Reconfigurable Fourier and Fermat Number Transform

Pandit Madhuri D.1, Dr. Godbole B. B.2, Nikam Reshma H.3

Abstract-- Reconfiguration is an essential part of Software Radio (SR) technology. In this SR context, the Fast Fourier Transform (FFT) operator is defined as a common operator for many classical telecommunications operations. The systems are designed with a new architecture for this operator that makes it a device intended to perform two different transforms. The first one is the Fast Fourier Transform (FFT) used for the operations in communication. The second one is the Fermat Number Transform (FNT) used for the finite operations in the Galois Field (GF). This operator can be reconfigured to switch from an operator dedicated to compute the FFT to an operator which computes the FNT in the Galois Field.

Index Terms-- Software Radio, Reconfigurable, FFT, FNT.

I. INTRODUCTION

Over the past years, a proliferation of communication standards has substantially increased the complexity of radio designs, the communication standards are implemented separately using dedicated instantiations which are difficult to upgrade for the support of new features. So the concept software radio (SWR)[1][2] introduced by J. Mitola. Software Radio basically refers to an ensemble of techniques which permits the reconfiguration of a communication system without the need to change any hardware system element. This reconfiguration implies the optimization of the hardware-software resources in the terminal architecture design [3]. So this optimization is helpful in a new area of research. In this optics, it present a new architecture for the FFT whose Butterfly is reconfigurable so as to perform two kinds of transforms over two different fields. The first one is the FFT carry out some function of telecommunication operation. The second one in the GF where the FFT will be reconfigured as an FNT.

II. FFT AND FNT

The Number Theoretic Transform (NTT) has been introduced as a generalization of the Discrete Fourier Transform (DFT) over residue class rings of integers [4][5]. Interesting applications of the NTT lies in fast coding, decoding[6], long integer multiplication, cryptography, digital filtering, image processing and deconvolution. For transform length equal to F_L where \( F_L = 2^{2^L} \) is the Fermat number, the NTT is called the Fermat Number Transform (FNT) which presents some advantages.

It is quite obvious, that FNT is suitable for VLSI implementations. The structure of the FNT is identical to that of the DFT for power of two lengths. Then the same algorithms can be used for the classical radix-2 FFT and the radix-2 FNT. The only one difference is the substitution of the complex multiplication in the Fourier transform by a modulo \( F_L \) real multiplication in the case of the FNT. The following gives the definitions of FFT and FNT.

Fourier transform theory over complex field as well as finite field is given as. In the complex field \((C)\), the Discrete Fourier Transform of \( f_\alpha = (f_0, f_1, ..., f_{N-L}) \), a vector of real or complex numbers, is a vector \( F_\alpha = (F_0, F_1, ..., F_{N-L}) \), given by[8][10],

\[
F_k = \sum_{n=0}^{N-1} f_n W_N^{kn} \quad k = 0, ..., N - 1
\]

Where, \( W_N = \exp(-2\pi j/N) \) and \( j = -1 \). \( W_N^{kn} \) is referred as the twiddle factor. The Fourier kernel \( \exp(-2\pi j/N) \) is an \( N^{th} \) root of unity in the field \( C \). In the finite field \( GF(q) \), an element \( \alpha \) of order \( N \) is an \( N^{th} \) root of unity. Drawing on the analogy between \( \exp(-2\pi j/N) \) and \( \alpha \), Fourier transform over finite field can be defined as follows let \( f=(f_0, f_1, ..., f_{N-L}) \) be a vector over \( GF(q) \), and let \( \alpha \) be an element of \( GF(q) \) of order \( N \). The Fourier transform of vector \( f \) is the vector \( F=(F_0, F_1, ..., F_{N-L}) \) whose components are given by[3][8],

\[
F_j = \sum_{n=0}^{N-1} f_n \alpha^{nj} \quad j = 0, ..., N - 1.
\]

Vector \( f \) is related to its spectrum \( F \) by,

\[
f_i = \frac{1}{N} \sum_{n=0}^{N-1} F_j \alpha^{ni} \quad i = 0, ..., N - 1.
\]

The application of the Discrete Fourier Transform in the complex field occurs throughout the subject of signal processing. The same transform technique can play an important role in the study and processing of \( GF(q) \) valued signals, \( q \) a prime number. It describes in details the practical realization of the FFT [7] operator defined in complex field and which can be reconfigured to become the FNT operator with arithmetic carried out modulo Fermat numbers. This reconfiguration consists in reconfiguring each Butterfly of the FFT structure.

479
In the next section it will present the Butterfly itself as a function which is constituted by several reconfigurable arithmetic operators.

III. RECONFIGURABLE BUTTERFLY

In the SWR concept, new area of research called "Parametrization" has been defined [8][9]. This technique consists to identify common resources, i.e Common Operator (CO) or Common Function (CF) between all the standards involved in the reconfiguration and in the standards themselves. Then, the trick is to exploit the same resources to execute two or more applications.[8]. In this context, the main goal of this work is to exploit the resources already present in the FFT structure to get the FNT one. With this purpose, the arithmetic operators i.e multiplier, adder and subtracter realizing operations over \( C \) should be redefined to realize a modulo\((F)\) operations[9]. Then the reconfiguration of the Butterfly (Figure 1) consists to reconfigure the aforementioned arithmetic elements. Then, one need to define a modulo\((F)\) multiplier, adder and subtracter.

![Figure 1. The complex butterfly](image)

**A. Modular Multiplication in GF \((F)\)**

The modulo \( 2^n + 1 \) multiplication is widely used in the computation of convolutions and in Residue Number Systems (RNS) arithmetic. Several architectures of modulo \( (2^n + 1) \) multiplier based on Ma’s algorithm [10].Indeed, there are two categories of algorithms for the modulo \( (2^n + 1) \) multiplication. The first one consists to perform the multiplication and after the correction [11]. The second one consists in the reduction of partial product. This modular multiplier can implemented in Spartan3, Virtex-II.

In Figure 2, the white elements indicate the elements not used in the case of operation over GF. As it is noticed in this figure, there is a reconfiguration of the connections inter-operators. The dotted lines connections represent the additional connections in this operating mode over GF. A basic modulo \((2^n + 1)\) multiplication algorithm consists in computing \( p=xy \), and dividing this product by \( 2^n + 1 \):

\[
xy \mod (2^n + 1) = p \mod (2^n+1)
\]

Here \( C_L \) and \( C_H \) the lower and higher words respectively of the product \( p \) as follows:

\[
C_L = \sum_{n=0}^{N-1} p_i 2^l \\
C_H = \sum_{n=0}^{N-1} p_i 2^l
\]

![Figure 2. The modulo \((2^n+1)\) multiplier](image)

The modulo \((2^n+1)\) operator depicted in Figure 2 is carried out by :

\[
xy \mod (2^n + 1) = (c_L, c_H + 2) \mod 2^n \text{if } c_L, c_H + 1 < 2^n = (c_L, c_H + 1) \mod 2^n \text{ otherwise}
\]

**B. Modular Addition in GF \((F)\)**

To perform an addition that returns directly the desired result, and adder shown in Figure 3. We define \( s^1, s^2 \) the sums at the first and second adders respectively with the \((n+2)\)-bit integer

\[
s^1 = [s^1_{n+1}s^1_{n}...s^1_{0}]
\]

\( x+y \)

The modulo \((2^n+1)\) addition can be expressed as:

\[
x + y \mod (2^n + 1) = \begin{cases} 
(2^n + 1) & \text{if } 0 \leq x + y < 2^n \\
(x + y) \mod 2^n + 2^n - 1 & \text{if } 2^n < x + y \leq 2^{n+1} \\
2^n & \text{if } (x = 2^n \text{ and } y = 0) \\
or(x = 0, y = 2^n)
\end{cases}
\]
Now, check the correctness of equation using figure 3.
First of all, let us consider \(x\) and \(y\) two elements of \(GF(F_2), 0 \leq x, y \leq 2^n, \text{Then}\)
\[0 \leq x + y \leq 2^{n+1}\]

It distinguishes the three following cases to establish the correctness of our algorithm:

1. For \(x + y = 2^{n+1} \text{ (i.e } x = y = 2^n)\)
   Here \(s_i = 2^{2n}(i.e. s_{n+1} = 1, s_i = 0 \text{ for } i = 0, \ldots n)\)
   Consequently \(s_n = 0 + 2^n - 1, s_n = 0,\)
   And algorithms return \(2^n - 1.\)

2. For \(x + y = 2^n (\text{ i.e } x = 2^n \text{ and } y = 0)\) we have:
   \[s_{n+1} = 1, s_n = 0,\]
   In this case \(s_1 = 1\) and the multiplexer selects \(2^n\) as result. This is only case where \(s_n = 1.\)

3. Finally, for \(0 \leq x + y < 2^n,\) we have:
   \[s_{n+1} = s_n = 0,\]
   And \((x + y) \mod 2^{n+1} = x + y\)

As known, the arithmetic subtracter is usually based on the arithmetic adder structure. For the modulo \((2^n + 1)\) subtracter, we propose an operator shown in Figure 4. The subtraction modulo can be \((2^n + 1)\) expressed as follows:

\[(x - y) \mod (2^n + 1) = \begin{cases} 2^n & \text{if } (x = 2^n \text{ and } y = 0) \\ (x + y + 1 + s_n) \mod 2^n & \text{otherwise} \end{cases}\]

The three following cases to establish the correctness of our algorithm:

1. If \(x \geq y \Rightarrow x + y + 1 \geq 2^{n+1},\)
   \[s_{n+1} = 1, s_n = 0,\]
   And algorithm returns \(x + y + 1.\)

2. If \(x \leq y \Rightarrow x + y + 1 < 2^{n+1}\)
   \[s_{n+1} = 0, s_n = 1,\]
   and algorithm returns \(x + y + 1 + 1.\)

3. If \((x = 2^n \text{ and } y = 0) \Rightarrow s_{n+1} = s_n = 1\)
   and the algorithm returns \(2^n.\)

Once the different elements of the Butterfly are defined, one can implement them to obtain the reconfigurable Butterfly. Figure 5 depicts the resulting hardware operator. The switch from an operating mode to another requires a change of the Fourier kernel and the reconfiguration of connection inter-operators. Assuming that the Butterfly is configured to operate over \(C\) and one wants to perform a calculation over \(GF(F_2).\) To do this, the Butterfly should download the primitive element \(a^i\), activate the different logic gate (AND, OR and the multiplexers) and reconfigure the connection inter-operators as shown in Figure 5. In the next section, the global architecture of the FNT is presented.
IV. THE FNT ARCHITECTURE

In the previous sections, presents the reconfiguration at a rather low level. The Butterfly constitutes a high parameterized function level. The fact to have this parameterized function allows to design a reconfigurable operator whose Butterfly forms the highest level operator. Figure 6 depicts the global reconfigurable operator. Over Complex field it is called FFT and over GF(Ft) is called FNT. This architecture has been validated by software.

![Figure 6. The architecture of FNT](image)

V. PROPOSED RECONFIGURABLE FFT WITH RESULT

For 8 point reconfigurable operation of Fourier and Fermat number Transform, control signal selects the operating mode. Control signal value (1 or 0) indicate that that the Fourier Transform and Fermat Transform are performed respectively. The table below shows the synthesis report of proposed work with the logic resource utilization.

<table>
<thead>
<tr>
<th>Device Utilization Summary (Estimated Value)</th>
<th>Logic Utilization</th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Slice</td>
<td>1242</td>
<td>3584</td>
<td>34%</td>
<td></td>
</tr>
<tr>
<td>Number of Slice Flip flop</td>
<td>1494</td>
<td>7168</td>
<td>20%</td>
<td></td>
</tr>
<tr>
<td>Number of 4 input LUTs</td>
<td>2008</td>
<td>7168</td>
<td>28%</td>
<td></td>
</tr>
<tr>
<td>Number of Mul 18*18s</td>
<td>16</td>
<td>16</td>
<td>100%</td>
<td></td>
</tr>
</tbody>
</table>

The following are simulation results of proposed work for 8 point FFT using Sparten3.

![Figure 7. Simulation result of reconfigurable FFT-FNT mode](image)

VI. CONCLUSION

The re-design of this of basic structure in such way that to operate as FFT as well as FNT. For this purpose a new reconfigurable arithmetic operators has been defined to build a reconfigurable Butterfly. Using this reconfigurable butterfly, FFT as common operator is obtained.

REFERENCES