HECTOR

D4.1
Demonstrator Specification

Project number:	ICT-644052
Project acronym:	**HECTOR**
Project title:	Hardware Enabled Crypto and Randomness
Project Start Date:	1st March, 2015
Duration:	36 months
Programme:	H2020-ICT-2014-1
Deliverable Type:	Report
Reference Number:	ICT-644052 / D4.1/ 1.0
Workpackage:	WP4
Due Date:	April 2017 - M26 (shifted from M24)
Actual Submission Date:	3rd May, 2017
Responsible Organisation:	MIC
Editor:	Marek Laban
Dissemination Level:	PU
Revision:	1.0
Abstract:	This deliverable contains detailed software and hardware specifications of the HECTOR demonstrator platforms as an output of task T4.1. The specifications are derived from selected applicative scenarios defined in deliverable D1.2. In particular, they concern the implementation of true random number generators, physically unclonable functions, and authenticated encryption with associated data algorithms as the main HECTOR outcomes.
Keywords:	Demonstrator, true random number generator, physically unclonable function, authenticated encryption

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 644052.

Editor

Marek Laban (MIC)

Contributors (ordered according to beneficiary numbers)

Sandra Lattacher, Martin Deutschmann (TEC)
Vladimir Rožić, Bohan Yang, Dave Singelée (KUL)
Viktor Fischer, Ugo Mureddu (UJM)
Alexandre Anzala-Yamajako (TCS)
Filippo Melzani (STI)
Marcel Kleja, Marek Laban (MIC)
Maria Eichlseder (TUG)
Gerard van Battum, Marnix Wakker (BRT)

Disclaimer

Executive Summary

This deliverable specifies hardware and software requirements on HECTOR demonstrators designed in the framework of workpackage WP4. The aim of the demonstrators is to validate *True Random Number Generator*, *Physically Unclonable Function* and *Authenticated Encryption with Associated Data* developed during the HECTOR project and to illustrate their potential relevance for the real-world security.

In order to evaluate characteristics of the proposed algorithms and hardware blocks in application domains, which can differ considerably in their requirements, the following three applicative scenarios have been selected:

- *Demonstrator 1*: Stand-alone, AIS20/31 compliant, high performance secure TRNG;
- *Demonstrator 2*: Secure portable USB data storage – encrypted data storage with user and device authentication for protecting data at rest;
- *Demonstrator 3*: Secure messaging device – data terminal with a strong authentication protocol for sending encrypted authenticated messages via an insecure communication channel.

The demonstrators will be implemented on two hardware platforms. While Demonstrator 2 and 3 will be portable, small form-factor devices powered by the USB bus, Demonstrator 1 needing higher performance will be powered by an external supply. The three demonstrators will feature the USB interface to be connected to the host computer and a metallic case ensuring an electric shielding and tamper evidence.

The specifications of demonstrators and the scope of their security evaluation is determined by their future application in practice.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This deliverable contains software and hardware specifications of the HECTOR demonstrator platforms as an outcome of Task 4.1. The HECTOR hardware-software demonstration platforms were chosen depending on outputs from work packages WP2 and WP3 as defined in Deliverable 1.2. According to the HECTOR Grant Agreement, the aim of the work package WP4 is to demonstrate the validity of the new approaches, building blocks and cryptographic primitives in common data security applications.
In the framework of WP2, the *True Random Number Generator (TRNG)* and *Physically Unclonable Function (PUF)* were proposed. The *Authenticated Encryption with Associated Data (AEAD)* algorithm is the outcome of WP3. The TRNG can be used either for generation of encryption keys or for providing general purpose random bit streams. The PUF can be adopted for device authentication or secure storage of a key, bounded to a particular device. The AEAD ensures data confidentiality, integrity and authenticity.
To illustrate the flexibility of the proposed building blocks, the demonstration of HECTOR outcomes have been divided into three usage scenarios – demonstrators. Following the project proposal, the demonstrators will showcase at least one instantiation of the above mentioned building blocks. Demonstrator 1 will feature a highly secured TRNG, according to the state of the art security requirements. Demonstrator 2 and 3 will leverage all three proposed building blocks, each of them in its own usage scenario.
The hardware architectures of the demonstration platforms reflect the required optimization towards application needs. To ease the integration, most of the building blocks of WP2 and WP3 have been developed on the HECTOR evaluation platform hardware, which is close to that of demonstrators. However, some integration tasks go beyond the scope of the work packages of the HECTOR project, e.g. the authentication protocol in section 3.2.1.
The deliverable is divided into three main chapters, each dedicated to one demonstrator. Each chapter starts describing the motivations behind the particular demonstrator. It continues determining its parameters and operation, defining the building blocks and specifying the underlying hardware. Finally, it delimits the scope of the security evaluation of the demonstrator. The whole document is completed by an extra chapter on general aspects of security evaluation applicable to the demonstrators.

Chapter 2

Demonstrator 1: Standalone, High Performance Secure Random Number Generation Device

2.1 Motivation

The aim of Demonstrator 1 is to show that the proposed TRNG designs and especially the new TRNG design approach can be successfully applied in high-end real-world data security applications, in which a secure and robust TRNG is an essential construction block. The proposed demonstrator could be used as a stand-alone security peripheral – the source of high quality random numbers – in high-performance data servers and communication systems.

2.2 Functional Description

Demonstrator 1 is a physical true random generator including cryptographic post-processing, which is aimed at generation of random bitstreams achieving security level PTG.3, as defined in AIS 20/31. The generation of random numbers is provided as a security service for the user. Two kinds of sources of randomness and associated random number generators exploiting these sources are demonstrated to be compliant with AIS 20/31 in the two operational modes of the device:

- PLL-based TRNG,
- Delay chain-based TRNG.

The PLL-based TRNG takes advantage of the physical isolation of the phase-locked loop (PLL) from the rest of the logic device. This physical isolation ensures higher independence of the source of randomness from the surrounding logic area. Another advantage of the PLL-TRNG is the repeatability of the design independently of device family and technology.
The delay chain-based TRNG takes advantage of the high-speed carry-chain primitives. These primitives are widely available on most FPGA devices from different vendors. These high-speed primitives are used to sample the jittery signals with high resolution. This enables the entropy extraction without the requirement for a high jitter accumulation time, which leads to a high-throughput TRNG.
In both operational modes, the quality of the generated numbers is tested using embedded statistical tests (the total failure test and online tests), which run continuously. Consequently,

the statistical test does not need to be launched manually.
Besides embedded statistical testing of the source of randomness, the deterministic part of the generator is tested at startup and after each reset using a known answer test (KAT).
The demonstrator has two functional modes of operation: a user mode and an evaluation mode. In the *user mode*, at each request, the user can:

1. Read up to 32 MB of internal random numbers (a cryptographically post-processed random bit stream),
2. Determine (read) the state of the generator,
3. Reset the generator.

In the *evaluation mode*, the evaluator (tester) can:

1. Read up to 32 MB of internal random numbers,
2. Determine (read) the state of the generator,
3. Reset the generator,
4. Read up to 32 MB of raw random numbers for testing purposes (according to AIS20/31),
5. Read up to 32 MB of internal data (depending on the type of generator), aimed at characterization of the source of randomness.

2.3 Building Blocks

Demonstrator 1 demonstrates the suitability of two types of HECTOR building blocks for high end cryptographic applications:

- Physical true random number generator (PTRNG) generating the raw binary signal,
- Cryptographic post-processing using a cryptographically secure deterministic random number generator (DRNG) continuously seeded by the PTRNG.

We recall that our intention was to implement the generator including the cryptographic post-processing block on a hardware platform similar to that of HECTOR evaluation boards, in which three types of FPGA were used: Altera Cyclone V, Xilinx Spartan 6, and Microsemi SmartFusion2 FPGA.
Figure 2.1 depicts the functional diagram of Demonstrator 1. In order to illustrate wide applicability of our approach, the HECTOR consortium decided to implement two types of the raw random bitstream generators in two types of FPGAs – Altera Cyclone V and Xilinx Spartan 6 FPGA. A phase locked loop (PLL) based generator is implemented in Altera Cyclone V FPGA and a delay chain (DC) based generator is implemented in Xilinx Spartan 6 FPGA. The control unit and the cryptographic post-processing is common for both generators and it is implemented in the third FPGA device – Microsemi SmartFusion2 FPGA.

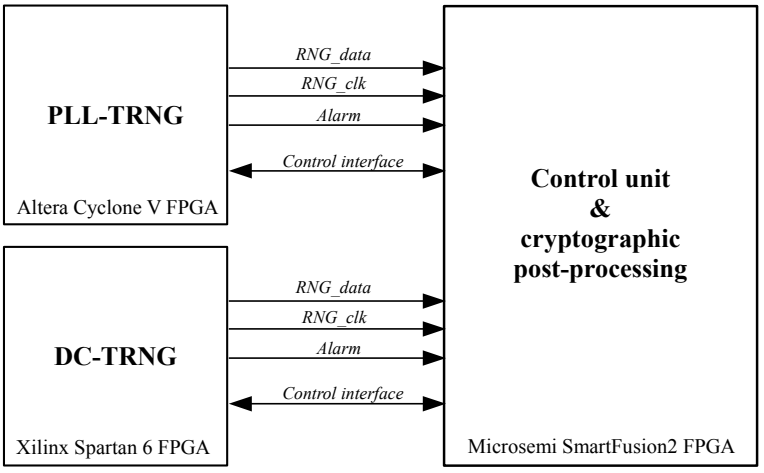

Figure 2.1: Functional diagram of Demonstrator 1

2.3.1 Secure PLL-based Generator of the Raw Random Bitstream

Principle

The TRNG exploiting the jitter introduced by the PLL, was first proposed in [10]. The PLL-based TRNG (PLL-TRNG) uses coherent sampling to generate a stream of random bits. The PLL plays two roles in the generator:

- The random jitter caused by electric noises inside the PLL serves as a source of randomness.
- The PLL guarantees the following relationship between its input and output frequencies:

$$f_1 = f_0 \cdot \frac{K_M}{K_D}, \tag{2.1}$$

  where K_M and K_D are multiplication and division factors of the PLL, respectively.

The block diagram of the PLL-TRNG is depicted in Fig. 2.2. The clock signal clk_1 is sampled in a D flip-flop (DFF) using the reference clock signal clk_0. The output of the flip-flop is decimated in the decimator, in which K_D samples (outputs of the DFF) are added modulo 2 to form one raw random bit at the output of the generator.
In this first configuration of the PLL-TRNG, the source of randomness is the tracking jitter of the PLL, i.e. the difference in phases between the reference clock (ideally jitter-free) and the jittery output clock of the PLL. Because of the PLL principle, the tracking jitter of the PLL is bounded and it depends on the jitter of the reference clock and the parameters of the PLL (the jitter of the voltage-controlled oscillator, the bandwidth of the filter and the dumping factor) [9].
Figure 2.3 depicts an example of input/output waveforms of the PLL-TRNG, in which the multiplication factor is $K_M = 5$ and the division factor is $K_D = 7$. It can be observed that the rising edges of the reference clock signal (clk_0) are placed in seven positions during one period $T_Q = K_D T_0 = K_M T_1$. In two of them, the rising edges of clk_0 appear when the sampled signal

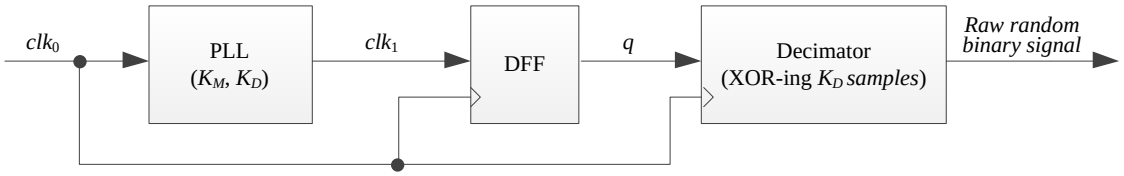

Figure 2.2: PLL-based TRNG

is equal to one (samples 3 and 6 taken in the first half of the sampled clock period). At the moment when two other rising edges occur, the sampled signal is equal to zero (samples 1 and 4 in the second half of the sampled period). At one rising edge of the reference clock the sampled clock rises from 0 to one (sample 0). Finally, the last two of the 7 samples belonging to the T_Q period appear close to the falling edge of the sampled signal (samples 2 and 5). The position of the seven samples repeats in all periods T_Q causing a pattern with few unstable bits at the DFF output.
The decimator from Fig. 2.2 can be seen as a one-bit counter counting bits equal to one during each period T_Q. The decimator value at the end of the T_Q period representing the TRNG output depends thus on the number of unstable (i.e. random) bits. However, we can remark that it also depends on the duty cycle of the sampled signal, which should remain stable.

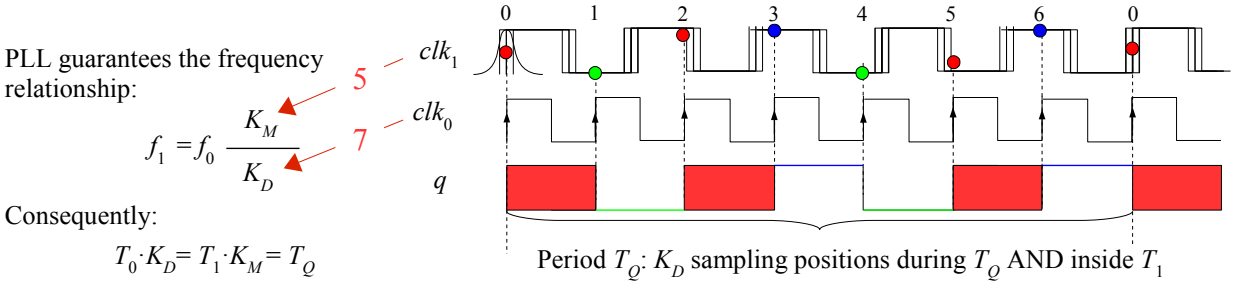

Figure 2.3: Example of the PLL-TRNG input/output waveforms

It was shown in [10] that if the standard deviation of the PLL output jitter (σ_{jit}) fulfills the following condition:

$$\sigma_{jit} > \text{MAX}(\Delta T_{min}), \tag{2.2}$$

at least one sample will be random during each period T_Q. The term $\text{MAX}(\Delta T_{min})$ in condition (2.2) represents the worst case (the longest) distance between the rising edges of the clock signal clk_0 and rising or falling edges of clk_1 during the T_Q period. It is given by

$$\text{MAX}(\Delta T_{min}) = \frac{T_0}{4K_M}\text{GCD}(2K_M, K_D) = \frac{T_1}{4K_D}\text{GCD}(2K_M, K_D), \tag{2.3}$$

where GCD means the Greatest Common Divisor.
As shown in [10], if K_M and K_D are relatively prime and K_D is odd, the TRNG output bit rate is $R = T_Q^{-1} = f_0/K_D$ and the sensitivity to jitter is $S = \Delta^{-1} = K_D/T_1$. The output bit rate and the sensitivity are closely related. Following relationships between parameters of the PLL-TRNG can be observed:

- to increase R and S, f_0 should be as high as possible,
- to increase R, K_D should be as low as possible,
- to increase S, K_M should be as high as possible.

From a One-PLL to a Two-PLL TRNG Design

In some technologies, condition (2.2) cannot be fulfilled using a single PLL. In this case, two PLLs connected in series or in parallel can be used to increase the bit rate and the sensitivity to jitter by increasing the multiplication and division factors (see the top panel and the bottom panel in Fig. 2.4, respectively). Although the effect of increasing the final multiplication and

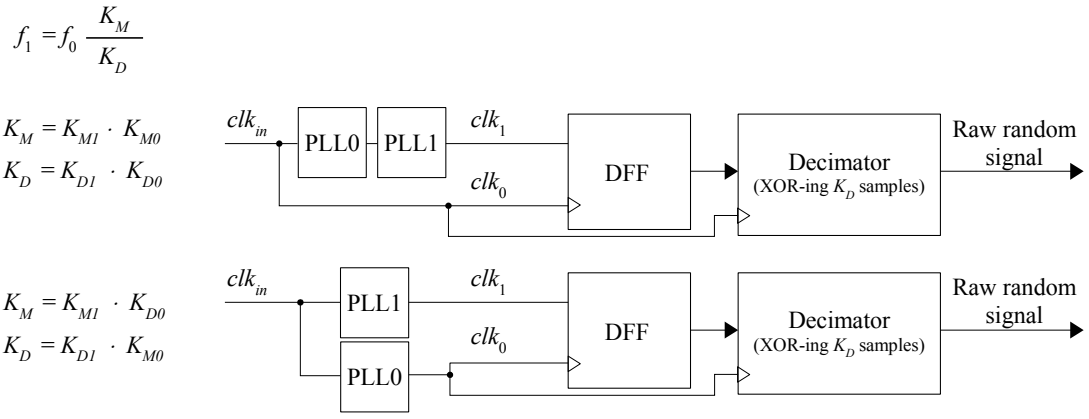

Figure 2.4: PLL-TRNG using two PLLs in series or in parallel

division factors is similar in both configurations of the PLL-TRNG, they differ significantly in the size of the exploited jitter [11]. In the cascaded connection of the two PPLs (top panel in Fig. 2.4), the jitter introduced by PLL0 is filtered out by PLL1. This is clearly not the case in the parallel configuration. Therefore, the parallel configuration of the PLL-TRNG is preferable and will be used in our design.

Measurement of the Tracking Jitter of the PLL

Thanks to the use of the PLL itself, the jitter at the PLL output, which is used as a source of randomness, can be easily quantified. The principle is presented in Fig. 2.5, which depicts the accumulation of reordered K_D samples obtained at the DFF output from Fig. 2.2.
We recall that $f_1 = f_0 \frac{K_M}{K_D} = f_0 \frac{5}{7}$, therefore $T_Q = T_0 \cdot K_D = T_1 \cdot K_M = 7T_0 = 5T_1$. To observe the jitter outside the device, we first acquire the bitstream obtained at the output of the flip-flop. Next, we reorder samples (bits) in the bitstream in each period T_Q following the formula:

$$j = (i \cdot K_M) \mathrm{mod} K_D. \tag{2.4}$$

Finally, we accumulate N times K_D samples in the memory (for illustration, $N = 2$ in Fig. 2.5, but in reality $N = 255$). It can be seen in the bottom panel of Fig. 2.5 that by reordering the samples using Eq. (2.4), we remove the pattern caused by the relationship between the frequencies of clk_0 and clk_1 and we can thus observe the form of the sampled signal (clk_1). The peak-to-peak size of the jitter observed during the time $t = N \cdot K_D \cdot T_0$ is proportional to the number of samples occurring at least once and less than N times (red dots in Fig. 2.5).

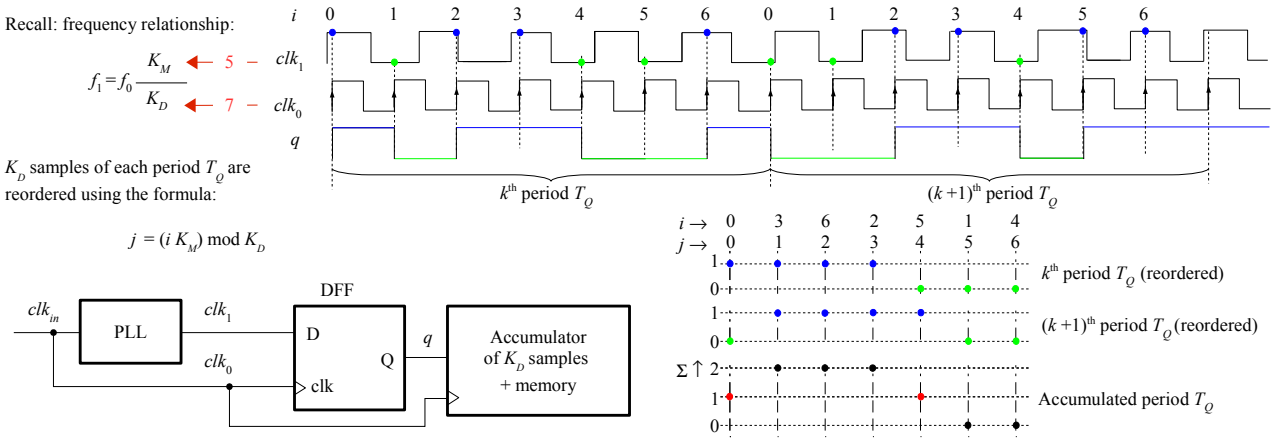

Figure 2.5: Principle enabling external observation of the jitter and its embedded measurement – number of unstable samples (red samples in the bottom panel) is proportional to the jitter

The main advantage of the method is that it can be easily implemented inside the logic device. While it is necessary that the PLL output is sampled internally, reordering, accumulation and processing of samples can be done outside the device. We will use this possibility to study the timing of clocks inside the device.

PLL-TRNG with Higher Entropy Rate

In this section, we propose a modified PLL-based RNG with a higher entropy rate. In the generator, k flip-flops sample k clock signals (k PLL outputs) having a phase difference of $180/k$ degrees. The outputs of flip-flops are XOR-ed and the obtained binary signal is used as an input signal for dedicated tests and for the decimator, which serves as a randomness harvesting block.
A simplified version of this kind of generator for $k = 2$ is presented in Fig. 2.6. The PLL has two clock outputs (clk_{10} and clk_{11}) having the same frequency and a phase difference of 90 degrees. Using k PLL output clock signals with different phases we increase the number of unstable samples and thus the entropy rate at the output. Note that the number of clocks must be chosen so that at any time, only one of k flip-flops will sample the input clock signal near its edge (rising or falling). This is quite easy to obtain.

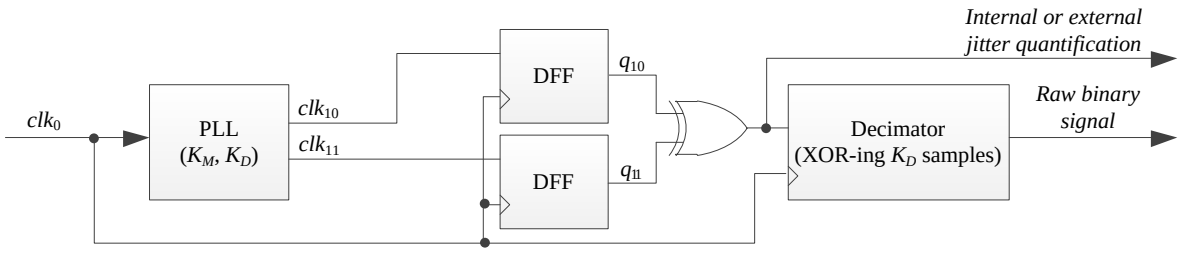

Figure 2.6: PLL-TRNG configuration with higher entropy rate and a possibility of external observation and internal quantification of the jitter (by using the output of the XOR gate)

HECTOR Demonstrator 1 PLL-TRNG Implemented in Cyclone V FPGA

The final version of the TRNG implemented in Demonstrator 1 in Cyclone V FPGA uses the differential jitter between two PLLs as a source of randomness. It is depicted in Fig. 2.7.

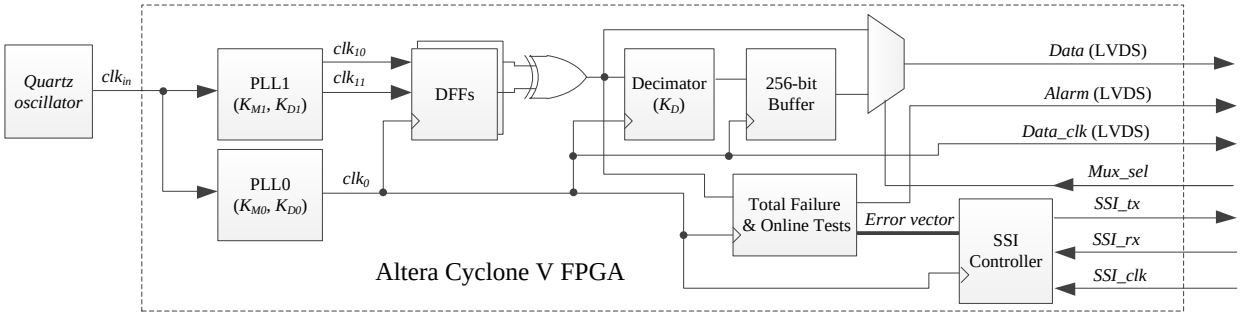

Figure 2.7: Block diagram of the PLL-TRNG design implemented in Cyclone V FPGA

The two PLLs are connected in parallel in order to increase the targeted differential jitter. PLL1 generates two clocks ($k = 2$) with a phase difference of $180/2 = 90$ degrees. These clock signals are sampled in two flip-flops on rising edges of the clk_0 clock signal. Outputs of flip-flops are XOR-ed together to get a decimator input signal. The decimator (a 1-bit counter) is incremented each time this signal is equal to one during K_D rising edges of the clk_0 clock signal (during one period T_Q). After each period T_Q the decimator value is saved in a 256-bit output buffer and the decimator is reset.
Since the coefficients of the two PLLs are set in such a way that the entropy rate per bit at decimator output is higher than required by the AIS 31 standard, the algorithmic post-processing is not needed.
The cryptographic post-processing is implemented in the Microsemi SmartFusion2 device, which controls Demonstrator 1 and which ensures data acquisition, post-processing and data interface with the host PC (not depicted in Fig. 2.7).
The error messages from the functional tests (e.g. PLLs not locked) and statistical tests (e.g. the total entropy failure and failure of Online tests) form an error vector, which is sent to the control interface of the RNG that can be reached from the HECTOR motherboard. If any of these tests fails, the *Alarm* is triggered, causing an interrupt condition in the control SoC.

PLL-TRNG Design Rationale

In this section, we present the design rationale of the proposed RNG, its blocks and functions.

Source of randomness The source of randomness in the proposed PLL-TRNG is the differential jitter (dynamic difference in phases) between clock signals generated in two PLLs connected in parallel as presented in Fig 2.7.
Figure 2.8 depicts the configuration of two PLL blocks as sources of randomness. PLL0 has multiplication and division factors $KM_0 = 37$, $KD_0 = 24$, respectively. PLL1 has multiplication and division factors $KM_1 = 19$, $KD_1 = 5$, respectively. The final multiplication and division factors of the PLL-TRNG are thus $KM = 456$, $KD = 185$. The reference clock frequency clk_0 is 192 MHz and the jittery clock frequency clk_1 is 475 MHz. This gives the bit rate of 1.04 Mbits/s at generator output, as required.

Both PLLs use asynchronous reset inputs. They can be restarted, for example, if the total failure alarm occurs. The PLL-TRNG output is allowed only if the two PLLs are locked. This is detected by the 'locked' PLL output signal. If the PLL locking fails, the corresponding error flag in the TRNG error vector is triggered.

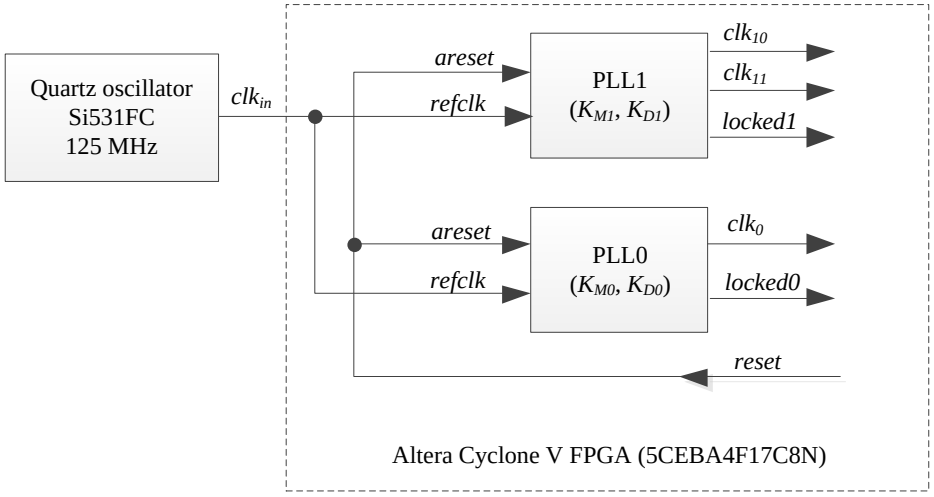

Figure 2.8: Configuration of two PLLs as sources of randomness

Digitization mechanism and generation of the raw random bitstream A simplified schematic diagram of the digitization mechanism and generation of random bitstream is depicted in Fig. 2.9. Two PLL1 output clock signals (clk_{10} and clk_{11}) shifted by 90 ° are sampled in the first couple of flip-flops (DFF_{00} to DFF_{01}) on the rising edges of the clock signal clk_0.

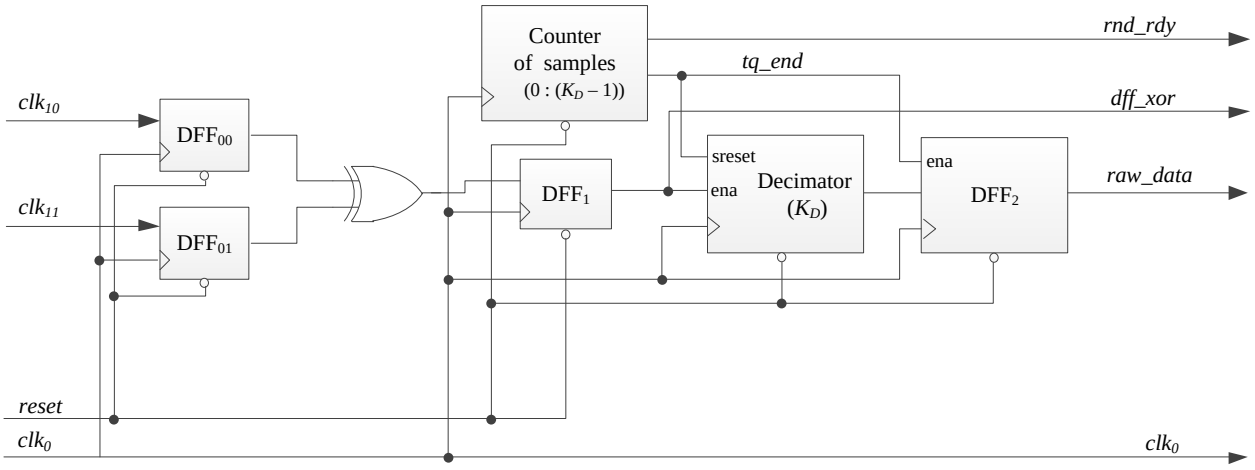

Figure 2.9: Schematics of the digitization mechanism

Post-processing of the raw random signal The entropy rate at RNG output depends on the size of the jitter and on the final size of multiplication and division factors (K_M and K_D, respectively). These factors are chosen in such a way, that the entropy rate per bit of the raw random binary signal (signal raw_data) is higher than 0.997 (the rate required by AIS 31 [13]). Therefore, any special post-processing algorithm is not needed and consequently not used. This value of the entropy rate is verified continuously and thus guaranteed by the online test.
For security reasons, the raw random binary signal is post-processed using a cryptographic post-processing algorithm of security level DRG.3. The architecture of the cryptographic post-processing block is described in Section 2.3.3.

Total failure test and online tests As explained at the beginning of the subsection, the source of randomness is the differential jitter between clock signals clk_0 and clk_{1i}. For the correct operation of the PLL-TRNG, the frequency ratio of the two clock signals generated by two PLLs must be maintained. The source of randomness can totally fail if any of the following conditions is fulfilled:

1. input clock clk_{in} from Fig. 2.8 is missing or not running (e.g. the oscillator does not oscillate),
2. at least one of two PLLs from Fig. 2.8 is not working properly (e.g. it is not locked) or not running at all,
3. one of the sampler stages (DFF_{0i}, the XOR gate, or DFF_1 from Figure 2.9) does not work properly.

If Condition 1 is fulfilled, Condition 2 is fulfilled, too. Therefore, Conditions 1 and 2 are verified by observing the *locked* signal of the two PLLs. If any of the sampler stages does not work properly, the decimator input does not have the expected behavior (it does not feature an almost regular pattern determined by the relationship between clock frequencies – the regular pattern should be impacted just randomly by a few of random samples), but it can:

1. be stuck to a constant value (one or zero),
2. feature a perfectly regular pattern without any random bits.

These two kinds of behavior will cause a total entropy failure of the PLL-TRNG and must be detected very fast.
The differential jitter can be evaluated internally based on the principle presented in Fig. 2.5 – the differential jitter and thus entropy rate is proportional to the number of unstable samples. The entropy rate per bit of the raw binary signal depends on the number of random events appearing during one period T_Q and on the position of random samples regarding the reference clock signal clk_0. If the jittery clock edge appears exactly on the rising edge of the reference clock, the corresponding sample features a maximum entropy.
Consequently, we propose to evaluate two parameters, P_1 and P_2:

- P_1 – number of accumulated samples with value $A_j \in (Tr_{min}; Tr_{max}) = (32; 224)$,
- $P_2 = \frac{1}{N^2} \sum_{j=0}^{K_D-1} A_j(N - A_j)$.

Note that the parameter P_1 is proportional to the size of the jitter (number of unstable samples during each period T_Q) and the parameter P_2 depends on the number of unstable samples *and their position* regarding the rising edge of the reference clock signal.
Figure 2.10 depicts the schematic diagram of the embedded statistical tests (Total failure and Online tests) dedicated to the PLL-TRNG, which are based on the computation of parameters P_1 and P_2 and their comparison with thresholds Tr_i determined from the stochastic model and the required size of the jitter needed to obtain sufficient entropy rate at generator output.
First, K_D samples of the *dff_xor* signal are accumulated in the embedded memory during N periods T_Q. Then, parameters P_1 and P_2 are computed from accumulated samples: a 10-bit counter counts how many of K_D samples are between thresholds Tr_{max} and Tr_{min} to obtain parameter P_1. To simplify computations, values Tr_{max} and Tr_{min} are chosen to be representable in 8 bits: $Tr_{max} = 7/8 \cdot 256 = 224_{(10)} = \text{E0}_{(16)}$ and $Tr_{min} = 1/8 \cdot 256 = 32_{(10)} = 20_{(16)}$. Finally, an 8x8-bit multiplier computes values $A_j(N - A_j)$, which are then accumulated in a 20-bit accumulator/adder to get parameter P_2.

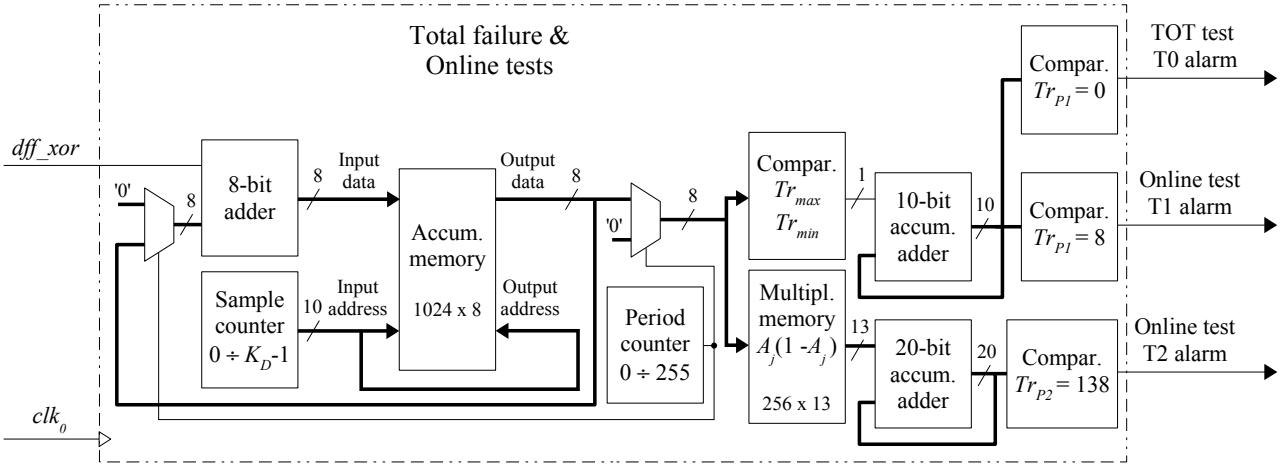

Figure 2.10: Schematic diagram of the PLL-TRNG dedicated embedded tests

If P_1 is equal to zero (i.e. no accumulated sample A_j is in interval $A_j \in (Tr_{min}; Tr_{max})$), the total failure alarm *T0* is triggered. If $P_1 < Tr_{P1} = 3$, the size of the jitter is not sufficient and the Online test alarm *T1* is triggered. Finally, if $P_2 < Tr_{P2} = 43068/256 = 168$, the entropy rate is not sufficient and the Online test alarm *T2* is triggered. Thresholds Tr_{P1} and Tr_{P2} are conservative values obtained from the stochastic model, which ensure minimum entropy rate per bit of 0.997 at the output of the TRNG.

Embedded tests and error messages The following security critical warnings (error messages) are sent by the PLL-TRNG core:

- *E0*: PLL0 not locked
- *E1*: PLL1 not locked
- *E2*: Entropy total failure ($P_1 = 0$)
- *E3*: Online test *T1* alarm ($P_1 < 3$)
- *E4*: Online test *T2* alarm ($P_2 < 168$)

Output interface TRNG control and output interface is divided in two parts (see Fig. 2.11):

- Fast serial data and alarm output,
- Slow serial control input/output.

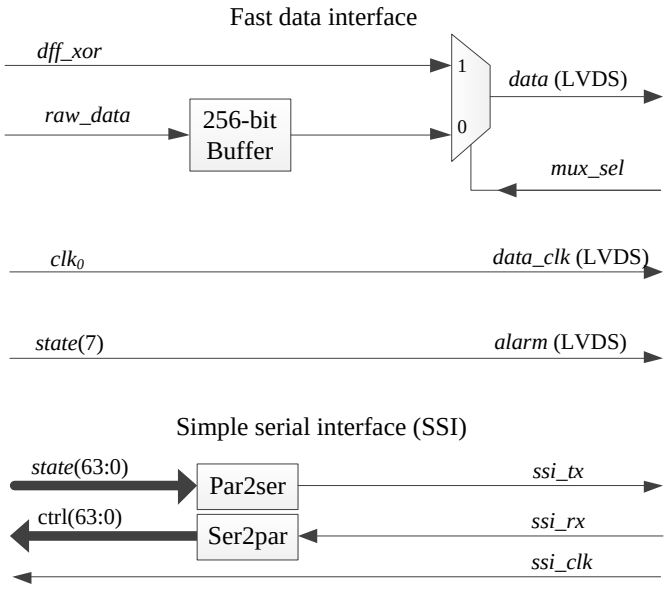

Figure 2.11: TRNG control and output interface

Two kinds of signals can be output via the fast data interface. To ensure high speed data transfers, these signals use the low voltage differential signaling (LVDS) technology:

- High-speed output of the DFF_1 flip-flop from Figure 2.9 (192.7 Mbits/s)
- Raw random bitstream (1.04 MB/s)

The high-speed output of the DFF_1 flip-flop (signal *dff_xor*) can be used to verify the duty cycle and the form of rising and falling edges of the sampled signal, and uniformity of distribution of samples.
The *alarm* signal can be used as interrupt request for the system on chip implemented in the control circuitry.

The TRNG is controlled using the synchronous serial interface (SSI) managed by the control device. The serial control input/output is clocked at 1 MHz by the *ssi_clk* signal. The application software can read the 64-bit state (*state*(63:0)) of the PLL-TRNG via the motherboard. The role of individual state bits is described in Table 2.1.

2.3.2 Secure DC-based Generator of the Raw Random Bitstream

Principle

The principle of a TRNG utilizing delay chains for entropy extraction was proposed in [17]. The Delay-Chain based TRNG (DC TRNG) uses jitter accumulated in a free-running ring oscillator as a source of randomness. The role of the delay chain is to sample the output of the ring oscillator using very high timing resolution (approximately 17 *ps* on a Xilinx Spartan-6

Table 2.1: PLL-TRNG state bits

Bits	Parameter	Description
$S(63:48)$	*Version*	16-bit version number
$S(47:5)$	*Reserved*	Not used in the current version
$S(4)$	*Error E4*	Online test T2 failure
$S(3)$	*Error E3*	Online test T1 failure
$S(2)$	*Error E2*	Total failure T0 alarm
$S(1)$	*Error E1*	PLL1 not locked
$S(0)$	*Error E0*	PLL0 not locked

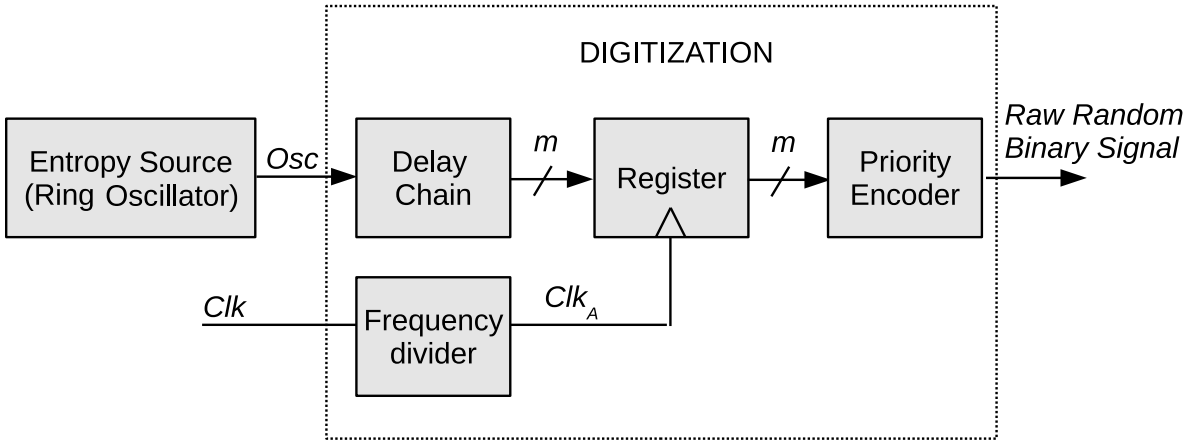

Figure 2.12: DC-based TRNG

FPGA). The data sampled by the delay chain is a digital representation of the waveform of the ring-oscillator output signal.
The block diagram of the DC TRNG is shown in Figure 2.12. The output *Osc* of the entropy source (free-running ring oscillator) is connected to a delay line consisting of m serially connected delay buffers. The output of each buffer connects to a flip flop. These m flip-flops contain a register that is sampled by a signal clk_A. This signal is obtained by down-sampling the reference clock signal provided by an on-board quartz oscillator. The down-sampling factor is a design parameter that is chosen to allow for a sufficient jitter accumulation time. The output of the register is connected to a priority encoder which encodes it into a single output bit.
The source of randomness in the DC-TRNG is the jitter accumulated in the free-running ring oscillator. Due to the white-noise sources that are present in all electronic circuits, the frequency of the free-running ring oscillator is not stable. Timing positions of the RO output signal edges, relative to the reference clock become more uncertain over time. The variance of this timing uncertainty is proportional to the jitter accumulation time (the period of the sampling signal clk_A).
Figure 2.13 shows an example of the waveform signals and the register data. At the rising edge of clk_A, the data from the delay chain are captured in the register. The register value corresponds to the *Osc* signal values in the previous half period. This time period contains at least one signal edge. The exact position of this edge is not predictable by any adversary because it is influenced by the noise in the oscillator. The most likely region of the signal edge is highlighted in gray and the distribution is shown at the bottom of Fig. 2.13. The exact position of this edge is captured in the register and encoded in the priority encoder. The neighboring positions are encoded using different bit values. The probability of generating a bit value 1 can be computed by integrating the parts of the distribution that are indicated in gray. Higher

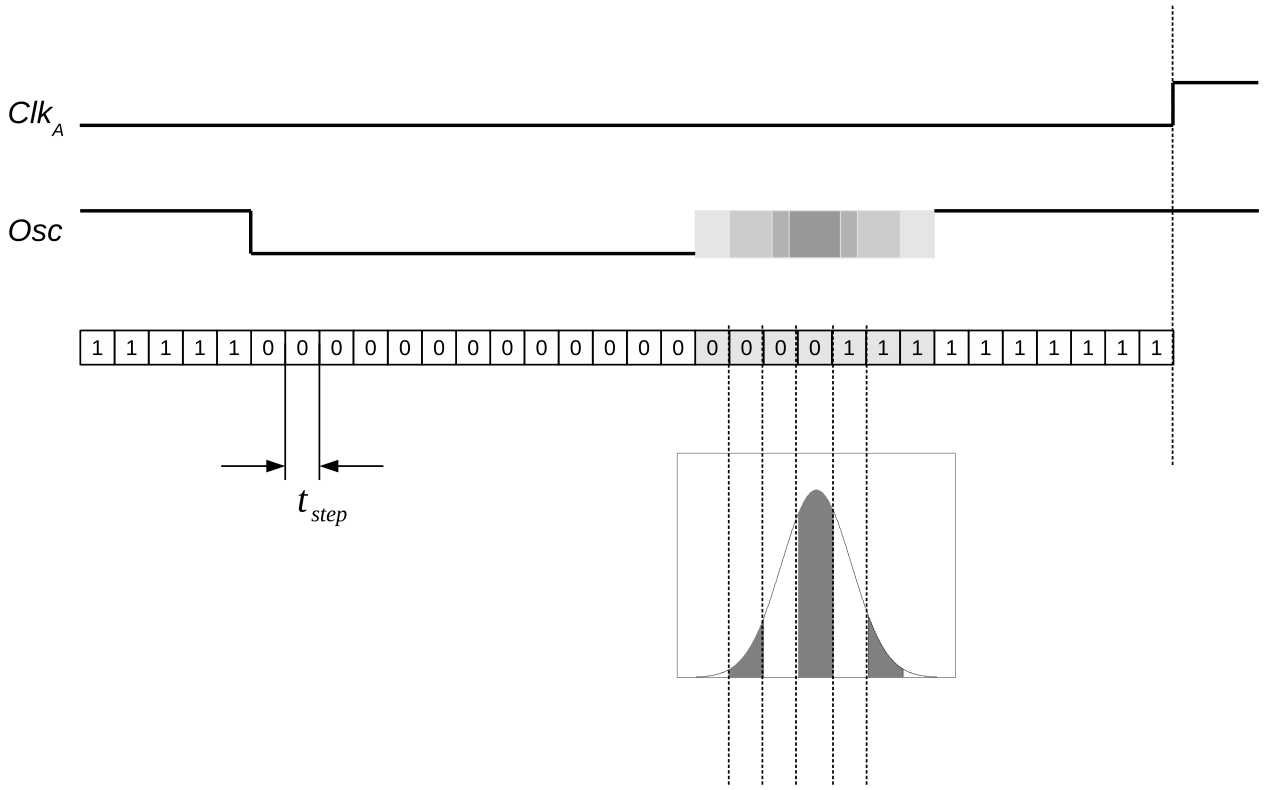

Figure 2.13: Example of the DC-TRNG waveforms of relevant signals.

variance of the accumulated jitter (achieved by allowing higher jitter accumulation time), as well as the higher timing resolution (lower t_{step}) both result in a more balanced probabilities of 0 and 1 and thereby in higher entropy.

Design parameters The design parameters of the DC-TRNG are:

- T_A – period of signal clk_A. This period is equal to the jitter accumulation time. Higher T_A results in higher entropy per bit of the raw data at the price of the reduced throughput.
- T_0 – period of the free-running ring oscillator.
- m – the number of elements in the delay chain. For a correct operation of the DC-TRNG it is essential that the condition

$$m > \frac{T_0}{2 \cdot t_{step}}, \tag{2.5}$$

is fulfilled. This condition guarantees that a signal edge is detected in every sample.

HECTOR Demonstrator 1 DC-TRNG Implemented in Spartan-6 FPGA

The implementation of DC-TRNG on Spartan-6 FGPA for Demonstrator 1 uses the jitter from a free-running ring oscillator as the source of randomness. The block diagram of the DC-TRNG design is shown in Fig. 2.14.
The ring oscillator is sampled by a tapped delay chain with a timing resolution around $17ps$. A system clock sourced by a quartz oscillator is used for circuit synchronization and triggering the sampling. The sampled result is converted to a raw binary bit using a priority encoder. Raw binary bits can be sent to the mother board directly for offline evaluations. Raw binary bits

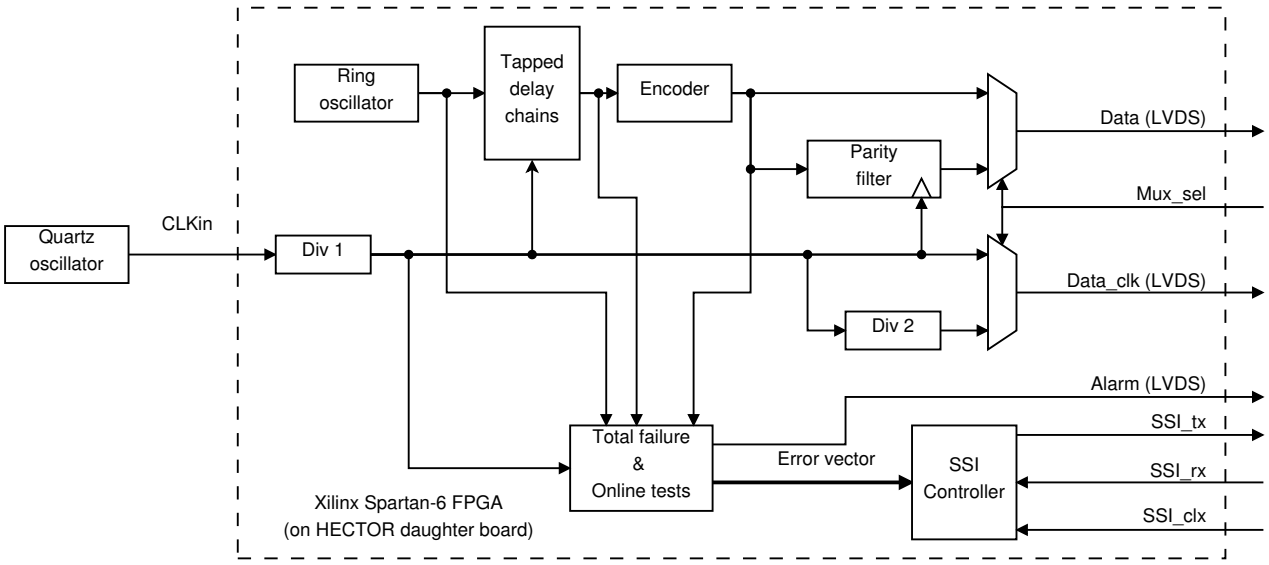

Figure 2.14: Block diagram of the DC-TRNG design

are compressed using a parity filter to improve entropy-per-bit. Outputs of the free-running oscillator and the tapped delay chain are evaluated for total failure tests. Two types of online tests are applied to the binary bits.
Two frequency dividers are used to generate different clock signals. The frequency divider 1 is chosen based on the jitter accumulation time. The divider 2 is used to generate data-clk signal when the post-processing is applied.

DC-TRNG Building Blocks

Source of randomness The source of randomness in DC-TRNG is the accumulated thermal jitter in the free-running ring oscillator implemented using a single look-up table (LUT), i.e. timing uncertainty of the ring oscillator signal edge position relative to the reference clock signal.

Digitization Mechanism and Generation of the Raw Random Bitstream Digitization part of DC-TRNG is composed of two parts:

- fast delay-chain line, which is connected to the output of the ring oscillator and performs time-to-digital conversion of the jittery signal edge.
- entropy extractor, which is used to detect the position of the jittery ring oscillator signal edge in the delay-chain line and to output one bit, corresponding to the parity of the delay-chain stage position.

Post-processing of the Raw Random Bits Raw random bits produced by the entropy extractor are statistically enhanced by using XOR post-processing, a compression technique that consists of xoring together (adding modulo 2) several consecutive raw random bits. The entropy rate of the raw random bits depends on the amount of accumulated jitter in the ring oscillator and the time step of the delay-chain (i.e. time step of time-to-digital conversion). Time step of the delay-chain depends on the platform on which DC-TRNG is implemented, while the amount of accumulated jitter can be increased (and consequently the entropy rate) by enabling oscillations of the ring oscillator for the longer period of time.

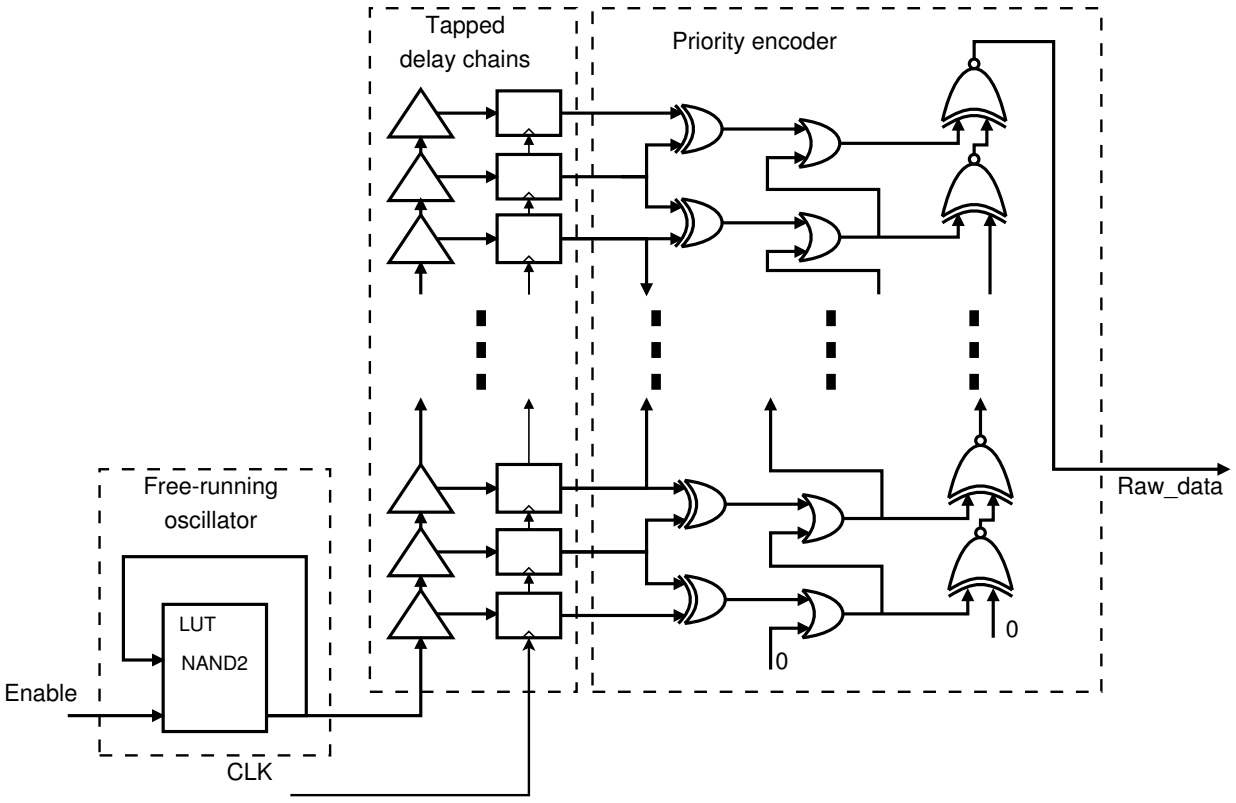

Figure 2.15: Schematics of the digital noise source.

Total Failure of the Source of Entropy and Its Detection In order to check whether the entropy source is totally break down, total failure tests are designed in this implementation. The source of randomness can totally fail if:

- the free-running ring oscillator is missing or not working,
- there is no edge sampled by the tapped delay chain.

We implement two total failure tests. The first total failure test is designed to detect whether the free-running ring oscillator is toggling or not. An edge detector is connected to the output of the ring oscillator. The edge detector is reset every two cycles. It triggers an alarm if the input signal has not changed.
The second total failure test is designed to generate an alarm if there is no edge sampled in the tapped delay chain. During normal operation, all taps cannot be all zeros nor all ones. Therefore, the equality of all captured values in the delay chain is used to generate the alarm signal.

Online Tests DC-TRNG contains two statistical tests that detect long-term weaknesses in the generated raw data bitstream. These two tests have different false-alarm rates. On-line test 1 (Sensitive test) has a higher false-alarm rate but is more efficient in detecting attacks. On line test 2 (Robust test) is less efficient in detecting attacks but it is more robust against false alarms.
On-line test 1 operates on a sequence of 512 consecutive raw bits. The test statistics is the count N_{111} of the template 111 in the sequence. This statistics is computed on-the-fly using a 9-bit counter. After every 512 bits, the counter value is compared with the pre-computed

(empirically determined) upper and lower boundaries. If the counter exceeds these boundaries, an alarm signal is generated. The false alarm rate of this test is 1%.
On-line test 2 also operates on a sequence of 512 consecutive raw bits. The test statistics C_1 is computed as follows:

$$C_1 = \sum_{i=1}^{511} b_i \oplus b_{i+1} \,, \tag{2.6}$$

where b_i denotes the i-th bit of the sequence. This statistics is computed using an xor gate and a 9-bit counter. After every 512 bits, value C_1 is computed and compared with the pre-computed boundaries. The alarm signal is generated if the counter value exceeds these boundaries. Afterwards, the counter is reset and the next 512 bits are tested. The false alarm rate of this test is 10^{-6}.

DC-TRNG Status Messages Similar to the PLL-TRNG, a SSI is used to control and check the status of DC-TRNG. The application software can read the 64-bit state (state(63:0)) of the DC-TRNG via the motherboard. The role of individual state bits is described in Table 2.2.

Table 2.2: DC-TRNG state bits

Bits	Parameter	Description
S(63:48)	Version	16-bit version number
S(47:4)	Reserved	Not used in the current version
S(3)	Error E3	On-line test 2 alarmed
S(2)	Error E2	On-line test 1 alarmed
S(1)	Error E1	No edge sampled in tapped delay chain
S(0)	Error E0	Ring oscillator stop resonating

2.3.3 Cryptographic Post-processing

Besides the algorithmic post-processing which is facultative depending on the reached entropy rate per output bit, the generator which should be compliant with AIS 31, level PTG.3, must post-process cryptographically the generated raw random numbers. The HECTOR consortium agreed upon using a pseudo-random number generator called here `CTR-DRBG` derived from the `CTR-DRBG` defined in [4] as a cryptographic post-processing of the PTRNG. The generator uses the approved algorithm `AES128` as a cryptographic primitive. This section will provide details about specification of the `CTR-DRBG` as a part of Demonstrator 1. The general construction of the DRNG adapts the approach described in [4] with the specific parameters choices outlined in Table 2.3. The different algorithms will be described in the following sections.

State of the DRNG & Internal Processes

Internal state S The internal state of the DRNG is shown in figure 2.1 and is composed of a value V updated every time 128 bits of output is produced, a key K updated by `Instantiate` or `Generate` calls.

$$S = \{0,1\}^{128} \times \{0,1\}^{128}, I = \{0,1\}^{256}, R = \{0,1\}^{t \cdot 128}$$

where t is the parameter of the `Generate` call

Security parameter (λ)	128
Block length (`blockLen`)	128
Counter field bit length (`ctrLen`)	32
Key length (`keyLen`)	128
Min. entropy	256
Seed length (`seedLen`)	256
Entropy input length	256 (`Instanciate` and `Generate`)
Personalization string length	0
Additional input length	0
Number of bits per requests	$t \cdot 128$, $t \in \mathbb{N}$
Max. number of requests between entropy introduction	1

Table 2.3: Parameters of the DRNG

Listing 2.1: DRNG internal state

```
typedef struct InternalState_t {
//V is a 128 bits value
        uint8_t V[16];
//K is a 128 bits value
        uint8_t K[16];
} DRNGInternalState_t;
```

Internal state Update function $\varphi : S \times I \rightarrow S$ The `Update` process updates the internal state of the DRNG by mixing in the data presented as input. The detailed process is outlined in algorithm 1. φ is one way even if D is known by the adversary as the updated internal state

Algorithm 1: `Update` process

Input: $K, |K| = 128, V, |V| = 128, D, |D| = 256$.
Result: updated values for K and V
begin
$V_{fixed} \longleftarrow V_0 \dots V_{95}$
$V_{ctr} \longleftarrow V_{96} \dots V_{127}$
$B_0 \longleftarrow D_0 \dots D_{127}$
$B_1 \longleftarrow D_{128} \dots D_{255}$
$V \longleftarrow B_1 \oplus AES128_K\left(V_{fixed} || \left(V_{ctr} + 2 \pmod{2^{32}}\right)\right)$
$K \longleftarrow B_0 \oplus AES128_K\left(V_{fixed} || \left(V_{ctr} + 1 \pmod{2^{32}}\right)\right)$
return (K, V)

is indistinguishable from a random string for an adversary unable to attack AES.

Instantiating the DRNG

The `Instantiate` process initiates the DRNG before any data can be extracted from it. The detailed process is outlined in algorithm 2.

Algorithm 2: `Instantiate` process

Result: $status$, (K, V)
begin
  $(status, entropyInput) \longleftarrow TRNG(128)$
  if $status \neq SUCCESS$ **then**
    return $status, (0^{128}, 0^{128}, 0)$
  $(status, N) \longleftarrow TRNG(128)$
  if $status \neq SUCCESS$ **then**
    return $(status, 0^{128}, 0^{128}, 0)$
  $S \longleftarrow entropyInput||N$
  $(K, V) \longleftarrow Update(0^{128}, 0^{128}, S)$
  return *(SUCCESS,K,V)*

Uninstantiating the DRNG

The `Uninstantiate` process clears the DRNG state – no data should be requested from the DRNG after the process has been called. The detailed process is outlined in algorithm 3.

Algorithm 3: `Uninstantiate` process

Input: $K, |K| = 128, V, |V| = 128$
Result: Cleared (K, V)
begin
  $K \longleftarrow 0^{128}$
  $V \longleftarrow 0^{128}$
  return (K, V)

Generating Pseudorandom Bits Using the DRNG

The `Generate` process produces pseudo random bits and updates the state of the DRNG. For efficiency reasons, the generate process only produces multiples of 128 bits of pseudo random bits. The detailed process is outlined in algorithm 4. ψ is one way as B is indistinguishable from a random string for an adversary unable to attack AES.
p_A is the distribution of the internal state after instantiation or generation and it depends on the 256 bits of seed introduced through the entropy source. The internal state of this DRNG meets the necessary entropy condition to resist high-potential attacks.

Conformance with Requirements of the DRG.3 class

In this section we will show that the instantiation of `CTR-DRBG` as defined above is conformant to the requirements of the RNG class DRG.3 defined in the AIS 20/31 [13]. The reader can match the different processes described above to the figure 2.16 to follow the justifications below.

DRG.3.1 The internal state of our instanciation of `CTR-DRBG` RNG requires a computational effort equivalent to 2^{256} elementary operations to determine the state and uses a PTRNG of class PTG.2 as random source.
Guessing the future pair K, V is sufficient to generate outputs.

Algorithm 4: `Generate` process

Input: $K, |K| = 128, V, |V| = 128, t$.
Result: $status$, B, updated (K, V)
begin
  $(status, S) \longleftarrow TRNG(256)$
  if $status \neq SUCCESS$ **then**
    return $status, 0^{t \cdot 128}, (K, V)$
  $(K, V) \longleftarrow Update(K, V, S)$
  for $i \in \{0, \ldots, t-1\}$ **do**
    $V_{fixed} \longleftarrow V_0 \ldots V_{95}$
    $V_{ctr} \longleftarrow V_{96} \ldots V_{127}$
    $B_i \longleftarrow AES128_K\,(V_{fixed} || \,(V_{ctr} + i \pmod{2^{32}}))$
  $(K, V) \longleftarrow Update(K, V, 0^{256})$
  $status \longleftarrow SUCCESS$
  return $status, B, (K, V)$

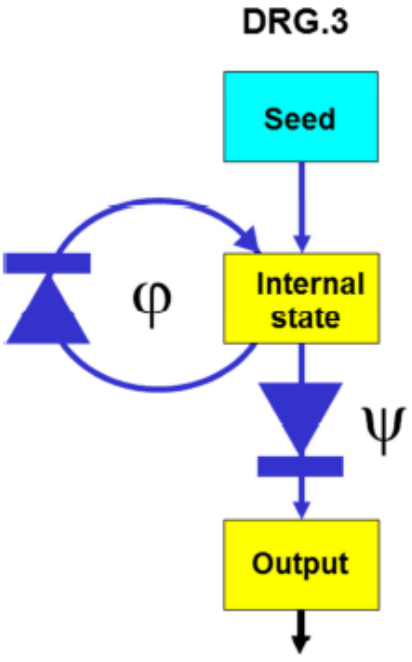

Figure 2.16: DRG.3 RNG class (from [13])

DRG.3.2 The RNG provides forward secrecy.
Predicting future outputs of the RNG requires to predict the outputs of the entropy source.

DRG.3.3 The RNG provides backward secrecy even if the current internal state is known.
The call to `Update` *at the end of the* `Generate` *process prevents that a compromise in the future could reveal information about past outputs.*

DRG.3.4 The RNG generates output for which 2 strings of bit length 128 are mutually different with overwhelming probability.
Within a call to the `Generate` *processes, the probability is actually 1. Across calls to this*

process and using classical assumptions on AES the probability of collisions grows as the one of a random mapping.

DRG.3.5 Statistical test suites cannot practically distinguish the random numbers from output sequences of an ideal RNG. The random numbers must pass test procedure A.
Those tests will be performed during the evaluation of Demonstrator 1.

2.4 Hardware Platform

The design of the Demonstrator 1 hardware platform is based on the one of the HECTOR evaluation platform, which uses one FPGA to implement the control unit and another one to implement the source of the raw random signal. In comparison with the HECTOR evaluation platform the Demonstrator 1 platform features also the following characteristics:

- two different FPGAs are used to implement two different PTRNG,
- the whole hardware platform is properly shielded,
- the platform includes a real-time counter (RTC) device,
- the platform includes additional visualisation elements.

One of the advantages of the proposed design is its easy migration from the HECTOR evaluation platform to the Demonstrator 1 platform due to similarity of both hardware platforms. Two different FPGAs are used to implement the sources of raw random bitstreams. The first one is Altera Cyclone V and the second one is Xilinx Spartan 6 FPGA. Both generators are connected to the main control unit using the same data interface as the HECTOR evaluation platform: both use four LVDS and three single-ended wires.
Demonstrator 1 features external synchronous 512 Mb (64 MB) low-power DDR SDRAM memory. It provides sufficient space for saving huge TRNG data in real time.
Contrary to HECTOR evaluation boards, Demonstrator 1 will not use SD cards to save generated data, because random data should be always “fresh”. Storing generated random data on the non-volatile SD cards could allow the data to be manipulated by the attacker. Only one SD will be connected to the USB hub – the application software can be stored in this card if needed.
The user is supposed to use Demonstrator 1 with a PC as an external USB device, or he can integrate it to the computer rack, using an internal USB cable and a disk drive power connector.

2.4.1 USB Connectivity

Demonstrator 1 will be implemented on a single PCB board. The board will feature two USB connectors, one for an external cable and another one to connect Demonstrator 1 to the motherboard of the PC. One of the two connectors will be selected by a USB switch, depending on the selected power supply (Fig. 2.18).
The USB connection between the SmartFusion2 FPGA device and the PC is ensured by two USB communication channels. The first one (exploiting the FTDI device FT232RL), which is a virtual COM port, is designed to control Demonstrator 1 by a simple UART protocol. The main advantage of this data interface is that it does not need any special software driver – the device is supported by all existing operating systems.

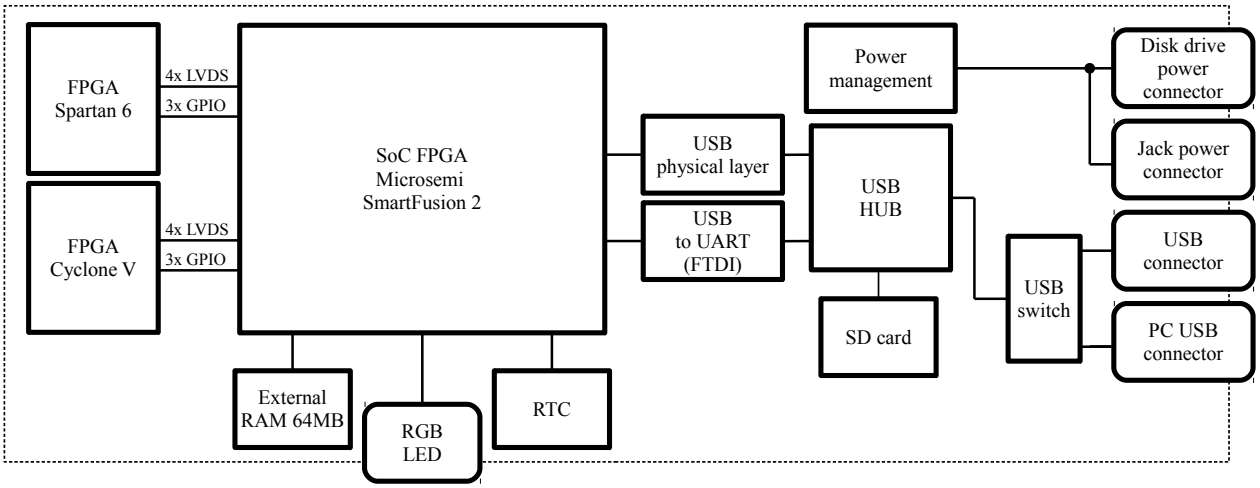

Figure 2.17: Demonstrator 1 hardware platform.

The second USB channel is designed to provide reliable and fast transfers of random data using the USB mass storage class interface. This interface is natively supported by all operating systems, too. This USB channel is realized using the USB physical layer circuit (USB3300), which creates an intermediate interface between the main SmartFusion2 device and the USB differential wires.
Both USB ports are connected to the USB HUB (USB2640). The selected USB HUB includes an integrated Micro SD card reader with a mass storage class interface, which is suitable for saving the application dedicated software.

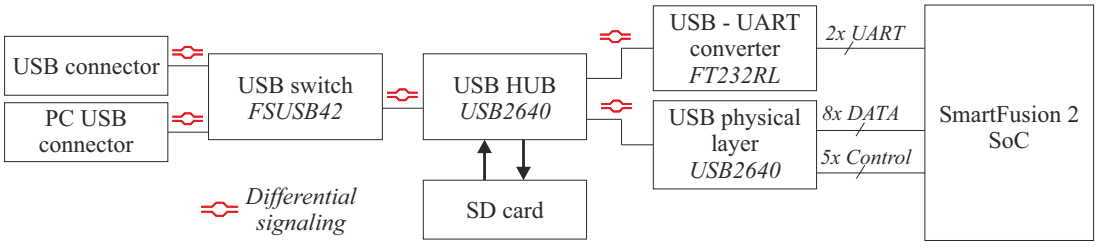

Figure 2.18: USB devices block diagram

2.4.2 Optical Signalization Elements

In practice, it is important to optically indicate the current state of the demonstrator. Several LEDs will be used to do it:

- Total failure of the PLL-TRNG implemented in Cyclone V FPGA (red LED),
- Total failure of the DC-TRNG implemented in Spartan 6 FPGA (red LED),
- Failure of the Online test in the PLL-TRNG (red LED),
- Failure of the Online test in the DC-TRNG (red LED),
- Failure of the Startup test in SmartFusion2 (red LED),
- TRNG data ready (green LED),
- UART Tx (green LED),
- UART Rx (green LED),

- Power ready (green LED).

2.4.3 Power Supplies

The whole device will be powered from two different connectors (Fig. 2.19). One connector dedicated to an external power supply adapter, and one for a disk drive power cable (available in common PCs). The demonstrator will be powered by default by 12V, but alternatively, a different power supply voltage can be used.

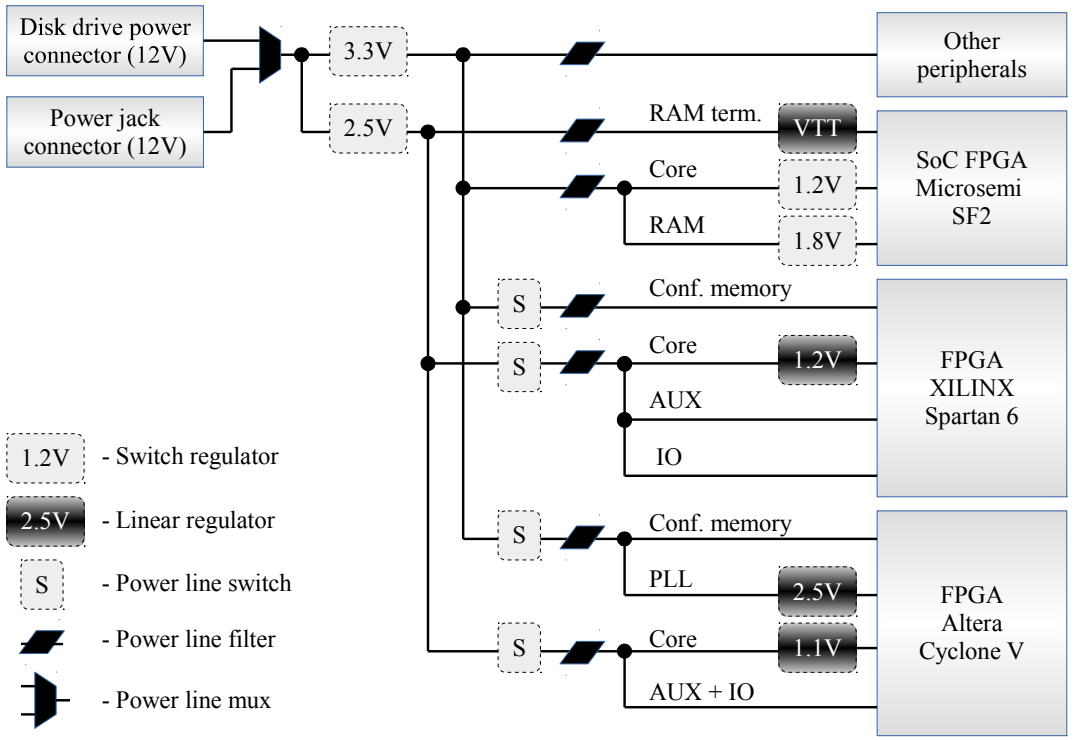

Figure 2.19: Demonstrator 1 power supply diagram.

The demonstrator will feature a complex power supply system. The linear voltage regulators will be used for the noise-critical parts due to their low noise feature. The switching power regulator will be used for the rest of the device, to ensure high power efficiency. The proper isolation filters will be used to reduce the possibility of side channel attacks. Selected power regulators will also have an option to be controlled from the control device. Indeed, shutting down some external devices can reduce EM emissions, but also overall temperature of the device.

2.4.4 Mechanical Design

The device will have dimensions of a 3.5-inch hard-disk (Fig. 2.20) to ensure its easy integration in the computers (e.g. in PCs or servers).
The printed circuit board (PCB) will be embedded in a special aluminium case, in order to:

- facilitate the heat dissipation,
- protect the device against a mechanical damage,
- create an EM shielding of the PCB sections,

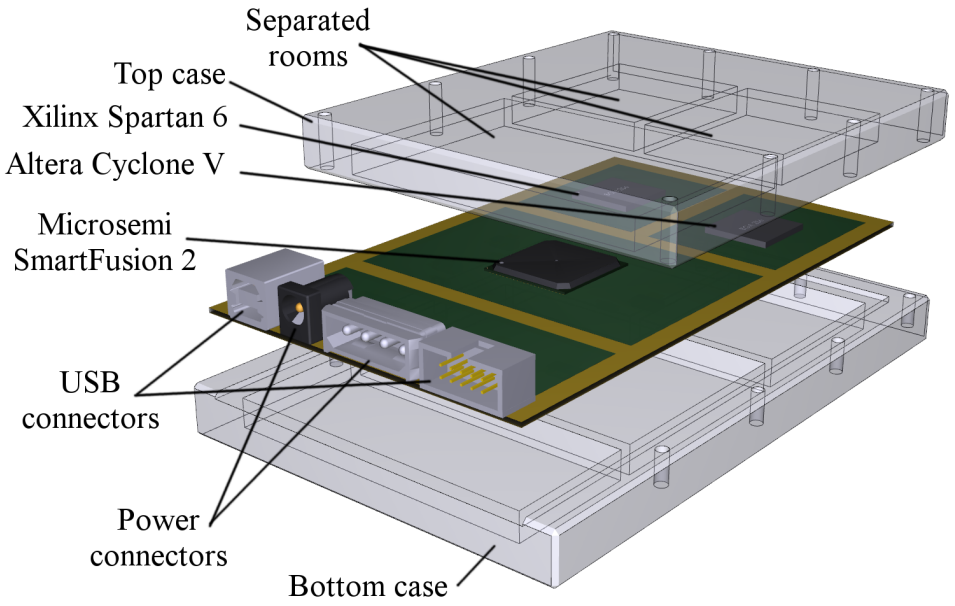

Figure 2.20: Design of the Demo 1 mechanical structure.

- protect the device against disassembling.

The case will feature separated rooms. The PCB will have detached areas for these rooms (Fig. 2.21). The physical separation will reduce the influence between FPGAs and stabilize their environmental conditions. The case will also allow to use an epoxy mass to make disassembling of the device more difficult.

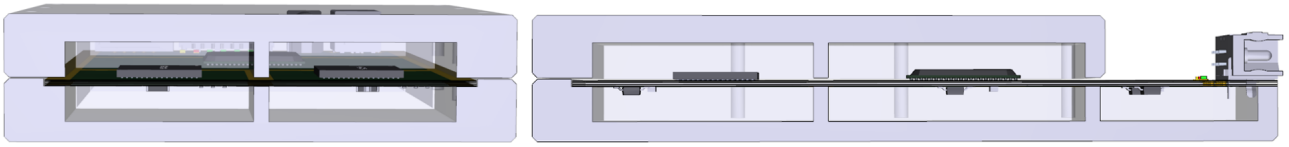

Figure 2.21: Mechanical design cross section of the Demo 1 - front view (on the left), side view (on the right)

2.5 Scope of Evaluation

Demonstrator 1 could serve in the future as a high-speed random number generator, exploiting one of two physical random number generators: a PLL-TRNG or a DC TRNG. This kind of generators could find their typical application in personalization centers where high volumes of high quality random numbers are required to initialize a large quantity of security devices such as smart cards. An attacker that would like to attack the system would try to weaken the generator, i.e. to reduce the entropy rate at generator output. This could be feasible only by physical attacks, which would typically need the attacker to have a physical access to the device (see Section 5.3).
Due to the targeted use-case, it is supposed that Demonstrator 1 will always operate in a controlled environment. As a result, potential attackers will not have physical access to generator, not even remotely through its interfaces. This means that deliberate attacks aiming at manipulating the generated random numbers are not in the scope of the security evaluation.

However, it is important that Demonstrator 1 will be robust regarding changes in environmental conditions and that it will feature long-term stability regarding the quality of generated numbers. The security evaluation will therefore focus on compliance of the design to AIS20/31, verification of the quality of the TRNG output, observation of entropy margins during variation of the TRNG operational conditions and verification if the behavior of the TRNG is in line with the stochastic model.

2.6 Conformance to Requirements

The following table shows the current status of Demonstrator 1 requirements defined in D 1.2. All the requirements which need to be justified on the final hardware and software will be reported in the upcomming deliverables D4.2 and D4.3.

Requirement	**Current status**	**Remark**
Functional requirements		
Ready after 3 seconds	$<$ 500 ms	–
Output data rate over 1Mb/s	1.04 Mb/s	–
Secure DRBG	DRG.3 Implemented	See 2.3.3
Hardware requirements		
Embedded processor on control FPGA	Compliant	ARM M3 embedded in SmartFusion 2 FPGA
USB connectivity on control FPGA	Compliant	USB Mass Storage Device interface used
Over 15k logic cells on control FPGA	27k logic cells	–
Over 10k logic cells on "other" FPGAs	49k and 14k logic cells	–
Total USB consumption smaller than 5W	To verify on the final hardware	–
Data interface faster than 2Mb/s	To verify on the final hardware	–
Security requirements		
TRNG is AIS20/31 PTG.3 compliant	Compliant	See 2.3.1
TRNG require less than 300mW	To verify on the final hardware	–
TRNG startup tests require less than 3 second to complete	$<$ 500 ms	–

Chapter 3

Demonstrator 2: Secure Portable USB Data Storage

3.1 Motivation

USB flash drive, *USB drive*, *USB stick*, *thumb drive*, *pen drive*, *jump drive*, *flash-drive* or *USB memory* are popular means of data storage, back-up and transfer. Such a device may contain sensitive personal information and/or company assets in digital form, e.g. bank statements, legal documents, commercial treaties, sales and billing documents, source codes, technical documentation or other intellectual property. The risk for the information to be compromised is high when commuting, travelling, or just leaving the drive unattended in a car or a hotel room. To **enhance privacy** one has to use a secure solution. Such a product then contains not only a flash memory and a USB interface but also a hardware encryptor. Vendors offer secure USB portable devices where data are protected by approved cryptographic algorithms, e.g. AES. However weaknesses have been discovered in such secure solutions. There is also a lack of trusted secure storage solutions engineered and manufactured in EU.
Another three reasons for Demonstrator 2 are to show that the proposed cryptographic primitives, algorithms and protocols can be successfully applied in a high-end real-world data security application, in which user and device authentication is sufficiently strong and secure at least in the lost and found scenario. **TRNG compliant with AIS 20/31 PTG.2** introduces the online test which immediately detects and stops any defect in random data generation. **Authenticated encryption** ensures confidentiality and integrity of the data in a single operation. **Physically unclonable function** binds the encrypted data to the particular piece of hardware. All of aforementioned principles are brand new, as it is the study of their behaviour and impact in the real world.
Demonstrator 2 was also inspired by the MICRONIC **hardware platform** available already at the beginning of the project.

3.2 Functional Description

Demonstrator 2 is a secure portable USB storage. It is a personal, single-user device. It protects the data stored on it while being at rest. It requires the user to authenticate before allowing access to the data. It is powered from the USB bus without the need of an external power source.
The device is either empty, deleted or enrolled. It is **empty** just after manufacturing. After that, the user enrolls it. After being **enrolled**, the embedded non-volatile memory (eNVM)

contains helper data, encrypted key and their authentication tag. The device is **deleted** on user request or after too many unsuccessful tries to enter the passphrase, then the data in the eNVM and on the SD card are zeroized.
The user **activate**s his enrolled device by entering the passphrase. The key is decrypted, authenticated and loaded to the designated register. The crypto block is then prepared to encrypt data on sector writings and decrypt data on sector readings. The device is ready to be mounted by the operating system.

3.2.1 Two-factor Authentication Protocol

The HECTOR project partners designed a two-factor authentication protocol which allows authentication of the user, the device and the helper data, and the authenticated encryption of the data encryption key. The protocol uses the sponge based authenticated encryption primitive ASCON-128a (ASCON$_{12,8}$-128-128).

0. During the **preparation** phase the helper data is generated and securely stored. See figure 3.1.
   (a) User enters passphrase PP, which is decoded from ASCII to create helper key $K_W = f(PP), f: \{A-Z, a-z, 0-9, @, \&\}^{22} \rightarrow \{0,1\}^{128}$;
   (b) PUF generates response R and extractor computes helper data W;
   (c) TRNG generates nonce N_W;
   (d) Crypto core is initialized with K_W and N_W;
   (e) Authentication of W takes place;
   (f) N_W, W and $tag(W)$ are stored in the eNVM.
1. During the **enrollment** phase the data encryption key is generated and securely stored. See figure 3.2.
   (a) User enters passphrase PP', which is decoded to create helper key $K'_W = f(PP')$;
   (b) eNVM outputs N_W, W and $tag(W)$;
   (c) Crypto core is initialized with K'_W and N_W; Verification of W takes place; Stored $tag(W)$ and freshly computed $tag'(W)$ authentication codes are compared;
   (d) If authentication succeds PUF generates response and using helper data W corrects R';
   (e) Helper key is masked with PUF response to create device key $K'_d = K'_W \bigoplus R'$;
   (f) TRNG generates data encryption key K_a and nonce N_a;
   (g) Crypto core is initialized with K'_d and N_a;
   (h) Authenticated encryption of K_a takes place;
   (i) N_a, $enc(K_a)$ and $tag(K_a)$ are stored in the eNVM.
2. During the **activation** phase the helper data integrity is verified and data encryption key is reconstructed and verified. (See Figure 3.3.)
   (a) User enters passphrase PP'', which is decoded to create helper key $K''_W = f(PP'')$;
   (b) eNVM outputs N_W, W and $tag(W)$;

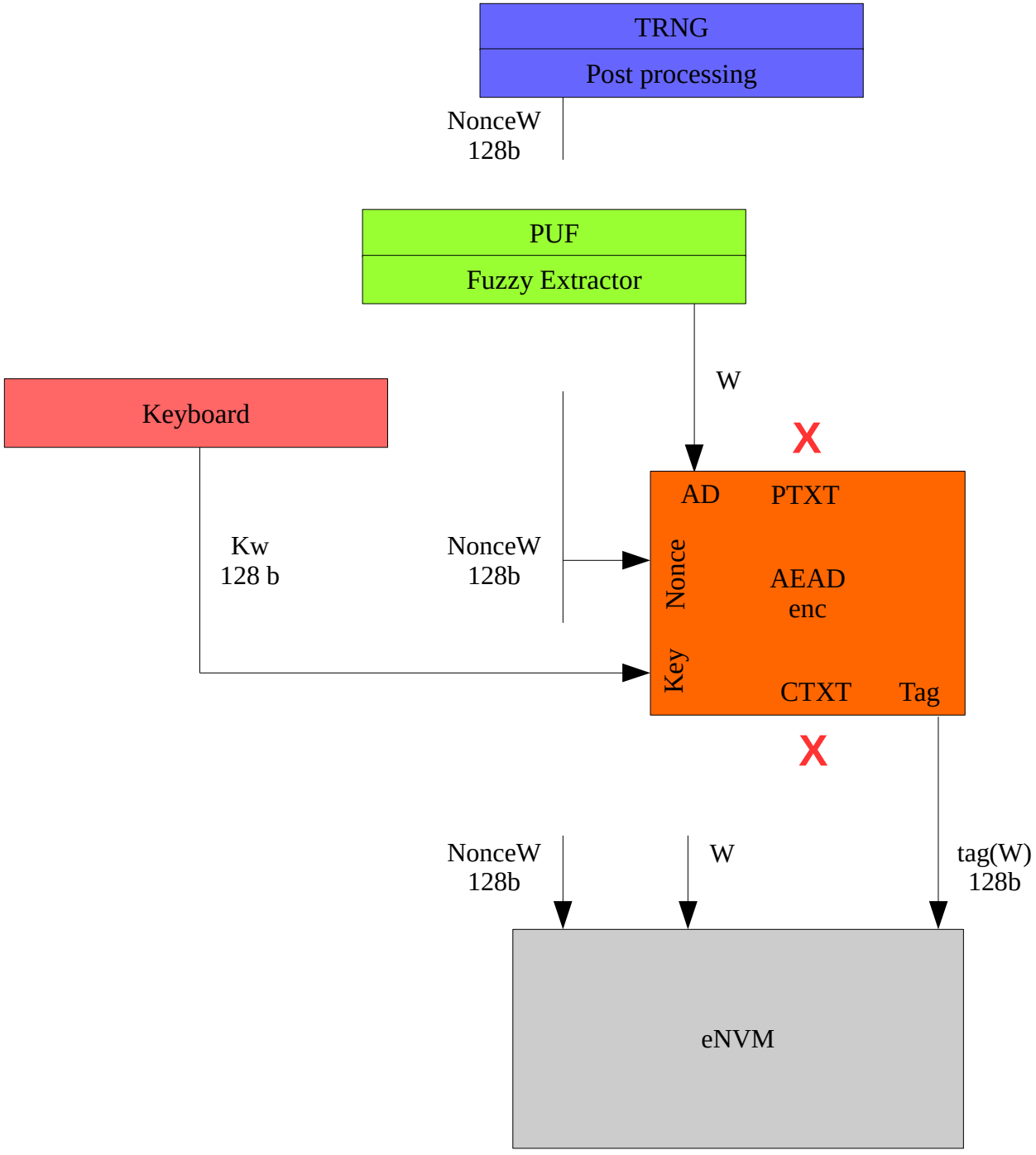

Figure 3.1: Demonstrator 2 helper data generation.

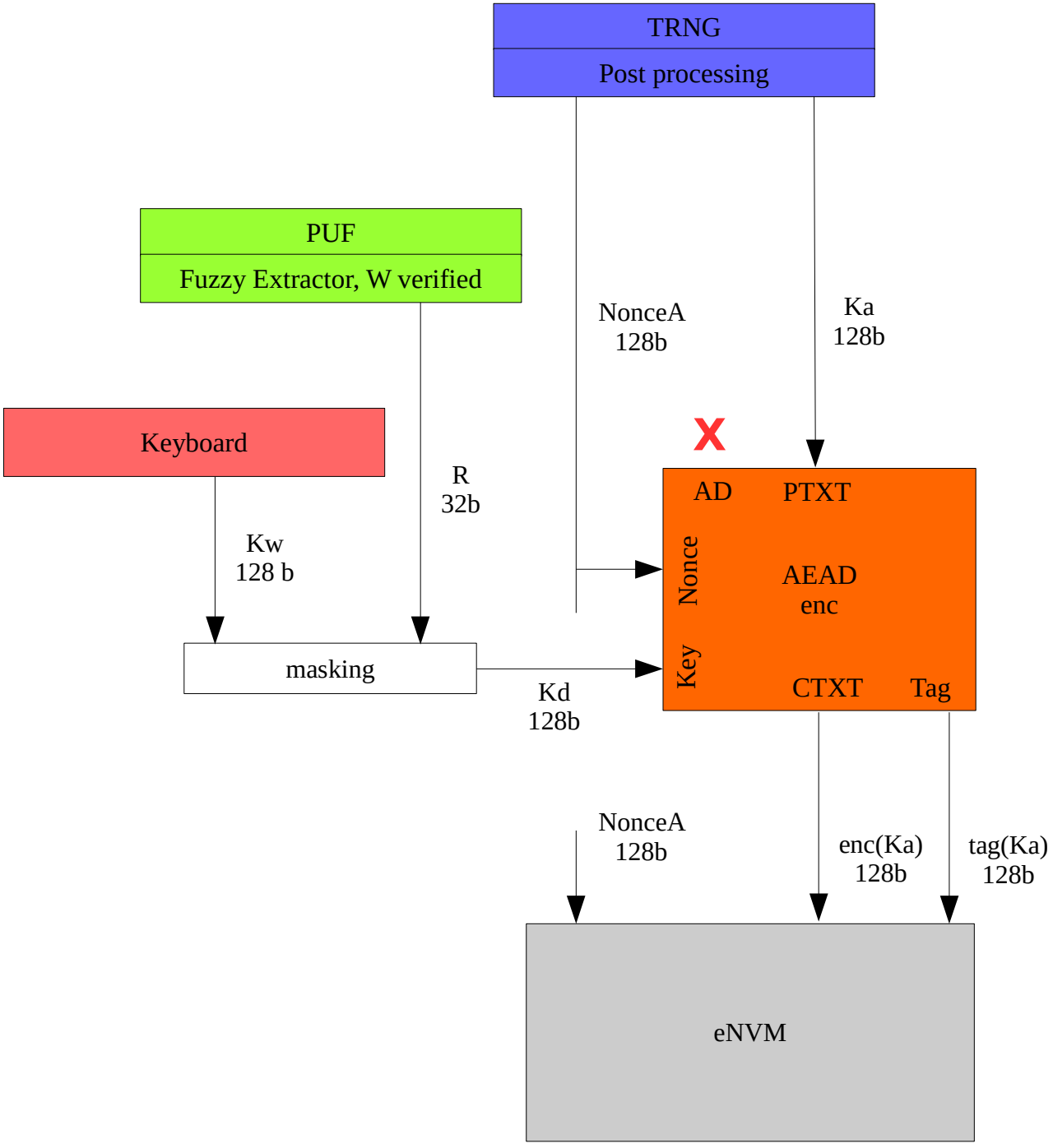

Figure 3.2: Demonstrator 2 enrollment.

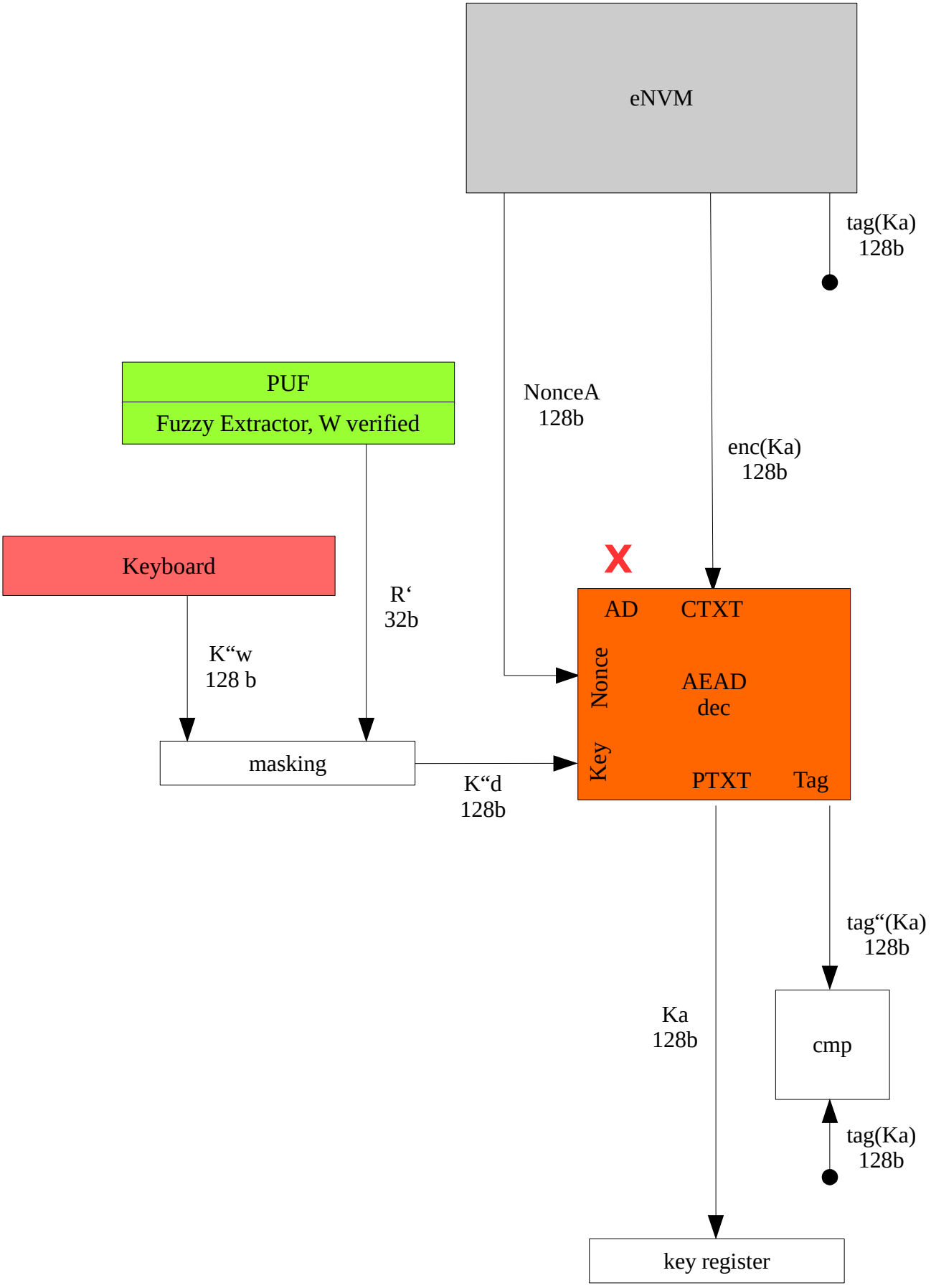

Figure 3.3: Demonstrator 2 activation.

(c) Crypto core is initialized with K''_W and N_W; Verification of W takes place; Stored $tag(W)$ and freshly computed $tag''(W)$ authentication codes are compared;

(d) If authentication succeds PUF generates response and using helper data W corrects R'';

(e) Helper key is masked with PUF response to create device key $K''_d = K''_W \bigoplus R''$;

(f) eNVM outputs N_a, $enc(K_a)$ and $tag(K_a)$;

(g) Crypto core is initialized with K''_d and N_a;

(h) Verified decryption of $enc(K_a)$ takes place; Stored $tag(K_a)$ and freshly computed $tag''(K_a)$ authentication codes are compared; If authentication succedes the device is ready else an error is reported.

3.2.2 Data Encryption Process

All data on the SD card are encrypted per sector. Handling each sector as a separate message allows random access to the sectors. Authenticated encryption does not preserve the data size as it needs to store a nonce and an authentication tag for each sector. That extra data will be stored in reserved space on the SD card along with the encrypted user data.
Note that despite the sector on the SD card has a fixed size of 512 bytes, the demonstrator may work with multiple sized sectors and so lower the overhead.

1. When writing data to the device each sector is **encrypted** and sent to the SD card along with a unique Nonce and an authentication tag. See figure 3.4.

   (a) MSS_USB controller sends data $SectorX$;

   (b) From TRNG output and write counter is $NonceX$ derived;

   (c) Crypto core is initialized with K_a and $NonceX$;

   (d) Authenticated encryption of $SectorX$ without any associated data takes place;

   (e) $NonceX$, $enc(SectorX)$ and $tag(SectorX)$ are stored in the SD card.

2. When reading data from the device each sector is **decrypted** and USB controller is then allowed to read the data. See figure 3.5.

   (a) $NonceX$ is read form the SD card. Crypto core is initialized with K_a and $NonceX$;

   (b) Data $enc(SectorX)$ are read from the SD card and decrypted.

   (c) Freshly computed authentication value $tag'(SectorX)$ is compared to the one stored on the SD card $tag(SectorX)$.

   (d) If the tags are equal, the USB controller reads the data $SectorX$ else an error is reported.

3.3 Building Blocks

3.3.1 Passphrase

A random sampled 128-bit number is encoded into the user passphrase. The random number will be used as the helper key and after masking by the PUF response as the device key, see

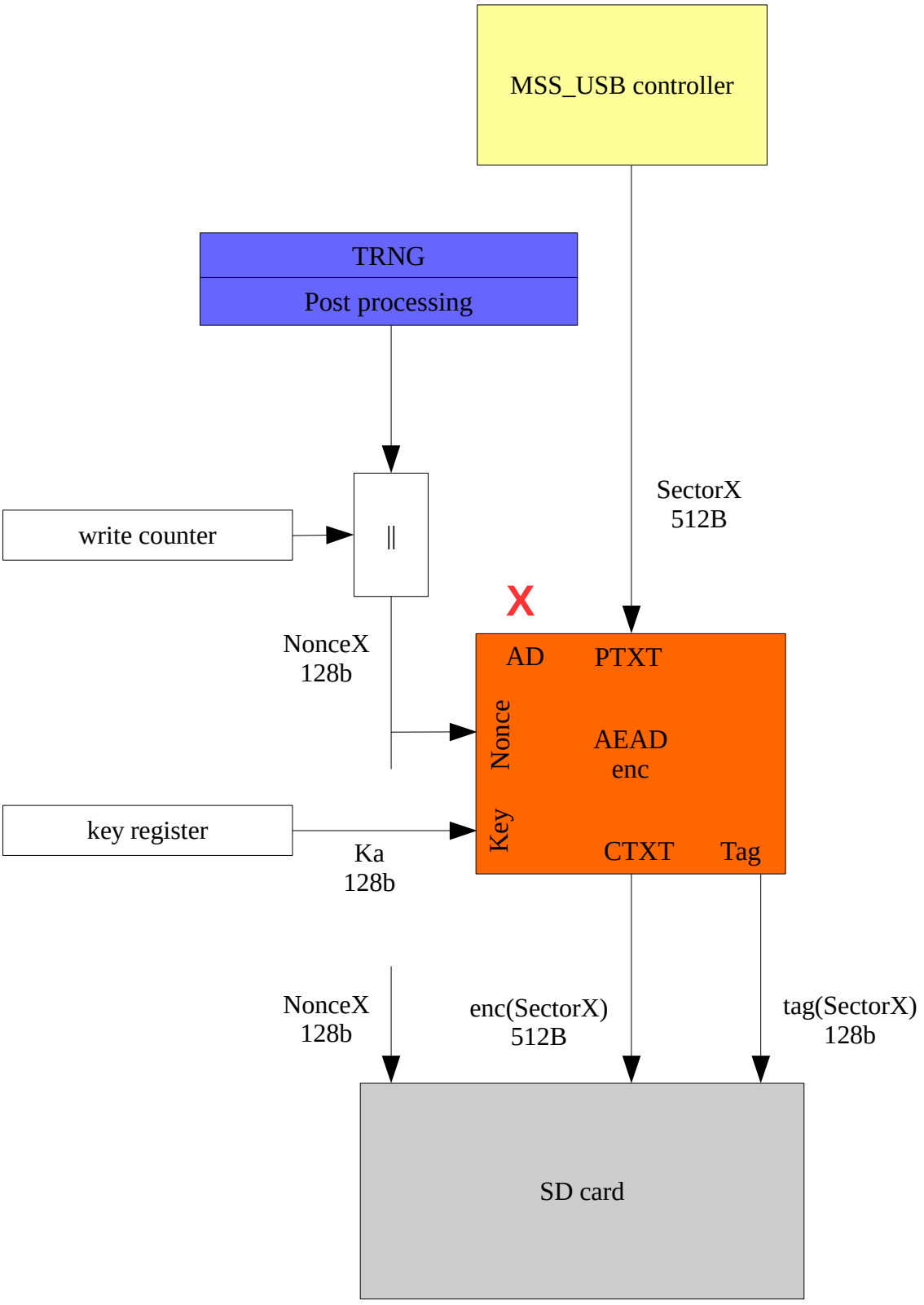

Figure 3.4: Demonstrator 2 data encryption.

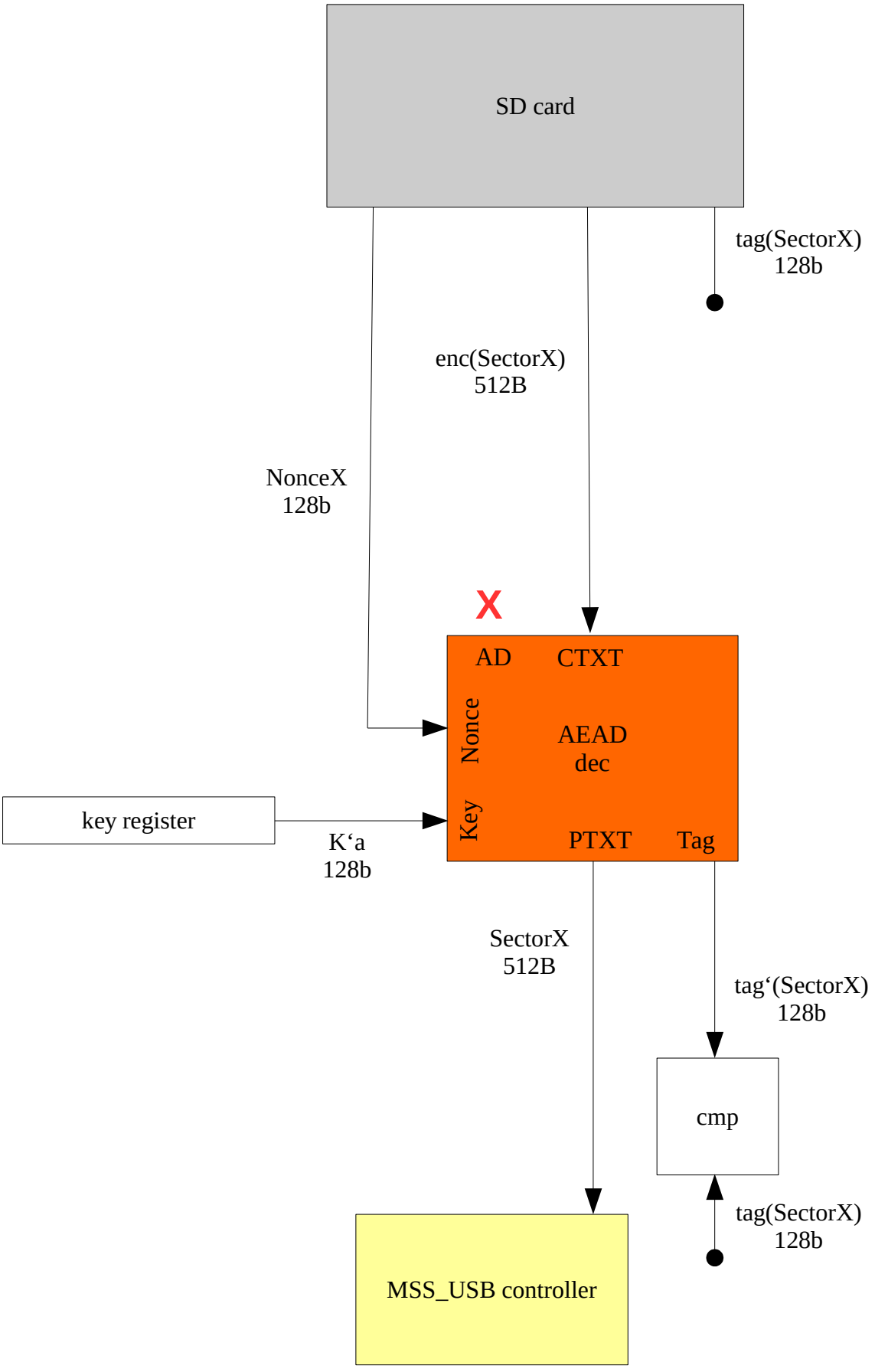

Figure 3.5: Demonstrator 2 data decryption.

Subsection 3.2.1. The passphrase is a string of 22 characters in the alphabet $\mathcal{A} = [A-Z, a-z, 0-9, @, \&]$, since $|\mathcal{A}| = 64$.
The complete alphabet the device recognizes contains also the additional special characters $:!?\_|.,*+/$.

3.3.2 Physically Unclonable Function

TERO-PUF

The Transient Effect Ring Oscillator (TERO) corresponds to a very specific configuration of an RS latch with additional gates and two inputs featuring the same voltage [16]. The TERO is an electronic circuit composed of an even number of inverters that oscillates temporarily. A typical TERO cell is depicted in Figure 3.6.

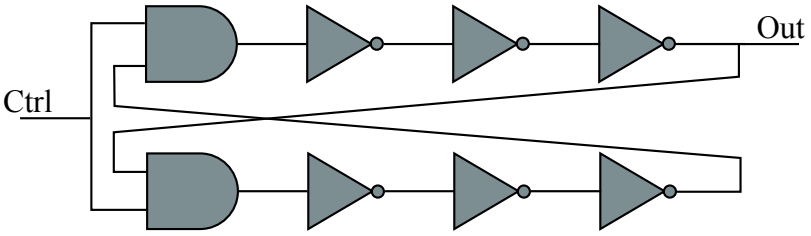

Figure 3.6: A typical TERO cell

It is composed of two AND gates and two inverter branches. Due to manufacturing process variabilities, by propagating two events at the same time in the loop, one event will be faster than the other and collision of both events will stop the oscillation. The duty cycle of oscillating output varies over time and after a certain number of oscillations, reaches the rate of either 0% or 100%. In this configuration, we compare the number of oscillations of two identically implemented TEROs. The first use of the TERO was introduced by Varchola et al. [18] for True Random Number Generator applications. Later Bossuet et al. [6] proposed a new PUF principle based on TEROs. The TERO PUF generates this response by comparing the number of oscillations of TERO cells. The TERO PUF is configured as presented in Figure 3.7.

The implemented PUF is composed of 128 TERO cells, two counters and a subtractor. Cells are divided in two blocks, A and B of 64 cells each. To avoid correlation, a cell of the block A is always compared to a cell of the block B and is used only once. One cell per block is selected using two demultiplexers. Two multiplexers are placed right after the cell blocks in order to drive the correct cell outputs to the clock counters. The cell selection is usually called a challenge. Compared TERO cells are triggered at the same time. We compare the number of oscillations at the output of two selected cells using a subtractor. We extract two bits per challenge. Counters and activation time of the control signal need to be sized according to the mean number of oscillations of the TERO cells. In our case, counters feature 10 bits and activation time is set to $2\mu s$. The strobe signal indicates that the output of the subtractor is ready to be read. The protocol to generate a 2-bits response to a challenge is the following:

1. Select the couple of TERO cells to be compared
2. Activate the TERO cells (ctrl signal = 1) for $2\mu s$
3. Deactivate the TERO cells

4. Generate a strobe (which means subtractor output is ready to be stored)
5. Reset counters

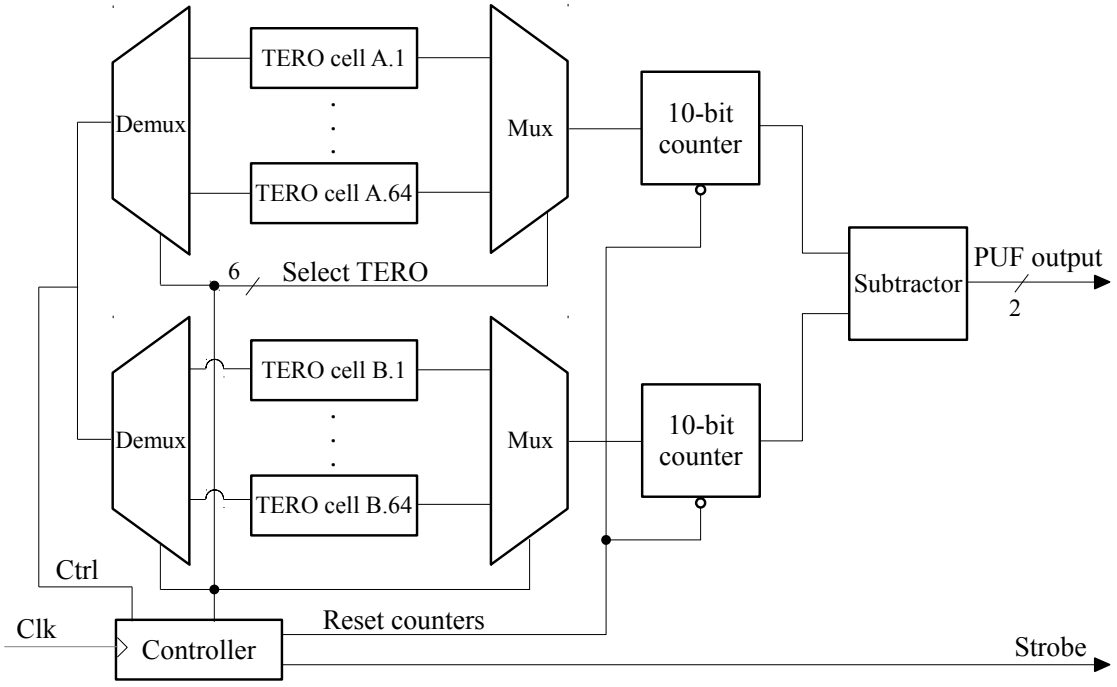

Figure 3.7: TERO PUF architecture

This is done for all implemented TEROs and the resulting bits are concatenated to form a 128-bits response. The response is stored in a shift register.

PUF characteristics and error-correction

Error correcting codes are implemented, whenever messages have to be reliably transmitted via a noisy channel. In the context of PUFs, error correcting codes are implemented to ensure that a secret can be successfully reproduced. The basic idea of error correcting codes is to enlarge the signal length by redundant information appended to the information part of the message, to improve the probability that the original message can be reconstructed from the possibly corrupted data received. On the sender side, there is an encoder, whose job is to take the message and to process it to produce the coded bits that are then sent over the channel. The receiver has a decoder whose job is to take the received (coded) bits and to produce its best estimate of the message. The encoder-decoder procedures together constitute channel coding. Good channel codes provide error correction capabilities that reduce the bit error rate considerably, or in the best case, entirely correct the errors which occurred. [3]

In summary, transmitted information is assumed to be correct on the receiver side, but in fact, the signal will be superimposed by noise with a specific bit error probability p_b. Table 3.1 shows first evaluation results of the TERO PUF. Their importance and impact will be discussed in the following sections.

bias:	54.15%
avg. bit error probability p_b:	5.15%
max. bit error probability p_b:	19.00%

Table 3.1: First evaluations on the characteristics of the TERO PUF

Assuming a given bit error probability of p_b we claim a failure error rate of $P_{fail} \leq 10^{-4}$ within Demonstrator 2 and 3. The failure error rate describes the probability that a string of n bits has more than t errors and is given by:

$$P_{fail} = \sum_{i=t+1}^{n} \binom{n}{i} p_b^i (1-p_b)^{n-i} = 1 - \sum_{i=0}^{t} \binom{n}{i} p_b^i (1-p_b)^{n-i} \tag{3.1}$$

Another limitation given in the requirements of Deliverable D1.2 is that for the PUF implementation only 5k logic cells should be used on the demonstrator platform. Related to the TERO PUF this equals a number of approximately 128 bit output bits of one PUF response. For the building block of authenticated encryption a 32 bit full entropy key is expected from the PUF block. Therefore, the post-processing needs to be calibrated based on these requirements.

Aiming for full entropy, we need to first check for the characteristics of the TERO PUF output. It is desirable for PUFs to have a fractional Hamming weight of close to 0.5 since this indicates that the PUF response does not show a preference for a certain value and is therewith unbiased. Maes et al. also mentioned in [14] that bias is the most common cause for PUFs to show non-randomness.
There are different approaches to extrapolate from the Hamming weight to the entropy, see Def. 1 and Def. 2, where one is a more conservative and therefore less practical approach than the other:

Definition 1. Min-Entropy: The min-entropy is the most conservative approach to indicate the amount of randomness present in the response and is therefore desired to be close to 1. We define p as the probability of a *1* or a *0* showing up in a string of bits, $p_i \geq 0$, $\sum_{i=1}^{2} p_i = 1$.

$$H_\infty = \frac{1}{N} \sum_{n=1}^{N} -\log \max_i p_i \tag{3.2}$$

Definition 2. Binary entropy function: The binary entropy function calculates the entropy of a Bernoulli process[1]. In comparison to the min-entropy, the binary entropy function takes a single real number as a parameter whereas the min-entropy takes a distribution or random variable.

$$H_b = -p \log p - (1-p) \log(1-p) \tag{3.3}$$

In addition, the efficiency of the helper data algorithm in use (Kang's scheme, see Figure 3.9) needs to be taken into account. It is determined by the min-entropy loss which is assumed to be the universal $(n-k)$ bound for a binary block code $[n, k, d]$ for constructing a secure PUF-based key generator, see Equation 3.4. In [14] it is shown, that the $(n-k)$ lower bound

[1] finite or infinite sequence of a binary random variable

even becomes negative for too high bias, making it absolutely clear that this is a pessimistic lower bound. In [7] Delvaux et al. derive new considerable tighter bounds for PUF-induced distributions that suffer from bias or spatial correlations.

$$\ell = \mathrm{H}_\infty \cdot n + k - n \tag{3.4}$$

For the TERO PUF in the context of the HECTOR demonstrator, we are limited to approx. 128 output bits. To improve the quality of the output bits and to ensure that we meet the given requirements, the situation calls for special measures:

Entropy estimation: For high-end security products, it is recommended going for the conservative approach of Min-Entropy. In HECTOR for demonstration purposes, the security requirements are eased, using the binary entropy function to extrapolate the entropy. Table 3.2 shows the impact of bias deviating from 50% and the difference for the resulting entropy in the PUF response for both approaches. Table 3.3 shows that the left over entropy after post-processing with three different codes is still sufficient even though that bias is present.

Majority voting: In [2], a method is described to receive the most often occurring values for a PUF sample. With this, we would be able to define a much more stable reference response. As a consequence, this will decrease the bit error probability to reasonable values. This is matter of ongoing work.

Dark bit selection: The idea of bit selection schemes is to discard less reliable bits before the PUF response will be further processed. This will also decrease the bit error rate significantly, but unfortunately, at the expense of source bits. Again, these evaluations are currently ongoing.

PUF type	**no. responses**	**bias**	$\mathrm{H_b}$	H_∞
TERO	100	50.00%	1.00	1.00
TERO	100	54.15%	0.99	0.89

Table 3.2: Calculation of min-entropy and binary entropy function based on bias

bias	$\mathrm{H_b}$	**code**	**left over entropy**$_{\mathrm{H_b}}$
54.15%	0.99	G(23,12,7)	58(60)
54.15%	0.99	BCH(127,64,21)	62(64)
54.15%	0.99	BCH(127,43,29)	36(43)

Table 3.3: Left over entropy depending on $\mathrm{H_b}$ (Golay is processed in 5 loops)

As can be seen in Figure 3.8, the restrictions in terms of size (length of usable PUF response), stability and the required failure error rate of $P_{\text{fail}} \leq 10^{-4}$ challenge us to bring the bit error rate down to an area of $2-3\%$. We will apply measures described above (majority voting, dark bit selection) to achieve the best possible trade-off. Those measures will be put in place during the integration of demonstrator 2 (and demonstrator 3) and reflected in deliverable D4.2.

	P_{fail}	codes Golay (23,12,7)	BCH (127,64,21)	BCH (127,43,29)
$\mathbf{p_b}$	2%	01.00e-03	05.42e-05	04.63e-08
	4%	01.23e-02	01.33e-02	01.81e-04
	6%	04.59e-02	01.41e-01	09.40e-03
	8%	01.07e-01	04.37e-01	08.32e-02
	10%	01.92e-01	07.35e-01	02.87e-01

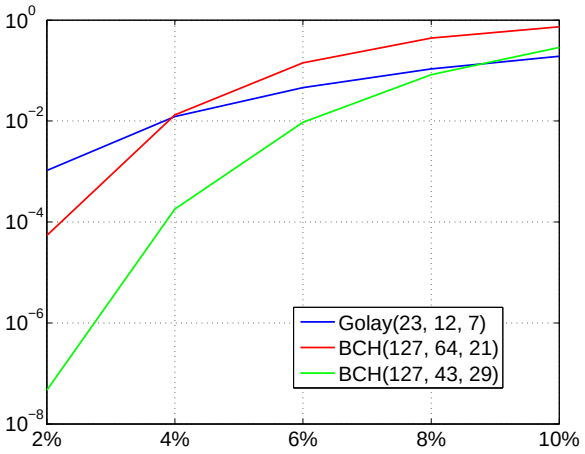

Figure 3.8: Failure rate P_{fail} as a function of different codes and bit error probabilities

Kang's scheme The idea described in [12] represents a very interesting alternative to the commonly used code-offset construction. Any error correcting code with an efficient syndrome decoding algorithm can be used in this scheme. One advantages of Kang compared to the conventional code-offset construction is that less helper data (N-K bits) need to be stored. In addition, there is no TRNG required during the enrollment procedure (see left part of Figure 3.9).

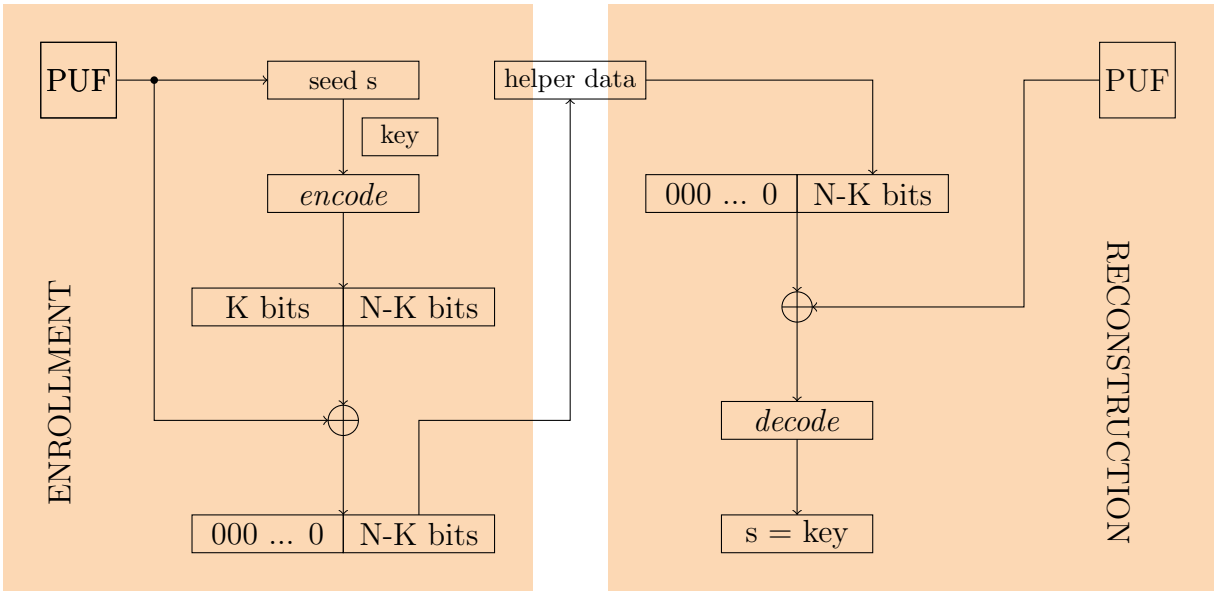

Figure 3.9: Fuzzy extractor based on codes with systematic encoding (Kang's scheme [12])

3.3.3 Authenticated Encryption with Associated Data

The sponge based authenticated encryption primitive ASCON-128a is used for both key reconstruction and data encryption. The only requirement put on the use of the algorithm is uniqueness of the $(key, Nonce)$ pairs. The ASCON instance is controlled by a state machine to process (encrypt/decrypt) 32 data blocks of 128 bits (512 byte sector), see listings 3.1 and 3.2. For optimization reasons the process may span over multiple sectors of 512 bytes. Encryption/decryption starts using a per-write unique $Nonce$ and computes an authentication tag on

the data. The *Nonce*s and the tags will be stored along with the encrypted user data on the SD card.

Listing 3.1: ASCON core interface.

```
entity ascon_core is

generic (
    UNROLLED_ROUNDS             : integer := 1;
    STATE_WORD_SIZE             : integer := 64;
    KEY_SIZE                    : integer := 128;
    DATA_BLOCK_SIZE             : integer := 128;
    ROUNDS_A                    : integer := 12;
    ROUNDS_B                    : integer := 8);

port (
    ClkxCI                      : in  std_logic;
    RstxRBI                     : in  std_logic;
    KeyxDI                      : in  std_logic_vector(KEY_SIZE-1        downto 0);
    DataxDI                     : in  std_logic_vector(DATA_BLOCK_SIZE-1 downto 0);
    DataxDO                     : out std_logic_vector(DATA_BLOCK_SIZE-1 downto 0);
    TagxDO                      : out std_logic_vector(DATA_BLOCK_SIZE-1 downto 0);
    DP_WriteNoncexSI            : in  std_logic;
    CP_InitxSI                  : in  std_logic;
    CP_AssociatexSI             : in  std_logic;
    CP_EncryptxSI               : in  std_logic;
    CP_DecryptxSI               : in  std_logic;
    CP_FinalEncryptxSI          : in  std_logic;
    CP_FinalDecryptxSI          : in  std_logic;
    CP_FinalAssociatexSI        : in  std_logic;
    CP_DonexSO                  : out std_logic);

end entity ascon_core;
```

Listing 3.2: ASCON core control example.

```
    process(PRSTN,  PCLK)
    begin
        if(PRSTN = '0')then
            state                 <= state_idle;
            task                  <= task_none;

        elsif(rising_edge(PCLK)) then

            case state is
                when state_idle           =>
                    case command_from_arm is
                        when X"00"  =>
                            state <= state_idle;
                            task  <= task_none;
                        when X"C1"  =>
                            state <= state_initialize;
                            task  <= task_sector_encrypt;
```

```
                when X"C2"  =>
                    state <= state_initialize;
                    task  <= task_sector_decrypt;
                when others =>
                    state <= state_error;
                    task  <= task_error;
            end case;
            block_counter <= (others => '0');

        when state_initialize      =>
            if ascon_done = '1' then
                if task = task_sector_encrypt or
                   task = task_sector_decrypt then
                    state <= state_associateFinal;
                else
                    state <= state_associate;
                end if;
            end if;

        when state_associate       =>
            state <= state_associate_wait;

        when state_associate_wait  =>
            state <= state_associateFinal;

        when state_associateFinal  =>
                state <= state_wait_for_data;

        when state_wait_for_data   =>
            if (task  = task_sector_encrypt and
                DATA_FROM_USB_RDY  = '1')   or
               (task  = task_sector_decrypt and
                DATA_FROM_CARD_RDY = '1')   then
                state <= state_encrypt;
            end if;

        when state_encrypt         =>
            state <= state_encrypt_wait;

        when state_encrypt_wait    =>
            if ascon_done = '1' then
                if block_counter(4 downto 0) = "11111" then
                    state <= state_finalize;
                else
                    state <= state_wait_for_data;
                end if;
                block_counter <= block_counter + 1;
            end if;

        when state_finalize        =>
            state <= state_finalize_wait;
```

```
            when state_finalize_wait   =>
                if ascon_done = '1' then
                    state <= state_continue;
                end if;

            when state_continue        =>
                    state <= state_initialize;

            when others                =>
                state <= state_error;

        end case;
    end if;
end process;

ascon_init            <= '1' when state = state_initialize    else
                         '0';

ascon_associate       <= '1' when state = state_associate     else
                         '0';

ascon_encrypt         <= '1' when (state = state_encrypt       or
                                   state = state_encrypt_wait) and
                                   task  = task_sector_encrypt else
                         '0';

ascon_decrypt         <= '1' when (state = state_encrypt       or
                                   state = state_encrypt_wait) and
                                   task  = task_sector_decrypt else
                         '0';

ascon_finalEncrypt    <= '1' when (state = state_finalize       or
                                   state = state_finalize_wait) and
                                   task  = task_sector_encrypt  else
                         '0';

ascon_finalDecrypt    <= '1' when (state = state_finalize       or
                                   state = state_finalize_wait) and
                                   task  = task_sector_decrypt  else
                         '0';

ascon_finalAssociate  <= '1' when state = state_associateFinal else
                         '0';
```

3.3.4 PLL-based TRNG with Embedded Tests

Demonstrator 2 features a physical true random number generator (PTRNG), which uses differential jitter between clock generated in two PLLs as a source of randomness. According to the deliverable D1.2, the RNG in Demonstrator 2 must fulfill security level PTG.2.

Principle

The principle and the architecture of the PTRNG implemented in Demonstrator 2 is the same as that of the PTRNG implemented in Demonstrator 1 and described in Section 2.3.1.
Just two differences exist between the two TRNG designs. First, instead of using an external quartz oscillator as a source of input clock signal *clk_in*, Demonstrator 2 uses RC oscillator, which is hardwired inside the SmartFusion2 SoC and which oscillates at a frequency of about 50 MHz as input clock generator. Consequently, multiplication and division factors of both PLL are adapted to this different frequency.
The second difference is related to the output circuitry: generated data and state register do not need to be output – they can be accessed inside the device by the embedded ARM processor.

HECTOR Demonstrator 2 PLL-TRNG Implemented in SmartFusion 2 FPGA

The final version of the TRNG implemented in Demonstrator 2 in SmartFusion2 SoC is depicted in Fig. 3.10.

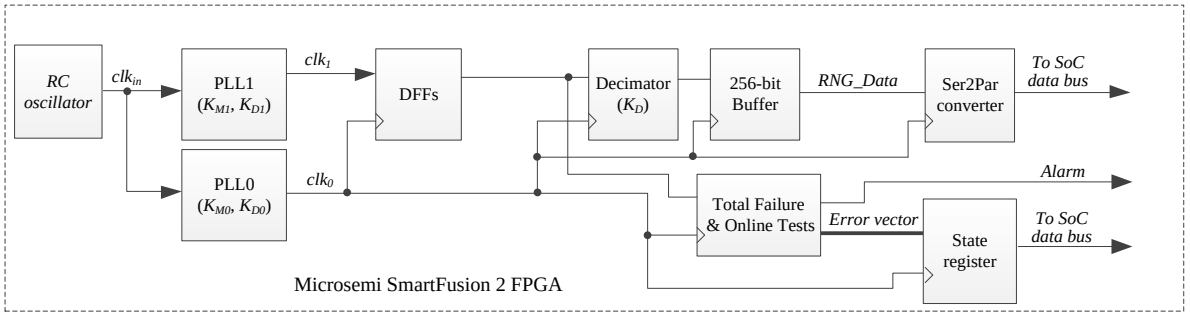

Figure 3.10: Block diagram of the PLL-TRNG design implemented in SmartFusion2 FPGA

The two PLLs are connected in parallel in order to increase the targeted differential jitter. PLL1 generates the clock signal clk_1 which is sampled in a flip-flop on rising edges of the clk_0 clock signal generated in PLL0. Output of the flip-flop goes to the decimator. The decimator (a 1-bit counter) is incremented each time this signal is equal to one during K_D rising edges of the clk_0 clock signal (during one period T_Q). After each period T_Q the decimator value (one bit) is saved in a 256-bit output buffer and the decimator is reset.
The input signal of both PLLs is generated in the RC oscillator embedded in the SmartFusion 2 FPGA. The oscillator oscillates at a frequency of approximately 50 MHz. PLL0 has multiplication and division factors $KM_0 = 31$, $KD_0 = 12$, respectively. PLL1 has multiplication and division factors $KM_1 = 37$, $KD_1 = 7$, respectively. The final multiplication and division factors of the PLL-TRNG are thus $KM = 444$, $KD = 217$. The frequency of the reference clock clk_0 is 129.17 MHz and the frequency of the jittery clock clk_1 is 264.28 MHz. This gives the bit rate of 600 kbits/s at generator output, which is much higher than required.
Since the coefficients of the two PLLs are set in such a way that the entropy rate per bit at decimator output is higher than required by the AIS 31 standard, the algorithmic post-processing is not needed.
The PTRNG implemented in Demonstrator 2 uses the same Total failure test (test *T0*) and Online tests (tests *T1* and *T2*) as the generator in Demonstrator 1 (see Section 2.3.1).
The error messages from the functional tests (e.g. PLLs not locked) and statistical tests (e.g. the total entropy failure and failure of Online tests) form an error vector, which is saved in a

state register that can be reached from the embedded ARM processor. If any of these tests fails, the *Alarm* is triggered, causing an interrupt condition in the control SoC.

3.4 Design Rationale

Demonstrator 2 as a portable secure USB storage a) communicates via USB and b) stores c) encrypted data on an SD card. The device interfaces on the red (plain) side the host computer via USB and on the black (encrypted) side the SD card. USB mass storage class is natively supported across the modern operating systems, and so, there is no need for additional proprietary software for the host computer.
SD card is used as a data storage. It is an autonomous, reliable device well defined in a series of specifications. It is equipped with a wear leveling mechanism to stretch the write cycle limit of the media. SD cards support, in the case of Demonstrator 2 encrypted, data back-up through disk imaging software.
Microsemi Smartfusion2 System on Chip (SoC) interfaces the USB and encrypts and decrypts the stored data. SoC is a low power, reconfigurable FPGA device with a 32-bit ARM processor. SoC is quite documented and supported with a design tools suite.
All data on the SD card are encrypted, protected by a strong cryptographic algorithm, thus resistant against off-line attacks. In the scope of the lost and found (stolen) scenario the solution has to be secure against the brute force attacks and no sensitive data must be stored on the device in plain.
A secure authentication protocol for both the device and the user has to be implemented to ensure that no critical data like confidential keys and passwords will be saved in the device in plain.
As Demonstrator 2 after being activated allows full access to the data, it is expected that the user activates and uses the device only in proven and controlled environment, i.e. eliminated possibility of interception, scanned against malware. This prerequisite leaves the side channel attacks out of scope.
The user authenticates to the device by entering a passphrase. The passphrase never leaves the USB device as it is typed on the built-in keyboard and displayed on the built-in display. Hence the device is immune against key-logger attacks. The self-generated keys and user passphrase will be protected against cloning through PUFs using a protocol consisting of two separate phases:
When at rest, on the move, lost or stolen the data on the removable media are securely protected by a strong authenticated encryption algorithm. Without knowledge of user's high entropy passphrase either the SD card or the whole device is useless for the attacker.b
Thus, when the device is powered-off or is not activated, it does not contain any key or security critical information in plain. Since the objective is to bind sensitive information to a device, we will focus here on weak PUFs and not consider strong PUFs. Additionally, the derivation of the user passphrase could reuse primitives from the post-processing step of the PUFs.
The encryption mechanism must support random access to sectors.

3.5 Hardware Platform

The hardware of Demonstrator 2 is based on MICRONIC's hardware which was developed at the beginning of the HECTOR project. The existing hardware accelerated the development of the final HECTOR Demonstrator 2 and 3 platform – the secure hand-held device.

The designed hardware is very similar to the HECTOR evaluation platform, but in order to meet the USB power requirements and to fit in a small form factor, the demonstrator consists only of the necessary components. See Figure 3.11. Therefore, we developed the resources for easy integration and evaluation of the demonstrator on the existing evaluation platform.
A closer description of the demonstrator components as well as differences from the evaluation platform are given in the following chapters.

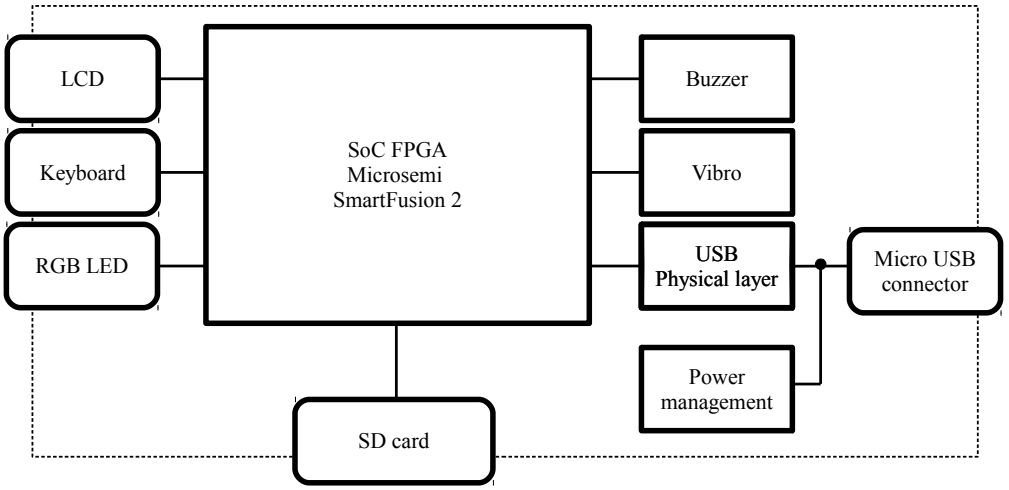

Figure 3.11: Demonstrator 2 and 3 hardware platform.

3.5.1 Microsemi SmartFusion2

Demonstrator 2 is based on Microsemi SmartFusion2 (SF2), a system on chip (SoC), which integrates a flash-based FPGA fabric and an 166 MHz ARM Cortex-M3 processor. There were several reasons for the SF2 choice, rather than similar Altera or Xilinx SoC devices: **a)** The main advantage of the SF2 is especially its low power consumption. This is required from a USB device which is bus powered. **b)** The SF2 also features a very simple ARM-M3 processor, fairly less complex than ARM-A9 in the Altera or Xilinx devices. **c)** The integrated configuration memory ensures integrity of the device and eliminates the need for an additional external memory.

3.5.2 USB Connectivity

The physical layer of the USB stack is governed by an external chip Microchip USB3300. The same chip is used in the evaluation platform. It is connected directly to the USB bus wires on one side and on the other side it contacts the USB Low Pin Count Interface (ULPI) of the SF2. The other layers of the USB stack are controlled by USB hardware controller and bare metal MSS_USB driver by Microsemi. The driver is configured to act as USB Mass storage class device. The class is natively supported across the common operating systems without the need of installing any additional driver or special software. The USB MSS_API consists of these nine call-back functions:

Listing 3.3: USB driver API.

```
uint8_t* usb_media_inquiry(uint8_t lun, uint32_t *len);
uint8_t  usb_media_init (uint8_t lun);
```

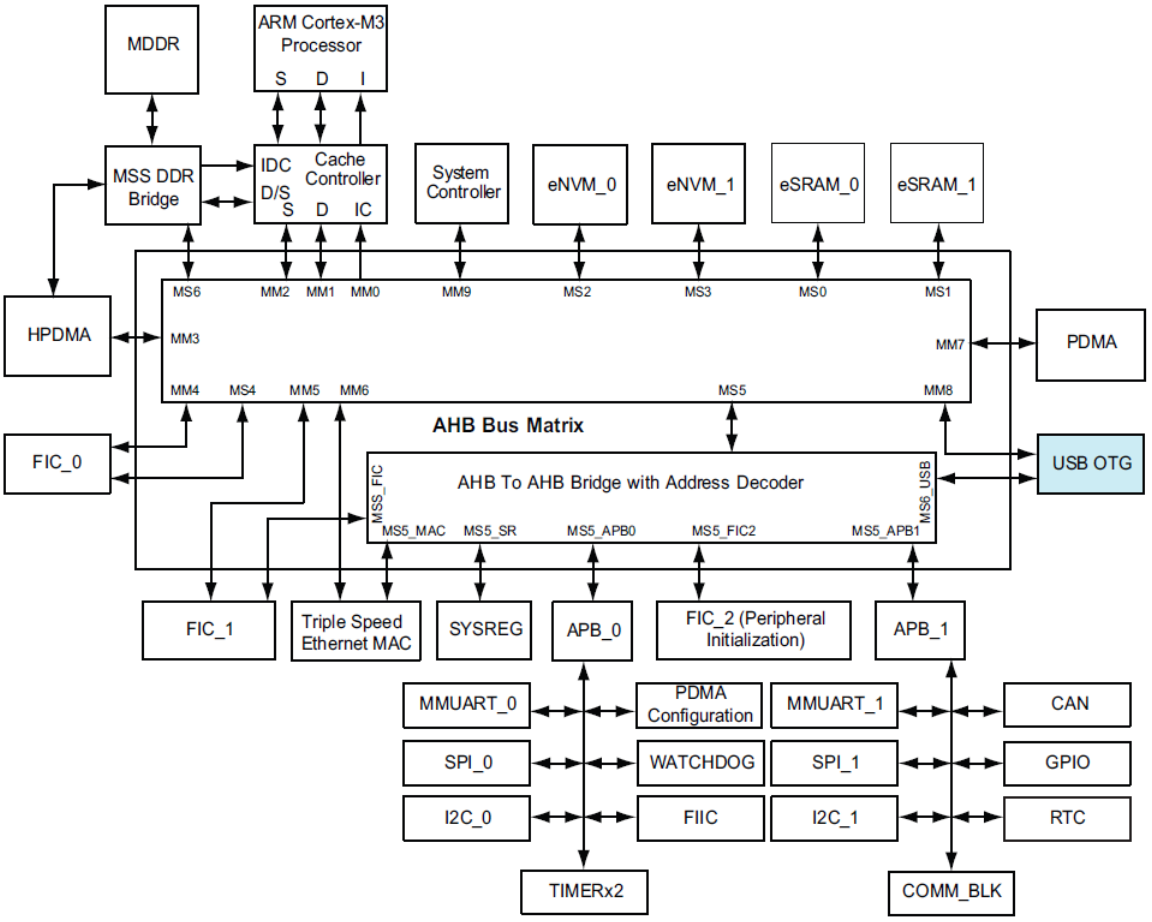

Figure 3.12: Microcontroller subsystem. From [15].

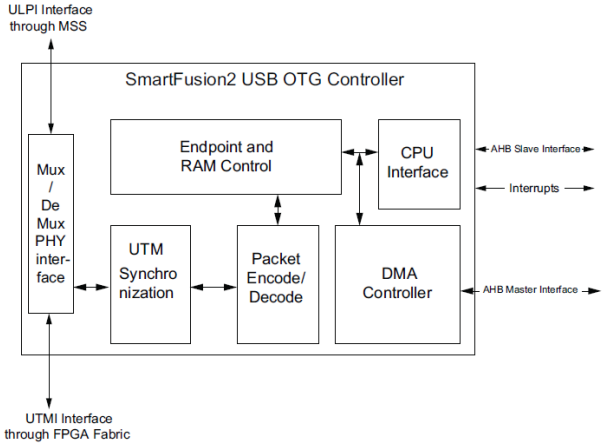

Figure 3.13: USB OTG hardware controller. From [15].

```
uint8_t  usb_media_get_capacity(uint8_t lun,
                                uint32_t *no_of_blocks, uint32_t *block_size);
uint8_t  usb_media_is_ready(uint8_t lun);
uint8_t  usb_media_is_write_protected(uint8_t lun);
uint32_t usb_media_read(uint8_t lun, uint8_t **buf, uint32_t lba, uint32_t len);
uint8_t* usb_media_acquire_write_buf(uint8_t lun, uint32_t lba, uint32_t *len);
uint32_t usb_media_write_ready(uint8_t lun, uint32_t lba, uint32_t len);
uint8_t  usb_media_get_max_lun(void);
```

Reading data: The function pointed by the `usb_media_read` function pointer is called with the logical block address and the length as parameters when the host wants to read the data from the storage medium. Application must provide a buffer and its length which can be sent to the host.

Writing data: The function pointed by the `usb_media_acquire_write_buffer` function pointer is called with the logical block address as a parameter when the host wants to write data into the storage medium. The function pointed by the `usb_media_write_ready` function pointer is called with the logical block address and the length as parameters when the data to be written is received from the host and is ready to be written on the storage medium. The data is stored in the previously provided write buffer using `usb_media_acquire_write_buffer`.

3.5.3 SD Card

A Secure Digital (SDHC/SDXC) card is connected to the FPGA fabric via general purpose input/output pins. The FPGA communicates with the card in the high speed mode with 3.3V signalling at frequency up to 50 MHz allowing a theoretical throughput up to 25 MB/sec [1]. The FPGA generates the clock for the bus and uses one bidirectional serial signal for commands and four bidirectional serial signals for 4-bit data tuples.

Listing 3.4: SD card interface.

```
entity sdc_if is
port (

    CLK                 : in    std_logic                    ;
    RSTN                : in    std_logic                    ;

    -- interface to APB (control)
    PSELx               : in   std_logic                     ;
    PADDR               : in   std_logic_vector( 31 downto 0);
    PENABLE             : in   std_logic                     ;
    PRDATA              : out  std_logic_vector( 31 downto 0);
    PWDATA              : in   std_logic_vector( 31 downto 0);
    PWRITE              : in   std_logic                     ;
    PREADY              : out  std_logic                     ;

    -- interface to BLACK_FIFO (data)
    BLACK_FIFO_TX_Q     : in    std_logic_vector(31 downto 0);
    BLACK_FIFO_TX_RE    : out   std_logic                    ;
    BLACK_FIFO_TX_RDCNT : in    std_logic_vector( 9 downto 0);
```

```
    BLACK_FIFO_RX_D     : out   std_logic_vector(31 downto 0);
    BLACK_FIFO_RX_WE    : out   std_logic                     ;
    BLACK_FIFO_RX_RDCNT : in    std_logic_vector(10 downto 0);

    -- SD card I/O signals
    SD_CLK              : out   std_logic                     ;
    SD_CMD              : inout std_logic                     ;
    SD_DAT              : inout std_logic_vector( 3 downto 0)

);
end sdc_if;
```

The atomic portion of data of the SD card is a 512-byte sector accompanied by a 16-bit cyclic redundancy code (CRC). The CRC is computed on the transferred data in the SD card interface implemented in the FPGA fabric. The interface is controlled by two final state machines, one for command and one for data. The data come in/go out through the black FIFO directly from/to the encryptor, so, the path is separated from the command one, as the commands come in plain from the ARM via the AMBA bus.
The speed of data transfer to/from an SD card depends on how often the data flow is interrupted. The seek time of successive sectors is significantly reduced comparing to random access. The data state machine used in Demonstrator 2 is optimized to achieve the highest possible speed.

3.5.4 Keyboard and Display

The user interface of Demonstrator 2 consists of a built-in touch capacitive keyboard and a non-illuminated display. The display indicates the state of the device and the result of the operations made by the user. The full QWERTY keyboard allows control of the device including also entering the passphrase.

3.5.5 Indication Elements

Demonstrator 2 is equipped with three additional indication elements apart from the display. It has a) multi-color LED, b) beeper and c) vibrating buzzer. The LED instantly shows the state of the device (i.e. deleted, enrolled, activated), flashes on data transfer and together with the two other indicators signals keystrokes. Each of the indicators can be turned off for a quiet operation.

3.5.6 Power Supply

The power supply is another crucial part of the Demonstrator 2. The device is powered from the bus. USB version 2.0 specification limits the current consumption of a device up to five units (1 unit is 100mA). To meet this specification the whole device was carefully designed to reduce the number of necessary components. See Figure 3.14.
The 5V power supply from USB is stepped down to 3.3V by a linear regulator. It ensures stable and low noise power supply for the PLL, crucial for the reliable PLL-based true random generator inside. The other, 1.2V switching, regulator has high efficiency which results to low current consumption. The rest of the components are powered from the same line as PLL, isolated by proper filters.

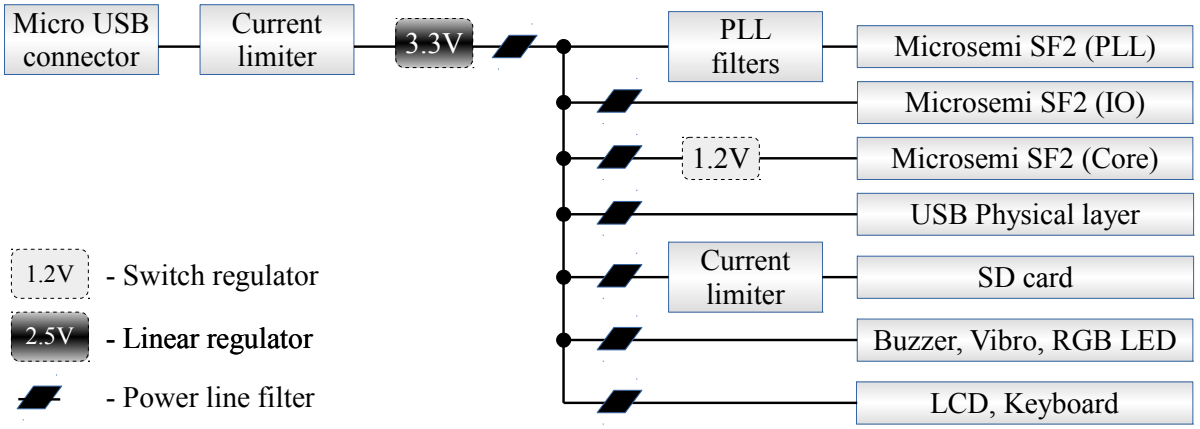

Figure 3.14: Power supply of the Demonstrator 2 and 3.

3.5.7 Mechanical Design

The mechanical design is not directly a part of the HECTOR project, but it is necessary for the demonstrator to operate and provides some of the security features.
The device is housed in a special compact case, which ensures integrity and functionality. See Figure 3.15. The case consists of a custom milled aluminium lid and a top panel featuring keyboard, display and the other indication elements.

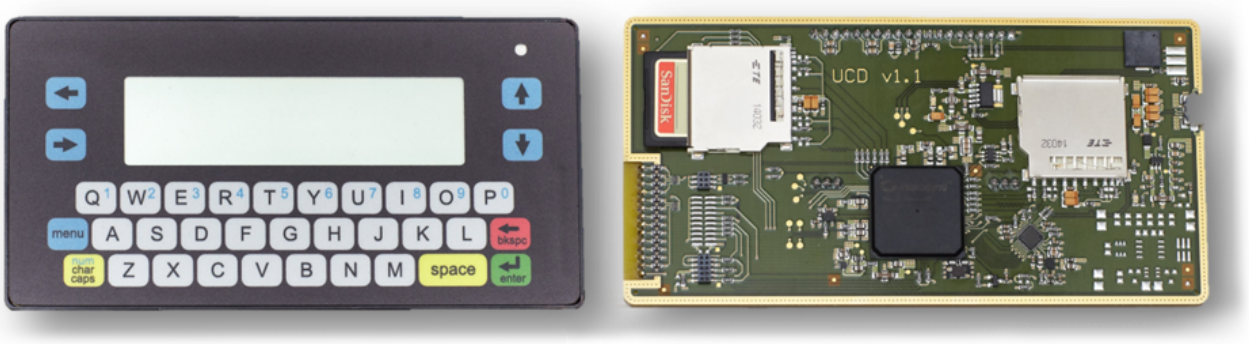

Figure 3.15: Demonstrator 2 and 3 HW platform.

The panel is made of several layers: from the top layer with keyboard description and its electronics, through the interlayers, down to the bottom layer where the display connects to the main printed circuit board. See Figure 3.16.
The aluminium lid attenuates EM emissions, provides good heat dissipation and makes the whole device mechanically robust. The lid also forms a barrier for potting the electronic assembly in epoxy which prevents the device from disassembling and provides tamper evidence.
The dimensions of the device are similar to the dimensions of a common smart phone. That form factor allows integration of a full QWERTY keyboard for a comfortable entry of complex passphrases.

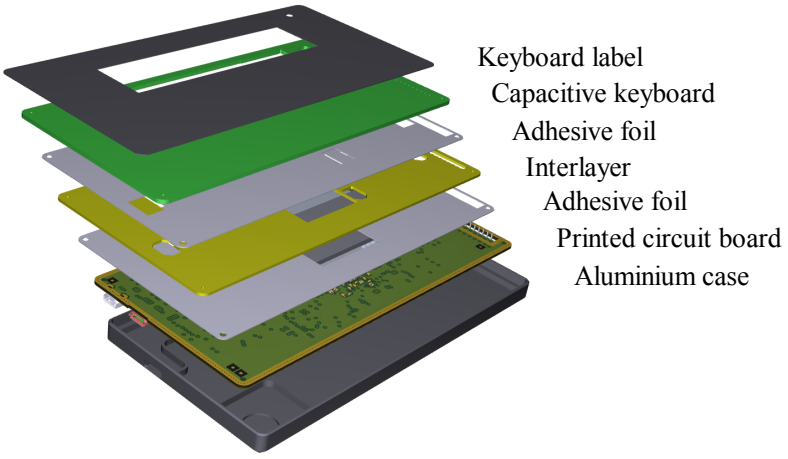

Figure 3.16: Layers of the Demonstrator 2 device.

3.6 Scope of the Evaluation

The secure USB storage device contains encrypted data that can be unlocked by supplying a user passphrase code through the built-in demonstrator keyboard. In principle USB storage devices are physically controlled by the user of the device. Security is provided by the protection of the device itself and by the fact that the owner is the only person that knows the passphrase. For evaluation of a secure USB storage device it is common practice to use the so-called "Lost and found" attack scenario. This means that the device must be capable to resist attacks to the data at rest when it falls into wrong hands - without the passphrase to be known. Examples are numerous by which high officials lose USB sticks while travelling. The Demonstrator 2 USB storage device must protect data against this scenario. When found (or stolen), an attacker can only try a limited number of passphrases before the device will permanently lock its data. In practice an attacker can disassemble the storage device and try to extract information about the cryptographic keys inside non-volatile memory, or obtain information that discloses the passphrase. During this stage physical attacks on the device are in scope (see Section 5.3).
In rare cases more sophisticated attacks might be envisioned, such as a two-stage attack. It must be noted that this is a scenario that has a high "James Bond" level and it is likely that this will only be done at state level espionage. However, if the value of the information is such high that it justifies this attack scenario, it is not likely that a USB device will be used as a transfer medium.

1. During the first stage the storage device is temporarily stolen and modified with a bug that aims to collect the passphrase. Typically a keyboard logger is installed that records the key-presses while the legitimate user unlocks the device. After modification the device is returned to the legitimate user. If the attack is done correctly, there is no visible evidence that the device has been tampered with.

2. During the second stage the storage device is stolen again with the passphrase stored by the bug. Using the passphrase, the device can be opened and all data copied.

Demonstrator 2 is able to resist against attacks using the "Lost and found" scenario. Once found, an attacker can disassemble the device by any applicable means since it does not have to be returned in good condition to the legitimate user. This means that the enclosure of the demonstrator does not have to be tamper evident to protect against modification or bug insertion.

3.7 Conformance to Requirements

The following table shows the current status of Demonstrator 2 requirements defined in D 1.2. All the requirements which need to be justified on the final hardware and software will be reported in the upcomming deliverables D4.2 and D4.3.

Requirement	Current status	Remark
Required building blocks		
TRNG block	PLL-TRNG	See 3.3.4
PUF block	TERO-PUF	See 3.3.2
Error correcting algorithm	Several options	See 3.3.2
Hash function (for the PUF and passphrase derivation)	ASCON	Used as the function for key derivation from the PUF and the passphrase
Authenticated encryption algorithm	ASCON	See 3.3.3
Functional requirements & properties		
Once powered on, the device requires less than 3 seconds to reach a ready state after which it can be used for its intended purpose	To verify on the final hardware	–
The device offers protection against passive physical attacks (side channel attacks)	Outside the scope	See 3.6
The device offers protection against key-logger attacks on the host PC	Compliant	See 3.5.4
The device reveals no sensitive information once powered off, in particular no offline dictionary attacks is possible on the user passphrase	Compliant	See 3.3.1
The probability of any authentication failure over the system life is lower than 10^{-4}	$P_{\text{fail}} = 10^{-4}$ (influenced by the applied error correcting code and the resulting bit error probability of the PUF)	See 3.3.2
Block level encryption	s Compliant	See 3.3.3
The speed at which data can be read from the host PC must be at least 2 MB/s	To verify on the final hardware	–
The speed at which data can be written by the host PC must be at least 2 MB/s	To verify on the final hardware	–
The user does not have to install any extra driver	USB Mass Storage used	See 3.5.2
The device does not need any external energy source, runs on power from the USB port	To verify on the final device	–

Additional IVs & MACs is stored in non-volatile memory	Nonces and tags will be stored along with the encrypted user data on the SD card.	See 3.2.2
IV is provided by the TRNG	Nonces will be derived from TRNG and write counter.	See 3.2.2
Requirements on hardware		
Demonstrator 2 includes a display for user interaction	Non-illuminated display included	See 3.5.4
Demonstrator 2 includes a keyboard for user interaction	Capacitive keyboard included	See 3.5.4
Demonstrator 2 includes at least one flash memory device	One removable SD card included	See 3.5.3
Demonstrator 2 limits EM emissions	Aluminium case used	See 3.5.7
Demonstrator 2 provides tamper evidence	Possible potting with epoxy	See 3.5.7
Demonstrator 2 is powered from a single USB port	Compliant	See 3.5.6
Requirements on the cryptographic primitives		
The PUF requires less power than 20 mW	To verify on the final hardware	–
The PUF requires less than 5k logic cells to implement	< 4.5k	See Deliverable 2.2
The failure rate of the PUF over the system life is lower than 10^{-4}	To verify on the final hardware	–
TRNG complies with the Class PTG.2 AIS 20/31 requirements	Compliant	See 3.3.4
TRNG achieves an output data rate of at least 10 kbits/s	> 500 kbits/s	–
TRNG block requires less than 100 mW	To verify on the final hardware	–
TRNG occupies less than 2 k logic cells	< 1 k	–
The dedicated startup tests require less than 1 second to complete	< 500 ms	–
AE algorithm is protected against side channel analysis	Outside the scope	See 3.6
AE algorithm achieves a throughput of at least 2 MB/s	To verify on the final hardware	–
AE algorithm require less than 10 k logic cells to implement	< 3k	See Deliverable 3.2
AE algorithm is optimized for 512-byte plaintexts	Compliant	See 3.3.3

Chapter 4

Demonstrator 3: Secure Messaging Device

4.1 Motivation

Enabling the ability for users to communicate securely remains today one of the major use cases of cryptography. Popular messaging applications such as Facebook Messenger, Whatsapp or Apple iMessage are able to handle traffic load of several billions of messages everyday. When one adds to that the volume of data sent via text message or emails, the importance of protecting those communications becomes obvious. Users concerned with security can already turn to PGP or S/MIME for email, or choose among the messaging apps the ones that claim to cryptographically protect user messages. These solutions however are designed to run on top of an operating system on commodity hardware (like a PC, a tablet or a phone) and cannot directly make use of the hardware-aware cryptography primitives developed in the framework of the HECTOR project. On the one hand the usage of the PUF coupled with user's passphrase can be used as a strong 2 factor authentication protocol that prevents an outsider from communicating with a peer even if the whole device is compromised. On the other hand the use of sponge based primitives dispenses from using ad hoc constructions to guarantee not only that messages are confidential and authenticated but also that the whole *stream of messages* hasn't been tampered with (replay-protection, reorder-protection).
On top of being inspired by the MICRONIC **hardware platform**, the aim of Demonstrator 3 is to demonstrate that the cryptographic primitives developed within HECTOR can successfully be applied in high-end real-world data security applications in which user and device authentication must be strong and confidential messages are both encrypted and authenticated.

4.2 Functional Description

In the following functional description the building blocks developed along the project are used via the following notations :

- $PUF()$ refers to the PUF principle described in Section 4.3.2
- *Encode*, *Decode* refers to the error correction scheme described in Section 4.3.2
- $\mathcal{AE}_{enc}$, $\mathcal{AE}_{dec}$, $\mathcal{SE}_{init}$, $\mathcal{SE}_{Wrap}$, $\mathcal{SE}_{Unwrap}$ refers to the authenticated encryption scheme and the associated stream-encryption scheme described in Section 4.3.4

- PP refers to the user passphrase described in Section 4.3.1

4.2.1 Initialization

The `Initialization` step is run with 2 devices, `deviceA` and `deviceB`, that have never been turned on before. Both those devices are connected via USB to a `SecureHost`. At the end of this phase both devices will be paired and they will be able to communicate over an untrusted channel.

1. Connect `deviceA` and `deviceB` to `SecureHost`
2. `deviceA` and `deviceB` both execute the following :
   (a) $PP \longleftarrow TRNG(96)$;
   (b) $K_{PUF}||Mask \longleftarrow PUF()$;
   (c) $K_{PUF}||S_{K_{PUF}} \longleftarrow Encode(K_{PUF})$;
   (d) $HD \longleftarrow S_{K_{PUF}} \oplus Mask$;
   (e) $(\emptyset, T_{HD}) \longleftarrow \mathcal{AE}_{enc}(PP, 0^{128}, HD, \emptyset)$;
   (f) $R \longleftarrow TRNG(128)$;
   (g) Send R to `SecureHost`;
3. `SecureHost` executes the following
   (a) receives R_A from `deviceA`;
   (b) receives R_B from `deviceB`;
   (c) $K_{comm} \longleftarrow \mathcal{H}(R_A||R_B)$;
   (d) Send K_{comm} to `deviceA`;
   (e) Send K_{comm} to `deviceB`;
   (f) Delete K_{comm} ;
4. `deviceA` and `deviceB` both execute the following
   (a) receives K_{comm} from `SecureHost`;
   (b) $(C_{comm}, T_{comm}) \longleftarrow \mathcal{AE}_{enc}(PP, K_{PUF}, \emptyset, K_{comm})$;
   (c) $U \longleftarrow (HD, T_{HD}, C_{comm}, T_{comm}, 0, 0)$;
   (d) U is stored in the eNVM.

4.2.2 Powering on the Device

The `PowerOn` step is run with a `device` connected via USB to an `UntrustedHost`. At the end of this phase the `device` is on, unlocked and ready to communicate over an untrusted channel with the device it was previously paired with.

1. Connect `device` to `UntrustedHost`
2. `User` inputs $\widetilde{PP}$ on `device`

(a) Retrieve U from eNVM ;
(b) if $U[4] > limit$, delete U and $reset$;
(c) $s \longleftarrow \mathcal{AE}_{dec}(\widetilde{PP}, 0^{128}, U[0], \emptyset, U[1])$;
(d) if $s = \perp$ then $U[4] \longleftarrow U[4] + 1$, $reset$;
(e) $\widetilde{K_{PUF}}||\widetilde{Mask} \longleftarrow PUF()$;
(f) $\widetilde{S_{K_{PUF}}} \longleftarrow S_{K_{PUF}} \oplus Mask \oplus \widetilde{Mask})$
(g) $K'_{PUF} \longleftarrow Decode(\widetilde{K_{PUF}}||\widetilde{S_{K_{PUF}}})$;
(h) $\widetilde{K_{comm}} \longleftarrow \mathcal{AE}_{dec}(\widetilde{PP}, K'_{PUF}, \emptyset, U[2], U[3])$;
(i) if $\widetilde{K_{comm}} = \perp$ then $U[4] \longleftarrow U[4] + 1$, $reset$;
(j) $U[4] \longleftarrow 0$
(k) $K_{comm} \longleftarrow \widetilde{K_{comm}}$
(l) $IV_{syn} \longleftarrow TRNG(128)$
(m) $(C_{syn}, T_{syn}) \longleftarrow \mathcal{AE}_{enc}(K_{comm}, IV, \texttt{0x01}||\texttt{0x00}^{15}, U[5] + 1)$
(n) $U[5] \longleftarrow U[5] + 1$;
(o) U is stored in the eNVM.

3. Send $(\texttt{cmd}, IV_{syn}, C_{syn}, T_{syn})$ to the peer.
4. Stream encryption initialization : $S_{\mathcal{SE}} \longleftarrow \mathcal{SE}_{init}(K_{comm}, U[5])$

4.2.3 Receiving `cmd` Messages

Upon receiving a message of type `cmd` the device will proceed to update its internal state and initialize a new stream encryption state.

1. Receive $(\texttt{cmd}, IV_{syn}, C_{syn}, T_{syn})$ from the peer.
2. $U_5 \longleftarrow \mathcal{AE}_{dec}(K_{comm}, IV_{syn}, \texttt{0x01}||\texttt{0x00}^{15}, C_{syn}, T_{syn})$;
3. if $U_5 = \perp$ then delete U, $reset$;
4. if $U_5 \neq U[5] + 1$ then delete U, $reset$;
5. $U[5] \longleftarrow U_5$;
6. Stream encryption initialization : $S_{\mathcal{SE}} \longleftarrow \mathcal{SE}_{init}(K_{comm}, U[5])$.

4.2.4 Sending `txt` Message

The length of messages typed up by the `User` is counted in characters and is limited to 159 bytes. To avoid traffic analysis all messages are padded with `0x00` to 159 bytes. It is then wrapped into the current stream encryption state and sent with the associated tag to the peer.

1. `User` provides a message msg, $l = |msg|$;
2. $M \longleftarrow msg||\texttt{0x00}^{159-l}$
3. $(S_{\mathcal{SE}}, C_{txt}, T_{txt}) \longleftarrow \mathcal{SE}_{Wrap}(S_{\mathcal{SE}}, \texttt{0x00}^{16}, l||M)$;
4. Send $(\texttt{txt}, l, C_{txt}, T_{txt})$ to the peer.

4.2.5 Receiving `txt` Messages

Upon receiving a message of type `txt` the device will proceed to decrypt and authenticate the message: if it is a valid message then it will be shown to the user, if not then the message will be trashed and the device will be restarted to set up a new stream.

1. Receive $(\texttt{txt}, l, C_{txt}, T_{txt})$ from the peer.
2. $(S_{\mathcal{SE}}, l||M) \longleftarrow \mathcal{SE}_{Unwrap}(S_{\mathcal{SE}}, \texttt{0x00}^{16}, C_{txt}, T_{txt})$;
3. if $l||M = \perp$ then delete *reset* ;
4. display the first l bytes of M to `User`

4.3 Building Blocks

4.3.1 Passphrase

The passphrase is a randomly chosen string of 16 characters in the alphabet $\mathcal{A} = [A-Z, a-z, 0-9, @, \&]$. Since $|\mathcal{A}| = 64$ we can claim that the user passphrase contributes 96 bits of entropy.
The complete alphabet the device recognizes contains also the additional special characters $:!?\_|.,*+/$ as was the case in Section 3.3.1.

4.3.2 TERO PUF and Extractor

As the two-factor authentication protocol is the same for both Demonstrator 2 and Demonstrator 3 we refer the reader to Section 3.3.2

4.3.3 PLL-based TRNG with Embedded Tests

As the two-factor authentication protocol is the same for both Demonstrator 2 and Demonstrator 3 we refer the reader to Section 3.3.4

4.3.4 ASCON & SpongeWrap

ASCON The ASCON [8] specification describes a family of authenticated encryption designs $ASCON_{a,b} - k - r$ where

- k is the key length, $k \leq 128$;
- r is the "rate" otherwise understood as the data block size;
- a is the number of repetitions of the permutation p during the initialization and finalization procedure;
- b is the number of repetitions of the permutation p during the processing of data.

Each family member is a pair of an authenticated encryption algorithm $\mathcal{E}_{a,b,k,r}$ and a decryption algorithm $\mathcal{D}_{a,b,k,r}$. In the framework of Work Package 4 the consortium agreed on the use of a instantiation dubbed "ASCON-128a" which is short for $Ascon_{12,8} - 128 - 128$. We chose in this document to denote the corresponding encryption and decryption algorithms $\mathcal{AE}_{enc}$, $\mathcal{AE}_{dec}$.
The inputs for the authenticated encryption procedure $\mathcal{AE}_{enc}$ are

- the secret key K of 128 bits;
- the nonce N of 128 bits;
- the associated data A;
- the plaintext P.

The output of the authenticated encryption procedure is an authenticated ciphertext C of exactly the same length as the plaintext P, and an authentication tag T of 128 bits, which authenticates both A and P:

$$\mathcal{AE}_{enc}(K, N, A, P) = (C, T)$$

The inputs for decryption and verification procedure $\mathcal{AE}_{dec}$ are

- the secret key K of 128 bits;
- the nonce N of 128 bits;
- the associated data A;
- the ciphertext C;
- the tag T of 128 bits.

The output of the decryption and verification procedure is the plaintext P if the verification of the tag is correct or $\perp$ if the verification of the tag fails:

$$\mathcal{AE}_{dec}(K, N, A, C, T) \in \{P, \perp\}$$

As one can read in the Section 4.2, in the framework of the work package WP4 AEAD is used in 4 ways:

1. Integrity only mode to protect the authenticity of the helper data;
2. Authenticated encryption of the communication key K_{comm};
3. Authenticated encryption of the `cmd` messages;
4. Stream encryption of the messages exchanged within a session between the peers.

Integrity only In this case we use the ASCON block to only provide integrity of the helper data before it is used by the PUF mechanism (see 3.3.2 for more details). The authenticated encryption process is therefore done without plaintext. The key is the binary representation of the ASCII user passphrase and is therefore 128 bits long while providing only 96 bits of security. The nonce is set to 0^{128} as the process is run only once at device setup after having chosen the user passphrase uniformly at random.

K_{comm} protection In this case we use the ASCON block to protect the communication key both in integrity and confidentiality. Being the element the densest in entropy we use the passphrase as a key and the post-processed PUF response as an IV. If need be the PUF response will be padded to fit into 128 bits. Such as the previous function this process is run only once during the Initialization step and therefore does not require the use of an additional random IV.

Protection of `cmd` messages In this case we use the ASCON block to protect resynchronization of devices after a tampering by the adversary or after reboot by sending the session number protected both in integrity and confidentiality. Here a random (provided by the TRNG block) IV is necessary and to avoid confusion between `cmd` messages and `txt` messages an authenticated content type is added to the message.

Stream encryption In an analogical manner as the description of MonkeyWrap [5] we describe here a process aiming at authenticating series of messages (with additional data). The stream encryption process is a triplet of algorithm $(\mathcal{SE}_{init}, \mathcal{SE}_{Wrap}, \mathcal{SE}_{Unwrap})$ The inputs for the initialization procedure $\mathcal{SE}_{init}$ are

- the secret key K of 128 bits;
- the nonce N of 128 bits.

The output of the initialization procedure is a state of 320 bits $S_{\mathcal{SE}}$:

$$\mathcal{SE}_{init}(K, N) = S_{\mathcal{SE}}$$

The inputs for the wrapping (authenticated encryption) procedure $\mathcal{SE}_{Wrap}$ are

- the state $S_{\mathcal{SE}}$ of 320 bits;
- the associated data A;
- the plaintext P.

The output of the wrapping procedure is the updated value of the state $S_{\mathcal{SE}}$, an authenticated ciphertext C of exactly the same length as the plaintext P, and an authentication tag T of 128 bits, which authenticates the whole sequences of messages:

$$\mathcal{SE}_{Wrap}(S_{\mathcal{SE}}, A, P) = (S_{\mathcal{SE}}, C, T)$$

The inputs for the unwrapping (authenticated decryption) procedure $\mathcal{SE}_{Unwrap}$ are

- the state $S_{\mathcal{SE}}$ of 320 bits ;
- the associated data C ;
- the tag T of 128 bits.

The output of the unwrapping procedure is the updated value of the state $S_{\mathcal{SE}}$ and the plaintext P if the verification of the tag is correct or $\perp$ if the verification of the tag fails:

$$\mathcal{SE}_{Unwrap}(S_{\mathcal{SE}}, A, C, T) \in \{(S_{\mathcal{SE}}, P), \perp\}$$

In the following we describe how to make use of the ASCON hardware block described in Section 4.3.5 to achieve stream encryption as described above. We assume access to the following procedures that update the 320 bits internal state of ASCON implicitly.

- $\mathcal{AE}_{Init}(K, N) = S$, where K is a 128 bits key, N a 128 bits and S a 320 bits internal state;
- $\mathcal{AE}^{S}_{AD}(A) = \emptyset$, where A is a 128 bits block of associated data;

- $\mathcal{AE}^S_{Encrypt}(P_i) = C_i$, where P_i is a 128 bits block of plaintext, C_i a 128 bits block of ciphertext;
- $\mathcal{AE}^S_{LastEncrypt}(\emptyset) = T$, where T is a 128 bits tag;
- $\mathcal{AE}^S_{Decrypt}(C_i) = P_i$, where P_i is a 128 bits block of plaintext, C_i a 128 bits block of ciphertext;
- $\mathcal{AE}^S_{LastDecrypt}(T) \in \{success, \perp\}$, where T is a 128 bits tag and the return value is *success* if the tag verification succeeds and $\perp$ otherwise.

Those procedures can be seen as trivial extensions from the provided hardware interface.
The Init procedure is shown in Algorithm 5.

Algorithm 5: $\mathcal{SE}_{init}$ procedure

Input: $K, |K| = 128, N, |N| = 128$
begin
  $S_{\mathcal{SE}} \longleftarrow \mathcal{AE}_{Init}(K, N)$
  return $S_{\mathcal{SE}}$

We call m the message supplied by the user, l its length in bytes with the restriction that $l \leq 160$, m is padded with 0x00 to 159 bytes. Finally, we call M the 160 byte string $l||m||\texttt{0x00}^{159-l}$ and M_i the i-th block of 16 bytes of M. The wrapping procedure is shown in Algorithm 6.

Algorithm 6: $\mathcal{SE}_{Wrap}$ procedure

Input: $S_{\mathcal{SE}}, M$
begin
  $\mathcal{AE}^{S_{\mathcal{SE}}}_{AD}(\texttt{0x00}^{16})$
  for $i \in \{0, \ldots, 9\}$ **do**
    $C_i \longleftarrow \mathcal{AE}^{S_{\mathcal{SE}}}_{Encrypt}(M_i)$
  $T \longleftarrow \mathcal{AE}^{S_{\mathcal{SE}}}_{LastEncrypt}()$
  return $(S_{\mathcal{SE}}, C, T)$

The unwrapping procedure is shown in Algorithm 7.

4.3.5 Hardware Interface

The general architecture of the hardware block instantiating the different hardware building blocks is shown in Figure 4.1.
The interaction between the different hardware blocks and the software running on the Cortex M3 is done by the Control block whose architecture is shown in Figure 4.2
The control interface communicates with the software through the use of a Command register that activates the different hardware blocks and a Status register that informs the software about the status of the hardware. The figure 4.3 shows how commands are handled by the control block.

Algorithm 7: $\mathcal{SE}_{Unwrap}$ procedure

Input: $S_{\mathcal{SE}}, C, T$
begin
  $\mathcal{AE}_{AD}^{S_{\mathcal{SE}}}(\texttt{0x00}^{16})$
  for $i \in \{0, \ldots, 9\}$ **do**
    $M_i \longleftarrow \mathcal{AE}_{Decrypt}^{S_{\mathcal{SE}}}(C_i)$
  $res \longleftarrow \mathcal{AE}_{LastDecrypt}^{S_{\mathcal{SE}}}(T)$
  if $res = \perp$ **then**
    return $(S_{\mathcal{SE}}, \perp)$
  return $(S_{\mathcal{SE}}, M)$

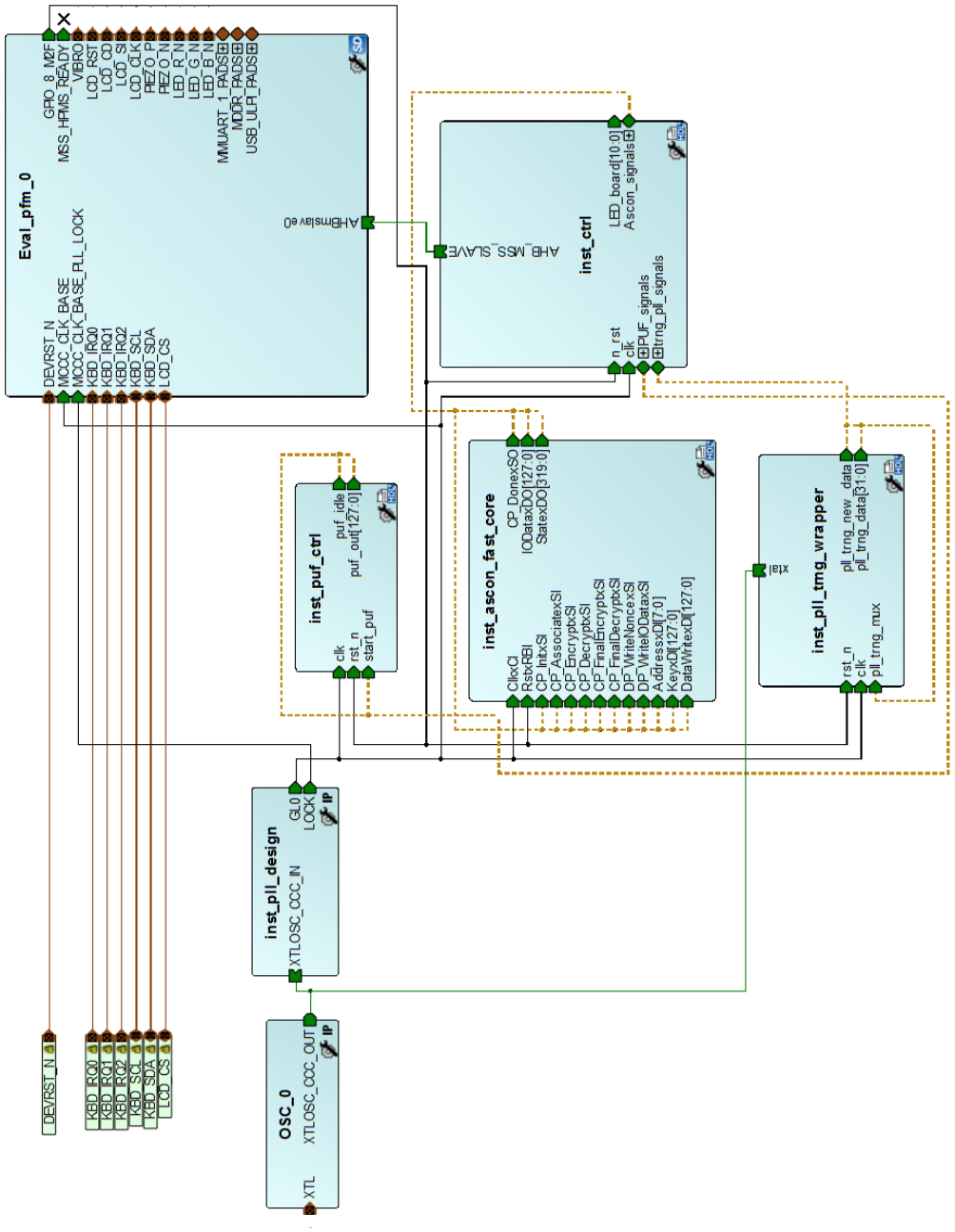

Figure 4.1: Architecture

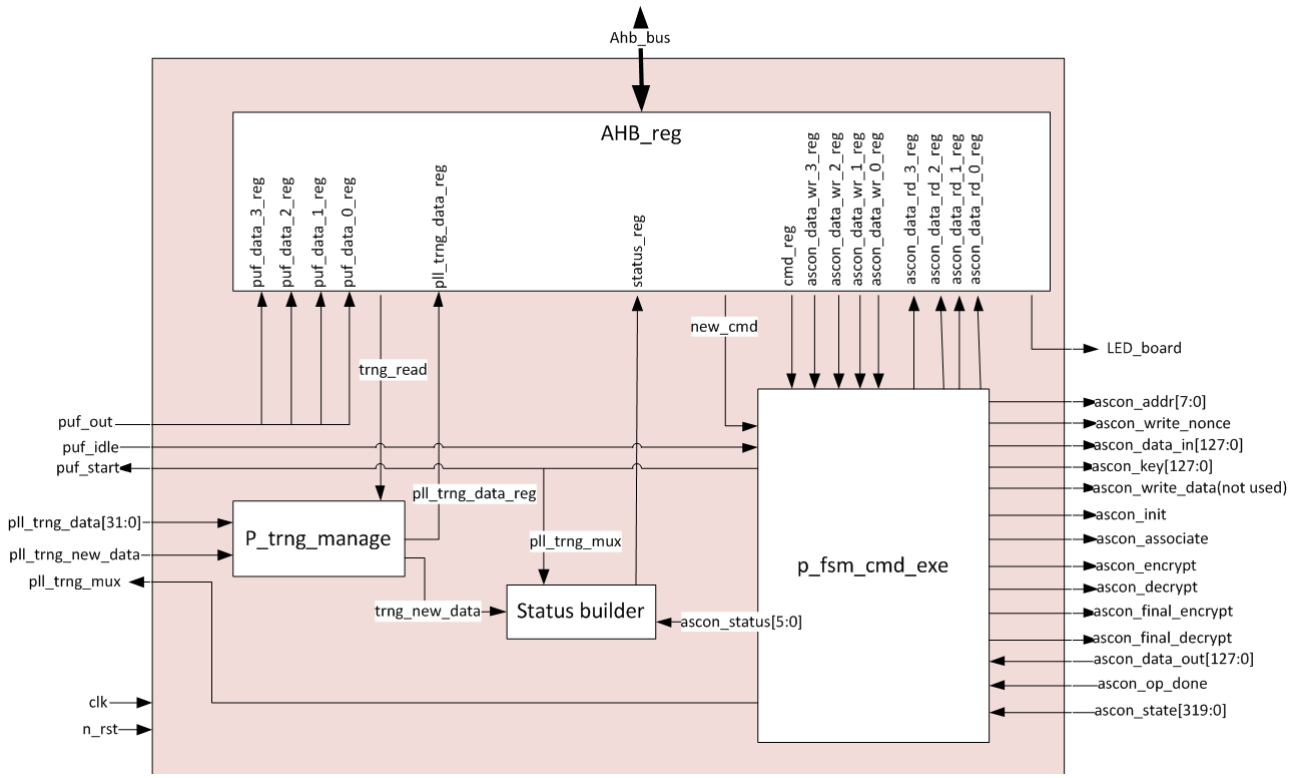

Figure 4.2: Interface of the control block

Command Register

The following table contains the available commands.

ID	Value	Description
MODE_IDLE	0x00000000	Do nothing
START_PUF	0x00000001	Starts PUF computation
PLL_MUX_H	0x00000002	Raw random data before the XOR decimator
PLL_MUX_L	0x00000003	Random data after the XOR decimator
ASC_WR_KEY	0x00000004	Writes the 128 bits key in ASCON block (from DATA_IN)
ASC_WR_NONCE	0x00000005	Writes the 128 nonce in ASCON block (from DATA_IN)
ASC_INIT	0x00000006	Computes initialization phase of ASCON
ASC_ASSO	0x00000007	Processes associated data (from the DATA_IN)
ASC_ENCRYPT	0x00000008	Computes ciphertext block (to DATA_OUT), then computes p^b
ASC_F_ENCRYPT	0x00000009	Computes last ciphertext block (to DATA_OUT), then computes p^a
ASC_DECRYPT	0x0000000A	Computes plaintext block (to DATA_OUT), then computes p^b
ASC_F_DECRYPT	0x0000000B	Computes last plaintext block (to DATA_OUT), then computes p^a
ASC_RD_TAG	0x0000000C	Computes the tag (to DATA_OUT)

As mentioned above those commands are handled by an automaton described in Figure 4.3.

Status Register

The following table describes the status register.

Bit number	Status field name	Description
0	ASCON_INIT	1 when ASCON is initializing
1	ASCON_ASSO	1 when ASCON is processing associated data
2	ASCON_ENCRYPT	1 when ASCON is encrypting
3	ASCON_DECRYPT	1 when ASCON is decrypting
4	ASCON_F_ENCRYPT	1 when ASCON is encrypting
5	ASCON_F_DECRYPT	1 when ASCON is decrypting
6	PUF_BUSY	1 when the PUF is being computed
7	NEW_TRNG	1 when a new word of random data is available
8	TRNG_MUX_STATE	State of the control signal of the TRNG mux
9	unused	
⋮	⋮	⋮
31	unused	

4.4 Design Rationale

Demonstrator 3 is a portable secure messaging device that not only encrypts and authenticates data but also authenticates the *stream* of messages sent and received by the peers (as well as their order).
The communication between devices is provided through 2 untrusted PC that only serve as network interfaces. In theory, with appropriate routing two such devices could communicate

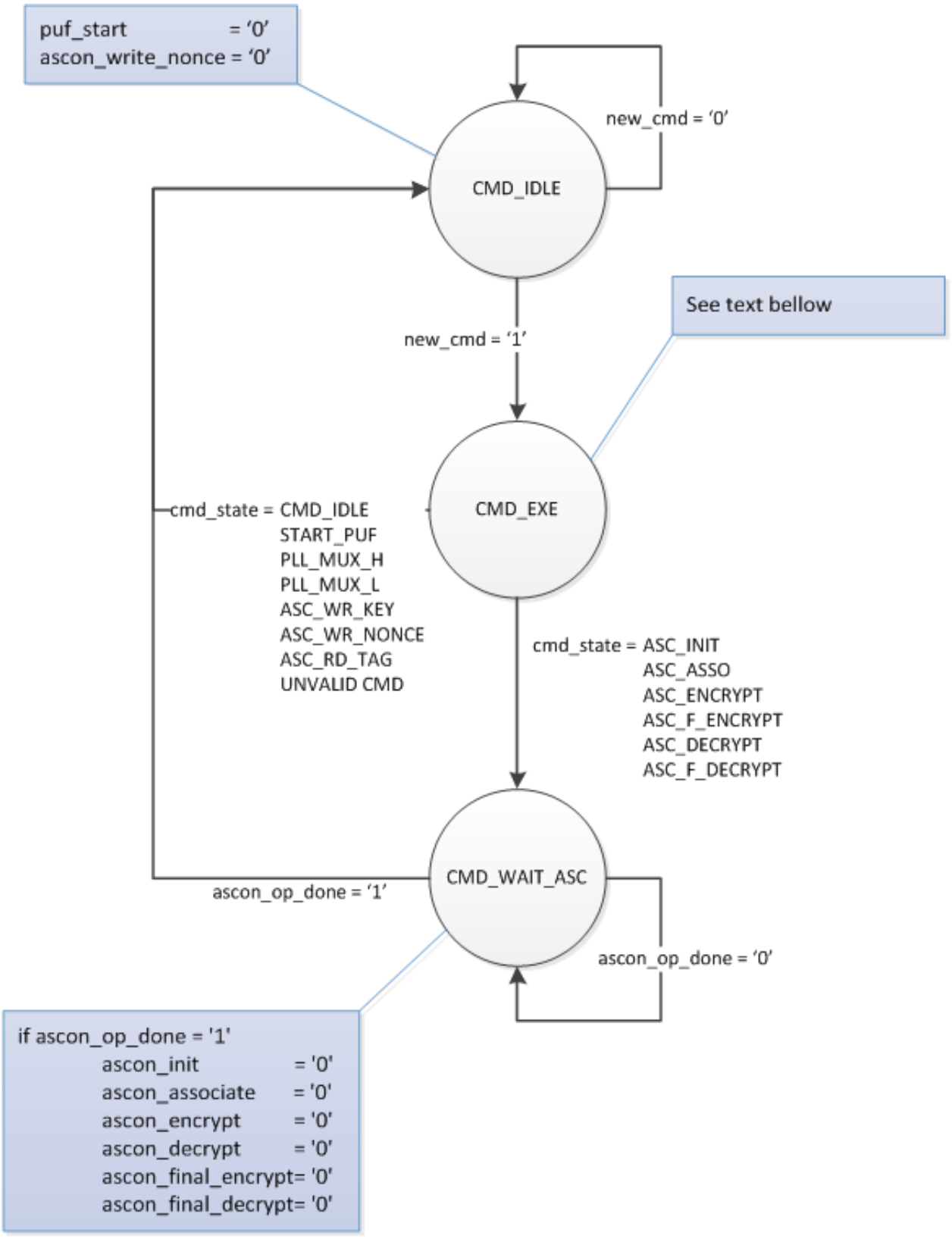

Figure 4.3: Handling the Command register

over the internet. In the context of the demonstration however, the two untrusted PC will be directly connected with an ethernet cable.
Sharing its hardware platform with Demonstrator 2 (see 3.5), Demonstrator 3 also makes use of the Microsemi Smartfusion2 System on Chip, which is a low power, reconfigurable FPGA device with a 32-bit ARM processor.
To match what users are accustomed to, the size of messages can range from 1 to 160 text characters. Those messages are encrypted and authenticated using the sponge-based authenticated encryption ASCON used both as a standalone AE scheme (in the 2-factor authentication scheme as well as in the command messages) and as an instance of the MonkeyDuplex construction in a stream encryption derived from MonkeyWrap. Thus we guarantee that no message will be shown to the user if it has been modified, replayed, served out of order or skipped.
When the device is turned off, the only informantion available in cleartext is the helper data associated with the PUF (authenticated upon powerOn), the counter of false passphrase entries and the session number (initially 0 for both peers and updated by each reset or on session failure). Therefore no sensitive data is stored in plaintext.
The two-factor authentication protocol involves a user passphrase that is never transmitted outside the device and a PUF that is by definition unique to a device. The device is immune against key-logger attacks since the user authenticates directly on the device. Theft-protection is provided as long as the user passphrase remains unknown to the attacker.

4.5 Hardware Platform

As the hardware platform is the same in Demonstrator 2 and Demonstrator 3 we refer the reader to Section 3.5. We note however that the SD-card mass storage is not used in Demonstrator 3. Additionally the communication between host PC and the hardware platform will be done in a different way. The Demonstrator 3 will use an USB virtual communication port (VCP) driver to create a serial link between PC and device. Afterwards, the PC application can be very simple, because the VCP protocol behaves very similarly to a common serial port. The using of the VCP does not require a special driver. The file with device information is only necessary. The using of the VCP driver on existing hardware eliminates integration of an additional chip (e.g UART to USB converter) and allows us to use one hardware for two different demonstrators.

4.6 Scope of the Evaluation

A typical use case for Demonstrator 3 is that a high ranking official of government or industry uses it for having contact with the basis to discuss matters. The device then guarantees confidentiality and integrity of the communication data. Such secure communication device is able to encrypt its communication channels by using an algorithm and secret keys. Algorithms are generally public while the keys have to be kept secret (Kerckhoff's principle). Therefore the keys stored inside the device are the primary assets and can be the target for an attack. Again, a few different attack scenarios can be envisioned (see Section 5.3).

Scenario 1 An attacker collects encrypted data during a period of time, then steals the communication device, extracts the keys and can in hindsight decrypt the communication streams and make use of the confidential content. This can be interesting for attackers, but limits the value of the obtained decrypted communication. For this attack scenario the device must have

protection against disclosure of the cryptographic keys, but does not have to be tamper evident. Tamper resistance is helpful, but should not be the primary protection method of the device.

Scenario 2 An attacker uses a two-stage approach, by which the cryptographic keys are extracted during a time interval where the device is temporarily not possessed by the owner (temporarily stolen). Assuming proper practical use (which should be reflected in guidance), such time period can range from minutes to maximum a day. Obtaining the cryptographic keys has the benefit for an attacker that he can decrypt and even change communication data in real-time. This increases the value of the attack. To make such attack more difficult the presence of tamper evidence measures is helpful, so it should be noticeable for the legitimate user that the device has been tampered with. Tamper resistance is also helpful since it increases the time an attacker needs to get access to the internals of the device. However, both tamper evidence and tamper resistance should not be the primary protection methods to rely upon. Attackers prefer methods to extract key information in a non-invasive way (without opening the device).

Scenario 3 Besides key-extraction, an attacker might also aim for direct access to keyboard, display and, bif used, audio channels. Built-in bugs may then tap directly into the plaintext part of the communication channel, completely bypassing any cryptographic protection. In the context of the HECTOR project direct attacks at plaintext signals are beyond the scope of the project.

4.7 Conformance to Requirements

The following table shows the current status of Demonstrator 3 requirements defined in D 1.2. All the requirements which need to be justified on the final hardware and software will be reported in the upcomming deliverables D4.2 and D4.3.

Requirement	Current status	Remark
Required building blocks		
TRNG block	PLL-TRNG	See Section 4.3.3
PUF block	TERO-PUF	See Section 4.3.2
Error correcting algorithm	Several options	See Section 4.3.3
Hash function	SHA-3	A software instance will be integrated on the `SecureHost`
Authenticated encryption algorithm	ASCON	See Section 4.3.4
Functional requirements		
"Ready" after 3 seconds	To verify on the final device	–
Protection against side-channel attacks	Out of scope	see Section 4.6
Protection against key-logger on host PC	Compliant	See Section 4.4
Protection against offline password brute-force	Compliant	See Section 4.4
Authentication failure rate $< 10^{-4}$	To verify on the final device	–

160 bytes plaintext	1–160 bytes	See Section 4.4
Doesn't show out of order messages	Compliant	See Section 4.4
Doesn't show replayed messages	Compliant	See Section 4.4
Doesn't show modified messages	Compliant	See Section 4.4
Hardware requirements		
Includes display for user interaction	Non-illuminated display included	See 3.5.4
Includes keyboard for user interaction	Capacitive keyboard included	See 3.5.4
Includes SD-card	One removable SD card included	Not necessary for demonstrator scenario
Limit EM emission	Aluminium case used	See 3.5.7
Provide tamper evidence	Possible potting with epoxy	See 3.5.7
Powered from USB	To verify on the final device	–
Cryptographic requirements		
PUF requires $< 20mW$	To verify on the final hardware	–
PUF requires $< 5k$ logic cells	$< 4.5k$	See Deliverable 2.2
PUF failure rate $< 10^{-4}$	$P_{\text{fail}} = 10^{-4}$ (influenced by the applied error correcting code and the resulting bit error probability of the PUF)	See 3.3.2
TRNG is AIS20/31 PTG.2 compliant	Compliant	See 4.3.3
TRNG output data rate $> 10kbits/s$	$> 500kbits/s$	–
TRNG require $< 100mW$	To verify on the final hardware	–
TRNG requires $< 2k$ logic cells	$< 1k$	–
TRNG startut tests require < 1 second to complete	$< 500ms$	–
AE is protected against side channel analysis	Out of scope	see Section 4.6
AE output data rate is $> 2Mbits/s$	To verify on the final hardware	–
AE requires $< 10k$ logic cells	$< 3k$	See Deliverable 3.2
AE is tailored for 1-160 bytes plaintext	Compliant	See Section 4.4

Chapter 5

Security Evaluation

The security evaluation work within HECTOR will be reported in several deliverables (D2.4, D3.3 and D4.3). These deliverables will explain in more detail how the product will be used (in terms of security), which threat models are envisioned and which attacks are applicable. A vulnerability analysis will show which threats will have a chance to be exploited in a successful attack. If insufficient countermeasures are present to mitigate such potential vulnerabilities, then tests will be formulated to verify the actual sensitivity of the product for that particular threat. The verdict on the overall security resistance will be based on a combination of the results of the vulnerability analysis and the test results.

5.1 Guidance Information

Equally as important as the security performance of the product is any additional guidance information that is provided with the product. Guidance information is the collection of documentation that describes how the product is to be used in order to fulfil its security promises. Guidance documentation usually consists of description of:

- How to use the product (user manual),
- A description to verify the security integrity of the product when it is received by the customer and before it is installed for the first time,
- Any weaknesses that have to be mitigated by additional security measures in order to fulfil the overall security functionality, such as by posing requirements for the physical environment or by limiting its use.

5.2 Role of the Demonstrator Physical Construction

The aim of the HECTOR project is to develop cryptographic hardware building blocks that have good performances, while at the same time are robust against attacks. For **Demonstrator 1** the security will not depend on tamper evidence and tamper resistance. For **Demonstrator 2** the overall product security is slightly increased by some tamper resistance. However, in the practice of commercially viable products the improved security does not outweigh the additional costs for effective tamper resistance. **Demonstrator 3** does benefit of a good tamper evident and tamper resistant construction of the enclosure. However, development of such physical enclosure which is an art by itself is not a goal of HECTOR. For this reason the evaluation of

all demonstrators does not focus on the physical protection as offered by the enclosure, but on the protection provided by the cryptographic building blocks.

5.3 Attacks in Scope

The HECTOR building blocks depending on the use cases and threat models should be resistant against attacks such as side channel analysis, perturbation attacks and fault injection, where applicable in the envisioned use cases. All cryptographic products that are used for high-security applications such as payment and identification should be resistant against these attacks. In these application domains also physical attacks at chip-level are applicable. However, the demonstrators are built using commercial off-the-shelf FPGAs, whose resistance against physical chip attacks is not representative for future commercial products that exploit HECTOR cryptographic building blocks using dedicated chips like ASICs or SOCs.

5.4 Evaluation Assurance Levels

Common Criteria is a standardised evaluation methodology (ISO-15048). This methodology recognises seven levels of evaluation assurance Evaluation Assurance Level, or EAL ranging from the lowest level EAL 1 (basic assurance) to the highest level EAL7 (formally proven assurance).
Common Criteria (CC) evaluations cover the security claims of the product plus the development process and environment. The idea behind this is that you start with the specifications of the security properties of your product. Sets of security requirements are pre-defined for important types of products, such as passports, payment cards, tachographs and other security-relevant IT equipment. These sets of requirements are contained in Protection Profiles, which act as templates for so-called Security Targets.
The Security Target contains the set of requirements that is claimed by a developer for a specific product before the CC evaluation is started. If the development process is well-organized, then the final product will provable contain the security properties as specified. In the context of the HECTOR project a full CC evaluation cannot be applied. CC is developed for real commercial products that are created by companies to serve a function in the market. In case of the HECTOR demonstrators there is no real product, as well as no suitable Protection Profile for the demonstrators to be used as template for the security requirements, the devices are developed by different developers and there is no product production. Therefore the evaluation of the development which is a considerable part of a CC evaluation is not applicable. However, the technical parts of CC can be used to evaluate security relevant behaviour of a product. We selected the following parts from CC which are meaningful for the HECTOR project:

Scoping of the Target Of Evaluation (TOE) This step is described in this document (see Sections 2.5, 3.6, 4.6,). It is based on the use-case of the IT product and envisions how it is being used in order to be able to understand the product design and formulate meaningful attack scenarios.

Design analysis A study of design documents is performed to understand the structure of the product in terms of subsystems and modules and their roles in achieving the desired security functionality.

Vulnerability analysis A vulnerability analysis is usually done in conjunction with formulation of attack scenarios. It is based on the security properties of the product, which may or may not allow certain scenarios to be successful. Therefore it is important to have good knowledge of all design considerations and detailed design information before commencing the Vulnerability analysis. At this stage this is a theoretical assessment, based on the knowledge of the design and implementation and experience with attack methods.

Formulation of viable attack scenarios Based on the use-case it is determined which attack scenarios are relevant and which steps are required to perform these scenarios.

Selection of tests and setting up a test plan The Vulnerability Analysis is as said a theoretical exercise. Based on the design some attacks can be ruled-out because applied countermeasures prove to prevent those attacks. A simple example is the use of a PIN retry counter, which prevents brute force attacks. Other attacks rely on inherent physical behaviour of the TOE, which cannot be predicted from the design and implementation alone. An example is how the hardware reacts on perturbation attacks: in case the program can be influenced by light flashes, it might be possible to reset the PIN retry counter.

Rating of reasonable attack scenarios In case the rating value exceeds a defined value, the product has a residual vulnerability. Such known weakness then has to be covered by additional security measures, such as protective software or organizational security measures.

Test execution Penetration testing and analysis of results.

Chapter 6

Summary and Conclusion

This deliverable was dedicated to the detailed specification of the demonstrator. It advances Deliverable 1.2 where requirements on the building blocks were previously specified. The document precisely described the selected primitives, developed in the scope of the work packages WP2 and WP3, which will be integrated in the HECTOR hardware platform. The primitives were selected with the goal to achieve the best security evaluation results in the following task T4.3.
The TRNG implemented in Demonstrator 1 will constitute a tangible device with specific and practical exploitation. In addition, the integration of the independent TRNG ensures its stable and reliable implementation isolated from the other function blocks.
On the contrary, Demonstrators 2 and 3 combine PUF, TRNG and AEAD with other building blocks in a single device. Their integration will bring other challenges and outcomes:

- practical profit in a real device (e.g. security vs effectiveness);
- interference between primitives (e.g. reliability of PUF function surrounded by the other operating blocks).

The integration will benefit also from the collaboration between academic and industrial partners, where scientific and practical experience can be exchanged.
While the goal of the work package WP4 is to demonstrate practical relevance of the HECTOR project results, the designed demonstrators are not intended to be neither final commercial products nor prototypes of them.
It is important to mention that the deliverable synthesized outputs of three different work packages, and involved collaboration of the majority of the partners. Therefore, the consortium had to make many compromises in order to achieve the best combination of the developed features. The main result of the task T4.1 is the final consensus on the content of the demonstrators – the most tangible output of the project. In the upcoming task T4.2 Integration of Hardware Modules, the consortium will progress following this deliverable.

Chapter 7

List of Abbreviations

AEAD	Authenticated Encryption with Associated Data
AE	Authenticated Encryption
AES	Advanced Encryption Standard
ASIC	Application Specific Integrated Circuit
ATX	Advanced Technology eXtended
CC	Common Criteria
CRC	Cyclic Redundancy Code
DC	Delay Chain
DFF	D Flip-Flop
DRNG	Deterministic Random Number Generator
DDR SDRAM	Double Data Rate Synchronous Dynamic Random Access Memory
EAL	Evaluation Assurance Level
EM	Electromagnetic
eNVM	Embedded Non-Volatile Memory
FIPS	Federal information processing standard
FIFO	First In First Out register
FPGA	Field Programmable Gate Array
GCD	Greatest Common Divisor
IV	Initialization Vector
KAT	Known Answer Test
LED	Light Emitting Diode
LVDS	Low Voltage Differential Signaling
LUT	Look-Up Table
MV	Majority Voting
SD	Secure Digital
SF2	Microsemi SmartFusion2
SSI	Synchronous Serial Interface

TOE	Target Of Evaluation
UART	Universal Asynchronous Receiver/Transmitter
PC	Personal Computer
PCB	Printed Circuit Board
PLL	Phase-Locked Loop
PTRNG	Physical True Random Number Generator
PUF	Physical Unclonable Function
RAM	Random Access Memory
RNG	Random Number Generator
RO	Ring Oscillator
RTC	Real Time Counter
SD card	Secure Digital card
SoC	System on Chip
TERO	Transient Effect Ring Oscillator
TRNG	True Random Number Generator
USB	Universal Serial Bus
ULPI	USB Low Pin Count Interface
UART	Universal Asynchronous Receiver Transmitter
VCP	Virtual Communication Port
VCO	Voltage Controlled Oscillator

Bibliography

[1] *SD Specifications, Part 1, Physical Layer Simplified Specification, Version 5.00*. SD Card Association, 2016.

[2] Frederik Armknecht, Roel Maes, Ahmad-Reza Sadeghi, Berk Sunar, and Pim Tuyls. Memory leakage-resilient encryption based on physically unclonable functions. In *Towards Hardware-Intrinsic Security*, pages 135–164. Springer, 2010.

[3] Prof. Hari Balakrishnan and Prof. George Verghese. Electrical Engineering and Computer Science Courses. Technical report, Massachusetts Institute of Technology (MIT), 2011.

[4] Elaine B. Barker and John M. Kelsey. SP 800-90A. Recommendation for Random Number Generation Using Deterministic Random Bit Generators. Technical report, Gaithersburg, MD, United States, 2012.

[5] Guido Bertoni, Joan Daemen, Michael Peeters, Gilles Van Assche, and Ronny Van Keer. CAESAR submission: Ketje v2. Technical report, STMicroelectronics, Radboud University Nmegen, September 2016.

[6] Lilian Bossuet, Xuan Thuy Ngo, Zouha Cherif, and Viktor Fischer. A PUF based on a transient effect ring oscillator and insensitive to locking phenomenon. *IEEE Trans. Emerging Topics Comput.*, 2(1):30–36, 2014.

[7] Jeroen Delvaux, Dawu Gu, Ingrid Verbauwhede, Matthias Hiller, and Meng-Day (Mandel) Yu. *Efficient Fuzzy Extraction of PUF-Induced Secrets: Theory and Applications*, pages 412–431. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[8] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schlaffer. Ascon v1.2, Submission to the CAESAR Competition. Technical report, Institute for Applied Information Processing and Communications Graz University of TechnologyInfineon Technologies Austria AG, September 2016.

[9] A. Fahim. *Clock generators for SoC processors*. Cluwer Academic Publisher, MA, USA, 2004.

[10] V. Fischer and M. Drutarovsky. True random number generator embedded in reconfigurable hardware. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002)*, volume 2523 of *LNCS*, pages 415–430. Redwood Shores, CA, USA, Springer Verlag, 2002.

[11] V. Fischer, M. Drutarovsky, M Simka, and N. Bochard. High performance true random number generator in altera stratix fplds. In *Field-Programmable Logic and Applications (FPL 2004)*, volume 3203 of *LNCS*, pages 555–564. Springer, 2004.

[12] Hyunho Kang, Yohei Hori, Toshihiro Katashita, Manabu Hagiwara, and Keiichi Iwamura. Cryptographie key generation from puf data using efficient fuzzy extractors. In *Advanced Communication Technology (ICACT), 2014 16th International Conference on*, pages 23–26. IEEE, 2014.

[13] W. Killmann and W. Schindler. A proposal for: Functionality classes for random number generators. AIS 20 / AIS31. Technical report, 2011.

[14] Roel Maes, Vincent van der Leest, Erik van der Sluis, and Frans Willems. Secure key generation from biased PUFs: extended version. *Journal of Cryptographic Engineering*, 6(2):121–137, 2016.

[15] Microsemi. Smartfusion2 microcontroller subsystem user guide, ug0331. Technical report, Microsemi Corporation, January 2017. Online. Available at: www.microsemi.com.

[16] L. M. Reyneri, D. Del Corso, and B. Sacco. Oscillatory metastability in homogeneous and inhomogeneous flip-flops. *IEEE Journal of Solid-State Circuits*, 25(1):254–264, Feb 1990.

[17] Vladimir Rožić, Bohan Yang, Wim Dehaene, and Ingrid Verbauwhede. Highly efficient entropy extraction for true random number generators on FPGAs. In *Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7-11, 2015*, pages 116:1–116:6, 2015.

[18] Michal Varchola and Milos Drutarovsky. *New High Entropy Element for FPGA Based True Random Number Generators*, pages 351–365. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.