

Design of Transport Triggered Architecture Processors for Wireless Encryption

Panu Hämäläinen, Jari Heikkinen, Marko Hännikäinen, and Timo D. Hämäläinen
Tampere University of Technology / Institute of Digital and Computer Systems
P. O. Box 553, FI-33101 Tampere, Finland
{panu.hamalainen, jari.heikkinen, marko.hannikainen, timo.d.hamalainen}@tut.fi

Abstract

Transport Triggered Architecture (TTA) offers a cost-effective trade-off between the size and performance of ASICs and the programmability of general-purpose processors. In this paper TTA processors for the RC4 and AES encryption algorithms of the new IEEE 802.11i WLAN security standard are designed. Special operations efficiently supporting the ciphers are developed. The TTA design flow is utilized for finding configurations with the best performance-size ratios. The size of the configuration supporting both the algorithms is 69.4 k gates and the throughput 100 Mb/s for RC4 and 68.5 Mb/s for AES at 100 MHz in the 0.13 μm CMOS technology. Compared to commercial processors of the same wireless application domain, higher throughputs are achieved at significantly smaller area and lower clock speed, which also results in decreased energy consumption.

1. Introduction

The performance requirements for security implementations have significantly increased with the communication speeds of data networks. For example, virtual private networks require high-speed implementations in busy corporate firewalls. On the other hand, a need for low-cost and low-power designs has emerged as wireless technologies for embedded, battery-powered devices have been developed. By combining high performance and low power, the security processing requirements can be met with lower energy consumption, resulting in longer operating times.

Whereas Application Specific Integrated Circuits (ASIC) offer the highest performance at low energy and low cost (in high volumes), the design times are long and upgrading is time-consuming and expensive. The flawed IEEE 802.11 Wireless Local Area Network (WLAN) security is a good example of a product in which support for low-cost upgrading would have been beneficial. High performance with reconfigurability and short development

times are achieved with Field Programmable Gate Arrays (FPGA). FPGAs are well-suited, e.g., for network routers but unsuitable for low-cost and low-power embedded devices. The embedded solutions should combine the benefits of ASICs and FPGAs by enabling good performance with a certain level of reconfigurability.

In this work a processor architecture called Transport Triggered Architecture (TTA) [4] is scrutinized for finding high-performance and low-cost processor designs with efficient support for wireless encryption. TTA is fully configurable and supports Instruction Level Parallelism (ILP) and application-specific operations. The development tools offer semi-automatic flow for straightforward processor design and implementation. In addition to good encryption performance-cost ratios, the designed processors and their special operations are general-purpose, enabling running other applications in them. Modifications are possible even in fielded devices by software updates. Compared to coprocessor schemes, TTA processors can offer better performance since off-chip communication bottlenecks are avoided.

The studied encryption algorithms are RC4 [22] and Advanced Encryption Standard (AES) [1] of the new IEEE 802.11i WLAN security standard [14]. Whereas the AES-based design is completely new, RC4 is included for backward compatibility with fielded hardware. Thus, support for both the algorithms is required in new products. Since the work of the 802.11i group as well as AES are the bases for a large number of new technologies, the TTA designs of this paper can be efficiently utilized in many other embedded products. For example, the new IEEE 802.15.4 low-rate, low-power personal area network uses AES. Currently, the maximum radio transmission speed of 802.11 is 54 Mb/s.

Generally, ASICs and FPGAs are implementation technologies and TTA is a processor architecture, implementable both in ASIC and FPGA technologies. However, in this work ASIC refers to a dedicated hardware implementation of an encryption algorithm. An ASIC is fixed and can only be utilized to this single task. Similarly, FPGA refers to an FPGA implementation of a single

algorithm. A TTA implementation refers to a TTA processor including software describing an algorithm. The processor is implemented in an ASIC technology.

Several specialized reconfigurable hardware architectures have been proposed for symmetric-key cryptography. The designs in [7], [9], [24], and [25] produce excellent results in terms of throughput and reconfigurability but areas are large and the application domain the same as that of FPGAs. References [19] and [10] are likely to achieve good performance-area ratios but they do not report the costs. Area-efficient designs are presented in [26] and [16]. However, they lack the flexibility and/or the automatic design flow of TTA. Implementing AES in an architecture comparable to TTA is studied in [20]. In this work considerably better performance is achieved with small area.

2. TTA Overview

Opposed to traditional, operation-triggered processor architectures, in TTA [4] operations occur as side effects of data transports. The execution begins when data are written to the operand registers of a Function Unit (FU). Only a single instruction, *move*, is required for low-level programming. The mirrored paradigm enables applying new scheduling, bypassing, and resource allocation techniques in high-level language compilers.

The TTA central processing unit is organized as a set of FUs and Register Files (RF). The data transfers between the entities are performed by an interconnection network consisting of a variable number of buses and bus connections. A FU may be anything between a simple shifter and a large Arithmetic Logic Unit (ALU). At least one FU operates as a Load-Store Unit (LSU) handling memory accesses.

The number of FUs, FU operands and results, RFs, RF ports, buses, and bus connections can be changed unlimitedly. The changes enable variable amount of ILP. TTA also allows adding special, application-specific operations implemented in Special Function Units (SFU). Currently, bus widths 1 and 32 bits are supported and 32-bit ALU operations have been implemented in the standard set of the FU operations. The buses are only simple connections multiplexed with AND-OR networks. Thus, even a larger number of buses does not excessively increase the processor size.

The TTA development environment [18], the *MOVE framework*, provides semi-automatic design process and short design times for application specific processors. The *MOVE compiler* translates the ANSI C/C++ description of the application into sequential MOVE code. The compiler supports all the described modifications. The *MOVE scheduler* is used for scheduling the sequential code to a specific TTA configuration. The *MOVE simulator* is provided for verifying and evaluating both the sequential and parallel codes and for assisting the scheduler.

Finding the best suited TTA configuration for a given application by hand is a time-consuming and error-prone task. Thus, the framework contains the *design space explorer* for automatically testing the suitability of a large number of TTA configurations for the application. The explorer evaluates the execution time, size, and energy consumption¹ and outputs configurations with the best values in all three categories. For the evaluations it utilizes the *MOVE estimator* which computes the measures according to presynthesized TTA resources. The exploration is performed in two phases. Initially, the explorer is given a large configuration with excessive amount of resources. The first phase reduces the resources but maintains the interconnect fully connected. In the second phase the unnecessary connections are removed from the configuration chosen by the designer.

The VHDL description for a TTA configuration can be automatically generated using the *MOVE Processor Generator* (MPG). The designer only has to provide the VHDL descriptions of SFUs. The TTA processor can be synthesized with third party synthesis tools.

3. Wireless Encryption Algorithms

The encryption algorithms of the 802.11i WLAN security standard [14] scrutinized for TTA implementations are RC4 and AES. The standard requires only the forward functionality of AES for en/decryption. Thus, the invert cipher is not discussed in this paper. However, with small modifications the TTA designs can be tuned for efficiently supporting also the inverted AES.

3.1. RC4

RC4 [22] produces a pseudo-random byte stream and en/decryption is performed by XORing the stream and the data. The encryption key size is between 1 and 256 bytes. Processing consists of two phases, initialization and pseudo-random byte generation. The internal state of the cipher is maintained in a 256-byte arrays S . During the initialization S is first linearly filled with the values from 0 to 255. The encryption key is used for scrambling the array. The phase is required for every new key.

The pseudo code of RC4 en/decryption (random byte generation) is presented in Fig. 1. The process uses two indices, i and j , for modifying the contents of S . The data is input and output in the byte array $data$. The process runs until all the bytes of the input are processed. During the random byte generation two entries in S are read and their locations are swapped. The third entry $S[t]$ is the produced

¹ Currently, the evaluation of energy consumption is under development. Thus, the energy measures are not considered in this work.

```

rc4(data) {
  i = 0; j = 0; k = 0;
  while not end of data {
    i = (i + 1) mod 256;
    j = (j + S[i]) mod 256;
    swap(S[i], S[j]);
    t = (S[i] + S[j]) mod 256;
    data[k] = S[t] xor data[k];
    k = k + 1; } }

```

Figure 1. RC4 en/decryption.

pseudo-random byte. The operations involved are addition modulo 256, memory access (read and write), and XOR.

3.2. AES

The AES algorithm [1] is a symmetric cipher that encrypts data in 128-bit blocks. It supports key sizes of 128, 192, and 256 bits. AES consists of successive, similar iteration rounds. Depending on the key size, the number of the rounds is 10, 12, or 14. Each round mixes the data with a 128-bit *roundkey*, which is generated from the encryption key. Currently, the 128-bit-key version is generally considered to provide adequate security. Decryption requires inverting the iterations resulting in at least partly separate data path. Even though only the 128-bit-key version is studied in the paper, the designs support also the others. The 802.11i encryption mode requires two AES passes per block of data.

The AES round operations are presented in Fig. 2. The cipher maintains an internal, 4-by-4 matrix of bytes, called *state*, on which the operations are performed. Initially, *state* is filled with the input data block XORed with the encryption key. Each round, except the last one, contains operations called *SubBytes*, *ShiftRows*, *MixColumns*, and *AddRoundKey*. The last round bypasses *MixColumns*.

SubBytes is an invertible, nonlinear transformation. It uses 16 similar 256-byte substitution tables (*S-box*) for independently mapping each byte of *state* into another byte. *S-box* entries are generated by computing multiplicative inverses in *Galois Field* $GF(2^8)$ and applying an affine transformation. *SubBytes* can be implemented either by computing the substitution [21] or using table lookups [13]. *ShiftRows* is a cyclic left shift of the second, third, and fourth row of *state* by 1, 2, and 3 bytes, respectively. *MixColumns* performs a modular polynomial multiplication in $GF(2^8)$ on each column. Instead of computing, *SubBytes* and *MixColumns* can also be combined into four large tables, called *T-boxes*. Performing the operations requires table lookups and XORing. The size of a *T-box* is 1,024 bytes. During each round *AddRoundKey* performs XOR with *state* and the *roundkey*. *Roundkey* generation includes *S-box* substitutions, word rotations, and XOR operations performed on the encryption key.

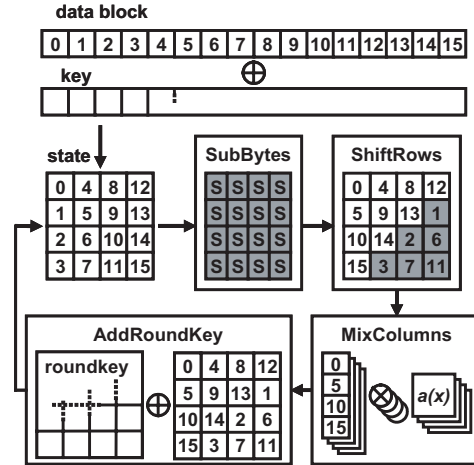


Figure 2. AES round operations.

4. Design Methodology

The design flow of the MOVE framework is utilized for developing TTA processors that efficiently support the two encryption algorithms. The analysis throughout the paper only concentrates on the encryption cores. Even though supported by the designed processors, the RC4 initialization and the AES roundkey generation are not examined. In order to estimate the possibilities for the utilization of ILP, the data dependencies of the algorithms are analyzed first. The parallelism analysis is valuable in estimating the amount of initial resources to be allocated for each algorithm and also for evaluating the results produced by the framework.

Next, the ciphers are scrutinized to identify operations that could be efficiently accelerated with SFUs. In this paper a requirement in the design of SFUs is that they can also be benefited from in other applications. A strictly application-specific SFU is added only if it has a significant effect on performance without a large processor area increase. For example, even though possible, implementing the complete AES cipher as an SFU was not considered. This finer-grained method produces a larger number of TTA configurations with different sizes and performances.

After the analyses the MOVE explorer is utilized for finding TTA configurations with the highest quality for the algorithms, with and without SFUs. The explorations are performed on the encryption cores. The *quality* of a design is defined as a throughput-size ratio

$$Q = T/S, \quad (1)$$

in which T is throughput in Mb/s and S is size in kgate. Hence, the unit of quality is Mb/s/kgate.

Even though tuning TTA for the encryption cores, the hardware is maintained general-purpose in order to allow running the rest of the cipher and other applications in it.

Only after finding the highest-quality configurations, they are further tuned to show how much the quality could be improved if the core was the only target application.

Two types of general-purpose, standard FUs are used in the designs, ALUs and LSUs. The ALU contains addition, subtraction, logical operations, comparisons, shifts, and zero/sign extensions. Multiplication and division are not included. The size of an ALU is 3,019 gates and latency one clock cycle. The size of a LSU size 1,140 is and read/write latency three clock cycles. A multiplier FU would add 3,120 gates and a divider 4,160 gates to the processor size. The TTA processor size and throughput values in the paper are for a 0.13 μm CMOS standard-cell technology, synthesized for a 100 MHz system clock. However, at least 250 MHz clock frequency is achievable using the TTA design flow [11]. The complete TTA processor size estimations do not include data or instruction memories. The term *estimated size* refers to the size computed by the MOVE estimator and *synthesized size* to the size produced by the Synopsys Design Vision 2003.06 synthesis tool. The results will show that the estimated size is very close to the synthesized size.

4.1. Comparison to Other Platforms

In order to evaluate the quality of the results, the TTA designs are compared to 8051, ARM7 and ARM9 processor as well as FPGA and ASIC implementations of the algorithms. 8051 is a 8-bit microcontroller and the ARMs are 32-bit Reduced Instruction Set Computer (RISC) processors utilized in embedded systems, especially in wireless platforms. Similarly to TTA, memories are not included in the size estimates. The 8051 features are derived from [8]. 100 MHz system clock speed is achievable with the synthesizable design in a 0.25 μm technology. Generally, 8051 chips are run at much lower speed, around 15 MHz.

The ARM size estimates are based on ARM7TDMI and ARM968E-S cores without caches [2]. According to [15], the gate count of a design is

$$g = \frac{A}{320F^2}, \quad (2)$$

in which A is the total area of the standard-cell layout and F is the minimum feature size of the technology. At $F = 0.13 \mu\text{m}$ the area $A = 0.26 \text{ mm}^2$ for ARM7 and $A = 0.59 \text{ mm}^2$ for ARM9. The clock speeds are 133 MHz and 220 MHz [2].

The RC4 performance in 8051 was estimated with the assembly implementation in [17]. The ARM performances were measured for the RC4 source code in [23], which is also used as the RC4 application in TTA. ARM Developer Suite 1.2 was utilized for producing speed-optimized measures. The RC4 implementation presented in [12] is used as the FPGA reference. The design was resynthesized to a newer Xilinx FPGA, XC2V250FG256-4. The equivalent

Platform	Size (kgates)	Clock (MHz)	RC4 (Mb/s)	AES (Mb/s)
8051	10.0	100	8.3	1.0
ARM7	48.1	133	36.7	14.2
ARM9	109.0	220	83.8	35.6
FPGA [12]	68.0	103	103.0	-
ASIC	3.0	200	200.0	-
FPGA [13]	670.0	134	-	1720.0
ASIC [21]	5.4	131	-	311.0

Table 1. Reference implementations.

gate count reported by the design tool is large due to the usage of embedded block RAMs for the S array. The VHDL design was also synthesized to the 0.13 μm technology at 200 MHz for producing an ASIC reference.

The 8051 performance for the 128-bit AES is derived from [5]. In [8] an instruction takes four oscillator cycles. The ARM performances were measured for the C source code in [6]. The source was also used as the AES application in TTA. The code is one of the fastest for the *gcc* compiler on which the MOVE compiler is also based. About 17% better ARM performance compared to the ARM9 measure of this paper is achieved in [3]. Unfortunately, the processor size is not reported for quality comparison.

The 128-bit-key encryption implementation containing the key generation logic in [13] is taken as the AES FPGA reference, resynthesized to the newer FPGA. Block RAMs were used for the S -boxes. The smallest AES implementation in [21] is used as the ASIC reference. The design contains encryption, decryption, and 128-bit key generation. According to the authors knowledge, the paper presents the highest-quality ASIC implementation. The reference implementations are summarized in Table 1.

5. TTA Designs for RC4

The operation dependencies in the random byte generation loop of RC4 are shown in Fig. 3. The operations presented in parallel are independent and can be executed simultaneously. The figure shows, as also stated in [23], that almost every statement of the algorithm depend on the statement immediately before it. Hence, the benefits of ILP for RC4 acceleration are very limited. Only the two writes of the swap operation and the computation of the index t can be performed in parallel.

More parallelism can only be found by unrolling the RC4 loop [23]. However, unrolling requires additional comparisons before the XOR operation (Fig. 1) if the byte length of the input does not always match the length of the produced pseudo-random stream. Thus, manual unrolling was not considered for RC4.

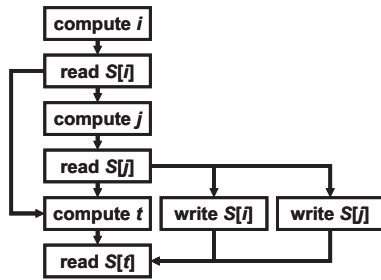


Figure 3. RC4 operation dependencies.

5.1. RC4 Special Operations

Three special operations were designed for RC4 acceleration. Since only 32-bit operations are currently implemented in the standard TTA hardware, a simple byte adder (i.e. modulo 256 adder) was implemented. A byte addition with a 32-bit adder requires masking the result with 0xFF or performing a zero extension, resulting in two clock cycles per addition in TTA. The single-cycle special operation *addb* was included in the ALU using the existing 32-bit adder hardware.

The latency of a memory access through the LSU is three clock cycles in TTA. Since a large portion of the statements in the RC4 loop requires accessing the *S* array, a decrease in the latency has a significant effect on the performance. A separate, single-cycle SFU (LUT-SFU) with read (*sread*) and write (*swrite*) operations was implemented for maintaining *S*. In addition to the short access latency, additional clock cycles are saved since the indices (*i*, *j*, *t*) can be directly used for referencing. Computing the absolute main memory addresses is avoided.

LUT-SFU contains a 256-byte single-port RAM with 8-bit address and data ports. The ports were wrapped inside a 32-bit interface in order to support connections to the 32-bit TTA buses. The size of the synthesized SFU is 2,012 gates. A large number of other encryption algorithms can make use of the SFU for look-up-table functionality [7]. Section 6 will utilize the SFU in the implementation of AES. Units for the same purposes have been designed in [7] and [16].

5.2. Design Space Explorations for RC4

Based on the parallelism analysis and manual experiments with few TTA configurations, it was deduced that two ALUs are sufficient for the RC4 initial configuration in the explorations. Similarly, the suitable amount for the other TTA resources was searched manually. A suitably-sized configuration results in shorter exploration time. The number of LSUs was also limited to two, since only single and dual-port memories were available in the used implementation technology. Also in general, two ports are of-

ten the maximum in memory technologies. The rest of the initial resources for the non-SFU configuration were three 16-register integer RFs, two 4-register boolean RFs, and six 32-bit buses. In the SFU configuration the same resources enhanced with the LUT-SFU were used.

5.3. RC4 Results

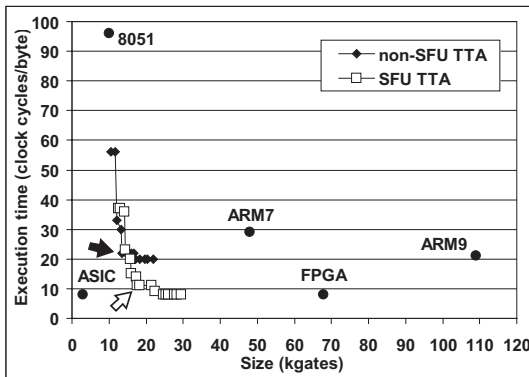
Fig. 4 presents the exploration results for the fully connected configurations output by the explorer. Due to the well-chosen initial configurations, the explorations took only about 20 minutes. The TTA throughputs are for 100 MHz. The arrows point to the TTA configurations with the highest qualities. The figure shows that after adding the special operations, the RC4 performance is significantly increased. In Fig. 4(a) the execution time of the fastest non-SFU configuration is 20 cycles/byte whereas for the fastest SFU configuration it is only 8 cycles/byte. In Fig. 4(b) the quality of a design increases the closer it is to the vertical axis and the further away it is from the horizontal axis. The characteristics of the highest quality configurations are presented in Table 2. The qualities are computed for the synthesized sizes. It can be seen that the estimator produces good results as the estimated sizes fairly precisely match with the synthesized ones. Note that also the RF sizes were reduced.

For comparison Fig. 4 includes RC4 measures for the reference implementations. TTA outperforms 8051 even with an equal-sized non-SFU configuration. Also, the throughput level of ARM7 is achieved at 70% smaller size. The throughput of ARM9 is higher due to the higher clock frequency. The ARM9 throughput is achieved at 100 MHz frequency with an 80% smaller SFU configuration. Even the cycle counts of FPGA and ASIC are reached in TTA with the special operations.

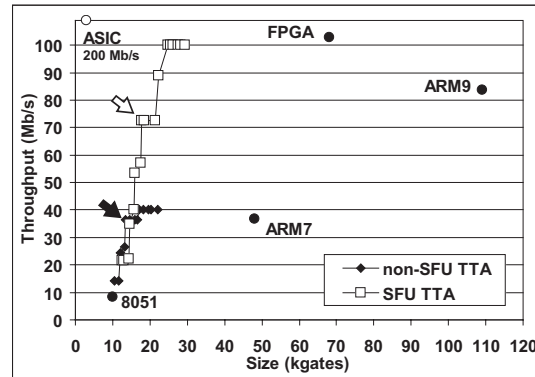
The comparison of the highest quality TTA configurations with the other platforms is presented in Fig. 5. The

Resource	non-SFU	SFU
ALUs	1	1
LSUs	1	1
integer RFs (4 registers)	3	3
boolean RFs (1 register)	1	1
32-bit buses	3	4
SFUs	-	1
Estimated size (kgates)	13.4	17.7
Synthesized size (kgates)	13.5	17.7
Throughput (Mb/s)	36.4	72.7
Quality (Mb/s/kgate)	2.70	4.11

Table 2. Non-SFU and SFU configurations with the highest quality for RC4.



(a)



(b)

Figure 4. RC4 exploration results in TTA: (a) execution time against size and (b) throughput against size. The arrows point to the configurations with the highest quality.

figure shows the results for the synthesized highest-quality TTA designs. The configurations *non-SFU-opt* and *SFU-opt* were generated by removing unused ALU operations from the non-SFU and SFU configurations and performing the second exploration phase. The sizes for these were taken from the estimator. In addition to the other processors, the quality of TTA for RC4 is higher than that of FPGA.

6. TTA Designs for AES

Compared to RC4, AES contains much more parallelism. In this paper the parallelism is divided into three levels: *block parallelism*, *round parallelism*, and *sub-round parallelism*. Block parallelism refers to processing several data blocks simultaneously. The applicability depends on the used encryption mode. For example, the inputs of the Counter (CTR) mode are independent from each other and can be processed in parallel. On the other hand, the Cipher

Block Chaining (CBC) mode requires that the previous data block is finished before processing the next one. The block parallelism is utilized, e.g., in pipelined hardware implementations.

Round parallelism exists if some of the cipher rounds are independent from each other. Those rounds could be processed simultaneously. However, the dataflow-oriented AES requires the output of the previous round before the next one begins and round parallelism cannot be utilized.

The parallelism inside a round, sub-round parallelism, can be examined using Fig. 2. In SubBytes each byte-wise S-box substitution can be performed simultaneously. Similarly, in AddRoundKey each *state* byte (even each *bit*) can be independently XORed with the roundkey bytes. MixColumns requires at least a column (a word) of *state* before processing can begin. Despite of the inapplicability of the round parallelism, the sub-round analysis reveals possibilities for interleaving consecutive rounds. As soon as a byte from the previous round is ready, the next S-box substitution for the byte can be performed. The same applies for AddRoundKey. MixColumns for a word can be performed when the four required bytes are produced. The method can also be utilized for interleaving at the data block level in feedback encryption modes.

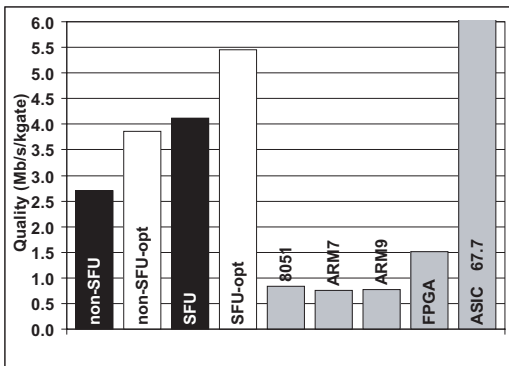


Figure 5. Qualities of RC4 implementations.

6.1. AES Special Operations

LUT-SFU designed for RC4 is also used for accelerating AES. In addition, two SFUs were designed, one for converting between byte and word representations (CONV-SFU) and another, AES-specific MIXADD-SFU. Processing an AES round with the special operations is depicted in Fig. 6.

LUT-SFU is included for fast look-up-table functionality. A single LUT-SFU carries out an 8-bit S-box substitution. Higher parallelism compared to a non-SFU design is possible since the number of simultaneous S-box accesses is not limited by LSUs. Initially, each LUT-SFU is filled with the S-box values from the main memory with the *write* operation, performed only once. Thereafter, the SFUs are used in read-only mode with the *sread* operation. The SFU can be utilized in the roundkey generation as well as performing the inverted AES operations.

Implementing AES in the 32-bit TTA requires maintaining *state* in four words containing the columns. A 32-bit MixColumns operation uses a byte from each row. The bytes are located in different columns (words) due to the ShiftRows operation. Similarly, the S-box (or T-box) lookups require referencing the table with bytes within the words. Thus, in order to generate the inputs for the operations, the bytes have to be shifted to suitable positions. The outputs are again packed to the words representing the columns of *state*. These conversions result in several shift and logic operations.

Another alternative is to maintain *state* in 16 words, each containing a byte of *state* aligned to the least significant byte of a word. With this scheme, ShiftRows and SubBytes can be implemented very efficiently as each byte can be accessed directly. However, the representation requires four times more moves for transporting *state* between FUs. In addition, it requires four XOR operations per 32 bits in AddRoundKey, totalling 16 XORs. When *state* is stored in four words, only four XOR operations are required.

CONV-SFU was designed for combining the benefits of both the representations. Its *bytes2word* operation can be

used for packing four bytes into a word and *word2bytes* for expanding a word into four bytes in a single clock cycle. CONV-SFU was wrapped inside a 32-bit interface for supporting connections to the TTA buses. The size of the SFU is 880 gates in the 0.13 μm technology. It can be used in the invert AES and for performing any byte-wise permutation in other block ciphers. Assembling and decomposing network packets often require similar transformations.

MIXADD-SFU performs a combined 32-bit MixColumns and AddRoundKey operation (*mixadd*) in a single clock cycle. MixColumns is trivial in hardware [1] and can be implemented with reasonably small amount of resources. On the contrary, in a 32-bit processor a large number of shift and logical operations are required [3]. Including the XORing of 32-bits of a roundkey in the unit saves four clock cycles per round compared to XORing in the ALU. The size of MIXADD-SFU is 1,326 gates.

6.2. Design Space Explorations for AES

In the AES source [6] the encryption core is fully unrolled. For shorter scheduling time the unrolling factor was decreased to four rounds. This still allows exploiting interleaving between AES rounds. Means for block level parallelism or block interleaving were not provided in this work. Similarly to RC4, the number of LSUs was limited to two. The number of other resources was increased for supporting the higher parallelism of AES. It was tested that eight ALUs already fully utilize the memory bandwidth in the non-SFU configuration. The number of ALUs was decreased to four in the SFU configuration since most of the processing is performed in the SFUs. The number of SFUs was adjusted to allow processing two words of *state* simultaneously, i.e., eight LUT-SFUs, four CONV-SFUs, and two MIXADD-SFUs were included. This was seen as a reasonable trade-off between available ILP and the maximum processor size. The rest of the resources were three 32-register integer RFs, two 4-register boolean RFs, and 16 32-bit buses.

In the original source code an AES round is implemented with the T-box method and *state* is stored in four words. Thus, the most utilized operations are memory accesses. Shifting and logical operations are required for indexing the T-boxes, permuting bytes, and combining the read values to produce round outputs. The performance is limited by the memory bandwidth. The finer-grained implementation with the SFUs provides more alternatives for the utilization of ILP. It requires much less memory bandwidth. The intermediate computation results can be kept in RFs or directly conveyed to the next FU. The SFU implementation also requires smaller amount of memory. 256-byte S-boxes are used instead of 1,024-byte T-boxes.

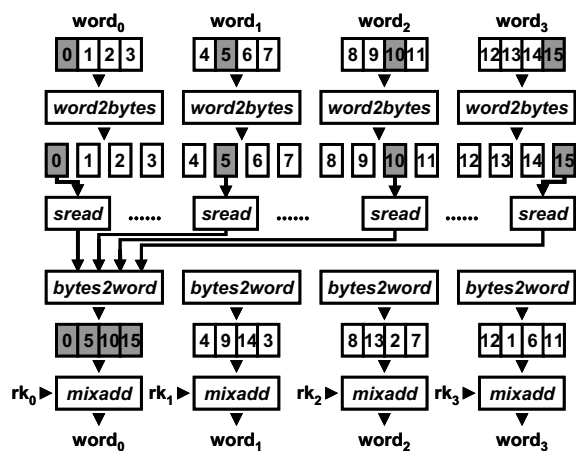


Figure 6. Using special operations for computing an AES round. The value rk_k is the k :th word of the current roundkey.

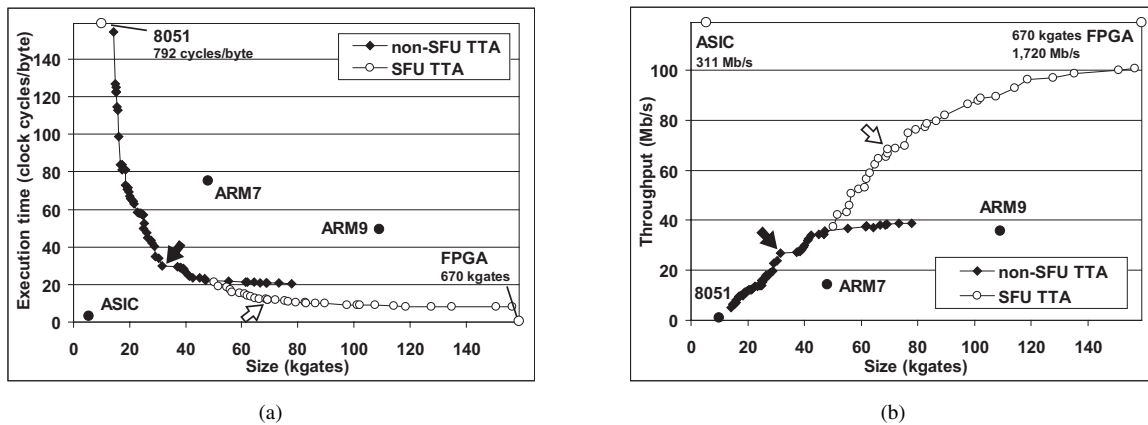


Figure 7. AES exploration results in TTA: (a) execution time against size and (b) throughput against size. The arrows point to the configurations with the highest quality.

6.3. AES Results

The AES exploration outputs for the fully connected configurations are presented in Fig. 7. The explorations took about six hours. The smallest non-SFU configuration is 72% smaller than the smallest SFU configuration. However, the smallest SFU configuration already achieves the performances of the fastest non-SFU configurations. On the other hand, the throughput of the largest SFU configuration is 160% higher than that of the largest non-SFU configuration. It was examined that the non-SFU performance saturates at 20 cycles/byte even if resources (other than LSUs) were added. It is restricted by the memory bandwidth. With the SFUs much higher performances are achievable.

Fig. 7 shows that the throughput of ARM7 is achieved at 48% smaller size with a non-SFU configuration. The throughput of ARM9 is also reachable with a non-SFU TTA at significantly smaller size. The smallest SFU configuration achieves the ARM9 throughput at the size of ARM7. The highest-quality TTA configurations are summarized in Table 3. The non-SFU configuration has 90% higher throughput at 35% smaller size than ARM7 and only 24% lower throughput than ARM9 at 71% smaller area. The size of the SFU configuration is between the ARMs. The throughput is significantly higher.

The dataflow-oriented AES algorithm can be implemented very efficiently in coarse-grained hardware. Therefore, the performances of the FPGA and ASIC designs are not achieved in TTA with the designed SFUs. Due to the fully utilized sub-round parallelism in the FPGA implementation, the throughput is very high, implying also good quality. The ASIC implementation computes 32-bits of a round at a clock cycle, resulting in a com-

pact and very high-quality design. The quality comparison similar to Section 5.3 is presented in Fig. 8.

As the highest-quality AES SFU configuration is a superset of the RC4 configurations, it is very well suited for the 802.11i processing. The RC4 throughput is 100 Mb/s which is enough for the highest transmission speeds. The rate of 54 Mb/s is achieved for the 802.11i AES mode by increasing the clock speed to 158 MHz.

7. Conclusions

TTA processors for efficient encryption in the 802.11i wireless network were designed. The designs outperform the 32-bit commercial processors of the same application

Resource	non-SFU	SFU
ALUs	2	2
LSUs	2	2
integer RFs (12 registers)	3	3
boolean RFs (1 register)	1	1
32-bit buses	6	9
LUT-SFUs	-	4
CONV-SFUs	-	1
MIXADD-SFUs	-	1
Estimated size (kgates)	31.5	71.1
Synthesized size (kgates)	32.9	70.4
Throughput (Mb/s)	27.0	68.5
Quality (Mb/s/kgate)	0.821	0.973

Table 3. Non-SFU and SFU configurations with the highest quality for AES.

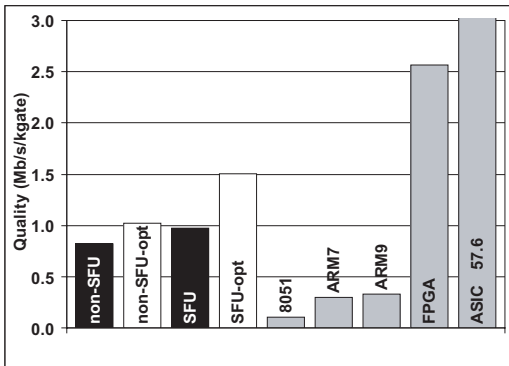


Figure 8. Qualities of AES implementations.

domain with significantly smaller area and lower clock frequency. For RC4 even the cycle counts of the ASIC were achieved. The AES performance can be increased closer to the dedicated hardware by adding resources and tuning the special operations more to the algorithm. In this work the die sizes were kept small and the designs general-purpose for low-cost and for supporting other network processor tasks. As supporting both the algorithms, the AES processor is very well suited for security processing in the embedded 802.11i devices. After the further development of the MOVE framework is finished, the energy consumptions of the designs will also be included in the evaluations.

Acknowledgments

Erno Salminen provided valuable assistance with the SFU design and the presentation of the results.

References

- [1] Advanced encryption standard (AES). FIPS-197, 2001.
- [2] ARM website. www.arm.com, 2005.
- [3] K. Atasu, L. Breveglieri, and M. Macchetti. Efficient AES implementations for ARM based platforms. In *Proc. 2004 ACM Symp. Applied Computing (SAC 2004)*, pages 841–845, Nicosia, Cyprus, Mar. 14–17, 2004.
- [4] H. Corporaal. *Microprocessor Architectures from VLIW to TTA*. Wiley & Sons, West Sussex, England, 1998.
- [5] J. Daemen and V. Rijmen. AES proposal: Rijndael, 1999.
- [6] C. Devine. AES code. www.cr0.net:8040/code/crypto, 2001.
- [7] A. J. Elbirt and C. Paar. Instruction-level distributed processing for symmetric-key cryptography. In *Proc. 17th IEEE Int. Symp. Parallel and Distributed Processing (IPDPS 2003)*, pages 78–87, Nice, France, Apr. 22–26, 2003.
- [8] Global UniChip Corporation, Taiwan. *UMCU-0051A-T 8-bit Microcontroller-Turbo*, 2001.
- [9] S. Hauck, T. W. Fry, M. M. Hosler, and J. P. Kao. The Chimera reconfigurable functional unit. *IEEE Trans. VLSI Systems*, 12(2):206–217, 2004.
- [10] J. R. Hauser and J. Wawrzynek. Garp: A MIPS processor with a reconfigurable coprocessor. In *Proc. 5th IEEE Symp. FPGA-Based Custom Computing Machines (FCCM'97)*, pages 24–33, Napa Valley, CA, USA, Apr. 16–18, 1997.
- [11] J. Heikkinen, J. Sertamo, T. Rautiainen, and J. Takala. Design of transport triggered architecture processor for discrete cosine transform. In *Proc. 15th Annu. Int. ASIC/SOC Conf.*, pages 87–91, Rochester, NY, USA, Sept. 25–28, 2002.
- [12] P. Hämäläinen, M. Hännikäinen, T. D. Hämäläinen, and J. Saarinen. Hardware implementation of the Improved WEP and RC4 encryption algorithms for wireless terminals. In *Proc. EUSIPCO 2000*, volume 4, pages 2289–2292, Tampere, Finland, Sept. 5–8, 2000.
- [13] P. Hämäläinen, M. Hännikäinen, M. Niemi, T. D. Hämäläinen, and J. Saarinen. Implementation of link security for wireless local area networks. In *Proc. IEEE Int. Conf. on Telecommunications (ICT 2001)*, volume 1, pages 299–305, Bucharest, Romania, June 4–8, 2001.
- [14] Wireless LAN medium access control (MAC) and physical layer (PHY) specifications: Medium access control (MAC) security enhancements. IEEE Std. 802.11i, 2004.
- [15] International technology roadmap for semiconductors: System drivers. Technical report, Sematech, 2003.
- [16] M. Lewis and S. Simmons. A VLSI implementation of a cryptographic processor. In *Proc. Canadian Conf. Electrical and Computer Engineering (CCECE 2003)*, pages 821–826, Montreal, Canada, May 4–7, 2003.
- [17] S. Ljungkvist. RC4 assembly source. www.8052.com/codelib/ArcFour.asm, 2003.
- [18] MOVE project website. www.tkt.cs.tut.fi/~move, 2005.
- [19] D. Olivia, R. Buchty, and N. Heintze. AES and the Cryptonite crypto processor. In *Proc. CASES 2003*, pages 198–209, San Jose, CA, USA, Oct. 30–Nov. 1, 2003.
- [20] S. Ravi, A. Raghunathan, N. Potlapally, and M. Sankaradass. System design methodologies for a wireless security processing platform. In *Proc. 39th Design Automation Conf.*, pages 777–782, New Orleans, LA, USA, June 10–14, 2002.
- [21] A. Satoh, S. Morioka, K. Takano, and S. Munetoh. A compact Rijndael hardware architecture with S-box optimization AES design. volume 2248 of *Lecture Notes in Computer Science*, pages 239–254, Germany, 2001. Springer-Verlag.
- [22] B. Schneier. *Applied Cryptography: Protocols, Algorithms and Source Code in C*. Wiley & Sons, USA, 2 edition, 1996.
- [23] B. Schneier and D. Whiting. Fast software encryption: Designing encryption for optimal software speed on the Intel Pentium processor. In *Proc. Fast Software Encryption Workshop 1997*, pages 242–259, Haifa, Israel, Jan. 20–22, 1997.
- [24] H. Singh, M.-H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. C. Filho. MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications. *IEEE Trans. Computers*, 45(5):465–481, 2000.
- [25] R. R. Taylor and S. C. Goldstein. A high-performance flexible architecture for cryptography. volume 1717 of *Lecture Notes in Computer Science*, pages 231–245. Springer, 1999.
- [26] L. Wu, C. Weaver, and T. Austin. CryptoManiac: A fast flexible architecture for secure communication. In *Proc. 28th Annu. Int. Symp. Computer Architecture (ISCA 2001)*, pages 110–119, Göteborg, Sweden, June 30–July 4, 2001.