A high-precision hardware-efficient radix-2\(^k\) FFT processor for SAR imaging system

Chen Yang\(^1\), Yizhuang Xie\(^{1,2}\), He Chen\(^1\), and Cuimei Ma\(^2\)

\(^1\) Beijing Key Laboratory of Embedded Real-time Information Processing Technology, Beijing Institute of Technology, Beijing 100081, China
\(^2\) Institute of Electronics, Chinese Academy of Sciences, Beijing 100190, China

a) xyz551.bit@bit.edu.cn

Abstract: This paper presents a high-precision, hardware-efficient FFT processor for an on-board SAR (synthetic aperture radar) imaging system. To meet the high resolution imaging and big data granularity processing requirements, a radix-2\(^k\) mixed FFT algorithm is proposed. The mixed radix FFT algorithm reduces the number of complex multiplication and the size of twiddle factor memory. To further reduce hardware resource and improve FFT precision, sufficient fixed-point simulation is performed for the fixed-point FFT processor design. As a proof of concept, a 32768-point fixed-point processor is implemented on XC6VCX240T FPGA platform. The proposed pipelined FFT processor achieves a signal-to-quantization noise ratio (SQNR) of 47.3 dB at 18-bit internal wordlength. Compared with Xilinx FFT v7.1 IP core, the results demonstrate that our design saves at least 11% memory and 57% arithmetic elements.

Keywords: synthetic aperture radar (SAR) imaging, fixed-point, radix-2\(^k\) pipeline FFT, CSD constant multiplier

Classification: Integrated circuits

References


1 Introduction

Fast Fourier Transform (FFT) is one of the most fundamental algorithms in digital signal processing domain. FFT is widely used in various applications such as 802.11n protocol—WLAN, long term evaluation (LTE), ultra-wideband (UWB) and digital video broadcasting (DVB). This change in applications greatly emphasizes the need for FFT processors. FFT algorithms, implementation architectures vary for different requirements. Different FFT processors is designed and implemented in previous literatures. For the 3GPP LTE specification, a 128- to 2048-point reconfigurable FFT is designed based on the SDF pipeline structure [1]. A novel 128/256-point pipeline FFT/IFFT processor for MIMO-OFDM system is designed in [2]. For WPAN applications, a 2048-point fixed-point FFT processor based on the mixed-radix multi-path delay feedback (MRMDF) structure is implemented in [3].

In this work, we mainly focus on a real-time SAR imaging system with a data granularity of 32768 x 32768. A wide observation swath, high resolution SAR imaging system requires long azimuth direction integration time and big range direction data rate. However, an on-board SAR data processor has a great limitation of hardware resource. Therefore, the FFT processor should be hardware-efficient. The radix-2 algorithm is a well-known simple algorithm for FFT processors, but it requires many complex multipliers. The radix-4 algorithm is primarily used for high data throughput FFT architectures, but requires a 4-point butterfly unit with high complexity. Radix-2^k algorithms simultaneously achieve a simple butterfly unit and reduce the number of complex multipliers [4, 5]. Thus we propose radix-2^3 and radix-2^4 mixed algorithm to implement a 32768-point FFT processor. In order to reduce hardware cost, we design two butterfly units, three CSD constant multipliers and a simple twiddle factor address generation block.

Techniques based on Field Programmable Gate Arrays (FPGAs) is a potential solution which satisfies all the computing constraints. However, to implement an
entire SAR imaging processing system using floating-point arithmetic on FPGA is inefficient. Our previous work [6, 7, 8] on implementing large-point, mixed radix FFT processors (32k, 64k, etc.) using fixed-point arithmetic shows that the fixed-point FFT can save at least 35% memory resource. The challenge of using fixed-point is how to handle the quantization error [9]. We carry out a group of fixed-point simulation to determine the internal wordlength of our design.

The rest of the article is organized as follows. In Section 2, radix-2\(^k\) algorithm is reviewed and the proposed mixed radix algorithm is induced. In Section 3, the low cost FFT processor architecture is introduced. Section 4 describes the fixed-point simulation. In Section 5, FPGA implementation results are provided and the comparison with some previous works is discussed. Section 6 summaries the main contributions of this work and concludes the paper.

2 FFT algorithm

2.1 Review of radix-2\(^k\) algorithm

A discrete Fourier transform (DFT) of length \(N\) is defined as follows:

\[
X(k) = \sum_{n=0}^{N-1} x(n)W_N^{kn}
\]  

(1)

The radix-2\(^k\) algorithm has the same butterfly structure regardless of the \(k\) value. However, the twiddle factor multiplication structure is varied with a factor \(k\). To induce the expression of radix-2\(^m\) algorithm, an \(m + 1\) dimension linear map is applied as follows:

\[
\begin{align*}
n &= \frac{N}{2}n_1 + \frac{N}{4}n_2 + \ldots + \frac{N}{2^m}n_{m+1}, \quad n_1, n_2, \ldots, n_m = 0, 1 \\
n_{m+1} &= 0, 1, \ldots, \frac{N}{2^m} - 1 \\
k &= k_1 + 2k_2 + \ldots + 2^{m-1}k_m + 2^mk_{m+1}, \quad k_1, k_2, \ldots, k_m = 0, 1 \\
k_{m+1} &= 0, 1, \ldots, \frac{N}{2^m} - 1
\end{align*}
\]

(2)

Then plug (2) into (1). The DFT expression changes as follows:

\[
X(k_1 + 2k_2 + \ldots + 2^mk_{m+1}) = \sum x\left(\frac{N}{2}n_1 + \frac{N}{4}n_2 + \ldots + n_{m+1}\right)W_N^{(k_1+2^m+2^m\ldots+k_{m+1})(\frac{N}{2^m}n_1+\ldots+n_{m+1})}
\]

(3)

As for radix-2\(^3\) and radix-2\(^4\) algorithms, the twiddle factor decompositions are shown in (4) and (5) respectively.

\[
W_N^{(k_1+2k_2+4k_3+8k_4+16k_5)(\frac{N}{2^1}n_1+\frac{N}{2^2}n_2+\frac{N}{2^3}n_3+\frac{N}{2^4}n_4+\frac{N}{2^5}n_5)} = (-1)^{n_1k_1}(-j)^{n_2k_2}(-1)^{n_3k_3}W_{8}^{n_4k_4}(-j)^{n_5k_5}W_{16}^{n_6k_6}
\]

(4)

\[
W_N^{(k_1+2k_2+4k_3+8k_4+16k_5+32k_6+64k_7+128k_8)(\frac{N}{2^1}n_1+\frac{N}{2^2}n_2+\frac{N}{2^3}n_3+\frac{N}{2^4}n_4+\frac{N}{2^5}n_5+\frac{N}{2^6}n_6+\frac{N}{2^7}n_7+\frac{N}{2^8}n_8)} = (-1)^{n_1k_1}(-j)^{n_2k_2}(-1)^{n_3k_3}W_{16}^{n_4k_4}(-j)^{n_5k_5}W_{64}^{n_6k_6}
\]

(5)
According to the expression of twiddle factors, it is clear that as the radix increases, constant factor such as $W_8^1$ and $W_{16}^1$ appears instead of true twiddle factors. Constant factor multiplication can be solved by designing corresponding constant factor multiplier. The number of twiddle factor directly affects the number of complex multipliers that will be occupied. We compared the number of constant twiddle factors, non-trivial twiddle factors (except constant factors) and total twiddle factors in different FFT algorithms. The result is shown in Table I.

### Table I. Number of twiddle factors in radix-2^k algorithm

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Constant Twiddle Factors</th>
<th>Non-trivial Twiddle Factors</th>
<th>Total Twiddle Factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radix-2</td>
<td>0</td>
<td>$\frac{1}{2} \log_2 N + 1 - N$</td>
<td>$\frac{1}{2} \log_2 N + 1 - N$</td>
</tr>
<tr>
<td>Radix-2^2</td>
<td>0</td>
<td>$\frac{1}{8} N \log_2 N + 1 - N$</td>
<td>$\frac{1}{8} N \log_2 N + 1 - N$</td>
</tr>
<tr>
<td>Radix-2^3</td>
<td>$\frac{N}{12} \log_2 N$</td>
<td>$\frac{7}{24} N \log_2 N + 1 - N$</td>
<td>$\frac{7}{24} N \log_2 N + 1 - N$</td>
</tr>
<tr>
<td>Radix-2^4</td>
<td>$\frac{5}{24} N \log_2 N$</td>
<td>$\frac{15}{64} N \log_2 N + 1 - N$</td>
<td>$\frac{15}{64} N \log_2 N + 1 - N$</td>
</tr>
</tbody>
</table>

2.2 Proposed algorithm

As discussed above, radix-2^k appears to be effective at decreasing the number of complex twiddle factor multiplications. The FFT length 32768 can be decomposed as $2^4 \cdot 2^4 \cdot 2^4 \cdot 2^3$. We note that the constant twiddle factor of radix-2^3 algorithm is also included in radix-2^4 algorithm. Therefore, the multiplier design of radix-2^4 butterfly unit can be reused in radix-2^3 butterfly unit. Hence, the 32768-point calculation can be completed by first computing 4096-point DFT using radix-2^4 algorithm and then computing 8-point DFT using radix-2^3 algorithm.

3 Proposed architecture

In this section, we discuss the 32768-point FFT processor proposed. For higher-radix algorithms, the SDF architecture is preferred since it requires less memory and fewer complex multipliers than the MDC architecture [10]. Based on SDF architecture, the FFT processor is implemented by cascading three radix-2^4 stages (Module 1) and one radix-2^3 stage (Module 2) in order to accommodate the FFT size. The block diagram is shown in Fig. 1. The highlighted PE stage i and the framed PE end shows the internal structure of the processing element (PE).

![Block diagram of 32768-point FFT processor](image-url)
3.1 Butterfly unit
As shown in Fig. 1, a radix-2^4 butterfly unit or a radix-2^3 butterfly unit consists of three or four basic radix-2 butterfly operations respectively. We divide the butterfly unit into two types shown in Fig. 2. The butterfly unit consists of adders, data buffer (FIFO) and multiplexers. BU I includes two extra multiplexers before the output which complete the −j multiplication. Thus, a radix-2^4 or radix-2^3 processing element is performed using BU I, BU II, constant multipliers and complex multipliers.

Fig. 2. Circuits architecture of butterfly unit I (BU I) and butterfly unit II (BU II)

3.2 CSD constant multiplier
The main advantage of radix-2^k algorithm is the decrease of twiddle factor multiplications. To further save hardware resource, we design canonical sign digit (CSD) constant multiplier. CSD representation can efficiently decrease the number of adders and shift registers. The constant twiddle factors and the corresponding 16 bit CSD code is listed in Table II. The code Ī stands for −1.

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Decimal</th>
<th>2’s Comp.</th>
<th>CSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>sin((\frac{\pi}{8}))</td>
<td>0.3827</td>
<td>0011000011111100</td>
<td>010(\overline{0}001000000100)</td>
</tr>
<tr>
<td>cos((\frac{\pi}{8}))</td>
<td>0.9239</td>
<td>0111011001000001</td>
<td>1000(\overline{0}00010000001)</td>
</tr>
<tr>
<td>sin((\frac{\pi}{4})) = cos((\frac{\pi}{4}))</td>
<td>0.7071</td>
<td>0101101010000010</td>
<td>10(\overline{0}010101000001)</td>
</tr>
</tbody>
</table>

According to Table II, comparing CSD code with 2’s complement code we can see that using CSD code obviously saves adders and shift registers. The block diagram of constant multipliers is shown in Fig. 3. Only adders and shift registers are occupied to implement one multiplier.

3.3 Address generation module
Except for the constant twiddle factors, other twiddle factors are saved in ROMs. The complex multiplications are performed using booth complex multipliers. For high radix FFT algorithms, the twiddle factor address access becomes more complex.
After analyzing the twiddle factor expression (5), we use only one accumulator (ACC) to perform address generation. The procedure can be decomposed into three steps: 1) generate a $4 + \log_2 M(i)$ bit accumulator of which the lower bits count for $n_5 = 0, 1, \ldots, M(i)$, 2) map the four higher bits via bit-reverse operation, 3) the address can be generated by the equation:

$$Addr(i) = (8k_4 + 4k_3 + 2k_2 + k_1) \cdot n_5$$

Fig. 4 shows the address generation procedure of the $i$th stage.

![Fig. 3. Block diagram of CSD constant multiplier](image)

After analyzing the twiddle factor expression (5), we use only one accumulator (ACC) to perform address generation. The procedure can be decomposed into three steps: 1) generate a $4 + \log_2 M(i)$ bit accumulator of which the lower bits count for $n_5 = 0, 1, \ldots, M(i)$, 2) map the four higher bits via bit-reverse operation, 3) the address can be generated by the equation:

$$Addr(i) = (8k_4 + 4k_3 + 2k_2 + k_1) \cdot n_5$$

Fig. 4 shows the address generation procedure of the $i$th stage.

4 Fixed-point simulation

Fig. 5 shows CS algorithm calculation flow. It is clear that FFT operation account for the most computation. It cost too much resource to implement a 32768-point FFT processor with floating point. Fixed-point implementation certainly reduces hardware cost. However, the dynamic range of fixed-point data type is limited and

![Fig. 5. Calculation flow of CS algorithm](image)
the influence of fixed-point expression quantization error is also unavoidable. Long wordlength means great precision and large dynamic range while short wordlength occupies less storage and operation unit. Consequently, a tradeoff between precision and hardware cost should be found.

In this brief, we adopt fixed-point simulation to solve the fixed-point wordlength optimization issue. SystemC [11] is a modeling language based on C++ that is intended to enable system-level design. It has been developed as a standardized modeling language for systems containing both hardware and software components. It is a bit-accurate tool for FPGA or ASIC verification. Hence, we use the SystemC fast fixed-point simulation data type: sc_fix_fast to perform the simulation in the following discussion.

Floating-point to fixed-point conversion and fixed-point adder, multiplication during FFT operation cause quantization error. Fig. 6 shows the signal quantization error ratio (SQNR) performance at different internal wordlength of a 32768-point FFT. The result is achieved using chirp signal which is the most regular radar waveform as the input signal. During 14 bit to 24 bit the SQNR shows a linear increase. The SQNR reaches the peak value at 25 bit and remains unchanged.

In addition to SQNR performance test, we also carry on the system level verification. The azimuth FFT and range FFT shown in Fig. 5 are replaced by fixed-point FFT at different wordlength. A typical SAR system test scenario is point target scene. The imaging result is shown in Fig. 7 and the corresponding resolution, PSLR (peak side-lobe ratio) and ISLR (integrated side-lobe ratio) are listed in Table III. The origin image is achieved using IEEE standard single precision floating-point data format. It is obvious that FFT wordlength seriously affect image quality. With the decrease of wordlength, the image contrast, PSLR, ISLR and resolution are all deteriorating.

To implement the FFT processor with SDF structure, memory is a main factor to take into consideration. Table IV shows the bit-count evaluation for different wordlength.
### Table III. Point target evolution result

<table>
<thead>
<tr>
<th></th>
<th>Azimuth PSLR (dB)</th>
<th>ISLR (dB)</th>
<th>Res. (m)</th>
<th>Range PSLR (dB)</th>
<th>ISLR (dB)</th>
<th>Res. (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>−25.76</td>
<td>−22.51</td>
<td>5.52</td>
<td>−25.02</td>
<td>−23.31</td>
<td>2.82</td>
</tr>
<tr>
<td>18</td>
<td>−25.77</td>
<td>−22.40</td>
<td>5.68</td>
<td>−24.43</td>
<td>−22.84</td>
<td>2.82</td>
</tr>
<tr>
<td>17</td>
<td>−24.32</td>
<td>−22.31</td>
<td>5.74</td>
<td>−24.27</td>
<td>−21.98</td>
<td>2.83</td>
</tr>
<tr>
<td>16</td>
<td>−23.24</td>
<td>−21.45</td>
<td>5.96</td>
<td>−22.95</td>
<td>−20.61</td>
<td>2.94</td>
</tr>
<tr>
<td>15</td>
<td>−19.58</td>
<td>−17.30</td>
<td>6.26</td>
<td>−21.06</td>
<td>−17.22</td>
<td>2.97</td>
</tr>
<tr>
<td>14</td>
<td>−17.35</td>
<td>−14.82</td>
<td>6.74</td>
<td>−19.32</td>
<td>−14.50</td>
<td>3.05</td>
</tr>
</tbody>
</table>

Fig. 7. Point target imaging result

(1) 14 bit
(2) 15 bit
(3) 16 bit
(4) 17 bit
(5) 18 bit
(6) Origin
On the premise of making comprehensive consideration for FFT SQNR, imaging result and memory occupation, we select 18-bit IQ wordlength to implement the proposed 32768-point FFT processor.

5 Implementation and comparison

In order to verify our design, we implement the mixed radix, SDF structure FFT processor on Xilinx XC6VCX240T device. The execution time is 328.38 µs @ 100 MHz and the max work frequency is up to 180 MHz.

We compare our CSD constant multiplier with Xilinx LogiCORE IP Multiplier v11.2. The Xilinx IP core can be configured in two modes: either a parallel architecture for general-purpose multiplication or a constant-coefficient architecture for customized factors. Table V shows the resource comparison between proposed CSD multiplier and IP core. Compared with parallel multiplier and constant-coefficient Multiplier, 79% and 53% slices is saved, respectively. It is more efficient to perform the constant multiplications using our customized CSD multiplier.

We select Xilinx LogiCORE IP Fast Fourier Transform v7.1 (xfft v7.1) as a suitable comparison, limited to the lack of literatures that focus on implementing large point FFT processors on FPGA device. Table VI shows the resource comparison result. To fairly compare our implementation with Xilinx LogiCORE, the wordlength of xfft v7.1 is set to 18-bit and pipeline architecture is adopted. The

### Table IV. Bit-count of 32768-point FFT processor at different wordlength

<table>
<thead>
<tr>
<th>Wordlength</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQNR (dB)</td>
<td>23.27</td>
<td>29.27</td>
<td>35.26</td>
<td>41.27</td>
<td>47.30</td>
<td>53.32</td>
<td>59.31</td>
</tr>
<tr>
<td>Bit-count (Mbit)</td>
<td>3.49</td>
<td>3.74</td>
<td>3.98</td>
<td>4.24</td>
<td>4.48</td>
<td>4.75</td>
<td>5.01</td>
</tr>
</tbody>
</table>

### Table V. Resource comparison between proposed and Multiplier v11.2

<table>
<thead>
<tr>
<th></th>
<th>Xilinx Multiplier v11.2</th>
<th>Xilinx Multiplier v11.2</th>
<th>CSD Constant Multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Parallel Multiplier</td>
<td>Constant-Coefficient Multiplier</td>
<td>Constant Multiplier</td>
</tr>
<tr>
<td>Slice Registers</td>
<td>319</td>
<td>98</td>
<td>62</td>
</tr>
<tr>
<td>Slice LUTs</td>
<td>322</td>
<td>111</td>
<td>50</td>
</tr>
<tr>
<td>Slices</td>
<td>85</td>
<td>38</td>
<td>18</td>
</tr>
</tbody>
</table>

### Table VI. Resource comparison between proposed and Xilinx LogiCORE

<table>
<thead>
<tr>
<th></th>
<th>xfft v7.1</th>
<th>Proposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice Registers</td>
<td>4502</td>
<td>4808</td>
</tr>
<tr>
<td>Slice LUTs</td>
<td>3692</td>
<td>3315</td>
</tr>
<tr>
<td>DSP48E1s</td>
<td>28</td>
<td>12</td>
</tr>
<tr>
<td>Block RAM (18 kb)</td>
<td>92</td>
<td>82</td>
</tr>
</tbody>
</table>
results demonstrate that our design saves at least 11% Block RAM and 57% DSP48E1s arithmetic elements compared with Xilinx FFT v7.1 IP core.

6 Conclusion

In this paper, we design a 32768-point FFT processor for an on-board SAR imaging system. Considering about the hardware limitation of on-board SAR imaging processing, the radix-2^3, radix-2^4 mixed algorithm is proposed and fixed-point data type is adopted. With the simple butterfly arithmetic unit, CSD constant multiplier and twiddle factor address generation block, our design saves at least 11% memory and 57% arithmetic elements compared with Xilinx FFT v7.1 IP core. Meanwhile, our 18 bit fixed-point processor achieves 47.3 dB SQNR and the corresponding imaging result is acceptable. The results indicates that our design is hardware efficient and provides sufficient accuracy.

Acknowledgements

This work was supported by the Chang Jiang Scholars Programme under Grant T2012122, the Hundred Leading Talent Project of Beijing Science and Technology under Grant Z141101001514005.