A Reprogrammable FPGA-Based ATM Traffic Generator

Pong P. Chu[1] and Brian Frantz
Department of Electrical Engineering
Cleveland State University, Cleveland, Ohio 44115

Abstract

Markov modulated process is widely used to model ATM (Asynchronous Transmission Mode) traffic sources. We develop an approximation algorithm and design an FPGA (Field Programmable Gate Array) based circuit to generate cell trigger emulating this process. The design takes advantage of the reprogrammability of SRAM or E²PROM FPGA. Instead of storing the process' various parameters in registers, our design "hardwires" these parameters into the circuitry and greatly simplifies the circuit complexity. Our testing circuit can be fitted into a moderate-sized FPGA chip and can generate up to 12M triggers per second.

1. Introduction

Future communication is expected to employ ATM based networks that supports a wide variety of services [10]. In order to test and evaluate the performance of the network and the switch, it is necessary to generate cell arrival closely resembling the real traffic [1,12]. Recording and reproducing the real traffic can be difficult and expensive because the process requires large storage space and high processing speed. The alternative is to use a concise mathematical model, such as Markov modulated process, to imitate the behavior of the traffic [4,6]. Because of high processing speed of ATM switch, custom hardware is needed to obtain the required cell arrival rate [7,8,12].

FPGA provides a cost-effective way to design medium complexity custom circuits [5]. Some devices, such as SRAM and E²PROM based FPGAs, can easily be reconfigured by loading the new fuse/memory map to the chip and even provide in-system reprogrammability. In other words, an FPGA chip can be thought of as a generic circuit and its functionality can be altered and "reprogrammed" by loading different files.

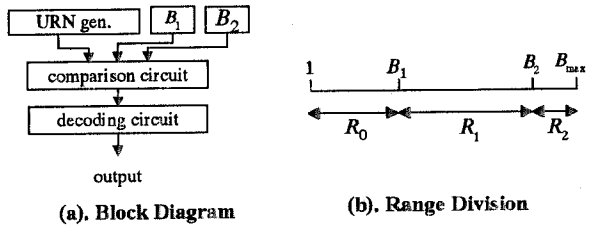
In this article, we describe the design and implementation of a modulated random sample generator based on reprogrammable FPGAs, and use the widely used Markov Modulated Bernoulli Process (MMBP) to demonstrate the design procedure. The remaining article is organized as follows: Section 2 describes the implementation of a hardware random sample generator; Section 3 discusses the approximation algorithm that converts the exponential distribution to a discrete distribution; Section 4 shows the data-path and control-path design of a 2-state MMBP generator; and the last section summarizes the study.

2. Hardware Random Sample Generator

Random sample generator is a circuit that generates random samples from a given probability distribution. Because of the system clock and the finite register length, only samples from discrete random variables can be generated and implemented by digital hardware. The implementation includes an N-bit Uniform Random Number (URN) generator, a comparison circuit, and a decoding circuit. The implementation can be best explained by an example. Consider a random variable n with a probability distribution given by

$$f(n) = \begin{cases} a & \text{if } n = 4 \\ b & \text{if } n = 5 \\ c & \text{if } n = 6 \\ 0 & \text{otherwise} \end{cases} \quad \text{where } a + b + c = 1$$

The random sample generator for this particular distribution is shown in Figure 1(a). An N-bit URN generator generates a uniformly distributed random number between 1 and B_{max}, where $B_{max} = 2^N - 1$. We can divide the interval [1, B_{max}] into three regions, R_0, R_1 and R_2, whose lengths are $\lfloor aB_{max} \rfloor$, $\lfloor bB_{max} \rfloor$ and $\lfloor cB_{max} \rfloor$ respectively, as shown in Figure 1(b). The boundary points of the three regions, B_1 and B_2 are $\lfloor aB_{max} \rfloor$ and $\lfloor aB_{max} \rfloor + \lfloor bB_{max} \rfloor$ respectively. The operation of the circuit is quite simple. The URN genera-



(a). Block Diagram (b). Range Division

Figure 1. Diagram of Random Sample Generator

[1] The work was partially supported by NASA Grant NAG3-1379

tor produces a random number and passes it to the comparison circuit. The comparison circuit compares the number with the two boundary points (which are held in two registers), determines the region the number resides, and outputs the region number (i.e., 0, 1 or 2). Since the region number is different from the desired random value, we need a decoding circuit to convert the region number to the corresponding value (i.e., 4, 5 or 6).

The URN generator and the comparison circuit are the two major parts of the implementation. True randomness is hard to obtain. However, a pseudo URN generator can be easily implemented by using an LFSR (Linear Feedback Shift Register), which is composed of a register and exclusive-OR gates [11]. With an N-bit shift register, it can generate a pseudo random sequence of $2^N - 1$ patterns. The length of the sequence can extended by increasing the size of the shift register. Because the length grows exponentially with N, the pseudo URN generator can easily accommodate ATM testing requirement. The comparison circuit is more difficult to design because of the large number of input signals. A circuit with N-bit URN generator and m regions will have mN input signals (1 N-bit random number input from the URN generator and m-1 N-bit boundary inputs from the registers), and $\lceil \log_2 m \rceil$ output signals. Because N is normally chosen between 30 and 50, the number of input signals (mN) can easily reach several hundreds and even thousands. Thus, if m is large, multi-level comparison circuit will be needed for implementation, which requires significant amount of hardware resources and will increase the overall propagation delay. The comparison circuit can be greatly simplified if the boundary points are constants. In this case, the registers, which hold the boundary points, can be removed and replaced with constant logic values. A closer observation shows that the comparison circuit now can be represented by a function with N input variables. The other m-1 N-bit boundary points are "hardwired" into the new logic function and do not need to be treated as the input signals. Thus, the comparison circuit now becomes an N-input, $\lceil \log_2 m \rceil$-output combinational circuit. Since N is limited between 30 and 50, it can be optimized and implemented by a two-level AND-OR PLA typed circuit.

The simplified constant approach is not feasible for conventional ASIC technology, such as the standard cell design, since only one set of values is built into the circuit and its usage is extremely limited. However, reprogrammable FPGA provides a way to utilize this approach. For a given set of values, we can set proper boundary points in the design file, synthesize the circuit, and construct the fuse/memory map. A custom chip can be obtained by loading the fuse/memory map to the FPGA chip. When a different set of values is desired, we can simply edit the boundary points, repeat the synthesis process and loads the new fuse/memory map. In other words, instead of loading boundary points to the registers as in the original implementation, the FPGA based design loads the new fuse/memory map file. This method can be incorporated into the E^2PROM and SRAM based FPGAs since they can be easily modified and reprogrammed.

3. Bernoulli and Exponential Sample Generators

The implementation of an MMBP generator consists of Bernoulli sample generators and exponential sample generators, which are used to emulate the cell arrival process and duration distribution respectively [7]. Bernoulli distribution is characterized by the parameter p and is defined as

$$f_b(n) = \begin{cases} p & \text{if } n = 1 \text{ (a cell is generated)} \\ 1-p & \text{if } n = 0 \text{ (a cell is not generated)} \end{cases}$$

The boundary point of this distribution can be determined by the procedure outlined in section 2.

Because of the discrete nature of the digital circuits, it is not possible to generate a continuous number, as required by the exponential sample generator. Thus, we use the geometrical distribution to approximate the exponential distribution. The geometrical distribution is characterized by a parameter q and is defined as

$$f_e(n) = \begin{cases} (1-q)q^{n-1} & \text{if } n \geq 1 \\ 0 & \text{otherwise} \end{cases}$$

This distribution has an infinite tail, which requires infinite comparison regions and thus cannot be implemented by a physical circuit. Thus, it needs to be approximated by a truncated geometrical distribution, which is defined as

$$f_g(n) = \begin{cases} \frac{1}{1-q^L}(1-q)q^{n-1} & \text{if } L \geq n \geq 1 \\ 0 & \text{otherwise} \end{cases}$$

L is a constant and represents the largest sample value allowed in this distribution.

Determining the boundary points of the truncated geometrical distribution is less straightforward. For a given circuit, the values of N (the number of bits of the URN generator) and m (the number of regions of the comparison circuit) are normally fixed. They are determined by the available hardware resource and do not vary with the value of q. Both N and m impose some constraints on the choice of boundary points. N sets a limit on the minimal "resolution" of the probability distribution since the smallest possibility could not exceed $\frac{1}{2^N - 1}$. m sets an upper limit on the maximal number of possible outputs of the random sample generator. Because of these constraints, $f_g(n)$ cannot be

directly used to obtain the boundary points. An algorithm is developed to determine the boundary points for a circuit with an N-bit URN generator and m comparison regions:

1. Solve the equation $\frac{1}{1-q^M}(1-q)q^{M-1} = \frac{1}{2^N-1}$ to obtain M, the largest sample value allowed.
2. If $M \le m$, list $f_g(n)$, $1 \le n \le M$, and determine the boundary points.
3. If $M > m$, obtain $f_g(n)$, $1 \le n \le M$, cluster $\left\lfloor \frac{M}{m} \right\rfloor$ adjacent terms and sum these terms, list the results, and determine the boundary points.

Step 1 determines the minimal resolution limit imposed by the N-bit URN generator. After solving the equation, we can obtain the maximal interval, M, of the truncated geometrical distribution. If M is smaller than or equal to m, as in step 2, we can simply list the discrete probability distribution (with $L=M$) and determine the boundary points. If M is larger than m, as in step 3, the desired truncated distribution cannot be implemented because the number of the distinguished outputs from the comparison circuit is limited. One possible solution is to use a truncated geometrical distribution with $L=m$. However, when q is large, M becomes much larger than m and this approximation is not accurate. The resulting distribution becomes relatively flat, with possible sample values between 1 and m. This cannot faithfully reflect the nature of geometrical distribution, which normally contains a long sharp tail. To overcome this, we use a clustering process shown in step 3 [9]. This process essentially "compresses" the original probability distribution to a simpler one.

4. A 2-State MMPP Sample Generator

A 2-state MMBP sample generator has been implemented for testing. Although the design is limited to 2-state MMBP generator, the same principle can be applied to implement multiple-state MMBP generator.

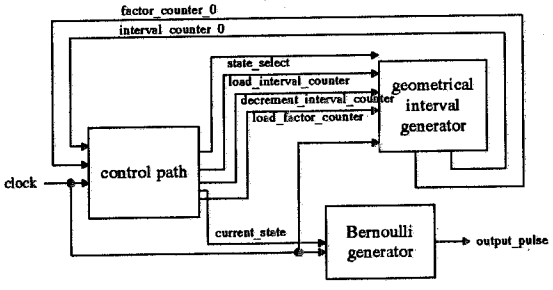


Figure 2. Top-Level Diagram

The block diagram of the 2-state MMBP traffic generator is shown in Figure 2. It can be divided into data path and control path. The data path includes two parts, the Bernoulli generator and the geometrical interval generator. The control path keeps track of the state of MMBP and generates proper control signals to load and decrement the counters inside the geometrical interval generator.

The Bernoulli generator is shown in Figure 3. It includes a URN generator and Bernoulli comparison circuits for states 0 and 1. The current_state signal from control path indicates the current state of MMBP and is used to select an input for the multiplexer. The geometrical interval generator is shown in Figure 3. It includes a URN generator, geometrical comparing circuits for states 0 and 1, an interval counter and a factor counter. The interval counter stores the output of the geometrical comparison circuits and the factor counter stores the cluster factor. The two counters work as a multiplication circuit and determine the overall length of an interval. The counters load initial values when the state machine transits to a new state, and then start to count down. When both of them reach 0, the URN generator will advance one step and obtain a new random number.

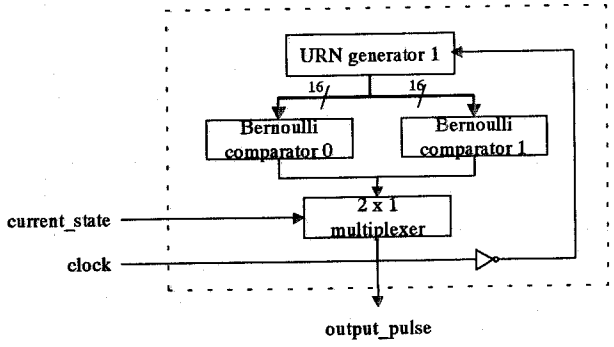


Figure 3. Bernoulli Generator

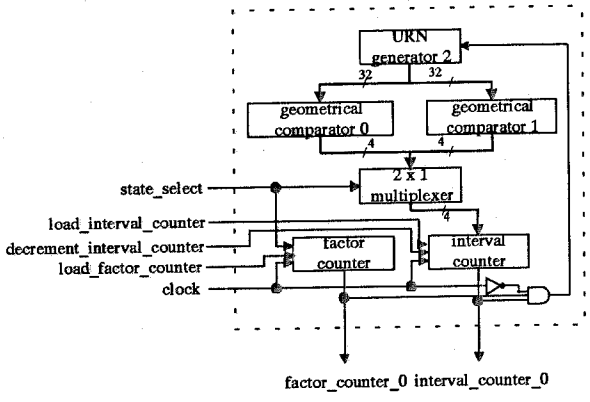


Figure 4. Geometric Generator

The state diagram of the control path is shown in Figure 5. The control path accepts two status signals, interval_counter_0 and factor_counter_0, and issues the state_select, load_factor_counter, and load_interval_counter signals to control the operation of the counters. A Moore output, current_state, indicates the current state and is used to select an output from the two Bernoulli generators.

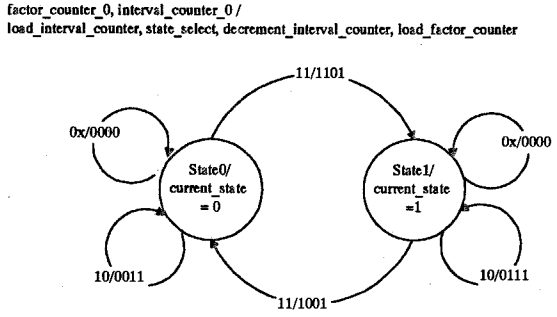


Figure 5. State Diagram of the Control Path

There are four parameters in the 2-state MMBP, for the Bernoulli processes in states 0 and 1 and the geometrical processes in states 0 and 1 respectively. The boundary points for these processes are calculated and hardwired into the Bernoulli comparator 0, Bernoulli comparator 1, geometrical comparator 0 and geometrical comparator 1 blocks respectively. The two multiplication factors (used as the initial values for the factor counters) are hardwired into the factor counter block. Only the values that cannot be pre-determined are stored in the registers (counters). A testing circuit is implemented by using Altera MAX Plus II[2] development system and FPGA chips [2]. The modules are implemented by using AHDL (Altera Hardware Description Language) [3] and the boundary values of various comparison circuits can be easily modified. It is possible to use another software shell to calculate these values, generate the AHDL file, and automate the entire process. In this experiment, a 16-bit URN generator is used in the Bernoulli generator and a 32-bit URN generator is used in the geometrical interval generator. The geometrical generator is divided into 10 regions. The entire circuit can be fitted into one EPM7128 chip, a moderate-sized FPGA chip with 5000 equivalent NAND gates. Because the boundary points change with the values of the four parameters, the resulting comparison circuits are different and may requires different amount of hardware resource. However, our experiences show that the variation is relatively small for different values. This may be due to the fact that there are 608 bits (9*32 bits for each geometrical comparator and 16 bits for each Bernoulli comparator) being compared and the difference is averaged out. The resulting circuit can support a clock rate up to about 12M Hz (i.e., the circuit can generate 12M triggers per second). Since an ATM cell contains 53 bytes, the circuit can emulate traffic arrival with a rate of 4.5G bits per second.

[2] Altera, MAX PLUS II, AHDL and EPM7128 are the registered trademarks of Altera Corporation.

5. Summary

We develop an approximation algorithm and design a custom circuit to emulate the MMBP traffic arrivals. The design is based on reprogrammable FPGA chips. We take advantage of the reprogrammability and hardwire various parameters of MMBP into the circuitry. This approach greatly simplifies the complexity of the comparison circuits and makes it possible to accommodate the entire circuit in a single FPGA chip. Our testing circuit can be fitted into a moderate-sized FPGA chip and can emulate a traffic source with a rate of 12M cells per second.

Bibliography

[1] O. Aboul-Magd and M. Wernik, "Traffic Experimentation in ATM Testbed," *GLOBCOM*, 1990, pp. 1445-1449.

[2] Altera, *Altera MAX Plus II Getting Started*, Altera, San Jose, 1994.

[3] Altera, *Altera Max Plus II AHDL*, Altera, San Jose, 1994

[4] J. J. Bae and T. Suda, "Survey of Traffic Control Schemes and Protocols in ATM Networks," *Proceedings of IEEE*, 1991, 79(2), pp. 170-189.

[5] S. D. Brown et al., *Field-Programmable Gate Arrays*, Kluwer, 1992.

[6] H. Brunel and B. G. Kim, *Discrete-Time Models for Communication System Including ATM*, Kluwer, 1993.

[7] P. Chu, "ATM Burst Traffic Generator," *Great Lakes Symposium on VLSI Design, 1995*.

[8] N. Endo et al., "Shared Buffer Memory Switch for an ATM Exchange," *IEEE Trans. on Communications*, January, 1993, vol. 41, no. 1, pp. 237--245.

[9] B. D. Frantz, *A Reprogrammable FPGA-Based ATM Traffic Generator which Emulates an MMPP Model, Master Thesis,* Cleveland State University, 1994.

[10] R. Handel and M. N. Huber, *Integrated Broadband Network*, Addison-Wesley Publishing, 1992.

[11] J. McClusky, *Logic Design Principles,* Prentice-Hall, Englewood Cliffs, 1986.

[12] M. J. Riezenman ed., "ATM Testing," *IEEE Spectrum*, June, 1994, vol 31, no 6, pp. 19-32.