A framework for assessing the use of SoC SRAMs as Physical Unclonable Functions

Omitted for blind review

***Abstract*—this paper discusses the development of a framework for assessing the use of SoC SRAMs as Physical Unclonable Functions. We evaluate the idea of considering the power-up state of a SRAM cell to identify its properties due to physical mismatches. We also explore the behavior of the memory by performing several power-on resets and gathering memory dumps through an asynchronous serial interface. We then provide a framework to retrieve and process this information and show a typical case where the startup stage shows itself to be a strong candidate to generate a Physical Unclonable Function using static memory cells on an off-the-shelf circuit.**

***Index Terms*— Physical Unclonable Functions, SRAM characterization and evaluation framework.**

I. Introduction

Physical Unclonable Functions (PUFs) are functions that provide a response based on a challenge posed to it and the physical structure of the circuit used to implement it. The goal is to achieve a function that is easy to evaluate but hard to predict before presenting the challenges to it. Among others, it has been reported applications of PUFS on device identification and authentication, binding software to hardware platforms and secure data storage [1].

There is still not a established set of properties a PUF must preset to be useful in security protocols applications but two are considered essential: 1) the output of one PUF instance must be impossible to predict based on the challenge-response pairs (CRP) of another instance and 2) each instance of a PUF must always provide the same output [2].

Some architectures have been proposed to implement such kind of function, many based on non-electronic devices. On the other side, we believe that for massive adoption of PUFs, they must be fully integrated with other electronic circuits to compose a System on Chip (SoC) solution with minimum or no modification on the fabrication process. One of the most promising solution is to implement the PUFs using SRAM memory cells present on modern chips [1].

This paper presents a framework developed for characterizing SRAM based PUFs. The information obtained with this framework will be later used to implement error correction codes for SRAM based PUFs and to help design SRAM cells better suited to be used as PUFs.

II. SRAM Based PUF

A CMOS SRAM cell is a six transistor (6T) device [3] as shown in Fig. 1 and is formed of two cross-coupled inverters and two MOS switches to implement access to the cell through the transistors AXL and AXR. It has been discussed in the literature that variations in the process of fabrication of cells using CMOS technology would generate a random behavior during power-up stages. There is, though, a new trend looking forward to demonstrate whether this behavior would be completely random or if some cells would tend to start in a predefined state. [3].

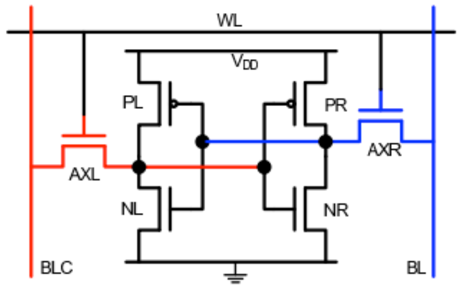

Fig. 1. Six transistor SRAM cell [3].

The startup scenario of digital memories was for a long time thought to be completely random. If the element is brought into an unstable state, it is not clear what will happen. But that does not mean that the startup state of a memory would be completely random. It could start varying its values or it could go back to a stable state. What we intend to show here is that some cells heavily prefer certain stable states. Moreover, this can often not be explained by the logic implementation of the cell, but it turns out that internal physical mismatch, e.g. caused by manufacturing variation, plays a role in this [2].

In this way, right after the memory power-up, it is hard to predict the state of the memory, i.e., the memory will present as a table filled with unpredictable data. If this data is uncorrelated between different chips and is the same all the time the SRAM is powered-up, the device may therefore be used as a PUF. In this case, the challenges would be the addresses of the memory and the responses would be the unpredictable content of the SRAM memory after the power-up.

III. SRAM PUF Characterization

In order to use SRAM memory cells as PUFs, one must confirm that the memories present the required properties. We chose to use the SRAM present on a 16-bit Texas Instruments MSP430F2013 microcontroller (MCU) instead of the ones present in a FPGA because, when using SoCs, the application designer would not have total control over the hardware functionality but therefore use standardized interfaces to access the memory. Besides that if we could use the SRAM on a MCU to implement a PUF we would have circuits implementing the PUF and their protocols in the same devices where their applications would be running. This MCU have a total of 128 Byte SRAM, sufficient to develop the framework that will be later used to characterize more complex devices.

To be used as a PUF, one must characterize the SRAM memory through repeated measurements of the same chip that we define as intra-class behavior. Hamming distance is the preferred metric to compare different circuits. The measurement of intra-class Hamming distance gives us an estimative of noise on the circuit and should be as small as possible. Such small errors could be later corrected by error correcting algorithms. [4]

It is important to also characterize the circuit behavior through measurements of different chips, defined here as inter-class behavior. The measurement of inter-class Hamming distance gives a clue on how predictable is the output of one PUF instance based on the information of another PUF instance. This problem will be tackled in a later work.

The framework proposed in this work is composed by a testbench and analysis software running on a PC connected to the MCU by a serial interface. This software runs the experiments as described in Section V and performs the analysis described in Section VI later on.

IV. ASYNCHRONOUS SERIAL COMMUNICATION

The first issue we had to deal with was the absence of a UART interface in the microcontroller. We needed to read information from the memory after the resetting the chip so for testing purposes it was implemented a simple asynchronous serial protocol to receive data from the microcontroller to C ANSI software written specially for this purpose. In Fig. 2 it is shown a timing diagram of the implementation of the protocol.

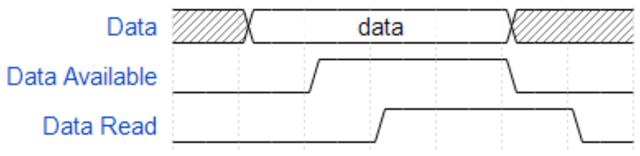

Fig. 2. Asynchronous Serial Protocol timing diagram

The idea here is to use three pins to implement the communication. The first one is the Data line which represents where the bit from the memory would be sent to. The Data Available pin is set after the data bit is written in the line, to inform the computer that there is data in the line to be read. The microcontroller enters then in a loop waiting for the Data Read pin to be set. Whenever this pin is set to high the MCU is supposed to interpret that the computer received the bit. Then, it drops the Data Available pin and enters in another loop waiting for the Data Read pin to be dropped from the computer. Whenever the MCU senses the Data Read pin has been dropped it finishes the transmission of the Data bit and the process starts all over again for the next bit to be sent. We use pins of a RS232 serial port on a PC to communicate with the microcontroller. The communication is done by using the pins CTS connected to P1.0, DSR to P1.1, RTS to P1.2, DTR to Vdd.

V. GATHERING MEMORY DUMPS

As mentioned before the purpose in studying PUF applications lies in the fact of defining a set that includes a number of challenge response pairs defined as CRP database. The idea is to generate a process that can be reproduced during usage to apply a challenge to the device and compare the result with a key or some sort of database and then evaluate whether the device is valid or not.

Using the protocol mentioned in the previous section we were capable of reading dumps automatically. We developed software that resets the microcontroller after waiting 2s in order to demonstrate if the number of resets performed in a single SRAM would reveal similar characteristics so that we could generate the CRP database.

The process starts by defining a macro MAX_REPETITIONS that represents how many times the microcontroller unit should be reset, in our case a hundred, and a macro called MEM_SIZE which in our case is 128 that represents the number of bytes to be read from the memory at once. The memory in the microcontroller being used starts at the address 0x200h and goes all the way to 0x27Fh.

The framework starts powering up the MSP board by setting the DTR pin to 1. Afterwards it enters in a loop waiting for the Data Available (Data-AV) pin to be set, which would mean that the board has fetched a byte from memory and intends to send the first bit through the serial line. Then, the data line is read and the Data Read (Data-Read) pin is set to 1, which would mean that the bit has been read from the line. Now the system waits for the Data-AV pin to be dropped from the MCU, meaning that the unit understood that the computer read the bit. At this point the software drops the Data-Read pin (RTS = 0) and increment BitCount. If BitCount equals to 7 it means that one byte has been fetched and the microcontroller can start reading another byte from memory. Otherwise it returns to the Wait stage to get the next bit. After repeating the process until all accessible memory has been read the framework decides if it is time to stop fetching data from the microcontroller or if another reset needs to be performed. This workflow can be seen graphically in Fig. 3.

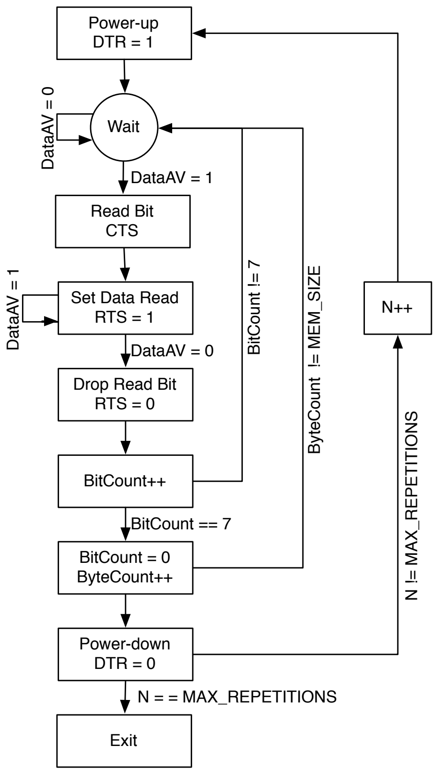

Fig. 3. Framework workflow

VI. Processing Memory Dumps

For the chosen MCU we gathered N memory dumps spaced by a time period of two seconds. This period would be sufficient to get rid of residual charge stored in the SRAM cells. The software then compares all possible pairs of memory dumps by creating exclusive-or (XOR) masks that contains zeros at the positions where the bits have not changed over two experiments and ones at the positions where bits have changed over different experiments. Counting the number of ones in each mask gives us the Hamming distance associated with each pair of measurements. By summing (OR) all the masks generated by the end of this process in a global mask, we could define an error rate as being the total number of bits that changed over the experiments at least once over the total number of bits available in the SRAM memory.

Table I shows an example of what the software does and the masks it produces by comparing three memory dumps. As the last two bits have changed over the measurements, the global mask will reflect this behavior by setting the last two bits and we would get a 25% error rate in this example.

TABLE I. Memory dumps comparison – Example for one byte

Dump number	***Memory[0x200h]***
1	0xFC – 1111 1100
2	0xFE – 1111 1110
3	0xFD – 1111 1101
MASK (1 – 2)	0x02 – 0000 0010 ($HD_{intra} = 1$)
MASK (1 – 3)	0x01 – 0000 0001 ($HD_{intra} = 1$)
MASK (2 – 3)	0x03 – 0000 0011 ($HD_{intra} = 2$)
Global MASK	0x03 – 0000 0011 ($HD_{intra} = 2$)

We also analyze the histogram of all intra-class Hamming distances calculated. This gives us a measurement on how much noise there is on the circuit (average) and how it is distributed over different experiments (standard deviation).

VII. Framework Evaluation

We present here a typical case we get by using the proposed framework. We performed a set of 100 memory dumps on a 3.3V power supply and room temperature. After evaluating the Hamming distances and plotting the histogram of them, we realize that the output mask reveals that only 59 bits out of 1024 (128B) have changed at least once during the process of resetting the circuit which results in 5.76% error rate in this typical case.

In fact as presented in Fig. 4 the error rate calculated by computing the logical sum of the individual masks represents the worst case scenario, because all bars in this histogram are placed under the value 0.03 which would mean 3% of variation in each comparison of two masks individually, which leads to the fact that the trade-off of increasing the computational effort to calculate all the possibilities has provided a better visualization of the results in the form of the histogram.

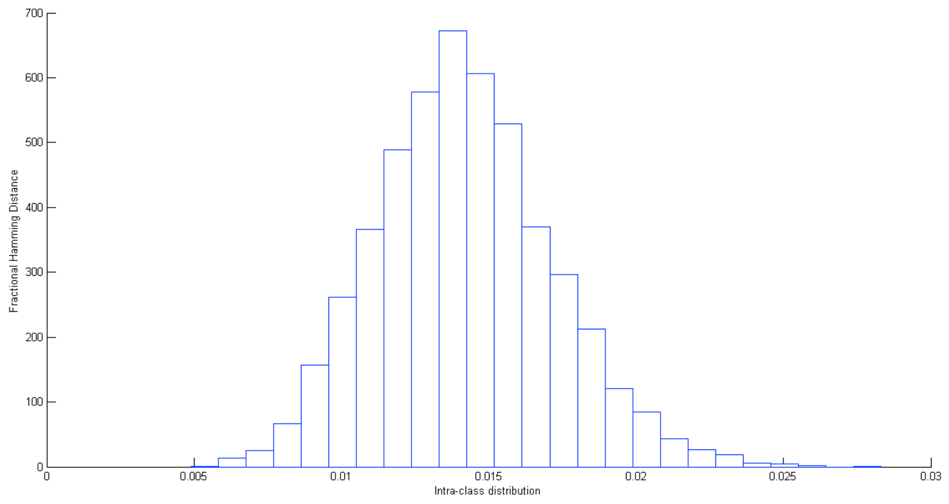

Fig. 4. Intra-chip Hamming distance histogram for Vdd = 3.3V and room temperature.

VIII. CONCLUSION

This work presented a framework to characterize SRAM memory PUFs. It allows one to gain knowledge on the intra-class behavior of SRAM circuits by getting several memory dumps of a chosen microcontroller and analyzing it under different temperatures and power supply voltages. We show a typical case using Texas Instrument MSP430F2013 microcontroller that presented a 5.76% error rate under typical operating conditions. Future works would include the inter-class characterization of the SRAM circuits and the proposal of error correction codes to deal with the noise presented by the circuits.

ACKNOWLEDGMENT

The authors would like to acknowledge Intel Labs University Research Office for financial support.

REFERENCES

[1] S. Katzenbeisser, U. Kocabaş, V. Rožić, A. Sadeghi, I. Verbauwhede, C. Wachsmann. PUFs: Myth, Fact or Busted? A Security Evaluation of Physically Unclonable Functions (PUFs) Cast in Silicon.

[2] R. Maes, I. Verbauwhede, "Physically Unclonable Functions: a Study on the State of the Art and Future Research Directions"

[3] Guajardo, J., Kumar, S.S., Schrijen, G. J., Tuyls, P.: FPGA Intrinsic PUFs and their use for IP Protection. In: Cryptographic Hardware and Embedded Systems Workshop, LNCS, vol. 4727, pp. 63 (80 (2007)),

[4] Christoph Bohm, Maximilian Hofer, Wolfgang Pribyl. A Microcontroller SRAM-PUF. Institute of Electronics. Graz University of Technology