




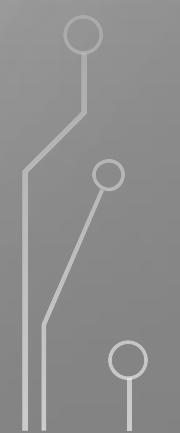
REED-MULLER CORRECTION CODES ON PUF RESPONSE

SECURE SYSTEM DESIGN PROJECT

A. DI MARTINO M63/654 – P. LIGUORI M63/556– S. BARONE M63/610

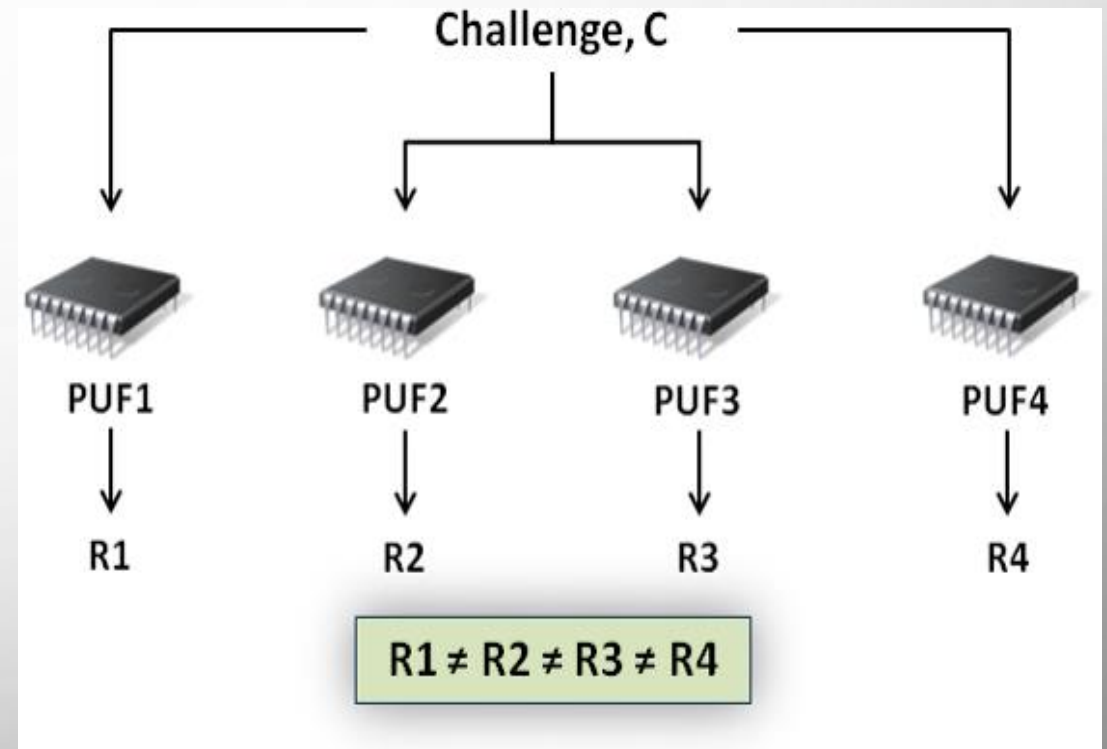


PUF - PHYSICAL UNCLONABLE FUNCTION

- A PUF is a physical entity that is embodied in a physical structure and is easy to evaluate but hard to predict.
 - Must be easy to make but practically impossible to duplicate, even given the exact manufacturing process that produced it.
 - depends on physical factors introduced during manufacture which are unpredictable
- 
- 

PUF: CHALLENGE RESPONSE AUTHENTICATION

- PUFs implement **challenge–response authentication** to evaluate this microstructure:
 - When a physical stimulus is applied to the structure, it reacts in an unpredictable (but repeatable) way due to the complex interaction of the stimulus with the physical microstructure of the device.
 - The applied stimulus is called the **challenge**, and the reaction of the PUF is called the **response**.

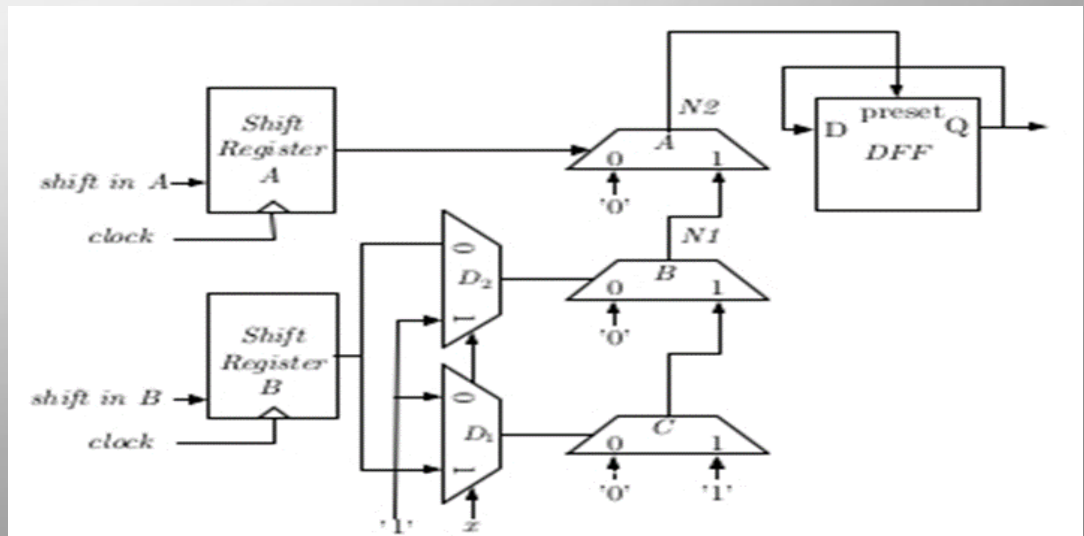
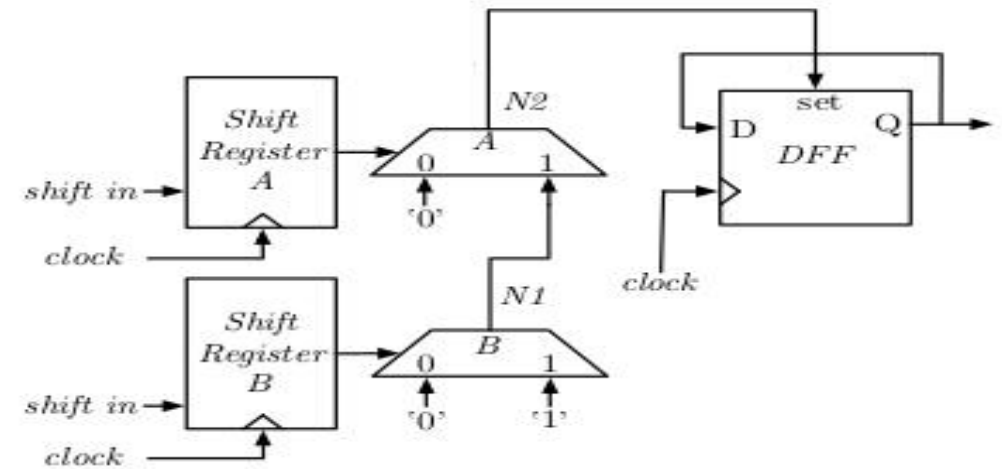


PUF-BASED AUTHENTICATION PROTOCOL

- It consists of two phases:
 1. **enrollment**: a third trusted party applies a significant number of randomly selected challenge to the PUF and stores the corresponding responses;
 2. **verification**: the trusted third party selects a challenge and gets the response from the PUF. If the answer matches the one previously stored, the device is authentic.
- It's vulnerable to man-in-the-middle attacks

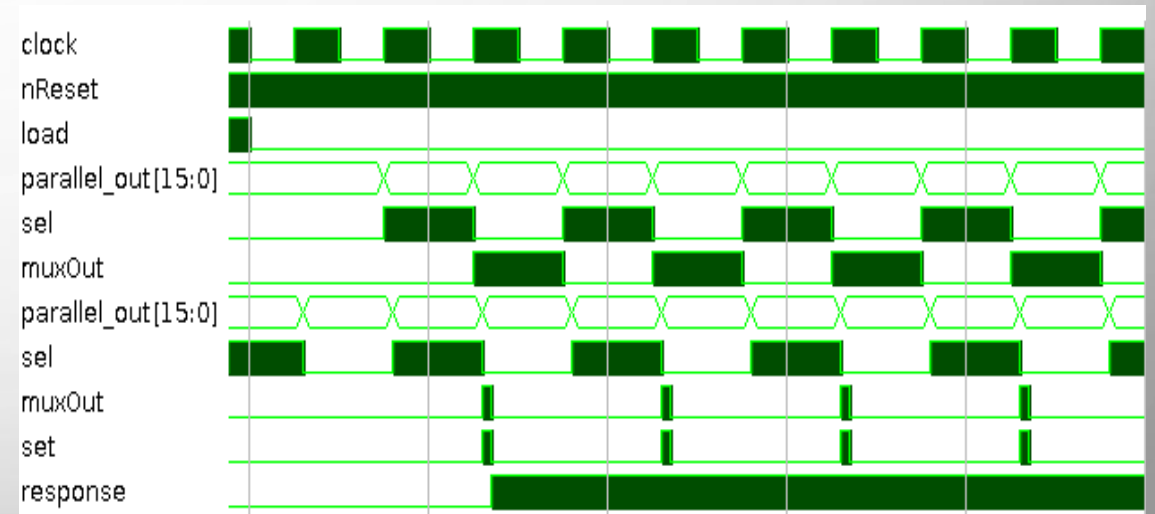
ANDERSON PUF

- Designed for the Xilinx Virtex V
- It bases its operation on the imperceptible differences in terms of delay, exhibited by its components when synthesized on FPGAs.
- It can be used to generate fingerprint (image above) or in challenge-response protocols (below image).



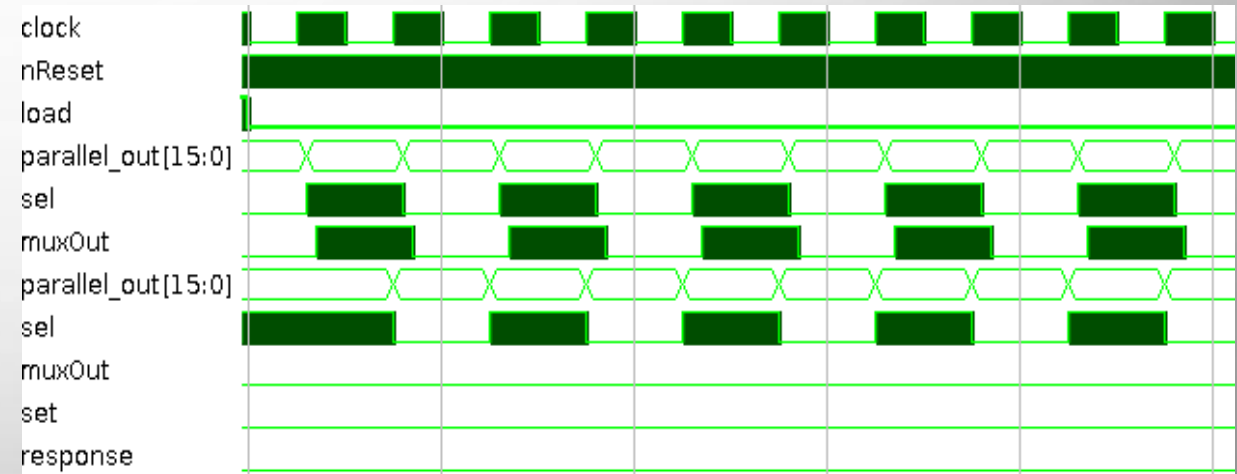
ANDERSON PUF (2)

- If shift-register A and multiplexer A are faster in terms of propagation delay than shift-registers B and multiplexers B, a spike "1" will appear on "preset" input of the flip-flop.
- If the duration of that spike is greater than the flip-flop setup time, then the output of flip-flop will assume '1' and it will keep it permanently.



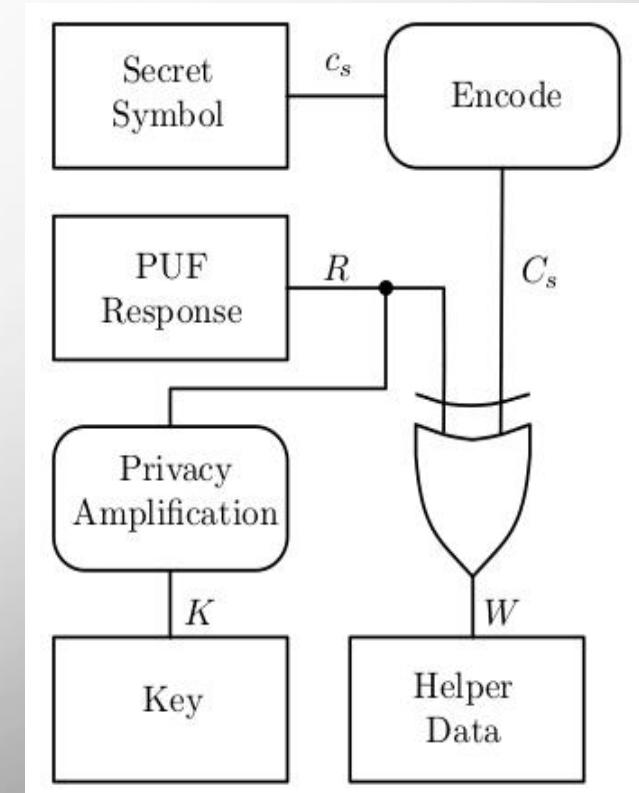
ANDERSON PUF (3)

- If shift-register B and multiplexer B are faster, in terms of propagation delay, than the shift-registers A and multiplexers A, the "preset" input of the flip-flop will always be '0', so the flip-flop will assume a constant '0' value.



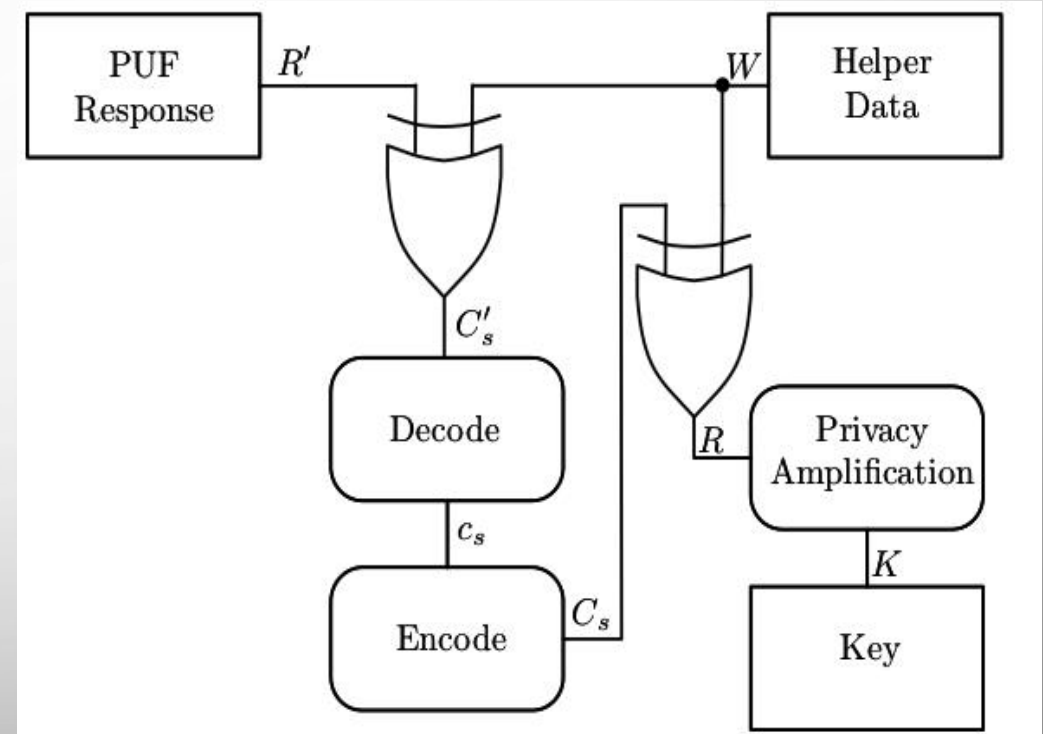
FUZZY EXTRACTOR: ENROLLMENT PHASE

- Puf is disturbed by the noise introduced by the sensitivity to variations in ambient conditions.
- We need a fuzzy-extractor.
- During the enrollment phase, a random number c_s is encoded with an error-correction technique, generating C_s . That number should be kept secret.
- PUF response R is placed in XOR with C_s to obtain helper-data W .
- The PUF response R is also used to generate the cryptographic key K (using the cryptographic-hash-function).



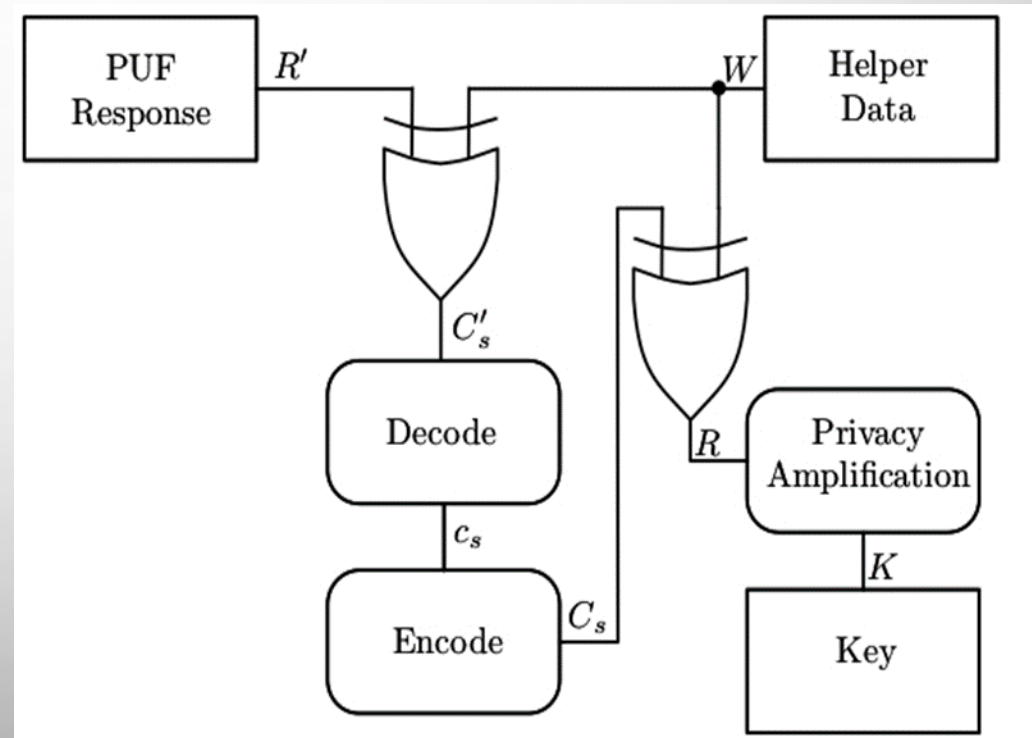
FUZZY EXTRACTOR: VERIFICATION PHASE

- After the device has been placed in its operating environment, only a noisy version of the PUF response R' , disturbed by the variation in environmental conditions, is available for use.
- The non-noisy version can be obtained from the noise using the helper-data W : by placing W in XOR with R' , we will get a noisy version of C_s , C'_s .



FUZZY EXTRACTOR: VERIFICATION PHASE (2)

- Dec decrypting C_s by using a **error correction code**, we will get the original c_s with no noise.
- Therefore, by coding c_s , we get the non-noisy version C_s .
- C_s is placed in XOR with W to obtain the non-noisy version of the PUF response, that is used to generate the encryption key K .



REED MULLER CODE: DEFINITION

- The error correction code used to get the original cs with no noise is the **REED-MULLER code**.
- For each positive integer m and each integer r with $0 \leq r \leq m$, there is an r -th order Reed-Muller Code $R(r,m)$.
- We consider the 1st order case ($r = 1$), because it maximizes the Hamming distance between the codes.
- Definition: The (first order) Reed-Muller codes $R(1,m)$ are binary codes defined for all integers $m \geq 1$, recursively by:
 - (i) $R(1,1) = \{00,01,10,11\}$
 - (ii) for $m > 1$, $R(1,m) = \{(u, u), (u, u+1) : u \in R(1, m-1) \text{ and } 1 = \text{all 1 vector}\}$

REED-MULLER CODE: DEFINITION (2)

- Thus:
- $R(1,2) = \{ 0000, 0101, 1010, 1111, 0011, 0110, 1001, 1100 \}$
- $R(1,3) = \{ 00000000, 00001111, 01010101, 01011010, 10101010, 10100101, 11111111, 11110000, 00110011, 00111100, 01100110, 01101001, 10011001, 10010110, 11001100, 11000011 \}$

GENERATOR MATRIX

- A generator matrix of $R(1,1)$ is:

1	1
0	1

- If G_m is a generator matrix for $R(1,m)$, then a generator matrix for $R(1,m+1)$ is:

G_m	G_m
$0 \dots 0$	$1 \dots 1$

REED DECODING

- One reason that Reed-Muller codes are useful is that there is a simple decoding algorithm for them. We illustrate the method known as **Reed Decoding** with an example.
- Consider the code $R(1,3)$ with generator matrix:

v0=	1	1	1	1	1	1	1	1
v1=	0	1	0	1	0	1	0	1
v2=	0	0	1	1	0	0	1	1
v3=	0	0	0	0	1	1	1	1

REED DECODING (2)

- The rows of the previous matrix are basis vectors for the code:
 - label them v_0 , v_1 , v_2 and v_3 .
- Any vector v of the code is a linear combination of these:
 - i.e., $v = a_0 v_0 + a_1 v_1 + a_2 v_2 + a_3 v_3$.
- Written as a vector, we have:
 - $v = (a_0, a_0 + a_1, a_0 + a_2, a_0 + a_1 + a_2, a_0 + a_3, a_0 + a_1 + a_3, a_0 + a_2 + a_3, a_0 + a_1 + a_2 + a_3)$.

REED DECODING (3)

- If no errors occur, a received vector $r = (y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7)$ can be used to solve for the a_i other than a_0 in several ways (4 ways for each) namely:

- $a_1 = y_0 \oplus y_1 = y_2 \oplus y_3 = y_4 \oplus y_5 = y_6 \oplus y_7$

- $a_2 = y_0 \oplus y_2 = y_1 \oplus y_3 = y_4 \oplus y_6 = y_5 \oplus y_7$

- $a_3 = y_0 \oplus y_4 = y_1 \oplus y_5 = y_2 \oplus y_6 = y_3 \oplus y_7$

(Symbol \oplus is a bitwise XOR)

- If one error has occurred in r , then when all the calculations above are made, 3 of the 4 values will agree for each a_i , so the correct value will be obtained by majority decoding.
- Finally, a_0 can be determined as the majority of the components of:

$$r \oplus a_1 \cdot v_1 \oplus a_2 \cdot v_2 \oplus a_3 \cdot v_3$$

REED DECODING: EXAMPLE

- Suppose that $v = 10100101$ is received as $1010\color{red}{1}101$.
- Using the previous formulas, we calculate:
 - $\alpha_1 = 1 = 1 = 0 = 1$ so $\alpha_1 = 1$
 - $\alpha_2 = 0 = 0 = 1 = 0$ so $\alpha_2 = 0$
 - $\alpha_3 = 0 = 1 = 1 = 1$ so $\alpha_3 = 1$

REED DECODING: EXAMPLE (2)

- Let's calculate a_0 :
- $a_0 = \text{received vector} \oplus a_1 v_1 \oplus a_2 v_2 \oplus a_3 v_3$

r =	1	0	1	0	1	1	0	1
$a_1 \cdot v_1 =$	0	1	0	1	0	1	0	1
$a_2 \cdot v_2 =$	0	0	0	0	0	0	0	0
$a_3 \cdot v_3 =$	0	0	0	0	1	1	1	1

- By performing a bitwise xor along the columns, we get a_0 :

$a_0 =$	1	1	1	1	0	1	1	1
---------------------------	---	---	---	---	---	---	---	---

- Applying a majority function, we calculate $a_0 = 1$

REED DECODING: EXAMPLE (3)

- $v = a_0 \cdot v_0 \oplus a_1 \cdot v_1 \oplus a_2 \cdot v_2 \oplus a_3 \cdot v_3$

$a_0 \cdot v_0 =$	1	1	1	1	1	1	1	1
$a_1 \cdot v_1 =$	0	1	0	1	0	1	0	1
$a_2 \cdot v_2 =$	0	0	0	0	0	0	0	0
$a_3 \cdot v_3 =$	0	0	0	0	1	1	1	1

- By performing a xor bitwise along the columns, we get v:

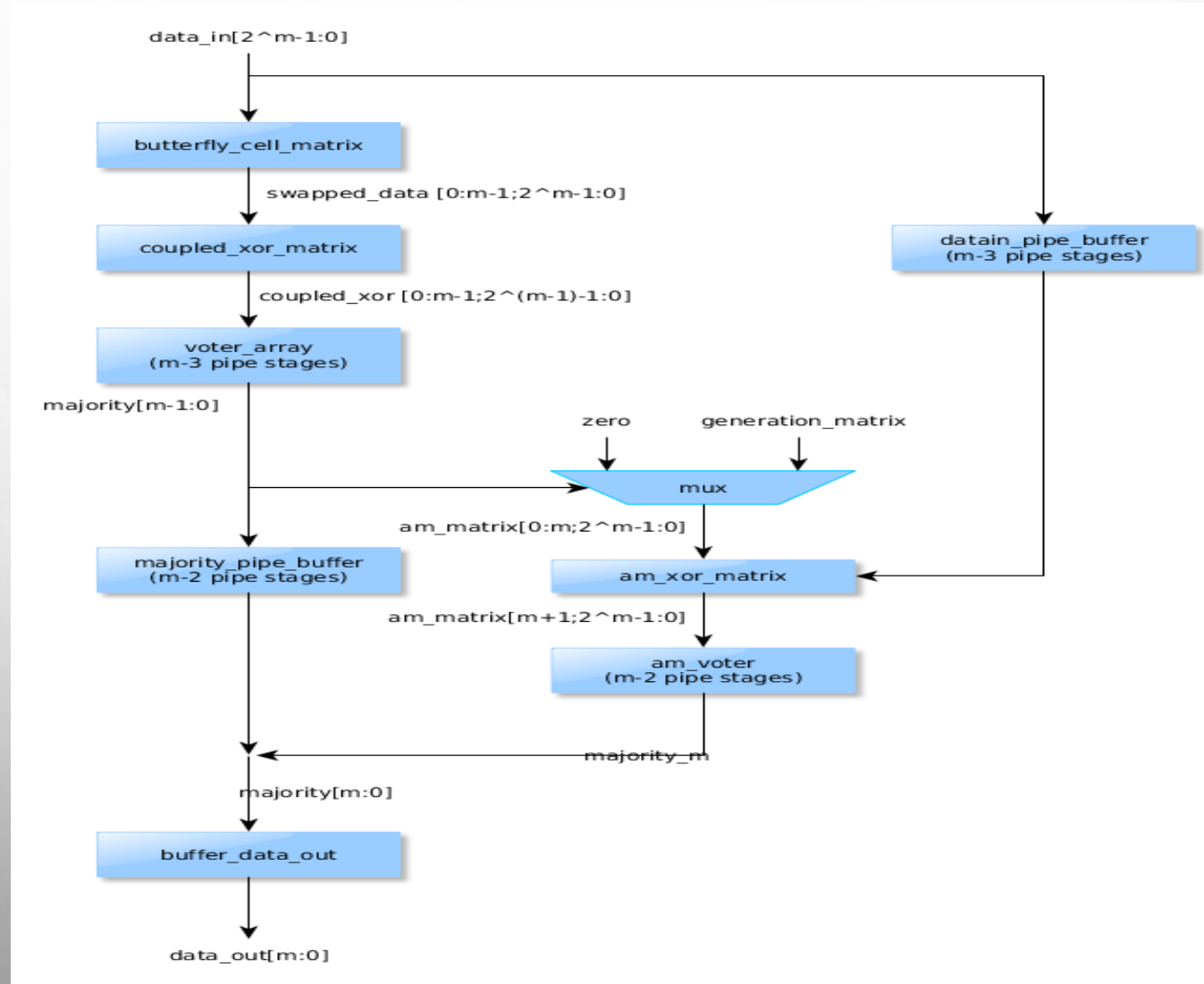
$v =$	1	0	1	0	0	1	0	1
-------	---	---	---	---	---	---	---	---

- So, we have corrected the wrong bit: 10100101

REED-MULLER DECODER IN VHDL

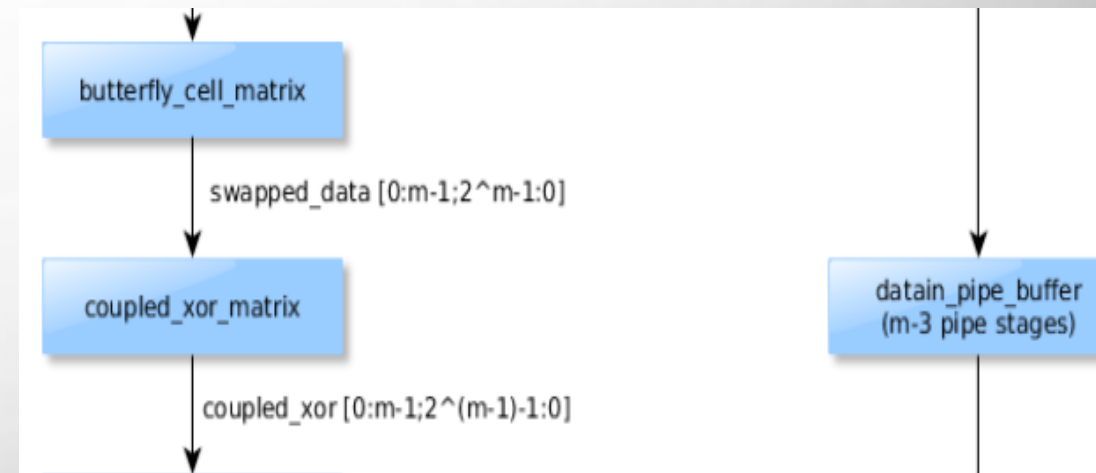
- The component is generic.
 - Only order r was fixed, equal to 1, so maximizing Hamming distance between the codes:
 - Mariner probes (1963-1973) fly-by of Venus, Mercury, Mars used RM (1.5) for encoding images sent to Earth.
 - The value of m , representing the length of the block ($N=2^m$), can be changed at each use.
- The intention was to make the component as fast and as small as possible.
- The implementation is predominantly combinatory with pipelining (using buffers).
- The circuit can also be used in other areas, not specifically for this project:
 - e.g. remote control and receiving data from a video camera mounted on a toy tank.

REED-MULLER DECODER IN VHDL (2)



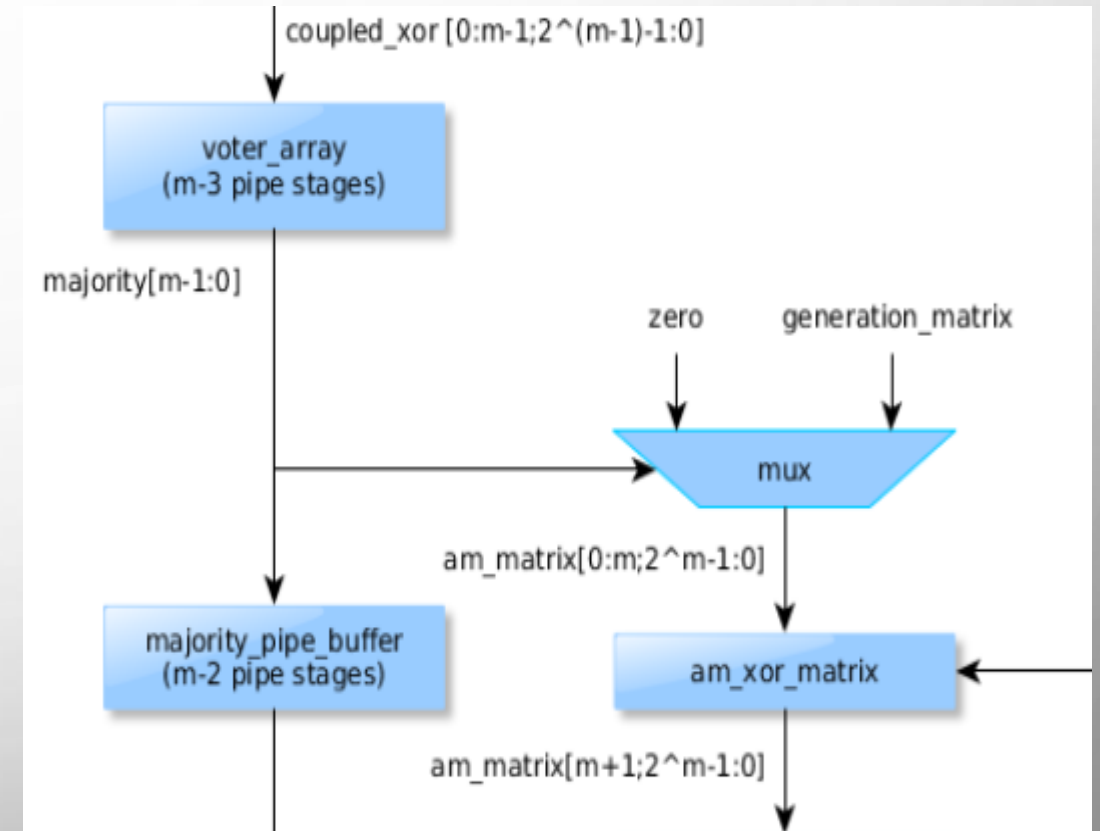
REED-MULLER DECODER IN VHDL (3)

- The decoder uses the majority-voter method.
- In input there is a signal of 2^m bits. It passes through the "butterfly_cell" block, which creates the swapped copy appropriately, in a pattern similar to that used for Benes networks.
- Each swapped copy forms the swapped_data matrix, consisting of m lines and 2^m columns.
- Each row passes through the "coupled_xor" block that performs the xor of adjacent columns, paired, producing the "coupled_xor" array, consisting of m rows and 2^{m-1} columns.



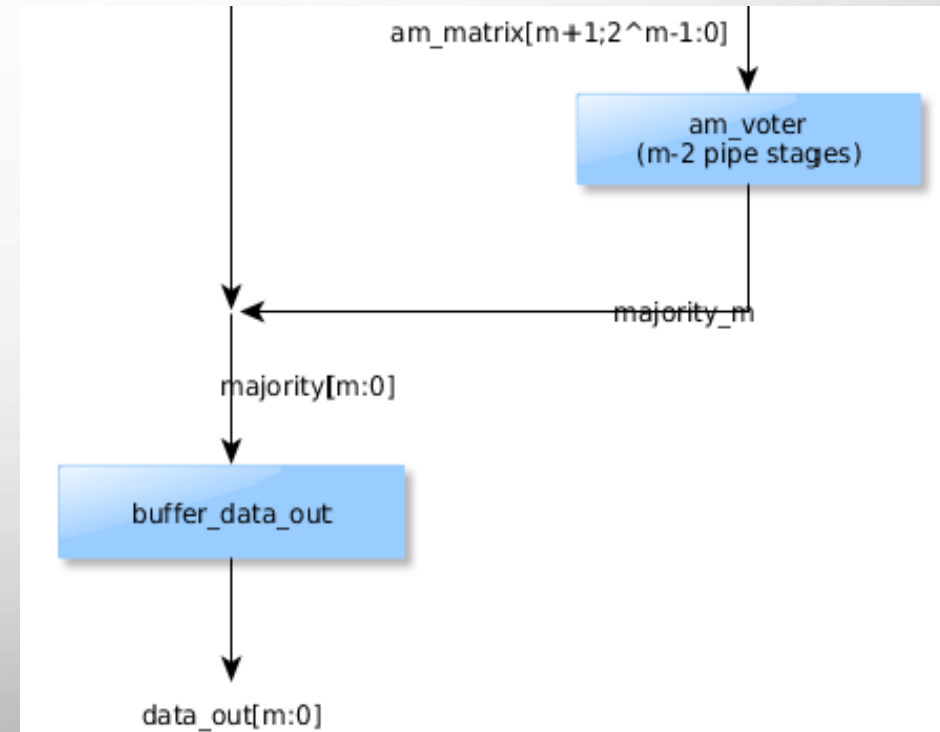
REED-MULLER DECODER IN VHDL (4)

- Each row of the array constitutes the input of one of the $m-1$ majority-voters used to compute the least significant bits of the decoded code.
- These bits will be used for calculating the most significant bit: each of them determines one of the rows of the `am_matrix` array:
 - The first line is the signal "data_in"
 - The other lines are:
 - G (m-i) if majority (i) = '1';
 - zero otherwise.

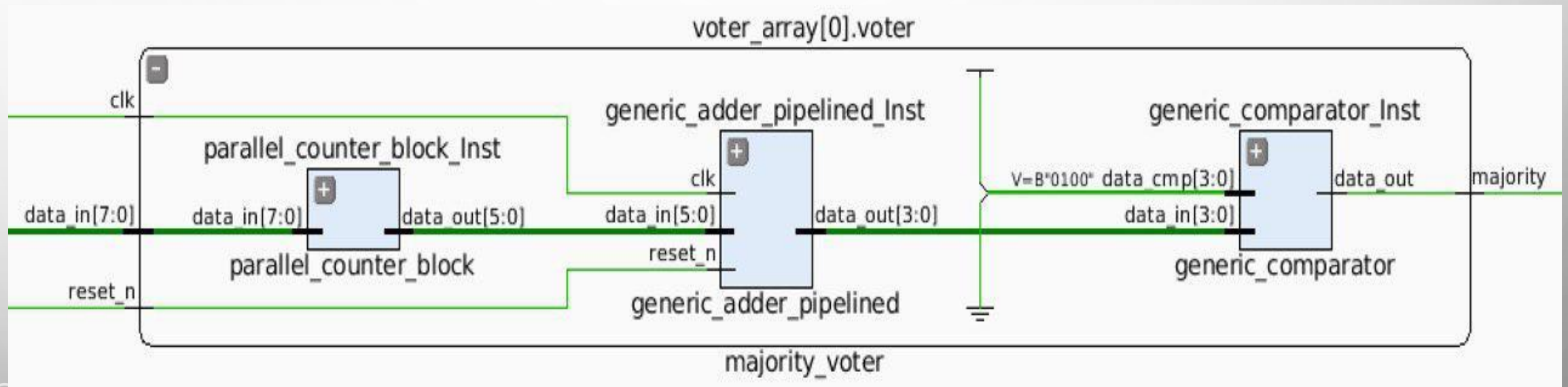


REED-MULLER DECODER IN VHDL (5)

- The lines of the `am_matrix` matrix are placed in xor between them to determine the input of the last majority-voter whose output is the most significant bit of the decoded code.



REED-MULLER DECODER IN VHDL: VOTER



REED-MULLER DECODER IN VHDL: VOTER (2)

- The implemented voter evaluates the majority of bits equal to 1 in the input string. The value of majority is 1 in case the number of bits equal to 1 is greater than $\text{width}/2$.
- The component is structured in three modules in sequence:
- Module 1: "parallel_counter_block":
 - In input there is a string of $n \geq 4$ bit, with n power of 2. The string is partitioned into 4-bit blocks.
 - The module consists of $\text{width}/4$ counters working in parallel, where each parallel counter encodes the bit number = 1 contained in the assigned nibble.

REED-MULLER DECODER IN VHDL: VOTER (3)

- Module 2: generic_adder_pipelined “
 - It inputs a bit string that is the concatenation of addendums. It outputs the total sum of the addendums.
 - Each adder takes the two addendums from the output string of the previous layer and the sum is delivered to the next level.
- Module 3: "generic_comparator“
 - Inputs:
 - A. the bit string to compare - data_compare_in (in our case is the total sum of bit = 1 of the initial string);
 - B. the bit string with which I want to compare - data_compare_cmp (in our case is the width/2 binary encoding)
 - It returns output "1" if $\text{data_compare_in} > \text{data_compare_cmp}$, "0" otherwise.

REED-MULLER PARAMETERS

ECC	M (ordine)	Lunghezza word (bit)	Lunghezza codeword (bit)	Distanza di Hamming (bit)	Errori/codeword (bit)	Decoder pipe stages (2m-4)
RM(1,5)	5	6	32	16	7	6
RM(1,6)	6	7	64	32	15	8
RM(1,7)	7	8	128	64	31	10
RM(1,8)	8	9	256	128	63	12

TIMING AND AREA

ECC	Component	Slice LUTs (17600)	Slice Luts %	Slice Registers (35200)	Slice Registers %	Slice (4400)	Slice %	LUT as Logic (17600)	LUT as Logic %	LUT as Memory (6000)	LUT as Memory %	LUT Flip Flop Pairs (17600)	LUT Flip Flop Pairs %
RM(1,5)	RMDecode r	221	1,26	175	0,50	69	1,57	216	1,23	5	0,08	108	0,61
	RMEncode r	16	0,09	0	0,00	5	0,11	16	0,09	0	0,00	0	0,00
RM(1,6)	RMDecode r	514	2,92	472	1,34	157	3,57	508	2,89	6	0,10	262	1,49
	RMEncode r	62	0,35	0	0,00	23	0,52	62	0,35	0	0,00	0	0,00
RM(1,7)	RMDecode r	1351	7,68	923	2,62	388	8,82	1216	6,91	135	2,25	715	4,06
	RMEncode r	127	0,72	0	0,00	48	1,09	127	0,72	0	0,00	0	0,00
RM(1,8)	RMDecode r	3094	17,58	2045	5,81	886	20,14	2830	16,08	264	4,40	1575	8,95
	RMEncode r	257	1,46	0	0,00	141	3,20	257	1,46	0	0,00	0	0,00

REFERENCES

1. Jiliang Zhang, Yaping Lin, Yongqiang Lyu, and Gang Qu - *A PUF-FSM Binding Scheme for FPGA IP Protection and Pay-Per-Device Licensing*
2. Mario Barbareschi, Pierpaolo Bagnasco, Antonino Mazzeo - *Authenticating IoT Devices With Physically Unclonable Functions Models*
3. Mario Barbareschi, Pierpaolo Bagnasco, Antonino Mazzeo - *Supply Voltage Variation Impact on Anderson PUF Quality*
4. Mario Barbareschi, Pierpaolo Bagnasco, Domenico Amelino and Antonino Mazzeo - *Designing an SRAM PUF-based Secret Extractor for Resource-Constrained Devices*
5. *Design and improvements of Anderson PUF for Xilinx Spartan-3 FPGA*
6. Massoud Malek - *Coding Theory*
7. <http://www-math.ucdenver.edu/~wcherowi/courses/m7823/reedmuller.pdf>
8. https://en.wikipedia.org/wiki/Physical_unclonable_function