A High Performance Implementation of Non-Power-of-Two FFT with EPUMA Platform

Zhenyu Liua, Qunfang Xiea, Hongkai Wanga,
Yanjun Zhanga*, Dake Liub,
a School of Information and Electronics, Beijing Institute of Technology, Beijing, 100081, China
bDepartment of EE, Linkoping University, Linkoping, 51583, Sweden

Abstract

Non-power-of-two points FFT processing is becoming more and more important due to the loading of new communication standard. In this paper, three non-power-of-two points (1152, 1200, 1536) FFT are implemented with a parallel computing platform, ePUMA. Simulation results show that the proposed implementation achieve better performance compared with the commercial TI DSP processor. The proposed implementation method can be applied to other points FFT or convolution algorithms.

© 2011 Published by Elsevier Ltd. Selection and/or peer-review under responsibility of Harbin University of Science and Technology. Open access under CC BY-NC-ND license.

Keywords: FFT, DSP, ePUMA

1. Introduction

Fast Fourier Transform (FFT) is one of the most powerful tools in digital signal processing applications and it is also the basic transformation employed by the latest wireless communication standard, LTE [1]. According to the LTE specification, the sequences length N is no longer limited to the field of power-of-two, but extends to an integer with factor of 2, 3 and 5, the field of non-power-of-two.

ePUMA [2,3] is a domain-specific embedded heterogeneous parallel computing multiprocessor platform. Parallel computing technology is a technical solution to meet the challenge of high computing
demands in modern telecommunication and multimedia applications. ePUMA is such a parallel computing system with master-multi-SIMD architecture with up to 516 computing channels.

The implementation of Non-power-of-two points FFT with ePUMA is discussed in this paper. The architecture of ePUMA platform and its related computing resources will be introduced in section 2. FFT algorithms and the implementation details will be introduced in section 3 and the performance will be given and evaluated in Section 4. A conclusion will be drawn in section 5.

2. Architecture of ePUMA

The architecture of ePUMA platform is illustrated in Fig. 1. It consists of four clusters, each containing one master controller, eight SIMD coprocessors, and one memory subsystem for on-chip communication.

![Fig. 1. Architecture of ePUMA](image)

In each SIMD processor, there are eight accessible computing channels and eight independent datapaths along with them. There are also eight multiplicands inside each SIMD, which can be used for internal computing commands but cannot be reached by external instructions. Taking advantage of these computing resources, a radix-4 FFT can be accomplished in one clock cycle by using a dedicated instruction, which contains sixteen 16-bit multiplications. A SIMD can also accomplish two radix-2 FFT computations in one cycle by applying a dedicated instruction.

3. Implementation of FFT

There are several FFT algorithms and their optimized or evolved algorithms [4, 5]. Zero-padding is the most direct way to implement non-power-of-two points DFT with fast radix-2 or radix-4 FFT algorithm. Its redundant input data result in low efficiency of hardware and poor signal-to-quantization-noise-ratio performance compared with other FFT algorithms. Winograd Fourier Transform Algorithm (WFTA) [5] employs the least multiplicands among the FFT algorithms, but at the cost of high programming complexity, large memory consumption and none in-place computing. The purpose of the proposed implementation is to achieve high performance, low clock cycle consumption. It is meaningless to reduce multiplication at the cost of clock cycle if the target platform has enough resources for computing. Considering the existence of radix-2 and radix-4 instruction and eight parallel multipliers, DFT is adopted to implement radix-3 and radix-5 computing and mixed-radix algorithm is used.

One key issue of FFT implementation with ePUMA is how to distribute the task to each sub-computing channel and organize the data access orderly to make full use of all processing channels. In most parallel computing platforms, the efficiency of data channels decides the processing ability of a parallel computing. In this section, the parallel data access of FFT implementation is discussed. The
method of data allocation is to avoid data access conflict and it can be applied to other points FFT implementations.

3.1. FFT algorithms

Input data is the data to be transformed with FFT algorithm. Generally, the transformation process is divided into several independent layers with a corresponding radix to speed up the transformation and reduce the complexity of processing. The input of the first layer of this procedure is the data given by the program or the external data inputted from outside of the processor. The input of the rest layer is the data outputted from its previous layer and stored as the input data of current layer. Input data here is not limited to the data inputted to the processor from outside, but also include the data inputted to each layer for transformation.

Input data permutation is used to group all inputted data into different vectors and to store them in a memory block of the vector memory. The purpose of data permutation is to distribute data to different memory blocks of a vector memory so that multiple data in different data channels can be accessed simultaneously. The process of 1536 points FFT is to illustrate how to allocate the input data for each layer with the LVM of ePUMA.

The process of 1536 points FFT is divided into six layers with radix-3,-4,-4,-4,-4 and radix-2 respectively. Six layers as shown in Fig.2 are adopted to process 1536 points FFT. In layer0, 512 radix-3 FFTs are processed and the results of processing are multiplied by \( w_{512}^{nk} \). In layer1, 3 groups FFTs are processed, each containing 128 radix-4 FFTs and the results of processing are multiplied by \( w_{512}^{nk} \). FFT processing is marked as “3*128 radix-4” to indicate the number of group and the processing load in each group. The processing from layer2 to layer 4 is shown in the Table 1. At last layer, layer5, 768 radix-2 FFTs are processed and the final results are obtained.

3.2. Input data distribution and permutation

From layer0 to layer5, radix-3, radix-4 and radix-2 FFT are implemented respectively. In each radix-n
FFT processing, n input data is required simultaneously and processed in parallel. Each input data is a complex vector of 16-bit real part and 16-bit imaginary part.

From layer0 to layer5, radix-3, radix-4 and radix-2 FFT are implemented respectively. In each radix-n FFT processing, n input data is required simultaneously and processed in parallel. In layer0, processor begins computing radix-3 FFT one by one with computing vector \(\{X_0, X_{512}, X_{1024}\}\) and is followed by \(\{X_1, X_{513}, X_{1025}\}\), \(\{X_2, X_{514}, X_{1026}\}\) etc. If input data is stored in LVM as normal sequential order, all three input vectors are located in the datapath0 and datapath1 for real and imaginary part respectively, as shown in Table 1 (a). It means that most datapath bandwidth is not used and three input vectors can only be reached one by one in three clock cycles for a one-cycle processing task. The data allocation in LVM for layer0 processing is shown in Table 1 (b). Two blank units are inserted ahead of input \(X_{512}\) and another two are inserted ahead of \(X_{1024}\). Thus, each input vector for one radix-3 processing is allocated in different datapaths and can be loaded to register in one memory access.

Table 1. Data allocation maps for each processing layer

From layer1 to layer4, radix-4 FFT need four vectors for one processing. An inversion implementation method is adopted to generate all data allocation maps for the whole procedure. We can make an input data allocation map, as shown in Table 1 (f), for the last layer (layer5). This map is also the result data map of its previous layer (layer4). Based on this data allocation map, the data map of its previous layer can be obtained, as shown in Table 1 (e). Then, one by one, all rest data allocations for each layer are drawn as shown in Table 1 (e) and (d). This inversion implementation method works well in 1152 and 1536 points FFT implementation and non-conflict memory access maps are generated accordingly.
4. Implementation results and evaluation

Three different points FFT, 1152, 1200 and 1536, are implemented in ePUMA platform. To evaluate these results, a fixed point pipelined TI DSP processor, TMS320C6454 [6], has been chosen for comparison. This DSP has similar VLIW architecture and computing ability with the target ePUMA. The benchmark of TI DSP [7] is provided with the power-of-two FFT points. So, only 1024 and 2048 points FFT performance are listed. The cycle counts of ePUMA 1024 and 2048 points FFT implementation are less than half of TI DSP processor consumed. For 1152 points FFT implementation, ePUMA is only 4.8% higher than the cycle count of TI DSP with 1024 points FFT implementation.

The performance of proposed implementation can be evaluated with TI dedicated FFT co-processor, FFTC [8]. Its computing ability and resources is more than twice of one SIMD in ePUMA. The performance of FFTC is shown in the fourth row of Table 2. The cycle counts of ePUMA are 13% and 35% more than the FFTC for 1536 and 1152 points respectively. Considering the computing resource gap between them, these two rates show good performance of ePUMA and efficiency of the implementation.

Table 2. resource utilization in FPGA

<table>
<thead>
<tr>
<th>Cycles</th>
<th>FFT</th>
<th>1024 points</th>
<th>1152 points</th>
<th>1200 points</th>
<th>1536 points</th>
<th>2048 points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>ePUMA (one SIMD)</td>
<td>1710</td>
<td>4074</td>
<td>7657</td>
<td>4296</td>
<td>4027</td>
</tr>
<tr>
<td>TMS320C6454</td>
<td>3878</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>8486</td>
<td></td>
</tr>
<tr>
<td>FFTC</td>
<td>2042</td>
<td>3026</td>
<td>3063</td>
<td>3794</td>
<td>4755</td>
<td></td>
</tr>
</tbody>
</table>

5. Conclusion

In this paper, three Non-Power-of-Two points: 1152, 1200 and 1536, FFT were implemented with a parallel computing platform, ePUMA. Data permutation technology was introduced and discussed in detail with an example of 1536 points FFT implementation. Simulation results show that the proposed implementation achieved better performance in terms of clock cycle compared with the commercial TI DSP processor. These results prove the efficiency of proposed implementation.

References