SCIREA Journal of Electrical Engineering

http://www.scirea.org/journal/DEE

January 20, 2022

Volume 7, Issue 1, February 2022

https://doi.org/10.54647/dee47263

A Performance Comparison Between Different Approaches for Implementation of FPGA-based Arbiter Physical Unclonable Function

Swati K. Kulkarni [1,*], Vani R. M [2], P. V. Hunagund[1]

[1]Applied Electronics Department, Gulbarga University, Gulbarga-585106, India

[2] University Science Instrumentation Center, Gulbarga University, Gulbarga-585106, India

Email: swatikulkarni494@gmail.com (Swati K. Kulkarni), vanirm12@rediffmail.com (Vani R. M), prabhakar_hungund@yahoo.co.in (P. V. Hunagund)

Abstract:

In our daily lives, embedded and Internet of Things (IoT) applications are becoming extremely important. It advised using intellectual property (IP) while developing embedded and IoT applications. Hardware security and electronic component counterfeiting pose a threat to IoT devices and IPs. Another consideration is that today's electronic industry must contend with a slew of security concerns, including IC cloning, reverse engineering, overbuilding, and physical manipulation. PUF is a part of hardware security, utilized for device authentication, IP core protection, and the creation of cryptographic keys. The PUF uses intrinsic properties of IC fabrication variances to deliver an exclusive identifier for every device. PUF inherently provides security properties. Already, many researchers have concluded that arbiter and SRAM PUF architectures are unsuitable for FPGAs because their delay skew is higher than the random manufacturing process variations. The designer should have an accurate understanding of the FPGA platform. We used the Xilinx FPGA to develop and compare three versions of Arbiters PUF: 64-bit Classical Arbiter PUF, 64-bit,128-bit PDL-based Arbiter PUF, and Wide multiplexer-based Arbiter PUF. A comparison was performed based on the design approach, resource consumption, power analysis, and PUF feature.

Keywords: Arbiter PUF, Programable Delay Line, Register Transfer Level (RTL), FPGA-SOC, Simulation, Synthesis, Placement & Routing and Hardware Validation etc.

1. Introduction

1.1 Motivation

Integrated Circuits are the fundamental electronic component for any kind of computing system. ICs are used in every computing application like commercial, industrial, or military, and many more sectors. Recently, several incidents get reported of the Pirated and Duplicate production of ICs. The Pirated and Duplicate production of ICs are damaging not only the electronic industries but the users too. The IC fabrication infrastructure is not available everywhere because it requires complex and costly infrastructure. Generally, IC fabrication is not taking place under any surveillance. This factor encourages an adversary to include functionalities that aren't part of the device's declared specifications. [17]. These added functions may get trigger at a specific event. A hardware trojan leaks secret information to the attacker also may disable the system remotely. This malicious modification is a hardware Trojan. Such inclusion of hardware has been found in the security devices also.

The IC fabrication or any embedded application heavily relies on the reuse of intellectual properties (IPs). IPs are more popular because they are pre-designed, pre-tested, and pre-verified macro. IPs are technology-independent, reusable, and saves design time. The IPs get converted into GDS-II for the ASIC fabrication. There are two kinds of IPs are available. If IPs embedded into the chip are called Hard IP. Whereas Soft IPs are the RTL block utilized in the integrated development environment (IDE) system design. At the beginning of the new system design, the designers verify third-party IPs are available that can directly integrate into the design IPs are typically outsourced and applied in the design. Because Ips are more popular, there is a chance that an adversary may inject hardware malware into it. The electronic industry faces threats from the hardware trojan. [12] [16].

Some systems have onboard security facilities features. A trusted execution environment is offered in high-end ARM microprocessors. This feature is popularly known as ARM Trust Zone. Many IoT devices and embedded systems that employ ARM cortex M series or other low-end microprocessors are unable to implement the trust zone functionality due to the high cost. These devices are the main target of the attackers can extract secret information. Cryptography is a popular and recommended method for data security. The cryptography system is standing on five pillars. Those are Confidentiality, Integrity, Authentication, Authorization, and Non-repudiation. All the pillars are equally contributing in data security. ‘The secret key’ will handle the entire cryptography system. Mathematically cryptographic algorithms are robust unless and until the 'secret key' remains 'secret' for the attackers [13].

The EEPROM, battery-backed SRAM, or EEPROM or Flash Memories are used to store the ‘Cryptographic key’. The problems associated with these memories are

1. They are very expensive and battery-backed SRAM may have limited battery life.

2. Performing the Side-Channel Attacks, an attacker can extract the information. There are different ways of performing side-channel attacks like timing attacks, Power Attacks, Fault Attacks, Design for Testability attacks, Cache Attacks, etc. An adversary tries to extract information from the chips regarding the ‘Cryptographic key’. We try to find a specific solution that tackles all of these concerns linked to hardware security after researching all of them [18].

1.2 Physically Unclonable Function

In the above-mentioned scenarios, the use of PUF technology provides an affordable security solution. IC fabrication is a complex process that comprises many stages like ion implantation, lithography, chemical vapor deposition, and many more. These stages are responsible for generating random inherent process variation inside the chip. These variations differ from chip to chip. Due to the process variation, two similar ICs may have difference in threshold voltage, Capacitance, other parameters like channel length or doping concentration, etc. These process variations are not measurable and controllable as they are in nanoscale. A PUF utilizes these inherent process variations to produce a unique identifier for each hardware device. In short, we can say that two identical PUF devices will generate different outputs for the same input. PUF is a low-cost, unpredictable, and reliable technique used as cryptography primitive for hardware security. PUF is a lightweight hardware security technique. In PUF devices secret key is not kept inside the memory or chip. Every time PUF device generates a new response or key. The PUF based device does not require the fixed key. The PUF generates a unique and random key every time by using the internal process variations [4]. The PUF input is known as a set of challenges. The PUF output is known as the response bit.

1.3 Review of past work

The PUF has categories based on different criteria, as shown in figure 1.

Criteria 1. The number of Challenge-Response Pairs determines whether a PUF is strong or weak (CRP). In strong PUF, CRP is greater. There are fewer or no CRPs in weak PUF.

Criteria 2. The two types of PUF utilized on a chip are non-silicon PUF and silicon PUF. Silicon PUFs are implemented on an ASIC or FPGA device.

Delay-based, Glitch-based, Memory-based PUF, Voltage-based, Current-based are the types of silicon PUF. The first time Pappu et al. have proposed the term “PUF” [1]. The Authors proposed an optical PUF in that the response obtained when shining a laser on a bubble-filled transparent epoxy wafer. The investigation on silicon-based PUF started in 2002. Gassend et al. [2] introduced the first Silicon-based PUF. The authors reported the fabrication process of the IC produces the internal process variation. Most of the research happens on the Delay-based PUF. The most popular Delay based PUFs are “Arbiter PUF” [7] and “Ring oscillator PUF (RO-PUF)” [3].

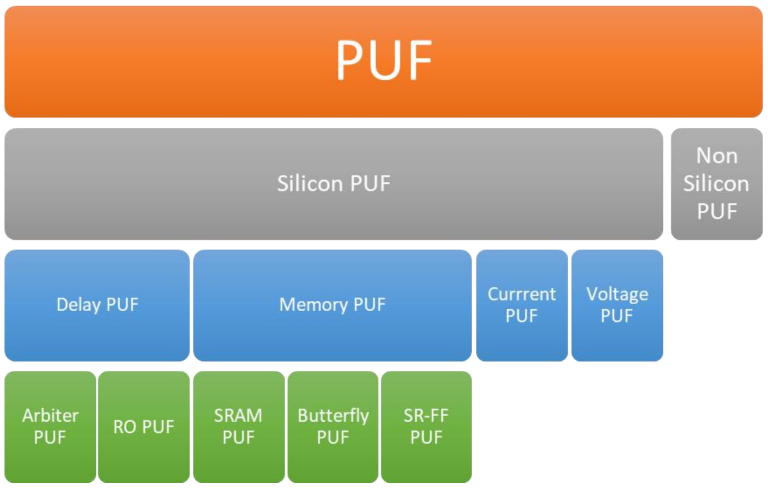


Figure 1 Classifications of PUF

The first "arbiter PUF" was introduced by Lee et al. [7]. The original arbiter PUF architecture consisted of two extended parallel chains of 2:1 multiplexer, each ending in a latch. [3][4] [6] investigated the Arbiter PUF in more depth afterward. The "Arbiter PUF" and the "Ring Oscillator PUF" are the two most prominent delay-based PUFs. The frequency of the Ring oscillator determines the output of "RO PUF". Aside from these two, the arbiter PUF is considered a "strong PUF". RO PUF is a "weak PUF". Suh and Devadas proposed The RO PUF [3] same ROs implemented on two similar devices that will generate different frequencies. Next, Maiti et al. [5] suggested the "configurable RO PUF". Suzuki et al. proposed another type of delay-based PUF called "Glitch PUF" (G-PUF) [9]. The authors claimed that “Anderson PUFs or Glitch PUFs” is most suitable for the FPGA implementation. Holcomb et al. [10] presented the first memory-based PUF, the "SRAM PUF". Because "SRAM PUF" is not appropriate for FPGA implementation (B-PUF), Kumar et al. proposed the " Butterfly PUF" [8]. Hata and Ichikawa were the first to introduce the "SR-L-PUF" [11]. Now also researchers are more inclined towards Hardware security and the PUF investigation.

2. Background

2.1 Traditional Arbiter PUF

The Traditional APUF comprises two parallel paths of multiple switching elements that are nothing but 2:1 Muxes. The paths are terminated by connecting them to the Din and Cin input ports of the D Flip-flop. Internally path swapping switches are inserted for dividing the lines into several subways [20]. The classical APUF has the internal swapped path. Switch input will select the switching elements. These parallel paths are triggered simultaneously by applying low to high signals. The select line of muxes acts as a secrete input (challenge) of PUF. The response will be for the particular challenge by applying the rising enable simultaneously to the two delay paths. The output will be zero if the din is slow compare to the clock input [23].

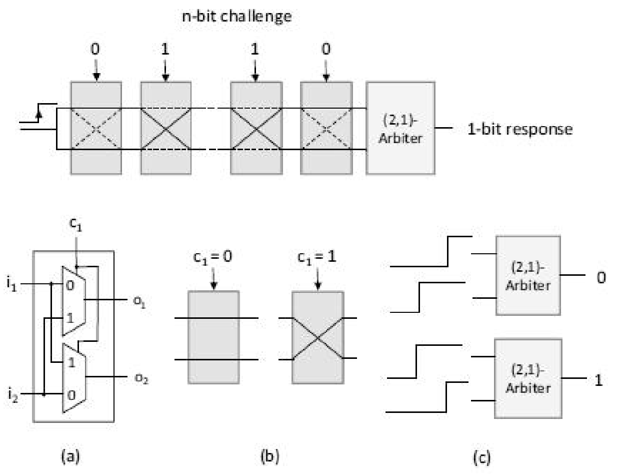


Figure 2 Classical Arbiter PUF [7]

Ideally, both the routes should be at an equal amount of delay as they are identical in length. At the same time, the internal process variations cause the delay difference. Naturally, there will be a race that happens on both paths. Arbiter will compare the delay between both routes. The Arbiter converts that analog time to a digital value. Based on the Delay factor, the entire APUF structure is partitioned into four regions

1. The part is before the first switching element.
2. Inside the switching path,
3. between the switching elements,
4. before the Arbiter.

All researchers have made it clear that while implementing APUF design on FPGA, each routing must be carefully handled and have symmetrical paths [14][15][16][20][29] [38]. It tends to delay imbalance and the biasing of the delay if the routing is not having symmetry. Delay bias affects the quality of output. FPGAs are pre-fabricated boards in that the logic arrays and routing layout are predefined. The designer has to take care of the architectural bias on the PUF design because the delay-based PUF implementation on FPGA becomes challenging. The delay biasing induced during FPGA implementation is called implementation delay or architecture biasing delay. Already, two methods were proposed to reduce this bias. A programmable delay line is the first way [28], and a double Arbiter PUF is the second [19]. The classical APUF has a swapped path between the switching elements The non-swapped routes between the switching elements in the Programmable delay line APUF [21]. On the other hand, double APUF is a simple structure like a classic APUF but exploits two separate APUFs to eliminate implementation bias effects. Also, double APUF requires more resources. In this article we implemented Classical arbiter PUF, PDL-based PUF and Wide-MUX based PUF. Wide-MUX based PUF also provides routing symmetry.

2.2 Programmable Delay Line

Programmable delay lines (PDLs) modify the signal delay or propagation in a controlled manner. PDL has different ways to control the signal delay. Internal delays are managed in analog design by changing the effective load capacitance, current, or threshold voltage. The digital circuit recommended that the propagation of the signal is controlled by incrementally changing the length of the signal propagation path; for the Arbiter, PUF design on ASIC or FPGA will follow the same method.

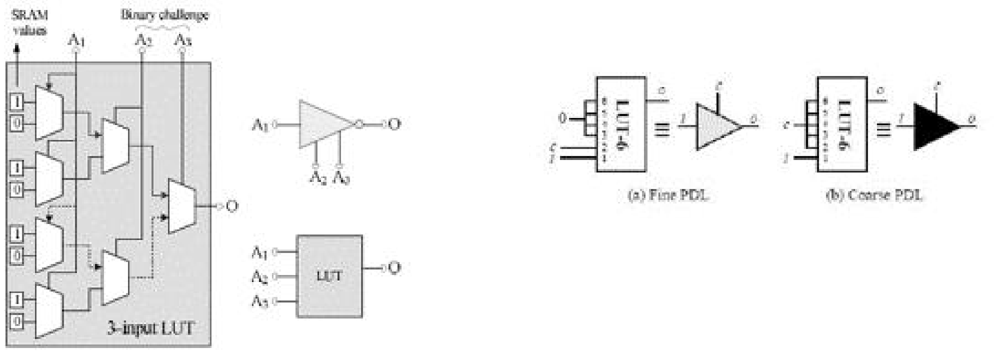


Figure 3. Internal schematic of Programable Delay Line [28]

The idea of PDL PUF is proposed by [28]. Many researchers had implemented PDL methods on the Xilinx older version of FPGA [22] [26] [33]. LUT are configured as per the user logic used to implement the user functions. The LUTs are used to implement the switching element on the FPGA. Figure 3 shows the internal architecture of LUT. Consider an example of NOT gate implementation where A1 input of the NOT gate and the remaining two inputs A2 and A3 are not used; we may consider those bits as don't care bits. These two bits control the delay of the actual function. Suppose A2A3 = 00 the signal propagates through the solid path that is a minimum delay path, whereas if A2A3 = 11, the signal propagates through the route of the dashed line marked that are maximum delay path. Following are the recommendations for APUF implementation on the FPGA [26][33].

1. Instantiate the device-specific primitives in the RTL
2. Connect the unused LUT pins to logic 0.

The Xilinx ISE FPGA Editor, XDL Utility, and Plan Ahead tool was proposed by [30] to implement APUF design on FPGA. The 'Hard macros' were designed by the designers to keep the routing symmetry. The 'Hard macros' in the APUF are created with a specific attribute (**LOCK PINS = "all"**). The FPGA editor has a function called Hard macro. It aids the designer in determining component location and routing [30].

2.3 Hardware- Software Co-design

FPGAs are the most recommended hardware platform from the researchers [31][32][34][35]. FPGAs are more flexible as they can be reprogrammable and reconfigurable. Also, they require less time to market, support complex design architecture. We used the Xilinx all programable SoC board or Zedboard based on the 28 nm technology. The ZedBoard has hard cored ARM processor that is the processing system (PS) and Artix 7 FPGA fabric (PL). The

Xilinx all seven series FPGA has a unified architecture. The designer can easily migrate the board. Zedboard FPGA has 53,000 Configurable Logic blocks. Each CLB consist of 2 Slices, each slice has four; 6 input LUTs, wide multiplexers, carry chain, four Data Flipflops and four flipflop or latches. This rich fabric helps the designers to implement their logic [39].

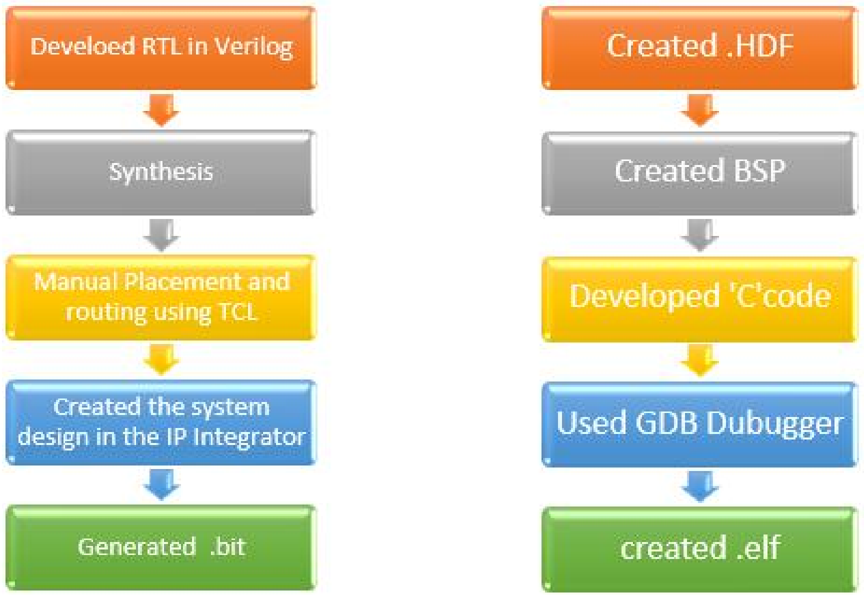


Figure 4. The flow of Hardware software co-design

Hardware-software co-design is the new approach for systematically building the design.

We divided the design into two parts shown in figure 4;

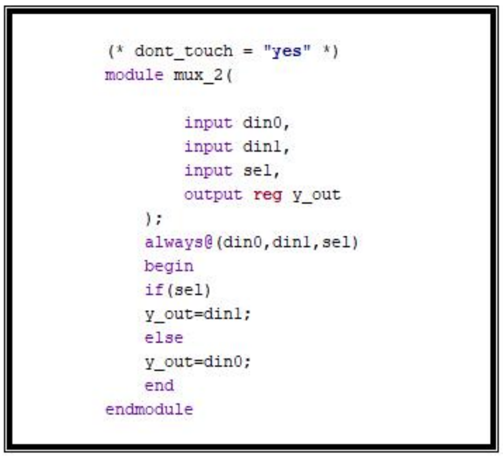
1. In this first part, we developed RTL, used IP from the Xilinx IP catalogue, did the placements, developed a system on the IP integrator, and generated a .bit file in the Xilinx Vivado tool.
2. In this second part, we created a board support package packet (BSP), System debugging, and an Executable Linkable Format (.elf) file for the program PS.

Finally, the '. bit' and the '. elf' files have been utilized to configure the FPGA [40].

3. Implementation of Arbiter PUF

3.1 Classical Arbiter PUF

In section 2.2, we discussed the construction and working of Classical Arbiter PUF (APUF). While designing the classical APUF, we developed the RTL with 2:1 MUX and D flip-flop as per figure 2. In this design, we have not used any FPGA primitives. The D-Flip-flop and the Verilog code 2:1 MUX (shown in Figure 5) was used to build the design [28].



```
(* dont_touch = "yes" *)
module mux_2(

        input din0,
        input din1,
        input sel,
        output reg y_out
    );
    always@(din0,din1,sel)
    begin
    if(sel)
    y_out=din1;
    else
    y_out=din0;
    end
endmodule
```

Figure 5. Verilog code for 2:1 MUX Classical Arbiter PUF

Figure 6 is the Synthesis schematic of the Classical APUF. It is a gate level implantation of the RTL.

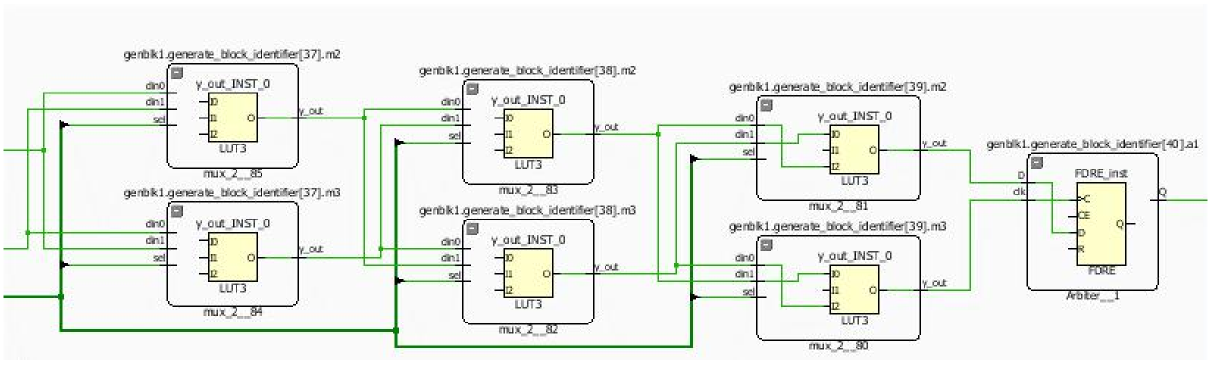

genblk1.generate_block_identifier[37].m2
genblk1.generate_block_identifier[37].m3
genblk1.generate_block_identifier[38].m2
genblk1.generate_block_identifier[38].m3
genblk1.generate_block_identifier[39].m2
genblk1.generate_block_identifier[39].m3
genblk1.generate_block_identifier[40].a1
y_out_INST_0
LUT3
mux_2__85
mux_2__84
mux_2__83
mux_2__82
mux_2__81
mux_2__80
FDRE_inst
FDRE
Arbiter__1

Figure 6. Synthesis Schematic of Classical Arbiter PUF

Figure 7 is a block diagram of the Classical APUF system. Block diagram consists of IP of the classical APUF, Zynq Processor, AXI_GPIO, and other required IPS.AXI_GPIOs are used to give challenges and other inputs to the PUF and collect the responses from PUF. Each GPIO has 2 channels and each one of them can handle 32 bits. For 64 bits we used (32+32) the AXI_GPIOS and 128 bits we used (32*4) the AXI_GPIOS APUF implementation.

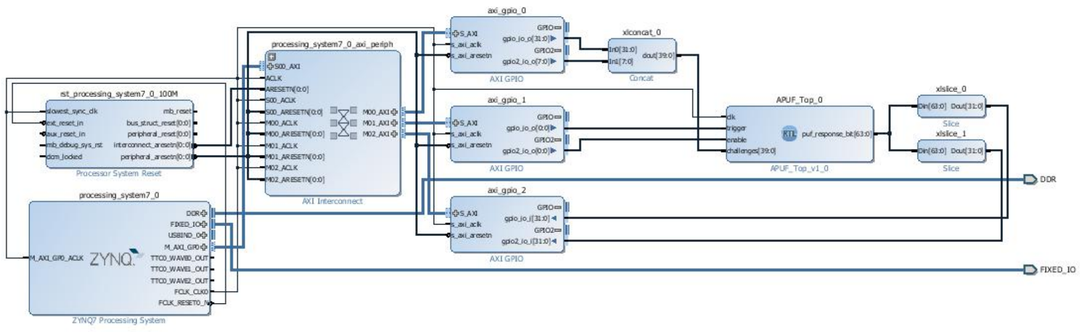
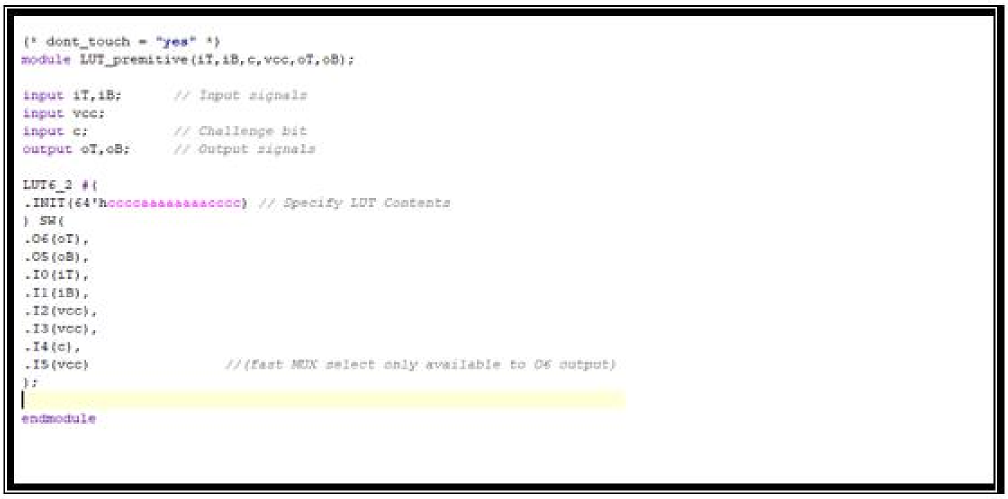


Figure 7 Classical Arbiter PUF System

3.2 PDL based Arbiter

The unused pins of LUTs are attached to logic '1' or '0' for the PDL APUF, as described in the previous section. The Xilinx 7 series FPGAs all include six input LUTs, with O6 and O5 output pins that are independent [27]. In 7 series FPGAs, the LUTs can be set as a 6-input LUT with one output or two 5-input LUTs with separate outputs for shared addresses or logic inputs. The RTL for the LUT primitive is shown in Figure 8. The chain of switching elements is made up of many RTL instantiations [25].



```
(* dont_touch = "yes" *)
module LUT_premitive(iT,iB,c,vcc,oT,oB);

input iT,iB;        // Input signals
input vcc;
input c;            // Challenge bit
output oT,oB;       // Output signals

LUT6_2 #(
.INIT(64'hccccaaaaaaaacccc) // Specify LUT Contents
) SW(
.O6(oT),
.O5(oB),
.I0(iT),
.I1(iB),
.I2(vcc),
.I3(vcc),
.I4(c),
.I5(vcc)                //(fast MUX select only available to O6 output)
);

endmodule
```

Figure 8. RTL to instantiate LUT primitive

The Figure 9 is a synthesis schematic of LUT primitives used to construct PDL-based APUF. Here I3 and I4 pin of LUT is tied to VCC. O5 and O6 output pins will construct the two parallel paths of APUF.

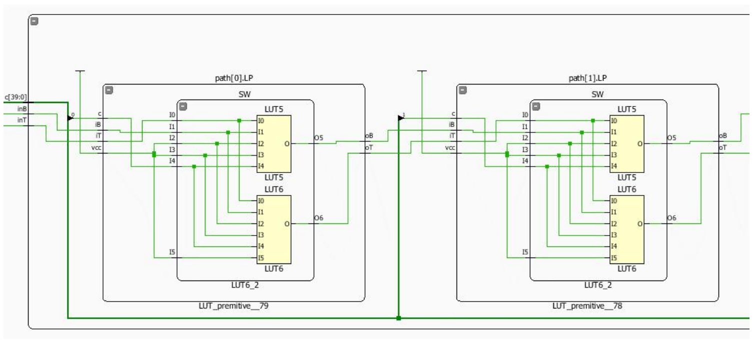


Figure 9. Synthesis Schematic of PDL based Arbiter PUF

The 128-bit PDL-based APUF RTL is converted into the IP module. Figure 10 shows the block schematic of the overall design.

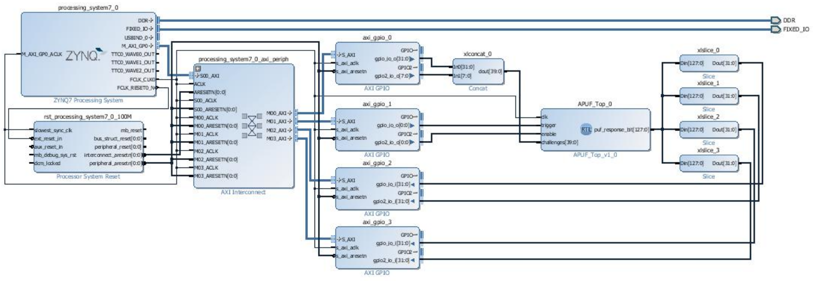
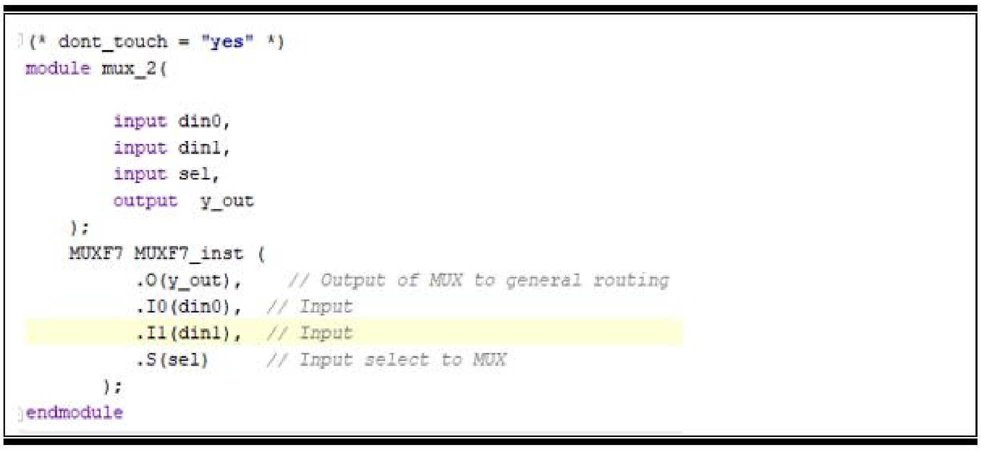


Figure 10. Top level design of the PDL- Arbiter PUF system in Vivado IP Integrator

3.3 Wide-MUX based Arbiter PUF

An alternative method to implement the Arbiter PUF is a wide Mux-based APUF that is less complicated than PDL. The authors [34] [36] [37] also used a wide Mux-based APUF, but the way of implementation of [34] and [36] Authors were different than this. The authors [34] utilized 3-1 DAPUF, that is an enhanced variant of 2-2 DAPUF, with three selector chains. [36] utilized a combination of LUTs and multiplexers. Authors [37] have implanted Double Arbiter PUF. We use the available wide multiplexers in the Xilinx 7 series FPGA in this technique, and it is straightforward to build and uses fewer resources. Multiplexer primitive shown in figure 11 has directly instantiated and formed the parallel path like a classical Arbiter design.


```
(* dont_touch = "yes" *)
module mux_2(

        input din0,
        input din1,
        input sel,
        output  y_out
    );
    MUXF7 MUXF7_inst (
          .O(y_out),    // Output of MUX to general routing
          .I0(din0),  // Input
          .I1(din1),  // Input
          .S(sel)     // Input select to MUX
        );
endmodule
```

Figure 11. RTL of the Mux_2 primitive

The synthesis design consists of a long chain of multiplexers. Figure 12 shows the synthesis schematic of the wide Mux-based APUF design strategy. Here we can see the MUXF7 is the resource primitive and the way they are connected using the swapped paths.

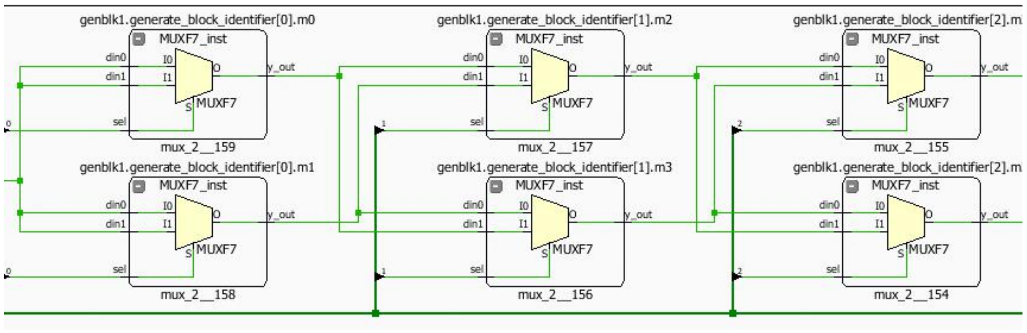

genblk1.generate_block_identifier[0].m0
MUXF7_inst
genblk1.generate_block_identifier[1].m2
genblk1.generate_block_identifier[2].m
mux_2__159
mux_2__157
mux_2__155
genblk1.generate_block_identifier[0].m1
genblk1.generate_block_identifier[1].m3
mux_2__158
mux_2__156
mux_2__154

Figure 12. Synthesis schematic of Wide-MUX-Arbiter PUF

Like a previous design here also created the 128-bit Wide-Mux APUF system block diagram by connecting all the required Ips in the Xilinx IP integrator as shown in figure 12.

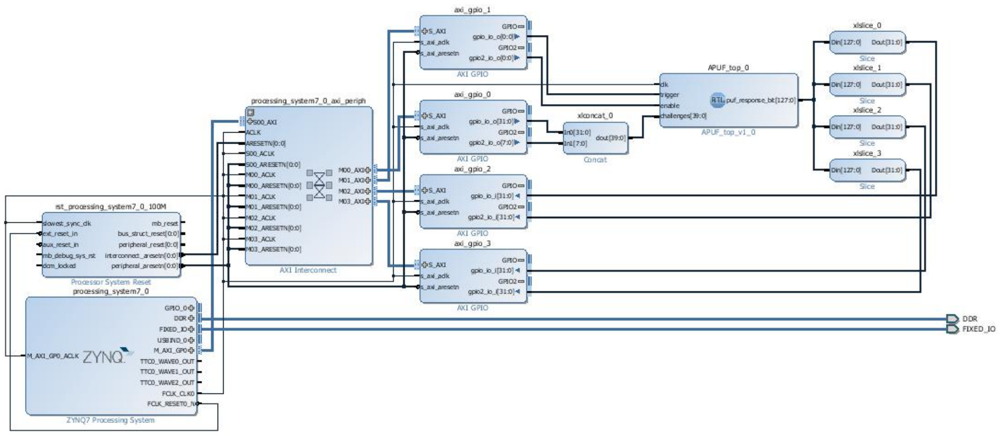


Figure 12. Top-level design of the 128-bit wide- Mux Arbiter PUF system in Vivado IP Integrator

Tcl scripting provides the manual placement of the gate-level netlist. set_property BEL and set_property LOC these two Tcl commands are used to place the design to the desired location. After the manual placing device well looks like Figure 13(a) Classical APUF, (b) PDL based APUF and (c) wide Mux-based APUF.

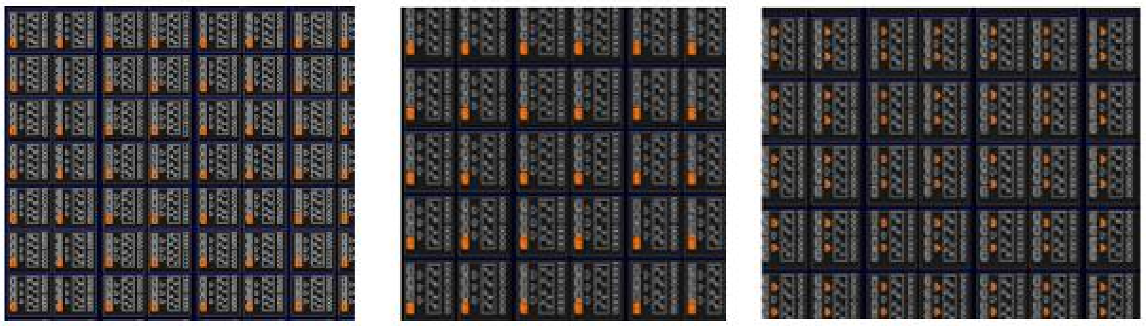


Figure 13 (a) Classical Arbiter PUF (b) PDL Based Arbiter PUF (c) Wide-MUX based Arbiter PUF

The Figure 14 (a), (b) and (c) shows the internal routing details of all three types of the designs respectively. Here the symmetric routing between the switching elements is maintained.

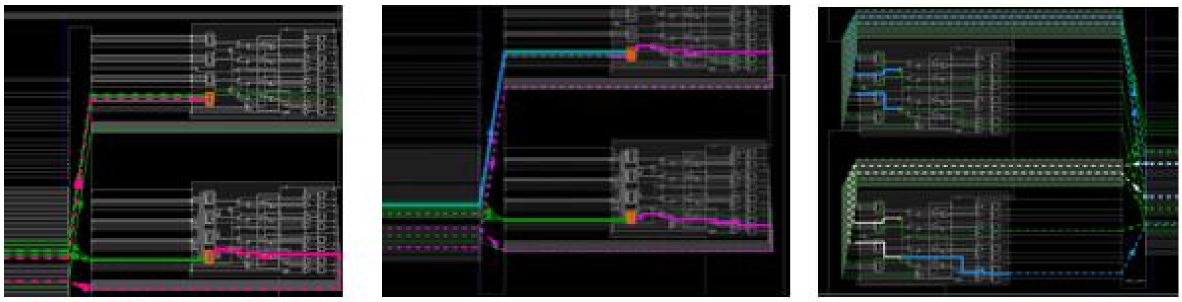


Figure 14 (a) Classical Arbiter PUF (b) PDL Based Arbiter PUF (c) Wide-MUX based Arbiter PUF

4. Experimental setup and Observations

4.1 Properties of PUF

1. **Reliability**: After several readings, reliability assesses how consistently the PUF replicates a response for the same challenge. Reliability is used to calculate the average intra-class score. The Hamming distance, HD (R, R'), is determined over x samples in the following way: For a single chip, represented as i, has n-bit reference response Ri(n) from the chip i at normal operating conditions and the same n-bit response obtained at different conditions R′i (n), respectively, for the challenge C, the average intra-chip HD for k samples/chips is defined as:

$$R = \frac{1}{x}\sum_{y=1}^{x}\frac{HD(Ri,R'iy)}{n} * 100 \qquad \text{................ (1)}$$

2. **Uniqueness**: To determine uniqueness, we utilize the Hamming distance (HD) between two PUF identifiers. If two chips, i and j (i != j), have n-bit replies, Ri and Rj for challenge C, respectively.

The PUF's uniqueness is determined by how well it can identify the device on which it is used. The average inter-class Hamming distance is computed as follows:

$$U = \frac{2}{m(m-1)} \sum_{i=1}^{m-1} \sum_{j=j+1}^{m} \frac{HD(Ri,Rj)}{n} * 100 \qquad \text{…………… (2)}$$

Here M is the number of devices, and Ri and Rj *are* the *n*-bit responses of *i*th and *j*th PUF instances respectively. The uniqueness of the PUF responses gets reduced if all of the chips have the same bit value ('0' or '1').

3. **Uniformity**: The uniformity of a PUF is a measure of how consistent the proportion of 0s and 1s in a PUF's response bits is. This percentage must be 50% for fully random PUF replies. The percentage Hamming Weight is used to measure the homogeneity of an n-bit PUF identification (HW).

$$(Uniformity)i = \frac{1}{n} \sum_{l=0}^{n} ri,l * 100\% \qquad \text{……………………(3)}$$

4.2 Experimental Setup

Zynq processor has an embedded Universal Asynchronous Receiver/Transmitter (UART) connected to the serial terminal of the laptop. The FPGA experimental setup is described in Figure 15. Xilinx Software Development Kit (XSDK) [40] is used to test the design of the software using C code. The SDK console terminal is used for the observation of the outputs. We developed the **set_challenge** function such that 10,000 unique and random challenges were applied 100 times to the design. We extracted the response bits from the PUF using the function **get_response**. The responses were then recorded by UART using the console terminal of SDK.

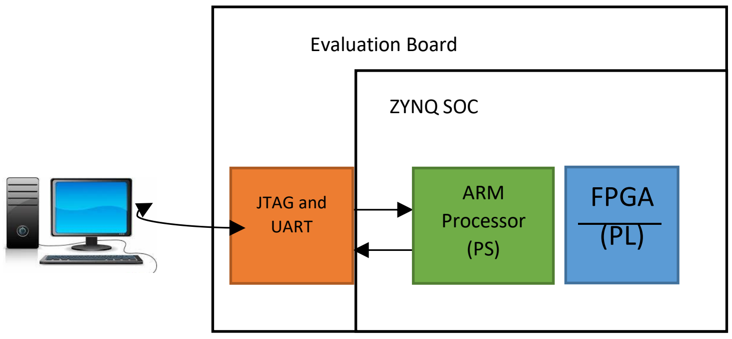
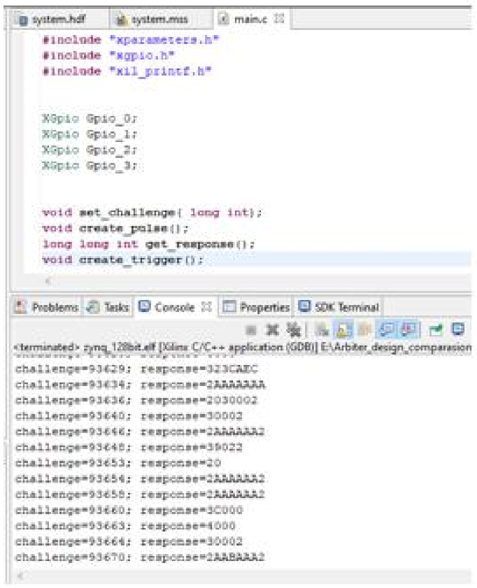


Figure 15 Experimental Setup [24]

PDL-based Arbiter and Wide-MUX-based arbiter, the challenge and response bits on the XSDK console has shown in Figures 16 (a) and (b). The experiment is conducted at ambient temperature. The PUF output responses are saved in a CSV file. The .csv file is imported into the MATLAB tool for finding out the characteristics of PUF like Uniformity, uniqueness, and reliability [24].

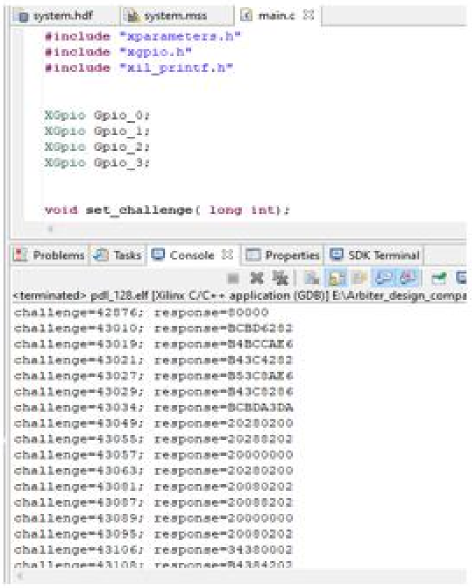


```
#include "xparameters.h"
#include "xgpio.h"
#include "xil_printf.h"


XGpio Gpio_0;
XGpio Gpio_1;
XGpio Gpio_2;
XGpio Gpio_3;


void set_challenge( long int);
void create_pulse();
long long int get_response();
void create_trigger();
```

```
<terminated> zynq_128bit.elf [Xilinx C/C++ application (GDB)] E:\Arbiter_design_comparasion
challenge=93629; response=323CAEC
challenge=93634; response=2AAAAAAA
challenge=93636; response=2030002
challenge=93640; response=30002
challenge=93646; response=2AAAAAA2
challenge=93648; response=39022
challenge=93653; response=20
challenge=93654; response=2AAAAAA2
challenge=93658; response=2AAAAAA2
challenge=93660; response=3C000
challenge=93663; response=4000
challenge=93664; response=30002
challenge=93670; response=2AABAAA2
```



```
#include "xparameters.h"
#include "xgpio.h"
#include "xil_printf.h"


XGpio Gpio_0;
XGpio Gpio_1;
XGpio Gpio_2;
XGpio Gpio_3;


void set_challenge( long int);
```

```
<terminated> pdl_128.elf [Xilinx C/C++ application (GDB)] E:\Arbiter_design_compa
challenge=42876; response=80000
challenge=43010; response=BCBD6282
challenge=43019; response=B4BCCAE6
challenge=43021; response=B43C4282
challenge=43027; response=B53C8AE6
challenge=43029; response=B43C8286
challenge=43034; response=BCBDA3DA
challenge=43049; response=20280200
challenge=43055; response=20288202
challenge=43057; response=20000000
challenge=43063; response=20280200
challenge=43081; response=20080202
challenge=43087; response=20088202
challenge=43089; response=20000000
challenge=43095; response=20080202
challenge=43106; response=34380002
challenge=43108; response=B4384202
```

Figure 16 (a) SDK output of PDL based Arbiter. **Figure 16 (b) SDK output of the mux-Arbiter**

4.3 Results

We have implemented 64-bit Classical APUF, 64 and 128-bit PDL-based APUF and MUX-based APUF. Table 1 shows the different cases of the APUF implementations. Table 1 shows that PDL based Arbiter and Wide-MUX based Arbiter have better performance than Classical APUF.

Table 1: Characteristics of PUF

Parameters	Expected Value	Classical Arbiter	PDL Based Arbiter		Wide-MUX based Arbiter	
	(Theoretical)	64-bit	64-bit	128-bit	64-bit	128-bit
Uniqueness	50%	33.08	44.3	40.02	46.4	47.7
Uniformity	50%	34.2	49.56	51.27	51.87	52.24
Reliability	100%	77.02	97.03	98.05	98.06	98.01

Table 2: Observations of the design

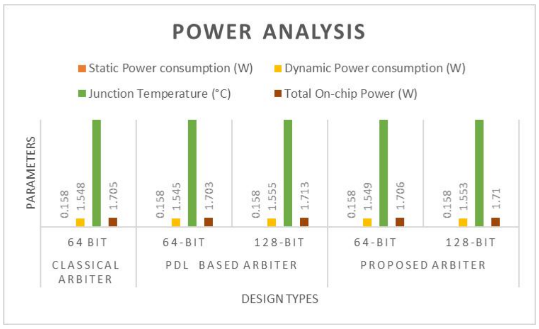
Sr. nos.	Criteria	Observations
Case I	Same design on two different boards (inter) with same challenges.	We observed unique and random response bits. It proves the uniqueness of the PUF
Case II	Same design, same board but different clock region (intra) with same challenges	We observed unique and random response bits.
Case III	Same design, Same board, same clocking region, with same challenges but on a different day and time	We observed same response bits again and again. It proves the reliability of the PUF
Case IV	Same design, same board, same region but same day and time with multiple challenges.	We observed unique and random response bits.

Table 3: Power Analysis Report

Parameters	Classical Arbiter	PDL -based Arbiter		Wide-MUX-based Arbiter	
	64-bit	64-bit	128-bit	64-bit	128-bit
Static Power consumption (W)	0.158	0.158	0.158	0.158	0.158
Dynamic Power consumption (W)	1.548	1.545	1.555	1.549	1.553
Junction Temperature (°C)	44.7	44.6	44.8	44.7	44.7
Total On-chip Power (W)	1.705	1.703	1.713	1.706	1.71

Table 4: Resource Utilization Report

Resource	Available	Classical Arbiter	PDL-based Arbiter		Wide-MUX-based Arbiter	
		64-bit	64-bit	128-bit	64-bit	128-bit
		Utilization				
LUT	53200	5184	2624	5248	64	128
Slice	13300	5174	2591	5181	2580	5179
FF	106400	128	128	256	128	256
MUX	26600	0	0	0	5120	10240



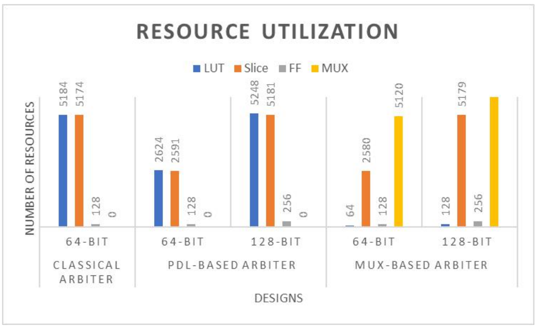


Figure 17 (a) Power Analysis **Figure 17 (b) Resource Utilization**

The design's power analysis is shown in Figure 17 (a). The total on-chip power, junction temperature, and thermal margin are all determined via power analysis. The system logic that has been implemented consumes 61 percent of the overall power. The resource's use overview is shown in Figure 17 (b). The design consumes less than 3% of total FPGA Logic resources and less than 10% of IO resources.

Conclusion:

Hardware security is a vast subject. PUF remains a promising method for safe key generation on chips for cryptographic applications. We successfully designed and implemented different design styles of APUF on the Xilinx SOC FPGA. We noted that the Wide- MUX-based APUF utilizes the optimal resources, power, and speed. The Wide- MUX-based APUF design is quite simple to compare to the earlier approach. Our future research will perform a Static timing analysis of this design and find out the valid CRPs for the IoT applications.

References:

[1] Pappu R, Recht B, Taylor J, Gershenfeld N. Physical one-way functions. Science. 2002 Sep 20;297(5589):2026-30.

[2] Gassend B, Clarke D, Van Dijk M, Devadas S. Silicon physical random functions. InProceedings of the 9th ACM Conference on Computer and Communications Security 2002 Nov 18 (pp. 148-160).

[3] Suh GE, Devadas S. Physical unclonable functions for device authentication and secret key generation. In2007 44th ACM/IEEE Design Automation Conference 2007 Jun 4 (pp. 9-14). IEEE.

[4] Suh GE, Devadas S. Physical unclonable functions for device authentication and secret key generation. In2007 44th ACM/IEEE Design Automation Conference 2007 Jun 4 (pp. 9-14). IEEE.

[5] Maiti A, Schaumont P. Improved ring oscillator PUF: An FPGA-friendly secure primitive. Journal of cryptology. 2011 Apr;24(2):375-97.

[6] Rührmair U, Holcomb DE. PUFs at a glance. In2014 Design, Automation & Test in Europe Conference & Exhibition (DATE) 2014 Mar 24 (pp. 1-6). IEEE.

[7] Lee JW, Lim D, Gassend B, Suh GE, Van Dijk M, Devadas S. A technique to build a secret key in integrated circuits for identification and authentication applications. In2004 Symposium on VLSI Circuits. Digest of Technical Papers (IEEE Cat. No. 04CH37525) 2004 Jun 17 (pp. 176-179). IEEE.

[8] Kumar SS, Guajardo J, Maes R, Schrijen GJ, Tuyls P. The butterfly PUF protecting IP on every FPGA. In2008 IEEE International Workshop on Hardware-Oriented Security and Trust 2008 Jun 9 (pp. 67-70). IEEE.

[9] Suzuki D, Shimizu K. The glitch PUF: A new delay-PUF architecture exploiting glitch shapes. InInternational Workshop on Cryptographic Hardware and Embedded Systems 2010 Aug 17 (pp. 366-382). Springer, Berlin, Heidelberg.

[10] Holcomb DE, Burleson WP, Fu K. Initial SRAM state as a fingerprint and source of true random numbers for RFID tags. InProceedings of the Conference on RFID Security 2007 Jul 11 (Vol. 7, No. 2, p. 01).

[11] Hata H, Ichikawa S. FPGA implementation of metastability-based true random number generator. IEICE TRANSACTIONS on Information and Systems. 2012 Feb 1;95(2):426-36.Handschuh H, Schrijen GJ, Tuyls P. Hardware intrinsic security from physically unclonable functions. InTowards Hardware-Intrinsic Security 2010 (pp. 39-53). Springer, Berlin, Heidelberg.

[12] Sklavos N. Securing communication devices via physical unclonable functions (PUFs). InISSE 2013 Securing Electronic Business Processes 2013 (pp. 253-261). Springer Vieweg, Wiesbaden.

[13] Rührmair U, Busch H, Katzenbeisser S. Strong PUFs: models, constructions, and security proofs. InTowards hardware-intrinsic security 2010 (pp. 79-96). Springer, Berlin, Heidelberg.

[14] Zhang JL, Wu Q, Ding YP, Lv YQ, Zhou Q, Xia ZH, Sun XM, Wang XW. Techniques for design and implementation of an FPGA-specific physical unclonable function. Journal of Computer Science and Technology. 2016 Jan 1;31(1):124-36.

[15] Morozov S, Maiti A, Schaumont P. A Comparative Analysis of Delay Based PUF Implementations on FPGA. IACR Cryptol. ePrint Arch.. 2009 Dec 19;2009:629.

[16] Guajardo J, Kumar SS, Schrijen GJ, Tuyls P. FPGA intrinsic PUFs and their use for IP protection. InInternational workshop on cryptographic hardware and embedded systems 2007 Sep 10 (pp. 63-80). Springer, Berlin, Heidelberg.

[17] Vaikuntapu R, Bhargava L, Sahula V. Golden IC free methodology for hardware trojan detection using symmetric path delays. In2016 20th International Symposium on VLSI Design and Test (VDAT) 2016 May 24 (pp. 1-2). IEEE.

[18] Ning H, Farha F, Ullah A, Mao L. Physical unclonable function: architectures, applications and challenges for dependable security. IET Circuits, Devices & Systems. 2020 Feb 7;14(4):407-24.

[19] Sahoo DP, Nguyen PH, Chakraborty RS, Mukhopadhyay D. Architectural Bias: a Novel Statistical Metric to Evaluate Arbiter PUF Variants. IACR Cryptol. ePrint Arch.. 2016;2016:57.

[20] Dubrova E. A reconfigurable arbiter PUF with 4 x 4 switch blocks. In2018 IEEE 48th International Symposium on Multiple-Valued Logic (ISMVL) 2018 May 16 (pp. 31-37). IEEE.

[21] Chatterjee D, Mukhopadhyay D, Hazra A. Interpose PUF can be PAC Learned. IACR Cryptol. ePrint Arch.. 2020;2020:471.

[22] Hori Y, Yoshida T, Katashita T, Satoh A. Quantitative and statistical performance evaluation of arbiter physical unclonable functions on FPGAs. In2010 International Conference on Reconfigurable Computing and FPGAs 2010 Dec 13 (pp. 298-303). IEEE.

[23] Chatterjee U, Chakraborty RS, Kapoor H, Mukhopadhyay D. Theory and application of delay constraints in arbiter PUF. ACM Transactions on Embedded Computing Systems (TECS). 2016 Jan 28;15(1):1-20.

[24] Alkatheiri MS, Zhuang Y, Korobkov M, Sangi AR. An experimental study of the state-of-the-art PUFs implemented on FPGAs. In2017 IEEE Conference on Dependable and Secure Computing 2017 Aug 7 (pp. 174-180). IEEE.

[25] Devadas S, Kharaya A, Koushanfar F, Majzoobi M. Automated design, implementation, and evaluation of arbiter-based PUF on FPGA using programmable delay lines. 2014 Aug 18.

[26] Anandakumar NN, Hashmi MS, Sanadhya SK. Compact implementations of FPGA-based PUFs with enhanced performance. In2017 30th International Conference on VLSI Design

and 2017 16th International Conference on Embedded Systems (VLSID) 2017 Jan 7 (pp. 161-166). IEEE.

[27] Habib B, Gaj K, Kaps JP. FPGA PUF based on programmable LUT delays. In2013 Euromicro Conference on Digital System Design 2013 Sep 4 (pp. 697-704). IEEE.

[28] Majzoobi M, Koushanfar F, Devadas S. FPGA PUF using programmable delay lines. In2010 IEEE international workshop on information forensics and security 2010 Dec 12 (pp. 1-6). IEEE.

[29] Tajik S, Dietz E, Frohmann S, Seifert JP, Nedospasov D, Helfmeier C, Boit C, Dittrich H. Physical characterization of arbiter PUFs. InInternational Workshop on Cryptographic Hardware and Embedded Systems 2014 Sep 23 (pp. 493-509). Springer, Berlin, Heidelberg.

[30] Sahoo DP, Chakraborty RS, Mukhopadhyay D. Towards ideal arbiter PUF design on Xilinx FPGA: A practitioner's perspective. In2015 Euromicro Conference on Digital System Design 2015 Aug 26 (pp. 559-562). IEEE.

[31] Gehrer S, Sigl G. Using the reconfigurability of modern FPGAs for highly efficient PUF-based key generation. In2015 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC) 2015 Jun 29 (pp. 1-6). IEEE.

[32] Pei S, Zhang J, Wang R. A low-overhead RO PUF design for Xilinx FPGAs. IEICE Electronics Express. 2018;15(5):20180093-.

[33] Machida T, Yamamoto D, Iwamoto M, Sakiyama K. A new arbiter PUF for enhancing unpredictability on FPGA. The Scientific World Journal. 2015 Jan 1;2015.

[34] Machida T, Yamamoto D, Iwamoto M, Sakiyama K. A new mode of operation for arbiter PUF to improve uniqueness on FPGA. In2014 Federated Conference on Computer Science and Information Systems 2014 Sep 7 (pp. 871-878). IEEE.

[35] Kulkarni S, Vani RM, Hunagund PV. FPGA based Hardware Security for Edge Devices in Internet of Things. In2020 5th International Conference on Communication and Electronics Systems (ICCES) 2020 Jun 10 (pp. 1133-1138). IEEE.Cui Y, Wang C, Chen Y, Wei Z, Chen M, Liu W. Dynamic Reconfigurable PUFs Based on FPGA. In2019 IEEE International Workshop on Signal Processing Systems (SiPS) 2019 Oct 20 (pp. 79-84). IEEE.

[36] Ikezaki Y, Nozaki Y, Yoshikawa M. IoT device oriented security module using PUF. In2016 IEEE International Meeting for Future of Electron Devices, Kansai (IMFEDK) 2016 Jun 23 (pp. 1-2). IEEE.

[37] Lao Y, Parhi KK. Statistical analysis of MUX-based physical unclonable functions. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2014 Apr 17;33(5):649-62.

[38] Kulkarni S, Vani RM, Hunagund PV. Designing of Arbiter PUF for Securing IP and IoT Devices. InData Intelligence and Cognitive Informatics 2021 (pp. 131-138). Springer, Singapore..

[39] Xilinx Vivado Design Suite User Guide Synthesis UG901 (v2019.2) January 27, 2020.

[40] https://www.xilinx.com/html_docs/xilinx2015_1/SDK_Doc/index.html