Application Note, V1.1, October 2007

AP16119

XC2000 & XE166 Families

Fast Fourier Transform Based on XC2000 & XE166 Microcontroller Families

Microcontrollers

Never stop thinking

Edition 2007-10

Published by
Infineon Technologies AG
81726 Munich, Germany

LEGAL DISCLAIMER

THE INFORMATION GIVEN IN THIS APPLICATION NOTE IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS APPLICATION NOTE MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. INFINEON TECHNOLOGIES HEREBY DISCLAIMS ANY AND ALL WARRANTIES AND LIABILITIES OF ANY KIND (INCLUDING WITHOUT LIMITATION WARRANTIES OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OF ANY THIRD PARTY) WITH RESPECT TO ANY AND ALL INFORMATION GIVEN IN THIS APPLICATION NOTE.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may be used in life-support devices or systems only with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

AP16119		
Revision History:	2007-10	V1.1
Previous Version:	none	

Page	Subjects (major changes since last revision)

We Listen to Your Comments

Any information within this document that you feel is wrong, unclear or missing at all?
Your feedback will help us to continuously improve the quality of this document.
Please send your proposal (including a reference to this document) to:

mcdocu.comments@infineon.com

Table of Contents Page

1 Introduction 5

2 FFT Algorithm 7
2.1 Radix-2 Decimation-In-Time FFT Algorithm 7
2.2 Radix-2 Decimation-In-Frequency FFT Algorithm 9
2.3 Complex FFT Algorithm 11
2.4 Calculation of Real Forward FFT from Complete FFT 13
2.5 Calculation of Real Inverse FFT from Complete FFT 14

3 XC2000/XE166 Implementation Note 16
3.1 Organization of FFT Functions 16
3.2 Implementation of Real Forward FFT 16
3.3 Implementation of Real Inverse FFT 17

4 Description of Implemented Functions 19
4.1 Binary Bit Reverse of Input Data 19
4.2 Floating to Fixed Point Format 1Q15 20
4.3 Real Forward Radix-2 Decimation-in-Time FFT 21
4.4 Real Inverse Radix-2 Decimation-in-Frequency FFT 22
4.5 Usage of FFT Functions 24

5 Conclusion 26

6 Reference 27

1 Introduction

Spectrum (Spectral) analysis is a very important methodology in Digital Signal Processing. Many applications have a requirement of spectrum analysis. The spectrum analysis is a process of determining the frequency domain representation of the sequence. The analysis gives rise to the frequency content of the sampled waveform such as bandwidth and centre frequency. One of the methods of doing the spectrum analysis in Digital Signal Processing is by employing the Discrete Fourier Transform (DFT).

The DFT is used to analyze, manipulate and synthesize signals in ways not possible with continuous (analog) signal processing. It is a mathematical procedure that helps in determining the harmonic frequency content of a discrete signal sequence. The DFT is defined by

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

where

$$W_N = e^{-j2\pi/N} = \cos(2\pi nk/N) - j\sin(2\pi nk/N) \tag{2}$$

Using equations (1) and (2) we can rewrite X(k) as

$$X(k) = \sum_{n=0}^{N-1} x(n)[cod(2\pi nk/N) - j\sin(2\pi nk/N)] \tag{3}$$

X(k) is the k^{th} DFT output component for k=0,1,2,....,N-1,
x(n) is the sequence of discrete sample for n=0,1,2,...,N-1,
j is imaginary unit $\sqrt{-1}$,
N is the number of samples of the input sequence (and number of frequency points of DFT output).

The DFT is used to convert the signal from time domain to frequency domain. The complementary function for DFT is the IDFT, which is used to convert a signal from frequency to time domain. The IDFT is given by

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k) W_N^{-nk}, \tag{4}$$

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)[\cos(2\pi nk/N) + j\sin(2\pi nk/N)] \tag{5}$$

Notice the difference between DFT and IDFT in equation (2) and (4) , the IDFT Kernel is the complex conjugate of the DFT and the output is scaled by N. W_N^{nk} , the Kernel of the DFT and IDFT is called the Twiddle-Factor and is given by,

In exponential form,

$e^{-j2\pi nk/N}$ for DFT
$e^{j2\pi nk/N}$ for IDFT

In rectangular form,

$\cos(2\pi nk/N) - j\sin(2\pi nk/N)$ for DFT
$\cos(2\pi nk/N) + j\sin(2\pi nk/N)$ for IDFT

Introduction

Within calculating DFT, a complex summation of N complex multiplications is required for each of N output samples. N^2 complex multiplications and N(N-1) complex additions compute an N-point DFT. The processing time required by large number of calculation limits the usefulness of DFT. This drawback of DFT is overcome by a more efficient and fast algorithm called Fast Fourier Transform (FFT). The radix-2 FFT computes the DFT in $N*\log_2(N)$ complex operations instead of N^2 complex operations for that of the DFT. (where N is the transform length.)

The FFT has the following preconditions to operate at a faster rate:

- The radix-2 FFT works only on the sequences with lengths that are power of two. The FFT has a certain amount of overhead that is unavoidable, called bit reversed ordering.
- The output is scrambled for the ordered input or the input has to be arranged in a predefined order to get output properly arranged. This makes the straight DFT better suited for short length computation than FFT.

2 FFT Algorithm

2.1 Radix-2 Decimation-In-Time FFT Algorithm

The decimation-in-time (DIT) FFT divides the input (time) sequence into two groups, one of even samples and the other of odd samples. N/2-point DFTs are performed on these sub-sequences and their outputs are combined to form the N-point DFT. First, x(n) the input sequence in the equation (1) is divided into even and odd sub-sequences.

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n) W_N^{2nk} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) W_N^{(2n+1)k} \qquad \text{for k=0 to N-1} \qquad (6)$$
$$= \sum_{n=0}^{\frac{N}{2}-1} x(2n) W_N^{2nk} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) W_N^{2nk}$$

But, $W_N^{2nk} = (e^{-j2\pi/N})^{2nk} = (e^{-j2\pi/(N/2)})^{nk} = W_{N/2}^{nk}$.

By substituting the following in equation (6)

h(n)=x(2n)
g(n)=x(2n+1) ,

equation (6) becomes

$$X(k) = \sum_{n=0}^{N/2-1} h(n) W_{N/2}^{nk} + W_N^k \sum_{n=0}^{N/2-1} g(n) W_{N/2}^{nk} \qquad \text{for k=0 to N-1} \qquad (7)$$
$$= H(k) + W_N^k G(k)$$

Equation (7) is the radix-2 DIT FFT equation. It consists of two N/2-point DFTs H(k) and G(k) performed on the subsequences of even and odd samples of the input sequence x(n), respectively. Multiples of W_N, the Twiddle-Factors are the coefficients in the FFT calculation. Further,

$$W_N^{k+N/2} = (e^{-j2\pi/N})^k \times (e^{-j2\pi/N})^{N/2} = -W_N^k \qquad \text{for k=0 to N/2-1,} \qquad (8)$$

equation (7) can be expressed in two equations.

$$X(k) = H(k) + W_N^k G(k) \qquad (9)$$

$$X(k+N/2) = H(k) - W_N^k G(k) \qquad (10)$$

The decomposition procedure can be continued until two-point DFTs are reached. Figure 1 illustrates the flow graph of a real 8-point DIT FFT.

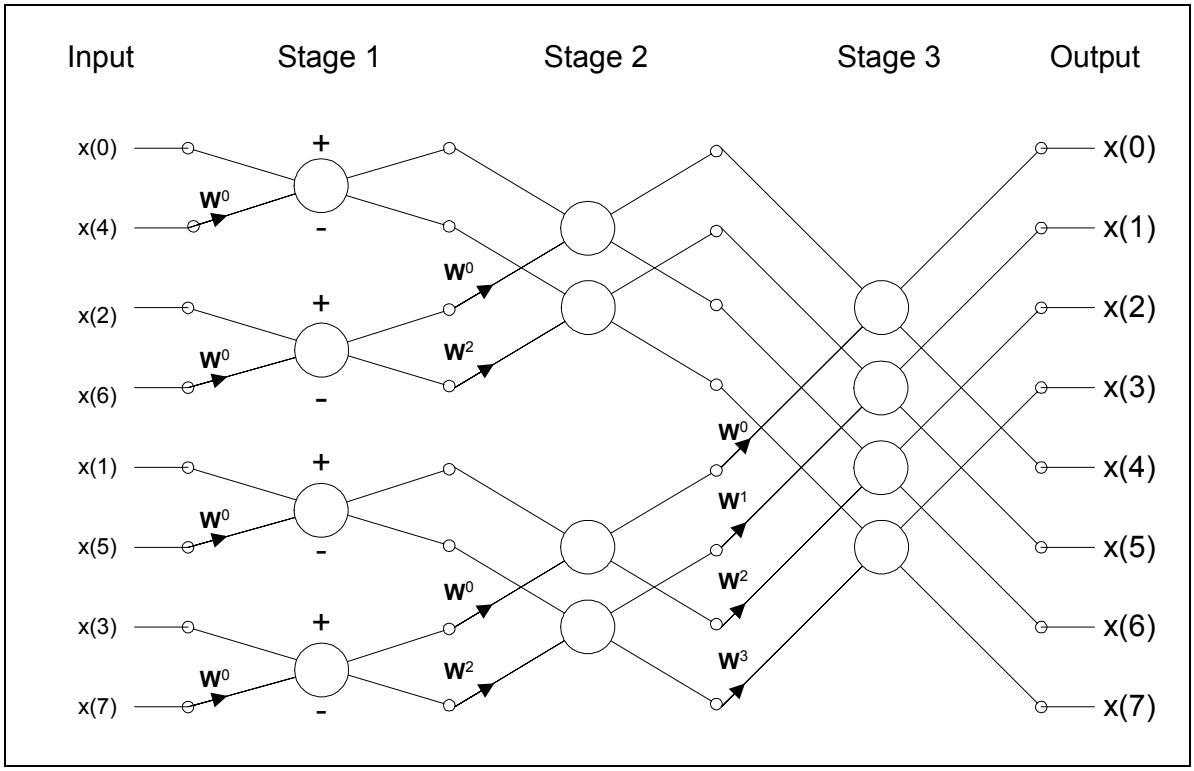

Figure 1: 8-point DIT FFT

Note that the input sequence x(n) in Figure 1 is in the scrabbled order. The same procedure can be repeated with the linear input order. Then we have the alternate form of the DIT FFT shown in Figure 2, where the output sequence X(n) is rearranged to appear in bit-reversed order. The only difference between Figure 1 and Figure 2 is a rearrangement of the butterflies and the computational load and final results are identical. In the application note the implementation of real-valued forward FFT is based on Figure 2 .

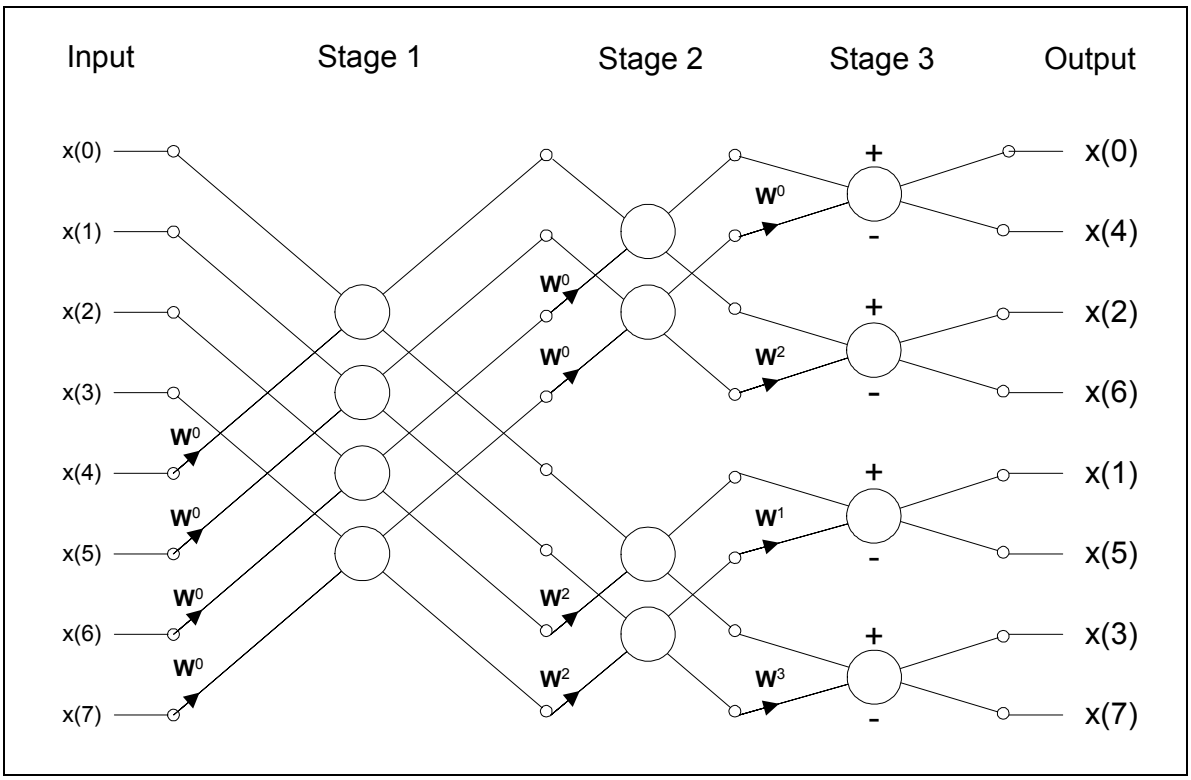

Figure 2: Alternate form of 8-point DIT FFT

2.2 Radix-2 Decimation-In-Frequency FFT Algorithm

A second variant of the radix-2 FFT is the decimation-in-frequency algorithm. In order to get this algorithm, we split the input sequence into the first and second halves and write the DFT as

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

$$= \sum_{n=0}^{\frac{N}{2}-1} x(n) W_N^{nk} + \sum_{n=0}^{\frac{N}{2}-1} x(n+N/2) W_N^{(n+1N/2)k} \quad , \qquad \text{for k=0 to N.} \tag{11}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} [x(n) + (-1)^k x(n+N/2)] W_N^{2nk}$$

For the even and odd numbered DFT points we get

$$X(2k) = \sum_{n=0}^{N/2-1} [x(n) + x(n+N/2)] W_{N/2}^{nk} \tag{12}$$

$$X(2k+1) = \sum_{n=0}^{N/2-1} [x(n) - x(n+N/2) W_N^n] W_{N/2}^{nk} \tag{13}$$

for k=0 to N/2.

As with the decimation-in-time algorithm, the N-point DFT is decomposed into two N/2-point DFTs. Using the principle repeatedly results in an FFT algorithm where the input values appear in their natural order, but where the output values appear in bit reversed order. The complexity is the same as for the decimation-in-time FFT. Figure 3 shows the flow graph of an 8-point DIF FFT. The comparison of Figure 1 and Figure 3 shows that the two graphs can be viewed as transposed versions of one another.

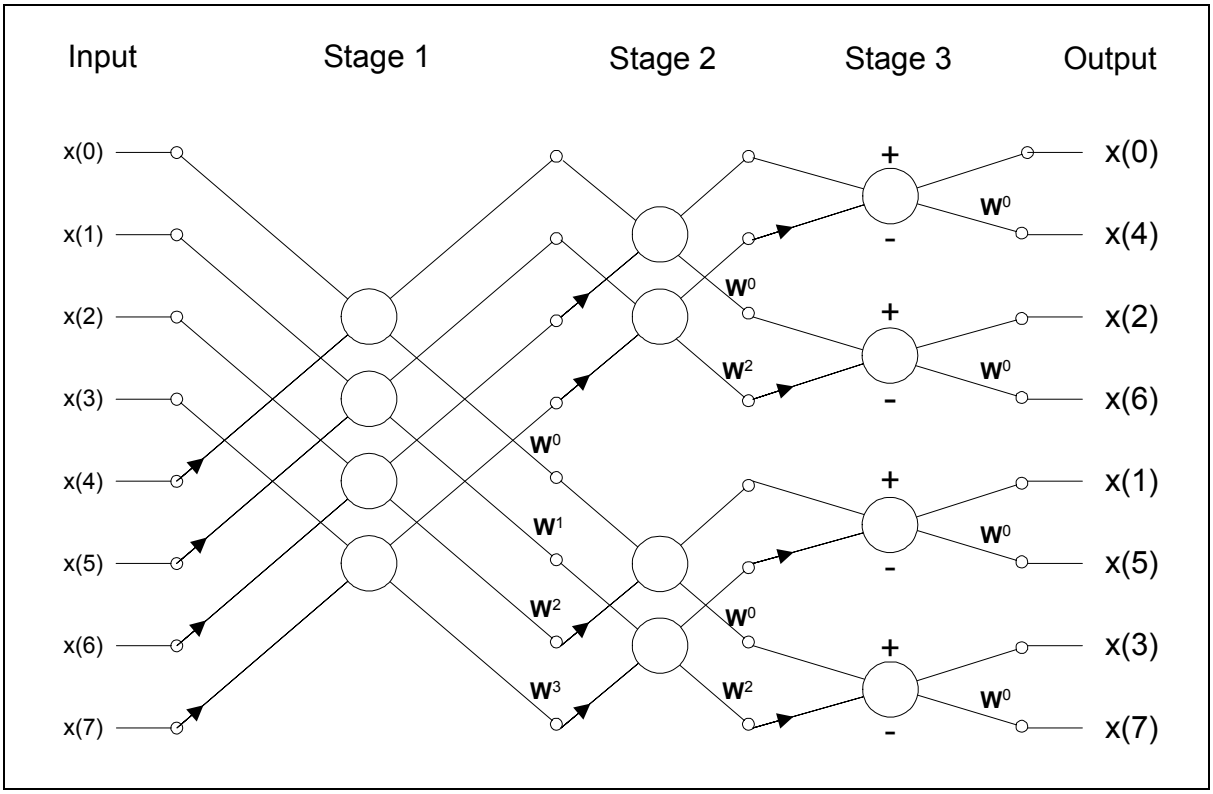

Figure 3: 8-point DIF FFT

Similar to FFT with decimation-in-time, if the inputs in Figure 3 are rearranged in bit-inverse order and the outputs in linear order, we have an alternate implementation for DIF FFT showed in Figure 4 .

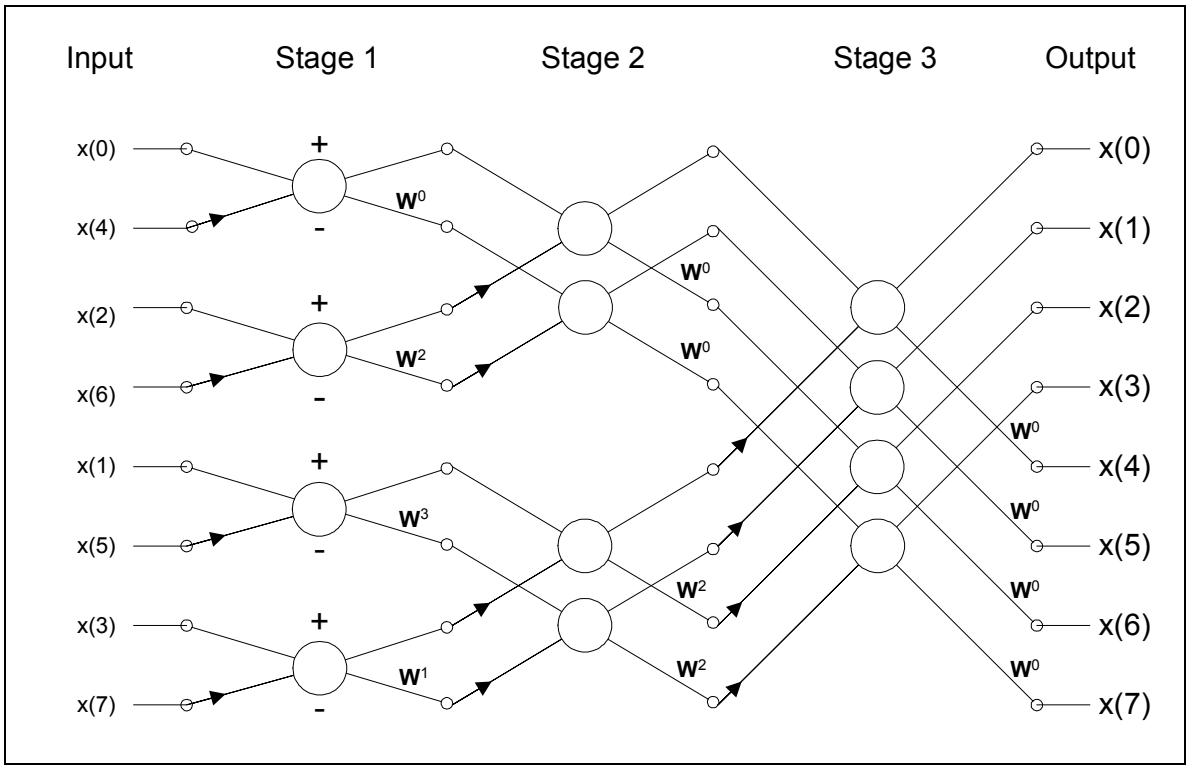

Figure 4: Alternate Form of 8-point DIF FFT

2.3 Complex FFT Algorithm

If the input sequence is complex, according to equation (9) and equation (10) , we can write the complex DFT as

$$X(k) = P(k) + W_N^k Q(k) \tag{14}$$

$$X(k + N/2) = P(k) - W_N^k Q(k) \tag{15}$$

for k=0 to N/2-1, where P(k) and Q(k) are the complex even and odd partial DFTs. As same as real FFT, the complex FFT has also two implementations, i.e. decimation-in-time and decimation-in-frequency complex FFT. Figure 5 shows an 8-point DIT complex FFT implementation. In Figure 5, each pair of arrows represents a Butterfly. The whole of the complex FFT is computed by different patterns of Butterflies. These are called groups and stages.

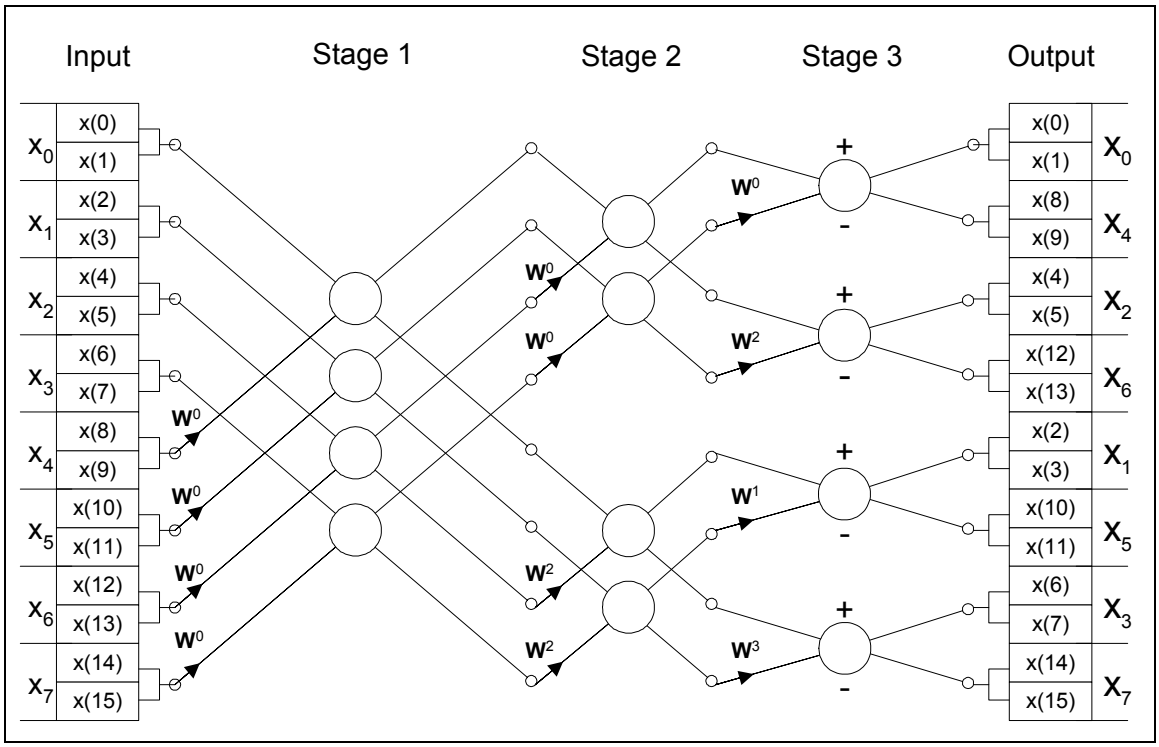

Figure 5: 8-point DIT complex FFT

For 8-point DIT FFT the first stage consists of one groups of four Butterfly, second consists of two groups of two butterflies and third stage has four group of one Butterflies. Each Butterfly is represented as in Figure 6 .

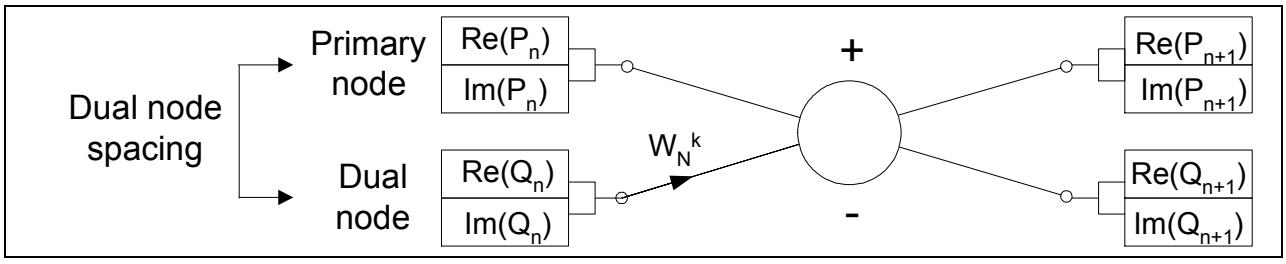

Figure 6: Butterfly of Radix-2 DIT complex FFT

The output is derived as follows

$$P_{n+1} = P_n + Q_n W_N^k \quad (16)$$

$$Q_{n+1} = P_n - Q_n W_N^k \quad (17)$$

where n represents the number of stages.

Of course, the complex FFT can also be implemented with decimation-in-frequency FFT showed in Figure 4 . In this case, an 8-point complex FFT has the implementation structure depicted in Figure 7.

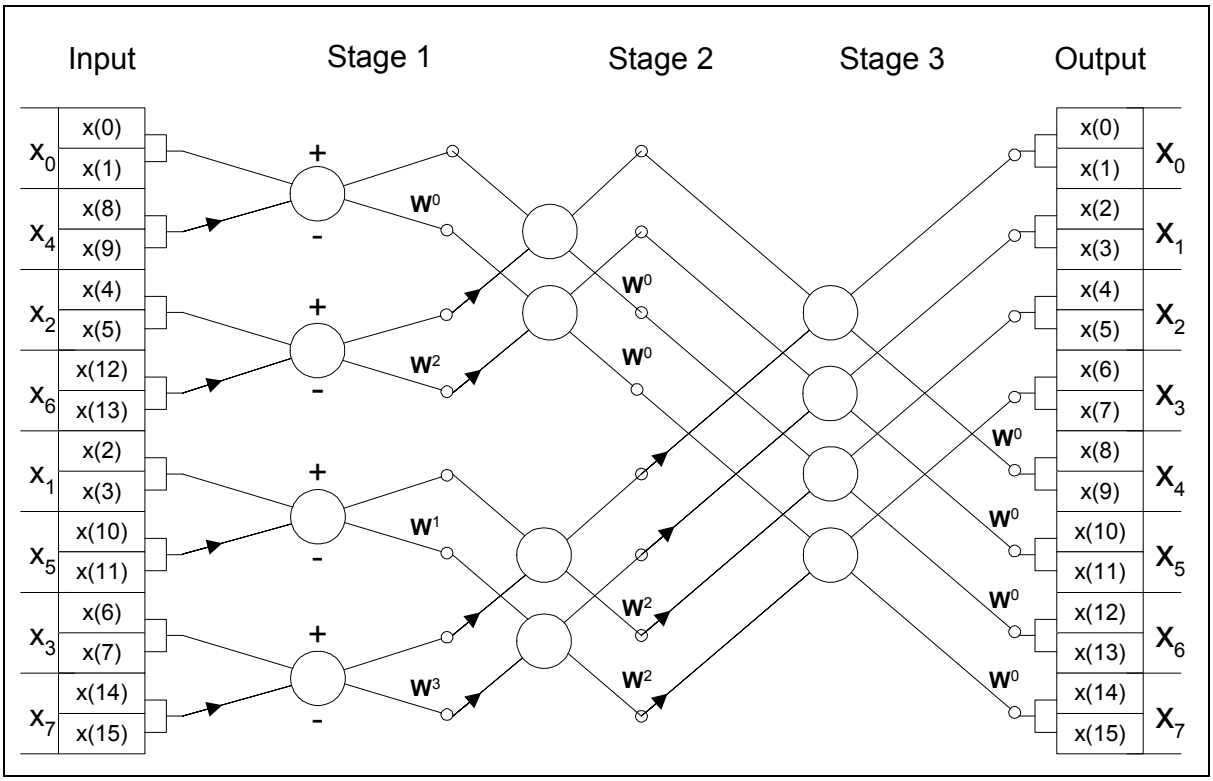

Figure 7: 8-point DIF complex FFT

The corresponding butterfly is illustrated in Figure 8 , in that the input-output relationship writes

$$P_{n+1} = P_n + Q_n \tag{18}$$

$$Q_{n+1} = (P_n - Q_n)W_N^k \tag{19}$$

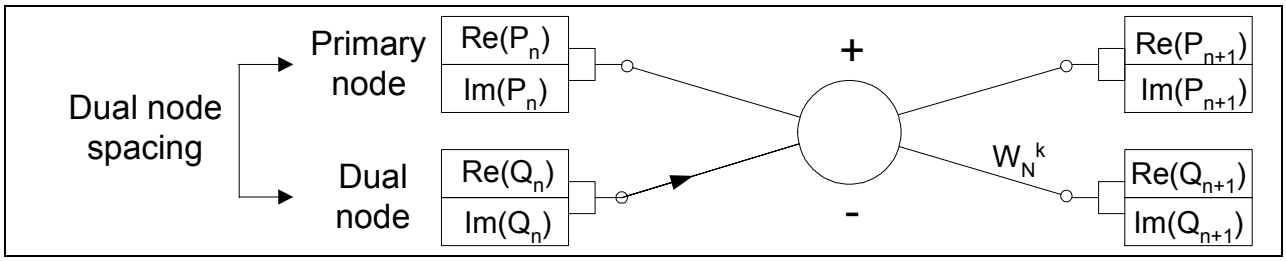

Figure 8: Butterfly of Radix-2 DIF complex FFT

2.4 Calculation of Real Forward FFT from Complete FFT

Having only real valued input data x(n), the computational effort of a N point real FFT can be reduced to a N/2 point complex FFT. Firstly, even indexed data h(n)=x(2n) and odd indexed data g(n)=x(2n+1) are separated. According to equation (9) and equation (10) , The spectrum X(k) can be decomposed into the spectra H(k) and G(k) as follows:

$$X(k) = H(k) + [\cos\tfrac{2\pi k}{N} - j\sin\tfrac{2\pi k}{N}]G(k) \tag{20}$$

for k=0 to N/2-1.

In order to cut the N-point real FFT into N/2-point complex transformation a complex input vector y(n) = h(n)+jg(n) is formed with the index n running from 0 to N/2-1. The real part values are formed by the even indexed input data h(n). The imaginary part is formed by the odd indexed input data g(n). Then y(n) is transformed into the frequency domain resulting in a spectrum consisting of the spectra H(k) and G(k).

$$Y(k) = H(k) + jG(k) = \mathrm{Re}\{Y(k)\} + j\,\mathrm{Im}\{Y(k)\} \tag{21}$$

Now the complex spectra H(k) and G(k) have to be extracted out of the complex spectrum Y(k). By employing symmetry relations, the spectra H(k) and G(k) can be derived from the spectrum Y(k) as follows:

$$\mathrm{Re}\{H(k)\} = \frac{\mathrm{Re}\{Y(N/2-k)\} + \mathrm{Re}\{Y(k)\}}{2} \tag{22}$$

$$\mathrm{Im}\{H(k)\} = \frac{\mathrm{Im}\{Y(k)\} - \mathrm{Im}\{Y(N/2-k)\}}{2} \tag{23}$$

$$\mathrm{Re}\{G(k)\} = \frac{\mathrm{Im}\{Y(k)\} + \mathrm{Im}\{Y(N/2-k)\}}{2} \tag{24}$$

$$\mathrm{Im}\{G(k)\} = \frac{\mathrm{Re}\{Y(N/2-k)\} - \mathrm{Re}\{Y(k)\}}{2} \tag{25}$$

Therefore, computing an N-point real forward FFT has the following steps:

1. Generating the N/2 point complex sequence y(n)=x(2n)+jx(2n+1),
2. Computing the complex spectra Y(k) using complex FFT,
3. Extracting H(k) and G(k) from computed Y(k) according to equations (22)-(25) ,
4. Calculating the first N/2+1 points of X(k) based on equation (20) using H(k) and G(k),
5. Using the symmetry relation of the spectra of the real sequence to get the complete N-point X(k).

2.5 Calculation of Real Inverse FFT from Complete FFT

Using the algorithm in section 2.4 we can calculate an N-point real-valued FFT through N/2-point complex FFT. Now we present the corresponding inverse FFT algorithm to repeat the original real-valued input sequences based on the FFT spectrum. Usually an N-point real-valued forward FFT produces N-point complex spectrum. To get the original real-valued sequences one can simply input this N-point complex spectrum to an inverse FFT. But it needs N-point real inverse FFT. With the symmetry properties of real-valued FFT an N-point real inverse FFT can be reduced to N/2-point complex inverse FFT.

This can be realized through two steps. In the first step the real FFT spectrum will be unpacked to the corresponding complex spectrum similar with the unpack stage in the forward FFT. Then, the results are inputted into an N/2-point inverse complex FFT to get real-valued sequences.

In the unpack stage only the first (N/2+1)-point spectrum are needed because the first (N/2+1)-point spectrum of an N-point real-valued FFT contain all information. According to equation (20) and the symmetry properties of H(k) and G(k), we have

$$\mathrm{Re}\{H(k)\} = \frac{\mathrm{Re}\{X(k)\} + \mathrm{Re}\{X(N/2-k)\}}{2} \tag{26}$$

$$\mathrm{Im}\{H(k)\} = \frac{\mathrm{Im}\{X(k)\} - \mathrm{Im}\{X(N/2-k)\}}{2} \tag{27}$$

for k=0 to N/2-1.

Defining

$$G'(k) = X(k) - H(k) \ , \tag{28}$$

and replacing (28) into (27) we get

$$\mathrm{Re}\{G'(k)\} = \frac{\mathrm{Re}\{X(k)\} - \mathrm{Re}\{X(N/2-k)\}}{2} \tag{29}$$

$$\mathrm{Im}\{G'(k)\} = \frac{\mathrm{Im}\{X(k)\} + \mathrm{Im}\{X(N/2-k)\}}{2} \tag{30}$$

Finally we have the expression of the complex spectrum Y(k) from equation (21)

$$\mathrm{Re}\{Y(k)\} = \mathrm{Re}\{H(k)\} - \sin\tfrac{2\pi k}{N} \cdot \mathrm{Re}\{G'(k)\} - \cos\tfrac{2\pi k}{N} \cdot \mathrm{Im}\{G'(k)\} \tag{31}$$

$$\mathrm{Im}\{Y(k)\} = \mathrm{Im}\{H(k)\} + \cos\tfrac{2\pi k}{N} \cdot \mathrm{Re}\{G'(k)\} - \sin\tfrac{2\pi k}{N} \cdot \mathrm{Im}\{G'(k)\} \tag{32}$$

where K=0 to N/2-1.

Summarily, computing an N-point real inverse FFT has the following steps:

1. Extracting H(k) according to equation (29) ,
2. Extracting G'(k) according to equation (31) ,
3. Computing the complex spectrum Y(k) based on H(k) and G'(k) using equation (32) ,
4. Calculating N/2-point inverse complex FFT with input Y(k).

3 XC2000/XE166 Implementation Note

3.1 Organization of FFT Functions

In the application note the radix-2 FFT is implemented. Basically there are the following three kinds of functions:

- Bit reverse function (Bit_reverse.asm)
- Floating point to 1Q15 format function (FloatToQ15.asm)
- FFT kernel function

The first two functions will be used as subroutines called by FFT kernel function. The subroutine Bit_reverse.asm is used for bit reversing the binary presentation of the input indices to get the output data indices. FloatToQ15.asm is used to change the floating point format to 1Q15 fractional format. The kernel FFT realizes the kernel FFT algorithm that may be complex and real FFT. Butterflies are implemented in the form of macros.

The kernel FFT implementation in this application note is based on an AppNote for C166 microcontroller, in which an implementation of a real-valued 1024-point radix-2 decimation-in-time forward FFT for C166 microcontroller family is described. However, there are many basic differences between the two implementations. Firstly, the FFT implementation for XC2000/XE166 is not only C-callable but also Assembler-callable, while the early implementation for C166 is only Assembler-callable. Secondly, the FFT implementation for XC2000/XE166 is performed with the new DSP MAC instruction set, while the early implementation for C166 has only used the normal instruction set. Therefore, the FFT implementation for XC2000/XE166 is a more optimal implementation in comparison with early implementation. Thirdly, the size of input data for the new FFT implementation for XC2000/XE166 can be changed from 2 to 2048 points, while the early implementation is only suitable for 1024-point FFT.

This application note provides not only forward FFT but also inverse FFT implementation as well as their implementation theory basis.

3.2 Implementation of Real Forward FFT

A real valued N point FFT can be reduced to an N/2 point complex FFT followed by an unweave phase in order to compute the N/2 point spectrum. The N/2 complex FFT consists of $\log_2(N/2)$ stages and each stage calculates N/4 butterflies.

The input data is stored in the vector FFT_in that consists of N 16 bit words and is defined in C main function. Since we perform an in-place FFT, the input data field will be overwritten by the output data. For providing the trigonometric functions, the precomputed sinus and cosine values are stored in memory. Because the input data and the trigonometric function values are represented by a 15 bit fixed-point fraction in two's complement (1Q15 format), the floating-point sinus and cosine values have to be changed into 1Q15 format with the routine FloatToQ15. To rearrange the bit reversed output of the complex FFT and to calculate the twiddle factor Wk, a bit reversal index table has been precomputed.

Regarding the N/2 point complex FFT, the number of twiddle factors amounts N/2. However, due to the symmetry only the first N/4 is used. The implementation consists of $\log_2(N/2)$ stages. Each stage contains basically three loops, i.e. Outloop, Mitloop and Inloop. One stage has one Outloop and N/2 Inloops, but different Mitloops due to different twiddle factors. Each Inloop implements one butterfly with a twiddle factor. For example, an 8-point DIT complex FFT in Figure 5 has the following structure:

Stage 1 (Outloop_1): 4 butterflies with W_0
Stage 2 (Outloop_2): 2 butterflies with W_0
2 butterflies with W_2
Stage 3 (Outloop_3): 1 butterfly with W_0

1 butterfly with W_1
1 butterfly with W_2
1 butterfly with W_3

The output of the decimation-in-time FFT shows a bit reversed order that has to be ordered to calculate the final frequency spectrum. Supposing the input data is in a sequential order, the indices of the output data can be easily computed by bit reversing the binary presentation of the input indices. The table below gives an example of the bit reversal for an 8-point FFT.

Order of input data		Order of output data	
Index	binary	binary	index
0	000	000	0
1	001	100	4
2	010	100	2
3	011	110	6
4	100	001	6
5	101	101	5
6	110	011	3
7	111	111	1

The second part of the program unweaves the bit reversed output of the N/2 point FFT to extract the N point real value FFT.

3.3 Implementation of Real Inverse FFT

From section 2.5 we know that the N-point real inverse FFT can be realized by N/2 point complex inverse FFT. Different from forward FFT, the unpack stage will be performed at beginning to get the N/2 point complex input spectra according to N point real FFT spectra.

The input data is the first (N/2+1)-point real FFT spectra and stored in the vector with size N/2+1 that consists of N+2 16 bit words and is defined in C main function. After unpacking the N/2 point complex spectra are stored in vector FFT_out. Note that the input data should be by 1/N scaled real FFT spectra. All data has 1Q15 format. Here we use the name real inverse FFT, which doesn't mean that the input data is a real value. Actually here the input data is a complex value coming from a real-valued FFT. The word real indicates that the input value comes from a real-valued forward FFT instead of a complex forward FFT.

The butterfly of an inverse FFT has the same structure as a forward FFT except for the twiddle factor. The twiddle factor of the inverse FFT butterfly has the form instead of . Similarly with forward FFT the inverse FFT can be also implemented with DIT and DIF. For example, for an 8-point real inverse FFT, if the DIF implementation in Figure 7 is used, there are following structures:

Stage 1 (Outloop_1): 1 butterfly with W_0
1 butterfly with W_1

1 butterfly with W_2
1 butterfly with W_3
Stage 2 (Outloop_2): 2 butterflies with W_0
2 butterflies with W_2
Stage 3 (Outloop_3): 4 butterflies with W_0

In this case the output shows the linear order while the input has bit-reversed order. The corresponding butterfly has the structure showed in Figure 8.

4 Description of Implemented Functions

4.1 Binary Bit Reverse of Input Data

This routine is a subroutine in the DIT FFT implementation packet and used to obtain bit reversed data in respect to the input data. The bit reversed data is also stored in the vector X.

For an 8 bit input data there is following algorithm:

before bit reverse: b7.b6.b5.b4.b3.b2.b1.b0
after bit reverse: b0.b1.b2.b3.b4.b5.b6.b7

Pseudo code:

```
{
    ; X = input/output data vector
    ; N = the size of the vector, N=2^n (n<12)

    DataS*  X;              //input/output vector
    DataS   N;              //size of vector N
    DataS   i, k, n;

    for(k=0; k<N; k++)
    {
      temp = 0;
      for(i=15; i>0; i--)
         temp = temp + (X(k)<<i)>>(15-i);
    //write the output into X
      X(k) = temp;
    }

    n = (DataS)log10(N);

    Y = n;
    return Y;          //Filter Output returned

}
```

Memory usage is showed in the Figure 9.

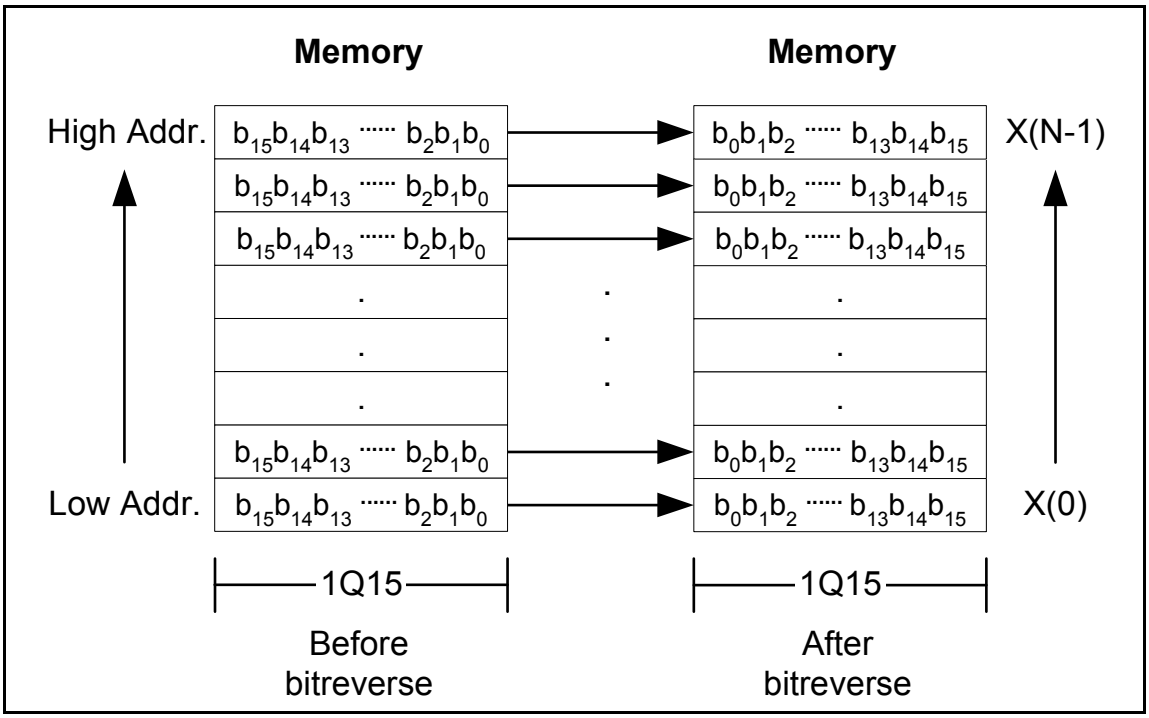

Figure 9: Memory location of bit inverse function

4.2 Floating to Fixed Point Format 1Q15

This routine is a subroutine in the DIT FFT implementation packet and used to change the floating point data into a 1Q15 fixed point format.

The floating point format has the structure:

1. word: s e e e e e e e e mmmmmmm
2. word: mmmmmmmmmmmmmmmm

where s =sign, e=exponent, m=mantissa. After format change we have the 1Q15 fixed point data with the structure:

s. b1 b2 b3 b4 ..... b15
sig. 2^{-1} 2^{-2} 2^{-3} 2^{-4} ..... 2^{-15}

For example:

binary	*hex*	*value*
0111 1111 1111 1111	7FFF	+1
0110 0000 0000 0000	6000	+0.75
1010 0000 0000 0000	A000	-0.75
1000 0000 0000 0000	8000	-1

Figure 10 shows the register usage with classic Tasking compiler.

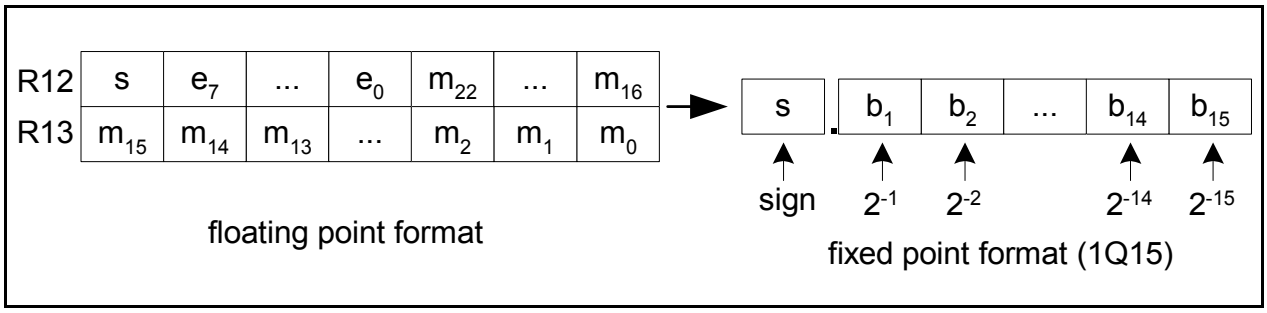

Figure 10: Registers used in format transfer

4.3 Real Forward Radix-2 Decimation-in-Time FFT

This function computes the N-point real forward radix-2 decimation-in-time Fast Fourier Transform on the given N-point real input array. The detailed implementation is given in the section 3.2.

The function is implemented as a complex FFT of size N/2 followed by a unpack stage to unpack the real FFT results. Normally an FFT of a real sequence of size N produces a complex sequence of size N or 2N real numbers that will not fit in the input sequence. To accommodate all the results without requiring extra memory locations, the output reflects only half of the complex spectrum plus the spectrum at the Nyquist point (N/2). This still provides the full information because an FFT of a real sequence has even symmetry around the Nyquist point.

Pseudo code:

```
{
    DataS*  table;          //sinus and cosine table
    DataS*  x;              //16 bit real input vector
    CplxS*  X;             //FFT output vector
    CplxS   P(n), P(n+1), Q(n), Q(n+1), Y(N/2), H(N/2), G(N/2);
    DataS   k,i,n;

    //building the complex sequences P(n) and Q(n)
    Q(n) = x(2n) + jx(2n+1);
    P(n) = x(2n+N/4) + jx(2n+N/4+1);

    //Outloop = 1 to exp=log2(N/2)
    for(k=0; k++; k<exp)
    {
      //Midloop = 1 to N/4 according to which stage
      for(i=0; i<Midloop; i++)
      {
         //Inloop = N/2 to 1 (number of butterflies)
         for(n=0; n<Inloop; n++)
         {
            P(n+1) = Re(P(n))+Re(Q(n))*cos(X)+Im(Q(n))*sin(X)
                     +j[Im(P(n))-Im(Q(n))*cos(X)-Re(Q(n)*sin/X)];
            Q(n+1) = Re(P(n))-Re(Q(n))*cos(X)-Im(Q(n))*sin(X)
                     +j[Im(P(n))-Im(Q(n))*cos(X)-Re(Q(n)*sin/X)];
         }
         //if all elements processed, jump out of the Midloop
      }
    }
    //output the first half of N/2 point complex FFT values Y,
    //Extracting H and G from computed Y according to equation (21),
    //Calculating the first N/2 points of X based on equations (22)-(25)using H
    //and G
}
```

Memory usage is in Figure 11.

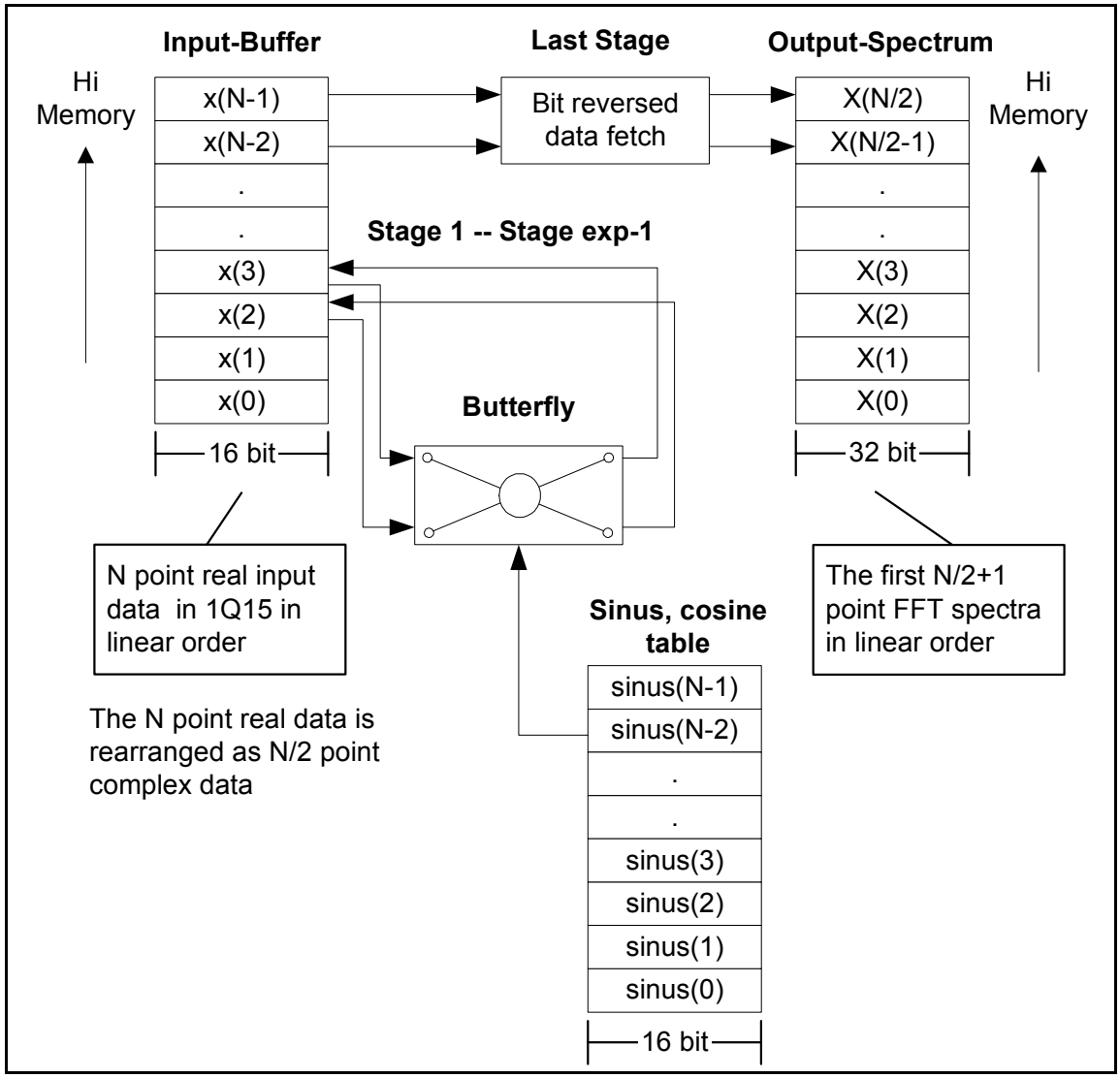

Figure 11: Memory location of real forward FFT

4.4 Real Inverse Radix-2 Decimation-in-Frequency FFT

This function computes the N-point real inverse radix-2 Fast Fourier Transform with decimation-in-frequency on the given N-point FFT spectra. The detailed implementation is given in the section 3.3 .

The function is implemented as an unpack stage followed by a complex inverse FFT of size N/2. The unpack stage aims to unpack the N-point real FFT as N/2-point complex FFT. The N/2-point complex inverse FFT is implemented with decimation-in-frequency structure showed in Figure 7, where the butterfly W_N^{nk} should be replaced by W_N^{-nk}.

Pseudo code:

```
{
    CplxS*  X;              //input FFT spectra
    DataS*  x;              //16 bit real output vector
    CplxS   P(n), P(n+1), Q(n), Q(n+1), Y(N/2), H(N/2), G'(N/2);
    DataS   k,i,n;

    //extracting H(k) according to Hyperlinkquation
```

```
    Re{H(n)} = {Re{X(n)} + Re{X(N/2-n)}}/2;
    Im{H(n)} = {Im{X(n)} - Im{X(N/2-n)}}/2;

    //extracting G'(k) according to Hyperlinkquation
    Re{G'(n)} = {Re{X(n)} - Re{X(N/2-n)}}/2;
    Im{G'(n)} = {Im{X(n)} + Im{X(N/2-n)}}/2;

    //computing the complex spectrum Y according to Hyperlinkquation
    Re{Y(n)} = Re{H(n)}-sin(2*pi*n/N)*Re{G'(n)}-
               cos(2*pi*n/N)*Im{G'(k)};
    Im{Y(n)} = Im{H(n)}+cos(2*pi*n/N)*Re{G'(n)}-
               sin(2*pi*n/N)*Im{G'(k)};
    //define
    P(n) = Y(2n); Q(n) = Y(2n+1);

    //N/2-point inverse complex FFT
    //Outloop = 1 to exp=log2(N/2)
    for(k=0; k++; k<exp)
    {
      //Midloop = 1 to N/4 according to which stage
      for(i=0; i<Midloop; i++)
      {
        //Inloop = 1 to N/4 (number of butterflies)
        for(n=0; n<Inloop; n++)
        {
         P(n+1) = Re{P(n)}+Re{Q(n)}+j[Im{P(n)}+Im{Q(n)}];
         Re{Q(n+1)} = (Re{P(n)}-Re{Q(n)})*cos(x)-
                      (Im{Q(n)}-Im{Q(n)}*sin(X);
         Im[Q(n+1)} = (Im{P(n)}-Im{Q(n)})*cos(X)+
                      (Re(Q(n)-Re{Q(n)})*sin/X);
        }
         //if all elements processed, jump out of the Midloop
      }
    }
}
```

Memory usage is showed in Figure 12.

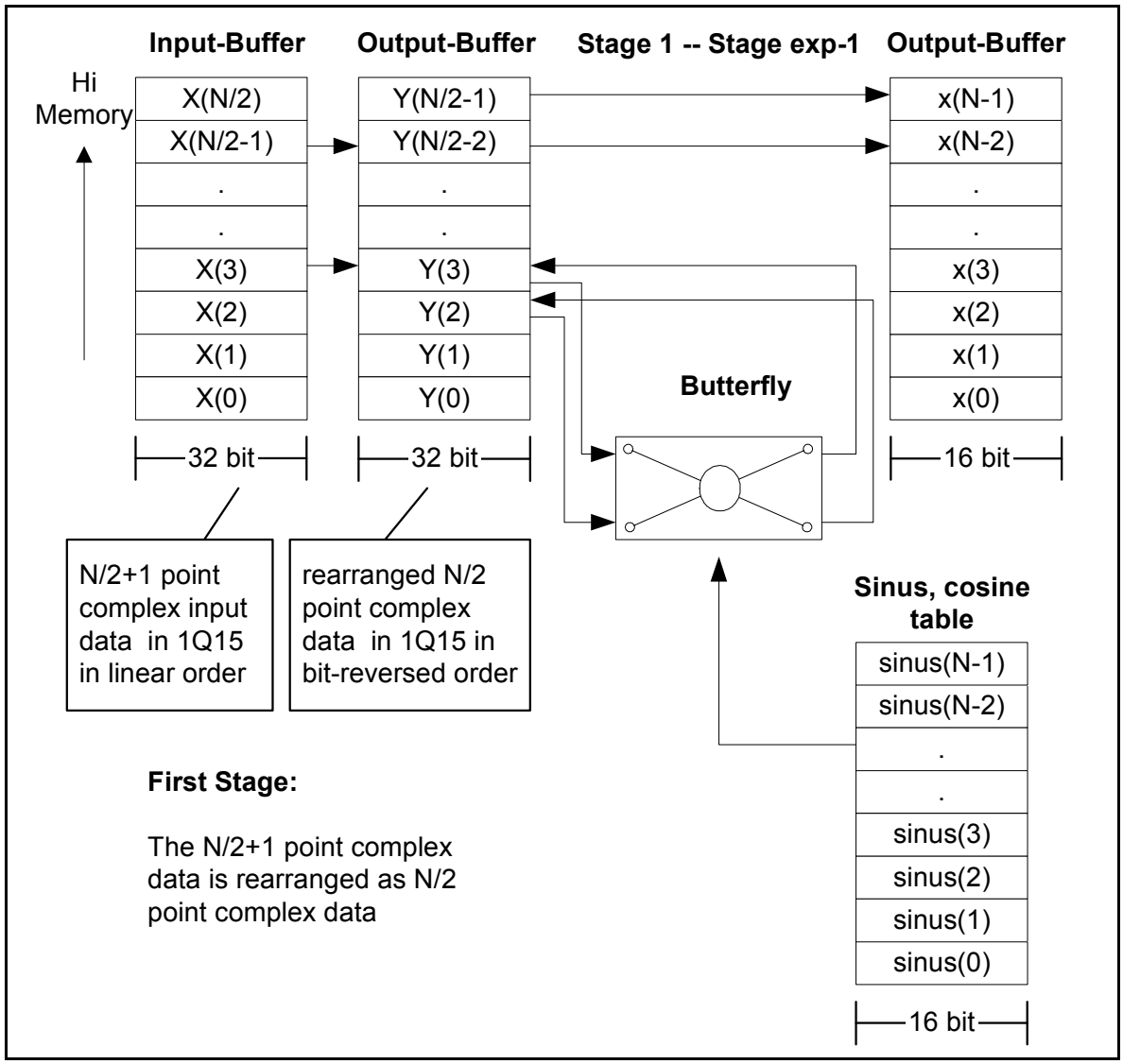

Figure 12: Memory usage of inverse FFT

4.5 Usage of FFT Functions

The functions related with FFT implementation are included in a DSP library freely provided by Infineon. By using the FFT implementation you just need to call the library functions as shown in following example:

```
#include "DspLib.h"
#include <stdio.h>
#include <stdlib.h>

#defineN_x      16

//input data vector in 1Q15 format
DataS  x[N_x] ={
 21061,/* 0 */  -3624,  /* 1 */    7564, /* 2 */  19130,  /* 3 */  27641,  /*    4
*/
 15609,/* 5 */ -21215,  /* 6 */   -6180, /* 7 */  28536,  /* 8 */  27319,  /*    9
*/
```

```
 -5880,/* 10 */ 25795,  /* 11 */ -28972, /* 12 */ -9642,  /* 13 */ 20521,  /*   14
*/
-32119,/* 15 */
};


void main()
{
   DataS  i, exp;
   DataS  X[N_x+2], table[3*N_x/4];
   DataS  index[N_x/2];
   float  y;

//generate trigonometric function table
   for(i=0; i<3*N_x/4; i++)
   {
      if(i<=N_x/2)
      {//change the format from floating to 1Q15
         y = FloatTo1Q15(2.*i/N_x);
         table[i] = Sine(y);
      }
      else
      {//change the format from floating to 1Q15
         y = FloatTo1Q15(2.*(i-N_x)/N_x);
         table[i] = Sine(y);
      }
   }

//generate the reverse index table
   for(i=0; i<N_x/2; i++)
      index[i] = i;

   exp = Bit_reverse(index, N_x/2);
   exp = exp+1;  //N_x = 2^exp

   // To point to the memory address, it needed to multiplay the indices by 4
   for(i=0; i<N_x/2; i++)
      index[i] = index[i] * 4;

/************ Perform the Fourier Transform *************/

//call real_DIT_FFT routine to perform real forward Fourier Transform
   real_DIT_FFT(x, index, exp, table, X);

}
```

In the above example the trigonometric function table can be pre calculated and stored in the flash for a fixed N. Then we can save the executing time for table calculation. The index table can be also done similar.

5 Conclusion

In this application note we have introduced different implementation structures of FFT, and their realizations based on XC2000/XE166 microcontrollers. Here we give the performance of the implementations. Following table lists the benchmarks of each implementation on XC2000/XE166, where N indicates the points of FFT transformation.

Table 1: Benchmark of FFT Implementation on XC2000/XE166

Function name	Cycles		Code Size
Real Forward Radix-2 Decimation-in-Time FFT	FFT kernel (FFT_cycle)	$\sum_{n=0}^{exp-2}\{6+2^n\times 29+\frac{N}{4}\times 36\}$ where exp=Log_2N.	442 Bytes
	Unpack stage (U_cycle)	71 + (N/4-1)*44	
	Total	**39 + FFT_cycle + U_cycle** Exemples: N = 8 : cycle = 380 N = 16 : cycle = 896 N = 1024: cycle = 109131	
Real Inverse Radix-2 Decimation-in-Frequency FFT	IFFT kernel (IFFT_cycle)	$\sum_{n=0}^{exp-2}\{7+2^n\times 28+\frac{N}{4}\times 33\}$ where exp=Log_2N.	366 Bytes
	Unpack stage (U_cycle)	6 + 48*N/4	
	Total	**32 + IFFT_cycle + U_cycle** Examples: N = 8 : cycle = 354 N = 16 : cycle = 833 N = 1024: cycle = 77670	

Infineon provides the free source code for all FFT implementations introduced in this application note. The routines are realized and optimized on XC2000/XE166. You can download the source code directly from Infineon homepage: http://www.infineon.com/C166DSPLIB.

6 Reference

[1] Guangyu Wang, User's Manual of XC166Lib, A DSP Library for XC166 Microcontroller Family, V1.1, Feb. 2004.

[2] 16-Bit DSP Library for 16-Bit Microcontroller with C166S V2 Core, V1.1, Feb., 2004, http://www.infineon.com/C166DSPLIB

[3] C166S V2 Core User's Manual, V1.7, January 2001.

[4] XC2200 User's Manual, V1.0, June 2006.

[5] Guangyu Wang, "Implementation of DSP Algorithms on Microcontrollers with MAC Unit", Elektronik, March 2005, Page 80-85.

[6] Guangyu Wang, "DSP Optimization Based on XC166 Microcontroller Architecture", Electronik, June 2006, Page 50-55.

[7] AP16113, "Fast Fourier Transform Based on XC2000 & XE166 Microcontroller Families".

http://www.infineon.com

Published by Infineon Technologies AG