Analysis of Twiddle Factor Memory Complexity of Radix-$2^i$ Pipelined FFTs

Fahad Qureshi and Oscar Gustafsson

N.B.: When citing this work, cite the original article.

©2010 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.


Postprint available at: Linköping University Electronic Press
http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-65855
Analysis of Twiddle Factor Memory Complexity of Radix-$2^i$ Pipelined FFTs

Fahad Qureshi and Oscar Gustafsson
Department of Electrical Engineering, Linköping University
SE-581 83 Linköping, Sweden
E-mail: {fahadq, oscarg}@isy.liu.se

Abstract—In this work, we analyze different approaches to store the coefficient twiddle factors for different stages of pipelined Fast Fourier Transforms (FFTs). The analysis is based on complexity comparisons of different algorithms when implemented on Field-Programmable Gate Arrays (FPGAs) and ASIC for different radix-$2^i$ algorithms. The objective of this work is to investigate the best possible combination for storing the coefficient twiddle factor for each stage of the pipelined FFT.

I. INTRODUCTION

Computation of the discrete Fourier transform (DFT) and inverse DFT is used in e.g. orthogonal frequency-division multiplexing (OFDM) communication systems and spectrometers. An $N$-point DFT can be expressed as

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^k, k = 0, 1, N-1$$

where $W_N = e^{-j2\pi/N}$ is twiddle factor, the $N$:th primitive root of unity with it’s exponent being evaluated modulo $N$, $n$ is the time index, and $k$ is the frequency index. Various methods for efficiently computing (1) have been the subject of a large body of published literature. They are commonly referred to as fast Fourier transform (FFT) algorithms. Also, many different architectures to efficiently map the FFT algorithm to hardware have been proposed [1].

A commonly used architecture for transforms of length $N = b^i$ is the pipelined FFT. The pipeline architecture is characterized by continuous processing of input data. In addition, the pipeline architecture is highly regular, making it straightforward to automatically generate FFTs of various lengths.

Figure 1 outlines the architecture of a Radix-$2^i$ single-path delay feedback (SDF) decimation in frequency (DIF) pipeline FFT architecture for length $N$. This architecture is generic while the required ranges of each complex twiddle factor multiplier is outlined in Table I for varying numbers of $i$. For the twiddle factor multipliers with small ranges special methods have been proposed. Especially one can note that for a $W_4$ multiplier the possible coefficients are $\{\pm 1, \pm j\}$ and, hence, this can be simply solved by optionally interchanging real and imaginary parts and possibly negate (or replace the addition with a subtraction in the subsequent stage). For larger ranges ($W_8$, $W_{16}$, and $W_{32}$) approaches have been proposed in [4], [6]–[8].

In this work we instead focus on using standard complex multipliers. However the twiddle factors calculated advance, stored in memories and retrieved for multiplication whenever necessary. The size of the twiddle factor memory for each stage depends upon some factors; arithmetic precision, number of FFT point and number of the stage. Usually for a long FFT the lookup tables are large in comparison with butterfly and complex multiplier. In [9], [10] methods are proposed to reduce the size of the memories by utilizing the octave symmetry of the twiddle factors, hence only storing values for angles between $0 \leq \alpha \leq \pi/4$. The memory then have at most $(N/8 + 1)$ words. However, the results in [9], [10] are given for complete FFTs using the same architecture for all memories and only for radix-$2^4$. In this work we show that octave symmetry is not always useful due to the overhead of multiplexers and negations. Furthermore, we will investigate the wordlength scaling effect as previous work has shown that the occupied cell area when synthesizing lookup tables does not grow linearly with the number of bits in the look-up table [11]. It is noted that one could use dedicated memory structures on the FPGAs, but depending on available resources and the size of the memories this may not be suitable. For using the dedicated memory structures a cost model is proposed in [12].

In next section the different architectures to implement the twiddle factor memories are explained. In Section III, we analyze and compare the implementation results of those architectures. Finally, some conclusions are presented.

II. ARCHITECTURES FOR TWIDDLE FACTOR MEMORIES

The twiddle factor memory should provide the real and imaginary parts of the twiddle factor. Typically, in a SDF

<table>
<thead>
<tr>
<th>Radix</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$W_N$</td>
<td>$W_{N/2}$</td>
<td>$W_{N/4}$</td>
<td>$W_{N/8}$</td>
<td>$W_{N/16}$</td>
</tr>
<tr>
<td>$2^2$</td>
<td>$W_4$</td>
<td>$W_N$</td>
<td>$W_4$</td>
<td>$W_{N/4}$</td>
<td>$W_4$</td>
</tr>
<tr>
<td>$2^3$</td>
<td>$W_4$</td>
<td>$W_8$</td>
<td>$W_N$</td>
<td>$W_4$</td>
<td>$W_8$</td>
</tr>
<tr>
<td>$2^4$</td>
<td>$W_4$</td>
<td>$W_8$</td>
<td>$W_{16}$</td>
<td>$W_N$</td>
<td>$W_4$</td>
</tr>
<tr>
<td>$2^5$</td>
<td>$W_4$</td>
<td>$W_8$</td>
<td>$W_{16}$</td>
<td>$W_{32}$</td>
<td>$W_N$</td>
</tr>
</tbody>
</table>

TABLE I
MULTIPLICATION AT DIFFERENT STAGES FOR DIFFERENT ARCHITECTURE.
Fig. 1. The $R^{2^i}$ single-path delay feedback (SDF) decimation in frequency (DIF) pipeline FFT architecture with twiddle factor stages.

Fig. 2. Block diagram of Single Look-up Table twiddle factor memory.

Fig. 3. Block diagram of twiddle factor memory with address mapping.

A. Single Look-up Table

The simplest approach, as shown in Fig. 4, is to just use a large look-up table to store the twiddle factors. For a $W_N$ multiplier, $N$ words need to be stored. Hence, for large $N$ one could expect this method to have a higher complexity compared to the reduced schemes. On the other hand it lacks any overhead. It should also be noted that this scheme possibly stores the same twiddle factor in several positions as the mapping is from row to twiddle factor and for radix-$2^i$ algorithms some twiddle factors appears more than once for $i \geq 2$.

B. Twiddle Factor Memory with Address Mapping

A possible simplification is to use an address mapping circuit that maps the row to the corresponding angle ($k$ in (1)) and use a memory storing the required elements only once. For the general case, we will need to store many, but not all, values, still using $N$ possible words even though many can be set to “don’t care”. Because of this one can expect the resources used for the look-up table to be reduced compared to the previous approach, given that the synthesis tool can benefit from it. The structure is shown in Fig. 3.

C. Twiddle Factor Memory with Address Mapping and Symmetry

Another modification, that was proposed in [9], [10], is to use the well known octave symmetry to only store twiddle factors for $0 \leq \alpha \leq \pi/4$. The additional cost is an address mapping circuit as discussed in the previous section as well as multiplexers to interchange the real and imaginary parts and possible negations. The main benefit is that only $N/8+1$ words are required to be stored. The resulting structure is shown in Fig. 4.

D. Address Mapping

The address mapping for a Radix-$2^i$ FFT is done as shown in 5. Here, the total length of the FFT is $2^L$ points and the resolution of the twiddle factor multiplier is $W_{2^i}$. It is worth noting that the address mapping for a given $W_N$ multiplier is independent of $L$. Clearly, $i$ will affect the complexity of the address mapping circuitry.

III. Analysis and Results

We have analyzed complexity of twiddle factor memory having resolution $\geq 64$ with different architectures, considering radix-$2^i$ algorithm with different values of $i$. The architectures of the twiddle factor memories have been coded.
in VHDL. These architectures were synthesized using the three
different synthesis tools, Mentor Graphics Precision targeting
an Altera Stratix-IV FPGA, ISE Xilinx targeting an Virtex-
4 FPGA and Synopsys Design Compiler targeting 0.35μm
CMOS standard cells. The twiddle factors are represented
using 16 bits each for real and imaginary parts. The two’s
complement representation of the numbers is used in the
twiddle factor memory. The resulting complexity for each
stage is illustrated in Figs 6, 7, and 8 for different technologies
Altera Stratix-IV FPGA, Virtex-4 FPGA and 0.35μm CMOS
ASIC, respectively.

Figures 6, 7, and 8 show that the twiddle factor memory
with address mapping and symmetry architecture is the most
advantageous one for high range. However, for small ranges,
the simple look-up table approach is most beneficial. The point
where address mapping and symmetry is more beneficial than
the simple look-up table moves further towards the higher
resolution of twiddle factor as the value of \( i \) increases.

In FPGA designs, the memory with address mapping is not
a beneficial choice because the synthesis tool does not utilize
the “don’t care” conditions. However in the ASIC designs it is
in the middle of the both, although never the best. To illustrate
the input of the wordlength, we synthesize a \( W_{1024} \) twiddle
factor using wordlengths varying from 10 to 18 bits to a Xilinx
Virtex-4 FPGA. The results are shown in Fig. 9 and shows the
expected linear behaviour. However, the offset, corresponding
to the constant wordlength circuitry like address generation,
differs between the approaches. Hence, one would expect that
for resolutions that gave similar complexity in Figs. 6, 7, and 8,
one would have to re-evaluate the best architecture based on
the used wordlength.

Figure 10 shows the complexity using the best architec-
ture of the twiddle factor memory for radix-2\( ^{2} \) algorithm in
different technologies. It can be seen that, the twiddle factor
complexity for the same twiddle factor increases as the value
of \( i \) increases in radix-2\( ^{2} \) algorithms.

Fig. 6. Radix2\( ^{1} \) SDF pipelined FFT twiddle factor memory complexity using
Mentor Graphics Precision targeting an Altera Stratix-IV FPGA.

Fig. 7. Radix2\( ^{2} \) SDF pipelined FFT twiddle factor memory complexity using
ISE Xilinx targeting an Virtex-4 FPGA.

Fig. 8. \( W_{1024} \) twiddle factor memory complexity for different wordlength
using ISE Xilinx targeting an Virtex-4 FPGA.

Fig. 9. \( W_{1024} \) twiddle factor memory complexity using 0.35μm CMOS
standard cells.
Table II shows twiddle factors for a 8192-point FFT single delay feedback pipelined architecture having resolution ≥ 64 for different radix-2^i algorithms. The complexity of each twiddle factor memory with best architecture by using the three different technologies are shown in Tables III, IV

### TABLE II
**Twiddle Factor Memory Complexity of 8192-FFT SDF Pipelined with Different Algorithms.**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>2^2 [2]</td>
<td>W_8192 W_2048 W_512 W_128</td>
</tr>
<tr>
<td>2^4 [4]</td>
<td>W_8192 W_512 - -</td>
</tr>
<tr>
<td>2^5 [5]</td>
<td>W_8192 W_256 - -</td>
</tr>
</tbody>
</table>

### TABLE III
**Twiddle Factor Memory Complexity of 8192-FFT SDF Pipelined with Different Algorithms (ALTERA).**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Memory complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>2^2 [2]</td>
<td>2650 729 240 95 3714</td>
</tr>
<tr>
<td>2^3 [3]</td>
<td>2835 581 96 3512</td>
</tr>
<tr>
<td>2^4 [4]</td>
<td>3002 339 - 3341</td>
</tr>
<tr>
<td>2^5 [5]</td>
<td>3123 157 - 3280</td>
</tr>
</tbody>
</table>

### IV. CONCLUSIONS

In this paper, we have analyzed the complexity of twiddle factor memories for pipelined FFTs considering different architectures. Analysis is based on complexity comparisons of different radix-2^i algorithms when implemented either on FPGAs (field programmable gate array) or standard cells. The results show that a plain lookup table is advantageous for low resolution memories while for larger resolution twiddle factor memories, utilizing octave symmetry and a address generator is advantageous. The break-point where the plain lookup table approach is advantageous increases with increasing i.

### REFERENCES