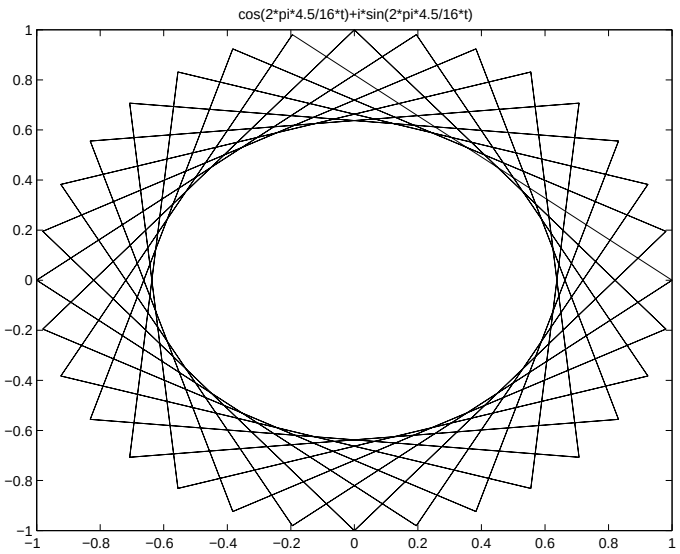
FFT, REALIZATION AND IMPLEMENTATION IN FPGA

Griffith University/Ericsson Microwave System AB 2000/2001

by

Magnus Nilsson



Supervisor, EMW: Rune Olsson

Supervisor, GU: Prof. Kuldip K. Paliwal

Signal Processing Laboratory, School of Microelectronic Engineering, Griffith University

Brisbane/Gothenburg 2000/2001

Uppgjord (även faktaansvarig om annan) - *Prepared (also subject responsible if other)*		Nr - *No.*		
EMWMSNN(Magnus Nilsson)		FX/D-2001:007		
Dokansv/Godkänd - *Doc respons/Approved*	Kontr - *Checked*	Datum - *Date*	Rev	File
EMW/FX/DC(Anders Wanner)		2001-02-12	A1	

Abstract

Ericsson Microwave Systems develops radar systems for military and civilian applications. In military environments high radar resolution and long range are desired, thus high demands must be met by the generated and transmitted radar signal.

In this report the design of a parallel Radix-4 Fast Fourier Transform algorithm is described. A theoretical review regarding Fourier theory and Fast Fourier Transform (Radix-2 and Radix-4) is done.

A complex parallel Radix-4 algorithm is simulated, implemented and realized in hardware using VHDL and a Xilinx Virtex-E 1000 FPGA circuit.

The VHDL code was simulated and synthesized in Ease and Synplify environment. The design was verified and the output was identical with the Matlab and VHDL simulations, proving speed improvements due to a parallel approach.

Uppgjord (även faktaansvarig om annan) - *Prepared (also subject responsible if other)*		Nr - *No.*		
EMWMSNN(Magnus Nilsson)		FX/D-2001:007		
Dokansv/Godkänd - *Doc respons/Approved*	Kontr - *Checked*	Datum - *Date*	Rev	File
EMW/FX/DC(Anders Wanner)		2001-02-12	A1	

Preface

This thesis is a part of my education towards a Master degree in Computer and Information Engineering at Griffith University, Brisbane, Australia. Project 1,2 and 3. MEE7097,MEE7098 and MEE7099.

The work has been done at Ericsson Microwave System AB in Mölndal Sweden, at the department FX/D

I would like to thank the following people who has been of great help to me during my work.

My supervisor Rune Olsson, EMW.

My manager Håkan Olsson/Anders Wanner, EMW.

Prof. Kuldip K. Paliwal, supervisor GU.

Daniel Wallström, EMW, for help with VHDL.

Dennis Eriksson, EMW, for help with Logical Analyser/Pattern generator.

Nils Dagås and Gabriel Gitye, EMW, for help with Matlab.

I would also like to thank the remaining staff at EMW/FX and GU who have been helpful to me.

Contents Page

1 INTRODUCTION

1.1 BACKGROUND

To implement the DFT (FFT) in hardware (real time system) required expensive solution often with ASIC (Application Specific Integrated Circuit). With the latest generation of FPGA (Field Programmable Gate Arrays) it is possible to implement very large amounts of logic in a single integrated circuit.

A manufacturer of FPGA named XILINX now has a drop-in module for their FPGAs which can execute a 1024-points FFT. It is interesting to evaluate and develop such a DFT or similar.

1.2 TASK

To study, implement and evaluate the DFT (Discrete Fourier Transform) in FPGA or similar.

1.3 TECHNICAL FUNCTION

The DFT shall collect data, execute a DFT or IDFT and output the data. The implementation shall be optimized on execution time, size (area) and cost.

Uppgjord (även faktaansvarig om annan) - *Prepared (also subject responsible if other)*		Nr - *No.*		
EMWMSNN(Magnus Nilsson)		FX/D-2001:007		
Dokansv/Godkänd - *Doc respons/Approved*	Kontr - *Checked*	Datum - *Date*	Rev	File
EMW/FX/DC(Anders Wanner)		2001-02-12	A1	

2 JEAN-BAPTISTE-JOSEPH FOURIER

The 21:st of March 1768, Jean-Baptiste-Joseph Fourier was born. He was born in poor circumstances in the small village of Auxerre, France.

Jean-Baptiste-Joseph Fourier introduced the idea that an arbitrary function, even a function defined by different analytic expressions in adjacent segments of its range (such as a staircase waveform) could nevertheless be represented by a single analytic expression.

Fourier's ideas encountered resistance at the time but has proven to be a central theorem to many of the later developments in mathematics, science and also engineering. As we all know, it is at the heart of the electrical curriculum today.

Fourier came across the idea in the connection with the problem of flow of heat in solid bodies, including the heat from the earth.

We have learned that Fourier was obsessed with heat, keeping his room really hot, uncomfortably hot for visitors, this when even wearing a heavy coat himself. Some has traced this obsession back to Egypt where he went 1798 with Napoleon on an expedition to civilize the country. By this time Fourier worked with his theories parallel to his official duties as a secretary of the Institut d'Egypte. At the time in Egypt, Fourier came in contact with the English Physisist Thomas Young (1773-7829), father of linearity, with whom he discussed his ideas and worked together on, among other things, the Rosetta Stone.

After returning back to Paris, Fourier had by 1807, despite official duties, completed his theory of heat conduction, which depended on the essential idea of analysing the temperature distribution into spatially sinusoidal componets. He was very criticized for his theory among the french scientists, among them where Biot and Poisson. Even though he was criticized for his theory he received a mathematic prize in 1811 for his heat theory.

The publication of his writing report "Théorie analytique de la chaleur" (The analytical theory of heat) in 1815 was also met with some criticism and this might be seen as an indication of the deep uneasiness about Fourier analysis that was felt by the great mathematicians of that day.

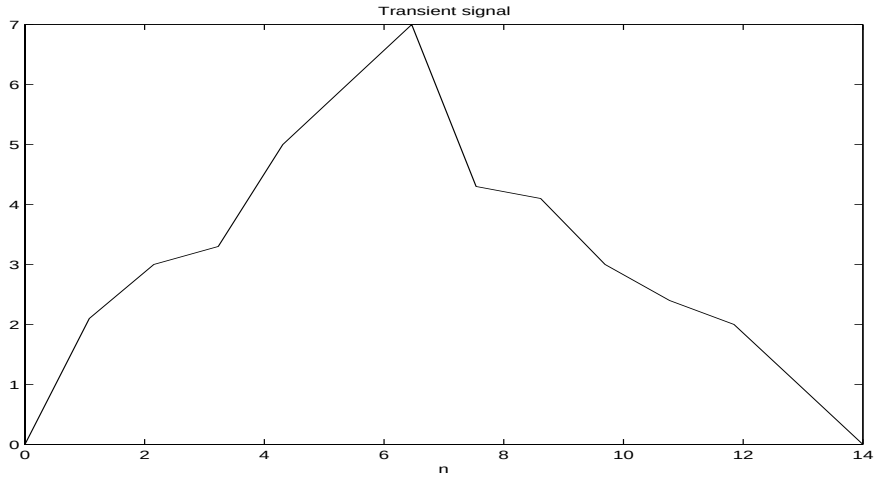
Jean-Baptiste-Joseph Fourier died in Paris the 16:th of May 1830. He never got married.

3 THE FOURIER TRANSFORM

One of todays principal analysis tool in many of todays scientific challenges is the Fourier Transform. Maybe the most known application of this mathematical technique is the analysis of linear time-invariant system. As this might be the most well known application, the Fourier Transform is essentially a universal problem solving technique. Its importance is based on the fundamental property that one can examine a particular relationship from an entirely different viewpoint. Simultaneous visualization of a function and its Fourier Transform is often the key to successful problem solving.

If we define a signal:

$$\lim_{t \to \pm\infty} y(t) = 0 \qquad \text{(EQ 1)}$$



A transient signals spectrum is characterised by the fact that it is continuous, this means that it holds infinite numbers of frequency components, although usually they are in a finite interval.

Mathematically one can define a signal, that vary periodically with time, to be a sum of discrete frequency components, where a simple relationship exists between the frequency components. This can be defined as a formula:

$$y(t) = \frac{1}{2} + \sum_{n=1}^{\infty} \{A_n \cos(n\omega_0 t) + B_n \sin(n\omega_0 t)\}$$

(EQ 2)

where

$$\omega_0 = \frac{2\pi}{T}$$

$$T = Periodicaltime$$

If we look back on our transient signal above, the mathematical consequence will be that the coefficients will be continues functions of the phase w. Equation 3 becomes:

$$y(t) = \int_0^{\infty} A(\omega)\cos(\omega t) + B(\omega)\sin(\omega t)$$

(EQ 3)

If we compare this equation with equation 2 we will see that the constant $A_0/2$ has disappeared, this though $A_0/2$ represents the time mean value of the signal and though it is a transient, the time mean value is zero. The Fourier coefficients A(w) and B(w) is defined by the Fourier integrals:

$$A(\omega) = 2\int_{-\infty}^{\infty} y(t)\cos\omega t dt$$

(EQ 4)

$$B(\omega) = 2\int_{-\infty}^{\infty} y(t)\sin\omega t dt$$

(EQ 5)

where

$$\omega > 0$$

When one wants to calculate the Fourier coefficients in the general case for the signal f(t), one should facilitate the calculations by introduce complex notation.The starting point for complex notation of the Fourier Transform is based on the formulas by Euler which gives a relation between the complex number j and the trigonometrical functions sine and cosine:

$$\cos\alpha = \frac{e^{j\alpha} + e^{-j\alpha}}{2}$$

(EQ 6)

$$\sin\alpha = \frac{e^{j\alpha} - e^{-j\alpha}}{2j}$$

(EQ 7)

Uppgjord (även faktaansvarig om annan) - *Prepared (also subject responsible if other)*		Nr - *No.*		
EMWMSNN(Magnus Nilsson)		FX/D-2001:007		
Dokansv/Godkänd - *Doc respons/Approved*	Kontr - *Checked*	Datum - *Date*	Rev	File
EMW/FX/DC(Anders Wanner)		2001-02-12	A1	

The equations 2, 6 and 7 will give us:

$$y(t) = \int_0^{\infty}\left[\left(A(\omega)\frac{e^{j\omega t}+e^{-j\omega t}}{2}\right)+\left(B(\omega)\frac{e^{j\omega t}-e^{-j\omega t}}{2j}\right)\right]d\omega$$

$$y(t) = \int_0^{\infty}\left(A\frac{e^{j\omega t}+e^{-j\omega t}}{2}+jB\frac{e^{j\omega t}-e^{-j\omega t}}{2}\right)d\omega$$

$$y(t) = \int_0^{\infty}\left(\frac{1}{2}(A-jB)e^{j\omega t}+\frac{1}{2}(A+jB)e^{-j\omega t}\right)d\omega$$

(EQ 8)

If we define Y(w):

$$Y(\omega) = \frac{1}{2}(A(\omega)-jB(\omega))$$

$$Y(-\omega) = \frac{1}{2}(A(\omega)+jB(\omega))$$

Equation 9a and 9b (EQ 9)

Then the equation 8 can be simplified by making the integration over the real area:

$$y(t) = \int_{-\infty}^{\infty} Y(\omega)e^{j\omega t}$$

(EQ 10)

This will then give us, by looking at equation 4 and 5:

$$Y(\omega) = \int_{-\infty}^{\infty} y(t)\cos\omega t\,dt - j\int_{-\infty}^{\infty} y(t)\sin\omega t\,dt$$

$$Y(\omega) = \int_{-\infty}^{\infty} y(t)(\cos\omega t - j\sin\omega t)\,dt$$

$$Y(\omega) = \int_{-\infty}^{\infty} y(t)e^{-j\omega t}$$

(EQ 11)

We will then define Y(w) as the Fourier Transform of y(t) and equation 10 as the Inverse Fourier Transform.

Uppgjord (även faktaansvarig om annan) - *Prepared (also subject responsible if other)*		Nr - *No.*		
EMWMSNN(Magnus Nilsson)		FX/D-2001:007		
Dokansv/Godkänd - *Doc respons/Approved*	Kontr - *Checked*	Datum - *Date*	Rev	File
EMW/FX/DC(Anders Wanner)		2001-02-12	A1	

4 THE DISCRETE FOURIER TRANSFORM

When sampling an arbitrary analog signal the sampled signal can be expressed as:

$$y(t) = y(0)\delta(0) + y(T_S)\delta(t - T_S) + y(2T_S)\delta(t - 2T_S) + \ldots + y((N-1)T_S)\delta(t - (N-1)T_S)$$

(EQ 12)

Where

$$\frac{1}{T_S} \geq 2 f_{max}$$

According to the Nyquist theorem.

The function described above is a sum of time delayed delta functions, each of them with the height $y(nT_S)$. The Fourier Transform for all of those functions equals the Fourier Transform for the undelayed function ie.

$$F\{y(nT_S)\delta(0)\} = y(nT_S)F\{\delta(0)\}$$

multiplied with respectively time delay factor:

$$Y(\omega) = y(0) + y(T_S)e^{-j\omega T_S} + \ldots + y((N-1)T_S)e^{-j\omega(N-1)T_S}$$

$$Y(\omega) = \sum_{n=0}^{N-1} y(nT_S)e^{-j\omega nT_S}$$

(EQ 13)

Since f = w/2pi is a discrete variable when we deal with a sampled signal and only adopt the discrete values:

$$0, \frac{1}{NT_s}, \frac{2}{NT_s}, \ldots, \frac{N-1}{NT_s} = \frac{k}{NT_s}$$

(EQ 14)

Where k = 0,1,2,.....,N-1

Equation 13 becomes:

$$Y\left(\frac{2\pi k}{NT_s}\right) = \sum_{n=0}^{N-1} y(nT_S) e^{-jn\frac{2\pi k}{NT_s}T_S} = \sum_{n=0}^{N-1} y(nT_S)\left(e^{-j\frac{2\pi}{N}}\right)^{nk}$$

(EQ 15)

Once again k = 0,1,2,.....,N-1

If we simplify the equation:

$$W_N = e^{-j\frac{2\pi}{N}}$$

(EQ 16)

We will get the final expression for the Fourier Transform:

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

(EQ 17)

k = 0,1,2,...,N-1

The factor W_N is called the Twiddle Factor.

5 DEVELOPMENT OF THE FAST FOURIER TRANSFORM

5.1 THEORY OF THE FAST FOURIER TRANSFORM

If we consider the equation 17:

$$Y(k) = \sum_{n=0}^{N-1} y(n) W_N^{nk}$$
$$k = 0, 1, 2, \ldots, N-1$$

and we consider the amount of additions and multiplications needed for computing the algorithm. For instance, let us consider the case when N = 4

$$Y(0) = y_0 W^0 + y_1 W^0 + y_2 W^0 + y_3 W^0$$
$$Y(1) = y_0 W^0 + y_1 W^1 + y_2 W^2 + y_3 W^3$$
$$Y(2) = y_0 W^0 + y_1 W^2 + y_2 W^4 + y_3 W^6$$
$$Y(3) = y_0 W^0 + y_1 W^3 + y_2 W^6 + y_3 W^9$$

Or simplified in the compact form:

$$Y(n) = W^{nk} y(k)$$

(EQ 18)

If we then consider the twiddle factor and y(k) we will in the worst case have two complex numbers. This fact will give us N^2 complex multiplications and (N)(N-1) complex additions. Suppose that we have a microprocessor that can do an addition or a multiplication in 1 micro second and that this processor should compute a DFT on a 1 kbyte set of samples. If we have N^2 complex multiplications and (N)(N-1) complex additions ~ $2N^2$ additions and multiplications: 2 x 1024^2 x 1 micro second = 2,1 second. This without taking into consideration the fact that the processor has to update pointers an so on. If we want the analyse to be made in real time we will have to have a distance between the samples that exceeds 2,1 second:

$$\frac{2.1}{1024} = 2.05ms$$

Which gives us:

$$\frac{1}{2.05} = 488Hz$$

as the maximum sampling frequency. By taking the Nyquist theorem in respect, we can not sample a signal that holds a frequency component that exceeds half the maximum sampling frequency = 488Hz/2 = 244Hz.

There are two obvious ways to improve and increase the bandwidth; a faster processor or optimizing the algorithm.

5.2 HISTORY OF THE FAST FOURIER TRANSFORM

In the beginning of the 1960's, during a meeting of the President's Scientific Advisory Commitee, Richard L. Garwin found out that John W. Tukey was writing about the Fourier Transform. Garwin was in his own research in a desperate need for a fast way to compute the Fourier Transform. When questioned, Tukey outlined to Garwin essentially what has led to the famous Cooley-Tukey algorithm.

To get some programming technique, Tukey went to IBM Research in Yorktown Heights and meet there James W Cooley, who quickly worked out a computer program for this algorithm. After a while, request for copies and a write-up began accumulating, and also Cooley was asked to write a paper on the algorithm which in 1965 became the famous paper "An algorithm for the machine calculation of complex Fourier series", that he published together with Tukey.

When publishing this paper, reports of other people using the same technique became known, but the original idea usually ascribe to Runge and König.

The Cooley - Tukey algorithm is also called the Radix - 2 algorithm, due to its signal splitting.

Uppgjord (även faktaansvarig om annan) - *Prepared (also subject responsible if other)*		Nr - *No.*		
EMWMSNN(Magnus Nilsson)		FX/D-2001:007		
Dokansv/Godkänd - *Doc respons/Approved*	Kontr - *Checked*	Datum - *Date*	Rev	File
EMW/FX/DC(Anders Wanner)		2001-02-12	A1	

6 THE RADIX - 2 ALGORITHM

Once again consider equation 17:

$$Y(k) = \sum_{n=0}^{N-1} y(n) W_N^{nk}$$
$$k = 0, 1, 2, \ldots, N-1$$

and we want to analyse the samples:

$$\{y(n)\} = \{y(0), y(1), y(2), \ldots, y(N-1)\}$$

If we consider the possibility to split the samples into odd and even samples:

$$\{y(2n)\} = \{y(0), y(2), y(4)\ldots, y(N-2)\}$$
$$\{y(2n+1)\} = \{y(1), y(3), y(5), \ldots, y(N-1)\}$$

Doing the DFT for those two sequences will give:

$$Y(k) = \sum_{n=0}^{\frac{N}{2}-1} y(2n) W_N^{2nk} + \sum_{n=0}^{\frac{N}{2}-1} y(2n+1) W_N^{(2n+1)k}$$

(EQ 19)

By extracting and simplifying the twiddle factor we are able to simplify even further:

$$Y(k) = \sum_{n=0}^{\frac{N}{2}-1} y(2n) W_N^{2nk} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} y(2n+1) W_N^{2nk}$$

(EQ 20)

$$W^2 = \left(e^{-j\frac{2\pi}{N}}\right)^2 = e^{-j\frac{2\pi}{N}2} = e^{-j\frac{2\pi}{N/2}} = W_{N/2}$$

(EQ 21)

$$Y(k) = \sum_{n=0}^{\frac{N}{2}-1} y(2n)W_{N/2}^{nk} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} y(2n+1)W_{N/2}^{nk}$$

(EQ 22)

By comparing this equation with equation 17 we will find that this by definition are two DFT's with length N/2.

$$Y(k) = D(k) + W_N^k E(k)$$
$$k = 0, 1, 2, \ldots, N-1$$

(EQ 23)

Where D and E represents the sums from equation 22. The computation gain by doing this will be: (as the multiplications in an ordinary DFT = N^2)

$$\left(\frac{N}{2}\right)^2 + \left(\frac{N}{2}\right)^2 = \frac{N^2}{4} + \frac{N^2}{4} = \frac{N^2}{2}$$

This number should be adjusted a bit though the twiddle factor should be multiplied with the odd sum, but this is of a first order of N.

If we study equation 23, we will find that k goes from 0 to N-1 but that D and E represents DFT of N/2. Generally for a DFT of length N is that it is periodical in k with N. This leads to that D and E in equation 23 is periodical with N/2.

$$D\left(k+\frac{N}{2}\right) = D(k)$$

(EQ 24)

$$E\left(k+\frac{N}{2}\right) = E(k)$$

(EQ 25)

Calculating the DFT:

$$Y(0) = D(0) + W_N^0 \times E(0)$$

$$Y(1) = D(1) + W_N^1 \times E(1)$$

$$Y(2) = D(2) + W_N^2 \times E(2)$$

$$\cdots$$

$$\cdots$$

$$Y\left(\frac{N}{2}-1\right) = D\left(\frac{N}{2}-1\right) + W_N^{\frac{N}{2}-1} \times E\left(\frac{N}{2}-1\right)$$

$$Y\left(\frac{N}{2}\right) = D\left(\frac{N}{2}\right) + W_N^{\frac{N}{2}} \times E\left(\frac{N}{2}\right) = D(0) + W_N^{\frac{N}{2}} \times E(0)$$

$$Y\left(\frac{N}{2}+1\right) = D\left(\frac{N}{2}+1\right) + W_N^{\frac{N}{2}-1} \times E\left(\frac{N}{2}+1\right) = D(1) + W_N^{\frac{N}{2}+1} \times E(1)$$

$$\cdots$$

$$\cdots$$

$$Y(N-1) = D(N-1) + W_N^{N-1} \times E(N-1) = D\left(\frac{N}{2}-1\right) + W_N^{N-1} \times E\left(\frac{N}{2}-1\right)$$

(EQ 26)

By symmetrically, the twiddle factor can be expressed as:

$$W_N^{k+\frac{N}{2}} = -W_N^k$$

(EQ 27)

Which gives us:

$$Y(0) = D(0) + W_N^0 \times E(0)$$

$$Y(1) = D(1) + W_N^1 \times E(1)$$

$$\cdots$$

$$Y\left(\frac{N}{2}-1\right) = D\left(\frac{N}{2}-1\right) + W_N^{\frac{N}{2}-1} \times E\left(\frac{N}{2}-1\right)$$

$$Y\left(\frac{N}{2}\right) = D(0) - W_N^0 \times E(0)$$

$$Y\left(\frac{N}{2}+1\right) = D(1) - W_N^1 \times E(1)$$

$$\cdots$$

$$\cdots$$

$$Y(N-1) = D\left(\frac{N}{2}-1\right) - W_N^{\frac{N}{2}-1} \times E\left(\frac{N}{2}-1\right)$$

(EQ 28)

By looking into equation 28 we will find one elementary building-block, the so called FFT-Butterfly.

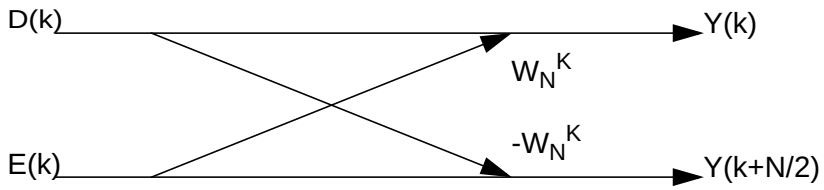


Fig.1. FFT-Butterfly

Which gives the equations:

$$Y(k) = D(k) + W_N^k \times E(k)$$

(EQ 29)

$$Y(k) = D(k) - W_N^k \times E(k)$$

(EQ 30)

Since dividing the sequences into smaller building blocks reduce the amount of multiplications, we will continue to divide the sequences into new blocks.

If we start with equation 22 and divide the sum into four new sums:

$$Y(k) = \sum_{n=0}^{\frac{N}{4}-1} y(4n) W_{N/2}^{2nk} + \sum_{n=0}^{\frac{N}{4}-1} y(4n+2) W_{N/2}^{(2n+1)k} +$$

$$W_N^k \left\{ \sum_{n=0}^{\frac{N}{4}-1} y(4n+1) W_{N/2}^{2nk} + \sum_{n=0}^{\frac{N}{4}-1} y(4n+3) W_{N/2}^{(2n+1)k} \right\} =$$

$$\sum_{n=0}^{\frac{N}{4}-1} y(4n) W_{N/2}^{2nk} + W_{N/2}^k \sum_{n=0}^{\frac{N}{4}-1} y(4n+2) W_{N/2}^{2nk} +$$

$$W_N^k \left\{ \sum_{n=0}^{\frac{N}{4}-1} y(4n+1) W_{N/2}^{2nk} + W_{N/2}^k \sum_{n=0}^{\frac{N}{4}-1} y(4n+3) W_{N/2}^{2nk} \right\}$$

(EQ 31)

And since

$$W_{N/2}^{2} = W_{N/4}$$

$$Y(k) = \sum_{n=0}^{\frac{N}{4}-1} y(4n) W_{N/4}^{nk} + W_{N/2}^{k} \sum_{n=0}^{\frac{N}{4}-1} y(4n+2) W_{N/4}^{nk} + W_{N}^{k} \left\{ \sum_{n=0}^{\frac{N}{4}-1} y(4n+1) W_{N/4}^{nk} + W_{N/2}^{k} \sum_{n=0}^{\frac{N}{4}-1} y(4n+3) W_{N/4}^{nk} \right\}$$

(EQ 32)

And by using equation 21 backwards

$$W_{N/2}^{k} = W_{N}^{2k}$$

$$Y(k) = \sum_{n=0}^{\frac{N}{4}-1} y(4n) W_{N/4}^{nk} + W_{N}^{2k} \sum_{n=0}^{\frac{N}{4}-1} y(4n+2) W_{N/4}^{nk} + W_{N}^{k} \left\{ \sum_{n=0}^{\frac{N}{4}-1} y(4n+1) W_{N/4}^{nk} + W_{N}^{2k} \sum_{n=0}^{\frac{N}{4}-1} y(4n+3) W_{N/4}^{nk} \right\}$$

(EQ 33)

And if we continue to divide into smaller sums until we only have N/2 2-points DFT's we will get the structure described below, the constrain though is that the length N should be a power of two. The example below shows the structure for N = 8.

Uppgjord (även faktaansvarig om annan) - *Prepared (also subject responsible if other)*		Nr - *No.*		
EMWMSNN(Magnus Nilsson)		FX/D-2001:007		
Dokansv/Godkänd - *Doc respons/Approved*	Kontr - *Checked*	Datum - *Date*	Rev	File
EMW/FX/DC(Anders Wanner)		2001-02-12	A1	

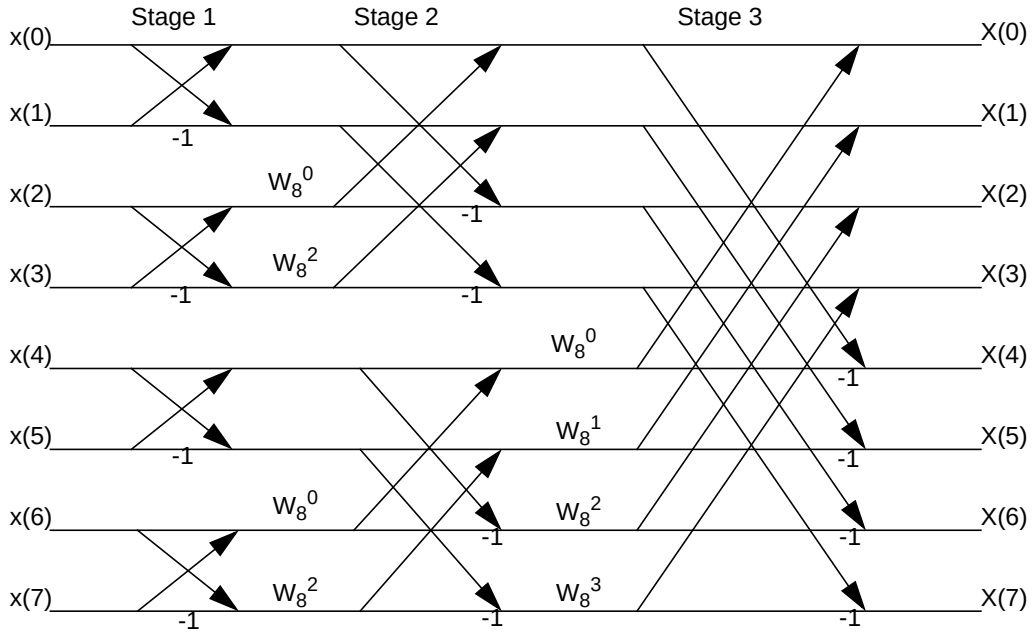


Fig.2. Radix-2 DFT structure

When computing a DFT using a Radix-2 algorithm for the case when $N = 2^x$ the decimation into smaller sums can be done $x = \log_2 N$ times, and this will give a total number of complex multiplications = $(N/2)\log_2 N$ and $N\log_2 N$ complex additions. The gain when comparing with a direct calculation is enormous as shown in the figure below:

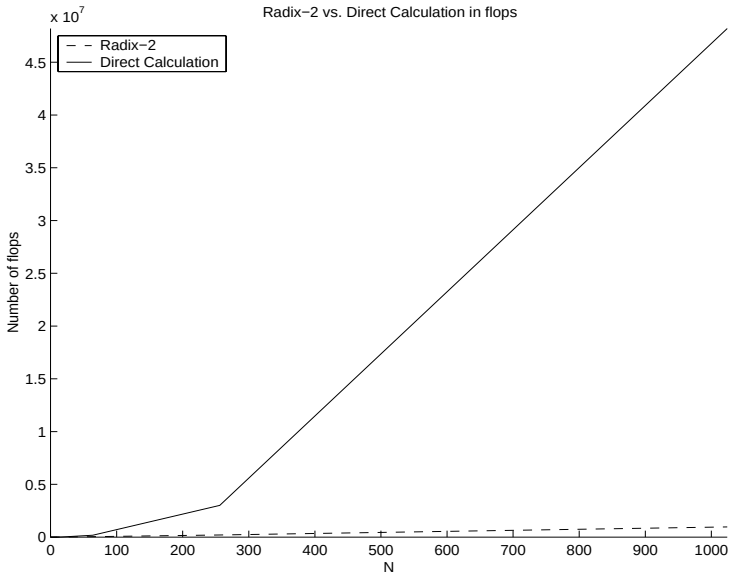


Fig.3. Radix-2 vs. Direct calculation in flops

Part of radix-2 Matlab algorithm.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialize variables.
t = 1:1:1024;
x = sin(2*pi*0.35*t)+sin(2*pi*0.25*t);
N = length(x);
b = bin2dec(fliplr(dec2bin(0:1:length(x)-1)))+1;
MC = x(b); % Make in bit reversed order
alfa = N/2;
beta = 1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate Twiddle factor
for n = 1:N/2
   W(n) = exp(-j*2*pi*(n-1)/N);
   W_r(n) = cos(2*pi*(n-1)/N);
   W_i(n) = -sin(2*pi*(n-1)/N);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate FFT using inplace non-recursive DIT FFT, radix-2
for h = 1:(log(N)/log(2))
    b = 2^(h-1);
    a = 1;
    aO = 1;
    for d = 1:alfa
        c = 1;
        for e = 1:beta
            a+b;
            temp1 = W(c)*MC(a+b);
            temp2 = MC(a);
            MC(a) = MC(a) + temp1;
            MC(a+b) = temp2 - temp1;
            a = a + 1;
            c = c + alfa;
        end
        a = aO + 2^(h);
        aO = a;
    end
    alfa = alfa/2;
    beta = beta*2;
end
```

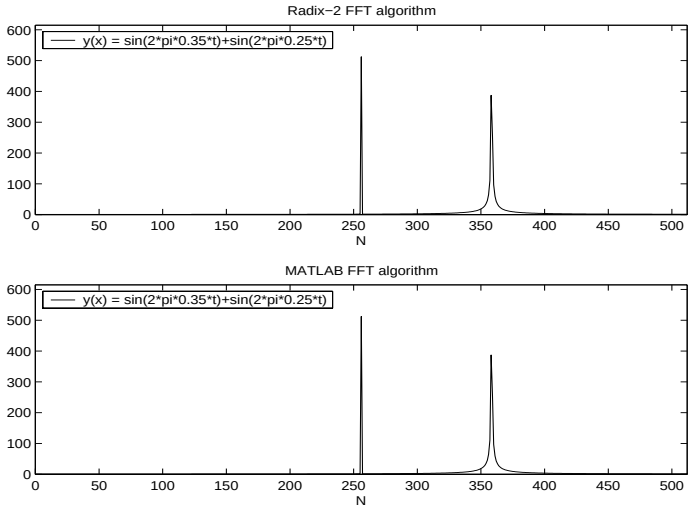


Fig.4. Radix-2 algorithm comp. with MATLAB function FFT

7 THE RADIX-4 ALGORITHM

By developing the Radix-2 algorithm even further and using the base 4 instead we will get the a more complex algorithm but with less computation power. As we will understand, we will get the new constrain where the number of data points N in the DFT has to be the power of 4 (i.e. $N = 4^x$). By doing in the same way as we did with the Radix-2 algorithm we divide the data sequence into four subsequence

$$\{y(4n)\} = \{y(0), y(4), y(8), \ldots, y(N-4)\}$$
$$\{y(4n+1)\} = \{y(1), y(5), y(9), \ldots, y(N-3)\}$$
$$\{y(4n+2)\} = \{y(2), y(6), y(10), \ldots, y(N-2)\}$$
$$\{y(4n+3)\} = \{y(3), y(7), y(11), \ldots, y(N-1)\}$$

By using the approach described in [8] and by applying:

$$X(p,q) = \sum_{l=0}^{3} [W_N^{lq} F(l,q)] W_4^{lp}$$
$$p = 0, 1, 2, 3$$

(EQ 34)

where F(l,q) is given by:

$$F(l,q) = \sum_{m=0}^{\left(\frac{N}{4}-1\right)} x(l,m) W_{\frac{N}{4}}^{mq}$$
$$l = 0, 1, 2, 3$$
$$q = 0, 1, 2, \ldots, \frac{N}{4} - 1$$

(EQ 35)

And where:

$$x(l,m) = x(4m+l)$$
$$X(p,q) = X\left(\frac{N}{4}p + q\right)$$

(EQ 36)

And the four N/4-point DFT's obtained from equation 35 are combined according to equation 34 and can be combined to yield the N-point DFT, as described in [8]:

$$\begin{bmatrix} X(0,q) \\ X(1,q) \\ X(2,q) \\ X(3,q) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} W_N^0 F(0,q) \\ W_N^q F(1,q) \\ W_N^{2q} F(2,q) \\ W_N^{3q} F(3,q) \end{bmatrix}$$

(EQ 37)

We also have to note that $W^0_N = 1$, which will give us three complex multiplications and 12 complex additions per Radix-4 butterfly. As the Radix-4 algorithm consists of v steps (log(N)/log(4)) where each step involves N/4 number of butterflies we will get 3*v*N/4 = $(3N/8)\log_2 N$ number of complex multiplications and $(3N/2)\log_2 N$ complex additions. If compared with the computational power used by the Radix-2 algorithm in chapter 5, we will find that we have a computer gain of 25% regarding the complex multiplications, but that the number of complex additions increases by 50%.

The matrix in equation 37 is better described with a Radix-4 butterfly:

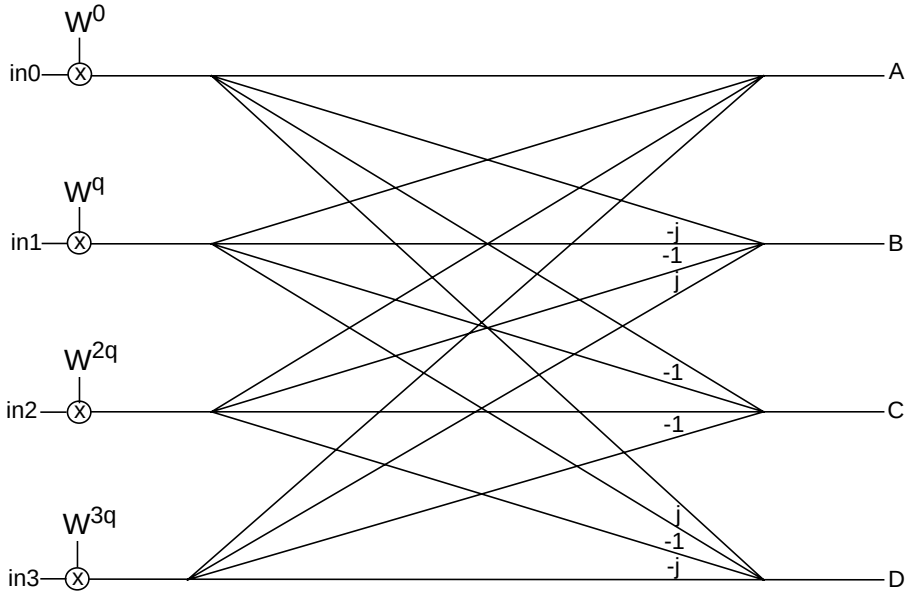


Fig.5. Radix-4 Butterfly, also referred to as Dragonfly

As we are interested in a complex FFT we need to derive the equations for the complex radix-4 algorithm.

$$\left.\begin{aligned} a &= in0 \times 1 \\ b &= in1 \times W^{q} \\ c &= in2 \times W^{2q} \\ d &= in3 \times W^{3q} \end{aligned}\right\} \Rightarrow \begin{aligned} A &= a + b + c + d \\ B &= a - c - j(b - d) \\ C &= a + c - (b + d) \\ D &= a - c + j(b - d) \end{aligned}$$

(EQ 38)

Which in the complex matter will give us, starting with the easiest ones:
(r =real, i = imag)

$$\begin{aligned} Ar &= ar + br + cr + dr \\ Ai &= ai + bi + ci + di \\ Cr &= ar - br + cr - dr \\ Ci &= ai - bi + ci - di \end{aligned}$$

(EQ 39)

And continuing with B gives:

$$B = ar + ai - cr - ci - jbr - jbi + jdr + jdi$$
$$B = ar + ai - cr - ci - jbr + bi + jdr - di$$

imag real

(EQ 40)

Divided into real and imaginary part:

$$\begin{aligned} Br &= ar - cr + bi - di \\ Bi &= ai - ci - br + dr \end{aligned}$$

(EQ 41)

And the last one gives:

$$D = ar + ai - cr - ci + jbr + jbi - jdr - jdi$$
$$D = ar + ai - cr - ci + jbr - bi - jdr + di$$

imag real

(EQ 42)

Divided into real and imaginary part:

$$Dr = ar - cr - bi + di$$
$$Di = ai - ci + br - dr$$

(EQ 43)

To get the inputs ar, ai, br, bi, cr, ci, dr and di, we will have to multiply the input in0r, in0i and so on with the twiddle factor. This render in:

$$br = in1r \times \cos(x) + in1i \times \sin(x)$$
$$bi = in1i \times \cos(x) - in1r \times \sin(x)$$

(EQ 44)

This is adequate for all input signals. X = dragonfly specific value (twiddle factor)

As the goal of this project is to implement a very fast fourier transform in a realtime programmable logic system, we want as few complex multiplications as possible, which yields lots of logic. With this in thoughts, to choose the Radix-4 algorithm for implementation was obvious, as it has less complex multiplications than the Radix-2 algorithm.

Part of radix-4 Matlab algorithm

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Innitialize variables.
t = 1:1:256;
x = sin(2*pi*0.35*t)+sin(2*pi*0.38*t);
x1 = x;
n = length(x);
t = log(n)/log(4);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Radix-4 Algorithm

for q = 1:t
    L = 4^q;
    r = n/L;
    Lx = L/4;
    rx = 4*r;
    y = x;
    for j = 0:Lx-1
        for k = 0:r-1
            a = y(j*rx + k + 1);
            b = exp(-i*2*pi*j/L)*y(j*rx + r + k + 1);
            c = exp(-i*2*pi*2*j/L)*y(j*rx + 2*r + k + 1);
            d = exp(-i*2*pi*3*j/L)*y(j*rx + 3*r + k + 1);
            t0 = a + c;
            t1 = a - c;
            t2 = b + d;
            t3 = b - d;
            x(j*r + k + 1) = t0 + t2;
            x((j + Lx)*r + k + 1) = t1 - i*t3;
            x((j + 2*Lx)*r + k + 1) = t0 - t2;
            x((j + 3*Lx)*r + k + 1) = t1 + i*t3;
        end
    end
end
```

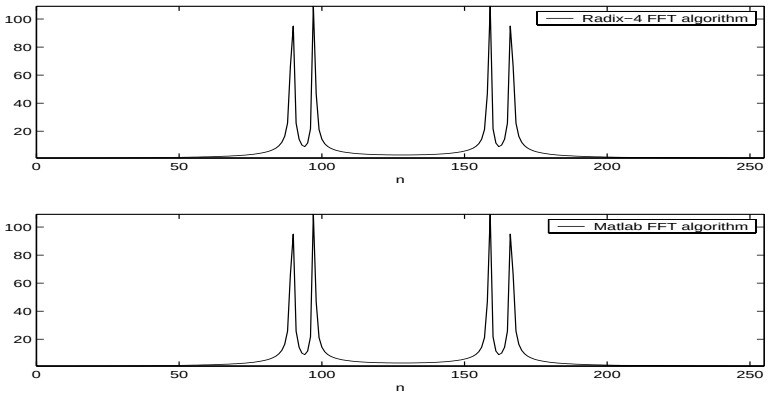


Fig.6. Radix-4 FFT algorithm compared with Matlab FFT

8 IMPLEMENTATION AND REALIZATION IN HARDWARE

Classical implementation of the FFT algorithm, with a processor or in hardware usually requires a sequential algorithm, in some cases recursive, this due to space and memory requirements. This slows down the execution time. By utilizing modern programmable circuits, like a FPGA, a parallel approach to the realization of FFT is available.

8.1 FPGA

The real-time FFT construction was meant to be realized in a FPGA, a field programmable gate array, constructed and manufactured by Xilinx, Inc. The Xilinx FPGA model Virtex-E is a state of the art programmable gate array for high speed, high complex logical construction. There is a great field of models, from small to large circuits. The logic inside a FPGA is constructed around a building block called CLB, Configurable logic block.

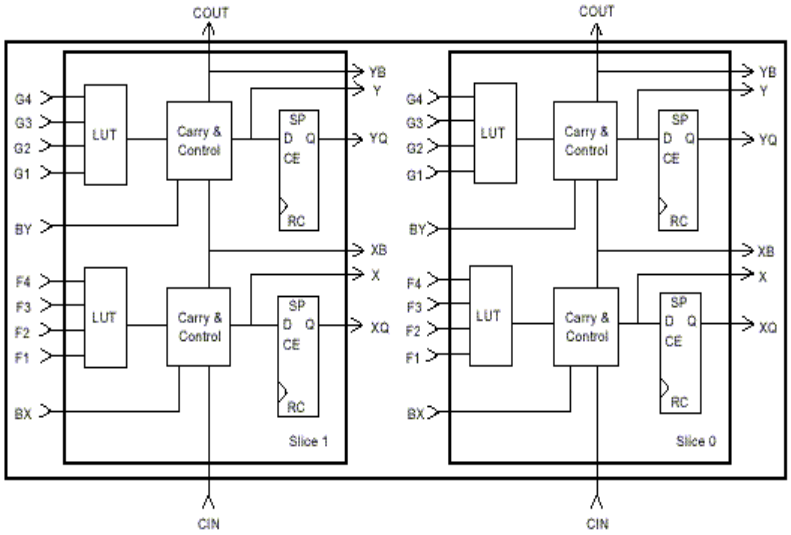


Fig.7. CLB, Configurable logic block. Courtesy of Xilinx Inc.

Each of these blocks are divided into two slices, where each slice consists of two look-up tables and some storage elements. The slices are internally connected in between and are the basic high-speed logic in the circuit.

Available for implementation of this project was a PCB with a Xilinx Virtex-E 1000 mounted. This circuit holds a CLB array of 64 x 96 = 6144 CLB blocks.

For more information about Xilinx Virtex-E, refer to Xilinx Vertex-E data book [13].

8.2 COMPLEX FFT

The Ericsson Microwave specification for the project was to simulate, realize and implement a complex FFT in a FPGA, a Xilinx Virtex-E 1000. The specifications for the FFT was:

FFT-length

Minimum: 16 complex samples
Maximum: 1024 complex samples
Typical: 64 or 256 (16)

Number of bits for the input signal

Minimum: 10 bits
Maximum: 16 bits
Typical: 12

The idea was to implement the FFT as a building-block in a construction, where the FFT-block will be placed after a quadrature divided A/D converted signal as described in the figure below:

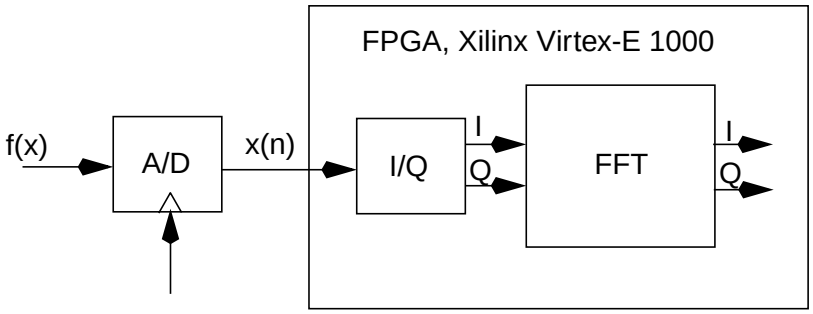


Fig.8. Construction configuration

8.3 BIT-LENGTH

The first thing to consider when implementing something discrete in hardware is to consider the bit length with which you want to represent your sample. The best way to do this is to simulate different types of bit lengths and compare the phase error and amplitude error factor with the constrains for your construction.

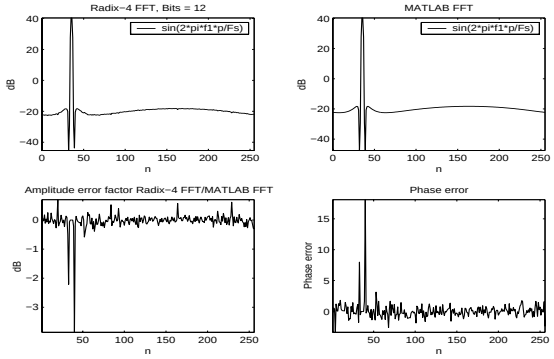


Fig.9. Radix-4 FFT, 12-bit length of samples

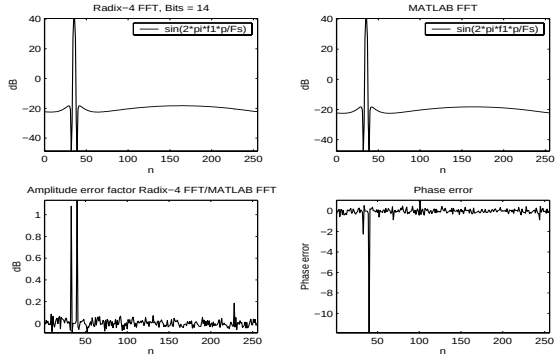


Fig.10. Radix-4 FFT, 14-bit length of samples

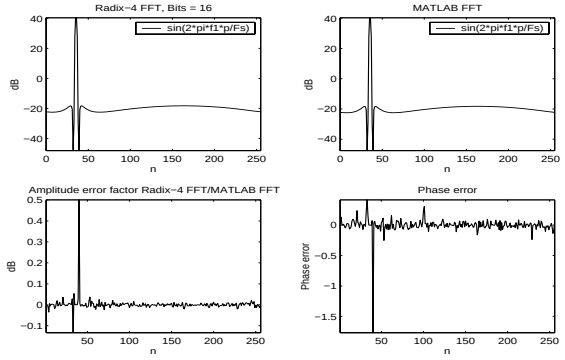


Fig.11. Radix-4 FFT, 16-bit length of samples

As the constrains for the real-time FFT construction was to minimize the phase and amplitude error as much as possible, but not more than that the construction could be realizable. The simulation results pointed towards 16 bits, as this result had a small value of phase error.

8.4 RADIX-4 FFT ALGORITHM, N = 64

The first attempt of the implementation phase was to implement a Radix-4 FFT algorithm, with length 64 complex samples. For a Radix-4 FFT with length N = 64, there are 3 dragonfly ranks, with each rank comprising 16 dragonflies.

In the first revision of the construction, the bit-length of the input samples to the first dragonfly rank was 12, this due to the precision of the quadrature block in figure 8. Those input samples were then multiplied with the phase factor for the correct block, also with a precision of 12 bits. As the complex output of the multiplication will generate $2^{12} * 2^{12}$ => 24 bits, the complex output of the multiplication is rounded of and truncated to 14 bits. This results in a 14 bits input to the second rank of dragonflies, which will by using the same model as for the first rank of dragonflies, generate a complex output with the length of 16 bits. As we also have a third rank of dragonflies, the complex output from our FFT construction will have 18 bits.

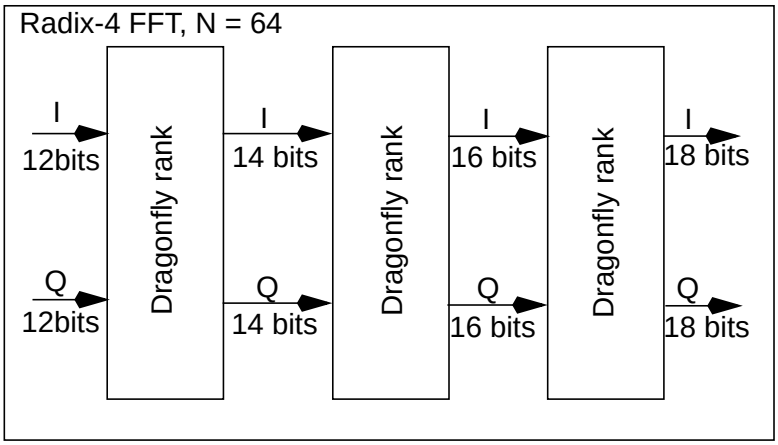


Fig.12. Radix-4 FFT, N = 64

The FFT-block was constructed using the software EASE and the programming language VHDL, i.e. Very high speed integrated circuit Hardware Description Language.

The software Ease is a block model description language that lets you construct the algorithm as blocks and takes care of the interconnection between the blocks and then generates the VHDL code for this interconnection [10].

Part of the VHDL - code for one of the dragonflies in the first of the ranks is as follows:

```
begin  -- process radix4
    if clk'event and clk = '1' then
      ar_temp  <= in0r*cos_0j;
      ai_temp  <= in0i*sin_0j;
      br_temp1 <= in1r*cos_1j;
      br_temp2 <= in1i*sin_1j;
      bi_temp1 <= in1i*cos_1j;
      bi_temp2 <= in1r*sin_1j;
      cr_temp1 <= in2r*cos_2j;
      cr_temp2 <= in2i*sin_2j;
      ci_temp1 <= in2i*cos_2j;
      ci_temp2 <= in2r*sin_2j;
      dr_temp1 <= in3r*cos_3j;
      dr_temp2 <= in3i*sin_3j;
      di_temp1 <= in3i*cos_3j;
      di_temp2 <= in3r*sin_3j;

      br_temp  <= br_temp1 + br_temp2;
      bi_temp  <= bi_temp1 - bi_temp2;
      cr_temp  <= cr_temp1 + cr_temp2;
       ci_temp  <= ci_temp1 - ci_temp2;
       dr_temp  <= dr_temp1 + dr_temp2;
       di_temp  <= di_temp1 - di_temp2;

      ar_round <= ar_temp((N*2-1) downto (N-3)) + round;
      ai_round <= ai_temp((N*2-1) downto (N-3)) + round;
      br_round <= br_temp((N*2-1) downto (N-3)) + round;
      bi_round <= bi_temp((N*2-1) downto (N-3)) + round;
      cr_round <= cr_temp((N*2-1) downto (N-3)) + round;
      ci_round <= ci_temp((N*2-1) downto (N-3)) + round;
      dr_round <= dr_temp((N*2-1) downto (N-3)) + round;
      di_round <= di_temp((N*2-1) downto (N-3)) + round;

      temp1r   <= ar_round((N+2) downto 1) + cr_round((N+2) downto 1);
      temp1i   <= ai_round((N+2) downto 1) + ci_round((N+2) downto 1);
      temp2r   <= ar_round((N+2) downto 1) - cr_round((N+2) downto 1);
      temp2i   <= ai_round((N+2) downto 1) - ci_round((N+2) downto 1);

      ar_out   <= temp1r + br_round((N+2) downto 1) + dr_round((N+2) downto 1);
      ai_out   <= temp1i + bi_round((N+2) downto 1) + di_round((N+2) downto 1);
      br_out   <= temp2r + bi_round((N+2) downto 1) - di_round((N+2) downto 1);
      bi_out   <= temp2i - br_round((N+2) downto 1) + dr_round((N+2) downto 1);
      cr_out   <= temp1r - br_round((N+2) downto 1) - dr_round((N+2) downto 1);
      ci_out   <= temp1i - bi_round((N+2) downto 1) - di_round((N+2) downto 1);
      dr_out   <= temp2r - bi_round((N+2) downto 1) + di_round((N+2) downto 1);
      di_out   <= temp2i + br_round((N+2) downto 1) - dr_round((N+2) downto 1);
    end if;
end process radix4;
end  a0 ; -- of Block1
```

The VHDL code above describes exactly the dragonfly illustrated in figure 5. For this block the phase/twiddle factor is simple and can easy be realized as a right shift of the input signal, but as we get towards the last dragonfly in the construction, the phase factor constant gets more complex and has to be realized as a high performance multiplier. The total block description is described in appendix A1.

The 64 complex input signals is shifted into the FFT-block using a shift register. This register is divided into a real and an imaginary part where the input (complex) gets a new sample every clock cycle. When the shift register gets full, it generates a valid signal that triggs the FFT-block that starts the FFT process. When the process is done, a new valid signal is generated, and an output shift register is started. For every clock cycle, a new processed value is delivered to the output. After all 64 values are delivered the valid signal gets low and shows that every sample has been shifted out.

The code is before realization simulated, using a simulation program called Modelsim, a VHDL simulation program. The result from this part showed perfect result when comparing with a MATLAB FFT of a sinusoidal signal.

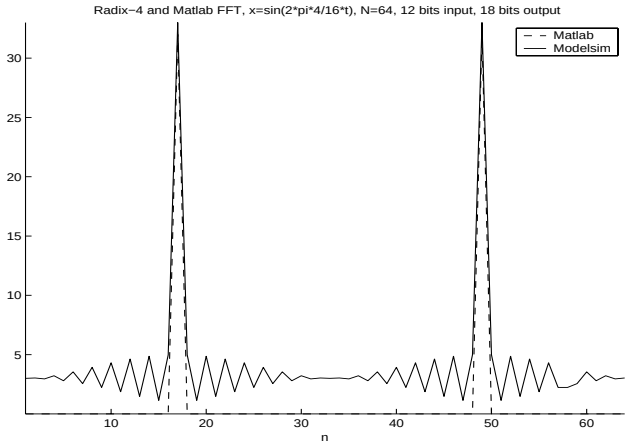


Fig.13. First FFT construction vs. Matlab FFT

What we can see from the plot in figure 13 is that we get a truncation and rounding error that will generate a noise in the FFT plot. Although the frequency peaks are in the correct position.

The next step after simulation is to realize the construction in the next program. This program is called Synplify and translates/syntesis from code to gate level.

The sad conclusion when reaching this level was that the construction was to large for a Virtex-E 1000 circuit and even for the next higher circuit, Virtex-E 2000, with twice as many CLB-blocks, but Xilinx has larger circuits, like the Virtex-E 3200 where this construction would be implementable. This means that we have to consider another bit-length or a shorter FFT-length.

The first try was to consider another bit length. By changing the design and the dragonfly blocks to 12 bits we will save lots of hardware, but we will get a higher amount of phase error.

The new dragonfly blocks (still as in figure 5) uses 12 bits as input and uses 12 bits precision on the phase factor. The output from the multiplication will generate 24 bits that gets rounded of and truncated back to 12 bits. The same applies for the second and the third dragonfly ranks. So the input will be 12 bits and the output will also be 12 bits.

A new consideration is also to instead of using four complex multiplier (8 real multiplier) as in the above mentioned implementation, we have to consider the choice when we reuse the same multiplier for all of the multiplications. This fact is better described by looking the VHDL code below.

Part of VHDL-code for Radix-4 dragonfly with shared multiplier:

```
begin  -- process radix4
      if clk'event and clk = '1' then
           if done_i = '1' then
           run <= '1';
               radix <= 0;
           end if;

      if run = '1' then
           case  radix  is
                when  0  =>
                    signal_in0 <= in0r;
                    constant_in0 <= cos_0j;
                    radix <= 1;
                when  1  =>
                    signal_in0 <= in0i;
                    constant_in0 <= sin_0j;
                    ar_temp <= signal_in0*constant_in0;
                    radix <= 2;
                when  2  =>
                    signal_in0 <= in1r;
                    constant_in0 <= cos_1j;
                    signal_in1 <= in1i;
                    constant_in1 <= sin_1j;
                    ai_temp <= signal_in0*constant_in0;
                    radix <= 3;
                when  3  =>
                    signal_in0 <= in1i;
                    constant_in0 <= cos_1j;
                    signal_in1 <= in1r;
                    constant_in1 <= sin_1j;
                    br_temp <= signal_in0*constant_in0 + signal_in1*constant_in1;
                    radix <= 4;
                when  4  =>
                    signal_in0 <= in2r;
                    constant_in0 <= cos_2j;
                    signal_in1 <= in2i;
                    constant_in1 <= sin_2j;
                    bi_temp <= signal_in0*constant_in0 - signal_in1*constant_in1;
                    radix <= 5;
                when  5  =>
                    signal_in0 <= in2i;
                    constant_in0 <= cos_2j;
                    signal_in1 <= in2r;
                    constant_in1 <= sin_2j;
                    cr_temp <= signal_in0*constant_in0 + signal_in1*constant_in1;
                    radix <= 6;
                when  6  =>
                    signal_in0 <= in3r;
                    constant_in0 <= cos_3j;
                    signal_in1 <= in3i;
                    constant_in1 <= sin_3j;
                    ci_temp <= signal_in0*constant_in0 - signal_in1*constant_in1;
                    radix <= 7;
                when  7  =>
                    signal_in0 <= in3i;
                    constant_in0 <= cos_3j;
                    signal_in1 <= in3r;
                    constant_in1 <= sin_3j;
                    dr_temp <= signal_in0*constant_in0 + signal_in1*constant_in1;
                    radix <= 8;
                when  8  =>
                    di_temp <= signal_in0*constant_in0 - signal_in1*constant_in1;
                    radix <= 9;
                when  9  =>
                    ar_out   <= ar_temp((2*N-1) downto(N)) + cr_temp((2*N-1) downto(N)) + br_temp((2*N-
1) downto(N)) + dr_temp((2*N-1) downto(N));
                    ai_out   <= ai_temp((2*N-1) downto(N)) + ci_temp((2*N-1) downto(N)) + bi_temp((2*N-
1) downto(N)) + di_temp((2*N-1) downto(N));
                    br_out   <= ar_temp((2*N-1) downto(N)) - cr_temp((2*N-1) downto(N)) + bi_temp((2*N-
1) downto(N)) - di_temp((2*N-1) downto(N));
                    bi_out   <= ai_temp((2*N-1) downto(N)) - ci_temp((2*N-1) downto(N)) - br_temp((2*N-
1) downto(N)) + dr_temp((2*N-1) downto(N));
                    cr_out   <= ar_temp((2*N-1) downto(N)) + cr_temp((2*N-1) downto(N)) - br_temp((2*N-
1) downto(N)) - dr_temp((2*N-1) downto(N));
                    ci_out   <= ai_temp((2*N-1) downto(N)) + ci_temp((2*N-1) downto(N)) - bi_temp((2*N-
1) downto(N)) - di_temp((2*N-1) downto(N));
                    dr_out   <= ar_temp((2*N-1) downto(N)) - cr_temp((2*N-1) downto(N)) - bi_temp((2*N-
1) downto(N)) + di_temp((2*N-1) downto(N));
                    di_out   <= ai_temp((2*N-1) downto(N)) - ci_temp((2*N-1) downto(N)) + br_temp((2*N-
1) downto(N)) - dr_temp((2*N-1) downto(N));
                     --done_o <= '1';
                     radix <= 10;
                when others =>
                radix <= 0;
                run <= '0';
                --done_o <= '0';
               end case;
           end if;
```

As we can see in this code for the first block, we utilize a SWITCH-CASE structure, using the same multiplier (actually two multipliers, one for the real part and one for the imaginary part) for all the multiplications between the phase factor and the input. This slows down the construction, as we have to use more clock cycles to get all the inputs through the dragonfly. The timing diagram below shows how the construction works.

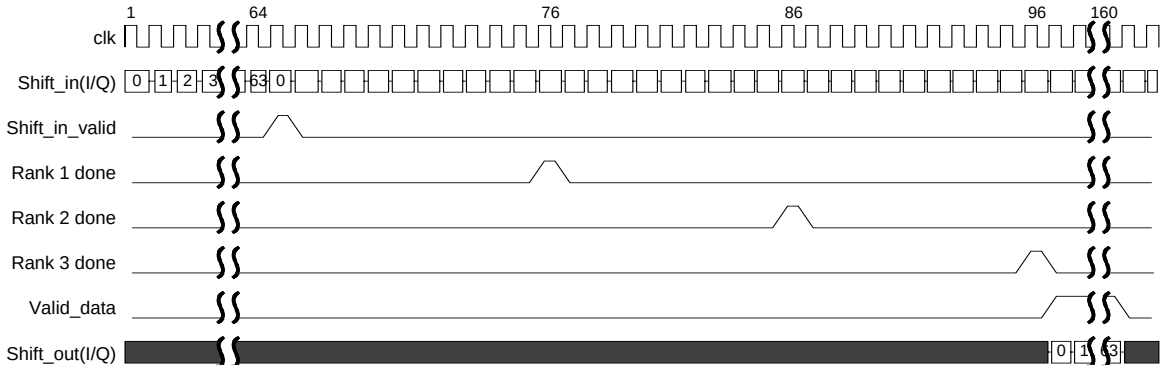


Fig.14. Timing diagram for Radix-4 FFT, shared multiplier

The grey marked area in Shift_out(I/Q) is invalid data. Also this construction showed to be to large for a Virtex-E 1000 circuit. The code translation program Synplify showed though that the construction is realizable in a Virtex-E 2000 at a clock rate of 55 MHz, this means that the computation phase alone is 640 nanoseconds in duration. This can be compared with the Xilinx Virtex LogiCore blocks that utilize 1.92 microseconds for the same computation[13].

Another constrain that the LogiCore block has, is that it has to have all the complex input samples in serial. It is easy to change the above described construction to get all the samples as a gigantic parallel bus, or just speed up the clock rate on the Shift_in register and the Shift_out register, $clk_{Shift}=N*clk_{FFT}$, as the clock rate constrain for this construction is in the multipliers in the dragonflies and not in the Shift registers. The total block diagram description is displayed in appendix A2.

As we wanted to implement and realize a construction in the accessible Virtex-E 1000, described in figure 8 above we once again have to reconsider the FFT-sample length and the bit length of the construction. As the above mentioned 12 bits FFT with length 64 complex samples, utilized almost 75% of a Virtex-E 2000 circuit, we can be quite sure that a 16 bits FFT with FFT length 16 complex samples will be possible to implement in a Virtex-E 1000.

8.5 RADIX-4 FFT ALGORITHM, N = 16

A new construction with the FFT length of 16 complex samples had to be made. The construction consists of two dragonfly ranks with each four dragonflies. It also consists of an input register that holds four predefined signals for the FFT, this instead of using the quadrature divided A/D input signal:

$$x1 = \sin\left(2\pi \times \frac{4}{16} t\right)$$

$$x2 = \sin\left(2\pi \times \frac{4.5}{16} t\right)$$

$$x3 = \cos\left(2\pi \times \frac{4}{16} t\right) + j\sin\left(2\pi \times \frac{4}{16} t\right)$$

$$x4 = \cos\left(2\pi \times \frac{4.5}{16} t\right) + j\sin\left(2\pi \times \frac{4.5}{16} t\right)$$

Where t goes from 1 to 16. The signals are then converted to two's complement using a Matlab function, TWOSCOMP(no_of_bits,DATA).

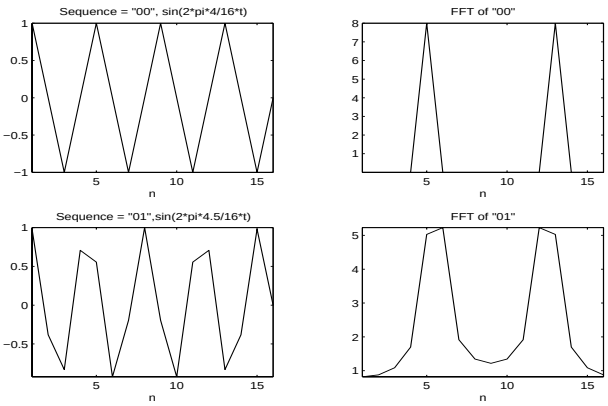


Fig.15. Input signal X1 and X2

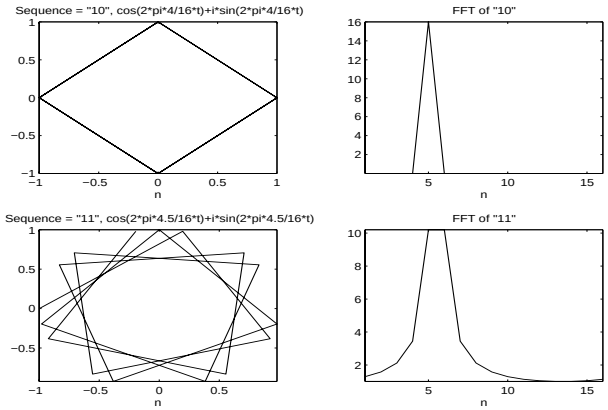


Fig.16. Input signal X3 and X4

The idea with those four signals is to let the FFT construction consider two real input signals, one within a FFT channel and one outside, and two complex input signals, also here one within a FFT channel and one outside.

The two signals that are outside the FFT channel will spread through all the channels, as we can see from the FFT plots above.

The FFT construction with bit length 16 and the four predefined signals, x1-x4, can be defined as:

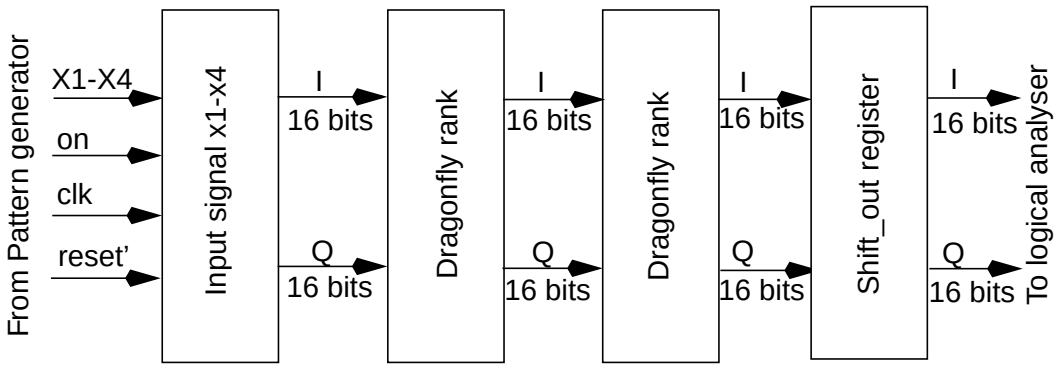


Fig.17. Radix-4 N = 16

The construction was syntesized in the code translation program Synplify and was realizable in 50 MHz using the below showed timing constrains.

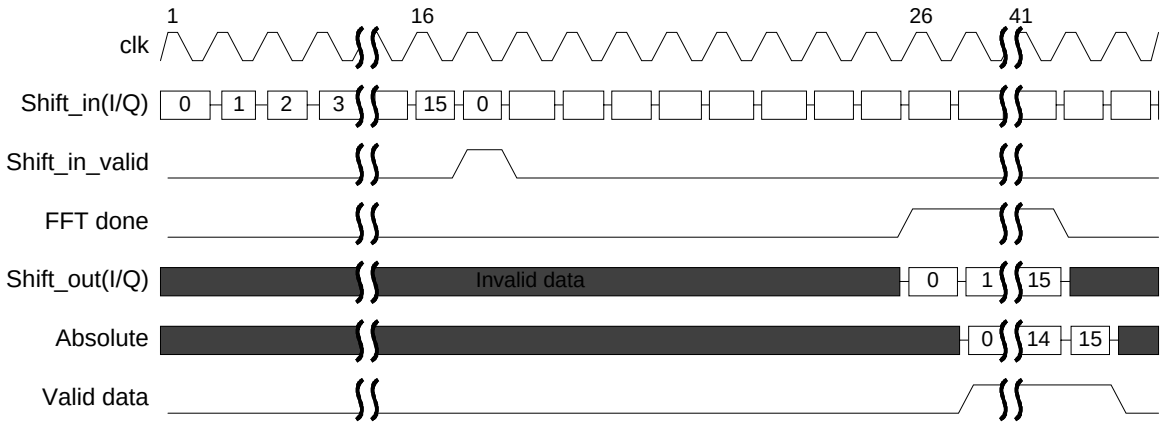


Fig.18. Timing diagram for Radix-4 FFT length 16, 16 bits

The input connection to the FPGA goes through a serial interface called HOT-LINK, a high speed serial interface, that is connected to a Pattern generator, a Hewlett Pacard HP16522A (200 MHz in 32 channels) for generating the input stimuli.

The output 16 bits vector from the real (I) and imaginary (Q) part, is taken care of by a Logical Analyser, a Hewlett Pacard HP16555D (2.0 M Samples, 110/500 MHz). As this instrument has the possibility to display the output both as listing and as a graph, a absolute value block was made and implemented after the Shift_out register:

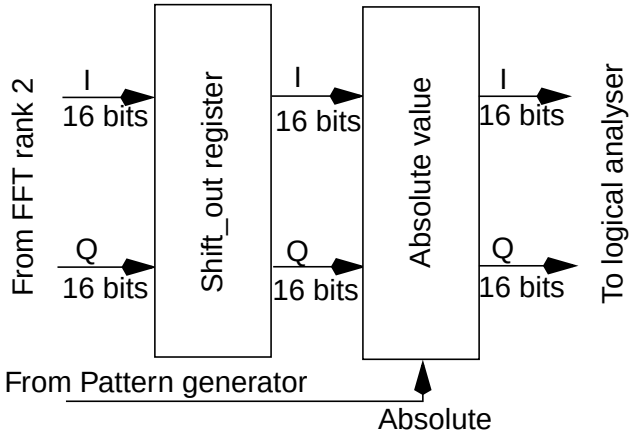


Fig.19. Absolute value block

As the calculation of the absolute value is a quite complex procedure to do, a alternative method is utilized. If we consider the following equations:

$$A = Max\{|I|, |Q|\}$$
$$B = Min\{|I|, |Q|\}$$
$$Absolutevalue = Max\left\{A, \frac{7}{8}A + \frac{1}{2}B\right\}$$

(EQ 45)

This method is quite easy to implement in hardware, and the precision of this method is +1% / -2% of variation on the output. This block is controlled by the Absolute trigger, connected to the Pattern generator. When the Absolute trigger = 1, the absolute value is delivered in the real part output channel to the Logic Analyser. The imaginary part equals zero.

The construction can be compared with Xilinx LogiCore block [13] that also uses 16 bits precision on the input and the phase factor. The Xilinx LogiCore block requires 16 clock cycles (at a clock rate of 120 MHz) when the one mentioned above, requires 10 clock cycles (at a clock rate of 50 MHz). The difference once again is that the Xilinx LogiCore block requires that the input data is delivered in serial, the above described block can take care of 16 new complex 16 bits samples on every clock cycle.

$$T_{FFTXilinx} = \frac{1}{120MHz} \times 16 = 133.33ns$$

(EQ 46)

$$T_{FFTEMW} = \frac{1}{50MHz} \times 10 \times \frac{1}{16} = 12.5ns$$

(EQ 47)

The construction require twice as many CLB's then the Xilinx LogiCore block (963 CLB's / 1876 CLB's). The total block diagram description is displayed in appendix A3.

9 VERIFICATION AND RESULTS

9.1 TEST PATTERN

To get a file to download to the configuration PROM, a circuit specific software called Design manger is utilized. When you then apply power to your PCB, the circuit, in this case the Xilinx Virtex-E 1000 will download the configuration file and load your design.

A verification/test pattern was programmed in the Pattern generator.

Test pattern:

```
1) reset = 0 (active low)
2) Synchronize HOT-LINK
3) absolute = 0 (output = real + imag part)
   sequence = 00 (signal x1)
   on_signal = 1 (the signal in on)
4) reset = 1
5) When output signal Test_valid_data = 1
   Collect fft_out_r (real part, I), and
   Collect fft_out_i (imag part, Q) in Logic Analyser
   (listning)
6) When Test_valid_data = 0 again
   reset = 0
7) Synchronize HOT-LINK
8) absolute = 1 (gives real = absolute value, imag = 0)
   sequence = 00 (signal x1)
   on_signal = 1
9) When Test_valid_data = 1
   Collect fft_out_r and display as graph
10)When Test_valid_data = 0 again
   Goto 1 but change to next signal (x2-x4)
```

The output was collected in the Logical Analyser and transferred to Matlab for verification.

9.2 MATLAB VERIFICATION

The files from the Logic Analyser was loaded in Matlab and compared with the result from a FFT made by Matlab itself on the same input signal (x1-x4).

When the absolute trigger from the Pattern generator is set to 1, the output graph from the Logical Analyser displayed the following result.

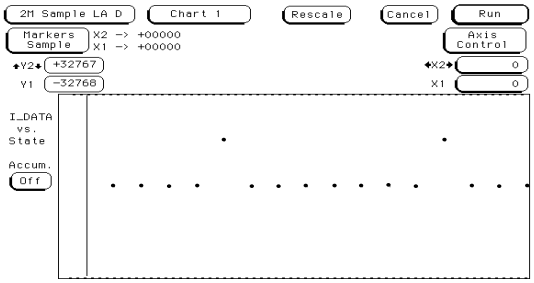


Fig.20. Output graph signal X1, absolute = 1

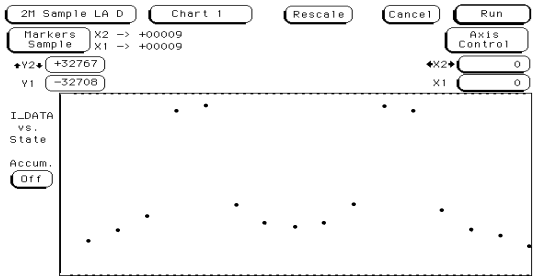


Fig.21. Output graph signal X2, absolute = 1

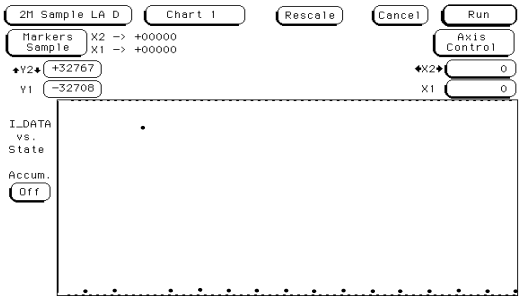


Fig.22. Output graph signal X3, absolute = 1

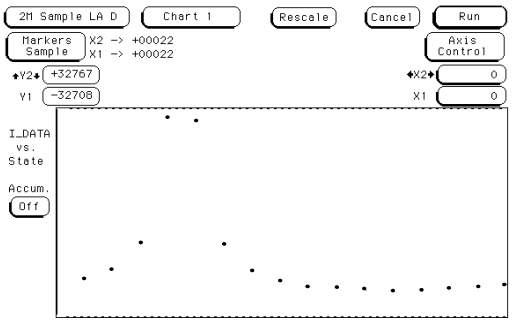


Fig.23. Output graph signal X4, absolute = 1

Uppgjord (även faktaansvarig om annan) - *Prepared (also subject responsible if other)*		Nr - *No.*		
EMWMSNN(Magnus Nilsson)		FX/D-2001:007		
Dokansv/Godkänd - *Doc respons/Approved*	Kontr - *Checked*	Datum - *Date*	Rev	File
EMW/FX/DC(Anders Wanner)		2001-02-12	A1	

If we compare figure 15 and 16 with figure 20 to 23, we will see that the construction is working properly. The value listing from the verification with signal X1-X4 was properly compared with FFT calculations in Matlab.

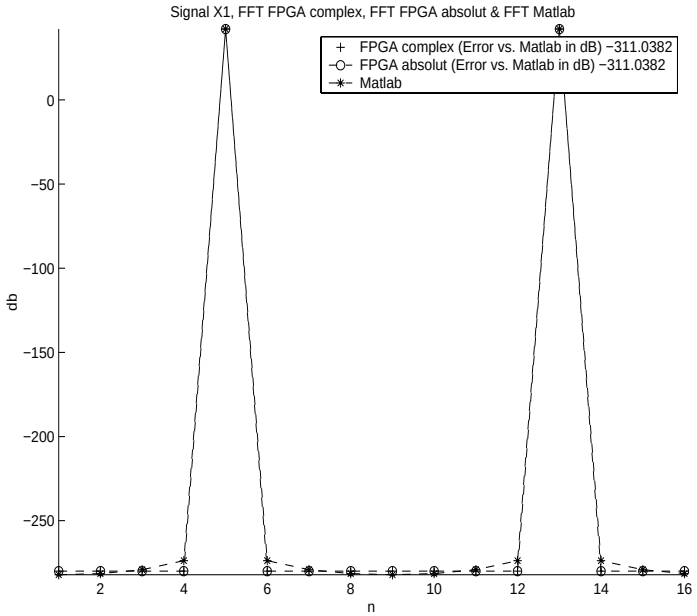


Fig.24. Output complex and absolute, signal 1 vs. Matlab

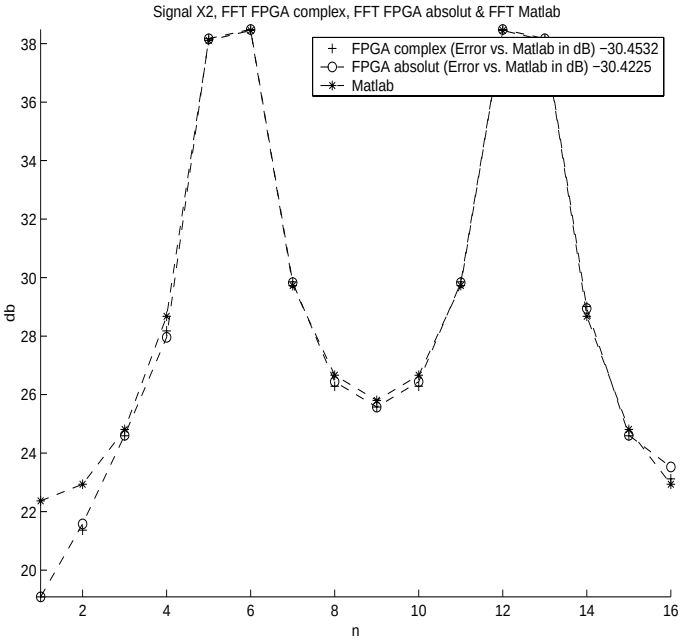


Fig.25. Output complex and absolute, signal 2 vs. Matlab

Uppgjord (även faktaansvarig om annan) - *Prepared (also subject responsible if other)*		Nr - *No.*		
EMWMSNN(Magnus Nilsson)		FX/D-2001:007		
Dokansv/Godkänd - *Doc respons/Approved*	Kontr - *Checked*	Datum - *Date*	Rev	File
EMW/FX/DC(Anders Wanner)		2001-02-12	A1	

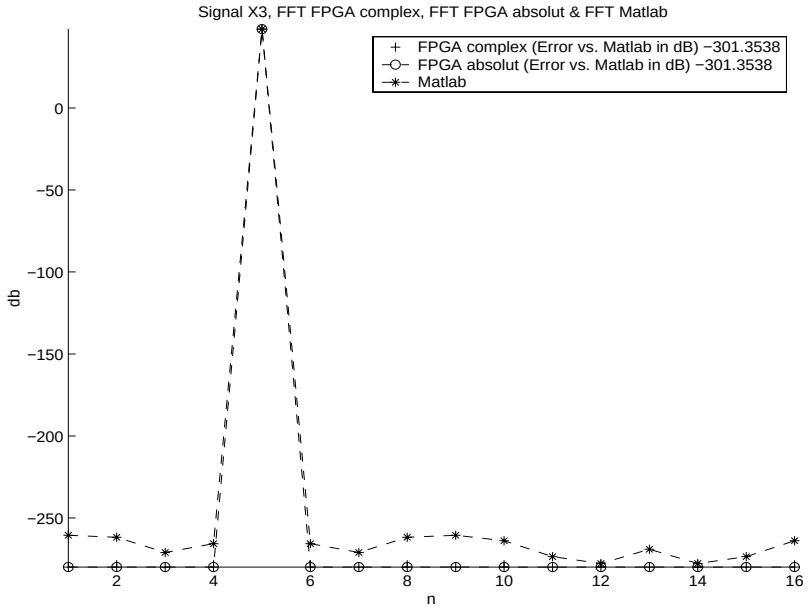


Fig.26. Output complex and absolute, signal 3 vs. Matlab

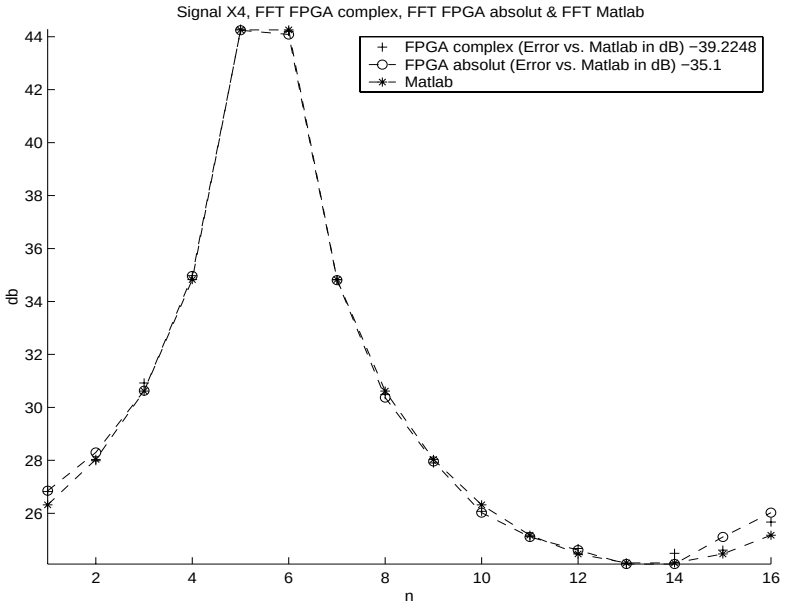


Fig.27. Output complex and absolute, signal 4 vs. Matlab

The Error value calculation was made according to the following formula

$$Error = 10\log 10\left(\frac{\sum(|FFT(Matlab)| - |FFT(FPGA)|)^2}{\sum(|FFT(Matlab)|)^2}\right)$$

(EQ 48)

10 CONCLUSION

- As the Radix-4 FFT algorithm utilizes less complex multipliers than the Radix-2 FFT algorithm, the Radix-4 algorithm is preferable for hardware implementation.
- A parallel programming approach seems to be the model when a real time system with high sampling rate is desired.
- To reach an acceptable level of phase error, it is desirable to use 16 bits precision on the input signal and the phase factor
- By using a separate clock with clock rate $clk_{Shift}=N*clk_{FFT}$, for the input and output shift registers, it would be possible to process a FFT on a signal of length N every clock cycle, clk_{FFT}.
- By using shared multiplier in the dragonflies, less CLB's is utilized, with the cost of longer execution time.

11 IDEAS FOR FURTHER STUDIES

As there are two FFT constructions of length N=64, one with precision 12 bits and one with precision 12, 14 and 16 bits, (dragonfly rank 1, 2 and 3) verified to be correct in Modelsim, it would be desirable to implement and verify those constructions when a circuit board with the necessary Xilinx Virtex-E circuit is available. Improvement and development of the input and output shift registers are also interesting as this would improve the bandwidth of a real time sampled signal when computing FFT.

12 REFERENCES

[1] Bergland, G. D.: 'A guided tour of the fast Fourier transform', IEEE Spectrum, July 1969

[2] Bracewell, R. N.: 'The Fourier Transform and its applications', The McGraw-Hill Companies, Inc, 2000, ISBN: 0-07-303938-1

[3] Brigham, E. O.: 'The fast fourier transform', Prentice-Hall, Inc, 1974, ISBN: 0-13-307496-X

[4] Cartwright, M.: 'Fourier Methods for mathematicians, scientists and engineers', Ellis Horwood Limited, 1990, ISBN: 0-13-327016-5

[5] Gray, R. M., Goodman, J. W.: 'Fourier Transforms, an introduction for engineers', Kluwer Academic Publishers, 1995, ISBN: 0-7923-9585-9

[6] Lasser, R.: 'Introduction to Fourier Series', Marcel Dekker, Inc., 1996, ISBN: 0-8247-9610-1

[7] Ma, Y., Wanhammar, L.: 'A hardware Efficient Control of Memory Addressing for High-Performance FFT Processors', IEEE Transaction on Signal Processing, Vol. 48, No. 3, March 2000

[8] Proakis, J. G.: 'Digital Signal Processing, Principles, algorithms and applications', Prentice Hall, Inc., 1996, ISBN: 0-13-394289-9

[9] Roche, C.: 'A Split-Radix Partial Input/Output Fast Fourier Transform Algorithm',

[10] Translogic: 'Ease and Eale User's Manual', Translogic BV, Ede, The Netherlands, 1998, http://www.translogiciccorp.com, (Acc 2001-02-05)

[11] Van Loan, C.:' Computational Frameworks for the Fast Fourier Transform', SIAM, 1992, ISBN: 0-89871-285-8

[12] Vretblad, A.: 'An introduction tp Fourier Analysis and some of its applications', Department of mathematics, Uppsala, Sweden, 1996, ISBN: 91-506-1171-2

[13] Xilinx Inc.: 'Xilinx Virtex-E Databook', http://www.xilinx.com, 2000-2001, (Acc 2001-02-05)

[14] Zonst, A. E.: 'Understanding the FFT', Citrus Press, 1995, ISBN: 0-9645681-8-7

Appendix A1 Ease block structure of Radix-4 FFT, N = 64

Appendix A2 Ease block structure of Radix-4 FFT, N = 64, shared mult

Appendix A3 Ease block structure of Radix-4 FFT, N = 16

Uppgjord (även faktaansvarig om annan) - *Prepared (also subject responsible if other)*		Nr - *No.*		
EMWMSNN(Magnus Nilsson)		FX/D-2001:007		
Dokansv/Godkänd - *Doc respons/Approved*	Kontr - *Checked*	Datum - *Date*	Rev	File
EMW/FX/DC(Anders Wanner)		2001-02-12	A1	

Appendix B Matlab code

Uppgjord (även faktaansvarig om annan) - *Prepared (also subject responsible if other)*		Nr - *No.*		
EMWMSNN(Magnus Nilsson)		FX/D-2001:007		
Dokansv/Godkänd - *Doc respons/Approved*	Kontr - *Checked*	Datum - *Date*	Rev	File
EMW/FX/DC(Anders Wanner)		2001-02-12	A1	

Appendix C Output listning