Hardware Channel Model for Ultra Wideband Systems

Wen-Chih Kan and Gerald E. Sobelman

Department of Electrical and Computer Engineering, University of Minnesota
Minneapolis, MN 55455 USA
kanx0008@umn.edu
sobelman@umn.edu

***Abstract*—We present a digital hardware model for ultra wideband channels. The system runs at 80 MHz on a Xilinx Virtex-4 xc4vsx35 FPGA. High-speed arithmetic operations including division, square root, powering and normal random number generator are analyzed and developed for use as basic components in the channel emulator. The design flow is based on Matlab Simulink as the model builder, followed by Xilinx System Generator to transform the Simulink model into a VHDL description which can be synthesized and mapped onto the FPGA device. Speed and area results are given for the synthesized designs.**

I. INTRODUCTION

Many hardware designs make use of FPGAs for rapid prototyping and functional evaluation. In a communications system, it is also important to account for the effects of the channel in order to correctly characterize the overall system performance. Ideally, the channel model can be described as a digitally equivalent circuit which can then be mapped onto hardware to allow for real-time testing of both a transceiver and the channel together. Ultra wideband (UWB) channel models [1] have been developed as Matlab code and approved by the study group IEEE 802.15.SG3a as a method to evaluate the physical layer performance. The complete functional block diagram can be drawn as in Fig. 1. Reference [1] also refers to [2] and [3] which deal with the time variability of UWB channels.

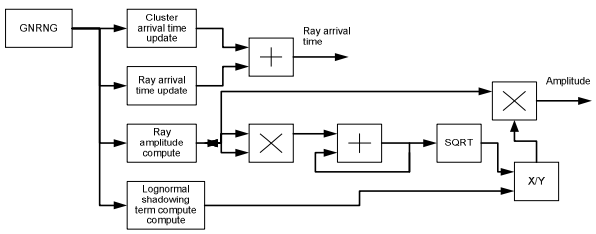

Fig. 1 Channel model architecture

We have used Matlab Simulink v6.1 to implement our UWB channel model emulator. Xilinx System Generator v7.1 was used to generate the VHDL model which was then used in the Xilinx ISE 7.1i for mapping and place and route. The device used for the implementation was a Xilinx Virtex-4 xc4vsx35. The numbers of slices required are obtained from ISE's Map Report. The maximum frequency values are from ISE's Post-Place & Route (pPnR) Static Timing Report. The word length is assumed to be 24 bits throughout the implementation. To generate this channel model on an FPGA, some basic digital arithmetic operators have to be designed and they must run at a very high speed in order for the channel model to meet the throughput requirements specified in [2] and [3].

Required digital arithmetic functions include a Gaussian noise random number (GNRN) generator (GNRNG), a division operation, a square-root computation (SQRT) and a base-two powering function (2^x). This paper is organized as follows. Section II. outlines the design of the GNRNG. In Section III., a radix-4 SRT division processor is discussed. The SQRT architecture and implementation is presented in Section IV. The final computational block, 2^x, is described in Section V. Section VI. presents results for the complete channel model. The conclusions are given in Section VII.

II. GAUSSIAN NOISE RANDOM NUMBER GENERATOR

The most often used methods to generate GNRNs are rejection-acceptance method, Box–Muller method, cumulative distribution function conversion method and polar method. Detailed comparison among those methods and their respective drawbacks were discussed in [5]. Speed and the quality of noise samples are the two main concerns in designing a good GNRNG for our digital channel modelling. Since all of the methods mentioned above produce GNRNs by performing operations on uniform variables, throughput and quality of the noise sample are difficult to obtain at the same time. In contrast, Wallace proposes an algorithm using an evolving pool of normal variables to generate additional normal variables, [4] and [5].

The basic architecture of a Wallace GNRNG is shown in Fig. 2 where a uniform random number generator is used to produce addresses to read/write the RAMs which store the transformed GNRNs. The transformation utilizes an addition operation which is followed by a sum-of-squares correction (SOSC). The key aspect, which allows this GNRN to run at high speed is that the Wallace method does not require complex arithmetic operations in order to generate new samples. We have also made use of the fact that two locations of a dual-port RAM can be read at the same time. Hence by adding one more data path in the SOSC, the GNRNG's speed can be doubled. In Table 1 our GNRNG's operating speed from the Xilinx pPnR static timing report is shown. Since our GNRNG produces two samples per clock, our GNRNG

generates 250 million samples per second. Also shown is the logic utilization (LU) reported from Xilinx Map Report.

TABLE I
PPNR TIMING AND LU OF THE WALLACE GAUSSIAN NOISE RANDOM NUMBER GENERATOR

Wallace GNRNG	Max. freq.	Latency	Slices
	125.471MHz	1,051 clocks	1,629

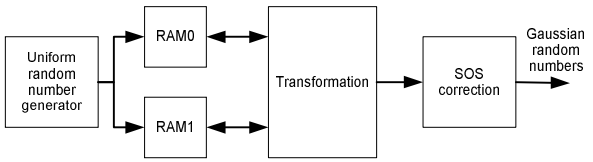

Fig. 2 Basic architecture of Wallace GNRNG

III. RADIX-4 SRT DIVISION IMPLEMENTATION

SRT is a digit-recurrence algorithm used to compute the division, as in (1), and square root.

$$r_{i+1} = br_i - q_{i+1}y, \quad (1)$$

where r_{i+1} is the *(i+1)-th* iterative remainder, b the quotient radix, q_{i+1} the *(i+1)-th* quotient digit, and y the divisor. Two main factors determine the operating speed of a division: the number of recurrences and quotient selection function. Basic SRT division has half or more of its iteration latency spent in quotient selection [12].

Higher radix SRT division reduces the number of iterations but its quotient selection logic becomes quite complex and consumes lots of chip area. On the other hand, lower radix division has simple quotient selection logic but the number of iterations required is large. Reference [7] analyzed several methods to perform the division operation and showed that radix-4 SRT division [6] method provided good execution time improvement with moderate increase in chip area. To design a 16 (or more)-bit iterative SRT division running at more than 100 MHz on FPGAs is non-trivial, if not impossible at the present time. Therefore, a pipelined array architecture is one way to achieve high-speed SRT division on 24-bit numbers, as shown in [8]. We chose to use radix-4 SRT pipelined array division using the digit selection function developed in [9]. To assure at least one digit value can be chosen for current iteration while keeping the new remainder bounded, it is necessary that

$$U_{d-1} \geq L_d, \quad (2)$$

where $[L_d\ U_d]$ is the digit selection interval that if br_i lies in that interval, then the next quotient radix, q_{i+1}, would be d. L_d and U_d are defined below, as in (3) and (4) where r is the redundancy factor. For minimally redundant radix-4 arithmetic, the value of r is $2/3$.

$$L_d = (d - r)y, \quad (3)$$

$$U_d = (d + r)y, \quad (4)$$

The architecture block diagram is provided in Fig. 3. The Xilinx pPnR and LU results are shown in Table 2. Table 3 is the quotient selection table developed by Kornerup for minimally redundant radix-4 SRT division. However, simplification on this selection logic into hard-decision type as described below is needed to increase the operating frequency.

$$q_{i+1} = -2 \text{ if } br_i < -1.5y, \quad (5a)$$

$$q_{i+1} = -1 \text{ if } -1.5y \leq br_i < -0.5y, \quad (5b)$$

$$q_{i+1} = 0 \text{ if } -0.5y \leq br_i < 0.5y, \quad (5c)$$

$$q_{i+1} = 1 \text{ if } 0.5y \leq br_i < 1.5y, \quad (5d)$$

$$q_{i+1} = 2 \text{ if } br_i \geq 1.5y, \quad (5e)$$

From (5a)-(5e), we can observe that the digit selection boundaries computation only involves shift and addition. Furthermore, since the value of y is always in the range of [1, 2), the value of $(1/6)y$ is always in the range of [1/6, 1/3) so that only the most important three fractional bits of br_i are needed to be preserved in the selection function. With this further simplification, the adders used in the selection logic are only required to be 8 bits wide so that it can run at a very high speed.

TABLE II
PPNR TIMING AND LU OF RADIX-4 SRT PIPELINED ARRAY DIVISION

Radix-4 SRT division	Max. freq.	Latency	Slices
	133.156MHz	32 clocks	2,250

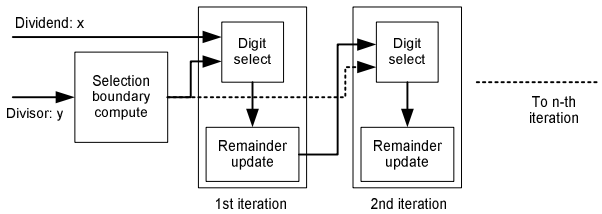

Fig. 3 Radix-4 SRT pipelined array division

TABLE III
QUOTIENT SELECTION TABLE BY KORNERUP, [9]

d =	-2	-1	0	1	2
$L_d(y)$	$-(2\frac{2}{3})y$	$-(1\frac{2}{3})y$	$-(\frac{2}{3})y$	$(\frac{1}{3})y$	$(1\frac{1}{3})y$
$U_d(y)$	$-(1\frac{1}{3})y$	$-(\frac{1}{3})y$	$(\frac{2}{3})y$	$(1\frac{2}{3})y$	$(2\frac{2}{3})y$

IV. SQUARE ROOTING

The recurrence update equation for an SRT-based SQRT function is (6). Since $b^{-i-1} << 1$ when i is large, (6) can be approximated as (7) below.

$$r_{i+1} = br_i - q_{i+1}(2\,q_i + q_{i+1}b^{-i-1}) \quad (6)$$

$$r_{i+1} = br_i - q_{i+1}(2\,q_i) \quad (7)$$

An SRT-based SQRT shares the same architecture as division and thus, the complexity of SRT SQRT is similar to its division counterpart. A modified Dijkstra SQRT is

presented in [10]. While this algorithm is radix-2, its complexity is less than half of that of radix-4 SRT division and thus lower than radix-4 SRT square root. Xilinx pPnR and LU results of our implementation are in Table 4. The structure of each iteration is illustrated in Fig. 4. Because our word length is 24 bits, an array processor needs 12 such structures cascaded together.

TABLE IV
PPNR TIMING AND LU OF DIJKSTRA SQRT

Dijkstra Square Rooting	Max. freq.	Latency	Slices
	150.263MHz	48 clocks	1,038

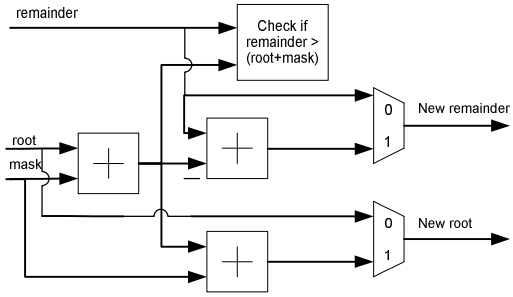

Fig. 4 Dijkstra SQRT

V. FUNCTION 2^X

The actual powering operation needed in the channel model is base 10, i.e., 10^x. On the other hand, base 2 powering is easier to implement. The transformation from base 10 to base 2 is shown in Fig. 5. However, in our channel model generator this transformation is not explicitly required since we can incorporate this constant, $log_2(10)$, into the channel model parameters which are pre-computed and stored as constants in the ROM.

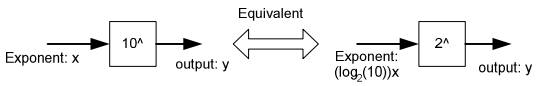

Fig. 5 Transformation from base 10 powering to base 2

A digital recurrence method is the primary approach used for implementations of hardware powering [13]. However, it exhibits the typical trade-off between the number of iterations and the size of the radix that is common to all digit recurrence algorithms. For the powering computation, each iteration requires a multiplication and an addition and therefore, if the digits are in radix-4 format, 7 stages are needed for a high speed pipelined array design. Thus, the complexity would be quite large. Alternatively, a Taylor series expansion is a traditional mathematical method to approximate functions.

$$f(x) = f(x_0) + (x - x_0) f'(x_0) + \frac{(x - x_0)^2}{2!} f''(x_0) + \mathbf{L} \qquad (8)$$

Choosing the order of the approximation is to trade off the size of the ROM and the number of multipliers/adders. We built our 2^x computation using the 3rd-order Taylor series expansion based on the algorithm presented in [11]. Since for 3rd-order approximation, we have

$$4m \geq d, \text{ and} \qquad (9)$$

$$3m + k \geq d \qquad (10)$$

where d is the word length, hence in our case, $m = 6$ and $k = 6$. Interestingly, instead of having five ROM tables of depth of 64, we could choose to use only two ROM tables with each having a depth of 64. This is because our function is powering and its derivatives are simply the same function multiplied by a constant. However, we need two more multipliers. The 3rd-order Taylor series expansion 2^x function approximation architecture is shown in Fig. 6. Xilinx pPnR and LU results for our implementation are given in Table 5.

TABLE V
PPNR TIMING AND LU OF FUNCTION 2^X USING TAYLOR SERIES EXPANSION

Base 2 Powering	Max. freq.	Latency	Slices
	135.117MHz	14 clocks	1,973

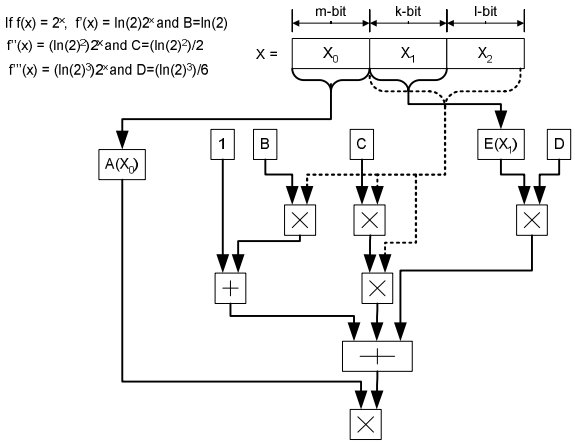

Fig. 6 3rd-order Taylor series expansion of 2^x

VI. COMPLETE HARDWARE CHANNEL MODEL

Reference [1] provides 4 different channel models for verification of the performance of UWB physical layer devices. There are seven key parameters that characterize the models. They are cluster arrival rate, ray arrival rate, cluster decay factor, ray decay factor, standard deviation of cluster lognormal fading term, standard deviation of ray lognormal fading term and the standard deviation of the lognormal shadowing term, all of which were derived based on the measurements reported in [14]. Since it is recommended that the channel follows a lognormal distribution, a GNRNG becomes a key part of this channel model simulator. In each channel realization, the model computes a cluster arrival time whenever this cluster arrival time isn't greater than 10 times the cluster decay factor. In one cluster, the model also computes a ray arrival time whenever this ray arrival time isn't greater than 10 times its decay factor. Each cluster and ray arrival time requires two GNRNs. Similarly, the signal amplitude of each ray needs two GNRNs with each for cluster lognormal fading and ray lognormal fading. To complete an amplitude computation, a 10^x function is needed since the model is lognormal distributed.

Since each channel realization has to be compensated with the lognormal shadowing term and the total energy has to be normalized to unity, one additional GNRN, square root and division are needed. As mentioned in [1], the time variability of the channel has to follow the recommendation set forth in [2] and [3] We can make use of that recommendation to determine how fast the channel model simulator runs. Reference [2] indicates that the channel has to change completely from packet to packet. Therefore, if we assume the average number of channel taps in a realization is one thousand and the packet duration for a 220 Mbps throughput application is 41.25μs, then the channel model hardware has to produce one channel tap in 41.25 ns, which is equivalent to a frequency of 24.24 MHz. Based on this speed, we find that the basic computational building blocks have to run at a speed of at least 100 MHz, which we have achieved. The complete architectural diagram of the channel model is given in Fig. 1. The implementation results from Xilinx pPnR and LU reports are listed in Table 6.

TABLE VI
PPNR TIMING AND LU OF UWB CHANNEL MODEL SIMULATOR

Channel model simulator	**Max. freq.**	**Latency**	**Slices**
	80.186MHz	1,089 clocks	7,747

VII. CONCLUSIONS

We built a complete UWB channel model emulator on a Xilinx Virtex-4 FPGA which runs at speeds up to 80 MHz. The logic utilization is 7,747 slices which consumes about half of the resources of a xc4vsx35 device. This leaves room for a transmitter and/or receiver to be implemented on the same FPGA device. In the process of building the overall channel emulator design, several high-speed arithmetic circuits were developed as well. Specifically, a 250 MHz Gaussian noise random number generator was developed. Also, a pipelined array division unit using a radix-4 SRT algorithm was created which runs at up to 133 MHz. A pipelined array square root processor using the Dijkstra algorithm runs at a speed of 150 MHz and also consumes relatively few resources. A 135 MHz 2^x processor using a 3rd-order Taylor series expansion method was also implemented. These individual arithmetic units may also prove to be useful in other applications which require high-speed arithmetic operators of those types.

ACKNOWLEDGMENT

We would like to thank Prof. Tzi-Dar Chiueh for valuable discussions.

REFERENCES

[1] J. Foerster, "Channel Modelling Sub-committee Report Final," IEEE P802.15-02/490-SG3a.

[2] S. V. Schell, "Analysis of Time Variance of a UWB Propagation Channel," IEEE P802.15-02/452-SG3a.

[3] A. Molisch, "Time variance for UWB wireless channels," IEEE P802.15-02/461-SG3a.

[4] C. Wallace, "Fast pseudorandom generator for normal and exponential variates," *ACM Tran. Math. Softw.*, vol. 22, no. 1, pp. 119-127, 1996.

[5] D.-U. Lee and et. al., "A Hardware Gaussian Niose Generator Using the Wallace Method," *IEEE Tran. on VLSI System*, vol. 13, no. 8, pp. 911-920, August 2005.

[6] D. L. Harris, S. F. Oberman, and M. A. Horowitz, "SRT Division Architectures and Implementations," *IEEE Symp. Comp. Arith.*, pp.18-25, July 1997.

[7] P. Soderquist and M. Lesser, "Division and Square Root Choosing the Right Implementation," *IEEE Micro.*, vol. 17, no. 4, pp. 56-66, July 1997.

[8] X. Wang and B. E. Nelson, "Tradeoffs of Designing Floating-Point Division and Square Root on Virtex FPGAs," *11th IEEE Symp. F.-P. Cust. Comp. Mach.*, April 2003.

[9] P. Kornerup, "Digital Selection for SRT Division and Square Root," *IEEE Tran. On Computers*, vol. 54, no. 3, pp. 294-303, March 2005.

[10] M. T. Tommiska, "Area-Efficient Implementation of a Fast Square Root Algorithm," *IEEE Proc. Dev. Circ. and Systems*, March 2000

[11] B. Lee and N. Burgess, "Some Results on Taylor-series Function Approximation on FPGA," *37th Asilomar Conference on Sig. Sys. and Comps.*, Nov. 2003.

[12] E. Rice and R. Hughey, "A New Iterative Structure for Hardware Division: the Parallel Paths Algorithm," *16th IEEE Symp. Comp. Arith.*, 2003

[13] J.-A. Pineiro and et al., "Algorithm and Architecture for Logarithm, Exponential, and Powering Computation," *IEEE Trans. Comp.*, vol. 53, no. 9, pp. 1085 – 1096, Sept. 2004.

[14] M. Pendergrass, "Emperically Based Statistical Ultra-Wideband Channel Model," IEEE P802.15-02/240-SG