Design and Implementation of a Multi-Core Crypto-Processor for Software Defined Radios

Michael Grand[1], Lilian Bossuet[2], Bertrand Le Gal[1], Guy Gogniat[3] and Dominique Dallet[1]

[1] IMS Lab., University of Bordeaux, firstname.lastname@ims-bordeaux.fr
[2] Hubert Curien Lab., University of Lyon, lilian.bossuet@univ-st-etienne.fr
[3] Lab-STICC Lab., University of Bretagne Sud, guy.gogniat@univ-ubs.fr

Abstract. This paper deals with the architecture, the performances and the scalability of a reconfigurable *Multi-Core Crypto-Processor* (MCCP) especially designed to secure multi-channel and multi-standard communication systems. A classical mono-core approach either provides limited throughput or does not allow simple management of multi-standard streams. In contrast, MCCP parallelism provides either high encryption data rate or simultaneous use of different ciphers. Up to eight cores can be used at the same time to reach a maximum throughput of 3460 Mbps. Nevertheless, parallelism is not sufficient for a multi-standard radio. Therefore, our architecture targets FPGA platforms to enable its evolution over the time by using hardware reconfiguration.

1 Introduction

Multiplicity of wireless communication standards (UMTS, WiFi, WiMax) needs highly flexible and interoperable communication systems. Software based solutions such as *Software Defined Radio* (SDR) are used to meet the flexibility constraint. Independently, most of radios have to propose cryptographic services such as confidentiality, integrity and authentication (secure-radio). Therefore, integration of cryptographic services into SDR devices is essential.

By using symmetric cipher and hash function, it is possible to propose such services. However, the use of cryptographic functions in secure systems tends to limit their overall throughput. Usually, in order to meet the system constraints (i.e. security, throughput, ...), cryptographic services are provided by cryptographic accelerators or programmable crypto-processors. While cryptographic accelerators provide the highest throughputs, they are, in most of the cases, algorithm dependant [12], [10]. In contrast, programmable crypto-processors are flexible, but they also provide lower encryption data rates [7], [5] than accelerators.

To improve the performance/flexibility trade-off, VLIW (*Very Long Instruction Word*) crypto-processor [15] or multi-crypto-processor [17] architectures have been designed. However, to the best of our knowledge, there is no published studies about multi-crypto-processor scalability. This paper presents a

reconfigurable *Multi-Core Crypto-Processor* (MCCP) especially designed to secure multi-standard and multi-channel communication devices. It also presents the MCCP scalability and the task allocation problem.

The paper is organized as follows: Section 2 presents previous works on high throughput cryptographic cores. Section 3 presents the architecture of the proposed MCCP. Its operation is explained in sections 4 and 5. Section 6 presents some results related to the implementation and the operation of the MCCP. Finally, section 7 concludes this paper.

2 Previous Works

Highest data rate are obtained by the use of highly pipelined cryptographic accelerators. In such cores, encryption algorithm is fully unrolled to produce an efficient pipelined design. For example, AES-GCM (*Galois Counter Mode* [1]) is especially designed to take profit of pipelined cores. There are several works dealing with hardware implementations of AES-GCM core [12], [14], [16]. These cores allow throughput of tens of gigabits per second, but this kind of architecture has several drawbacks.

Firstly, algorithm unrolling leads to high hardware resource consumption. Secondly, data dependencies in some block cipher modes (i.e CCM for *Counter with CBC-MAC Mode* [3]) make unrolled implementations useless. Finally, complex designs are needed when multiplexed channels use different standards. In consequence, pipelined cores are better suited for mono-standard radio than for multi-standard ones.

In fact, multi-standard systems often embed programmable crypto-processors. Such crypto-processors are highly flexible but commonly provide lower throughput than dedicated accelerators. Cryptonite [5] is a programmable crypto-processor which supports AES, DES, MD5 and others cryptographic algorithms. It is built around two clusters. Each cluster provides cryptographic functions used by block cipher algorithms. This implementation targets ASIC platform and reaches a throughput of 2.25 Gbps at 400 MHz for the AES-ECB algorithm. Celator [7] is composed of several *Processing Elements* (PE) which are connected together to form a matrix like a block cipher state variable. According to PE configuration, cryptographic functions are applied on the grid at each clock cycle. Celator is able to compute AES, DES or SHA algorithms, providing for example a throughput of 46 Mbps at 190 MHz when computing AES-CBC algorithm.

Another approach is to use multi-core architectures to reach high throughputs and remain flexible. Such architecture is composed of several *Cryptographic Cores* and at least one controller acting as a task scheduler. Independent cores may process messages from same channel or from different channels. By this way, high throughputs can be reached on multi-channel radios whatever the operation mode. Basically, multi-core approach for crypto-processors has almost the same benefits and drawbacks as multi-core general purpose processor approach (MPSoC issue). CryptoManiac [17] is based on a multi-core architecture, it achieves a throughput of 512 Mbps on ASIC platform at 360 MHz.

3 Multi-Core Crypto-Processor Architecture

3.1 Proposed Approach

Our work focuses on secure and high throughput SDR base-station where the MCCP is used as a red/black boundary [2] which provides most of the necessary cryptographic services needed by an SDR [2]. Such base-station has no resource-restriction that allows to use multi-core architectures to fulfil the above constraints. Currently, up to eight *Cryptographic Cores* may be embedded into one MCCP processor, if necessary more than eight cores could be used.

While architectures presented above target ASIC platforms, our architecture targets FPGA platforms in order to be as flexible as software components embedded in SDR. By this way, our Multi-Core Crypto-Processor (MCCP) can be updated by using hardware reconfiguration for security or interoperability reasons. Following section deals with the MCCP architecture.

3.2 General Architecture

Our MCCP is embedded in a much larger platform [8] including one main controller and one communication controller which manages communications going through the radio. MCCP does not generate session keys itself. Keys are generated by the main controller and stored into a dedicated memory. MCCP architecture is scalable; the number of embedded crypto-core may vary from two to eight per step of two. As said above, the maximum number of cores is fixed to eight. A four-core architecture is illustrated in Figure 1.

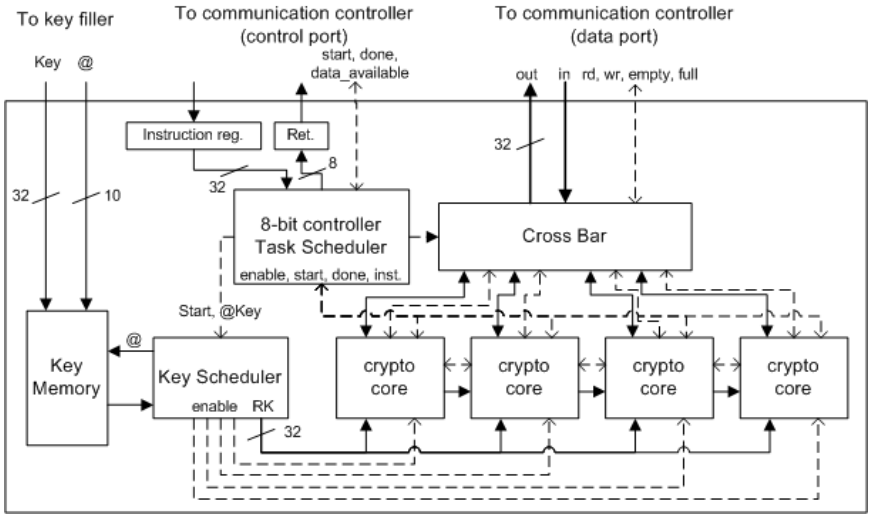

Fig. 1. Four-core MCCP Architecture

Proposed MCCP embeds one *Task Scheduler* which distributes cryptographic tasks to *Cryptographic Cores*. The *Task Scheduler* implementation uses a simple 8-bit controller which executes the task scheduling software. It manages the *Key Scheduler*, the *Cross Bar* and the *Cryptographic Cores*. *Task Scheduler* receives its orders from a 32-bit *Instruction Register* and returns values to the communication controller through the 8-bit *Return Register*. Some signals (*Start* and *Done*) are used to synchronize instruction execution.

Each *Cryptographic Core* communicates with the communication controller through the *Cross Bar*, it enables the *Task Scheduler* to select a specific core for I/O access. The *Key Scheduler* generates round keys from the session key stored in the *Key Memory*. Before launching the key scheduling, the *Task Scheduler* loads the session key ID into the *Key Scheduler* which gets the right session key from the *Key Memory*. To improve system security, the *Key Memory* cannot be accessed in write mode by the MCCP. In addition, there is no way to get the secret session key directly from the MCCP data port.

3.3 Cryptographic Core Architecture

Architecture of *Cryptographic Core* is presented in figure 2. On this schematic, dashed lines represent control signals. Each *Cryptographic Core* communicates with the communication controller and other cores through two FIFOs (512×32bits) and one *Shift Register* (4×32bits). Cryptographic functions are implemented in the *Cryptographic Unit* and they can be used through the *Cryptographic Unit Instruction Set Architecture*. *Cryptographic Unit* provides supports for AES encryption algorithm and several modes of operation (CCM, GCM, CTR, CBC-MAC). However, the AES cipher can be exchanged with any 128-bit block cipher at design step or during processor life using FPGA reconfiguration. AES round keys are pre-computed and stored in the *Key Cache*.

Cryptographic Core handles several block cipher operation modes which leads to complex control state machines. A more flexible approach is to use a general purpose simple processor to generate instruction flows executed by each *Cryptographic Unit*. Use of such processor allows us to simplify execution of loop conditions for packet encryption/decryption. Because this controller does not perform heavy computations, we use an 8-bit processor providing a simple instruction set. This processor communicates with the *Task Scheduler* to receive orders and return execution results. At prototyping step, a modified 8-bit Xilinx PicoBlaze controller [6] has been used. To save resources, it shares its double port instruction memory with its right neighbouring *Cryptographic Core*.

In the case of AES-CCM, the MAC value returned by the AES-CBC-MAC algorithm needs to be encrypted using the AES-CTR mode. To improve performances, AES-CCM needs to be computed in a concurrent way with two cores. The first one executes AES-CTR algorithm, while the second one executes AES-CBC-MAC algorithm. Inter-*Cryptographic Core* ports are used to convey temporary data, such as the MAC value, from a core to another.

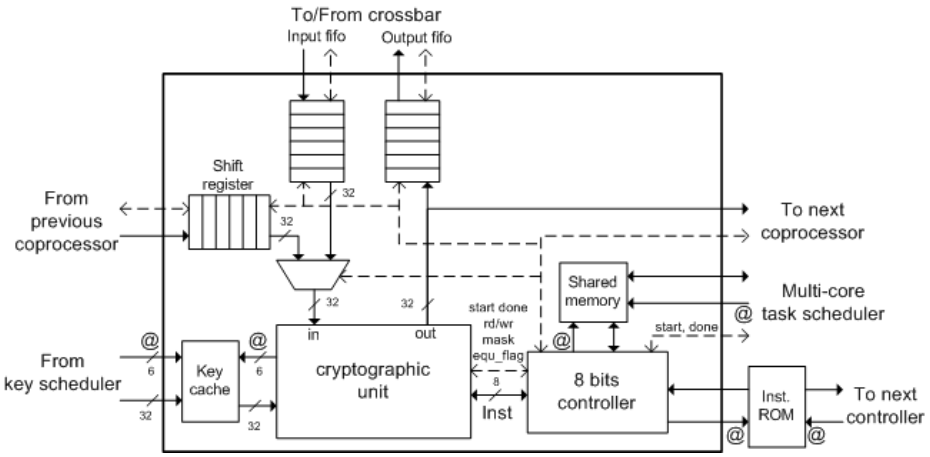

Fig. 2. Cryptographic Core Architecture

4 MCCP Operation

4.1 Packet Processing

Incoming packets are processed in the following way:

1. The *Task Scheduler* sends an instruction to the selected *Cryptographic Core* through the shared memory and triggers a *start* signal.
2. *8-bit Controller* starts pre-computations needed by the selected algorithm and loads data from the input FIFO once they are available.
3. Data are processed by blocks of 128 bits and filled into the output FIFO.
4. When all data have been processed, the *8-bit Controller* sends a *done* signal to the *Task Scheduler*. In order to protect the *Communication Controller* from software attacks (e.g. eavesdropping, spoofing, splicing), output FIFO is re-initialized if plaintext data does not match the authentication tag. Each FIFO can store a packet of 2048 bytes.

Packet processing is made according to several modes of operation described in the following section.

4.2 Available Modes of Operation

MCCP can execute AES-GCM, AES-CCM, AES-CTR, AES-CBC-MAC or any other block cipher mode of operation which only uses a 128-bit block cipher, XOR operators and counter values. Available modes of operation are described below:

– Packets from a same channel can be concurrently processed with different *Cryptographic Core* using the same channel configuration.

– Packets from different channels can be concurrently processed with different *Cryptographic Core* using different channel configurations .
– Any single packet can be processed on any *Cryptographic Core*.
– Using inter-core communication port, any single CCM packet can be processed with two *Cryptographic Cores*.

Right mode of operation is selected by the *Task Scheduler* according to requested channel algorithm and available resources. The *Task Scheduler* algorithm, which is used to allocate tasks on the MCCP cores, is described in section 5.

4.3 Implementation of Cryptographic Algorithms

ENCRYPT/DECRYPT MCCP instructions are used to launch packet processing. Once such instruction has been received by the MCCP, the communication controller sends data into the cores input FIFOs. Data must be sent in a specific way to be correctly interpreted by the cores. At first, *Initialization Vector* must be filled into the FIFO, then packet data must be filled. To finish, communication controller may append a message authentication tag. *Cryptographic Unit* only embeds basic operators in addition to AES core and GHASH core (GCM mode [1]). Therefore, it cannot be used to format the plain text according to the specifications of block cipher modes of operation. In consequence, the communication controller must format data prior to send them to the cryptographic cores.

Packet processing can start as soon as an 128-bit word is available in the FIFO. At lowest level, overall performances are limited by AES core computation delays. But, at higher level, performances are limited by the main loops of block cipher modes. The computation time of these loops may be used to approximate the maximum throughput of the implemented ciphers. Loop computation times, in number of clock cycle, for 128-bit AES-GCM and AES-CCM are equal to:

$$\begin{aligned} T_{GCMloop} &= T_{CTRloop} = T_{AES} = 49 \\ T_{CCMloop\_2cores} &= T_{CBCloop} = T_{AES} + T_{XOR} = 55 \\ T_{CCMloop\_1core} &= T_{CTRloop} + T_{CBCloop} = 104 \end{aligned}$$

Eight cycles must be added to these values for 192-bit keys and eight more cycles must be added for 256-bit keys. Also, pre and post loop invariant computations must be taken into account:

$$\begin{aligned} T_{GCMinv128} &= 337 \\ T_{CCMinv\_2cores128} &= 354 \\ T_{CCMinv\_1core128} &= 710 \end{aligned}$$

In addition, packet processing delay must include I/O access delays. Write delays into a *Cryptographic Core* can be neglected because packet processing starts as soon as one 128-bit word is available in the input FIFO. In contrast, read delays

cannot be neglected (see item 4 in Section 4.1). The following formulae allows computation of packet processing delays for a specific algorithm:

$$T_{total} = T_{loop} * packet\_size + T_{inv} + packet\_size/bus\_size \quad (1)$$

5 Task Allocation on MCCP

5.1 Channel Modelling

In order to evaluate MCCP performances, cryptographic channels have to be modelled. In general, packets pass through a channel in an aperiodic way. However, there is a fixed minimum delay between two packet arrivals due to the limited bandwidth of underlying physical channels. Therefore, packet processing tasks can be modelled using the *sporadic task model* [13]. Indeed, the processing of packets belonging to a channel C_i can be modelled as a sporadic task $\tau_i = (e_i, p_i, d_i)$ where:

- e_i is the processing delay of one packet belonging to C_i.
- p_i is the minimum delay between arrival of two packets belonging to C_i .
- d_i is the maximum allowed delay between the arrival of a packet and the end of its processing. If the end of a packet processing occurs after the deadline d_i then, it is said that the *deadline is missed*. Such parameter is really important for real-time communications where latency and jitter have to be reduced.

5.2 Hypothesis and Constraints

Current MCCP architecture does not enable to delay the processing of a specific packet. In consequence, no scheduling algorithm are involved in the task allocation process and packets are processed in their order of arrival. However, a scheduling algorithm could be implemented at the *main controller* side.

To remain as general as possible, this article propose to evaluate the performances of our MCCP processor under an unscheduled load. In the remainder of this article, we only consider the worst case where sporadic tasks always behave as periodic tasks.

In this work, a set of tasks τ is *allocable* under the previous hypothesis if and only if the processor load does not exceed the MCCP capacity. Such condition is equivalent to (2) for a MCCP processor which implements $m \geq 1$ cores and executes n concurrent channels.

$$\sum_{i=1}^{n} e_i/p_i \leq m \quad (2)$$

5.3 Task Allocation Algorithm

Current MCCP release implements a simple task allocation algorithm. In a first time, *Task Scheduler* waits for the end of communications through the *Cross Bar*. Then, when an incoming packet is available, it is sent to the first idle core. If no packet are available, *Task Scheduler* tries to retrieve data from any core with data pending. To conclude, if data are being transferred, algorithm goes to the first step else it goes to the second step. Further details on the assessment of performance of our algorithm are given in the following section.

It is noteworthy that this algorithm gives an higher priority to incoming packets in order to maximize the utilization of processors. Also, it does not try to allocate a specific channel to a core in order to save time needed by round key computations. In fact, keys are computed almost on the fly when the first 128-bit packet block is processed, hence, round key generation delays can be neglected. In addition, this algorithm gives the same priority to each channels. Currently, this algorithm does not support allocation of two cores per packet in the case of the CCM mode.

5.4 Performance Estimation of the Task Allocation Algorithm

Let studying the impact of n channels on the maximum *maintainable* throughput when our task allocation algorithm is used. For a fixed number of channels n and a fixed number of processors m, we have:

$$\alpha = \sum_{i=1}^{n} e_i / p_{max_i} \tag{3}$$

Where p_{max_i} is equal to the physical maximum throughput of the channel C_i. According to the condition (2), the set of channels C is *allocable* if $\alpha \leq m$. If it is not the case and if we assume that all packets have the same priority, then, a *allocable* set of tasks can be obtained by increasing each p_{max_i} by a factor α/m. Therefore, the actual *minimum arrival delay* of channel C_i must be equal to

$$p_i = \frac{\alpha}{m} * p_{max_i} \tag{4}$$

To conclude, maximum throughput of a *allocable* set of channels seems to be easily computable. Unfortunately, this analysis does not take into account I/O accesses over the MCCP *Cross Bar*. In fact, the delay needed to forward a packet to a processor is not null and MCCP *Cross Bar* remains locked while a packet forwarding is in progress. This phenomenon must be taken into account to obtain realistic throughput estimations.

Section 6.3 gives experimental results in order to estimate the effects of the *Cross Bar* congestion on the maximum *maintainable* throughput.

6 Results

6.1 Implementation Results

Proposed MCCP has been described with VHDL and synthesized using Xilinx ISE 12.1 tool. For MCCP hardware implementation, we use a Xilinx Virtex 4 SX35-11 FPGA. Implementation results are detailed in Table 1 (no DSP block is used).

Results show that resource utilization increases linearly with the number of implemented cores. In addition, maximum frequency remains stable no matter how many cores are implemented. The six-core MCCP version exhibits slightly lower performances than others implementations. But, small change in MCCP architecture may improve performance to reach 190 MHz. To conclude, MCCP architecture is well scalable.

Nevertheless, *Cross Bar* bandwidth may be a bottleneck for MCCP scalability. A 64-bit version of the *Cross Bar* has been implemented to prevent bus congestion when eight cores are used. Section 6.3 provides more details regarding the performances of each of these implementations.

Nb of cores	Bus size	Slices	BRAM	Max. Frequency (MHz)
2	32	2071	15	192
4	32	4055	26	192
6	32	6026	37	185
8	32	7760	48	192
8	64	8714	64	192

Table 1. MCCP Resource Utilization

6.2 Cryptographic Core Performances

This section details raw performances of MCCP cryptographic cores. Actual *Cryptographic Core* throughput depends on packet size (see Section 4.3), higher throughput are obtained from larger packets. Table 2 summarized these results. In addition, for each Table 2 column, the first number denotes the theoretical throughput calculated from loops computation time, while the second number corresponds to the processing time of a 2 KB packet measured from HDL simulations.

Table 2 shows that, in the case of AES-CCM, packet processing on one core is more efficient than packet processing on two cores. Indeed, $2\times$*1 core per packet* configuration provides a better throughput than *2 cores per packet* configuration. However, latency of the first solution is almost two times greater than latency of the second solution. As a consequence, designers should make scheduling choices according to system needs in terms of latency and/or throughput.

Key Size (bit)	AES-GCM (Mbps)	AES-CCM (Mbps)	
	1 core per packet	1 core per packet	2 cores per packet
128	496 / 437	233 / 214	442 / 393
192	426 / 382	202 / 187	386 / 348
256	374 / 337	178 / 171	342 / 313

Table 2. Cryptographic Core Throughputs at 190 MHz (Theoretical/2KB)

6.3 MCCP Performances

To estimate performances of our architecture under an heavy load, a *transaction level model* of the MCCP processor has been developed in Java. This high level model takes into account the packet processing delays and the I/O access delays to provide an accurate estimation of the *maintainable* throughput. In this section, it is assumed that:

- Packets are not scheduled (i.e. they arrive in a random order).
- Channels only uses CCM and GCM algorithms with 128-bit keys.
- Maximum throughput per channel is given by: $p_{max_i} = e_i$.

For each number of channels, we simulated all combinations of channel configuration and we picked the worst case to obtain the maximum *maintainable* throughput whatever the channel combination. Figure 3 shows the simulation results where α is the parameter defined in (3) and n the number of channels.

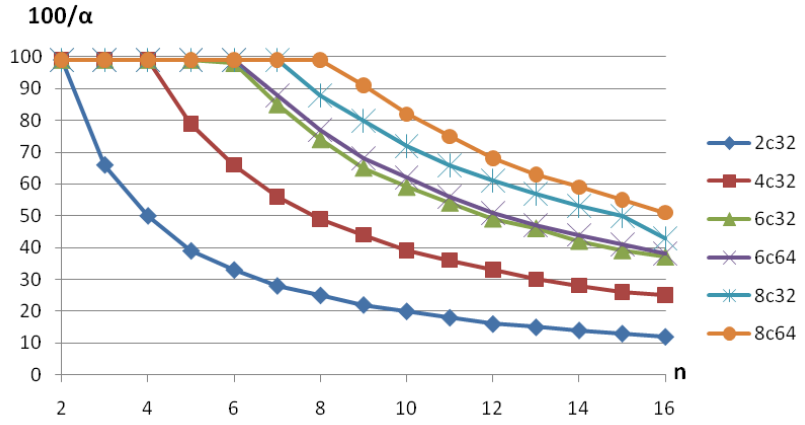

Fig. 3. $100/\alpha = f(n)$

Figure 3 shows that curves follow, as expected, an $1/x$ law. Also, we can see that bus size does not limit the throughput when less than eight cores are implemented. However, due to bus congestion, the 32-bit version of the eight core

MCCP provides worst performances (-8% on average) than the 64-bit version. This drawback will become a serious issue for MCCPs which implement an high number of cryptographic cores. In this case, others architectures like *Network on Chip* may be more a suited. To conclude, the maximum *maintainable* throughput on an eight cores 64-bit MCCP is around 3460 Mbps when just one channel is active.

7 Conclusion and Future Works

This work shows that the scalable multi-core architecture presented in this paper provides a good trade-off between flexibility, performances and resource consumption. Its 3.4 Gbps maximum throughput makes it particularly suitable for high throughput multi-channel communication systems. Proposed MCCP supports the commonly used CTR, CBC-MAC, CCM and GCM block cipher modes. Also, AES core may be easily replaced by any other 128-bit block cipher (such as Twofish) thanks to the FPGA hardware reconfiguration. To conclude, last section shows that the scalability of our architecture is limited by the *Cross Bar* congestion. In further works, the use of *Network on Chip* architectures may be a solution to overcome this problem.

References

1. Special publication 800-38a (2001), http://csrc.nist.gov/
2. Security supplement to the software communications architecture specification. April 30, 2004 (April 2004), http://sca.jpeojtrs.mil/
3. Special publication 800-38c (2004), http://csrc.nist.gov/
4. Baruah, S.: The non-preemptive scheduling of periodic tasks upon multiprocessors. Real-Time Systems 32, 9–20 (2006), http://dx.doi.org/10.1007/s11241-006-4961-9
5. Buchty, R., Heintze, N., Oliva, D.: Cryptonite - a programmable crypto processor architecture for high-bandwidth applications. In: Heidelberg, S.B.. (ed.) Organic and Pervasive Computing - ARCS 2004. Lecture Notes in Computer Science, vol. 2981/2004, pp. 184–198 (2004)
6. Chapman), X.K.: Picoblaze user resources, http://www.xilinx.com
7. Fronte, D., Perez, A., Payrat, E.: Celator: A multi-algorithm cryptographic co-processor. In: Proc. International Conference on Reconfigurable Computing and FPGAs ReConFig '08. pp. 438–443 (Dec 3–5, 2008)
8. Grand, M., Bossuet, L., Gogniat, G., Gal, B.L., Dallet, D.: A reconfigurable crypto sub system for the software communication architecture. In: Proceedings MILCOM2009 (2009)
9. Guan, N., Yi, W., Gu, Z., Deng, Q., Yu, G.: New schedulability test conditions for non-preemptive scheduling on multiprocessor platforms. In: Proc. Real-Time Systems Symp. pp. 137–146 (2008)
10. Hodjat, A., Verbauwhede, I.: Area-throughput trade-offs for fully pipelined 30 to 70 gbits/s aes processors. IEEE Transactions on Computers 55, 366–372 (2006)
11. Jeffay, K., Stanat, D.F., Martel, C.U.: On non-preemptive scheduling of period and sporadic tasks. In: Proc. Twelfth Real-Time Systems Symp. pp. 129–139 (1991)

12. Lemsitzer, S., Wolkerstorfer, J., Felber, N., Braendli, M.: Multi-gigabit gcm-aes architecture optimized for fpgas. In: CHES '07: Proceedings of the 9th international workshop on Cryptographic Hardware and Embedded Systems. pp. 227–238. Springer-Verlag, Berlin, Heidelberg (2007)
13. Mok, A.K.: Fundamental design problems of distributed systems for the hard-real-time environment. Tech. rep., Cambridge, MA, USA (1983)
14. Satoh, A., Takeshi, Aoki, T.: High-speed pipelined hardware architecture for galois counter mode. In: Heidelberg, S.B.. (ed.) Information Security. Lecture Notes in Computer Science, vol. 4779/2007, pp. 118–129 (2007)
15. Theodoropoulos, D., Siskos, A., Pnevmatikatos, D.: Ccproc: A custom vliw cryptography co-processor for symmetric-key ciphers. In: Becker, J., Woods, R., Athanas, P., Morgan, F. (eds.) Reconfigurable Computing: Architectures, Tools and Applications, Lecture Notes in Computer Science, vol. 5453, pp. 318–323. Springer Berlin / Heidelberg (2009)
16. Wang, S.: An Architecture for the AES-GCM Security Standard. Master's thesis, University of Waterloo (2006)
17. Wu, L., Weaver, C., Austin, T.: Cryptomaniac: a fast flexible architecture for secure communication. In: ISCA '01: Proceedings of the 28th annual international symposium on Computer architecture. pp. 110–119. ACM, New York, NY, USA (2001)