Rethinking Secure FPGAs: Towards a Cryptography-friendly Configurable Cell Architecture and its Automated Design Flow

Nele Mentens [1], Edoardo Charbon [2] and Francesco Regazzoni [3]
[1]imec-COSIC – KU Leuven, Belgium, nele.mentens@kuleuven.be
[2]AQUA – EPFL, Switzerland, edoardo.charbon@epfl.ch
[3]ALaRI – USI, Switzerland, regazzoni@alari.ch

Abstract

This work proposes the first fine-grained configurable cell array specifically tailored for cryptographic implementations. The proposed architecture can be added to future FPGAs as an application-specific configurable building block, or to an ASIC as an embedded FPGA (eFPGA). The goal is to map cryptographic ciphers on combinatorial cells that are more efficient than general purpose lookup tables in terms of silicon area, configuration memory and combinatorial delay. As a first step in this research direction, we focus on block ciphers and we derive the most suitable cell structure for mapping state-of-the-art algorithms. We develop the related automated design flow, exploiting the synthesis capabilities of Synopsys Design Compiler and the routing capabilities of Xilinx ISE. Our solution is the first cryptography-oriented fine-grained architecture that can be configured using common hardware description languages. We evaluate the performance of our solution by mapping a number of well-known block ciphers onto our new cells. The obtained results show that our proposed architecture drastically outperforms commercial FPGAs in terms of silicon area and configuration memory resources, while obtaining a similar throughput.

1 Introduction

The capability of changing, at least to some extent, or updating the functionality of an electronic system after its deployment has always been desirable. In a typical system composed of hardware and software, such capability is usually guaranteed by software routines. Software, however, despite being extremely flexible, is much slower than its hardware counterpart (sometimes too slow to

[1]This work has been submitted to the IEEE for possible publication. Copyright may be transferred without notice, after which this version may no longer be accessible.

meet the requirements of the target application). FPGAs have been proposed as a solution to achieve a performance comparable to a dedicated hardware implementation while maintaining the possibility of being updated and reconfigured in the field.

The first FPGAs consisted of only lookup tables (LUTs) which were programmed by means of a configuration file, generated according to the function to be implemented. Their use, at that time, was mainly for prototyping and testing designs before ASIC fabrication. Soon, however, FPGAs also started to be used as general purpose hardware platforms, since they were extremely suitable for addressing the need of low-volume markets, reducing non-recurring engineering costs and allowing the user to access the latest technological nodes at a fraction of the ASIC cost. With the growth of the use of FPGAs as general purpose platforms came the need of having less generic reconfigurable hardware blocks, still capable to implement any design, but including specialized blocks for implementing recurring and relevant functions. As a result, FPGAs started on the one hand to include fast carry chains for arithmetic operations, Digital Signal Processing (DSP) blocks for signal processing and even more complex blocks, such as whole processors. On the other hand, the basic configurable cells evolved to become more and more efficient. This trend of improving the basic cells while extending the capacity of the specialized cells is certainly going to continue in future.

Cryptography is one of the main applications that are often deployed on FPGAs. Cryptographic primitives, such as block ciphers, public-key algorithms, and hash functions have been successfully implemented as stand-alone designs or as part of a complete system-on-chip. Further, dedicated circuits implementing physical(ly) unclonable functions (PUFs) or bitstream decryption blocks have been added to FPGAs by the vendors. Finally, with the advent of side-channel attacks, FPGAs are an attractive platform for implementing protected designs as well as for benchmarking the resistance against power analysis attacks.

However, surprisingly, despite such a massive use of reconfigurable hardware for cryptography, to date, the possibility of designing a cryptography-friendly, fine-grained reconfigurable cell has rarely been considered and certainly not explored yet in the right depth. In this paper, envisioning that the next application-specific block included on FPGAs will be devoted to cryptography, we design a new reconfigurable cell, conceived specifically for implementing cryptographic algorithms in an efficient way. As a first step in this direction [1], we consider block ciphers, covering all the possible constructions (SPN, ARX, Feistel and stream-cipher-like ciphers), and side-channel protecting threshold implementations of block ciphers [2]. We expect that authenticated encryption algorithms, hash functions and public-key algorithms based on binary field arithmetic can be easily mapped onto our new architecture as well, since they leverage atomic operations that are similar to the ones we consider for the construction of our configurable cell. We do not optimize our cell for public-key algorithms based on prime fields, since these can already be efficiently implemented using the DSP blocks in FPGAs [3, 4].

Our new cell, which we call cFA, is a configurable full-adder-based cell with six inputs, two outputs, and four configuration bits for programming the functionality. Our cell can be configured to implement up to eight basic arithmetic logic functions. cFA cells are combined with flipflops into cFA slices that have a structure that is similar to Xilinx slices, allowing us to use the routing capabilities of Xilinx tools.

We also propose a tool chain that maps any design, written in a hardware description language (HDL), to our novel fine-grained reconfigurable architecture. Our approach is oriented towards a maximal re-use of existing synthesis and place & route tools, such that we can benefit from the decades of experience of large EDA companies. In particular, our tool flow builds on Synopsys Design Compiler for synthesis and on Xilinx ISE for placement and routing.

We believe that cryptography is the next application that will be considered by FPGA designers, observing what happened in processor designs, where, after the basic instructions, designers added in sequence instructions for arithmetic operations (which have been already added to FPGAs), instructions for signal processing (which have been already added to FPGAs), and instructions for cryptography (which are not added to FPGAs yet). Our solution can be added as a small, crypto-friendly reconfigurable hardware block to be included as a new type of cell, together with other reconfigurable cells, in the next generation of FPGAs. Another application scenario uses our cFA cell in a small embedded FPGA (eFPGA) to be added to an ASIC design or a microprocessor (the interest in this direction is proven by the recent acquisition of Altera by Intel). Finally, reconfigurability will guarantee so-called cryptographic agility, allowing cryptographic algorithms to be upgraded or updated depending on newly detected vulnerabilities or changing standards. This is a fundamental requirement for current and future secure IoT devices and cyber-physical systems.

2 Related Work

The most closely related work in the direction of configurable cell architectures supporting cryptography is presented by Elbirt and Paar in [5]. They propose the Cryptographic (Optimized for Block Ciphers) Reconfigurable Architecture (COBRA), which is a coarse-grained architecture, consisting of configurable cells with 32-bit buses. The cells contain bit-wise XOR, AND and OR gates, adders/subtracters, 4-to-4-bit and 8-to-8-bit LUTs, modulo multipliers/squarers, shift/rotate blocks and GF(2^8) constant multipliers. The tool flow consists of an assembler that operates via a Very Long Instruction Word (VLIW) format. Therefore, mapping a cryptographic algorithm onto the COBRA architecture requires a COBRA-specific assembly-code program. The performance of RC6, Rijndael and Serpent is evaluated on the COBRA architecture, implemented in the ADK TSMC 0.35 micron library. The results show that COBRA outperforms microprocessor architectures, but leads to an inferior performance in terms of throughput and area compared to an FPGA architecture fabricated in a comparable technology.

Also related is the work of Taylor and Goldstein [6], proposing PipeRench, a coarse-grained reconfigurable architecture which consists of parallel stripes of processing elements with pipelining registers in between. The architecture is implemented in a 0.25 micron technology. The authors evaluate a number of block ciphers, namely Crypton, IDEA, RC6, Twofish and various AES candidates. Speedups of a factor 2 to 12 are reported over conventional processors. A comparison to FPGA architectures is not carried out. The PipeRench architecture can be configured using a dedicated compiler that takes a specific dataflow intermediate language (DIL) as an input [7].

In comparison to COBRA and PipeRench, our cFA-based architecture is extremely fine-grained. It supports any hardware design described in an HDL, significantly extending the design space and thus allowing to achieve better results in terms of area, throughput and latency. Furthermore, the design flow of our cell leverages state-of-the-art design commodities (Synopsys Design Compiler for synthesis and Xilinx ISE for placement and routing), with the twofold advantage of not requiring novel training for designers and of benefiting from the decades of experience of EDA companies and, automatically, from future improvements of the used underlying tools. To the best of our knowledge, our proposal is the first reconfigurable architecture tailored to cryptography which uses a fine-grained approach and the first one which exploits standard HDL languages and EDA commodities for the design flow.

Our work also touches the research area of designing embedded application-specific processors for cryptography. One example is the SPARX processor, proposed by Bache et al., that efficiently implements threshold-protected ARX-based ciphers [8]. The architecture we propose, allows to efficiently realize threshold implementations of many more types of block cipher structures, thus covering a much wider range of algorithms compared to the SPARX processor. A flexible cryptographic engine for FPGAs has also been presented by Gulcan et al. [9]. It is based on the block cipher Simon and is capable of performing pseudo-random number generation, hashing and variable-key symmetric encryption. Their architecture, however, offers capabilities for implementing only one cipher. Our architecture instead allows to implement all the recently proposed block ciphers, and it is quite likely that it will also be suitable for future block cipher designs, since the trend followed in the last decades suggests that, also in the future, cryptographic algorithms will be based on the operations and structures well supported by our architecture.

3 The New Configurable Cell Architecture

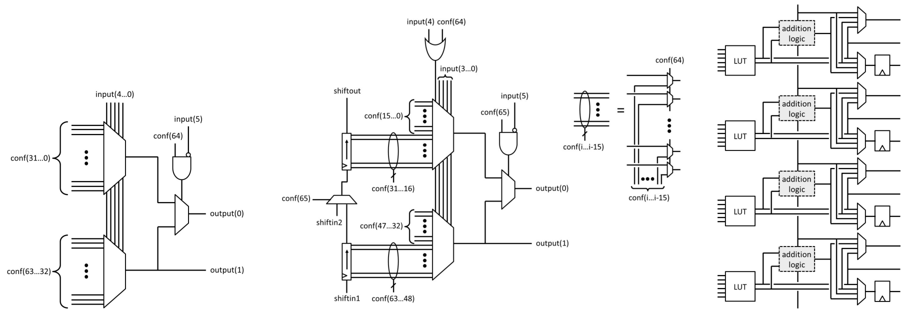
3.1 Comparison Basis

There are two ways for comparing in a fair way our new configurable cell architecture with existing FPGAs. The most realistic approach would require the same silicon technology used by commercial FPGAs for implementing an optimized custom design of our configurable cell and for comparing the obtained results with the performance of commercial FPGAs. However, accessing ex-

Table 1: Post-synthesis area and combinatorial delay of the re-implemented configurable cells in recent Xilinx FPGAs, synthesized using the NanGate 45nm standard cell library.

	SLICEL	SLICEM
Area (μm^2)		
	443.2	1438.3
Combinatorial delay (ns)		
LUT-in to slice-out	0.55	0.64
LUT-in to C_{out}	0.48	0.57
C_{in} to C_{out}	0.07	0.07
C_{in} to S	0.07	0.07

actly the same technology is very difficult. The second approach consists in re-implementing the configurable cells of recent FPGAs with an easily accessible library, and using the same technology for implementing our configurable cell. We followed the second approach. As a reference, we selected the Xilinx cells described in [10], which we implemented as depicted in Fig. 1. We synthesized the architectures of Fig. 1 using Synopsys Design Compiler and the open source NanGate 45nm standard cell library [11] to allow full reproducibility of our results. Table 1 reports the pre-layout area and combinatorial delay of the slices.



(a) LUT inside SLICEL

(b) LUT inside SLICEM

(c) simplified SLICEL/SLICEM structure

Figure 1: Architectures showing the way we re-implemented the LUTs and slices in recent Xilinx FPGAs.

3.2 From Cryptography to a New Configurable Cell

Our main design goal is to improve the efficiency of cryptographic algorithms while supporting cryptographic agility. Ideally, our configurable cell should be smaller and faster than a LUT and should use less configuration bits. Further, our cell should lead to an architecture that allows an efficient mapping of existing and future cryptographic algorithms.

We focus, at first instance, on block ciphers that have been proposed in the past decades. An overview of existing (lightweight) block ciphers is given in [12]. Most of these block ciphers can be categorized into SPN-based ciphers (SPN = substitution-permutation network), ARX-based ciphers (ARX = addition, rotation and XOR), stream-cipher-like ciphers and Feistel-based ciphers. The most frequently occurring operations in these ciphers are

1. bit permutation,
2. rotation,
3. addition modulo 2^n (in ARX-based ciphers),
4. addition modulo 2, i.e. exclusive OR (XOR),
5. substitution box (S-box).

In hardware, the first two operations are implemented through routing, while the last three operations require combinatorial logic.

Further, it is important to take into account side-channel attacks [13], in which secret information is extracted through side-channels, such as the power consumption, the electromagnetic radiation or the timing behavior of the chip. Threshold implementations, as proposed by Nikova et al. in [2], provide a provably secure way to protect a circuit against Differential Power Analysis (DPA) attacks of a specific order. In a threshold implementation, the linear parts of a cipher are repeated according to the number of shares. The non-linear parts (mostly realized as substitution boxes) are preferably expressed in terms of quadratic functions with pipelining registers in between, in order to minimize the number of required shares. A large number of examples are given by Bilgin et al. in [14]. Taking threshold implementations into account, we add the following (sixth) item to the list of commonly used operations in block ciphers:

6. quadratic functions (for the construction of threshold implementations of substitution boxes).

Analyzing the logic we need for the implementation of the listed operations, we notice that operations 4-6 can be expressed in terms of quadratic functions. As an example, we give the algebraic normal form (ANF) of the function $f : GF(2)^4 \rightarrow GF(2)$:

$$\begin{aligned} f(x,y,z,w) = {} & a_0 \oplus a_1 x \oplus a_2 y \oplus a_3 z \oplus a_4 w \\ & \oplus a_{12}xy \oplus a_{13}xz \oplus a_{14}xw \oplus a_{23}yz \oplus a_{24}yw \oplus a_{34}zw, \end{aligned} \tag{1}$$

in which the inputs x, y, z and w as well as the coefficients a_i and a_{ij} are elements of $GF(2)$, taking two possible values $0, 1$. Both the additions (denoted by $\oplus$) and the multiplications in the equation are in $GF(2)$, i.e. the addition is an XOR and the multiplication is a logical AND.

Operation 3 in the list is the addition of two n-bit numbers, in which the $(n+1)$-th bit of the sum is omitted. The straightforward way of implementing an addition, is through a ripple-carry adder, consisting of a sequence of full adders. A full adder has three inputs (A, B and C_{in}) and computes a sum output (S) and a carry output (C_{out}) as follows:

$$\begin{aligned} S &= A \oplus B \oplus C_{in}, \\ C_{out} &= AB + (A+B)C_{in}. \end{aligned} \tag{2}$$

Here, the $+$ operator denotes a logical OR.

We can reduce our search for an adequate configurable cell to the search of a cell that efficiently implements Eqs. (1) and (2). However, the carry computation in Eq. (1) can be rewritten as a quadratic function in ANF as follows:

$$C_{out} = AB \oplus BC_{in} \oplus AC_{in} \tag{3}$$

which clearly shows that all terms in Eq. (1) can be generated by the sum and carry circuits in full adders, except for the constant term a_0. Therefore, we decide to use the full adder as a basis for our new configurable cell.

3.3 Optimization of the New Configurable Cell

3.3.1 First Version of the Cell:

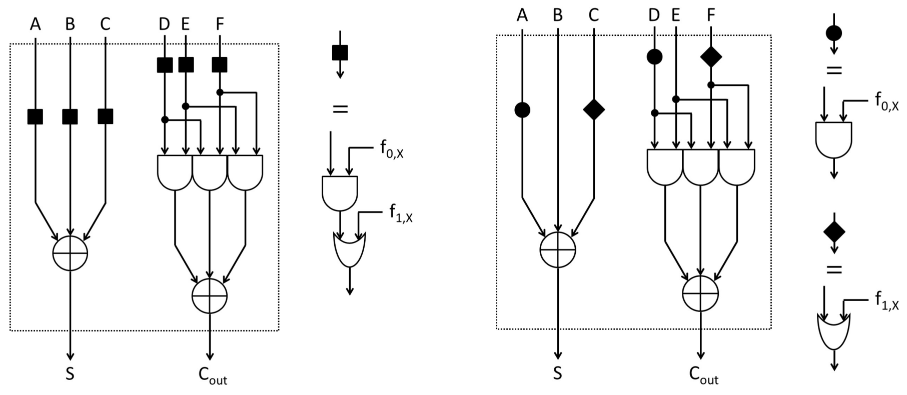
The first version of our configurable full-adder-based cell (cFA) is depicted in Fig. 2a. It consists of a sum circuit, computing the sum (S) of the first three input bits (A, B and C), and a carry circuit, computing the carry-out (C_{out}) of the other three input bits (D, E and F). For each input bit, two configuration bits ($f_{0,X}$ and $f_{1,X}$, with $X = A, B, C, D, E, F$) determine whether the bit is fed through or absorbed, such that a 0 or a 1 is applied to the circuit. This results in a cell with 12 configuration bits. The sum circuit can be configured to 3^3 functions:

$$S = (f_{1,A} + f_{0,A}A) \oplus (f_{1,B} + f_{0,B}B) \oplus (f_{1,C} + f_{0,C}C). \tag{4}$$

The carry-out circuit can as well be configured to 3^3 functions:

$$\begin{aligned} C_{out} = &(f_{1,D} + f_{0,D}D)(f_{1,E} + f_{0,E}E) \\ &\oplus (f_{1,D} + f_{0,D}D)(f_{1,F} + f_{0,F}F) \\ &\oplus (f_{1,E} + f_{0,E}E)(f_{1,F} + f_{0,F}F). \end{aligned} \tag{5}$$

Post-synthesis results for the first version of the cFA cell are given in the second column of Table 2.



(a) First version of the cell (b) Second version of the cell

Figure 2: Details of the proposed full-adder-based configurable cells (cFAs).

Table 2: Post-synthesis area and combinatorial delay of the two versions of our configurable cell (cFA), synthesized using the NanGate 45nm standard cell library.

	version 1	version 2
Area (μm^2)		
S circuit	6.384	4.788
$Cout$ circuit	5.054	3.990
total	11.438	8.778
Combinatorial delay (ns)		
input to S	0.18	0.16
input to $Cout$	0.16	0.10

3.3.2 Second Version of the Cell:

In the second version of the cell we further optimize the area, the combinatorial delay and the number of configuration bits. Reducing the number of configuration bits can be achieved by observing that, in the first version of the cell, several combinations of the configuration bits lead to the same function, because the cell is symmetric in both the sum and the carry-out computation. Therefore, it is not necessary to foresee both an AND and an OR gate for each input bit. Providing one input with an AND gate and another one with an OR gate for both the sum and the carry-out circuits leads to a reduction of the number of configuration bits as well as a reduction in the logical delay and the area of the cell. This way, the number of configuration bits are reduced from 12 to 4. This results in the second version of our cFA, which is shown in Fig. 2b. The eight functions that can be obtained, are given in Table 3.
Although the second version of the cFA has a slightly more limited functionality than the first, the post-synthesis results, given in Table 2, show a clear

Table 3: Functionality of the second version of the cFA cell, determined by the configuration bits, in which $\overline{X}$, XY, $X+Y$ and $X \oplus Y$ denote an inversion, a logical AND, a logical OR and an XOR, respectively.

$f_{0,A}$	$f_{1,C}$	S
0	0	$0 \oplus B \oplus C$
0	1	$0 \oplus B \oplus 1 = \overline{B}$
1	0	$A \oplus B \oplus C$
1	1	$A \oplus B \oplus 1 = \overline{A \oplus B}$
$f_{0,D}$	$f_{1,F}$	C_{out}
0	0	$0 \oplus 0 \oplus EF = EF$
0	1	$0 \oplus 0 \oplus E = E$
1	0	$DE \oplus DF \oplus EF = DE + (D+E)F$
1	1	$DE \oplus D \oplus E = D + E$

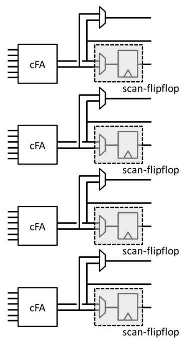


Figure 3: Architecture of a cFA slice, combining four cFA cells and four flipflops.

advantage of the second version over the first in terms of area and combinatorial delay. Therefore, we choose the second cFA (depicted in Fig. 2b) in our further experiments.

3.4 Merging cFA Cells into cFA Slices

In order to be able to re-use the routing capabilities of commercial FPGA design tools, we integrate cFA cells into a cFA slice in combination with flipflops and multiplexers. The resulting slice is shown in Fig. 3. Each cFA cell has an accompanying flipflop, that can be connected to either the S or the C_{out} output of the cFA cell. The combination of the multiplexer with the flipflop is implemented as a scan-flipflop (used as a regular internally used standard cell). A cFA slice has four configuration bits for each cFA and one configuration bit for each multiplexer, which results in 24 configuration bits per slice. The area and combinatorial delays of the slice are given in Table 4.

Table 4: Post-synthesis area and combinatorial delay of the cFA slice, synthesized using the NanGate 45nm standard cell library.

	cFA slice
Area (μm^2)	
	44.688
Combinatorial delay (ns)	
cFA-in to mux-out	0.23
cFA-in to direct-out	0.10

4 The Tailored Tool Flow

The tool flow that we developed to automatically map HDL designs onto an array of cFAs, is depicted in Fig. 4 and consists of three steps:

1. modify the HDL description such that all S-boxes are a composition of quadratic functions,
2. synthesize the resulting HDL design into a netlist that consists of standard cells from a tailored library,
3. translate the netlist into a netlist consisting of cFA slices and generate configuration data.

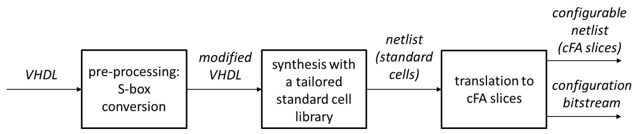

VHDL
pre-processing: S-box conversion
modified VHDL
synthesis with a tailored standard cell library
netlist (standard cells)
translation to cFA slices
configurable netlist (cFA slices)
configuration bitstream

Figure 4: The proposed tool flow, tailored to the cFA architecture.

4.1 Step 1

Since the cFA cell is especially suitable for the implementation of quadratic functions, the first step in the proposed tool flow translates all S-boxes into HDL descriptions consisting of quadratic functions. This also holds for threshold implementations, in which pipelining registers are needed in between quadratic functions. Pipelining registers bound the propagation of glitches that could contain exploitable side-channel information and, consequently, reduce the number of required shares, as explained in [2]. In ARX-based designs, no pre-processing is needed, since the non-linear operation, i.e. the addition modulo 2^n, will automatically be translated into a ripple-carry adder (consisting of full adders) in Step 2. For threshold implementations of the modulo 2^n adder, we follow the approach of Schneider et al. in [15]. Note that our goal is not to automatically translate non-protected designs into threshold-protected designs; Step 1 only concentrates on the S-box in the cipher. It is the task of the designer to provide an HDL description of the hardware design.

4.2 Step 2

In the synthesis step, we want to map the design onto an array of cFAs using Synopsys Design Compiler. To do this, we start from the functions listed in Table 3. Of these eight functions, six are directly implemented by standard cells in the NanGate 45nm library. Only the $A \oplus B \oplus C$ and $DE + (D + E)F = DE \oplus DF \oplus EF$ functions are not present in the NanGate 45 nm library. We therefore add two gates with the given functionality to the library, and we remove all gates that are not in Table 3, except for the full adder gate and the D-flipflop with asynchronous set and reset. Since the eight functions in the table as well as the full adder will eventually be mapped onto the cFA gates, they will all have the same area and delay in the resulting configurable array. Therefore, we modify the area and the delay of these gates in the library according to the values given in Table 2 for the second version of the cFA.

4.3 Step 3

The outcome of Step 2 is a netlist containing the eight gates in Table 3, full adder gates and D-flipflops with asynchronous set and reset. Since the four functions in the top part of Table 3 are independent of the four functions in the bottom part of the table, it is straightforward to merge any top-part function with any bottom-part function into one cFA. However, inside a cFA slice (as shown in Fig. 3), only one of the cFA outputs can be connected to a flipflop, which is taken into account during the merge. The 24 configuration bits for each cFA slice are combined into a configuration bitstream. This way, the output of Step 3 is a configurable netlist, i.e. a netlist consisting of only cFA slices, and a configuration bitstream.

4.4 From a netlist of cFA slices to a placed and routed design

Since our cFA slice has an interface that is similar to the interface of a Xilinx slice, the Xilinx tools for placement and routing can be re-used to transform our netlist of cFA slices into a placed and routed design. Therefore, we can evaluate the performance of our cFA architecture by mapping a hardware design to both our cFA architecture and a Xilinx FPGA, comparing the resources and delay of the slices only, excluding routing.

5 Experimental setup and results

In this section, we validate the performance of our new configurable cell and the suitability of the related design flow by mapping several block ciphers on our

Table 5: Properties of the evaluated block ciphers.

cipher	structure	size		remarks
		key	block	
AES-128	SPN	128	128	NIST standard
PRESENT-80	SPN	80	64	ISO/IEC standard
SPECK-128/128	ARX	128	128	proposed by the NSA
NOEKEON	SPN	128	128	direct-key mode
KTANTAN-64	stream	80	64	direct-key mode

Table 6: Operations in the evaluated block ciphers, in which (N)LFSR denotes a (non-)linear feedback shift register.

cipher	operations	
	non-linear	linear
AES-128	S-box: inversion in $GF(2^8)$	LFSR, XOR
PRESENT-80	4-bit S-box algebraic degree 3	upcounter, XOR
SPECK-128/128	addition modulo 2^{64}	upcounter, XOR
NOEKEON	4-bit S-box algebraic degree 3	LFSR, XOR
KTANTAN-64	NLFSR	LFSR, XOR

architecture. In Sect. 5.1, we introduce the ciphers and architectures that are mapped onto the proposed cFA array using the tailored tool flow. In Sect. 5.2 we present and discuss the obtained results.

5.1 Evaluated Ciphers and Architectures

We select several representative block ciphers from [12] and consider architectures for encryption only. The ciphers are selected with the goal of maximizing the coverage of different block cipher structures, operations and types of key schedules. Tables 5 and 6 summarize our selection. AES [16], PRESENT [17] and NOEKEON [18] are SPN-based, with 8-bit and 4-bit S-boxes. SPECK [19] is ARX-based and KTANTAN [20] is based on a stream-cipher-like structure. KTANTAN is the direct-key version of KATAN, and for NOEKEON we also opted for the direct-key mode. We did not include a Feistel cipher, because the Feistel structure is implemented through routing, while the operations in Feistel ciphers are similar to those in other ciphers.
The ciphers in direct-key mode, NOEKEON and KTANTAN, are implemented as shown in Fig. 5. The state register in NOEKEON is 128 bits wide and a final linear function is in place to compute the ciphertext. The 128-bit key is applied to the non-linear state update function and to the final linear function. In KTANTAN, the 64-bit state register is updated as a non-linear feedback shift register (NLFSR). There is no final linear function, i.e. the ciphertext is taken directly from the output of the non-linear state update function. Both ciphers use an 8-bit linear feedback shift register (LFSR), that is initialized with

a specific non-zero value at the start of the encryption, to generate a round constant for the state update function. The plaintext is loaded into the state register through a multiplexer at the start of the encryption.

The architecture of the other three ciphers is given in Fig. 6. These ciphers use a key schedule that computes a round key for each non-linear state update. The key schedule itself is also a non-linear function. In PRESENT, the state register and the key register are 64 and 80 bits wide, respectively. In AES and SPECK, both the state register and the key register are 128 bits wide. A multiplexer is used to load the plaintext and the key into the state register and the key register, respectively, at the start of the encryption. Either an 8-bit upcounter, generating a round number for the PRESENT and SPECK key schedule, or an 8-bit LFSR, generating a round constant for the AES key schedule, are included for the state update function. For the AES S-box, we use Canright's representation, described in [21].

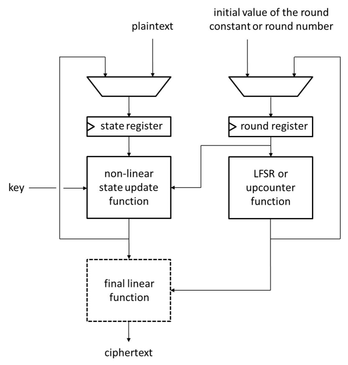

plaintext
initial value of the round constant or round number
state register
round register
key
non-linear state update function
LFSR or upcounter function
final linear function
ciphertext

Figure 5: Architecture of NOEKEON and KTANTAN.

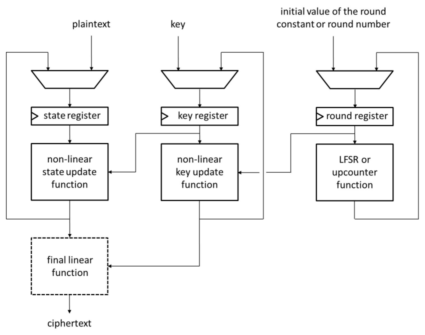

plaintext
key
initial value of the round constant or round number
state register
key register
round register
non-linear state update function
non-linear key update function
LFSR or upcounter function
final linear function
ciphertext

Figure 6: Architecture of AES, PRESENT and SPECK.

Further, we design threshold implementations with 3 input shares and 3 output shares. They are based on similar architectures, with a shared state update function and a shared key schedule. For the shared AES S-box, we follow the approach described in [22]. For the other ciphers, the non-linear functions are decomposed into quadratic functions with pipelining registers in between. For the addition modulo 2^{64} in SPECK, we use a pipelined structure of shared full adders, as proposed in [15]. All the designs are described in VHDL and synthesized using the tool flow described in Sect. 4. These experiments use only parallel round-based architectures (as shown in Figs. 5 and 6), but our tool flow supports any design implemented in an HDL.

5.2 Results

We compare the mapping of the considered block ciphers onto our new configurable array with the mapping onto a state-of-the-art commercial FPGA. We use the ISE Design Suite 14.7 of Xilinx to synthesize the block cipher designs for a Virtex 7 FPGA. Since our design flow allows to use the routing capabilities of Xilinx tools, it is possible to directly compare the results of our cell architecture with the results obtained for Virtex 7.

We report on the area based on the number of SLICEL and SLICEM, and on the critical path based on the logical depth in terms of LUTs and fast adders. The estimates of both the area and the combinatorial delay of the Xilinx cells (re-implemented by us according to Fig. 1) are given in Table 1, based on the NanGate 45nm standard cell library. For the configurable array consisting of our cFA slices, we use the tool flow described in Sect. 4. The two experiments are run with the same VHDL code. Table 7 shows the results for the evaluated ciphers. In the table, TI stands for threshold implementation and PRESENT-80-D3 and PRESENT-80-D2 denote versions of PRESENT in which the S-box is described as a function of degree 3 and decomposed into two quadratic functions, respectively.

As mentioned, the reported results do not take routing into account. Since our cell has 6 inputs and 2 outputs, just like Xilinx LUTs, and since we merge our cells into slices with an interface comparable to the interface of Xilinx slices, routing strategies similar to those in commercial Xilinx FPGAs can be applied to our configurable cell array. Additionally, the main advantage of our cell is the reduced area and number of configuration bits, with a comparable combinatorial delay. These figures of merit can be inferred after synthesis.

When carrying out the direct comparison with Xilinx based only on configurable cells, without taking into account routing, our cFA-based architecture is much more efficient in terms of silicon area, both for the logic and for the configuration memory. When we look at the critical path, our architecture outperforms Xilinx FPGAs for some designs, but gives worse results for others. Especially for ARX-based ciphers, the dedicated fast carry chains in commercial FPGAs result in a lower critical path. The obtained results are anyway encouraging. In fact, since the resource occupation of our solution is significantly less than the one of commercial FPGAs, we could consider to add dedicated carry chains in our cFA slice as well to further increase the performance of our architecture, while still maintaining an extremely limited area occupation.

Table 7: Comparison of our architecture to a Xilinx Virtex 7 FPGA for the evaluated ciphers. The table shows the number of SLICEL and SLICEM primitives (for the Xilinx architecture) and the number of cFA cells (for our architecture). It also shows the area in μm^2, the critical path in *ns* and the number of configuration bits, denoted with conf, for both architectures.

cipher	Xilinx					cFA array			
	SLICEL	SLICEM	area	critical path	conf	cFA	area	critical path	conf
AES-128	404	0	179,053	4.95	105,040	624	27,886	4.97	14,976
PRESENT-80-D3	70	0	31,024	1.65	18,200	190	8,491	0.95	4,560
PRESENT-80-D2	74	0	32,797	1.65	19,240	139	6,212	1.64	3,336
SPECK-128/128	130	0	57,616	5.99	33,800	294	13,139	9.91	7,056
NOEKEON	168	0	74,458	3.30	43,680	288	12,871	2.41	6,912
KTANTAN-64	80	0	35,456	2.2	20,800	119	5,318	1.95	2,856
AES-128-TI	2,076	120	1,092,679	2.2	582,640	3,058	136,656	1.54	73,392
PRESENT-80-TI	350	0	155,120	1.65	91,000	638	28,511	1.01	15,312
SPECK-128/128-TI	436	0	193,235	1.10	113,360	1556	69,535	1.33	37,344
NOEKEON-TI	846	0	374,947	2.75	219,960	952	42,543	2.12	22,848

Although our experiment evaluates five block ciphers and their threshold implementations, we expect that other block ciphers will achieve similar results, confirming the performance of our solution. This expectation is justified by the fact that other block ciphers use structures and operations similar to the ones examined. The same expectation holds for authenticated encryption algorithms, hash functions and public-key algorithms based on binary field arithmetic. Public-key algorithms based on prime fields can probably not be mapped onto our cFA architecture in an efficient way, but they already benefit from the DSP slices available in commercial FPGAs.

As a final note, we want to stress on the fairness of the comparison between our cell and the reference Xilinx one. In particular, we know that our re-implementation of the Xilinx configurable cells features a higher area and larger combinatorial delay than the real-life results. However, also our newly proposed cFA cell would be much smaller and faster if it would have been implemented in a commercial technology using an optimized custom design instead of a collection of standard cells. Furthermore, our results do not include routing yet. We expect that routing would introduce a larger overhead in our solution than in a Xilinx FPGA. In fact, for each of the evaluated ciphers, the number of cFA cells is larger than the number of SLICEL/SLICEM cells. Nevertheless, we believe that, given the drastic area reduction in both cell logic and configuration memory, the additional routing overhead would still lead to favorable results for our cFA architecture.

6 Conclusions

We proposed a new configurable cell that is particularly suitable for the implementation of block ciphers. The cell is a full adder with configurable inputs (cFA). A cFA-tailored tool flow was developed in order to map a HDL description on the configurable array. The cFA and the tool flow have been successfully validated using block ciphers with different structures and operations as well as threshold implementations of the ciphers. The results show that our solution outperforms the LUT-based configurable cells of commercial FPGAs in terms of area and SRAM configuration resources, while offering comparable critical paths. We believe that the positive results of our solution will also be confirmed in other block ciphers, since they make use of the same basic operations that our cFA cell was optimized for. The same holds for authenticated encryption algorithms, hash functions and public-key algorithms based on binary field arithmetic. This makes our cell an appealing solution for being integrated in future FPGAs as an application-specific configurable cell array or as an embedded FPGA (eFPGA) in ASICs. Our solution efficiently enables cryptographic agility, a fundamental property for secure IoT devices and cyber-physical systems.

References

[1] N. Mentens, E. Charbon, and F. Regazzoni, “Rethinking secure fpgas: Towards a cryptography-friendly configurable cell architecture and its automated design flow,” in *Field-Programmable Custom Computing Machines (FCCM), 2018 IEEE 26th Annual International Symposium on.* IEEE, 2018.

[2] S. Nikova, C. Rechberger, and V. Rijmen, “Threshold implementations against side-channel attacks and glitches,” in *ICICS*, ser. LNCS, vol. 4307. Springer, 2006, pp. 529–545.

[3] P. Sasdrich and T. Güneysu, “Efficient elliptic-curve cryptography using Curve25519 on reconfigurable devices,” in *ARC*, ser. LNCS, vol. 8405. Springer, 2014, pp. 25–36.

[4] D. Chen, N. Mentens, F. Vercauteren, S. Roy, R. Cheung, D. Pao, and I. Verbauwhede, “High-speed polynomial multiplication architecture for ring-LWE and SHE cryptosystems,” *IEEE TCAS I*, vol. 62, no. 1, pp. 157–166, 2015.

[5] A. J. Elbirt and C. Paar, “An instruction-level distributed processor for symmetric-key cryptography,” *IEEE TPDS*, vol. 16, no. 5, pp. 468–480, 2005.

[6] R. R. Taylor and S. C. Goldstein, “A high-performance flexible architecture for cryptography,” in *CHES*. Springer, 1999, pp. 231–245.

[7] M. Budiu and S. Goldstein, “Fast compilation for pipelined reconfigurable fabrics,” in *FPGA*, 1999, pp. 195–205.

[8] F. Bache, T. Schneider, A. Moradi, and T. Güneysu, “SPARX - a side-channel protected processor for ARX-based cryptography.”

[9] E. Gulcan, A. Aysu, and P. Schaumont, “BitCryptor: Bit-serialized flexible crypto engine for lightweight applications,” in *INDOCRYPT*, ser. LNCS, vol. 9462. Springer, 2015, pp. 329–346.

[10] Xilinx, “7 series FPGAs configurable logic block,” 2016.

[11] N. Inc, “Nangate freepdk45 open cell library,” 2017.

[12] CryptoLUX, “Lightweight block ciphers,” https://www.cryptolux.org/index.php/Lightweight_Block_Ciphers, 2017.

[13] P. C. Kocher, J. Jaffe, and B. Jun, “Differential power analysis,” in *CRYPTO*, ser. LNCS, vol. 1666. Springer, 1999, pp. 388–397.

[14] B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, and G. Stütz, “Threshold implementations of all 3 x 3 and 4 x 4 S-boxes,” in *CHES*, ser. LNCS, vol. 7428. Springer, 2012, pp. 76–91.

[15] T. Schneider, A. Moradi, and T. Güneysu, "Arithmetic addition over boolean masking - towards first- and second-order resistance in hardware," in *ACNS*, ser. LNCS, vol. 9092. Springer, 2015, pp. 559–578.

[16] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*, ser. Information Security and Cryptography. Springer, 2002.

[17] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: an ultra-lightweight block cipher," in *CHES*, ser. LNCS, vol. 4727. Springer, 2007, pp. 450–466.

[18] J. Daemen, M. Peeters, G. V. Assche, and V. Rijmen, "NOEKEON," http://gro.noekeon.org/.

[19] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The SIMON and SPECK families of lightweight block ciphers," Cryptology ePrint Archive, Report 2013/404, 2013.

[20] C. D. Cannière, O. Dunkelman, and M. Knezevic, "KATAN and KTANTAN - A family of small and efficient hardware-oriented block ciphers," in *CHES*, ser. LNCS, vol. 5747. Springer, 2009, pp. 272–288.

[21] D. Canright, "A very compact S-box for AES," in *CHES*, ser. LNCS, vol. 3659. Springer, 2005, pp. 441–455.

[22] A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang, "Pushing the limits: A very compact and a threshold implementation of AES," in *EUROCRYPT*, ser. LNCS, vol. 6632. Springer, 2011, pp. 69–88.