JES
Journal of Engineering Sciences

Vol 13, Issue 03, MARCH /2022

ISSN NO:0377-9254

IMPLEMENTATION OF SELF TIMED RING BASED TRNG WITH DIGITAL CLOCK MANAGERS USING VERILOG

[1]HAFSA HAKEEM, [2]Mr.T. CHAKRAPANI
[1]M.Tech Student, [2]Associate Professor
DEPARTMENT OF ECE
St.Johns College Of Engineering and Technology, Yemmiganur.

Abstract:
Random numbers are required for cryptographic applications such as IT security products, smart cards etc. Hardware based TRNGs are widely employed. Cryptographic algorithms are implemented on Field Programmable Gate Arrays (FPGAs). In this work a True random number generator (TRNG) employed for cryptography application was designed. The existing design was based on ring oscillators as a noise source. The proposed design was based on principle of self-timed ring with feedback structure (FSTR). To solve the shortcomings and Jitter from the oscillators being the source for the randomness, this work proposes an improved FSTR–TRNG architecture suitable for FPGA based applications. To the best of our knowledge this is the first reported work which incorporates tunability in a fully digital TRNG. The main advantages of the proposed TRNG are its on-the-fly tunability through Dynamic Partial Reconfiguration (DPR) to improve randomness qualities.

I.INTRODUCTION

TRNGs have become indispensable component in many cryptographic systems, including PIN/password generation, authentication protocols, key generation, random padding and nonce generation. TRNG circuits utilize a non-deterministic random process, usually in the form of electrical noise, as a basic source of randomness. Along with the noise source, a noise harvesting mechanism to extract the noise, and a post-processing stage to provide a uniform statistical distribution are other important components of the TRNG. Our focus is to design an improved FPGA based TRNGs, using purely digital components. Using digital building blocks for TRNGs has the advantage that the designs are relatively simple and well-suited to the FPGA design flow, as they can suitably leverage the CAD software tools available for FPGA design. However, digital circuits exhibit comparatively limited number of sources of random noise, e.g. metastability of circuit elements, frequency of free running oscillators and jitters (random phase shifts) in clock signals. As would be evident, our proposed TRNG circuit utilizes the frequency difference of two oscillators and oscillator jitter as sources of randomness.

Reconfigurable devices have become an integral part of many embedded digital systems, and predicted to become the platform of choice for general computing in near future. From being mainly prototyping devices, reconfigurable systems including FPGAs are being widely employed in cryptographic applications, as they can provide acceptable to high processing rate at much lower cost and faster design cycle time. Hence, many embedded systems in the domain of security require a high quality TRNG implementable on FPGA as a component. We present a TRNG for Xilinx FPGA based applications, which has a tunable jitter control capability based on DPR capabilities available on Xilinx FPGAs. The major contribution of this paper is the development of an architecture which allows on–the–fly tunabilty of statistical qualities of a TRNG by utilizing DPR capabilities of modern FPGAs for varying the DCM modeling parameters. To the best of our knowledge this is the first reported work which incorporates tunability in a TRNG.

This approach is only applicable for Xilinx FPGAs which provide programmable clock generation mechanism, and capability of DPR.DPR is a relatively new enhancement in FPGA technology, whereby modifications to predefined portions of the FPGA logic fabric is possible on–the–fly, without affecting the normal functionality of the FPGA. Xilinx Clock Management Tiles (CMTs) contain Dynamic Reconfiguration Port (DRP) which allow DPR to be performed through much simpler means [1]. Using DPR, the clock frequencies generated can be changed on–the–fly by adjusting the corresponding DCM parameters. DPR via DRP is an added advantage in FPGAs as it allows the user to tune the clock frequency as per the need. Design techniques exist to prevent any malicious manipulations via DPR which in other ways may detrimentally affect the security of the system [2].

Mersenne Twister (MT) is a widely-used fast TRNG (PRNG), designed by Matsumoto [8]. More CPU time is required for initialization than for generation in MT and hence, next to Mersenne Twisters, WELL generators were introduced by Panneton [9]. CPUs for personal computers later, acquired new features of SIMD operations (i.e., 128- bit operations) and multi-stage

pipelines.128-bit based PRNG was proposed which was named as SIMD-oriented Fast Mersenne Twister (SFMT), which is analogous to MT using SIMD operations proposed by Saito[7]. Tsoi[10] mentioned that if the function call is avoided, WELL may be slower than MT for some CPUs. The SFMT TRNG is a very fast generator with satisfactorily high-dimensional equidistribution property. Then TRNGs based on linear recurrences modulo 2 were introduced. Linear Feedback Shift Register TRNGs, also called Tausworthe generators, which work on linear recurrences modulo 2. Trinomial-based generators have important statistical defects, but combining them can yield generators that are relatively fast and robust. Such combinations have been proposed and analyzed by Matsumoto and Wang [11, 12]. The generators given in are for 32-bit computers. Nowadays 64-bit computers are becoming increasingly common and so it is important to have good generators designed to fully use the 64-bit words given by P. L'Ecuyer [6] The huge-period generators proposed thereafter were not quite optimal. New generators with better equidistribution and bit-mixing properties were required.

The state of these generators evolves in a more chaotic way than for the Mersenne twister. The reduction of the impact of persistent dependencies among successive output values can be observed in certain parts of the period of the Mersenne twister which was given by Saito [7]. A generator with a period of can be implemented using k flip-flops and k LUTs, and provides k random output bits each cycle. Despite these advantages, FPGA-optimized generators are not widely used in practice, as the process of constructing a generator for a given parameterization is time consuming, in terms of both developer man hours and CPU time. While it is possible to construct all possible generators ahead of time, the resulting set of cores would require many megabytes, and be difficult to integrate into existing tools and design flows. Faced with these unpalatable choices, engineers under time constraints understandably choose less efficient methods, such as combined Tausworthe generators [3] or parallel linear feedback shift registers (LFSRs). using cheap bit-wise shift-registers to provide long periods and good quality without requiring expensive resources. The number of bits generated per cycle is chosen generally to meet the needs of the application.

Permutation of the resulting outputs is given to the XOR gates. The output of the XOR gates are then given to the PIPO SRs, where the XOR gate outputs are shifted and thus random number generation takes place successfully. The Random Number Generation is performed as per the methodology. The simulations are performed in Model Sim 6.4a which is a tool and synthesized using Xilinx Plan Ahead Virtex5 kit verified on the Spartan 3E kit and the programming is written using Verilog.

The results that are obtained from the tools and the design summary obtained from Xilinx 8.1i are shown below. The initial seed is given as input. The seed is permuted.
The results for 8-bit RNG are discussed below. The same scheme is carried out for 64-bit RNG. The permuted bits' output is given to the XOR gates. For 8-bit RNG the number of XOR gates is 8(t=8). The concept of permutation is used up for improving randomness among bits and thus employing unpredictability.

1.2 problem statement

The first and last bits are interchanged. The same concept of permutation is used for different bit RNGs. The permuted outputs are fed into the XOR gates and for remaining inputs to XOR gates round basis is used. Thus, the obtained XOR gate output bits are fed in a parallel basis into the PIPO SR. The resulting outputs generate the random number cycle. The cycle is fed into the SISO SR [FIFO] of varying lengths (length=k).

The length should not exceed r. As each bit crosses the flip-flop, it will be set to zero. Thus, random number generation takes place. The resulting random numbers are generated such that their period is 2r -1. If the number of bits is 16, then the period is 216-1. The count of all zero state is reduced since the all zero state leads to idle condition. Random number and random bit generators, RNGs and RBGs, respectively, are a fundamental tool in many different areas. The two main fields of application are stochastic simulation and cryptography. In stochastic simulation, RNGs are used for mimicking the behavior of a random variable with a given probability distribution. In cryptography, these generators are employed to produce secret keys, to encrypt messages or to mask the content of certain protocols by combining the content with a random sequence. A further application of cryptographically secure random numbers is the growing area of internet gambling since these games should imitate very closely the distribution properties of their real equivalents and must not be predictable or influenceable by any adversary.

A TRNG is an algorithm that, based on an initial seed or by means of continuous input, produces a sequence of numbers or respectively bits. We demand that this sequence appears "random" to any observer. This topic leads us to the question: What is random? Most people will claim that they know what randomness means, but if they are asked to give an exact definition they will have a problem doing so. In most cases terms like unpredictable or uniformly distributed will be used in the attempt to describe the necessary properties of random numbers. However, when can a particular number or output string be called unpredictable or uniformly distributed? In Part I we will introduce three different approaches to define randomness or related notions.

In the context of random numbers and RNGs the notions of "real" random numbers and TRNGs (TRNGs)

appear quite frequently. By real random numbers we mean the independent realizations of a uniformly distributed random variable, by TRNGs we denote generators that output the result of a physical experiment which is considered to be random, like radioactive decay or the noise of a semiconductor diode. In certain circumstances, RNGs employ TRNGs in connection with an additional algorithm to produce a sequence that behaves almost like real random numbers. However, why would we use RNGs instead of TRNGs? First of all, TRNGs are often biased, this means for example that on average their output might contain more ones than zeros and therefore does not correspond to a uniformly distributed random variable. This effect can be balanced by different means, but this post-processing reduces the number of useful bits as well as the efficiency of the generator.

Another problem is that some TRNGs are very expensive or need at least an extra hardware device. In addition, these generators are often too slow for the intended applications. Ordinary RNGs need no additional hardware, are much faster than TRNGs, and their output fulfills the fundamental conditions, like unbiasedness, that are expected from random numbers. These conditions are required for high quality RNGs but they cannot be generalized to the wide range of available generators. Despite the arguments above, TRNGs have their place in the arsenal. They are used to generate the seed or the continuous input for RNGs. In [Ell95] the author lists several hardware sources that can be applied for such a purpose. Independently of whether a RNG is used for stochastic simulation or for cryptographic applications, it has to satisfy certain conditions. First of all the output should imitate the realization of a sequence of independent uniformly distributed random variables. Random variables that are not uniformly distributed can be simulated by applying specific transformations on the output of uniformly distributed generators, see [Dev96] for some examples or [HL00], which provides a program library that allows to produce non-uniform random numbers from uniform RNGs. In this paper we limit our discussion on generators that imitate uniformly distributed variables. In a binary sequence that was produced by independent and identically (i.i.d.) uniform random variables the ones and zeros as well as all binary n-tuples for $n \geq 1$ are uniformly distributed in the n-dimensional space.

Further more there exists no correlation between individual bits or n-tuples, respectively. From the output of a high quality RNG we expect the same behavior. For some generators those conditions can be checked by theoretical analysis, but for most RNGs they are checked by means of empirical tests. Moreover, a good RNG should work efficiently, which means it should be able to produce a large amount of random numbers in a short period of time. For applications like stochastic simulation, stream ciphers, the masking of protocols or online gambling, huge amounts of random numbers are necessary and thus fast RNGs are required. In addition to the conditions above RNGs for cryptographic applications must be resistant against attacks, a scenario which is not relevant in stochastic simulation. This means that an adversary should not be able to guess any current, future, or previous output of the generator, even if he or she has some information about the input, the inner state, or the current or previous output of the RNG.

The topic of cryptographic RNGs concerns both mathematicians and engineers. Most of the time engineers are more interested in the design of specific RNGs or test suites, whereas mathematicians are more concerned with definitions of randomness, theoretical analysis of deterministic RNGs and the interpretation of empirical test results. In this thesis we try to address both disciplines by giving the description of five real-world cryptographic RNGs as well as the necessary mathematical background. We hope that this thesis helps to get a compact overview over the topic of cryptographic RNGs.

1.3 Random Number Generation: Types and Techniques

Random Number Generation: Types and Techniques A degree of randomness is built into the fabric of reality. It is impossible to say for certain what a baby's personality will be, how the temperature will fluctuate next week, or which way dice will land on their next roll. A planet in which everything could be predicted would be bland, and much of the excitement of life would be lost. Because randomness is so inherent in everyday life, many researchers have tried to either harvest or simulate its effect inside the digital realm. Before accomplishing this feat, however, many important questions need to be answered. What does it mean to be random? How does a person go about creating randomness, and how can he capture the randomness he encounters? How can someone know if an event or number sequence is random or not? Over generations, the answers to these questions have progressively been developed.

This paper takes a look at the current solutions, and attempts to organize the methods for creating chaos. Defining Random It is impossible to appreciate a TRNG without first understanding what it means to be random. Developing a well-rounded definition of randomness can be accomplished by studying a random phenomenon, such as a dice roll, and exploring what qualities makes it random. To begin, imagine that a family game includes a die to make things more interesting. In the first turn, the die rolls a five. By itself, the roll of five is completely random. However, as the game goes on, the sequence of rolls is five, five, five, and five. The family playing the game will not take long to realize that the die they received probably is not random.

1.4 objective

The goal of this paper is the design, analysis and implementation of an easy-to-design, improved, low-overhead, tunable TRNG for the FPGA platform. The following are our major contributions:

1) We investigate the limitations of the FSTR–TRNG [3] when implemented on a FPGA design platform. To solve the shortcomings, we propose an improved FSTR–TRNG architecture suitable for FPGA based applications. To the best of our knowledge this is the first reported work which incorporates tunability in a fully digital TRNG.

2) We analyze the modified proposed architecture mathematically and experimentally.

3) Our experimental results strongly support the mathematical model proposed. The proposed TRNG has low hardware overhead, and the random bitstreams derived from the proposed TRNG passes all tests in the NIST statistical test suite.

II.LITERATURE REVIEW

From the rigorous review of related work and published literature, it is observed that many researchers have designed random number generation by applying different techniques. Researchers have undertaken different systems, processes or phenomena with regard to design and analyze RNG content and attempted to find the unknown parameters. A TRNG is an algorithm for generating a sequence of numbers that approximates the properties of random numbers. These sequences are not truly random. Although sequences that are closer to truly random can be generated using hardware TRNGs, pseudorandom numbers are important in practice for simulations (e.g., of physical systems with the Monte Carlo method), and are important in the practice of cryptography. Ray C. C. Cheung, Dong-U Lee, John D. Villasenor [1], presented an automated methodology for producing hardware-based TRNG designs for arbitrary distributions using the inverse cumulative distribution function (ICDF).

The ICDF is evaluated via piecewise polynomial approximation with a hierarchical segmentation scheme that involves uniform segments and segments with size varying by powers of two which can adapt to local function nonlinearities. Analytical error analysis is used to guarantee accuracy to one unit in the last place (ulp). Compact and efficient RNGs that can reach arbitrary multiples of the standard deviation can be generated. For instance, a Gaussian RNG based on our approach for a Xilinx Virtex-4 XC4VLX100-12 field programmable gate array produces 16-bit random samples up to 8.2delta. It occupies 487 slices, 2 block-RAMs, and 2 DSP-blocks. The design is capable of running at 371 MHz and generates one sample every clock cycle. The designs are capable of generating random numbers from arbitrary distributions provided that the ICDFs is known. GU Xiao-chen, ZHANG Min-xuan [2] presented ―Uniform TRNG using Leap-Ahead LFSR Architecture‖. Introducing a new kind of URNG using Leap-Ahead LFSR Architecture which could generate an mbits random number per cycle using only one LFSR.

2.1 Single Phase FSTR-TRNG Model

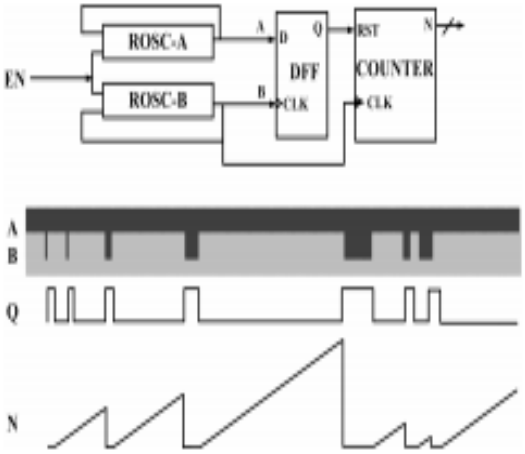


Fig. 1: Architecture of single-phase FSTR–TRNG [3].

The FSTR-TRNG circuit [3] is a fully-digital TRNG, which relies on jitter extraction by the FSTR mechanism, originally implemented as a 65-nm CMOS ASIC. The structure and working of the (single phase) FSTR-TRNG can be summarized as follows, in conjunction with Fig. 1:

1) The circuit consists of two quasi-identical ring oscillators (let us term them as ROSCA and ROSCB), with similar construction and placement. Due to inherent physical randomness originating from process variation effects associated with deep sub-micron CMOS manufacturing, one of the oscillators (say, ROSCA) oscillates slightly faster than the other oscillator (ROSCB). In addition, the authors [3] proposed to employ trimming capacitors to further tune the oscillator output frequencies.

III. PROPOSED SYSTEM

Random numbers used for various purposes in information security systems are essential for secret key generations in symmetric key cryptography systems, private and public keys in public key cryptography systems, and digital signatures and authentication protocols. Random number generators are divided into two groups: pseudo-random number generators (PRNGs) and true random number generators (TRNGs). PRNGs have high throughput but do not meet the unpredictability of random numbers because it is based on deterministic algorithms. TRNGs produce random numbers using randomness generated from physical noise sources such as radiation, thermal noise, jitter, ring oscillator. TRNG usually consists of three blocks; an entropy source, an entropy extraction circuit, and a post processing circuit. Entropy source is responsible for generating entropy from the randomness in physical processes. Many techniques have been reported, including ring oscillator, phase

locked loop (PLL), cellular automata, and crosstalk for use as entropy sources. Entropy extraction circuit should be designed to obtain as much entropy as possible from the entropy source. Post-processing circuits are used to hide defects in entropy sources and entropy extraction circuits, or to provide tolerances in the presence of environmental changes and modulation. Postprocessing circuits include Von Neumann correction, linear feedback shift register (LFSR), and XOR reduction [1].

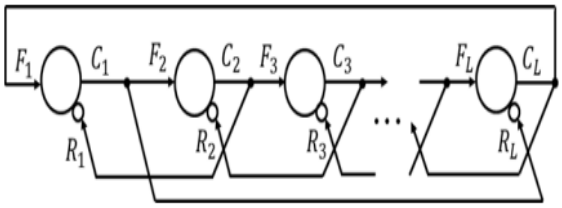


Figure 1. General structure of self-timed ring

3.1. SELF-TIMED RING

A. Self-timed ring structure

Figure 1 depicts the general structure of a self-timed ring (STR) based on a micro-pipeline and two-phase handshake protocol [2]. The ring stage of STR consists of Muller gate and inverter. When inputs have the same value, the output ے retains its previous value. When the values of the two inputs ے and ؛ are different, the value of input ے is exported to the output ے .Input ے is the output of the previous ring stage and input ؛ is the output of the next ring stage. The outputs of the previous and next ring stages are used to make the output of the current ring stage.

B. Token and Bubble [3]

Token and bubble are information that indicates the correlation between the current ring stage and the next ring stage. If output ے □of the current ring stage and output ے□ ◌ाि of the next ring stage are different as in equation (1), the current ring stage contains a token. If the two outputs ے □and ے□ ◌ाि have the same value as in equation (2), the current ring stage contains a bubble. Tokens and bubbles also affect the conditions under which the STR oscillates, with the following conditions: ①ఘ ≤ 1

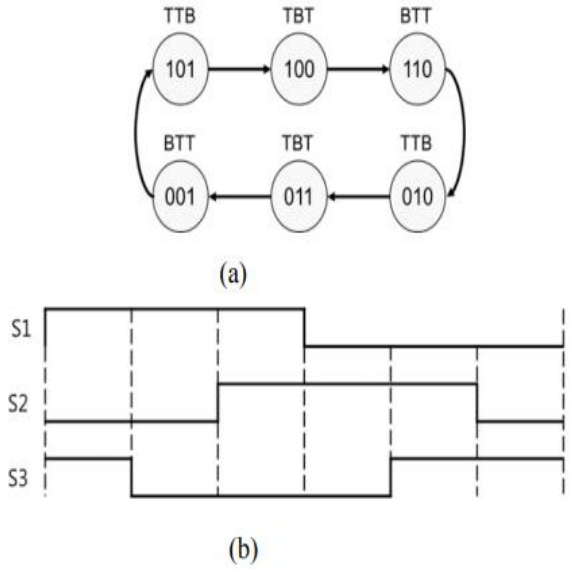


Figure 2. (a) State transition graph of a 3-stage ring with 2 tokens (b) Timing diagram of a 3-stage ring with 2 tokens

The propagation path of data is determined by the correlation between token and bubble. If the current ring stage contains a token and the next ring stage contains a bubble, ے □becomes ے□ ◌ाि and the token and bubble are exchanged with each other. Figure 2-(a) shows the data propagation path of a 3-stage STR with two tokens. Figure 2-(b) shows that the output of each stage is oscillating. C. Oscillation mode of the self-timed ring The oscillation modes of the STR have a burst mode and an evenly-space mode, which are affected by the Charlie effect. The behavior of the STR for each mode is depicted in Figure 3. The Charlie effect is a phenomenon in which the time difference between inputs affects the delay of the Muller gate. The shorter the time difference, the longer the propagation delay. When the two inputs are entered at a short time interval, the delay on the ring stage pushes the outputs of the ring stage out of each other, causing the STR to operate in an evenly-space mode. The oscillation mode and operating point of the STR depend on the design parameters, and the conditions for operating in evenlyspace mode are as shown in equation (3), he denote static forward delay and static reverse delay, respectively.

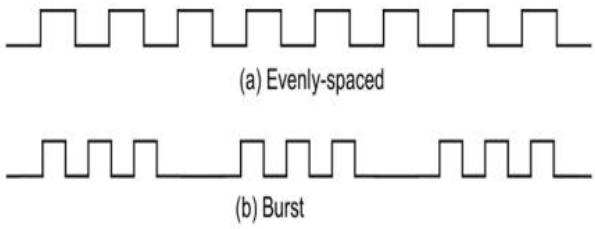


Figure 3. Evenly-spaced and burst propagation modes in self-time

JES
Journal of Engineering Sciences

Vol 13, Issue 03, MARCH /2022

ISSN NO:0377-9254

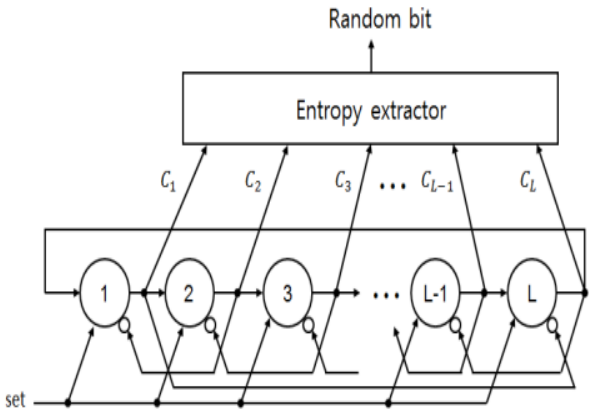


Figure 4. Architecture of the FSTR-TRNG

TRNG BASED ON STR WITH FEEDBACK STRUCTURE

The TRNG based on STR with feedback structure (FSTRTRNG has the structure as shown in Figure 4. The STR used as an entropy source was designed with a micro-pipeline structure. Each 4-input ring stage was implemented with a look-up table (LUT) instead of logic gate. The number of ring stages and the óò/ó ratio are variably determined depending on the device environment or entropy extraction circuit. In our design, parameters such asó ò/ó ratio, and oscillation mode were determined with reference to [4]. Entropy extraction circuit consists of a serial-input parallel-output (SIPO) shift register, multiplexers, D-type flip-flops, and XOR gates, as shown in Figure 5. The outputs of the ring stages, which are sampled by D-type flip-flops. The sampling takes place at the jitter boundary of STR output transition. The initial value of the SIPO shift register is determined to be 2 − 1 to set the output of the multiplexers to the initial values of ring stages. The final output of the FSTR-TRNG is applied to the input of SIPO shift register. If this value has randomness, the final output values during the -clock cycles are random. The outputs of SIPO shift register are used as enable signals for the multiplexers. The outputs of the multiplexers are used as inputs to the XOR gate to produce the final output value, and they are XORed with the output of the ring stage to be used as inputs to the multiplexer. The number of ring stages of the FSTR-TRNG was determined to be a multiple of eleven, and it was set to operate in evenly-spaced mode.

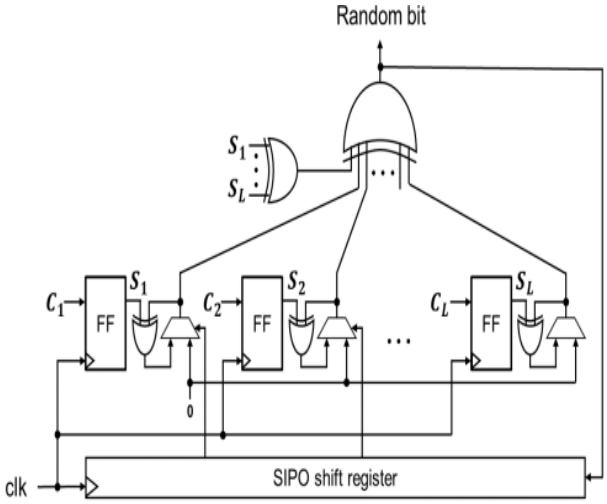


Figure 5. Proposed entropy extractor with feedback structure

IV. EXTENSION SYSTEM

4.1 Design Overview

Tunability is established by setting the DCM parameters on–the– fly using DPR capabilities using DRP ports. This capability provides the design greater flexibility than the ring oscillator-based FSTR-TRNG. The difference in the frequencies of the two generated clock signals is captured using a DFF. The DFF sets when the faster oscillator completes one cycle more than the slower one (at the beat frequency interval). A counter is driven by one of the generated clock signals, and is reset when the DFF is set. Effectively, the counter increases the throughput of the generated random numbers. The last three LSBs of the maximum count values reached by the count were found to show good randomness properties.

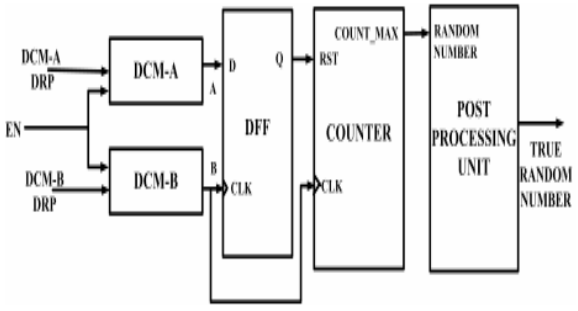


Fig. 1: Overall architecture of proposed Digital Clock Manager based tunable FSTR–TRNG

Fig. 1 shows the overall architecture of the proposed TRNG. In place of two ring oscillators, two DCM modules generate the oscillation waveforms. The

DCM primitives are parameterized to generate slightly different frequencies, by adjusting two design parameters M (Multiplication Factor) and D (Division Factor). In the proposed design, the source of randomness is the jitter presented in the DCM circuitry. The DCM modules allow greater designer control over the clock waveforms, and their usage eliminates the need for initial calibration [3].

Additionally, we have a simple post-processing unit using a Von Neumann Corrector (VNC) [5] to eliminate any biasing in the generated random bits. VNC is a well-known lowoverhead scheme to eliminate bias from a random bitstream. In this scheme, any input bit "00" or "11" pattern is eliminated; otherwise, if the input bit pattern is "01" or "10", only the first bit is retained. The last three LSBs of the generated random number is passed through the VNC. The VNC improves the statistical qualities at the cost of slight decrease in throughput.

The initial value of the LFSR is called the seed, and because the operation of the register is deterministic, the stream of values produced by the register is completely determined by its current (or previous) state. Likewise, because the register has a finite number of possible states, it must eventually enter a repeating cycle. However, an LFSR with a wellchosen feedback function can produce a sequence of bits which appears random and which has a very long cycle. Applications of LFSRs include generating pseudo-random numbers, pseudo-noise sequences, fast digital counters, and whitening sequences. Both hardware and software implementations of LFSRs are common.

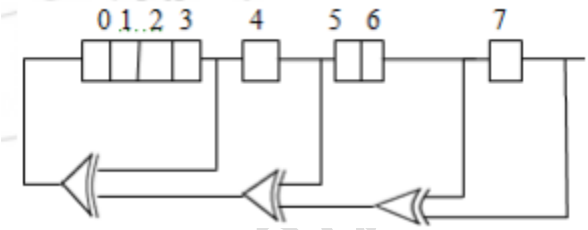


Fig 4 Galois LFSRs

The feedback tap numbers correspond to a primitive polynomial in the table so the register cycles through the maximum number of 256 states excluding the all-zeroes state. The bit positions that affect the next state are called the taps. In the diagram the taps are [7,6,4,3]. The rightmost bit of the LFSR is called the output bit. The taps are XOR'd sequentially with the output bit and then fed back into the leftmost bit. The sequence of bits in the rightmost position is called the output stream. The arrangement of taps for feedback in an LFSR can be expressed in finite field arithmetic as a polynomial mod 2. This means that the coefficients of the polynomial must be 1's or 0's.

This is called the feedback polynomial or reciprocal characteristic and 3rd bits (as shown), the feedback polynomial is X 7 +X6 +X4 +X3 +1 The 'one' in the polynomial does not correspond to a tap – it corresponds to the input to the first bit (i.e. x 0, which is equivalent to 1). The powers of the terms represent the tapped bits, counting from the left. The first and last bits are always connected as an input and output tap respectively. The LFSR is maximal-length if and only if the corresponding feedback polynomial is primitive. This means that the following conditions are necessary (but not sufficient): The number of taps should be even. The set of taps – taken all together, not pairwise (i.e. as pairs of elements) – must be relatively prime. In other words, there must be no divisor other than 1 common to all taps.

4.2 Galois LFSRs:

Named after the French mathematician Évariste Galois, an LFSR in Galois configuration, which is also known as modular, internal XORs as well as one-to-many LFSR, is an alternate structure that can generate the same output stream as a conventional LFSR (but offset in time). In the Galois configuration, when the system is clocked, bits that are not taps are shifted one position to the right unchanged. The taps, on the other hand, are XOR'd with the output bit before they are stored in the next position. The new output bit is the next input bit. The effect of this is that when the output bit is zero all the bits in the register shift to the right unchanged, and the input bit becomes zero. When the output bit is one, the bits in the tap positions all flip (if they are 0, they become 1, and if they are 1, they become 0), and then the entire register is shifted to the right and the input bit becomes 1.

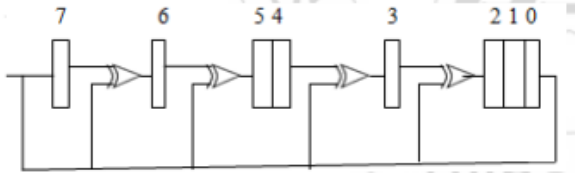


Fig 5 Galois register LFSRs

To generate the same output stream, the order of the taps is the counterpart (see above) of the order for the conventional LFSR, otherwise the stream will be in reverse. Note that the internal state of the LFSR is not necessarily the same.

The Galois register shown has the same output stream as the Fibonacci register in the first section. A time offset exists between the streams, so a different start point will be needed to get the same output each cycle. Galois LFSRs do not concatenate every tap to produce the new input (the XOR'ing is done within the LFSR and no XOR gates are run in serial, therefore the propagation times are reduced to that of one XOR rather than a whole chain), thus it is possible for each tap to be computed in parallel, increasing the speed of execution. In a software

implementation of an LFSR, the Galois form is more efficient as the XOR operations can be implemented a word at a time: only the output bit must be examined individually.

4.3Tuning Circuitry

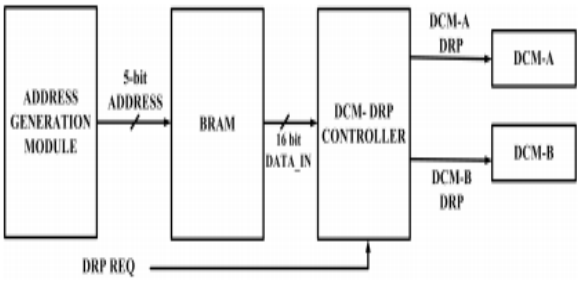


Fig. 6: Architecture of tuning circuity.

The architecture of the tuning circuitry is shown in Fig. 3. The target clock frequency is determined by the set of parameter values actually selected. The random values reached by the counter, as well as the jitter are related to the chosen parameters M and D (details are discussed in Section IV). This makes it possible to tune the proposed TRNG using the predetermined stored M and D values. As unrestricted DPR has been shown to be a potential threat to the circuit [6], the safe operational value combinations of the D and M parameters for each DCM are predetermined during the design time, and stored on an on-chip Block RAM (BRAM) memory block in the FPGA.

There are actually two different options for the clock generators – one can use the Phase Locked Loop (PLL) hard macros available on Xilinx FPGAs, or the DCMs. We next describe analytical and experimental results which compelled us to choose DCM in favor of the PLL modules for clock waveform generation.

V. SIMULATION RESULTS

5. 1 power report

2. Summary

2.1. On-Chip Power Summary

On-Chip Power Summary				
On-Chip	Power (mW)	Used	Available	Utilization (%)
Clocks	1.30	3	---	---
Logic	0.00	10	11776	0
Signals	0.00	20	---	---
IOs	0.00	20	372	5
Quiescent	31.52			
Total	32.83			

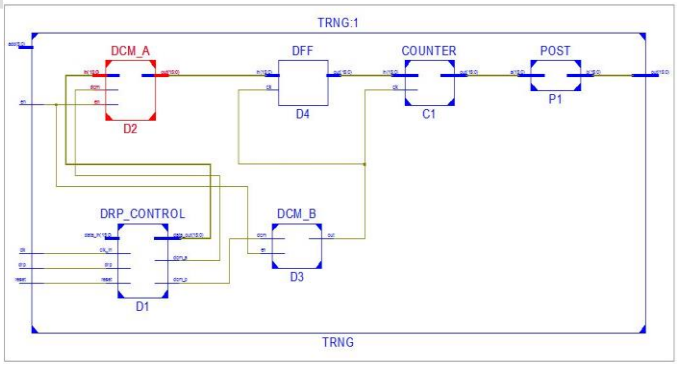
5.2 Timing report

```
================================================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'clk_out'
  Total number of paths / destination ports: 9 / 7
--------------------------------------------------------------------------------
Offset:              6.769ns (Levels of Logic = 2)
  Source:            C1/out_7 (FF)
  Destination:       out<8> (PAD)
  Source Clock:      clk_out rising

  Data Path: C1/out_7 to out<8>
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FD:C->Q               2   0.591   0.590  C1/out_7 (C1/out_7)
     LUT2:I0->O            1   0.648   0.420  P1/Mxor_b<8>_Result1 (out_8_OBUF)
     OBUF:I->O                 4.520          out_8_OBUF (out<8>)
    ----------------------------------------
    Total                      6.769ns (5.759ns logic, 1.010ns route)
                                       (85.1% logic, 14.9% route)

================================================================================
```

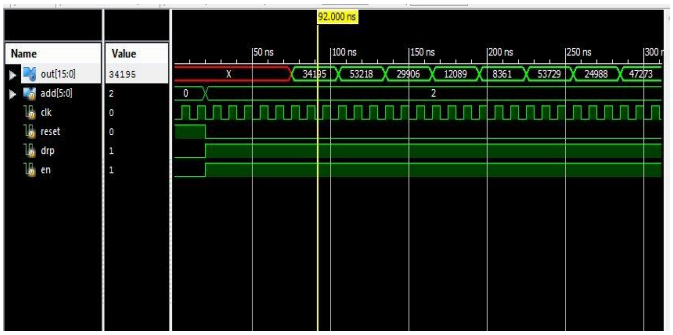
5.3 Design Summary

Device Utilization Summary (estimated values)			[-]
Logic Utilization	Used	Available	Utilization
Number of Slices	9	5888	0%
Number of Slice Flip Flops	10	11776	0%
Number of 4 input LUTs	17	11776	0%
Number of bonded IOBs	20	372	5%
Number of GCLKs	1	24	4%

5.4 RTL Schematic



5.5 SIMULATION RESULTS



VI. . CONCLUSION

We have presented an improved fully digital tunable TRNG for FPGA based applications, based on the principle of FSTR and clock jitter, and with in-built error correction capabilities. The TRNG utilizes this tunability feature for determining the degree of randomness, thus providing a high degree of flexibility for various applications.

FUTURE SCOPE

Presently, we are dealing with the 4-digit numeric OTP system in this project. So, in order to improve the security, we need to develop an Alpha numeric (symbol base) OTP systems.

REFERENCES

[1] D. B. Thomas and W. Luk, "The LUT-SR Family of Uniform TRNGs for FPGA Architectures," IEEE Transactions on Very Large-Scale Integration (VLSI) Systems, March 2012.
[2] D. B. Thomas and W. Luk, "FPGA-optimized uniform TRNGs using lot and shift registers," in Proc. Int. Conf. Field Program. Logic Appl., 2010, pp. 77–82.
[3] D. B. Thomas and W. Luk, "FPGA- optimized high - quality uniform TRNGs," in Proc. Field Program. Logic Appl. Int.Conf., 2008, pp. 235-244.
[4] D. B. Thomas and W. Luk, "High quality uniform random number generation using LUT optimized state-transition matrices," J. VLSI Signal Process., vol. 47, no. 1, pp. 77–92, 2007.
[5] F. Panneton, P. L'Ecuyer, and M. Matsumoto, "Improved long period generators based on linear recurrences modulo 2," ACM Trans. Math. Software, vol. 32, no. 1, pp. 1–16, 2006.
[6] P. L'Ecuyer, "Tables of maximally equidistributed combined LFSR generators," Math.Comput., vol. 68, no. 225, pp. 261– 269, 1999.
[7] M. Saito and M. Matsumoto, "SIMD-oriented fast mersenne twister: A 128-bit TRNG," in MonteCarlo and Quasi- Monte Carlo Methods. NewYork: SpringerVerlag, 2006, pp. 607–622.
[8] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623- dimensionally equidistributed uniform pseudo-TRNG," ACM Trans. Modeling Comput. Simulate., vol.8, no. 1, pp. 3–30, Jan. 1998.
[9] F. Panneton, P. L'Ecuyer, and M. Matsumoto, "Improved longperiod generators based on linear recurrences modulo 2," ACM Trans. Math. Software, vol. 32, no. 1, pp. 1–16, 2006.
[10] K. H. Tsoi, K. H. Leung, and P. H. W. Leong, "Compact FPGAbased TRNGs", in IEEE Symposium on FPGAs for Custom Computing Machines, IEEE Computer Society, Washington, DC, 2003, p. 51.