Final report on PUF project

Simone Asnaghi
Davide Tacconi

05/02/2022

Politecnico di Milano - Embedded System

Accademic year: 2021/2022

Sommario

1. Source code 4

1.1. single_bit_PUF 4

1.2. Arbiter 4

1.3. Starter 5

1.4. Enmapper 5

1.5. Demapper 6

1.6. Controller 6

1.7. IBS 7

1.8. Arbiter_PUF 7

2. Simulation 8

2.1. 1_bit_APUF_tb 8

2.2. Enmapper 8

2.3. Demapper 8

2.4. Controller 9

2.5. IBS 9

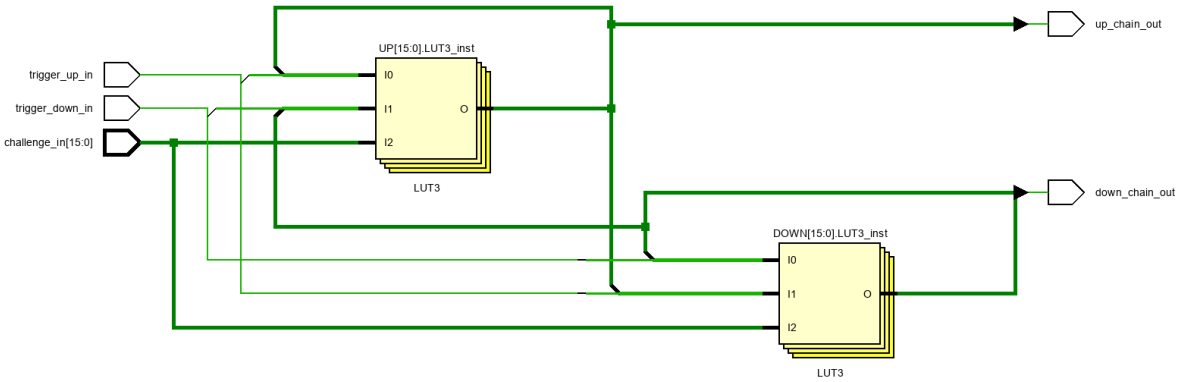
2.6. Arbiter_PUF 9

3. Conclusions 10

1. Source code

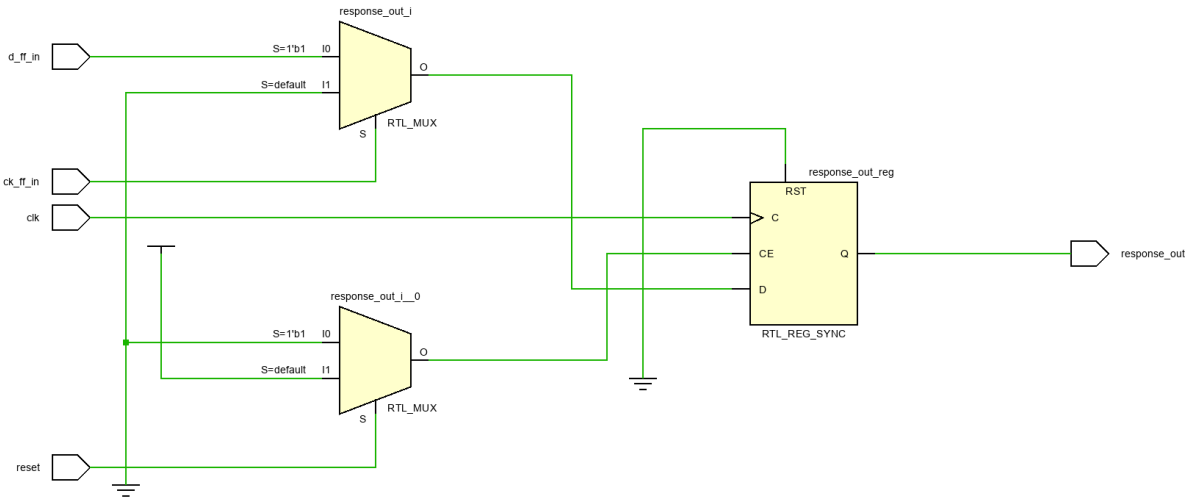
1.1. single_bit_PUF

This is the base module of our system. It includes the muxes series to create both upper chain and lower chain. Two outputs are generated, used as inputs of the arbiter.



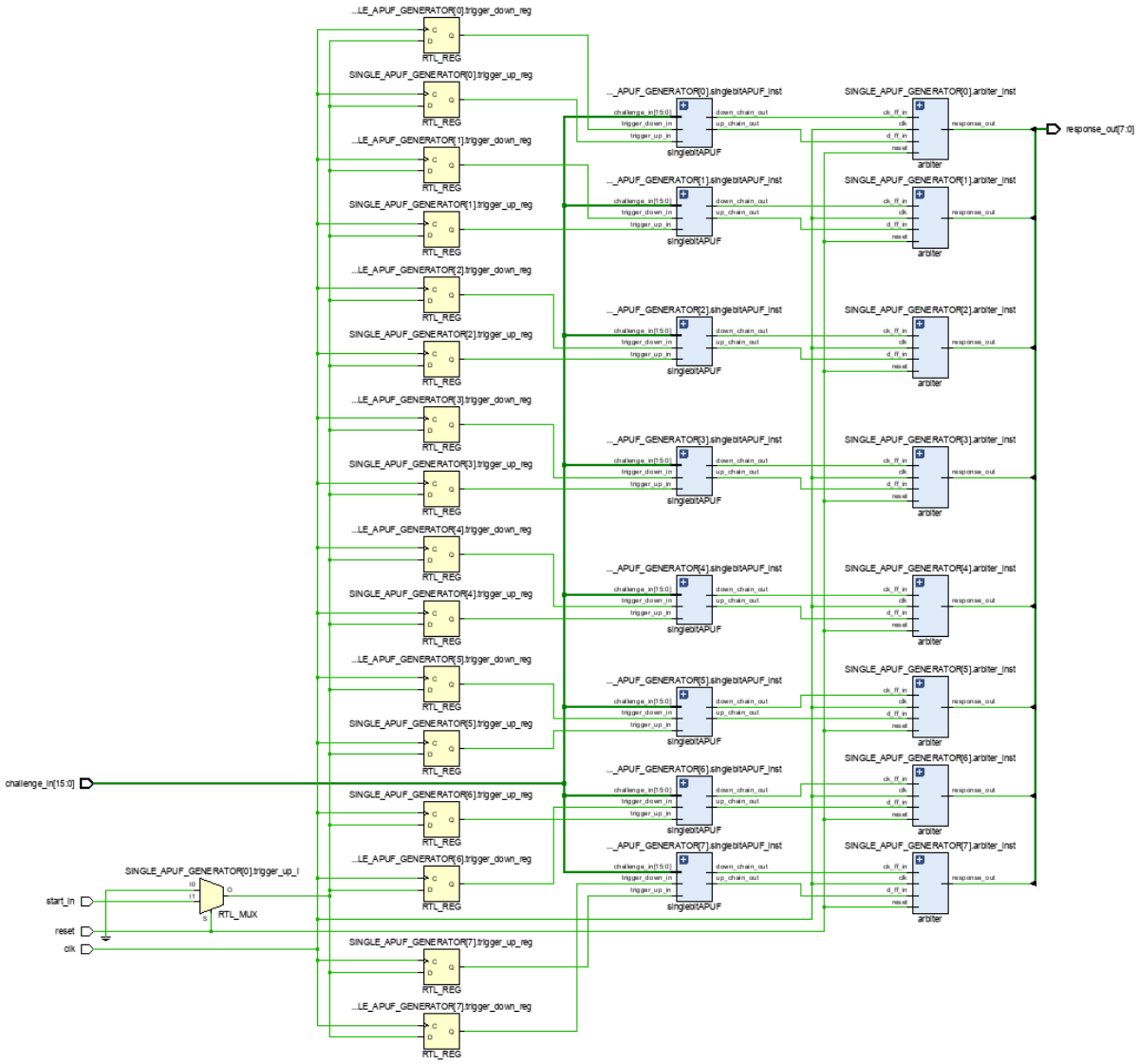
1.2. Arbiter

This module is the final arbiter of the chain. It's implemented using a flip-flop with the upper chain signal as input data and the lower chain signal as clk that drives the flip-flop.



1.3. Starter

This module is the controller of the APUF. It's used to assert triggers when a new challenge is given and that relaunches the procedure if we want more response using the same challenge.



1.4. Enmapper

This module receives response stack and a vector B. It searches the maximum or minimum of each response row, according to the value of B for the row selected.
It generates an output vector in which it stores the pointers to the maximum or minimum position of each row.
Schematic is not insert in this section because it's too large.
In this module the B vector is generated internally. We had to decide whether generate a random vector B using the randomize function, but as known a true random number generator doesn't exist, or create a fake logic circuit that generates B. We choose the

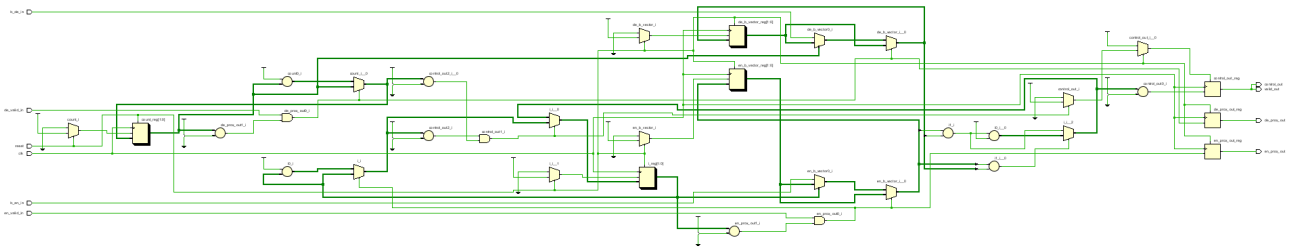
latter, using some logic ports to generate values to fill positions in B vector. In this way, the vector is generated quite randomly, since different conditions of working of the system could create different values and therefore a quite random vector B. Another possible solution could be received B as input but this solution could cause saturation of I/O ports in a low level application board, like the basys3 (Artix 7 FPGA chip) used by us.

1.5. Demapper

This module receives response stack and the pointer vector generated by the enmapper. Its role is to generate a vector B' by determining if response selected by pointer are actual maximum or minimum of each row.
Schematic is not insert in this section because it's too large.
In this module we don't need to generate any B vector and all the inputs are given.
Interesting in this module, is the way to search the maximum and the minimum using an FSM system.

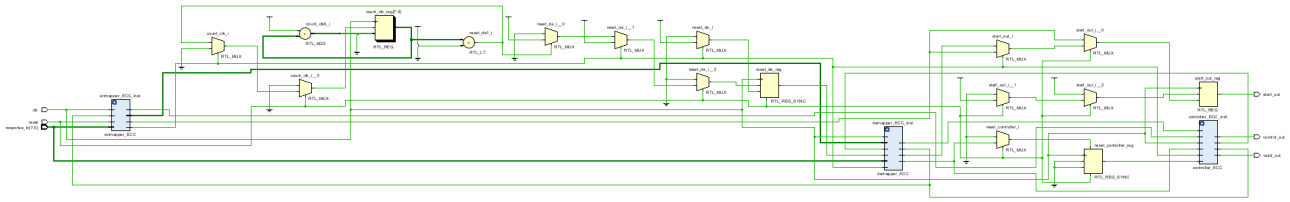
1.6. Controller

This module is the controller of the IBS system. It confronts B and B' to determine if the response is correct or must be discarded.
It also provides a enable pin to deactivate the IBS module and therefore copies directly the response given as input directly to its output.
In this module we don't generate any B vector inside because is given as input, taken from the Enmapper module.



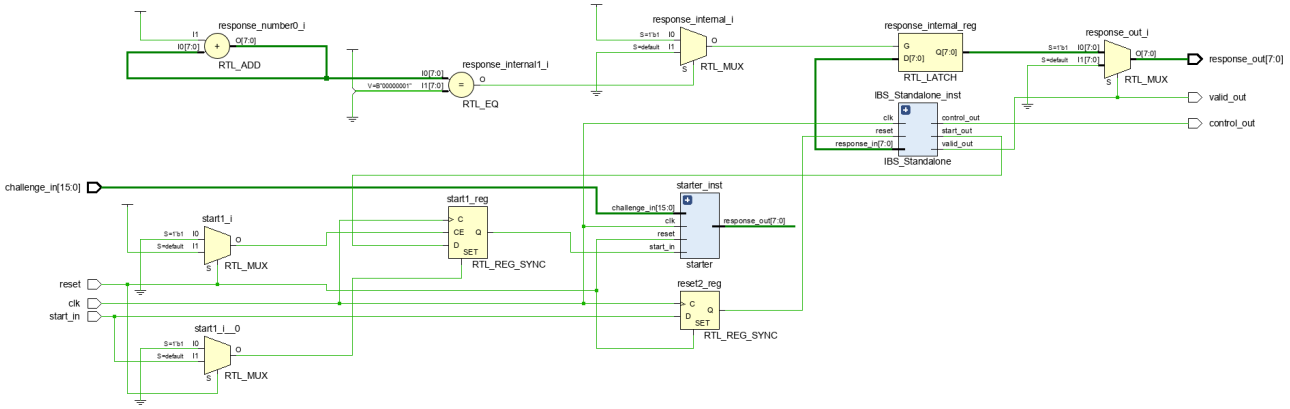
1.7. IBS

This module contains the instantiation of enmapper, demapper and of the controller, with the role to create the subsystem of the IBS standalone. The role of this subsystem is to determine whether response is correct or not.



1.8. Arbiter_PUF

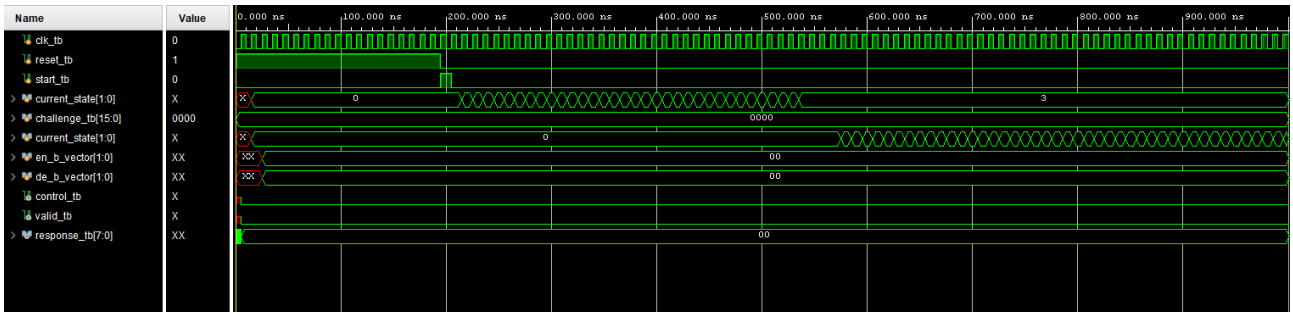
This module is the bigger one. It contains both IBS and APUF subsystem.



2. Simulation

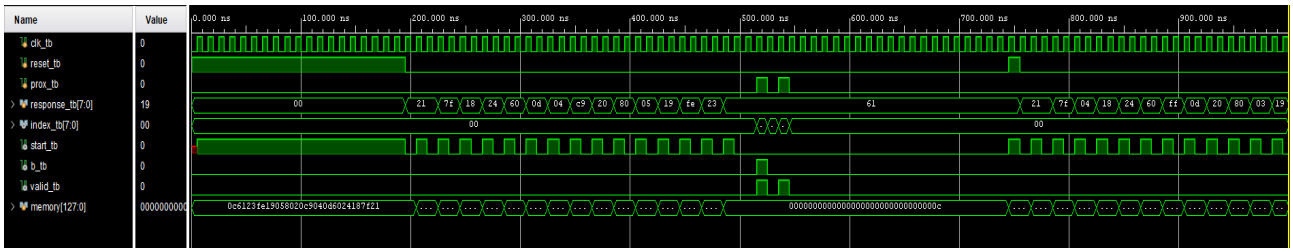
2.1. 1_bit_APUF_tb

This is the main simulation of the Arbiter PUF subsystem. Inside this block under test there are: starter, single_bit_puf and the arbiter.
As the simulation inserted next to this paragraph shows, in our implementation, the delay introduced by the final arbiter cause the response to be fixed to '0'. This is an unexpected behaviour.



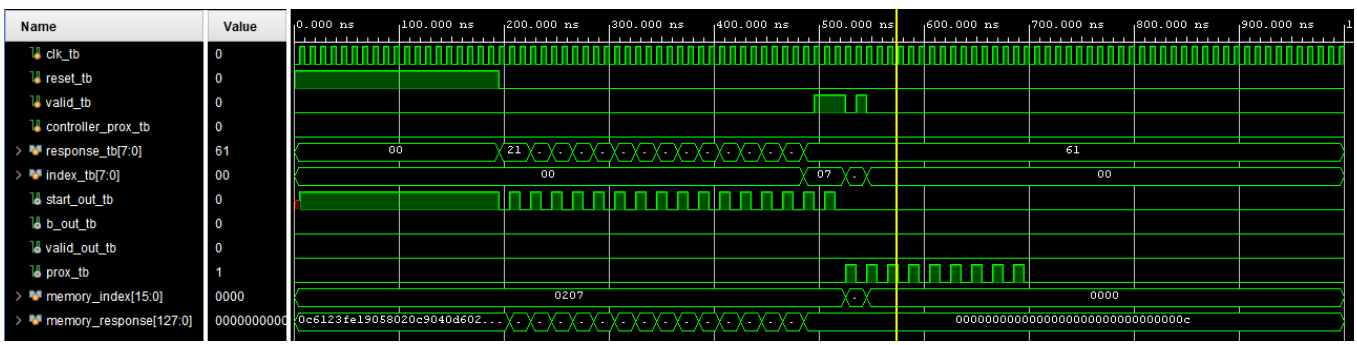
2.2. Enmapper

This simulation shows the behaviour of the enmapper module alone, tested individually to understand whether it works or not.
Simulation shows that the module is correct and works as expected.



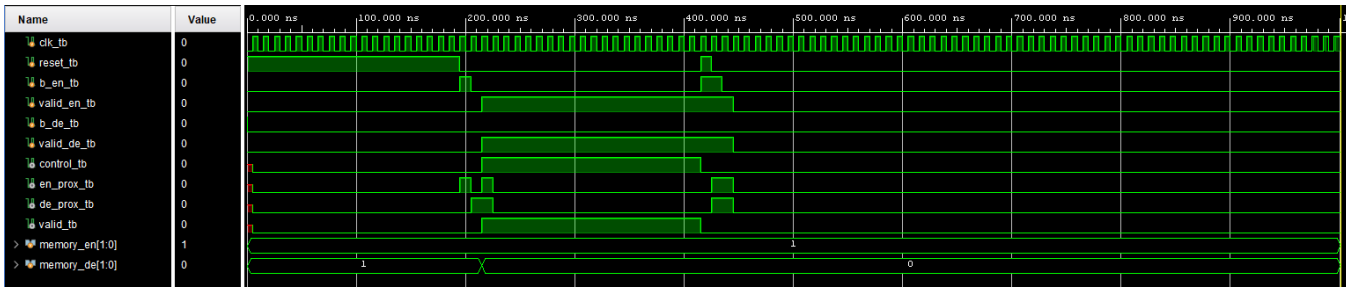
2.3. Demapper

This simulation shows the behaviour of the demapper module alone. As for the enmapper this simulation is used to determine the correctness of the module.
Also in this case, the simulation shows that the module has the behaviour we expected.



2.4. Controller

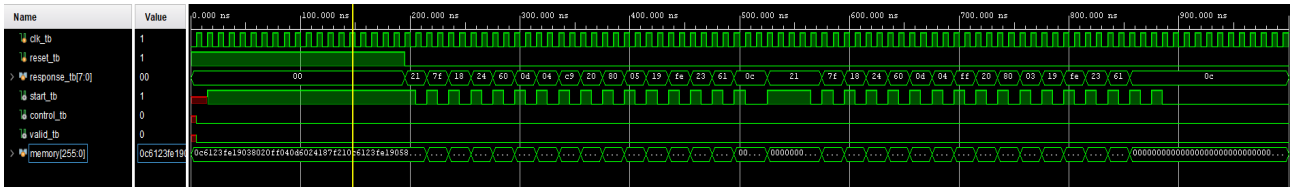
This simulation has the role to test the correct behaviour of the controller module alone. Without any extra component module must prove that its able to confront B and B', and after this check to give as output the correct response.



As this simulation waveform shows the module works as expected.

2.5. IBS

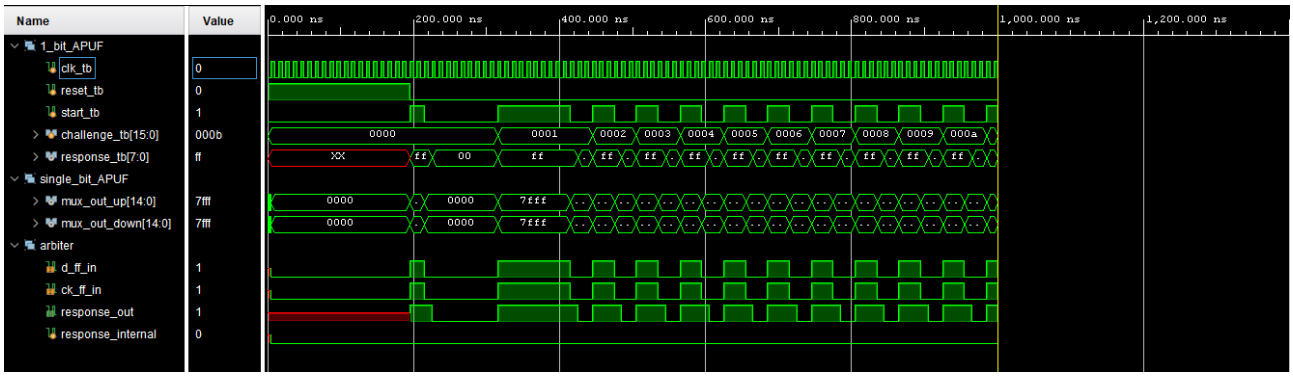
Simulation is used to determine if the union of enmapper, demapper and controller works properly together.
This means that the timing of signals used to share information between components must be precisely tuned.



Even if the inserted image is a bit small, it's possible to see that the module works correctly and, therefore, is possible to implement everything into the main module that compose the whole system.

2.6. Arbiter_PUF

This is the simulation of the whole PUF system, without the IBS module.
Simulation shows that system works in the correct way.



3. Conclusions

As written in the part of the source code the module of the single bit APUF shows at the output of the post implementational timing simulation always a fixed value, that sometimes is ‘0’, but other times is ‘1’.
This behaviour is unexpected, but we haven’t found any reason. To be sure that this is a problem we should test it on an actual device and see if the simulator is only showing this behaviour for software reasons.
There’s the possibility that in the PUF module timing’s constraints are not respected:
if delay and setup timing are not respected could happen that even if clk is faster, so output should be ‘1’, it’s ‘0’ in simulation’s result.

PS: Source code for the project is given with this report inside a zip file. Extract and double click on .xpr file to open the project.