A NOVEL MICROPROCESSOR-INTRINSIC PHYSICAL UNCLONABLE FUNCTION

Abhranil Maiti

Electrical & Computer Engineering
Virginia Tech
Blacksburg, Virginia, 24061
email: abhranil@vt.edu

Patrick Schaumont

Electrical & Computer Engineering
Virginia Tech
Blacksburg, Virginia, 24061
email: schaum@vt.edu

ABSTRACT

We present a novel Physical Unclonable Function (PUF) exploiting the variability existing in a microprocessor pipeline to uniquely identify the microprocessor chip. The PUF accepts a microprocessor instruction as a challenge and produces the delay in a data path or a control path in the microprocessor as the response. The delay value is captured by over-clocking the microprocessor. The entire mechanism can be controlled by the microprocessor itself. Moreover, this PUF requires no dedicated hardware resources. It is a microprocessor-intrinsic PUF solution. We demonstrate our proposed idea using the SPARC instruction set implemented in a 32-bit LEON3 processor in a Spartan 3E FPGA. Our implementation based on the characterization of a subset of five SPARC instructions can produce 37 secure response bits for authentication.

1. INTRODUCTION

An on-chip Physical Unclonable Function (PUF) exploits manufacturing process variation inside integrated circuits (ICs) to generate chip-unique challenge-response pairs (CRPs). The relation between a challenge and the corresponding response is determined by complex statistical variation in logic and interconnect in an IC. Many useful applications for PUFs have been proposed so far due to their ability to generate a random key, to store a key without dedicated memory, and to prevent physical tampering. For example, they can be used in device authentication and secret key generation [1]. Guajardo et al. discussed the use of PUFs for Intellectual Property (IP) protection, remote service activation, and secret key storage [2]. A PUF-based RFID tag has also been proposed to prevent product counterfeiting [3].

In this paper, we introduce a novel PUF that extracts variability existing in a microprocessor pipeline to uniquely identify the microprocessor chip. This PUF accepts a microprocessor instruction as a challenge and produces the delay in a data path or a control path in the microprocessor as the response. The delay value is captured by operating the microprocessor pipeline at an elevated clock frequency, also known as over-clocking. The entire PUF mechanism can be controlled by the microprocessor itself using software programs. Moreover, the proposed PUF does not require any dedicated hardware or any type of modification in the existing hardware except a circuit that can vary the clock frequency of the microprocessor. We assume the availability of a digital clock manager (DCM) or a phase locked loop (PLL) that can be used for clock generation/variation in general, not just for the PUF. In an FPGA, a DCM or a PLL is available as a standard component and hence, can be used for this PUF with no extra cost. In this context, one may argue that an FPGA can be configured solely with the PUF for authentication and then reconfigured with the main application so that the area consumed by the PUF does not become an issue. However, configuring an FPGA separately for authentication and the main application will lead to the overhead of reconfiguration. In our method, the PUF can exist along with the main application while consuming minimal area.

Why do we need a new PUF? This is an important question, given the number of PUF proposals already available. Most of the proposed PUFs require a significant amount of hardware resources. For example, a ring oscillator-based PUF requires a pair of ring oscillators to generate one bit of PUF response [1]. A butterfly PUF consumes two latches to produce the same [4]. However, there are PUFs that can be implemented without the need of dedicated hardware resource; they are called *intrinsic* PUF. Guajardo et al. exploited random power-up values of SRAM cells to build an SRAM-PUF [5]. Similar work has been done by Holcomb et al. [6]. The main drawback of this type of PUFs is that they require power cycling. Hence, the random information is not easily accessible. Maes et al. showed that the power-up states of the D flip-flops (DFFs) inside an FPGA can be captured by modifying the programming bitfile [7]. In the DFF-based PUFs also, in order to extract the PUF CRPs, one has to power cycle the chip which is not the best practical solution.

Our proposed PUF is also intrinsic in nature. The key difference between the existing intrinsic PUFs and the pro-

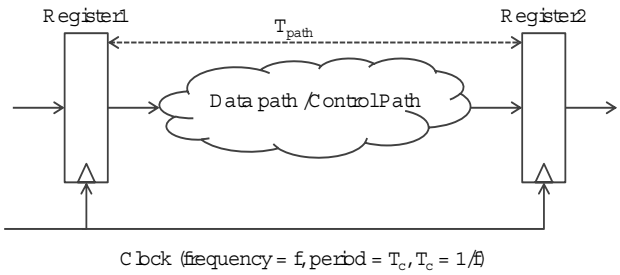

Fig. 1. A pipelined delay path

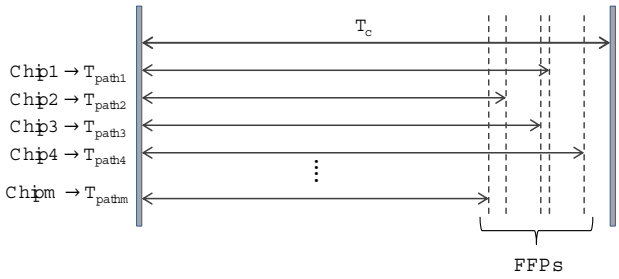

Fig. 2. Variable FFPs based on variable slacks in data path / control path

posed one is that our PUF does not depend on the power-up state of the circuit. As a result, the proposed PUF can evaluate the CRPs anytime during the operation of a circuit.

We identify several benefits out of the proposed PUF. First, a microprocessor is a ubiquitous circuit element, present in almost every embedded application. This makes it an ideal candidate for an intrinsic PUF. Second, we use microprocessor instructions to build the PUF challenge-response mechanism. This provides a way of integrating low-level hardware information (variability) with the high-level software applications (software programs). Finally, any post-processing of the PUF data can be flexibly done in software obviating any need of error-correction hardware. The main contributions of this work are :

- We first propose how variability in the data path and the control path of a microprocessor pipeline can be captured using microprocessor instructions. We introduce a challenge-response mechanism based on the variability information.
- We demonstrate the idea on a SPARC instruction set implemented in a 32-bit LEON3 processor in a Spartan 3E FPGA. We analyzed different types of instructions such as arithmetic instructions, logical instructions, and control instructions. Our results show good amount of uniqueness among the chips we tested in terms of the PUF responses. It produced 37 secure response bits for authentication based on a subset of five SPARC instructions. Furthermore, they also exhibit a high value of reliability.

The proposed approach or anything similar has not been reported in literature previously as far as our knowledge extends. In this paper, we focus on establishing the idea that process variation information inherent in a microprocessor can be extracted using microprocessor instructions to create a PUF challenge-response mechanism. A complete characterization and validation of this PUF in terms of stress testing (variation in temperature and supply voltage, aging) and security analysis are expected future work, but it is out of the scope of the present contribution. The rest of the paper is organized as follows. We introduce the detailed architecture of the proposed PUF in Section 2. We present the experimental results in Section 3. Section 4 discusses the related work. We conclude in Section 5.

2. MICROPROCESSOR-INTRINSIC PUF

In this PUF, we aim to exploit the delay variability that exists in several data paths and control paths in a microprocessor pipeline. Depending on the instruction that is run on the microprocessor, a particular set of data paths or control paths are activated. Due to manufacturing process variation, these paths have variable delay from one chip to another. We extract this variable delay to authenticate a chip. Figure 1 shows a typical pipelined delay path. In a microprocessor pipeline, similar paths exist. The clock period $T_{\rm c}$ is constrained by the following relationship:

$$T_{\rm c} \geq t_{\rm c\_q} + t_{\rm comb} + t_{\rm setup} \quad (1)$$

where $t_{\rm c\_q}$ is clock-to-output delay of the register 1, $t_{\rm comb}$ is the combinational path delay, and $t_{\rm setup}$ is the set up time of the register 2. We represent $t_{\rm c\_q} + t_{\rm comb} + t_{\rm setup}$ as $T_{\rm path}$. The value of $T_{\rm path}$ for a particular path varies from one chip to another due to process variation. In addition, the value of $T_{\rm path}$ also depends on the type of instruction being executed, as well as its operands. If we increase the clock frequency i.e. reduce the clock period, an instruction will fail once the value of the reduced $T_{\rm c}$ reaches the value of $T_{\rm path}$. In other words, the instruction will reach a failure point once the slack (which is the difference between $T_{\rm path}$ and the original $T_{\rm c}$) becomes negative. We call the point of failure for a single instruction a frequency failure point (FFP). This has been demonstrated in Figure 2; the value of $T_{\rm path}$ for a particular data path/control path varies randomly over a set of m chips. The instruction set of a microprocessor leads to a set of FFPs. The objective of the proposed PUF is to authenticate a microprocessor through the FFPs of its instruction set, or a suitable subset of it.

2.1. How do we generate a PUF response bit?

We first explain a simple concept that shows how a response bit can be generated using the FFPs corresponding to an instruction. We observed that an instruction does not stop executing abruptly at a particular frequency while the microprocessor is over-clocked. It goes through a transition phase in which there are several frequencies at which an instruction executes with partial failure. This means when run several times, it fails a fraction of the total number of times it is run. With further increase in the clock frequency, it fails completely beyond a certain frequency. Essentially, the failure of an instruction due to over-clocking occurs through a region and not at a point. Figure 3 shows such a scenario. (It is a hypothetical example and not based on real data.) It shows how a particular instruction fails due to over-clocking for three different chips. The failure behavior is estimated by running the instruction $k(k > 0)$ times at different frequencies and counting the number of times it is correctly executed; we call it the pass count (pc). It is represented along the Y-axis in Figure 3. Before the frequency f_1, the pc of chip1 is k. It starts failing partially at f_1. This means that the pc is a fraction of k at f_1. Finally, the pc becomes zero at f_2. The region between f_1 and f_2 (colored in gray) is the transition region for chip1 for the instruction executed. Due to process variation, chip2 and chip3 have their transition regions for the same instruction at different frequencies as illustrated in Figure 3. The transition region for chip2 and chip3 are not highlighted for simplicity. Now, to generate a response bit r at an arbitrary frequency f for a particular instruction, let us apply a simple quantization rule as follows:

$$r = \begin{cases} 1 \text{ if } pc/k > 0.5 \\ 0 \text{ otherwise} \end{cases} \quad (2)$$

Following the above rule, three chips in Figure 3 produce different response bits at different frequencies. For example, at f_2, chip1, chip2, and chip3 produce '0','1', and '1' respectively. At f_3, they produce '0', '0', and '1' respectively. We can create a string of response bits this way by sampling an instruction at different frequencies. This is the basic working principle of the proposed PUF.

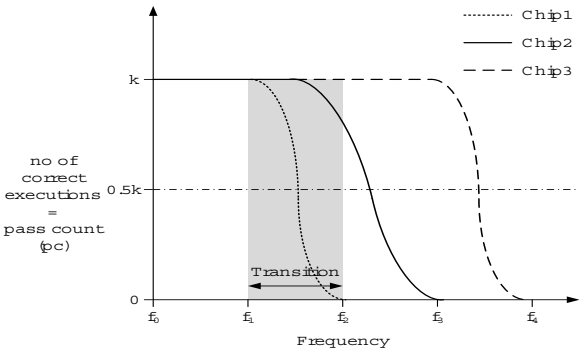

Fig. 3. Comparison of failure behavior of an instruction across a set of chips

2.2. Formalizing the PUF mechanism

We follow the standard PUF mechanism consisting of two phases: enrollment and evaluation. In the enrollment, we characterize several instructions at different levels of over-clocking to measure the transition regions. We slightly modify the basic quantization rule proposed in the previous section. We divide the transition region in four parts based on the pc value and assign a binary value (used as PUF response, r) to those sub-regions using the follow rule:

$$r = \begin{cases} 00 \text{ if } pc/k \leq 0.1 \\ 01 \text{ if } 0.1 < pc/k \leq 0.5 \\ 10 \text{ if } 0.5 < pc/k \leq 0.9 \\ 11 \text{ if } pc/k > 0.9 \end{cases} \quad (3)$$

The idea is depicted in Figure 4. Suppose, at five consecutive sampling frequencies in the transition region, a chip produces the following pc values for an instruction: 95, 87, 46, 17, and 5. Assuming k=100, the corresponding responses would be 11, 10, 01, 01, 00 according to Equation (3). The resultant response is the concatenation of these bit-strings i.e. 1110010100.

The region between $0.5k$ and $0.9k$ is more likely to execute an instruction correctly but it is distinct from the region where the processor has 100% pc. On the other hand, the region between $0.1k$ and $0.5k$ is more likely to execute an instruction incorrectly but it is distinct from the region where the pc is 0. This is the reason why we divided the transition region in four parts. The reason behind choosing the threshold of $0.1k$ and $0.9k$ is to reduce errors in response bits. Due to noise in the circuit, the pc around the frequency f_{start} (refer to Figure 4) in the transition region may fluctuate between 100% and values slightly lower than 100%. Similarly, near the frequency f_{end} (refer to Figure 4), it may fluctuate between 0 and values slightly higher than 0. During enrollment, we collect three types of information: a) the frequency at which an instruction starts failing partially. (the transition start point f_{start}) b) the frequency at which it completely stops executing. (the transition end point f_{end}) c) the sampling frequencies in the transition region and the pc values at the sampling frequencies including f_{start} and f_{end}. These three types of information are captured and stored in a secure database as failure transition information (FTI). Each of these three types of information are utilized for authentication as they vary from one chip to another.

During evaluation, a verifier can check the chip's claimed identity as follows. First, the verifier defines a challenge by selecting one or more instructions and a set of frequencies at which the instructions will be run. The claiming chip then

executes the selected instructions at those frequencies and returns the binary responses for each using Equation (3). Finally, the verifier calculates the binary responses using the FTI stored in his secure database, and compares with that returned by the chip. A match implies an authentic chip. To clone a chip, an adversary needs to know the precise detail of the FTI of a chip for all characterized instructions.

3. RESULTS

We characterized three types of instructions: arithmetic, logical, and control transfer. We present results for addition, unsigned multiplication, unsigned division, AND operation, and a branch instruction. We implemented our proposed PUF using the SPARC instruction set in a 32-bit LEON3 processor. It is tested on five Xilinx Spartan3E 500 FPGAs under normal temperature and regular supply voltage. The processor is configured with an internal RAM of 32 Kbytes along with a multiplier and a divider unit. It does not have any external memory, instruction cache or data cache. All the instructions are written in assembly code and loaded to the internal memory using the JTAG-based debug interface of the LEON3 processor.

It is important to mention that while the instructions characterized for PUF fail with over-clocking, rest of the microprocessor still functions correctly. When the clock frequency is restored to the normal value, the microprocessor comes back to fully functional state with the characterized instructions executing correctly. This is achieved by small, step-by-step variation of frequency. We used a step of 0.5 MHz. This way the microprocessor can run the PUF fully autonomously. We controlled the processor frequency by two different methods. Initially, we used an external, high-speed pulse generator. In a further refinement of the design, we replaced the external pulse generator by the on-board FPGA frequency generators (DCMs), to demonstrate that the proposed PUF can also run fully autonomously. The step of 0.5 MHz is easily available using the pulse generator. Even a DCM in a Virtex V FPGA can achieve this step as our implementation shows. Moreover, a frequency step as low as 0.038 MHz using a DCM in an FPGA has been reported in [8]. The results presented in this paper are generated using the frequency generator.

To save the value of pc in memory, we first characterized the FTI of the store instruction and ensured that the clock frequency never exceeds the transition region of the store instruction. This characterization also showed that the FTIs of the arithmetic instructions, logical instruction, and the control instructions are located at a much lower frequency than that of the store instruction. This ensures that the collected data represents the variability of the instruction we want to characterize (such as addition and multiplication) and not the variability of the store instruction.

3.1. Performance evaluation of the PUF

Two parameters are used to evaluate the performance of the PUF: uniqueness and reliability. Uniqueness is an estimate of the ability of a PUF to uniquely distinguish different chips based on the generated responses. Reliability quantifies the reproducibility of the response bits. If two chips, i and j $(i \neq j)$, have n-bit responses, R_i and R_j respectively for the challenge C, the uniqueness among m chips is defined as:

$$Uniqueness = \frac{2}{m(m-1)} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \frac{HD(R_i, R_j)}{n} \times 100\% \quad (4)$$

HD stands for Hamming distance. For a chip i, the reliability is estimated as follows.

$$Reliability = 100\% - \frac{1}{L} \sum_{l=1}^{L} \frac{HD(R_i, R'_{i,l})}{n} \times 100\% \quad (5)$$

where $R'_{i,l}$ is the l-th sample of R_i, and L is the total number of samples. In order to evaluate these parameters, we need

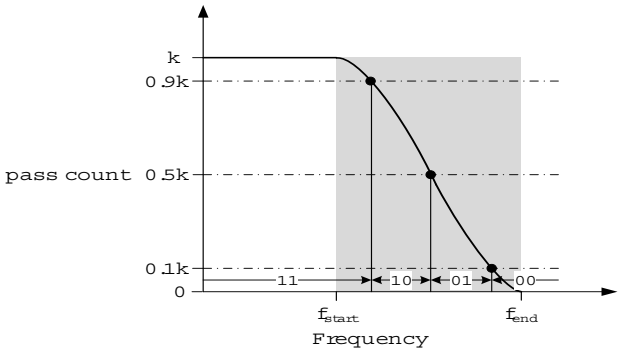

Fig. 4. Response generation in the proposed PUF

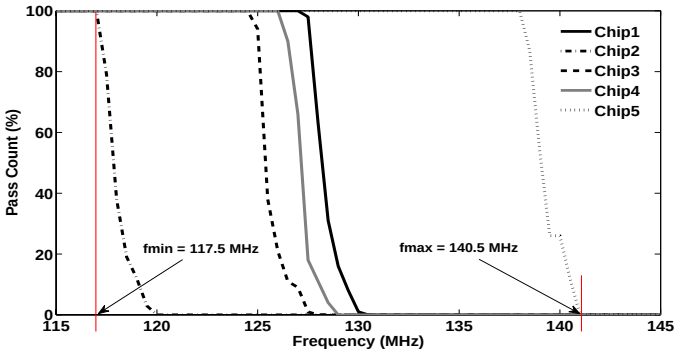

Fig. 5. Result for ADD 0x7FFFFFFF, 0x1

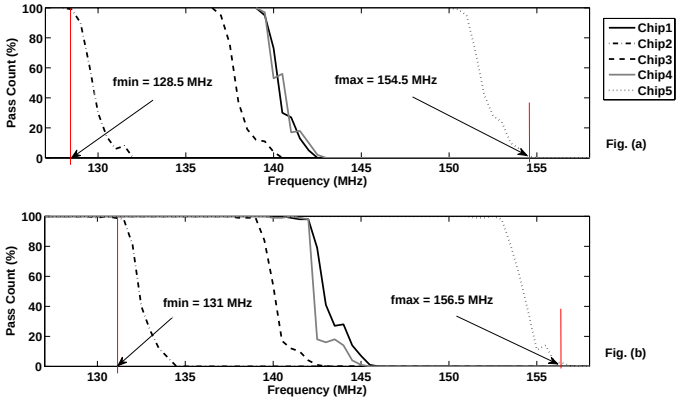

Fig. 6. (a)Result for MUL 0xFFFFFFFF, 0xFFFFFFFF (b)Result for MUL 0xFFFFFFFF, 0x80000001

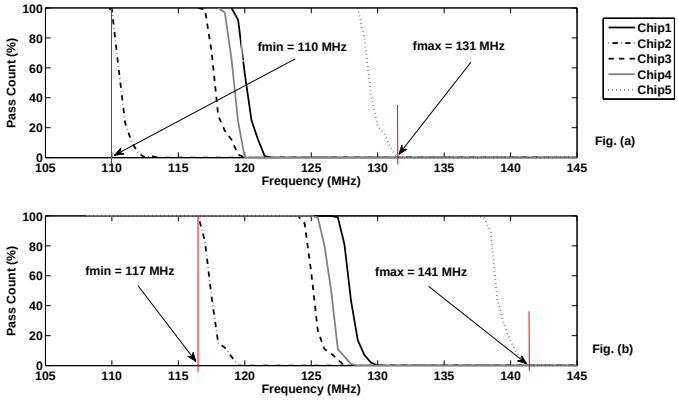

Fig. 7. (a)Result for DIV 0xFFFFFFFE00000001,0xFFFFFFFF (b)Result for DIV 0xFA0, 0x14

to form an n-bit response string using the PUF. For the proposed PUF, not only the transition regions for an instruction are located at different frequencies for different chips, the length of the transition region also varies from one chip to another. We used 90nm technology FPGAs, and the length of the transition region is 2 MHz on an average. This requires careful calibration of the PUF measurements. We use a simple method to construct the response string in order to evaluate PUF performances. Let us explain with an example of the addition instruction. Figure 5 shows the result in terms of pc values for the operation adding 0x7FFFFFFF and 0x1. The pc values are shown as percentage of k with k = 100 (refer to Equation (3)). Among the five chips, chip2 has the lowest f_{start} (shown as f_{min} in Figure 5), and chip5 has the highest f_{end} (shown as f_{max} in Figure 5). Before f_{min}, each of the chips has 100% pc value, and hence all of them produce '11' in this region. On the other hand, after f_{max}, all the chips have a pc value of 0 leading to '00' response by all. Therefore, there is no information available

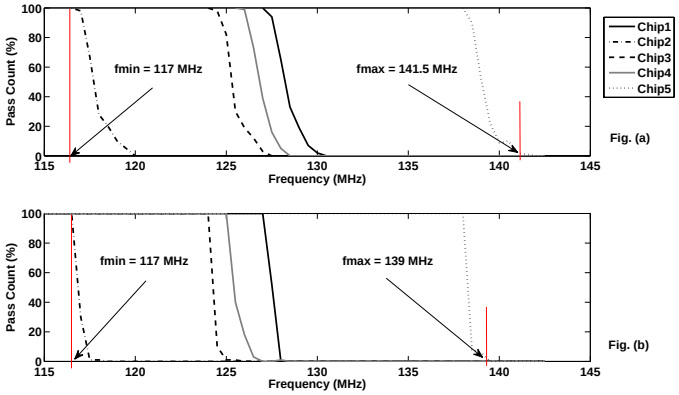

Fig. 8. (a)Result for AND 0xFFFFFFFF, 0xAAAAAAAA (b)Result for BGE instruction

outside the range from f_{min} to f_{max}. Therefore, we derive response bits using Equation (3) starting at f_{min} until f_{max} with a step of 0.5 MHz. With f_{min}= 117.5 MHz and f_{max} = 140.5 MHz, there are 47 sampling points in total each producing 2 response bits according to Equation (3). Therefore, there are 47 $\times$ 2 = 94 responses produced by each of the chips. The average uniqueness among five chips based on a 94-bit response string is 38.7% (refer to Table 1). We also tested different input operands such as different length of carry propagation. However, our observation is that no significant variability is found based on carry length variation for the addition instruction.

For the unsigned multiplication operation, we present results for two pairs of operands: (0xFFFFFFFF, 0xFFFFFFFF) and (0xFFFFFFFF, 0x80000001). Figure 6 shows that this instruction has an FTI that is dependent on the input operands. For each of the tested chips, the transition region for the two multiplications are located at different frequencies. For example, in Chip2, f_{start}=128.5 MHz for the first pair of operands (shown as f_{min} in Figure 6(a)) while f_{start}=131 MHz for the second pair of operands (shown as f_{min} in Figure 6(b)). Using the same method as described for the addition instruction, the multiplication instruction produced 106 response bits for 53 sampling frequencies for the first pair of operands. The second pair of operands produced 104 response bits for 52 sampling frequencies. The average uniqueness values are 36% and 36.1% respectively.

Similar to the unsigned multiplication operation, we observed operand dependency in case of the unsigned division instruction also. We present two cases: (0xFFFFFFFE00000001, 0xFFFFFFFF) and (0xFA0, 0x14). Figure 7 shows that the two pairs of operands led to FTIs that are located at distinct frequency locations. The uniqueness values are 38.1% and 37.3% respectively for the two cases respectively.

There are nine different logic instructions supported by the SPARC instruction set. We characterized all of them. The results show that all of the instruction have a very simi-

Table 1. Summary of PUF performance for different operations

Operation (No of Responses)	Uniqueness	Reliability
ADD 0x7FFFFFF, 0x1 (94)	38.7%	97.4%
MULT 0xFFFFFFFF, 0xFFFFFFFF (106)	36%	98.1%
MULT 0xFFFFFFFF, 0x80000001 (104)	36.1%	98%
DIV 0xFFFFFFFE00000001, 0xFFFFFFFF (86)	38.1%	99%
DIV 0x0000000000000FA0, 0x14 (98)	37.3%	95.6%
AND 0xFFFFFFFF, 0xAAAAAAAA (100)	36%	99%
BGE (90)	40.6%	98.3%

lar FTI. We have presented the result for the AND operation with the operand pair (0xFFFFFFFF, 0xAAAAAAAA) in Figure 8(a). The measured uniqueness is 36%.

Finally, we present results for the control transfer instruction. There are sixteen branch instructions supported by the SPARC instruction set. Like the logical operations, they also show similar FTIs. Figure 8(b) shows the result for the instruction BGE (branch on greater or equal). It can be noticed that unlike other types of instruction, the branch instruction has a very short transition region for all the chips and reaches a pc value of 0 abruptly. The average uniqueness produced by this instruction is 40.6%.

Table 1 shows a summary for different instructions used as challenge for the proposed PUF. Though the average uniqueness value shows good result, it deviates from the ideal value of 50% of fully random responses. This is because the proportion of '0's and '1's in the produced responses is partly biased. The smaller population size of five chips partially contributes to a low value of uniqueness. We present the uniqueness value without any post-processing. By using hashing or other techniques, as has been used by several other PUFs, the uniqueness can be improved. Moreover, the pre-transition region produces only '11' as response while the post-transition region produces only '00'. However, it still helps to distinguish one chip from another if two chips have different FTIs. The experimental results support that. For example, in Figure 5, from the frequency 120 MHz to 127.5 MHz, chip1 is in pre-transition region producing only '11' as responses. In the same frequency range, chip2 is in post-transition region producing only '00' as responses, and hence they are distinguished. The PUF shows high reliability. The reliability parameter is estimated based on 100 measurement samples. A value close to 100% indicates that the PUF responses are highly reliable at the normal operating condition. We plan the evaluation of environmental reliability (temperature, voltage) as future work.

3.2. Security analysis of the proposed PUF

We estimate the number of secure response bits for authentication. By secure bits, we mean those response bits that cannot be predicted given the information about other response bits. The number of secure response bits is a function of the number of instructions we characterize, as well as the frequency spacing with which we sample the transition region (0.5 MHz in our experiment). These are the design parameters of the proposed PUF.

Let us assume that an adversary is attempting to predict the response r at a frequency f_i. We assume that she knows the value of r at the previous sampling frequency f_{i-1} and the next sampling frequency f_{i+1} ($f_{i-1} < f_i < f_{i+1}$). Then she can guess r at f_i to some extent. If r at f_{i-1} and r at f_{i+1} are same, it is obvious that r at f_i has the same value too. This is because with increase in frequency, pc can either stay the same or decrease. On the other hand if r at f_{i-1} and r at f_{i+1} are not same, different possibilities may arise. Table 2 shows the possibilities depending on all possible combination of r at f_{i-1} and r at f_{i+1} when they are different. Assuming the possible values of r at f_i are equally likely, we can estimate the number of secure bits at f_i using Shannon's formula. For example, the first row in the third column in Table 2 shows that the possible value of r at f_i is '11' or '10' i.e two possibilities. Hence the number of secure bits is $2 \times (1/2) \times (-log_2(1/2)) = 1$. In this case, r cannot have values '01' and '00' as pc may either stay constant or may decrease with increase in frequency. Similarly, for the second row, it is $3 \times (1/3) \times (-log_3(1/3)) \approx 1.5$.

This analysis shows that secure bits are available at the transition regions. There are seven transition regions for five

Table 2. Secure response bit estimation chart

r at f_{i-1}	r at f_{i+1}	Possible r at f_i	No of secure bits at f_i
11	10	11, 10	1
11	01	11, 10, 01	1.5
11	00	11, 10, 01, 00	2
10	01	10, 01	1
10	00	10, 01, 00	1.5
01	00	01, 00	1

Table 3. Estimated Number of secure response bits

Chip1	Chip2	Chip3	Chip4	Chip5	Average
38	38	37	38	34	37

instructions with multiplication and division having two operations each. With each transition region producing around five secure bits on an average, each chip has 37 secure bits on an average as shown in table Table 3.

4. RELATED WORK

Besides the SRAM PUF and the DFF-based PUF that have been already discussed, there are several other works that are relevant to this work. Suh et al. proposed a secure processor architecture using PUF as a source of secret [9]. The proposed architecture provided secure environment for trusted computing by preventing physical attacks on memory and tampering of program. This work did not propose any new PUF; they used arbiter PUF in their architecture. In our work, we exploit the variability in a processor pipeline to build a new PUF. In another work, hardware authentication is achieved by calculating a checksum using micro-architectural complexity of a processor model [10]. This technique can distinguish one processor model from another; distinguishing one chip from the other is not possible by it. We instead focus on chip-level authentication.

Wong et al. used variable clock to characterize the delay variability in several components of an FPGA such as LUTs, flip-flops [8]. In our proposed PUF, we also employ variable clock frequency to a soft-core microprocessor in an FPGA in order to measure variability. However, the work by Wong et al. only focused on characterizing variability in generic FPGA components. We instead built a PUF mechanism using the variability in a microprocessor pipeline. Majzoobi et al. used a very similar technique as proposed in [8] to build a PUF [11]. However, their PUF exploits the delay variability of FPGA building blocks, and not of a microprocessor pipeline like ours.

5. CONCLUSION

We introduced a microprocessor-intrinsic PUF. We introduced the concept supported by an on-chip implementation and experimental results. The experimental results shows a moderate value of uniqueness in the response bits, whereas the reliability is high at normal operating condition. As part of the future work, we plan to characterize those instructions that have not been tested. We would also like to test the reliability of the PUF under varying temperature and supply voltage. A detailed security analysis is also a part of the future plan.

Acknowledgment

This research was supported in part through NSF grant no 0964680 and NSF grant no 0855095.

6. REFERENCES

[1] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Proceedings of the 44th annual Design Automation Conference*, ser. DAC '07, 2007, pp. 9–14.

[2] J. Guajardo, S. Kumar, G.-J. Schrijen, and P. Tuyls, "Brand and ip protection with physical unclonable functions," in *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, may 2008, pp. 3186 –3189.

[3] S. Devadas, E. Suh, S. Paral, R. Sowell, T. Ziola, and V. Khandelwal, "Design and implementation of puf-based "unclonable" rfid ics for anti-counterfeiting and security applications," in *RFID, 2008 IEEE International Conference on*, april 2008, pp. 58–64.

[4] S. Kumar, J. Guajardo, R. Maes, G.-J. Schrijen, and P. Tuyls, "Extended abstract: The butterfly puf protecting ip on every fpga," in *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*, 2008, pp. 67 –70.

[5] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, "Fpga intrinsic pufs and their use for ip protection," in *Proceedings of the 9th international workshop on Cryptographic Hardware and Embedded Systems*, ser. CHES '07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 63–80.

[6] D. Holcomb, W. Burleson, and K. Fu, "Power-up sram state as an identifying fingerprint and source of true random numbers," *Computers, IEEE Transactions on*, vol. 58, no. 9, pp. 1198 –1210, sept. 2009.

[7] R. Maes, P. Tuyls, and I. Verbauwhede, "Intrinsic pufs from flip-flops on reconfigurable devices," in *3rd Benelux Workshop on Information and System Security (WISSec 2008)*, Eindhoven,NL, 2008, p. 17.

[8] J. S. J. Wong, P. Sedcole, and P. Y. K. Cheung, "Self-measurement of combinatorial circuit delays in fpgas," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 2, pp. 10:1–10:22, June 2009.

[9] G. Suh, C. O'Donnell, I. Sachdev, and S. Devadas, "Design and implementation of the aegis single-chip secure processor using physical random functions," in *Computer Architecture, 2005. ISCA '05. Proceedings. 32nd International Symposium on*, june 2005, pp. 25 – 36.

[10] D. Y. Deng, A. H. Chan, and G. E. Suh, "Hardware authentication leveraging performance limits in detailed simulations and emulations," in *Proceedings of the 46th Annual Design Automation Conference*, ser. DAC '09, 2009, pp. 682–687.

[11] M. Majzoobi, A. Elnably, and F. Koushanfar, "Fpga time-bounded unclonable authentication," in *Proceedings of the 12th international conference on Information hiding*, ser. IH'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 1–16.