International Journal of Modern Electronics and Communication Engineering (IJMECE) ISSN: 2321-2152
Volume No.-7, Issue No.-2, March, 2019

FPGA-based Implementation of Galois Field Arithmetic

Kinny Garg 1st
Ph.D. Research Scholar,
I.K.G. Punjab Technical University, Jalandhar, India.
Email: kinnygarg25@gmail.com

Dr. J.S. Sohal 2nd
Director, Ludhiana College of Engg. & Tech.,
KataniKalan, Ludhiana, India.
Email: jssohal2001@yahoo.co.in

Abstract: Error control coding for detecting and correcting errors in digital signals has assumed utmost importance in the modern world. This paper proposes a novel way to implement the Galois field arithmetic, the cornerstone of error-control coding, on Field Programmable Gate Arrays (FPGAs) using Verilog hardware description language (HDL). In this paper, we develop and implement the Galois Field arithmetic operations such as multiplying two elements, converting an element from primitive element power form to cartesian (or vector) form and vice-versa, and finding the inverse of an element, on FPGA. Error control coding is a good candidate for FPGA-based hardware implementation because FPGAs come with the advantages of reconfigurability, reprogrammability, design flexibility, reduced development times, and speedy operations.

Keywords: Error-control coding,Galois Field arithmetic, Field Programmable Gate Arrays (FPGAs), Digital signals

I. ERROR-CONTROL CODING

Digital signals exhibit greater reliability as compared to analog signals especially when the environment prone to noise. It is easier to detect digital signals correctly when the noise is not so strong because the detection of digital signals merely requires symbols to be classified as either “0” (binary zero) or “1” (binary one). However, when the noise is strong, the detection of digital signals may not be correct and it is prone to errors. [1] For instance, in the presence of noise, a detector may detect a binary symbol as “1” when it is “0” in reality, and vice-versa. In such noisy environments, it is possible to correct the errors at the cost of some redundant bits, which can be annexed with the data bits. This method is called as *error-control coding*.

Several important factors need to be considered while choosing an error-control code. The first important factor is the type of error that can occur while transmitting a digital signal, such as random error, burst error, or byte error. While a random error, as the name suggests, can occur anywhere in the data, a burst error is confined to contiguous portion of data. A byte error, on the other hand, occurs in a small block of data. Further, an error can be a mix of more than one of the above types. Each of these different types of errors require different treatment for detection as well as correction of errors. Table 1 lists typical error-control codes. [2]

Error-control coding plays a pivotal role in the modern digital systems. Two primary reasons can be attributed to the ever-increasing role of error-control codes in digital systems. The first reason is from the supply-side. The advances in solid state electronics allow even complex codes to be implemented on the hardware. The second reason is from the demand-side.[6] The ever-increasing need to transmit and receive enormous amount of data reliably has necessitated the demand for error control codes. In today’s world, the error-control coding is extensively used in satellite and mobile communications, broadcasting, semiconductormemory systems, magnetic recording systems, optical disk systems, and video systems.

Types of error	Codes	Decoder
Random	Self-orthogonal convolution codes BCH codes Convolution codes withViterbi decoding Convolution codes with sequential decoding Concatenated codes (REED-Solomon + Convolution)	Small ↕ Large
Burst	Iwadre codes Fire codes Reed-Solomon codes Doubly coded Reed-Solomon codes	Small ↕ Large
Byte	Reed-Solomon	

Table 1. Typical Error-Control Codes

II. GALOIS FIELD ARITHMETIC

A *finite field* is a field with a finite number of elements. Any mathematical operation carried out over a finite field results in another number that is in the field. The *order* of the field is defined by the count of element in it. A field of the order q is denoted by GF(q), where GF stands for Galois Field. GF(q) exists only when q can be expressed as $\mathbf{q = p^m}$, where p is a prime number and m is a positive integer. GF(p^m) exists for every p and m, and there is essentially only one field that has p^m elements.[7]

The prime number p is called the *characteristic* of the field. If a finite field has the characteristic p, then

$$p\beta = \beta + \beta + \ldots + \beta = 0 \text{ (p times)}$$

for any element β in the field. If p = 2, then

$$\beta + \beta = 0$$

$$\boxed{\beta = -\beta}$$

for any element β in the field.

Elements of GF(q) are expressed as

$$0, 1, \alpha, \alpha^2, \dots, \alpha^{q-2}.$$

The element 1 may be written as α^0. So all the elements of a finite field except 0 are powers of α. αis called as *primitive element* or *primitive root* and has the following properties:

$$\alpha^{q-1} = 1$$
$$\alpha^i \neq 1,\ 1 \leq i \leq q-2$$

A finite field is constructed using a *generator polynomial* (GP), which is also called *primitive polynomial* f(x) as it has no factors or is irreducible. A root of an equation f(x)=0 is called a zero of f(x). A primitive element of GF(p^m) is a zero of a primitive polynomial of order m i.e. f(α) = 0. The degree of the elements, which the GP or primitive polynomial describes, is one less than that of the GP. Using that α satisfies f(α) = 0, where f(x) is a primitive polynomial, an element of GF(p^m) is expressed as a polynomial in α :

$$\alpha^j = a_0 + a_1\alpha + a_2\alpha^2 + \dots + a_{m-1}\alpha^{m-1},\ a_i \in GF(p).$$

Here, only finite fields with characteristic 2 i.e. GF(2^n) have been taken up.

Illustration 1

Consider GF(2^3). There are only two primitive polynomials of degree 3 possible for this field : $f(x) = x^3 + x + 1$ and $f(x) = x^2 + x + 1$. Any one of the primitive polynomials can be chosen to generate the elements of the field. Each primitive polynomial generates a maximal but unique sequence.

Table 2. shows the elements of GF(2^3) generated by using the primitive polynomial $f(x) = x^3 + x + 1$. All elements (except 0) of this field can be expressed by polynomials formed by using only 1, α, and α^2 as shown below :

- Since α is a zero of $f(x) = x^3 + x + 1$, therefore $\alpha^3 + \alpha + 1 = 0$. Thus $\alpha^3 = \alpha + 1$ (since $\beta = -\beta$).
- $\alpha^4 = \alpha.\alpha^3 = \alpha.(\alpha + 1) = \alpha^2 + \alpha$.
- $\alpha^5 = \alpha.\alpha^4 = \alpha.(\alpha^2 + \alpha) = \alpha^3 + \alpha^2 = \alpha^2 + \alpha + 1$.
- $\alpha^6 = \alpha^2.\alpha^4 = \alpha^2.(\alpha^2 + \alpha) = \alpha^4 + \alpha^3 = (\alpha^2 + \alpha) + (\alpha + 1) = \alpha^2 + 1$.

Table 2. Elements of GF(2^3) where α is a zero of $f(x) = x^3 + x + 1$

As powers of primitive element	As polynomials	As vectors
α^0	1	(001)
α^1	α	(010)
α^2	α^2	(100)
α^3	α + 1	(011)
α^4	$\alpha^2 + \alpha$	(110)
α^5	$\alpha^2 + \alpha + 1$	(111)
α^6	$\alpha^2 + 1$	(101)

It can be observed that $\alpha^7 = \alpha.\alpha^6 = \alpha.(\alpha^2 + 1) = \alpha^3 + \alpha = (\alpha + 1) + \alpha = \alpha^0 = 1$ and sequence starts to repeat (i.e. $\alpha^8 = \alpha^1$). A reducible or non-primitive polynomial does not produce this maximal sequence. [8]

Illustration 2

Consider GF(2^4). There are only two primitive polynomials of degree 4 possible for this field : $f(x) = x^4 + x + 1$ and $f(x) = x^4 + x^3 + 1$. Table 3. shows the elements of GF(2^4) generated by using the primitive polynomial $f(x) = x^4 + x + 1$.

Table 3. Elements of GF(2^4) where α is a zero of $f(x) = x^4 + x + 1$

As powers of primitive element	As polynomials	As vectors
α^0	1	(0001)
α^1	α	(0010)
α^2	α^2	(0100)
α^3	α^3	(1000)
α^4	α + 1	(0011)
α^5	$\alpha^2 + \alpha$	(0110)
α^6	$\alpha^3 + \alpha^2$	(1100)
α^7	$\alpha^3 + \alpha + 1$	(1011)
α^8	$\alpha^2 + 1$	(0101)
α^9	$\alpha^3 + \alpha$	(1010)
α^{10}	$\alpha^2 + \alpha + 1$	(0111)
α^{11}	$\alpha^3 + \alpha^2 + \alpha$	(1110)
α^{12}	$\alpha^3 + \alpha^2 + \alpha + 1$	(1111)
α^{13}	$\alpha^3 + \alpha^2 + 1$	(1101)
α^{14}	$\alpha^3 + 1$	(1001)

Multiplication in a finite field can be performed according to the formula:

$$\alpha^i.\alpha^j = \alpha^{(i+j) \bmod (q-1)}$$

where*a* mod *b* is the remainder obtained when *a* is divided by *b*.

Addition in a finite field can be easily performed by expressing each element in its respective polynomial form.

If $\alpha^i = a_0 + a_1\alpha + a_2\alpha^2 + \dots + a_{m-1}\alpha^{m-1}$
and $\alpha^j = b_0 + b_1\alpha + b_2\alpha^2 + \dots + b_{m-1}\alpha^{m-1}$
then $\alpha^i + \alpha^j = (a_0 + b_0) + (a_1 + b_1)\alpha + (a_2 + b_2)\alpha^2 + \dots + (a_{m-1} + b_{m-1})\alpha^{m-1}$
where $(a_i + b_i)$ is performed under modulo *p* addition.

III. DESIGN FLOW

1. **Verilog HDL Design Entry:** Text Editor creates the Verilog file qs_fpga.v.
2. **Functional Simulation:**Modelsim simulates the design for functional verification.
3. **Verilog Code Synthesis:** Leonardo Spectrum creates the qs_fpga.edfnetlist file. This file is imported into the Figaro Place & Route tool.
4. **FPGA Place & Route Using Figaro:**qs_fpga.edf is imported into Figaro to place and route the design, qs_fpga.bst is created.
5. **Post-layout Co-verification:** Post-layout Co-verification provides a simulation of the FPGA design, including the timing information from the FPGA targeted to the architecture of the design, using Modelsim.

6. **Device Programming:** fpslic_qs_fpga.bst is created and downloaded to the device.

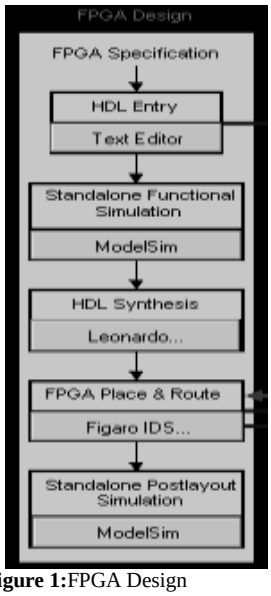

Figure 1: FPGA Design

IV. PROGRAMMING ON HARDWARE

The programs designed in a hardware description language (HDL) may be functionally good and could also produce the desired results on simulation. However, quite often these programs cannot be synthesized. We encountered this problem while synthesizing all Verilog codes designed for Galois Field Arithmetic (which otherwise on compilation and simulation gave correct results) during my research work. This can be best understood by taking a simple example of a loop control statement such as for or while. Verilog executes all expressions, whether evaluating to a constant or a variable, inside the parentheses. But synthesis demands the expression inside a loop control statement to be a constant or evaluating to a constant. All the programs have been modified (and some even rewritten also) to overcome this problem during the research work. The Verilog codes designed and implemented on FPGA for Galois Field Arithmetic Operations such as multiplication, conversion from primitive element power to cartesian / vector form and vice-versa, and inversion are given (along with functional simulation outputs) on succeeding pages. [3]

Software and hardware tools used for programming

- AT94K Series FPSLIC (Field Programmable System Level Integrated Circuit)
- ModelSim PE 5.6a
- System Designer 3.0

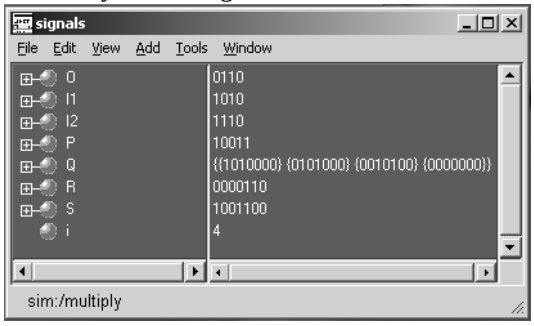

Figure 2: Verilog Functional Simulation output Of Multiplier

In Figure 2, there are two inputs I1 and I2 with values 1010 and 1110, respectively. From Table 3, the equivalent values of 1010 and 1110 are α^9 and α^{11} respectively. Now using the multiplication formula, $\alpha^i . \alpha^j = \alpha^{(i+j) \bmod (q-1)} = \alpha^{(9+11) \bmod (16-1)} = \alpha^{20 \bmod 15} = \alpha^5$. The value of output (O) corresponding to α^5 is 0110 (as correctly seen in the simulation output).

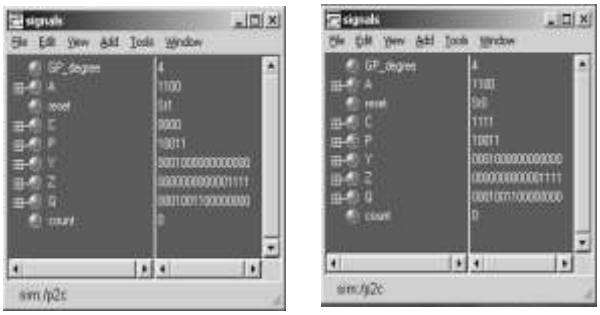

Figure 3: Verilog Functional Simulation outputs of P2C

The Figure 3 depicts the conversion from primitive element power (A) form to cartesian/vector (C) form. The value given to input (A) is 1100 which is equivalent to $2^2 + 2^3 = 12$ or α^{12}. After conversion from primitive element power form to cartesian form, the output is 1111, which is in line with that shown in Table 3.

Figure 4: Verilog Functional simulation output of C2P

Figure 4 depicts the conversion from Cartesian/Vector (C) primitive element power (A). The value given to input (A) is 1111 in the cartesian form, which is equivalent to $\alpha^3 + \alpha^2 + \alpha + 1 = \alpha^{12}$ in the primitive element form. Therefore, after conversion from the cartesian form to the primitive element power form, the output is 12 (decimal) or 1111 (binary), which is in line with that shown in Table 3.

Figure 5: Verilog Functional simulation output of Inverse

Figure 5 shows the computation of inverse of 1111, which is 1000 as shown in the simulation output.

V. CONCLUSION

Galois Field Arithmetic Operations have been successfully designed in Verilog HDL and implemented on FPGA based AT94K Series FPSLIC (Field Programmable System Level Integrated Circuit) kit. Coupled with the advantages of reprogrammability, reconfigurability, design flexibility, reduced development time, cost & space saving and speedy operations, the use of FPGA for implementation of Galois Field Arithmetic Operations is justified. It can be used further for the development of various error-control functions useful in communications.

REFERENCES

[1] A. Houghton, Error Coding for Engineers, Kluwer Academic Publishers, 2001.

[2] Hideki Imai, Essentials of Error-Control Coding Techniques, Academic Press, Inc., 1990.

[3] Pak K. Chan &SamihaMourad, Digital Design using Field Programmable GateArrays, Prentice-Hall, Inc., 1994.

[4] Samir Palnitkar, Verilog HDL: A Guide to Digital Design and Synthesis, Sun Microsystems, Inc., 1996.

[5] T. K. Moon, "Error Correction Coding", John Wiley & Sons, 2005.

[6] J. G. Proakis, "Digital Communications", PrenticeHall, 4th edition, 2005

[7] E. Artin,Galois Theory, 2nd Ed., Notre Dame Press, 1956.

[8] R.S. Aggarwal: A text book on modern Algebra

Magazines

[1] Embedded Systems Programming Issues: January 2005, October 2004, and April 2004.

[2] Electronics WorldIssues: May 2004, June 2004.

AUTHORS' BIOGRAPHIES

KinnyGarg is pursuing Ph.D. from I.K.G. Punjab Technical University, Jalandhar, India. She has an experience of more than 5 years in teaching. She has received M.E. degree in Electronics and Communication Engineering from ThaparUniversity, Patiala in 2009 and B.Tech. degree in Electronics and Communication Engineering from GianiZail Singh College of Engineering & Technology, Bathinda in 2007. Her research interests are in the area of networking, wireless communication, and cryptography.

Dr. J.S. Sohalis the Director of Ludhiana College of Engineering and Technology, KataniKalan, Ludhiana.He has a professional experience of more than 35 years spanning across teaching, research, and administration. He has served as the Dean FET, Prof & Head DET, and DSW at GNDU, Amritsar. He has also served as a Professor and the Head of the Department of Computer Science and Electrical Engineering, PAU, Ludhiana. He has beena member of theAcademic Council of BOS in CSE & IT at PTU, Jalandhar. He has guided 7 Ph.D. students and severalM.Tech students.