and WBUS to guarantee that the configuration arrives at the AE prior to the data.

(2) Data could only be transferred to one AE in the SDT input phase while the configuration can be transferred to several AEs in the SCT.

(3) The WBUS controller grants requests from WDMA only when it has received signals from the corresponding AE indicating that it is properly configured by the SCT and the previous data processing on this AE, if exists, has been finished and the result has been sent out.

In the processing phase, the AE executes the required operation on the input data. Near the end of the processing, the AE sends out a NEARREADY signal to the CAC and read bus (RBUS) controller. Upon receiving the NEARREADY signal, if a waiting SDT belonging to the same MT exists, the CAC configures the AE to this SDT; otherwise, it allocates another MT to this AE. One AE can be allocated to two SDTs or to one SDT plus one SCT. Actually, the SDT/SCT arrangement can transfer input data to the WDMA/CDMA pair and then wait for completion of the current processing task. This is controlled by the WBUS controller as mentioned before. In this way, there is no conflict at this AE and the input data for the next SDT/SCT is transferred to the WDMA/CDMA in advance. After the RBUS controller receives the NEARREADY signal, it allocates a free read output DMA (RDMA) for this SDT and notifies the CAC that this RDMA is responsible for SDT the output phase. Based on this information, the CAC notifies the inter bus controller what output address to use when this RDMA applies for the inter bus. In this way, after the NEARREADY signal is sent out by the AE, the system starts setting this SDT output configuration. When the present process ends, the result can be sent out immediately without delay.

These two features introduced by the NEARREADY signal dramatically reduce the input and output waiting times and may completely eliminated them. Therefore, this architecture can guarantee seamless data processing as long as the time to send out the NEARREADY signal is set properly. In the final design, the NEARREADY signal is sent out no less than 200 cycles prior to the finish time of the entire execution based on simulation analysis. If the AEs do not support the NEARREADY signal, the finish signal could be used instead to guarantee correct function, but seamless data processing could not be achieved.

When the data processing finishes, the SDT enters its output phase. The RBUS transfers the result to the allocated RDMA, and then the inter bus transfers the result in the RDMA to the memory address received from the CAC as mentioned before. This ends the entire SDT process.

2.5 Task, MT, and CAC

In the scenario considered, the NP may require the

HPNSA to execute only one algorithm or several algorithms in a chain. Since each algorithm in the task requires one MT, the task contains only one MT in the first case and several MTs in the second case. The number of SDTs in an MT is determined by the input data length. The SCT is more complicated because the MT may occupy AEs when it is newly created by the task, or when AEs of the corresponding type are freed by other MTs. For all these cases, an SCT should be generated. The CAC is in charge of receiving descriptors from the NP, generating MTs, SCTs, and SDTs, and guaranteeing the right execution order of the MTs, SDTs, and SCTs, as well as generating control signals for the inter bus, WBUS controller, and RBUS controller.

2.6 Simulations

A behavior-level simulation was implemented in the C++ programming language to evaluate the HPNSA performance, determine the optimum design parameters, and certify the control logic. An object-oriented programming style is used to better describe hardware modules. The relationships between the objects in the simulation are shown in Fig. 3. In this figure, DES_ALLOC, ECC_ALLOC, CMDA_ALLOC, WDMA_ALLOC, and RDMA_ALLOC stand for the allocation units for the AEs and DMAs in the CAC, while TASK_GEN is responsible for generating random descriptors. This figure is only for demonstration, and the number of resources shown is not exact. Actually, these parameters can easily be altered in the simulation to determine the optimal parameters. This program uses simulation steps to simulate the clock. Each object records its current state, and renews their states when a new simulation step begins. The program calculates the system performance indices, such as the bus throughput based on the intelligent property modules processing rates.

The PCI-X bus limits the upper bound of the whole system throughput to $133 \text{ MHz} \times 64 \text{ bit} = 8.512 \text{ Gbps}$. The output throughput limit for block cipher algorithms is 4.256 Gbps since the input data and output data have almost the same lengths. The performance evaluation is then based on how close the throughput is to this upper limit.

Since the DES has the quickest processing rate and is the most demanding on the data transfer and DMAs,

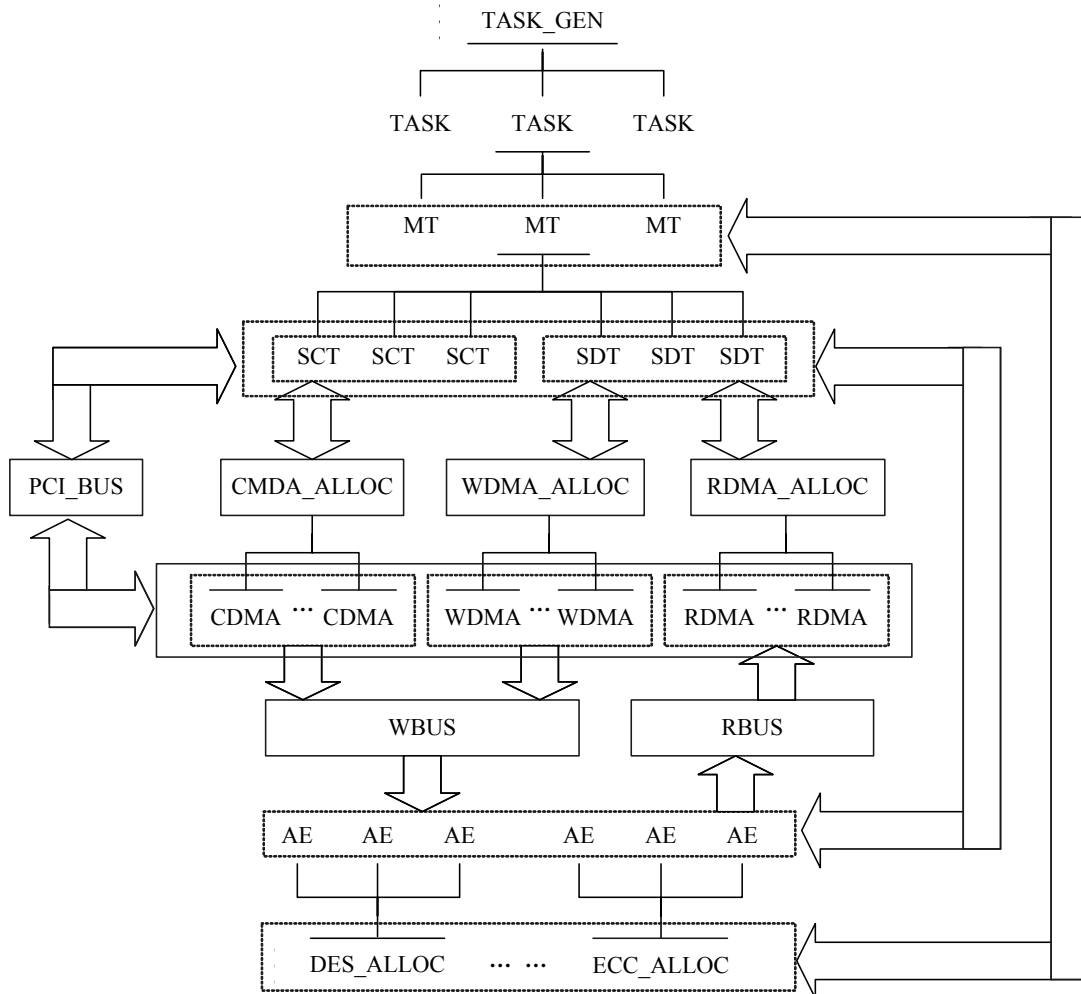


Fig. 3 Object relationships

it is used to determine the optimal number of DMAs. The simulations show that with 4 DES engines, 1 CDMA, 2 WDMAs, and 2 RDMA, the output throughput is 4.157 Gbps when 300 DES tasks are simulated. This output throughput is very close to the theoretical upper bound of 4.256 Gbps. If less DMAs are used, the throughput will not reach 4 Gbps. Thus, the simulations show that the architecture is very efficient for high speed data transfer. Another result that could be drawn from the simulation is that the architecture can satisfy the demand of at most four DES engines because of the transfer rate bottleneck imposed by the PCI bus. Actually, the fourth DES engine only increases the total output throughput by about 10%. However since the triple-DES algorithm, which executes more slowly than the DES, shares the DES engines with the DES algorithm, the 4 DES engines are worthwhile. This method was also used to set the optimal number of AEs for the other types. With the

given performance requirements, the simulations showed that 4 DESs, 4 AESs, 2 ECCs, 2 RSAs, 1 RNG, and 4 hash engines gave the best results.

No PCI bus conflicts with other systems are considered in the analysis. The simulations show that the DMAs are more useful with severe PCI bus conflicts because the DMAs cache the input and output data. Therefore, the DMAs make the system more robust, so the final design uses 1 CDMA, 3 WDMAs, and 3 RDMA to increase the system performance when contentions on the PCI bus are severe.

3 Conclusions

This paper shows that the specialization of the intelligent property modules and the SoC framework are vital for attaining good configurability, extensibility, and high performance in computing intensive SoCs. Different architectures are proposed and, although the

architectures are all highly configurable, extensible, and efficient, they are suitable for different situations. An implementation scheme that guarantees seamless data processing is given for the HPNSA architectural design. A behavior-level simulation shows that if the parameters are properly set, the HPNSA throughput approaches the theoretical limit of the PCB bus.

References

- [1] Galanis M D, Dimitroulakos G, Goutis C E. Performance improvements in microprocessor systems utilizing a coprocessor data-path. In: International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation. New Jersey, USA: IEEE Press, 2006: 85-92.
- [2] Rincon A M, Lee W R, Slattery M. The changing landscape of system-on-a-chip design. In: Proceedings of the IEEE on Custom Integrated Circuits Conference. New Jersey, USA: IEEE Press, 1999: 83-90.
- [3] Ryu K K, Shin E, Mooney V J. A comparison of five different multiprocessor SoC bus architectures. In: Proceedings of Euromicro Symposium on Digital Systems Design. New Jersey, USA: IEEE Press, 2001: 202-209.
- [4] Ryu K K, Mooney V J. Automated bus generation for multiprocessor SoC design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2004, **23**(11): 1531-1549.
- [5] Wang Haixin, Yue Yao, Zhang Chunming, Bai Guoqiang, Chen Hongyi. A novel unified control architecture for a high-performance network security accelerator. In: Proc. International Conference on Security and Management. Las Vegas, USA, 2007: 538-544.
- [6] IBM Incorporation. CoreConnect bus architecture. <http://www.ibm.com/us/>, 2001.
- [7] ARM Incorporation. AMBA peripherals. <http://www.arm.com/>, 1999.
- [8] VME64 standard. <http://www.vita.com/specifications.html>, 2001.
- [9] PCI-X 2.0a protocol spec. <http://www.pcisig.com/>, 1999.
- [10] PCI Express Base 2.0 specification. <http://www.pcisig.com/>, 2003.
- [11] Schild K, Würtz J. Scheduling of time-triggered real-time systems. *Constraints*, 2000, **5**(4): 335-357.
- [12] Schwiegeishohn U, Yahyapour R. Improving first-come-first-serve job scheduling by gang scheduling. *Job Scheduling Strategies for Parallel Processing*, 1998, **1459**: 180-198.
- [13] Shah N. Understanding network processors [Dissertation]. Berkeley: University of California, 2001.
- [14] Xie H, Zhou L, Bhuyan L. Architectural analysis of cryptographic applications for network processors. In: 8th International Symposium on High-Performance Computer Architecture. New Jersey, USA: IEEE Press. <http://www.cs.ucr.edu/~bhuyan/papers/np1.pdf>, 2002.
- [15] Motorola Incorporation. MPC190 hardware reference manual. <http://www.freescale.com>, 2001.
- [16] Broadcom Incorporation. BCM 5840 gigabit security processor. <http://www.broadcom.com>, 2003.
- [17] Zhang Chunming, Yue Yao, Wang Haixin, Bai Guoqiang, Chen Hongyi. An improved AHB bus frame adapted to a high-performance network security accelerator. In: IEEE Conference on Electron Devices and Solid-State Circuits. HsinChu, Taiwan, China: IEEE Press, 2007: 1159-1162.