A Systolic Algorithm and Architecture for Inversion in GF(2^8)

1Ashwini Shende, 2Ujwala Ghodeswar & 3G.G.Sarate
1&2 Yeshwantrao Chavan College of Engineering, 
3Government Polytechnic, Amravati
E-mail: ashu.shende09@gmail.com, sakharkar2002@yahoo.co.in, ggs_anshu@rediffmail.com

Abstract – In this paper, we present pipelined systolic architecture suitable for inversion in finite field GF(2^8) using modified Euclidean algorithm. The architecture is based on distributed control mechanism where ring counters are distributed into the computing cells. Each cell or processing element process data in parallel fashion by passing them from one processing element to other processing element in rhythmic fashion. Due to its structural regularity and consequent suitability for VLSI implementation this architecture achieves better overall performance when compared with all the previously proposed architectures.

Index Terms— Finite field inversion, modified extended Euclidean algorithm, pipelined architecture, systolic arrays, VLSI architecture.

I. INTRODUCTION

Finite fields have found applications in error-correcting codes [4], switching theory, digital signal processing [1], [2], [3], [10] and powerful mathematical framework for many computer applications including cryptography [1], [2], [3], [4], [9] because the arithmetic in these fields does not involve carries and can be employed to implement a system efficiently. Optimal design of a finite field arithmetic circuit that has short critical path delay (CPD), high throughput, short latency, and small area and/or time complexities has been an extensive research topic. The most complicated arithmetic operations include inversion and division. Systolic architecture [1], [2], [3], [7] is a general methodology for inversion operations. Systolic arrays, first introduced by Kung [7], have become popular because they are capable of achieving high throughput by maximizing pipelining and eliminating global data interconnects. Because simple, regular communication and control structures have advantages over complicated ones in design and implementation, cells in a systolic system are interconnected to form a systolic array or a systolic tree. Here data flows from memory in rhythmic fashion, passing through many processing elements before it returns to memory. Being able to use each input data a number of times, modular expansibility, simple and regular data and control flows, use of simple and uniform cells, elimination of global broadcasting and fan-in and fast response time are some of the advantages of systolic approach.

The most important aspect in the design of a systolic array is the mapping of the algorithm to the processor array. Recursive algorithms with regular data flows are suitable for systolization. In this paper, a modified Euclidean algorithm [1], [2], [3], is considered as a way of computing inverses in GF (2^8) and a systolic architecture is proposed based on this algorithm. Implementation based on modified Euclidean algorithm presents a high degree of parallelism and pipelinability at bit level which can be easily optimized to achieve local data flow and to eliminate the global interconnects which represents most important bottleneck in today’s sub-micron process. Our architecture implements control mechanism in a distributed fashion [1] and hence its critical path delay does not depend on the circuitry of adders but on delays of two logic gates only. The control mechanism can also be centralized as in [1] but has larger critical path delay as compared to architecture with distributed ring counters.

II. MODIFIED EUCLIDEAN ALGORITHM

Let \( g(x) = x^8 + g_7 x^7 + \ldots + g_1 x + 1 \) be an irreducible polynomial of degree \( m \) (that is 8) over GF(2). The field GF(2^8) can be viewed as a set of binary polynomials \( a(x) = a_7 x^7 + a_6 x^6 + \ldots + a_1 x + a_0 x^0 \) with addition and multiplication modulo \( g(x) \). If \( a(x) a(x) = 1 \pmod{g(x)} \), \( a(x) \) is said to be the
multiplicative inverse of \(a(x)\) modulo \(g(x)\), denoted as \([a(x)]^{-1} (\text{mod } g(x))\).

Our modified Euclidean algorithm keeps track of four binary polynomials, \(u(x), v(x), s(x)\) and \(t(x)\), stored in ten 9-bit registers with bits numbered from left to right as \((m,m-1,...,0)\) (where \(m=8\)). However,

\[
u(x) = u_j x^8 + ... + u_1 x + u_0 \text{ and similarly} \]

\[
\begin{align*}
u(x) &= v_j x^8 + ... + v_1 x + v_0, \text{ while} \\
s(x) &= s_j x^8 + ... + s_1 x^7 + s_0 x^8 \text{ and similarly} \\
t(x) &= t_j x^8 + ... + t_1 x^7 + t_0 x^8;
\end{align*}
\]

That is, the bits at the left ends of \(u\) and \(v\) are the highest degree terms of \(u(x)\) and \(v(x)\), respectively, whereas the bits at the right ends of \(s\) and \(t\) registers are their respective highest degree terms. Our modified Euclidean algorithm is as follows:

1. **Initialization:** \(u_i^{(0)} = g, v_i^{(0)} = 0\) for \(i=0,1,...,8; s_i^{(0)} = 0, v_i^{(0)} = 1, s_i^{(0)} = a_i; s_i^{(0)} = 0\) for \(i = 1,...,m\), and \(\text{sign}^{(0)} = 1\).

Table 1 shows initialization of four basic binary polynomials \(u(x), v(x), s(x)\) and \(t(x)\).

2. For \(j=1,2,...,2m-1\) do compute the control signals \(C_1 = v_m^{(j-1)}\), \(C_2 = C_1 \cdot \text{sign}^{(j)}\) and \(C_3 = \text{sign}^{(j)}\), and for \(i=0,1,\ldots,m\), compute

\[
( v_{i-1}^{(j-1)} + u_{i-1}^{(j-1)}, s_{i-1}^{(j-1)} + t_{i-1}^{(j-1)}) \quad \text{if } C_1 = 1
\]

\[
( u_{i}^{(j-1)}, v_{i}^{(j-1)} + t_{i}^{(j-1)}) \quad \text{if } C_2 = 1
\]

\[
( r_{j}^{(j-1)}, r_{j}^{(j-1)}, \ldots, r_{m-1}^{(j-1)}, r_{0}^{(j-1)}) \quad \text{if } C_3 = 1
\]

\[
( r_{j}^{(j-1)}, r_{j}^{(j-1)}, \ldots, r_{m-1}^{(j-1)}, 0) \quad \text{if } C_3 = 0
\]

3. **Output:** \(a_i = j^{2m-1}\) for \(i = 0,\ldots,m-1\).

Note that \(u_i^{(j-1)} = v_i^{(j-1)} = s_i^{(j-1)} = t_i^{(j-1)} = 0\) for all \(j\).

### III. IMPLEMENTATION OF CONTROL CELL AND COMPUTING CELL

The control cell and computing cell required for proposed systolic architecture is given in figure 1 and figure 2, respectively. The control cell consists of one MUX and one AND gate. MUX receives input as \(v_{m}^{(j-1)}\) and \(r_{i}^{(j-1)}\) where \(\text{sign}^{(j)}\) acts like select line and gives the output depending on the value given to it. Thus, at the output of MUX we get control signal \(C_3\) given as

\[
C_3 = \text{sign}^{(j)} + \left( \text{sign}^{(j+1)} \cdot \overline{C_1} \right) + \left( \text{sign}^{(j+1)} \cdot r_{i}^{(j+1)} \right)
\]

The control signal \(C_2\) is the output of AND gate obtained by performing anding operation on \(v_{m}^{(j-1)}\) and \(\text{sign}^{(j)}\). Similarly, \(C_1\) is equal to the bit supplied to \(v_{m}^{(j-1)}\). In this way all the three control signals are obtained at the output of control cell 1. Thus, in this distributed architecture out of \(m+1\) bits in \(j^{th}\) row only \(r_{i}^{(j-1)}\) is used in generating the control signals in the \(j^{th}\) control cell. Both \(r_{i}^{(j-1)}\) and \(v_{m}^{(j-1)}\) are supplied to the control cell in \(j^{th}\) row from the leftmost computing cell in the \((j-1)^{th}\) row. The control cell thus needs to store only the \(\text{sign}^{(j)}\) bit. \(\text{sign}^{(0)}\) is initialized to 1. The update \(\text{sign}^{(j)}\) can be implemented with a 2-input MUX gate controlled by \(\text{sign}^{(j-1)}\) because \(\text{sign}^{(j-1)} = 1\) implies that \(r_{i}^{(j-1)} = 0\). These control signals are then passed to neighboring computing cells. The computing cell consists of three MUX, two AND gates and two XOR gates. And finally at the end of 15\(^{th}\) row we get the desired output. Thus, at the output we get the inversion of \(a(x)\) that is \(a(x) = [a(x)]^{-1} \text{mod } g(x)\)

**TABLE I** INITIALIZATION

<table>
<thead>
<tr>
<th>i</th>
<th>(u_i^{(0)})</th>
<th>(t_i^{(0)})</th>
<th>(v_i^{(0)})</th>
<th>(s_i^{(0)})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

ISSN (Print): 2321-5747, Volume-1, Issue-1, 2013

20
IV. PIPELINED ARCHITECTURE WITH DISTRIBUTED RING COUNTERS

The proposed pipelined systolic inversion architecture consists of array of cells with \(2m-1\) (that is 15) rows and \(m+2\) (that is 10) columns. The total array consists of 150 cells in all with the leftmost cell in each row known as control cell and the remaining \(m+1\) cells are computing cells as explained above. Thus, there are 15 control cells and 135 computing cells in this systolic architecture. The pipelined architecture with distributed control mechanism for inversion in GF(\(2^8\)) is shown in figure 3. Each of the computing cells receives input from two or three computing cells situated in the upper row of that corresponding cell, performs necessary computation and gives the output to computing cell located in the row below it. The control cell thus generates the control signal \(C1\), \(C2\) and \(C3\) for each row in the array and passes them to adjacent computing cell. Each computing cell of corresponding row updates the bits in the cell using control signal \(C1\) and \(C2\), and passes the control signals to the next computing cell in the same row. It is also necessary to propagate control signal \(C3\) as it controls the updates of the ring counters and passes it to all the computing cells. Thus the results of computation of each computing cell (viz. (1), (2) and (3) is passed to adjacent cells in the next row. This distributed bit-level pipelined architecture is thus designed to process continuous inputs. Systolic arrays thus have balanced, uniform, grid-like architectures of special processing elements that process data like n-dimensional pipeline.

The input to the array is given keeping in mind our four binary polynomials, \(u(x)\), \(v(x)\), \(s(x)\) and \(t(x)\). The input is considered as an example say \(a(x) = x^6 + x^3 + 1\) and primitive polynomial is given as \(g(x) = x^8 + x^4 + x^3 + x^2 + 1\). Considering the above taken \(a(x)\) and \(g(x)\) we get, \(a(x) = t(x) = x^7 + x^6 + x^4\) which is the result of inversion operation in GF(\(2^8\)) using systolic architecture with distributed ring counters.

Table 2 shows hardware required for proposed systolic inversion architecture using distributed control mechanism. The gate number of proposed architecture has smaller number of gate counts. Also, the number of MUX required are less as compared to other architectures. There is no use of adder circuit [1], [2] which achieves smaller critical path delay as compared to our circuit.

<table>
<thead>
<tr>
<th>TABLE 2. Hardware required for architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Distributed ring counter</strong></td>
</tr>
<tr>
<td><strong>OR gates</strong></td>
</tr>
<tr>
<td><strong>NOT gates</strong></td>
</tr>
<tr>
<td><strong>Adder</strong></td>
</tr>
<tr>
<td><strong>AND gates</strong></td>
</tr>
<tr>
<td><strong>XOR gates</strong></td>
</tr>
<tr>
<td><strong>MUX</strong></td>
</tr>
</tbody>
</table>

V. CONCLUSION

In this paper, systolization as a solution to the VLSI design complexity of computational intensive and iterative algorithm has been reviewed. A major complexity issue, the interconnect bottleneck in the submicron design of algorithms like modified Euclidean algorithm has been explained. The inversion operation was found to be less complex without using adders. Thus, a novel class of pipelined systolic architecture to implement inversion over galois fields has been proposed. For the purpose of verification, the array was designed in Tanner EDA tool using 180nm technology.
VI. REFERENCES


