Xilinx Open Hardware 2018

FPGA-based platform for Lightweight Cryptographic Algorithms

Students:
Andrea Castellani
Samuele Cornell

Supervisor:
Massimo Conti

Contents

1 Introduction

In recent years we have seen an astonishing growth rate of highly constrained interconnected device systems for various and indeed formidable applications. However the increasingly pervasiveness of such devices raises equally formidable security and privacy concerns which have concurrently sparkled an increasing research in cryptographic primitives that are suitable for these devices. For this reason, for the first time, the NIST has issued an open call to develop a lightweight encryption standard [1] for resource-constrained IoT devices. The standard will consist of a portfolio of lightweight algorithms based on symmetric cryptography. This is because IoT-related applications encompasses a too broad range of devices to be covered with an unique cryptographic primitive. Devices have different power budgets and different computing power and range from passive RFID tags to more complex sensors with 32-bit MCUs. This is why the NIST call is divided into distinct application profiles to cover all the possible applications and use-cases for IoT devices, with each profile having different algorithm requirements and being targeted to a particular class of devices.

As of today, the NIST has specified two different profiles aimed at developing Authenticated Encryption with Associated Data (AEAD) algorithms [2]. The first is targeted both at software and hardware implementation and requires additional hashing capability while the second is targeted only to extremely constrained devices and hardware-only implementation such as RFIDs and doesn't require hashing capability. Both should offer significant improvements over current standards (AES) in software (MCU), ASIC and FPGA implementations. While the call for such AEAD algorithm with optional hashing will start in the following months, there is already a florid research around lightweight cryptographic primitives suitable for this type of algorithm.

While there are known and widely accepted tools for assessing the performance and resource utilization of such ciphers on MCUs and software, such as eBACS [3] and FELICS [4], the same is not true for hardware implementations where there is no uniform metrics nor platforms that are used thoroughly in the literature, thus fair comparison of different cryptographic primitives is somewhat problematic for this reason. Unfortunately, in most of the papers, regarding hardware implementation performance of lightweight ciphers only ASICs are considered, the most used metrics are Gate Equivalent GE area and throughput and power consumption at a 100 kHz clock frequency. All of these parameters however strongly depend on the technology process and to a lesser extent on the standard cell library and even by the synthesis and implementation software. This, aside from the fact that not all the metrics listed above are the most common but are not universally used, make the comparison between different works, as has already been said, extremely problematic. Another potential issue, lies on how the metrics are effectively computed as there isn't again wide agreement on a standard methodology. For example in some works, power consumption rather than being estimated with a simulation is taken directly only from the area in GE. Similar considerations can be made for FPGA performance. In fact, in the rare case where is reported, there is not a de-facto standard evaluation platform that allows for a fair comparison between different ciphers.

In this work we will compare three different promising State-Of-The-Art lightweight block ciphers which can be used as the building block for the aforementioned AEAD algorithm on the Digilent Zybo Zynq-7000 ARM/FPGA board. Those three lightweight block ciphers algorithms were chosen after an extensive review of the literature as they appear among the most probable winners of the call in our humble opinion. A purposely made 128 key length version of the AES will be also used in the comparison. In particular, as we are evaluating three different lightweight block ciphers, we will focus our attention on resource utilization and power consumption. For what regards power consumption we will obtain power consumption figures from simulation and also experimentally probing the board.

With this work, beside contributing with FPGA implementations of three new ciphers, we want to show how an FPGA board can be used as a common cipher evaluation platform for hardware performance. In fact the use of an already available FPGA platform such as the Zybo has many advantages over existing non-homogeneous and confusing comparison methods which, in literature, are mostly based on ASICs:

- **Standard**

  It is possible to implement various cryptographic primitives on the same device at different levels of parallelization and pipelining. This allows for a fair comparison. This is clearly not limited to lightweight block ciphers but also extends to other families of cryptographic primitives such as Quantum

resistant Public key algorithms which are also in evaluation today as they will replace the RSA in the near future [5].

- **Accessible**

  A board like the one we used is easily accessible while CAD synthesis, implementation and simulation tools and cell libraries are rather expensive. Moreover, in our opinion, also FPGA synthesis and implementation tools like the Vivado are more user friendly and more accessible than aforementioned ASIC ones and allow for good power consumption estimation as we will show. This has the obvious benefit that more researchers and students around the world can contribute to security-related research as not all institutions can cope with the cost of expensive tools.

- **Multi-platform benchmarking**

  FPGAs, at least for what concerns crypto-primitives, allow for an evaluation that is obviously hardware oriented but also reflective of a cipher software performance. In fact, most of ASIC oriented crypto-primitives are built purposely from the ground up for using Scan Flip-Flops, which are basically standard Flip-Flops but with the Data input controlled by a multiplexer. This type of Flip-Flops were originally developed for testing purposes but, are widely used in these types of ciphers to perform permutations in an extremely efficient manner. This has the obvious drawback that, such cipher being built to use such components, have poor software performance as those permutations in software requires several clock cycles. Because FPGAs doesn't allow for efficient synthesis of such components and at the same time retain hardware parallelization and pipelining, using FPGAs as a benchmarking platform for ciphers tends to privilege ciphers which have also good software performance.

- **Experimental verification**

  The use of a common FPGA board allows for experimental verification of power and energy consumption, thus going beyond figures obtained through mere simulation or worst by merely obtaining power figures from the resource obtained after synthesis (GEs in ASICs, slices in FPGAs). This is clearly not possible with ASICs as it would be prohibitively expensive. Moreover having an hardware implementation potentially allows to test the cipher, on the same platform, also for Side Channel attacks which are crucial in IoT related security.

Another argument in favor of FPGAs for evaluating lightweight ciphers is the fact that FPGAs are increasingly used in IoT and, for this reason, they are also explicitly mentioned in the aforementioned NIST call. In fact, embedded FPGAs and SoCs like the Zynq are especially suited in applications where interfacing with multiple sensors is required as they allow for parallelization and agnostic sensor interface. This is often the case for many critical applications such as industrial, automotive, military et-cetera; but as the number of sensors and interconnected devices is growing this could be well extend to other, less critical and specialized, areas. FPGAs devices are thus ideal for IoT gateways and central hubs for IoT networks. Those devices doesn't have the same extremely low power budgets like deployable sensors, but require more resources and a special attention to security as it constitutes a Singular Point of Failure in a network and can be directly connected to the Internet. The flexibility and reconfigurability offered by FPGAs in this scenario is a distinct advantage as it allows for future proofing.

2 Description

As we said in the Introduction, we have chosen three different lightweight block ciphers after an extensive review of literature. We have examined and compared 34 different ciphers and up to 175 different hardware implementations and chosen Simon [6], Simeck [7] and SKINNY [8] as the among the most promising for the upcoming NIST call. We based our decision around Figure of Merits (FOM) especially important in the field of lightweight cryptography for IoT:

- ◇ Power and Energy Consumption
- ◇ Area in Gate Equivalent
- ◇ Security

Other FOMs like throughput and maximum clock frequency were also considered but were deemed as secondary in importance for IoT applications. We have also made this choice to encompass two different design philosophies.
Simon and Simeck are two very similar ciphers and have a Feistel Network (FN) structure (like the old DES). This makes them extremely power savvy and because their round function is extremely simple also they tend to occupy very little area.
SKINNY on the other hand, has a Substitution Permutation Network (SPN) (like the AES), this makes this cipher a little more complex even if the authors have taken precautions to keep it as simple as possible. For example, the SBOX is expressed via simple logic operations. It has also the advantage of having mathematically proven security bounds for certain types of attacks and in general the SPN structure is considered better, from a security standpoint, than the Feistel.

We have written VHDL code for a total of 20 implementations of such three ciphers, for different key/plaintexts sizes but also at various degrees of serialization, in other words at different datapath sizes. We have also written VHDL code for fully parallel (128-bit datapath) AES 128/128 (128 bits key and 128 bits plaintext) to be used in the comparison with aforementioned ciphers. In the Table 2.1 a full list of our implementations is reported.

Table 2.1: Our ciphers implementations.

Cipher	**Block Size**	**Key Size**	**Datapath**	**Structure**
AES	128	128	128	SPN
SIMECK	32/48/64	64/96/128	1/16/24/32	FN
SIMON	32/48/64/128	64/96/128	1/16/24/32/64	FN
SKINNY	64/128	64/128	4/8/64/128	SPN

Because the ciphers specifications vary with plaintext and key sizes and because the implementations varies strongly with the degree of serialization (a fully parallel implementation is extremely different from a 1bit serial implementation of the same cipher) we had to write the code for each version and we generated many Vivado project to make the code easier to read. We also made our best efforts to optimize each different version of the ciphers for minimum resource utilization as this will lead to less power consumption. This was an extremely hard task as it required a deep understanding of the cipher operations in order to be able to perform them in a serialized fashion in the best possible way. At the same time, while doing our best to implement in the most efficient way each cipher, we took proper precaution in keeping a somewhat standard structure among the different ciphers in order to do a fair and unbiased comparison. The state machine for each cipher has been kept as homogeneous as possible and the same is true for the top entity in each design which serves only the purpose of testing the correct operation of the cipher.

Each cipher we implemented is organized as illustrated in Figure: 2.1.
The top entity for each design is used only for testing the cipher as we are merely interested in getting the power and energy consumption and other FOMs for only one encryption operation in order to compare the

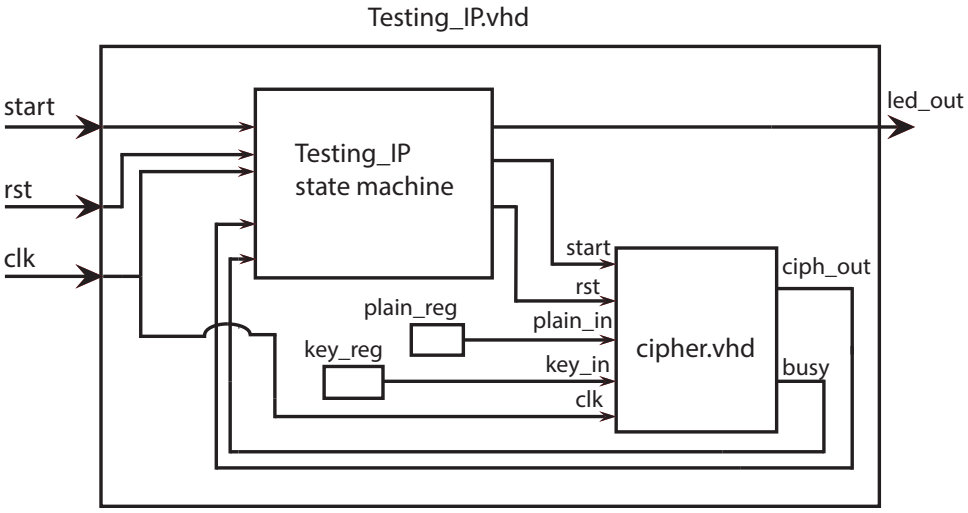

Figure 2.1: General block diagram for our designs.

ciphers. This top entity acts as a test-bench for the cipher, the start and rst ports are mapped to the Zybo Zynq-7000 ARM/FPGA board external JA Pmod N16 and L15 connectors respectively. In this way the cipher encryption operation can be triggered via an external signal which will be used also as a trigger for an oscilloscope which will be employed to validate experimentally the power figures obtained through simulation in Vivado and Xilinx Power Estimator (XPE). The top entity Testing_IP has the task to perform only one encryption as the start port goes high. A state machine is used in this entity to manage the cipher proper operation; for example is used to correctly load the plaintext and the ciphertext. The led_out port of the top entity is instead used merely as a visual cue of the fact that the encryption was successful. Plain_reg and key_reg are basically registers which contains a fixed test vector for the plaintext and the key respectively.

The entity for the cipher is kept very simple in accordance to the literature: only the core of the cipher is implemented, again to allow a fair comparison, no particular mode of operation was implemented and the cipher simply operates in Electronic Code Book (ECB) mode. Again this is standard in literature when comparing lightweight ciphers and other cryptographic primitives but it must be kept in mind that such mode of operation is not secure in practical applications and a proper mode must be employed. This is very simple to do and requires minimal modifications to our code.

Each cipher entity has been designed to have the same ports: a reset port (rst), a start port which signals the cipher to start the encryption and a plaintext and key input ports. Those last two ports can be of varying size, it depends on the key and plaintext size in the cipher specification but also on the datapath of the implementation. For example for a 128 bit key and plaintext cipher which is implemented in a 8bit serial fashion those ports will be simply vectors of size 8. Another thing to note is that we have implemented the encryption only version of the ciphers because for the ones we have chosen, encryption and decryption are practically the same.

For our project the target platform was, as it was already mentioned, the Digilent Zybo Zynq-7000 ARM/FPGA board. We choose the clock frequency of 125 MHz for all of our designs instead of the 100 kHz clock frequency often found in literature for several reasons. Firstly because it is simply the native internal clock frequency of the PL in the Zybo board so using this clock frequency for all the logic has the obvious benefits of reducing the need of additional components which would be employed to scale the clock frequency. In fact, if a MMCM was used to scale the frequency a large portion of the power consumption would be largely due to this component. while this potentially would have little effect in simulation, as in the XPE tool it is possible to check the power figure for each entity, it would be indeed a problem when performing experimental measures via an oscilloscope on the board as the power requirements for this component will mask the power consumption of ciphers. Secondly the 100 kHz benchmarking frequency used in literature is totally arbitrary and doesn't actually reflect any practical application. Even for RFIDs the operating

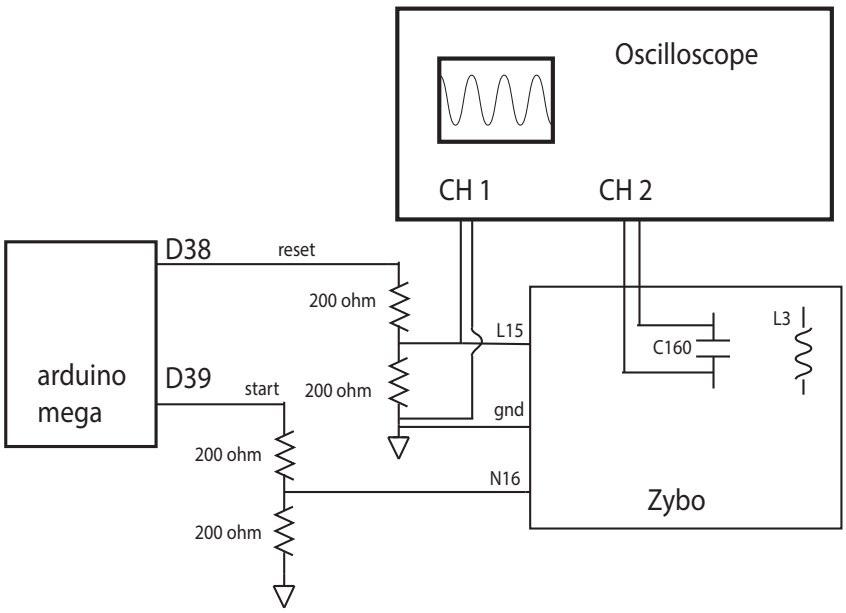

Figure 2.2: Experimental measurements setup.

frequency is in the MHz range and not at those extremely low clock frequencies. Lastly because it has no effect if the goal is comparing the power consumption between the ciphers as we will implement them at the same clock frequency of 125 MHz. Moreover it is well known that dynamic power consumption scales approximately linearly with the frequency: thus the frequency at which we will make the comparison doesn't have any practical impact for dynamic power:

$$P_{tot} = P_{stat} + P_{dyn} = I_{off} V_{DD} + p_{sw} C_L f_c V_{DD}^2 \tag{2.1}$$

where p_{sw} is the switching probability, I_{off} is the leakage current, V_{DD} is the supply voltage and C_L the capacitance load.

The power consumption figures for all ciphers, as it has been mentioned, were estimated with the Vivado Power Analysis tool and with Xilinx Power Estimator using a generated .saif file produced via Vivado Post-Implementation Timing Simulation of designs. In particular we are interested in the dynamic power consumption of each cipher and also on the board resource utilization. Board resources utilization is especially important as this last reflects the area the cipher requires, a figure which is associated with static power consumption in ASICs (while in FPGAs the static power is independent of user design).

The dynamic power figure has also been verified experimentally with the setup illustrated in Figure 2.2 for some versions of the ciphers we have implemented. In particular, as the measurement process was time intensive, we have chosen to perform experimental measurements only for the most promising cipher implementations in terms of resources and power consumption obtained by the XPE tool.

An Arduino Mega is used to generate the start signals for the aforementioned Testing_IP which will then promptly make the cipher under test perform one encryption with a pre-defined tests vectors. If the encryption is successful a LED on the board turns on. The reset signal is necessary in order to perform more than one measurements as it resets the state of the Testing_IP which will then be able to make the cipher perform another encryption. These signals are generated via a simple code running on the Arduino Mega in such a way that an encryption is performed every second. The Arduino Mega basically sets its D38 (reset) and D39 (start) pins high every second, these pins are connected respectively to the Pmod JA L15 and N16 pins of the Zybo board via a voltage divider which consists of 200 ohm resistors and has the task of halving the input voltage from the Arduino 5V to 2.5V as the JA Pmod I/O ports are 3.3 V level. It was chosen to generate these signals externally from the board as in this way the start signal can be used to trigger the acquisition of the oscilloscope. In order to get the dynamic power consumption for one cipher

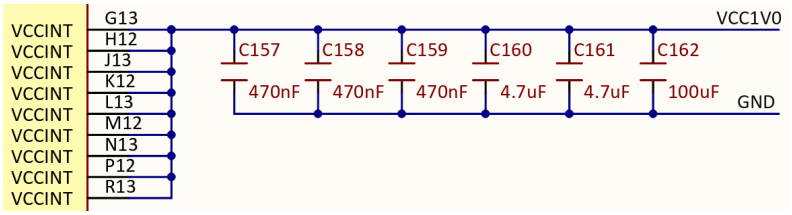

Figure 2.3: Capacitor C160 from Zybo schematic (source: [9]).

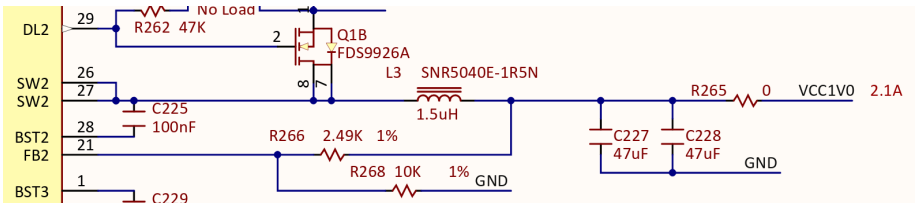

Figure 2.4: Inductor L3 and resistence R265 from Zybo schematic (source [9]).

encryption we had to get via the oscilloscope the current and voltage at the Programmable Logic (PL) of the Zynq chip. Fortunately the Zybo is well documented as the schematic is made available by Digilent [9]. From the datasheet of the Zynq-7000 XC7Z020 it is possible to see that the power supply pin for the PL is the VCCINT. In the Zybo schematic it can be seen that this pin is fed by VCC1V0 supply line which is generated via an ADP5052ACPZ-R7 buck regulator IC made by Analog Devices.

To get the voltage for the VCCINT pin probing any capacitor in parallel with this port can suffice. We have chosen C60 only because we found it easier to probe it with the oscilloscope. We have used a simple Analog Discovery 2 oscilloscope in particular, mainly because of its 14-bit ADC resolution as in this case, as we are talking of dynamic powers in the milliwatts, resolution was deemed more important than sampling frequency which was set at 100 MHz.

Unfortunately to get the current, it is not an easy task and we had to take several workarounds. From the schematic it is possible to see that the only element where one can theoretically get the current is where the R265 0 ohm resistor is placed. In theory it is possible to swap this resistor with one with an higher, albeit small, resistance (for example a 0.1 ohm would suffice) and use then the oscilloscope to probe the voltage drop on it. The current will be simply derived from the ohm law and with a good resistor one can easily get sufficient precision. Unfortunately it wasn't possible to modify the board as it belongs to our university, what we did instead was deriving the current from the L3 inductor voltage drop as this is the only other component which can give us a glimpse of the current drawn by VCCINT. Sadly our Zybo doesn't have a power management IC with a current monitoring pin. The newer Zybo Z7 has one (Texas Instruments TPS25940) with a purposely IMON pin for current monitoring and even a dV/dT pin. This would have made easier to perform our measurements by a large margin.

Nevertheless, we found a workaround to use the L3 inductor for our scope. The R266 and R268 voltage divider used for voltage sensing in the buck regulator feedback loop basically drains no current (it was also checked experimentally) in this way we can approximate that all the current in the inductor goes towards the downstream circuit which consists of various capacitor in parallel and the VCCINT port. The current in the inductor can be estimated via numerical integration performed in the time frame where the cipher is encrypting the plaintext.

Moreover, instead of directly obtain the total power figure from the current on the inductor and the voltage on the capacitor for one encryption operation, in order to cope with the non-negligible loading effect of our oscilloscope which has a rather large parasitic capacitance of 24pF, we decided to get the difference in power consumption between the encryption phase where the cipher is effectively operating and the board idle power consumption. In this way we should be able to obtain only the dynamical power consumption of our designs, due to the switching activity during encryption operation. However we will not be able to estimate the static power consumption, but as it as already been said, because it is not reflective of user design in FPGAs,it is of little interest in our scope. Instead, for example, as a metric for estimating static power in

ASICs from our setup, we will use the resources used in the FPGA (slices, LUTs etc) which reflects the area occupied by the design. Also the dynamic power consumption of the cipher obtained in this way will be averaged from multiple measurements in order to cope with possible noise sources and other non-stationary zero mean disturbances in the measurement process.

The Python code used to process the .csv files obtained by the Analog Discovery 2 acquisitions in order to get the current in the L3 inductor and in order to compute the dynamic power consumption is described in detail in Section 4. Unfortunately, it wasn't possible to cope with the typical inductor big 30% tolerance (inductor datasheet [10]), this error unfortunately propagates up to our dynamic power estimations. However, as we will show, we were still able to obtain power figures which are close to what we expected from XPE and Vivado tools with, all in all, a very simple and accessible measurement setup. As we have said a more precise estimation would have involved a modification of the board or getting access to a Zybo Z7, two options which were not practicable for us.

3 Hardware

In the preceding Section, we have briefly described some of the design choices we have made, here instead, because of the amount of different cipher implementations and because an in depth discussion will require delving into the field of lightweight cryptography we will limit our discussion only to some optimization we have made and to the basic description of each cipher operations. We have already also talked about how we decided to structure the designs to allow a fair and unbiased comparison and why some choices were taken such that our designs in a certain sense can be considered "minimalistic". In Figure 3.1 a generic illustration, valid for all the ciphers we have implemented is reported (clock lines are not drawn). As it was said we strived to keep this very same architecture for each cipher.

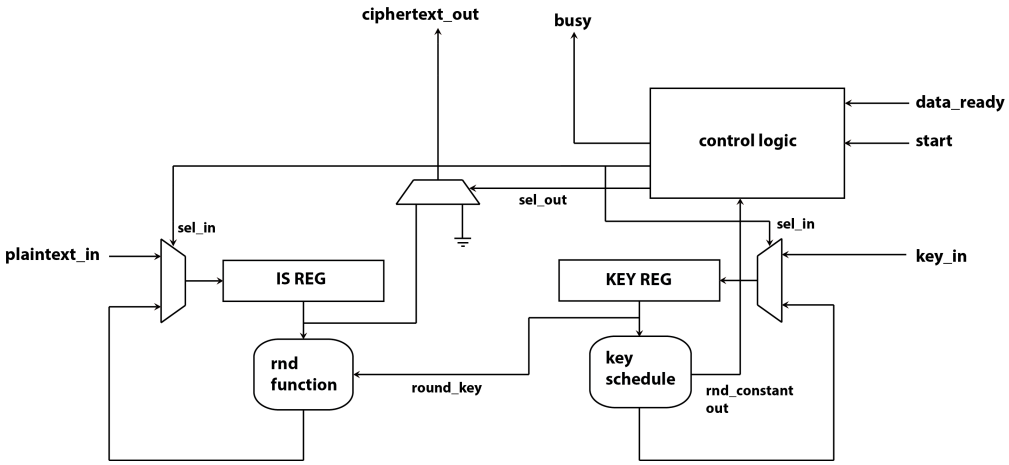

Figure 3.1: Block diagram of top entity for a generic cipher.

The control logic is basically a state machine and the cipher operations are in general managed by a counter and a linear feedback shift register (lfsr) which is also used to generate round-unique constants necessary for the correct cipher operation (rnd_constant_out in the Figure 3.1). These constants are employed generically in the key schedule (in SKINNY also in the round function) to deter related-keys attacks. When the data_ready port is high the cipher goes into the loading state, the plaintext and key input are mapped by multiplexers respectively into the internal plaintext and key registers. The busy output port during loading operation is set to high. This operation can require more clock cycles if the cipher is not implemented in a fully parallel fashion. In our implementation of the AES 128/128 this loading operation requires one clock cycle but for example in our implementation of SKINNY 128/128 8-bit serial this operations requires 128/8= 16 clock cycles. This is managed by an internal counter (or, when is possible, using the same lfsr used to generate round constants to save resources) and as the loading is complete the busy output port is set to '0' to point out to the "master" circuit that the loading phase has been completed and the cipher goes into an idle state. After the key and plaintext have been loaded, if the start port goes high the cipher begins the encryption operation and goes into encrypting state and busy is set to high.

The contents of the internal state register (IS REG) are then processed by the rnd_function which is different for each cipher and then the result is put back into the register. The same applies to the KEY REG as the key is processed by the key scheduling algorithm first and then put back in it. This operation is repeated as many times as it is specified in the cipher specification: for AES 128/128 10 rounds suffices, for Simon 128/128 68 rounds instead are necessary due to its Feistel structure which achieves slower diffusion. The control logic is responsible to determine when the encryption is completed, this is usually done either by a counter or simply using the same lfsr used for round constant generation and a comparator. At the end of the encryption the cipher outputs the ciphertext and busy returns to low. In all other states the ciphertext_out port is set to '0'.

The internal state registers and key registers in the serial implementations of SKINNY and in all implementations of Simon and Simeck (because of their Feistel Network structure) are implemented as shift

registers as this is more resource-efficient. As an example in Figure 3.2 our design for the key schedule register of the Simon cipher is illustrated. this key scheduling algorithm is strictly valid for the versions of this cipher where the key length is the double of the plaintext size (e.g. Simon 32/64) for other implementations, such as Simon 128/128, it is actually different as the key is divided onto two registers only but the same technique using shift registers which we will discuss below applies nonetheless.

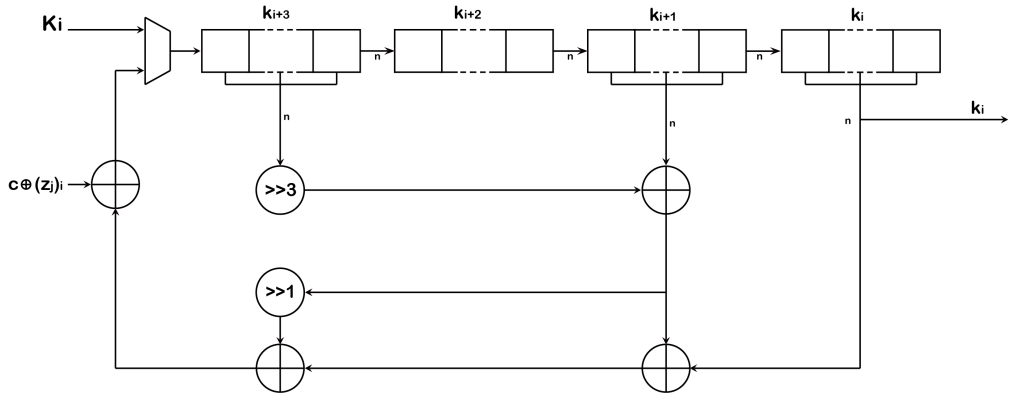

Figure 3.2: Simon key schedule.

In the Figure above $>> x$ denotes a right circular bit shift of x bits (also known as ROtate Right ROR) and $\oplus$ denotes the XOR operation. K_i is the master key input while k_i is the round key and $c_i \oplus z_i$ is the round dependant constant which is generated by the result of the lfsr xored with another constant which depends on the cipher version.
As it can be seen the k_{i+3} round key (which will be used three rounds later from the current) is generated from the other round keys via simple logic operations. In this way, in the first 4 rounds the 4 portions of the master key will be used. Because only a portion of the key register is used as the round key, it was almost straightforward to implement the Simon key schedule in this way. In fact it was purposely made to be implemented using shift-registers. This is however true as long as one is considering datapath implementations equal to the Key Size divided by four or more. In fact, the scheme in Figure 3.5 is employed in all our parallel implementations of the Simon.

However for 1-bit datapath (fully serial) implementations the original structure of the algorithm must be modified to make it possible such design. In fact, it would not be possible to perform in particular the right circular shift by one at the bottom of the Figure 3.2 in a bit-serial fashion as we are manipulating one bit at a time. It is however possible to perform the right circular shift by three via a 2:1 multiplexer which, opportunely piloted by a counter will select the right bits from the $k_{i+3} \dots k_i$ shift registers. What it can be done is find an equivalent mathematical form the cipher key schedule operation that allows to perform the right circular shift by one. This is extremely simple, if we denote k_{next} as the result of the operations before the XOR with $c_i \oplus z_i$ and as k_x:

$$k_x = (k_{i+3} >> 3) \oplus k_{i+1}$$

then k_{next} can be simple expressed by:

$$k_{next} = (k_x >> 1) \oplus k_x \oplus k_i = [(k_{i+3} >> 3) \oplus k_{i+1}] >> 1 \oplus k_x \oplus k_i$$

$$k_{next} = [(k_{i+3} >> 4) \oplus (k_{i+1} >> 1)] \oplus k_x \oplus k_i$$

and thus the scheme in Figure 3.2 becomes the one in Figure 3.3 below where the shift operation has been "brought up" near the key register.

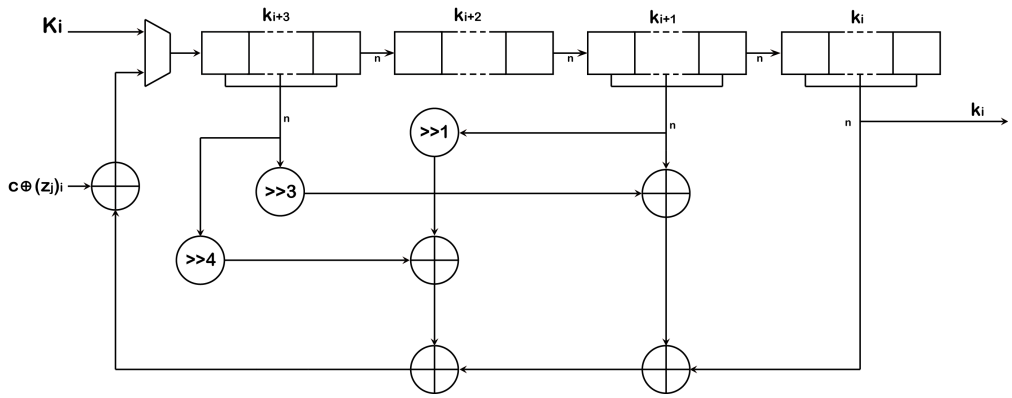

Figure 3.3: Simon modified key schedule.

Now that an equivalent form for the cipher key scheduling algorithm has been found (the security has not been compromised as the algorithm remains the same), it is possible to implement this algorithm in a 1-bit datapath design as illustrated in Figure 3.4.

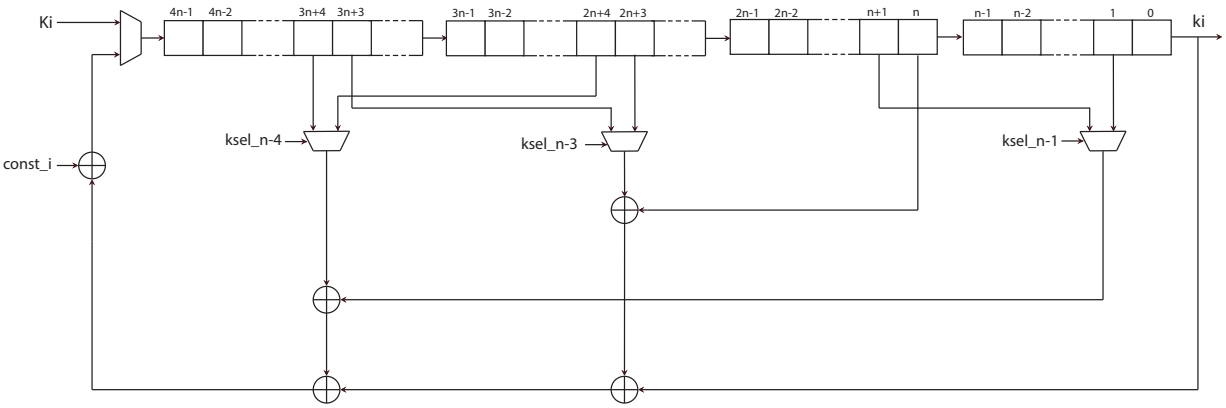

Figure 3.4: 1Bit-serial Simon key schedule.

The multiplexers implement the right circular shifts above, for example the leftmost mux is responsible for the circular right shift of 4 bits, it ksel_n-4 will be managed by the cipher state machine and a counter, when the counter is less or equal than n-4, ksel_n-4 will select the left input for the multiplexer and when it is greater than n-4 it will select the right input; as the register shifts this will be akin to the required ROR operation.

Regarding the Simon internal state and round function we will omit it here for brevity as this very same technique was used when implementing the design.

A similar strategy can be also applied to the Simeck family of block ciphers which has a similar structure being Feistel Network based and also because it borrows some design considerations directly from the Simon. Our designs for Simon and Simeck round functions and Internal states are reported in Figure 3.5 below for reference. The Figure illustrates the parallel (n-word datapath for Feistel ciphers) implementations but, as the reader can guess, with the use of multiplexers as above can be easily extended to the 1-bit serial version. We will not talk of other minor optimizations and design choices we have made which were tailored each time to the specific cipher implementation, for example, in Simon 1-bit serialized implementations we have used a two bit Johnson counter together with the lfsr instead of instantiating another 4-bit counter to count the rounds and determine the end of the encryption operation. Minor optimizations like this, altogether, allowed us to save resources in the FPGA.

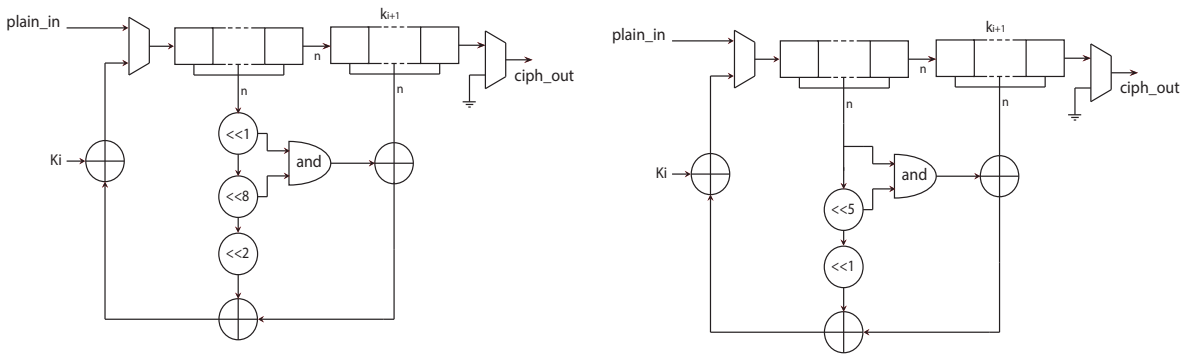

Figure 3.5: Round functions for Simon (right) and Simeck (left).

To conclude this overview of our designs for the ciphers we will like to show our implementation of the SKINNY internal state as this too required some clever arrangement. SKINNY differs totally from the Simon and Simeck ciphers above as it is a Substitution Permutation Network ciphers akin the AES. Thus its internal state is arranged in a matrix form and operations are performed either on each element of this matrix (which is called a word/cell and in skinny it is either of 4-bits or 8-bits depending on its plaintext size) or on the rows and columns. For the key the same applies, it is also arranged as a matrix and in SKINNY the key scheduling algorithm consists in the xoring of some elements of this matrix with constants generated by a lfsr and a permutation which is performed over the entire key register in this matrix form. In Figure 3.6 the main operations of the SKINNY are illustrated.

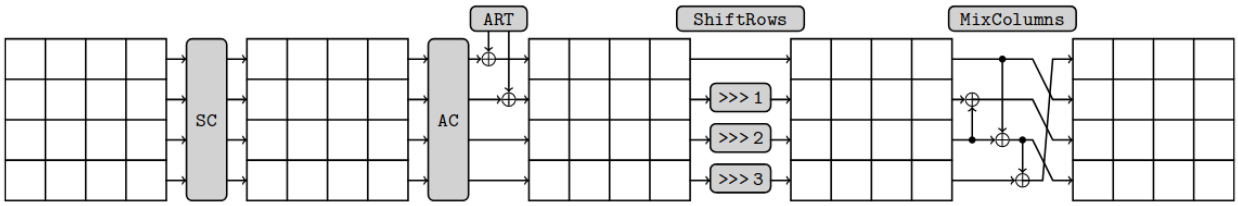

Figure 3.6: SKINNY internal state and operations (source [8]).

The first operation is called Subcells (SB) and is basically an SBOX, the second is called AddConstants (AC) and it consist in xoring the first two rows of the Internal State (IS) with a round-dependant constant generated by a lfsr, following the round key is xored with these first two rows. These three operations are relatively easy to serialize with a datapath equal to the word size (cell of the matrix), the internal state would be implemented as a shift register and each cell of the matrix (4-bit or 8-bit) would be processed by the SBOX and following the XORs with the constant and the round key will be applied for the first 8 words using a counter and some control logic. However, for the last two operations above illustrated, ShiftRows and MixColumns this is not as easy. In fact they operate on each row at the same time and it is not trivial to serialize them in an efficient manner. In detail, ShiftRows is simply a circular right shift different for each row and MixColumns is simply a permutation and mutual xoring between the rows. To perform these operations in a 4-bit/8-bit datapath design we decided to implement the IS as illustrated in Figure 3.7 below.

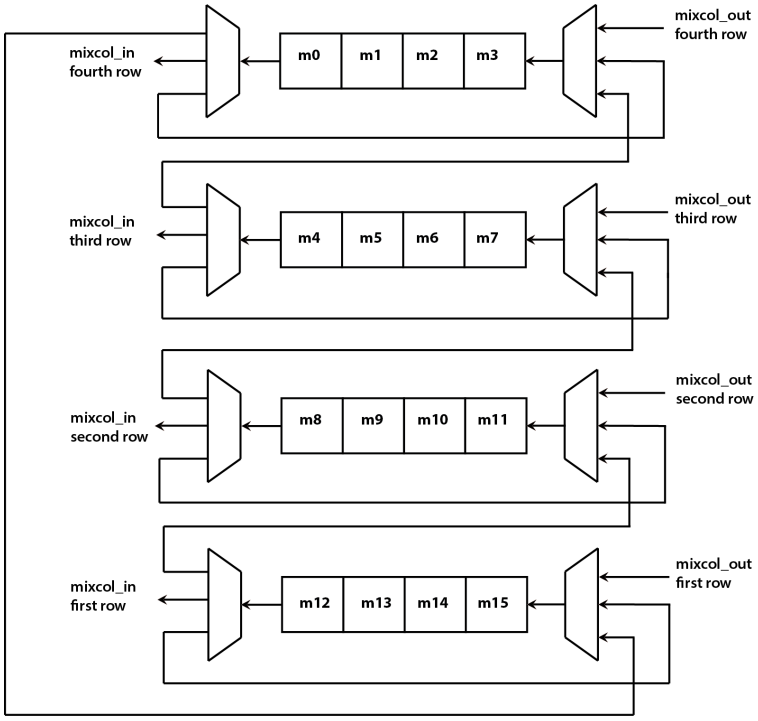

Figure 3.7: Serialized design for SKINNY internal state.

Each row is implemented as a shift register with its own CE (Clock Enable, not showed in Figure), the inputs and outputs for each row are selected via multiplexers. In this way via proper control logic selecting the outputs and the inputs for each row, it is possible to implement ShiftRows and MixColumns in a serialized fashion. For example for ShiftRows basically all the muxes will be configured so that the output of each row is mapped to its input. Via a counter then and a comparator the CE of each row will be toggled depending on the right shifts that each row has to perform. This operation such serialized requires 7 clock cycles. For MixColumns which in our design is performed in another entity the output and inputs of each row/shift-register will be mapped to the inputs and outputs of this entity and in this way the MixColumns operation can be performed in 4 clock cycles as each rows shifts right. For the first three operations instead the IS is configured so that the output of each row is input to the input of the next, in this way the IS becomes more of a contiguous shift-register and the SBOX operation and the following two can be applied on each cell sequentially as the register shifts. This will require 16 clock cycles. In this way many resources of the FPGA will be spared and power consumption will be reduced. Again as a tradeoff, as in all serialized designs, the throughput is diminished.

Finally regarding our AES-128/128 design that we have developed purposely for comparison reasons has the same top structure and control logic of the other ciphers. Its datapath is of 128-bit so it is fully parallel, it achieves encryption in 10 rounds. We explored the opportunity of using the Canright description of the AES SBOX [11] where the SBOX is implemented via logical operations but it was found that on FPGAs it is more resource efficient to use the simple standard LUT (Look-Up Table) representation. Instead for the RCON generation (Round-constant generation) of the AES we have used the algebraic approach as these can be derived algebraically. It was found that this approach to RCONs is more resource efficient than simply storing the RCONs for all 10 rounds in memory.

4 Software

In this section we will make a brief description of the Python code used to process the csv files generated from the oscilloscope acquisitions. The code is open-source and uses full open-source libraries such as NumPy and SciPy. It was written in Jupyter Notebook IPython environment. As it has already been said in Section 2, we obtained the ciphers dynamic power consumption probing capacitor C160 and inductor L3 on the board. In Figure 4.1, an oscilloscope acquisition for the voltage across C160 when the AES-128/128 is encrypting is reported:

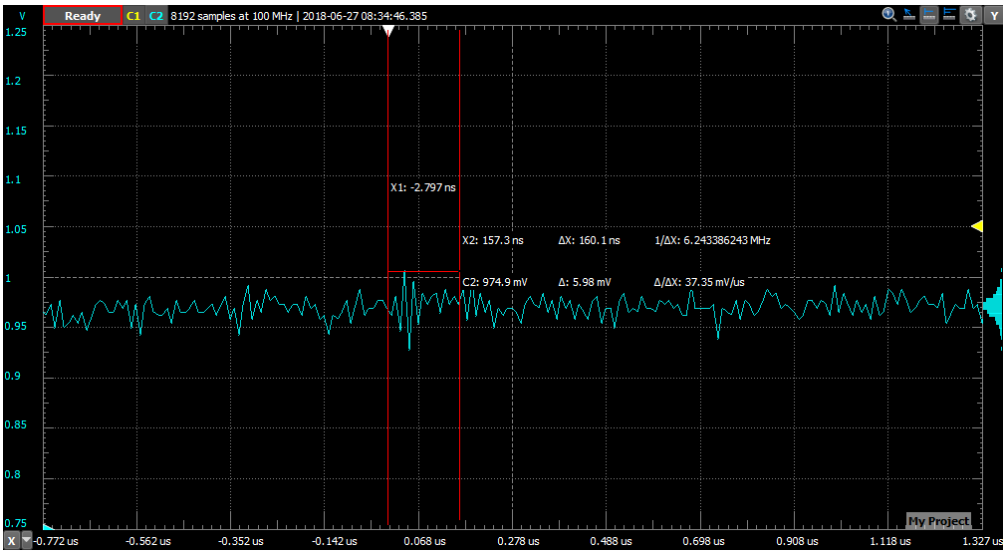

Figure 4.1: Voltage acquisition from oscilloscope across C160 when the AES-128/128 is encrypting.

In the Figure above it has been highlighted the time-frame where the AES-128/128 is performing the encryption. As it can be seen there is much noise in the signal due to other spurious sources and FPGA activity which are not correlated to the cipher operation.

In the Figure 4.2 below instead, we report the oscilloscope acquisition for the voltage drop across the L3 inductor with, again, AES-128/128 performing an encryption. The visible resonances are largely due to the oscilloscope parasitic capacitance. Differential measurements were taken precisely to counter the loading effect of the oscilloscope.

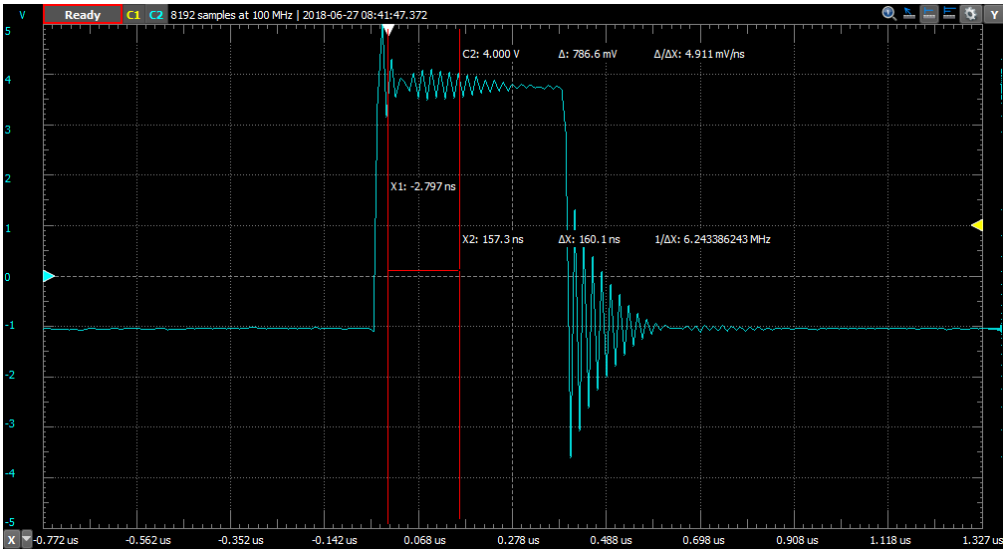

Figure 4.2: C.

To take proper care of the noise and other spurious disturbances in our measurement we took 20 different acquisitions for each cipher examined: 10 when the cipher is effectively encrypting and 10 when the cipher is in idle. In the python code the acquisitions are loaded via a simple functions into an array, with the time axis discarded. The result is a two dimensional array where each entry is the array of the voltage values in the acquisitions.

The current through the inductor is computed via cumulative trapezoidal numerical integration using SciPy `cumtrapz`.

```
current_active = cumtrapz(v_l3_active/H, dx= sampling_time, axis= 1)
current_idle = cumtrapz(v_l3_idle/H, dx= sampling_time, axis= 1)
```

The 30% tolerance of the inductor is accounted by calculating the minimum and maximum values for the currents.

```
pwr_active= np.outer(rms(current_active_max, axis=1),rms(v_c160_active,axis=1))
```

An outer product (because we have a two channel oscilloscope the voltage drop on L3 and on C160 are measured in different moments and thus are not correlated) is used to compute the active power which will then be a matrix. Finally the power difference between the measurements made when the cipher is encrypting and those made when it is in idle is computed and the mean is taken:

```
pwr_diff = np.mean(pwr_active - pwr_idle)
```

This power difference ideally should be equal to the power dissipation due to our design encryption process and thus must be roughly the same as the logic power figure given by the Xilinx power Estimator.

5 Design Reuse

Our entire project has been made open source and is available on GitHub. We have included our full Vivado projects directories for all the ciphers, a completely separate project for each different implementation, and also oscilloscope acquisitions .csv files and the Python code available via Jupyter Notebooks. The code has been widely commented and we have also inserted readme files to document further our ciphers implementations. We hope that this work could be useful not only for research but also for more practical, more contingent, purposes.

Because of this we have also made an example of how, one of our implementations, our AES 128/128 design, can be made AXI-stream compatible with minor modifications. This also extends to the other ciphers we have implemented. We have chosen the AES as an example because it is the only which, as of today, is an accepted standard and thus it is the one with most practical use. The other ciphers, even if not standardized yet, can be used already in proprietary applications. The full Vivado project for this example is also available in the repository.

6 Results

In this Section we will show the results we have obtained for our FPGA implementations of the ciphers and we will compare these results with those available in literature, which as it has been said, are relative to ASIC implementations. Vivado 2018.1 was used for all our designs and a target frequency of 125 MHz was chosen. As for the synthesis and implementations strategies we have chosen to leave them to the default settings, to not influence the results (as the optimizations algorithms can work better for a cipher and worse for another, biasing the result). As for the implementation per se, there weren't significant challenges as the most challenging part of this project was actually the design itself, especially for serialized version of the ciphers it required a deep understanding of the algorithms and clever optimizations of the resources. Another challenge was also trying to make the ciphers as homogeneous as possible in the control logic in order to not biasing the comparison.

Our implementation and simulation results for all our designs are reported in Table 6.1. Aside from common figures such as FPGA resources (Slices, Flip Flops, Look-Up-Tables) we are reporting the dynamic power, throughput and two Figure-Of-Merit (FOM) which are widely used in literature: Efficiency and Energy/bit. The power figures have been obtained through Xilinx Power Estimator and Vivado power estimator using .saif files for the switching activity obtained with Post-Implementation Timing Simulations in Vivado. Efficiency being Mbps/slices is self-explanatory: the lower the area and the higher the throughput, the more a design will be efficient. Energy per bit, instead, refers to the energy required for each ciphertext bit, the lower this figure, the more efficient is the cipher. As the energy is the product between total power (clocking + logical) and the cipher latency (time employed to perform one encryption); again the lower the latency (and thus the higher the throughput) and the lower the power the better a cipher will be performance-wise. As for clock cycles reported in the Table, clock cycles necessary for resetting, loading the cipher and outputting the ciphertext are accounted. We have not included in the power figures static power because, as it has already been said, it is not reflective of the design logic in FPGAs and we will instead use directly the slices occupied by the design as an equivalent metric to allow a comparison with GE area used in ASICs, where, on the contrary, area (and thus the design) reflects static power consumption.
In Table 6.2 are reported some results which can be found in literature for the same ciphers. The implementations are relative to ASICs because, as it already been said, in literature implementations are compared in ASICs even if this has some serious drawbacks as we have previously said in the Introduction. Those implementations are all relative to different cell libraries and fabrication processes but with same 180 nm process technology. Moreover the power figures from those ASIC implementations are derived differently each time. For some the authors have estimated the power from simulations for others the power is directly calculated from the Area in GEs. Actual experimental power consumption figures in literature are never provided. Power is relative to a clock frequency of 100 kHz which is the standard in literature and in the table is comprehensive of both clock and logic. Also the entry for clock cycles reported in Table 6.2 refer only to the encryption phase, as implementations in literature have often bare-minimum or no control logic and thus loading phase, for example, in this case is not included.

Table 6.1: Simulation results on Zybo FPGA.

Cipher	Size	Data-path	Slices	FFs	LUT		Clock Cycles	Thp (Mbps)	Power (mW)		Efficiency (Mbps/slices)	Energy/bit (pJ/bit)
					Logic	Mem			Clk	Logic		
Skinny	64/64	4	37	73	105	20	834	9.59	2.00	1.83	0.26	399.28
		64	38	142	98	-	35	228.57	2.61	1.77	6.02	19.16
	64/128	4	61	110	152	24	1545	5.17	2.47	2.02	0.08	867.13
		64	52	206	123	-	40	200.0	3.41	3.81	3.85	36.10
	128/128	8	73	125	186	20	1034	15.47	2.85	3.32	0.21	398.74
		128	85	270	238	-	44	363.64	4.09	7.78	4.28	32.64
Simeck	32/64	1	18	39	40	9	613	6.53	1.30	0.57	0.36	286.58
		16	31	98	73	4	42	95.23	2.06	1.29	3.07	35.18
	48/96	1	19	41	48	10	1013	5.92	1.31	0.57	0.31	317.41
		24	31	110	86	12	46	130.43	2.47	1.72	4.21	32.12
	64/128	1	18	40	44	10	1605	4.98	1.30	0.57	0.28	375.17
		32	50	181	122	10	54	148.1	3.33	2.60	2.96	40.03
Simon	32/64	1	16	42	43	8	613	6.53	1.30	0.60	0.41	291.18
		16	31	109	76	-	42	95.23	2.21	1.41	3.07	38.01
	48/96	1	18	44	50	8	1013	5.92	1.31	0.64	0.33	329.23
		24	41	159	107	-	46	130.43	2.81	2.30	3.18	39.18
	64/128	1	26	46	50	9	1605	4.98	1.31	0.60	0.19	383.19
		32	56	209	133	-	54	148.1	3.43	3.09	2.65	44.01
	128/128	1	22	44	56	11	4613	3.46	1.31	0.61	0.16	553.56
		64	97	272	198	-	77	207.79	4.11	4.78	2.14	42.78
AES	128/128	128	362	279	1296	-	20	800	4.25	18.35	2.21	28.25

Table 6.2: Implementations results found in literature

Cipher	Size	Data-path	GE	Clock Cycles	Thp (kbps)	Power (uW)	Efficiency (kbps/GE)	Energy/bit (pJ/bit)
Skinny [12]	64/64	4	988	704	9.09	2.47	9.20	271.70
		64	1223	32	200	3.06	163.53	15.29
	64/128	4	1399	788	8.12	3.50	5.81	430.63
		64	1696	36	177.78	4.24	104.82	23.85
	128/128	8	1840	872	14.68	4.60	7.98	313.38
		128	2391	64	320	5.98	133.84	29.89
Simeck [13]	32/64	1	505	571	5.60	1.26	11.09	225.45
		16	695	36	88.90	1.74	127.91	19.54
	48/96	1	715	960	5	1.79	6.99	357.50
		24	1027	40	120	2.57	116.85	21.40
	64/128	1	924	1524	4.20	2.31	4.55	550.00
		32	1365	48	133.30	3.41	97.66	25.60
Simon [14]	32/64	1	523	571	5.60	1.31	10.71	233.48
		16	722	36	88.90	1.81	123.13	20.30
	48/96	1	739	960	5.00	1.85	6.77	369.50
		24	1055	40	120	2.638	113.74	21.979
	64/128	1	958	1524	4.20	2.40	4.38	570.24
		32	1417	48	133.33	3.54	94.09	26.57
	128/128	1	1234	4414	2.90	3.09	2.35	1063.79
		64	2090	70	182.90	5.23	87.51	28.57
AES [15]	128/128	128	12405	10	1280	31.01	103.18	24.23

As it can be seen from the comparison between the two Tables 6.1 and 6.2, the results found are comparable. In fact if the difference between the clock frequencies of more than three orders of magnitude between our implementations and those in literature is accounted (dynamic power is roughly linear with frequency as espressed in Equation 2.1) the results are similar. More importantly, what matters, is that the relationships between different ciphers is preserved in terms of power and other FOMs. In other words the platform along with the design methodology chosen, allow a meaningful comparison of the ciphers in with all the advantages mentioned in the Introduction.

For example, it is clear that from both Tables, aside AES, a parallel implementation of SKINNY 128/128 has the highest power consumption but achieves also high efficiency and low energy per bit figures because it needs fewer rounds for the encryption process and thus has an high throughput. While the Simon 128/128 and Simeck 64/128 instead due to the fact that need more rounds are less efficient and require more energy per bit. Nonetheless the power consumption for both these ciphers is lower and thus can be the preferred choice for applications (such as passive RFIDs) where the power consumption is the main requirement. For other, more common, applications, which include battery-powered sensors for example, energy is usually the main concern and thus SKINNY should be preferable. Also from both Tables it is clear that Simeck performs slightly better than Simon as it has slightly lower power consumption while other parameters (such as the throughput) are the same. This is because the Simeck employs a simpler key scheduling algorithm. The AES defends itself well in this comparison because of his extremely high throughput but its power consumption is several times higher than the highest found in its lightweight competitors.

Finally for what concerns, serialized implementations, again these trade-off lower power consumption at the expense of lower throughput and higher latency. Thus these implementations are less energy efficient and are in fact preferable in applications such as passive RFIDs.

Based on these results we performed experimental measurements on the Zybo board to obtain the dynamic power figure and validate experimentally some of the data in Table 6.1 obtained through simulations. We have chosen to measure the dynamic power consumption with the method described in preceding Sections for some ciphers. We have chosen the 128-bit key versions because these are the ones suitable for the upcoming NIST AEAD call. The experimental figures for dynamic power consumption are in Table 6.3. Again as it can be seen our measurements are in agreement with the figures we got from simulation.

Table 6.3: Measured dynamic logic power.

Cipher	**Block Size**	**Key Size**	**Datapath**	**Measured Logic Power (mW)**
AES	128	128	128	14 ± 0.43
SKINNY	128	128	128	5.4 ± 0.16
SIMON	128	128	64	
SIMECK	64	128	32	

7 Conclusion

In this work we have showed how a fair comparison of lightweight cryptography algorithms can be made using a Zybo Zynq-7000 ARM/FPGA board; an approach which has several advantages to existing ones in literature based on ASIC libraries.

We implemented and compared 21 different designs relative to four different block ciphers: the Advanced Encryption Standard and other three promising lightweight block ciphers suitable for the upcoming NIST lightweight cryptography call. In particular, we made our best efforts to optimize each design for minimum resource utilization and at the same time keep an homogeneous structure and control logic for the designs in order to not biasing the comparison.

Critical figures of merit such as power consumption, resources utilization, Efficiency and Energy per bit have been obtained through simulation and compared to the ones obtained from literature. From this comparison it emerges that our work allows for equivalent results without the need for ASIC libraries and tools and with the advantage of having a design implemented on a physical board on which real-world measurements can be performed.

For what regards dynamic power consumption we made some simulation with the Vivado Power Estimator tool, but we have also obtained real-world measurements on the board. This required a particular approach as our board doesn't allow for easy current probing without modifications. Nonetheless, we obtained real world data which is in agreement with our simulated values.

Moreover, although our project was mainly conceived for research purposes, design reuse was also taken in consideration and because of this, we provided an example of how one of our implementations (AES-128/128) can be adapted for AXI-Stream.

Future works include performing Differential Power Analysis and other Side Channel Attacks on our implementations on the board and also investigating high throughput fully-unrolled implementations of the ciphers.

References

[1] Draft submission requirements and evaluation criteria for the lightweight cryptography standardization process. `https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/Draft-LWC-Submission-Requirements-April2018.pdf`. Accessed: 07-06-2018.

[2] Larry Bassham, Çağdaş Çalık, Kerry McKay, Nicky Mouha, and Meltem Sönmez Turan. Profiles for the lightweight cryptography standardization process. 2017.

[3] Daniel J Bernstein and Tanja Lange. ebacs: Ecrypt benchmarking of cryptographic systems. 2009.

[4] Daniel Dinu, Alex Biryukov, Johann Großschädl, Dmitry Khovratovich, YL Corre, and Léo Perrin. Felics–fair evaluation of lightweight cryptographic systems. In *NIST Workshop on Lightweight Cryptography*, volume 128, 2015.

[5] Submission requirements and evaluation criteria for the post -quantum cryptography standardization process. `https://csrc.nist.gov/csrc/media/projects/post-quantum-cryptography/documents/call-for-proposals-final-dec-2016.pdf`. Accessed: 07-06-2018.

[6] Ray Beaulieu, Stefan Treatman-Clark, Douglas Shors, Bryan Weeks, Jason Smith, and Louis Wingers. The simon and speck lightweight block ciphers. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2015.

[7] Gangqiang Yang, Bo Zhu, Valentin Suder, Mark D Aagaard, and Guang Gong. The simeck family of lightweight block ciphers. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 307–329. Springer, 2015.

[8] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The skinny family of block ciphers and its low-latency variant mantis. In *Annual Cryptology Conference*, pages 123–153. Springer, 2016.

[9] Digilent. Zybo schematic. `https://reference.digilentinc.com/_media/zybo:zybo_sch.pdf`. Accessed: 07-06-2018.

[10] Snr5040e-1r5n. `http://pdf.datasheet.live/datasheets-1/3l_electronic/SNR5040E-470M.pdf`. Accessed: 07-06-2018.

[11] David Canright. A very compact s-box for aes. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 441–455. Springer, 2005.

[12] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The skinny family of block ciphers and its low-latency variant mantis. Cryptology ePrint Archive, Report 2016/660, 2016. `https://eprint.iacr.org/2016/660`.

[13] Gangqiang Yang, Bo Zhu, Valentin Suder, Mark D. Aagaard, and Guang Gong. The simeck family of lightweight block ciphers. Cryptology ePrint Archive, Report 2015/612, 2015. `https://eprint.iacr.org/2015/612`.

[14] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The simon and speck families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. `https://eprint.iacr.org/2013/404`.

[15] Lejla Batina, Amitabh Das, Baris Ege, Elif Bilge Kavun, Nele Mentens, Christof Paar, Ingrid Verbauwhede, and Tolga Yalcin. Dietary recommendations for lightweight block ciphers: Power, energy and area analysis of recently developed architectures. Cryptology ePrint Archive, Report 2013/753, 2013. `https://eprint.iacr.org/2013/753`.