Power Consumption in DFTs for OFDM Systems

MASTER THESIS
APPLIED SIGNAL PROCESSING
AND IMPLEMENTATION (ASPI)

Group 1042
Peter August Simonsen
Jes Toft Kristensen

AALBORG UNIVERSITY

Institute for Electronic Systems
Fredrik Bajers Vej 7B
Telefon 96 35 98 36
Fax 98 15 36 62
http://www.esn.aau.dk

Title:
Power Consumption in DFTs for OFDM Systems

Project period:
P10, fall semester 2008

Project group:
ASPI 08gr1042

Members:
Peter August Simonsen
peter@augusts.dk
Jes Toft Kristensen
jes@buskefjomp.dk

Supervisors:
Anders B. Olsen
Jesper M. Kristensen

Copies: 6

Pages in report: 106

Appendices: 1 CD

Printed June 3, 2008

Abstract:

This Master Thesis of "Applied Signal Processing and Implementation" specialization at Aalborg University is an investigation of FFT algorithms in OFDM receivers and the algorithms power usage on customizable platforms.
The project focuses on mobile applications and cooperative radios, wherein only a part of the received frequency spectrum is needed. This can be exploited by special FFT algorithms to yield a lower operations count and intuitively a lower power consumption. However, what is not reflected in the operations count is the power-consumption of the controlling HW/SW. This thesis seeks to investigate the possibilities and tradeoffs, with regards to power usage, when computing a subset of the frequency spectrum, as opposed to the full spectrum.
Initially, the concept of cooperative radio and a signal model for OFDM is defined. Afterwards, two Fourier transform algorithms - a full Split-Radix FFT and an FFT algorithm computing only a subset of the spectrum (SFFT) - are examined and mapped to a Cyclone III FPGA architecture. Next, the power performance of each implementation is examined and an investigation into possible improvements is performed. In conclusion the algorithms are compared to a performance measure of computational complexity traditionally used to theoretically evaluate FFT algorithms.
The test results shows that the SFFT is not feasible with regards to power usage, without further improvements. These improvements include, among others, an enhanced power-off mechanism when subsystems are not in use. If a power-off state is introduced it is predicted that the SFFT becomes feasible and that computational complexity corresponds to the power usage for this implementation.

AALBORG UNIVERSITET

Institut for Elektroniske Systemer
Fredrik Bajers Vej 7B
Telefon 96 35 98 36
Fax 98 15 36 62
http://www.esn.aau.dk

Titel:
Power Consumption in DFTs for OFDM Systems

Projekt periode:
P10, forårssemester 2008

Projekt gruppe:
ASPI 08gr1042

Medlemmer:
Peter August Simonsen
peter@augusts.dk
Jes Toft Kristensen
jes@buskefjomp.dk

Vejledere:
Anders B. Olsen
Jesper M. Kristensen

Kopier: 6

Sider i rapport: 106

Antal bilag: 1 CD

Printet June 3, 2008

Synopsis:

Dette master-projekt på "Applied Signal Processing and Implementation" specialet ved Aalborg universitet er en undersøgelse af FFT algoritmer til OFDM modtagere og disse algoritmers energiforbrug på konfigurerbare platforme.
Projektet fokuserer på mobile kommunikation og kooperativ radio, hvor kun en del af det modtagne frekvensspektrum er nødvendigt at demodulere i modtageren. Dette kan udnyttes i specielle FFT algoritmer til at give en lavere beregningskompleksitet og intuitivt deraf et lavere effektforbrug. Men i beregningskompleksiteten er effektforbruget af det styrende HW/SW ikke inkluderet. Dette projekt undersøger de muligheder og afvejninger, mht. effektforbrug, når kun en del af frekvens-spektret beregnes, i modsætning til at beregne det fulde frekvensspektrum.
Til at begynde med introduceres kooperativ radio som koncept og en signalmodel for OFDM opstilles. Bagefter udforskes to FFT algoritmer - en Split-Radix FFT der beregner det fulde spektrum og en FFT algoritme der kun beregner en del af spektret (SFFT) - og disse implementeres på en Cyclone III FPGA arkitektur. Herefter udforskes hver implementations effektforbrug og mulige effektmæssige forbedringer undersøges. Afslutningsvist testes testes algoritmerne og resultaterne sammenlignes med beregningskompleksiteten, der traditionelt bruges til at evaluere FFT algoritmer.
Testresultaterne viser at SFFT algoritmen ikke er hensigtsmæssig mht. effektforbrug uden yderligere forbedringer. Disse forbedringer er blandt andet en forbedret power-off mekanisme, når undersystemer ikke er i brug. Hvis et power-off stadie introduceres, viser beregninger, at SFFT algoritmen bliver mere effektiv end Split-Radix FFT algoritmen og at beregningskompleksitet korrelerer med effektforbruget for denne implementation.

Preface

This report is documentation for the master thesis project in Applied Signal Processing and Implementation (ASPI) concerning "Power Consumption in DFTs for OFDM Systems" at the Institute of Electronic Systems at Aalborg University (AAU). The report is prepared by group 08gr1042 and spans from February 1st to June 4th, 2008. The project is supervised by Anders Brødløs Olsen and Jesper Michael Kristensen, both from Center for Software Defined Radio (CSDR) at AAU.

The report is divided into three parts. These parts correspond to the project phases of analysis, design or mapping, and evaluation of achieved results. The bibliography is found on page xiv with references to the bibliography in square brackets as in [08gr1042, 2008]. The cited source [08gr1042, 2008] is the accompanying CD attached to the inside of the report cover. This CD contains the code and test material produced during the project period and an electronic copy of this report in pdf.

Peter August Simonsen

Jes Toft Kristensen

Contents

List of Figures

List of Tables

Notation

The notation used throughout this report is documented below.

Symbol	Association
s	Mathematical variables in italics
10101001_b	A binary number, representing the integer 169
$\bar{\bar{\mathbf{A}}}$	The matrix A
$\overline{\mathbf{b}}$	The vector b
$\{x\}_y$	x modulus y
$\lceil a \rceil$	The expression of a ceiled
$\lfloor a \rfloor$	The expression of a floored
$\mu(x)$	The mean of x
$\mathbb{F}[a]$	The discrete Fourier transform of a
˜statement	The binary negated statement
[08gr1042, 2008, p. 42]	Bibliographic reference to index [08gr1042, 2008] page 42

Nomenclature

BS Base Station, page 1

cdf Cumulative Distribution Function, page 53

DFT Discrete Fourier Transform, page 20

FFT Fast Fourier Transform, page 3

FPGA Field Programmable Gate Array, page 4

FSM Finite State Machine, page 6

FSMD Finite State Machine with Datapath, page 6

HDL Hardware Description Language, page 34

ISI InterSymbol Interference, page 13

LAB Logic Array Block, page 32

LE Logic Element, page 32

LSB Least Significant Bit, page 56

MAC Multiply and ACcumulate, page 72

MS Mobile Station, page 1

OFDM Orthogonal Frequency-Division Multiplexing, page 2

pdf Probability Density Function, page 52

PLL Phase Locked Loop, page 31

SFFT Sørensen FFT, page 23

SQNR Signal to Quantization Noise Ratio, page 87

Twiddle Factor , page 20

Bibliography

The Project Group 08gr1042, June 2008. Additional materials for the project can be found on the accompanying CD.

AAU. *JADE Project Deliverable: D3.1.1*. Aalborg University, 2004.

Agilent. *Agilent 34401A Multimeter - Product Overview*. Agilent Technologies, 2007. get from: `http://cp.literature.agilent.com/litweb/pdf/5968-0162EN.pdf`.

Altera. *An OFDM FFT Kernel for WiMAX - Application note 452*. Altera Corporation, 1.0 edition, 2007a. get from: `http://www.altera.com/literature/an/an452.pdf`.

Altera. *Cyclone III FPGA Starter Kit User Guide*. Altera Corporation, 1.0.0 edition, 2007b.

Altera. *Nios II Processor Reference Handbook*. Altera Corporation, 2008a. get from: `http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf`.

Altera. *Cyclone III Device Handbook*. Altera Corporation, 2007c.

Altera. *Quartus II Version 7.2 Handbook,*. Altera Corporation, 7.2.0 edition, 2007d.

Altera. *Quartus II Device Support Release Notes*. Altera Corporation, 2008b. get from: `http://www.altera.com/literature/rn/rn_qts_72sp2_dev_support.pdf`.

Altera. *FPGA Power Management and Modeling Techniques*. Altera Corporation, 1.0 edition, 2007e.

Jeffrey G. Andrews. *Fundamentals of WiMAX*. Prentice Hall, 2007. ISBN 0-13-222552-2.

Abdellatif Bellaouar and Mohammed I. Elmasry. *Low-Power Digital VLSI Design*. Kluwer Academic Publishers, 1st edition, 1995. ISBN 0-7923-9587-5.

David M. Bradley and Ramesh. C. Gupta. *On the Distribution of the Sum of n Non-Identically Distributed Uniform Random Variables*. Department of Mathematics and Statistics, University of Maine, Orono, ME, 2007. URL `citeseer.ist.psu.edu/449216.html`.

Suvra Sekhar Das. *Techniques to Enhance Spectral Efficiency of OFDM Wireless Systems*. Center for TeleInFrastruktur (CTIF), September 2007. ISBN 87-92078-07-9.

P. Duhamel and H. Hollmann. *'Split Radix' FFT Algorithm*. IEEE, 1 edition, 1983. Electronics Letters, 5th January 1984, Vol. 20, No. 1.

Pierre Duhamel. *Implementation of "Split-Radix" FFT Algorithms for Complex, Real and Real-Symmetric Data*. IEEE, 1986. IEEE Transactions on acoustics, speech and signal processing, vol. ASSP-34, No. 2, April 1986.

Daniel D. Gajski. *Principles of Digital Design*. Prentice Hall, 1997. ISBN 0-13-242397-9.

Steven G. Johnson and Matteo Frigo. *A modified split-radix FFT with fewer arithmetic operations*. IEEE, 1st edition, 2007. IEEE Trans. Signal Processing 55 (1), 11-119.

Youngok Kim and Jaekwon Kim. *Low Complexity FFT Schemes for Multicarrier Demodulation in OFDMA Systems*. IEICE, 1st edition, 2007. IEICE Transactions on Communication, November 2007, Vol. E90-B, No. 11, pp. 3290-3293.

E. Lawrey. *Multiuser OFDM*. ISSPA, 1st edition, 1999. Proc. IEEE International Symbiosum on Signal Processing and its Applications, August 1999, Vol. 2, pp. 761-764.

John D. Markel. *FFT Pruning*. IEEE, 1971. IEEE Transactions on Audio and Electroacoustics, Vol. AU-19, No. 4, December 1971.

Yannick Le Moullec. *DSP Design Methodology*. AAU, 2007. Lecture notes for mm1 of course in DSP Design Methodology, ASPI8-4 `http://kom.aau.dk/~ylm/aspi8-4/aspi8-4-part1-2007.pdf`.

Charles D. Murphy. *Low-Complexity FFT Structures for OFDM Trancievers*. IEEE, 2002. IEEE transactions on communication, vol.50, no. 12, December 2002, pp. 1878-1881.

Erik L. Oberstar. *Fixed-Point Representation & Fractional Math*. Oberstar Consulting, 1.2 edition, 2007.

Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*. Prentice-Hall Inc., 2nd edition, 1998.

Henrik Schulze and Christian Lüders. *Theory and Applications of OFDM and CDMA*. John Wiley & Sons, Ltd., 2005. ISBN 0-470-85069-8.

K. Sam Shanmugan and A. M. Breipohl. *Random Signals, Detection, Estimation and Data Analysis*. Wiley and Sons, 1st edition, 1988. ISBN 0-471-81555-1.

David P. Skinner. *Pruning the Decimation in-Time FFT Algorithm*. IEEE, 1976. IEEE transactions on acoustics, speech and signal processing, April 1976, pp. 193-194.

A. N. Skodras and A. G. Constantinides. *Efficient computation of the split-radix FFT*. IEEE, 1 edition, 1992. IEEE PROCEEDINGS-F, Vol. 139, No. 1, FEBRUARY 1992.

Henrik V. Sørensen and C. Sidney Burrus. *Efficient Computation of DFT with Only a Subset of Input or Output Points*. IEEE, 1st edition, 1993. IEEE Transactions on Signal Processing, Vol 41. No 3, March 1993.

John F. Wakerly. *Digital Design, Principles and Practices*. Prentice Hall, 3rd edition, 2001. ISBN 0-13-090772-3.

Chapter 1

Introduction

In the introduction the purpose, objectives and methodology of the project is presented. First, an informal introduction to multiuser OFDM is given along with the motivation for examining FFT algorithms and FPGA implementations of these in a power consumption context. Next, fundamental project delimitations are introduced regarding FFT algorithms for examination and FPGA platform for implementation are presented. Further discussions of these delimitations are carried out in the analysis part of the report. Finally, the project methodology and report structure is presented.

1.1 Cooperative Radio and Multiuser OFDM

Figure 1.1 shows a wireless communication scenario, where multiple users, or mobile stations (MS), are communicating with a base station (BS) and each other.

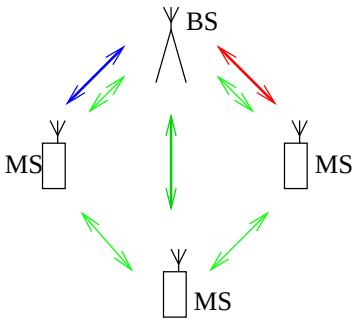

Figure 1.1: *Multiuser Cooperative Radio Scenario. The communication path may be either directly from base station to mobile station or data can be relayed through other mobile stations to get to the destination.*

The inter MS communication may both be data exchanged between the local MSs, or data from the base station relayed through one MS to the destination. If for instance the direct channel from BS to destination MS cannot accommodate the required bandwidth. Such a system requires some cooperation between devices and a method for dividing the channel between BS and MS links.

In Orthogonal Frequency-Division Multiplexing (OFDM) the spectrum or channel is divided into a set of mutually orthogonal subcarriers, which are used to modulate the data to be transmitted. In Multiuser OFDM the subcarriers may be assigned to different communication paths as exemplified in figure 1.2, where the spectrum are divided into blocks of subcarriers for each path.

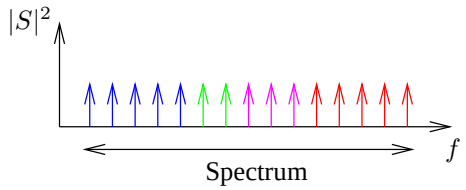

Figure 1.2: *Example of subcarrier allocation for each communication path, where each color is assigned to a different path.*

In OFDM the Discrete Fourier Transform (DFT) and it's inverse counterpart plays a significant role, as it is used to modulate data symbols onto the subcarriers in the transmitter and demodulate the data at the receiver. A more elaborate analysis of an OFDM system and the DFT function herein is presented in chapter 2.

1.2 Project Purpose and Objectives

In this project, focus is turned to the DFT block of the OFDM receiver used to demodulate the system subcarriers. A range of low complexity Fast Fourier Transform algorithms have been developed to reduce to number of computations when calculating the DFT, both for calculation of all subcarrier transforms and for calculation of only a subset of the subcarrier transforms, which is relevant when only the data designated for one MS is of interest.

Classic evaluation methods of Fourier transform algorithms intended for OFDM systems are based on number of additions and multiplications spent calculating the actual transform [Murphy, 2002]. While this measure gives a theoretic indication of which algorithm is optimal in a given situation, the measure based on needed calculation only, does not take into account the control structures needed to manage and ensure timing of input and output data for the calculation units in an implementation of the algorithms.

When turning focus to implementations of algorithms, the cost function changes from counting calculations to a combination of algorithm execution time, hardware area utilization (e.g. logic units in FPGAs, memory usage), and power consumption:

$$\mathbf{C}ost \;=\; \mathbf{P}ower \times \mathbf{A}rea \times \mathbf{T}ime \tag{1.1}$$

Since keeping power consumption as low as possible is key in battery powered mobile devices, the purpose of this project is to evaluate different DFT schemes for OFDM or Cooperative Radios with regards to power consumption in FPGA implementations. Requirements for the DFT is taken from the WiMAX standard, to set constraints on calculation time which is based on a standard targeted at mobile devices.

The goal of applying a power consumption measure to FPGA implementations of Fourier transform algorithms is to investigate how the theoretical measure of computational complexity compares to performance achievements of actual implementations. Evaluating the performance of these implementations, across a variable number of needed subcarriers and using a power measure, will show in which situation it is advantageous to use each of the investigated algorithms.

The results of these implementation evaluations are finally compared with the computational complexity measure to determine if each of the investigated algorithms have more or less relevance in actual implementations than their computational complexity suggest.

1.3 Problem Specification

How well does the performance measure of computational complexity compare to power consumption in FPGA implementations of DFT algorithms for multiuser OFDM?

1.4 DFT Algorithms

For this project two Fast Fourier Transform (FFT) algorithms are chosen for comparison. The algorithms considered in this project are:

- **Split-Radix Fast Fourier Transform (SRFFT):**

  One approach is to calculate the full transform, regardless of how many subcarriers, that are of interest. Several algorithm exist which can perform this task. The radix-2 FFT is a recursive decomposition into two DFTs of N/2 length, the radix-4 makes use of decompositions into four N/4 DFT, and the Split-Radix FFT employs decomposition into one N/2 and two N/4 DFTs. The SRFFT has proven to feature one of the lowest computational complexities of the mentioned algorithms [Duhamel and Hollmann, 1983] and is therefore chosen for investigation in this project.

- **"Sørensen" Fast Fourier Transform (SFFT):**

  As mentioned above, one user may only need to receive the data sent using a subset of the available subcarriers. Therefore methods calculating only a subset of the FFT have been developed, where the SFFT [Sørensen and Burrus, 1993] reduces the number of calculations by only calculating decomposition into a set of small FFTs and then recombining the results for only the subset of subcarrier that are of interest.

Each of the investigated algorithms are elaborated in sections 3.3 and 3.4

1.5 Implementation Prerequisites and Constraints

The DFT algorithms are examined when implemented on a Field Programmable Gate Array (FPGA) platform . The configurability of FPGAs allows for faster development of systems combining predeveloped building blocks, like a DFT, to compose a system fitting the application in question. With FPGA families emerging developed for low power consumption (e.g. Altera Cyclone III) FPGAs become applicable in battery powered mobile devices. Therefore, an FPGA platform is used to evaluate the power consumption of the examined DFT algorithms.

The selected platform is the Altera Cyclone III Starter Kit. This selection is based on the toolchain, support in the form of the Quartus II and associated software, and knowledge available at Aalborg University.

Using a development kit allows for focusing on the mapping of algorithms onto the architecture, and evaluating and comparing these implementations. This focus comes at the cost of reduced possibilities for dimensioning the system to fit the requirements and may introduce some unnecessary overhead for the design to fit the hardware. Still, the development board provides a well defined platform for comparison of the algorithms and the resources saved from not designing the platform can be used to focus on answering the problem specification stated in section 1.3.

1.6 Project Methodology

The purpose of the design methodology is to supply a structured approach to the analysis, design and evaluation of results obtained in the project. Therefore, the following structure of analysis, design and evaluation also reflects the structure of this report. The project methodology is depicted in figure 1.3.

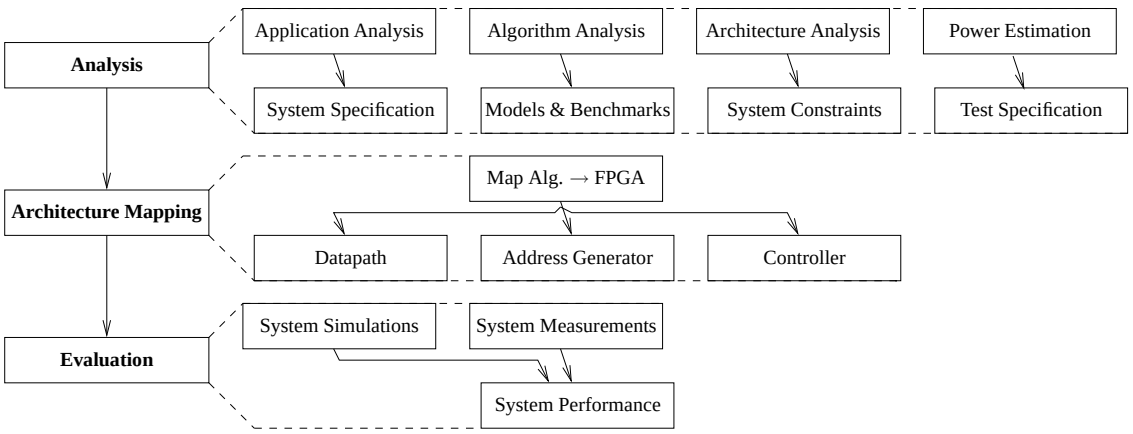

Figure 1.3: *Project Methodology overview. For each part of the project - Analysis, Mapping and Evaluation - tasks and results are depicted. Architecture Mapping and Evaluation are repeated for each algorithm.*

1.6.1 Analysis

The project analysis makes use of the three domains of the A^3 model [Moullec, 2007], Application, Algorithm, and Architecture, shown in figure 1.4, to divide the system analysis into three main parts:

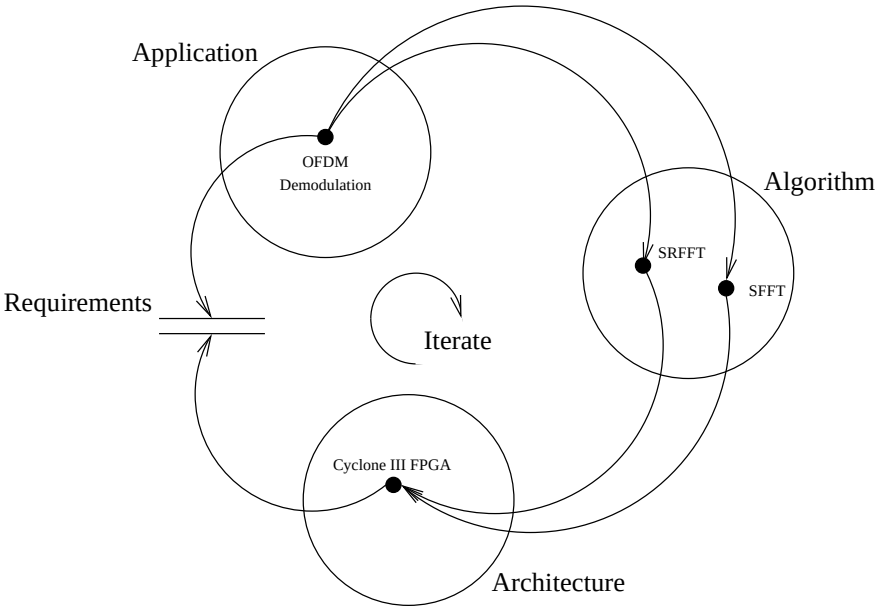

Figure 1.4: *A^3 model for project.*

- **Application:**
  In the application domain, an analytical OFDM model is presented to determine the context in which, the FFT implementations are to function. Based on this system model, the functional requirements of the FFT block, and specifications and constraints of a test system for validation is determined.

- **Algorithm:**
  In the algorithms domain, the two Fourier transform algorithms used to solve the specified task of the OFDM FFT block are analyzed. The analysis focus on derivation of the structure of computations and computational complexity of each algorithm. The computational complexity measure of each algorithm is evaluated in same test cases as are defined for power analysis of the implementations for comparison and evaluation. Furthermore, each algorithm is modelled in C where an outline of a datapath and control structure for the following mapping is designed.

- **Architecture:**
  In the architecture domain, the FPGA hardware used to implement the investigated FFT algorithms are analyzed. The characteristics of the FPGA development kit of choice, i.e.

available hardware and system limitations, are examined. Next the development tools and methods available for the platform are described, as are the possibilities for simulating and measuring the power consumption of each algorithm implementation.

1.6.2 Architecture Mapping

The system design in this project is concerned with mapping each of the algorithms onto the FPGA architecture. The mapping is done using a datapath and control structure approach, as shown in figure 1.5. This mapping method is based on the Finite State Machine with Datapath (FSMD) approach to digital design presented in [Gajski, 1997, page 320-322].

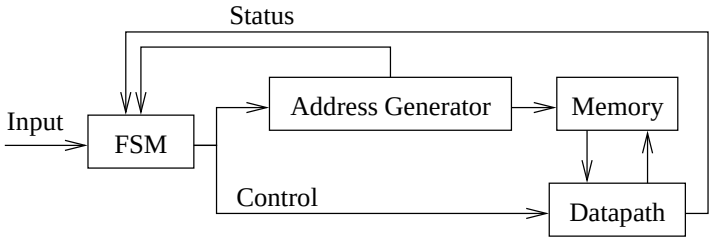

Figure 1.5: *General structure model for design mapping of algorithms using a finite state machine for controlling an address generator and datapath.*

Each mapping design consists of a datapath, where the algorithm calculation units are contained. The control structure consists of a Finite State Machine (FSM) , which is used to setup the datapath to do the relevant operations on the data, and an address generator, used to retrieve data from memory.

This approach differs slightly from [Gajski, 1997, page 320-322] since program counters keeping track of algorithm progress and the logic used to generate addresses for memory access are extracted from the datapath and handled explicitly in the project. This additional partitioning of the datapath is done in order be able to design the calculation units of the data path separately and next design memory handling units to fit these calculation units.

When partitioning the design in the algorithm part II, the design is divided into three domains and shown in figure 1.6 on the facing page; environment, control and data domains.

The environment domain contains peripherals and interfaces to the implemented algorithms. As the algorithms only perform the FFT task in the OFDM demodulation, see figure 2.1 on page 12, the environment domain provides a convenient representation of the surrounding system and requirements which are somewhat common to the data and control domains. In effect, the environment domain thus contains the memory part of figure 1.5 and system clock domains. The data domain contains the datapaths of the mapped algorithm and the control domain contains the FSM controller and address generators for memory access.

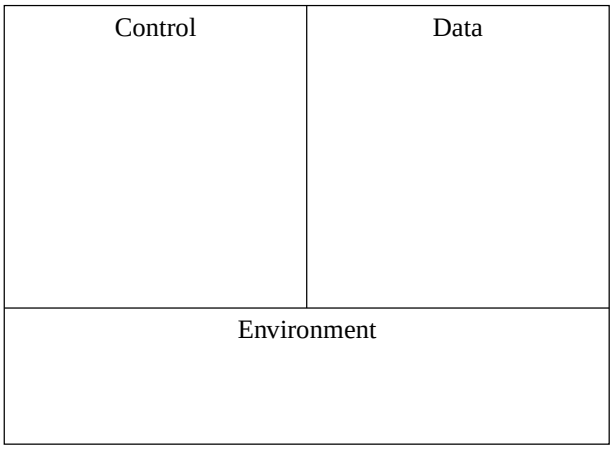

Figure 1.6: *The abstraction model used to describe algorithm mapping. The model contains the three domains: environment, control path and datapath. These domains and their interfaces are used as a basis for describing the mapping from mathematical algorithm to FPGA implementation.*

1.6.3 Evaluation

The final part of the project is concerned with verifying the functionality of the implemented algorithms and evaluating the systems with regards to the power performance measure setup in the analysis. This evaluation is carried out using both the available power analysis tool provided by the FPGA manufacturer and by measurements of the FPGA power consumption in a set of test scenarios that ultimately allows for comparing the obtained results with the theoretical performance derived from algorithm computational complexity.

This comparison is used to answer the problem specification stated in section 1.3, and to evaluate the coherence between algorithm computational complexity and power consumption.

Part I

Analysis

This part moves the problem specification previously defined through the application level. This includes a closer examination of the system model, examination of the Fourier transform algorithms and the FPGA, both functional and power-wise

Initially the concept of OFDM is introduced and the system is specified. Afterwards two FFT algorithms are examined mathematically and finally compared complexity-wise. Chapter 4 introduces the FPGA platform and associated tools while chapter 5 presents a power performance simulation and measurement.

Contents

Chapter 2

Application Analysis

The application analysis presents the system model which is an OFDM downlink transmission scheme, that can be used in cooperative radios. Based on this model a test system is specified to enable testing the functionality of the proposed DFT algorithms.

2.1 System Model

This section is a general introduction to the OFDM downlink scenario considered in this project. First the concept of orthogonal frequencies is explained, followed by how this principle is used to communicate data symbols from a base station through a wireless channel to a mobile station, where only a subset of the subcarriers is of interest. The main sources on which the following presentation of OFDM is based are [Schulze and Lüders, 2005, sec. 4.1] and [AAU, 2004, chap. 2].

2.1.1 Orthogonal Frequencies

Orthogonal Frequency Division Multiplexing (OFDM) is a framework for multicarrier transmission where several data symbols are transmitted at the same time by modulation with orthogonal subcarriers. In a system with N subcarriers, the baseband signal for one OFDM symbol period (T_u) may be written as:

$$s(t) = \sum_{n=-N/2}^{N/2-1} X_n e^{j\frac{2\pi nt}{T_u}} \quad , 0 \leq t \leq T_u \tag{2.1}$$

where X_n is the n'th data symbol. In the frequency domain the signal has the form of:

$$S(\omega) = \frac{1}{\sqrt{T_u}} \sum_{n=-N/2}^{N/2-1} X_n \delta(\omega - \frac{n}{T_u}) \tag{2.2}$$

The OFDM symbol duration (T_u) being an integer m n multiple of each subcarrier duration, $T_c \cdot n = T_u$, is the key to orthogonality between the subcarriers. This orthogonality may be

proven by calculating the cross correlation between two subcarriers, which possess the properties of spacing and time duration mentioned above. The values of the data symbols X_n may be left out of this calculation since they are constant over the entire symbol interval.

$$\int_0^{T_u} (e^{j\frac{2\pi n_1 t}{T_u}})^* \cdot (e^{j\frac{2\pi n_2 t}{T_u}})\, dt \tag{2.3}$$

$$= \int_0^{T_u} e^{j\frac{(2\pi n_2 - n_1)t}{T_u}}\, dt \tag{2.4}$$

$$= \delta(n_2 - n_1) \tag{2.5}$$

Equation (2.5) show that two subcarriers only correlate if n_1 and n_2 are equal, i.e. are located at the same frequency.

2.1.2 An OFDM Downlink System

Figure 2.1 shows the structure of an OFDM downlink system.

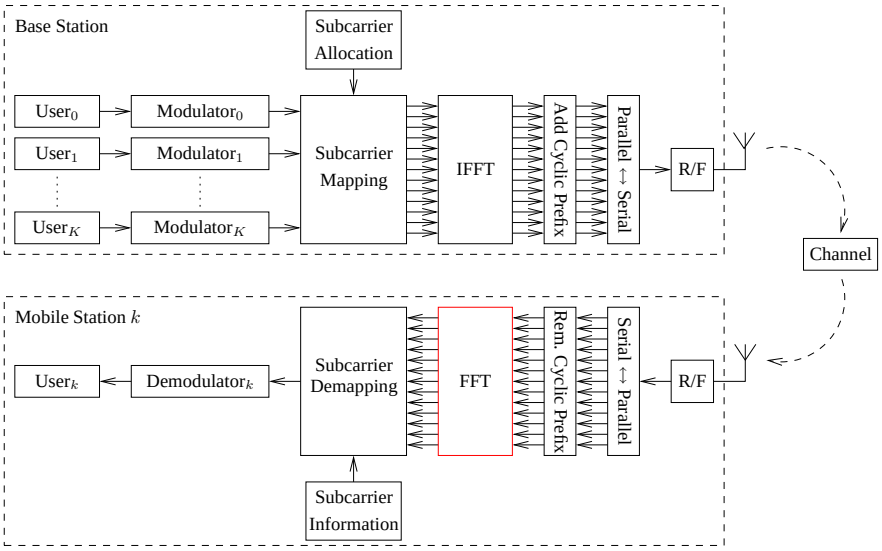

Figure 2.1: *System model of general OFDM downlink. The red marked FFT block of the mobile station is to be implemented and evaluated.*

Base Station

At the Base Station, data symbols are distributed across the subcarriers assigned to each user. The mapped data symbols are next mixed onto the assigned subcarriers by use of an inverse Fourier transform. This produces a signal of the same length as the FFT, which has the form of equation (2.1).

In the next part of the system, the output of the IFFT block is extended by a cyclic prefix:

$$s'(t) = \begin{cases} s(t + T_u - T_g) & 0 \leq t < T_g \\ s(t - T_g) & T_g < t < T_s \\ 0 & \text{Otherwise} \end{cases} \tag{2.6}$$

T_g is the length of the cyclic prefix, which must have the following property:

$$\delta\delta_t + \tau_{max} < T_g \tag{2.7}$$

where δ_t is the maximum time synchronization offset between BS and MS and τ_{max} is the maximum delay spread of the wireless channel. By adding this cyclic prefix it is ensured that the MS always is able to receive a sample of length T_u, which does not feature Intersymbol Interference (ISI) caused by the reception of multiple reflections of s' at the MS.

Finally, the signals for each T_s interval are concatenated and modulated onto a carrier sinusoid with frequency f_c to form the BS output signal, o(t):

$$o(t) = \sum_{s=0}^{S-1} s'(t - sT_s) \cdot e^{2\pi f_c t} \tag{2.8}$$

where S is the number of T_s symbol intervals to be transmitted.

Wireless Channel

Passing the signal, $o(t)$, through a wireless channel is analogue to passing the signal through a FIR filter, which models the fading and reflecting of the signal which is experienced from BS to MS. Such a filter $h_{u,s}(t)$ for a specific user u and OFDM symbol period s may be written as:

$$h_{u,s}(t) = \sum_{l=0}^{L-1} h_{u,l}[s]\delta(t - \tau_l), \quad \text{where} \quad sT_s < t < (s+1)T_s \tag{2.9}$$

where l is one of the L multipaths received at the MS. $h_{u,l}[s]$ is the complex gain of the l'th multipath of the u'th user for OFDM symbol interval s.

In addition to the filtering characteristics of the multipath fading channel, the received signal at the MS will feature a noise contribution, $v(t)$. At the u'th mobile station, the received signal, $r(t)$ is thus:

$$r(t) = \Re\left\{(s'(t) * h_{u,s}(t))e^{j2\pi f_c t}\right\} + v(t), \quad \text{where} \quad sT_s < t < (s+1)T_s \tag{2.10}$$

Mobile Station

The first task at the receiver is to down convert $r(t)$ from the carrier frequency band to baseband. Since perfect synchronization of time and system clock frequency cannot be assumed, the received baseband signal becomes:

$$r'(t) = (s'(t - \delta_t) * h_{u,s}(t))e^{j\delta_\omega t} + v'(t), \quad \text{where} \quad sT_s < t < (s+1)T_s \tag{2.11}$$

where δ_t is the time synchronization mismatch between BS and MS and δ_ω is the system frequency mismatch.

Next the cyclic prefix is removed from each T_s long signal block, to get a received version of $s(t)$ for the s'th symbol, denoted $y_s(t)$:

$$\begin{aligned} y_s(t) &= r'(t' + T_g - sT_s), \quad \text{where} \quad 0 < t < T_s - T_g && (2.12) \\ &= (s'(t - \delta_t + T_g - sT_s) * h_{u,s}(t))e^{j\delta_\omega t} + v'_s(t) && (2.13) \end{aligned}$$

Here it becomes clear that if the cyclic prefix duration, T_g, satisfies the requirement of Equation (2.7), removing the cyclic prefix will effectively remove any intersymbol interference introduced from both time synchronization mismatch between BS and MS, and channel delay spread, since these effects will be constrained to the first T_g part of each T_s interval.

Next, to extract the data intended for MS$_k$, $y_s(t)$ is correlated with each of the M_k the subcarrier frequencies assigned to user k:

$$Y_s[m] = \frac{1}{\sqrt{T_u}} \int_0^{T_u} y_s(t) e^{-j(\omega_m + \delta_\omega)t}\, dt, \quad \text{where} \quad \leq m \leq M_k - 1 \tag{2.14}$$

Equation (2.14) is recognized as the Fourier transform of $y_s(t)$ evaluated in a single subcarrier frequency, $\omega_m + \delta_\omega$. To uncover the components of $Y_s[m]$ we start be evaluating the Fourier transform of $y_s(t)$, which may be written as:

$$\begin{aligned} Y_s(\omega) &= \mathbb{F}\left[y_s(t)\xi(t)\right] && (2.15) \\ &= \mathbb{F}\left[\left((s'_s(t - \delta_t + T_g - sT_s) * h_{u,s}(t))e^{j\delta_\omega t} + v'_s(t)\right)\xi(t)\right] && (2.16) \\ &\approx \mathbb{F}\left[\left((s_s(t - \delta_t) * h_{u,s}(t))e^{j\delta_\omega t} + v'_s(t)\right)\xi(t)\right] && (2.17) \\ &= e^{-j\omega\delta_t}\mathbb{F}\left[s_s(t) * h_{u,s}(t)\right] * \delta(\omega - \delta_\omega) * \Xi(\omega) + \mathbb{F}\left[v'_s(t)\right] * \Xi(\omega) && (2.18) \end{aligned}$$

where the term $e^{j\omega\delta_t}$ is the constant phase shift introduced by the time synchronization mismatch, $\xi(t)$ is a rectangular window of length T_u with corresponding Fourier transform $\Xi(\omega)$:

$$\Xi(\omega) = T_u \cdot e^{j\pi\omega T_u} \cdot \text{sinc}(\omega T_u) \tag{2.19}$$

Equation (2.18) may be rewritten to:

$$\begin{aligned} Y_s(\omega) &= e^{-j\omega(\delta_t + \pi T_u)} \cdot \Bigg[\ldots && (2.20) \\ &\ldots \sum_{k=-N/2}^{N/2-1} X_s[k] H_{u,s}[\frac{k}{T_u}] \text{sinc}\left(T_u(\omega - \frac{k}{T_u} - \delta_\omega)\right) + N_s(\omega)\Bigg] \end{aligned}$$

where $N_s(\omega) = \mathbb{F}\left[v'_s(t)\right] * \Xi(\omega)$.

Since $Y_s[m] = Y_s(\omega_m)$ and $\omega_m \in \frac{T_u}{n} | 0 \leq n \leq N - 1$ we get:

$$Y_s[m] = e^{-j\omega_m(\delta_t + \pi T_u)} X_{u,s}[m] H_{u,s}[m] + N_s[m]; \delta_\omega = 0; \tag{2.21}$$

where $X_{u,s}[m]$ is the m'th symbol for user u in OFDM symbol interval s, located at ω_m, $H_{u,s}[m] = H_{u,s}(\omega_m)$ and $N_s[m] = N_s(\omega_m)$. Finally, we include the constant phase shift, $e^{-j\omega_m(\delta_t+\pi T_u)}$, in the channel gain coefficient, $H_{u,s}[m]$, to get $H'_{u,s}[m]$:

$$H'_{u,s}[m] = e^{-j\omega_m(\delta_t+\pi T_u)} H_{u,s}[m] \tag{2.22}$$

since terms will be estimated together, when estimating the equalization factor, $Z_{u,s}[m]$, used to produce a estimate of $X_{u,s}[m]$:

$$\hat{X}_{u,s}[m] = Z_{u,s}[m]H_{u,s}[m]X_{u,s}[m] + Z_{u,s}[m]N_s[m] \tag{2.23}$$

This concludes the analytical system model, presenting the process of modulating and transmitting a symbol from the base station to estimation of a symbol at the mobile station.

2.2 Subcarrier Allocation Schemes

The task of allocating subcarriers for multiple users in OFDM systems to maximize spectrum utilization is an entire field of study of it's own. The topic of subcarrier allocation is not discussed in detail here, but the principal structures of subcarrier locations are described in order to outline the possible conditions under which the DFT is utilized.

Three basic structures of subcarrier allocations exist (Kim and Kim [2007] and Lawrey [1999]), which are shown in figure 2.2 on the following page and described below:

- **Clustered Subcarrier Allocation (CSA):**
  With CSA the subcarriers are allocated as a set on consecutive subcarriers are assigned to each user. This allocation may be static for the entire BS-MS link period or a frequency hopping sequence may be used to increase the mean performance of the link, by avoiding static location of the allocated subcarriers in a spectrum null.

- **Comb Spread Subcarrier Allocation (CSSA):**
  Another way of avoiding subcarrier locations in a spectrum null is to allocate the subcarriers in a comb structure, where the subcarriers are spread over the entire spectrum.

- **Adaptive Subcarrier Allocation (ASA):**
  Finally, ASA uses spectrum sensing to determine the locations in the spectrum to place the subcarriers to maximize throughput. This way each user will always be assigned the best channel available, thus increasing the overall system performance.

2.3 System Specification

In order to focus on the implementation challenges and performance achievements of the OFDM downlink FFT block, marked with red color in figure 2.1, the following delimitations are introduced in the test system used for validating the proposed implementations:

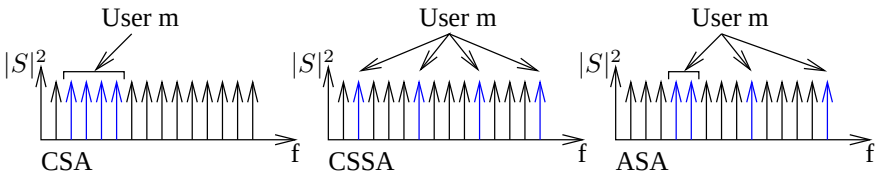

Figure 2.2: *Subcarrier allocation principles. CSA allocates subcarriers for a user as a block of consecutive subcarriers. CSSA spreads the allocated subcarriers evenly across the spectrum. ASA allocates the subcarriers adaptively to maximize channel throughput, thus no structure of subcarrier locations are assumed.*

- **Ideal RF transmission and channel**

  The RF carrier modulation and demodulation and channel effects applied to a signal transmitted through a wireless channel are not included in the test system, since the FFT block does not process the received signal to neutralize the transmission effects.

- **Ideal synchronization between BS and MS**

  An extension of ideal transmission between BS and MS is the assumption of ideal synchronization in time and frequency. Time synchronization eliminates the need for the test system to include handling the addition and removal of cyclic prefixes, and frequency synchronization, or system clock matching, preserves the orthogonality between subcarriers.

- **Data symbols to be transmitted are QPSK modulated**

  The test system input is randomly generated QPSK symbols with amplitude $||X_n||^2 = 1$. The system input is only used for validation of the results calculated by the FFT block, thus there need not be an elaborate coding and decoding of payload data.

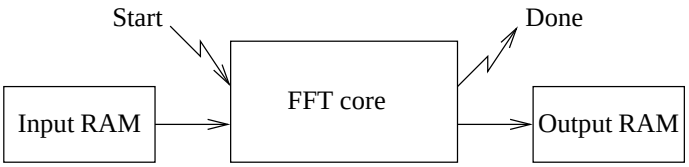

Figure 2.3: *Principal blocks of test system. The system consists of a FFT core, RAM for input and output data, a signal starting calculation of the FFT and a signal for indicating when calculation has finished.*

The system to be implemented on a FPGA platform is shown in figure 2.3. This test system consists:

- Memory containing a test vector with MatLAB simulations of a mobile station FFT block input frame and a second vector containing subcarrier locations. When the FFT is calculated, the results are placed in the same memory as the input.
- FFT core for demodulating the necessary subcarriers.
- Control signals for indication of input data ready (start) and result (output data) ready (done).

Finally, a set of global system constraints are taken from the WiMax standard [Das, 2007, table 2.3] to define the basic system structure with regards to length of the full FFT length and the OFDM symbol duration. These constraints are then used define the system parameters for completion time and number of subcarriers to be demodulated. These constants are shown in table 2.1.

Parameter	System Constraint	WiMAX standard range
Frame duration [ms]	2	min. 2
Full FFT length	1024	128 - 2048
FFT Time [ms]	0.5	0.1
Number of subcarriers	1 - 250	-

Table 2.1: *System constraints for FFT size and timing performance.*

Initially, the system constraints was set using the WiMAX frame duration as the basis for choosing a maximum FFT completion time. As seen in the table, this approach is erroneous, since the actual OFDM symbol duration in WiMAX is approximately 0.1 ms [Andrews, 2007, table 2.3]. Still, the system constraints for FFT completion time and a second constraint of one symbol per 2 ms allows for better margins to investigate and expose the effects of clock frequency and idle time on system power consumption in the test and design space exploration chapters 9 and 10.

Furthermore the number of subcarriers to be demodulated has been limited to a maximum of 250. This limit has been set since some subcarriers will be reserved for pilot channel and since it is unlikely that one user, accessing the system, will be assigned all system subcarriers. Thus a system able to demodulate all subcarriers at once would be over dimensioned.

With an overview of a generic OFDM system and the specification of the functional performance of the FFT block investigated, the next chapter examines the FFT algorithms of interest to establish a basis for the mapping of these algorithms onto the FPGA architecture.

Chapter 3

Fourier Transform Algorithms

This chapter serves as an introduction to the selection of FFT algorithms and subsequent analysis of each algorithm. The chapter starts with a discussion of the algorithm selection and continues with the derivation of each algorithm. At last the computational complexities of the algorithms is compared.

3.1 Algorithm Selection

The literature on optimized FFT computation focuses heavily on computational complexity and not directly on power usage. Thus to select candidate algorithms, the computational complexity is used as comparison parameter, as a means to exploit the existing literature.

The algorithm with lowest computational complexity for full length FFT computation, as of this writing, is the work proposed in [Johnson and Frigo, 2007]. This is based on the split-radix FFT proposed in [Duhamel and Hollmann, 1983]. Unfortunately the work presented in [Johnson and Frigo, 2007] uses a recursive formulation with different behavior depending on recursion levels, which is unsuited for hardware implementation compared to the formulation in [Duhamel and Hollmann, 1983] which has a more straightforward formulation. Instead the original algorithm as presented in [Duhamel and Hollmann, 1983] and [Skodras and Constantinides, 1992] is selected, as this algorithm presents a simple structure and is deemed representative for a fast full-length FFT. Thus a basic split-radix FFT is selected for the full-length FFT.

For computing individual output points of an FFT, three different approaches are considered. These are the direct computation via implementation of the Fourier transform integral [Oppenheim and Schafer, 1998, p. 561], computation via Transform Decomposition as in [Sørensen and Burrus, 1993] and the pruning schemes presented in [Markel, 1971] and [Skinner, 1976].

The pruning schemes presented by Markel and Skinner is based on the radix-2 FFT algorithms and reduces the computational complexity by examining the needed input/output points and determine the resulting data dependencies, which results in knowledge of which butterflies not to compute.

The transform decomposition divides the FFT transform into several smaller transforms, each of lower complexity, and then recombines the result to compute the needed output points.

Of the pruning and transform decomposition schemes, the transform decomposition is selected as it is the most flexible with regards to frequency placement of the output points. The pruning method looses efficiency in this regard as placement of output points could produce data dependencies in the radix-2 FFT where all the butterflies are needed, thus loosing the advantage of the pruning method. Furthermore the transform decomposition is shown in [Sørensen and Burrus, 1993, Fig. 12] to have a generally lower computational complexity.

Examining the computational complexity of the direct computation versus the transform decomposition the direct computation is only feasible for a low number of needed output points. Thus the transform decomposition, or Sørensen FFT (SFFT), is selected as non-full-length FFT algorithm.

In conclusion the split-radix FFT presented in [Duhamel and Hollmann, 1983] and the transform decomposition presented in [Sørensen and Burrus, 1993] is selected for further investigation and implementation. Next a general introduction to the Fourier transform and the selected algorithms will be analyzed.

3.2 Discrete Fourier Transform

The Discrete Fourier Transform (DFT) is presented in [Oppenheim and Schafer, 1998, p. 561] and is shown below for completeness

$$\begin{aligned} X[k] &= \sum_{n=0}^{N-1} x[n] \cdot \exp\left(\frac{-2\pi j \cdot n \cdot k}{N}\right) \\ X[k] &= \sum_{n=0}^{N-1} x[n] \cdot T_N^{n \cdot k}, \quad T_a^b = \exp\left(\frac{-2\pi j \cdot a}{b}\right) \end{aligned} \tag{3.1}$$

Where T_a^b is known as the "Twiddle Factor" . The time domain input data $x[n]$ is of length $n = 0 \ldots N-1$ and is transformed via (3.1) to the frequency domain signal $X[k], k = 0 \ldots N-1$, where k represents frequency-indexes.

The following split-radix FFT computes the entire range of k whereas the Sørensen FFT only computes a subset of k called l. The computed subset l has the property $l \in k$ and L is the total number of indexes of k found.

3.3 Split-Radix FFT Algorithm

This chapter presents the Split-Radix FFT implementation used for testing against the Sørensen FFT algorithm. First, the analytical algorithm of the Split-Radix calculation of a DFT is presented.

3.3.1 SRFFT Derivation

The derivation is started by examining (3.1) and splitting the summation into even and odd parts

$$\begin{aligned} X[k] &= \sum_{n=0}^{N/2-1} x[2n] \cdot T_N^{2nk} + \sum_{n=0}^{N/2-1} x[2n+1] \cdot T_N^{(2n+1)k}, \\ k &= 0 \dots N-1 \end{aligned} \tag{3.2}$$

changing the indexes to reflect x_e and x_o for x even and odd, and noticing the double angular velocity and thus periodicity in $N/2$ of the even parts, the equation can be rewritten as

$$X[k] = \sum_{n=0}^{N/2-1} \left(x_e[n] + x_e[n+N/2]\right) T_{N/2}^{nk} + T_N^k \sum_{n=0}^{N/2-1} x_o[n] \cdot T_{N/2}^{nk} \tag{3.3}$$

which has saved $N/2$ multiplications in the even part.

The odd part sequence is pre-multiplied by T_N^k, which can be remedied by decomposing the odd parts into an even and odd sequence again. It thus becomes a general radix-4 decomposition with $k = 0 \dots N/4-1$, as is shown below.

$$X[k] = \sum_{n=0}^{N/4-1} x[n]T_N^{nk} + \sum_{n=N/4}^{N/2-1} x[n]T_N^{nk} + \sum_{n=N/2}^{3N/4-1} x[n]T_N^{nk} + \sum_{n=3N/4}^{N-1} x[n]T_N^{nk} \tag{3.4}$$

$$\Updownarrow$$

$$\begin{aligned} X[k] &= \sum_{n=0}^{N/4-1} x[n]T_N^{nk} + \underbrace{T_N^{Nk/4}}_{-j} \sum_{n=0}^{N/4-1} x[n+N/4]T_N^{nk} \dots \\ &\dots + \underbrace{T_N^{Nk/2}}_{-1} \sum_{n=0}^{N/4-1} x[n+N/2]T_N^{nk} + \underbrace{T_N^{3Nk/4}}_{j} \sum_{n=0}^{N/4-1} x[n+3N/4]T_N^{nk} \end{aligned} \tag{3.5}$$

To extract the even and odd parts of x_o of (3.3), the radix-4 decomposition is examined for parts $X[4k+1]$ and $X[4k+3]$ and noting that $T_N^{n(4k+1)} = T_N^n \cdot T_{N/4}^{nk}$

$$\begin{aligned} X[4k+1] &= \sum_{n=0}^{N/4-1} \big(x[n] - x[n+N/2] \dots \\ &\dots -jx[n+N/4] + jx[n+3N/4]\big) T_N^n T_{N/4}^{nk} \end{aligned} \tag{3.6}$$

$$\begin{aligned} X[4k+3] &= \sum_{n=0}^{N/4-1} \big(x[n] - x[n+N/2] \dots \\ &\dots +jx[n+N/4] - jx[n+3N/4]\big) T_N^{3n} T_{N/4}^{nk} \end{aligned} \tag{3.7}$$

With the even part from (3.3) is written as

$$X[2k] = \sum_{n=0}^{N/2-1} \left(x_e[n] + x_e[n+N/2]\right) T_{N/2}^{nk} \tag{3.8}$$

Where equations (3.6) to (3.8) are recognized as yet more DFT-transforms.

This decomposition can thus be repeated on each sequence until a 2-point DFT is performed. Let $m = 1 \ldots M$ signify the decomposition stage, where M is the maximum number of decompositions.

The two-point DFT is a trivial butterfly as shown below

$$X[0] = x[0]T_2^0 + x[1]T_2^0 = x[0] + x[1] \tag{3.9}$$

$$X[1] = x[0]T_2^0 + x[1]T_2^1 = x[0] - x[1] \tag{3.10}$$

Thus by combining (3.6) - (3.10) the split-radix FFT can be performed.

3.3.2 Datapath Derivation

The odd indexes of the SRFFT equations, given in the previous section, can be reorganized for better data access by exploiting common operations. Grouping terms by real and imaginary pre-multiplication-factors and removing the factors from the DFT decomposition, the equations becomes

$$\begin{aligned} x_m[4k+1] &= \sum_{n=0}^{N/4-1} \Big[\left(x_{m-1}[n] - x_{m-1}[n+N/2]\right) \ldots \\ &\ldots \quad -j\left(x_{m-1}[n+N/4] - x_{m-1}[n+3N/4]\right) \Big] T_N^{mn} \end{aligned} \tag{3.11}$$

$$\begin{aligned} x_m[4k+3] &= \sum_{n=0}^{N/4-1} \Big[\left(x_{m-1}[n] - x_{m-1}[n+N/2]\right) \ldots \\ &\ldots \quad +j\left(x_{m-1}[n+N/4] - x_{m-1}[n+3N/4]\right) \Big] T_N^{3mn} \end{aligned} \tag{3.12}$$

$$x_m[2k] = \sum_{n=0}^{N/2-1} \left(x_{m-1}[n] + x_{m-1}[n+N/2]\right) \tag{3.13}$$

where the additional m multiplication in the twiddle factors is from the recursive decomposition.

A datapath capable of performing this operation is shown in figure 3.1 on the next page.

3.3.3 Graphical Example

A graphical example of a 32 point SRFFT is shown in figure 3.2 on page 24. Notice the bit reversed output and the multiplication of the twiddle factors in the lower part of the L-butterfly. The decopmosition structure of the SRFFT is shown as blocks.

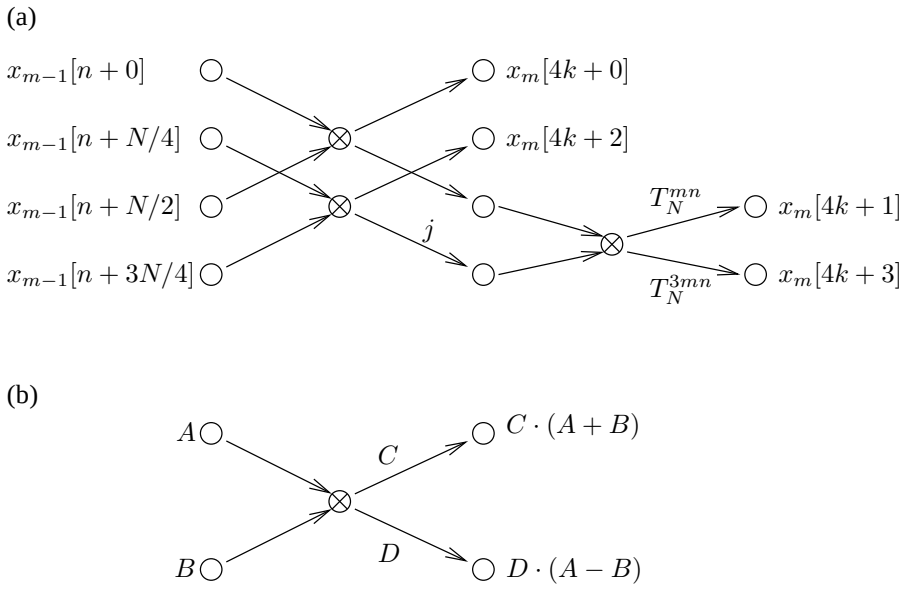

Figure 3.1: *a) Example of an L-shaped datapath performing one step in the SRFFT. b) Example of used butterfly which is performing addition, multpilication and subtraction.*

3.4 Sørensen FFT

The algorithm presented in [Sørensen and Burrus, 1993] is also known as the "Transform Decomposition", but will henceforth be referred to as the "Sørensen FFT" (SFFT) .

The improvements presented in the SFFT is that of splitting the DFT equation (3.1) of page 20 into a reusable part and a part which is distinct for each k to be computed. With this it is possible to compute the reusable part once and finish the calculation for each needed k.

3.4.1 SFFT Derivation

To derive a reusable part in (3.1), it is necessary to avoid the data dependency on k in the twiddle factors. This is achieved by changing the divisor of the twiddle factors, N in $T_N^{n \cdot k}$, to a smaller value P. This will introduce periodicity in the complex exponential in $T_P^{n \cdot k}, P < N, n = 0 \ldots N-1$ thus providing a reusable partition.

Equation (3.1) can be split into two sums of length P and Q.

$$Q = N/P \tag{3.14}$$

$$n = Q \cdot n_1 + n_2, \tag{3.15}$$

$$n_1 = 0 \ldots P-1, \quad n_2 = 0 \ldots Q-1 \tag{3.16}$$

$$X[k] = \sum_{n_2=0}^{Q-1} \sum_{n_1=0}^{P-1} x[n_1 \cdot Q + n_2] \cdot T_N^{(n1 \cdot Q + n_2) \cdot k} \tag{3.17}$$

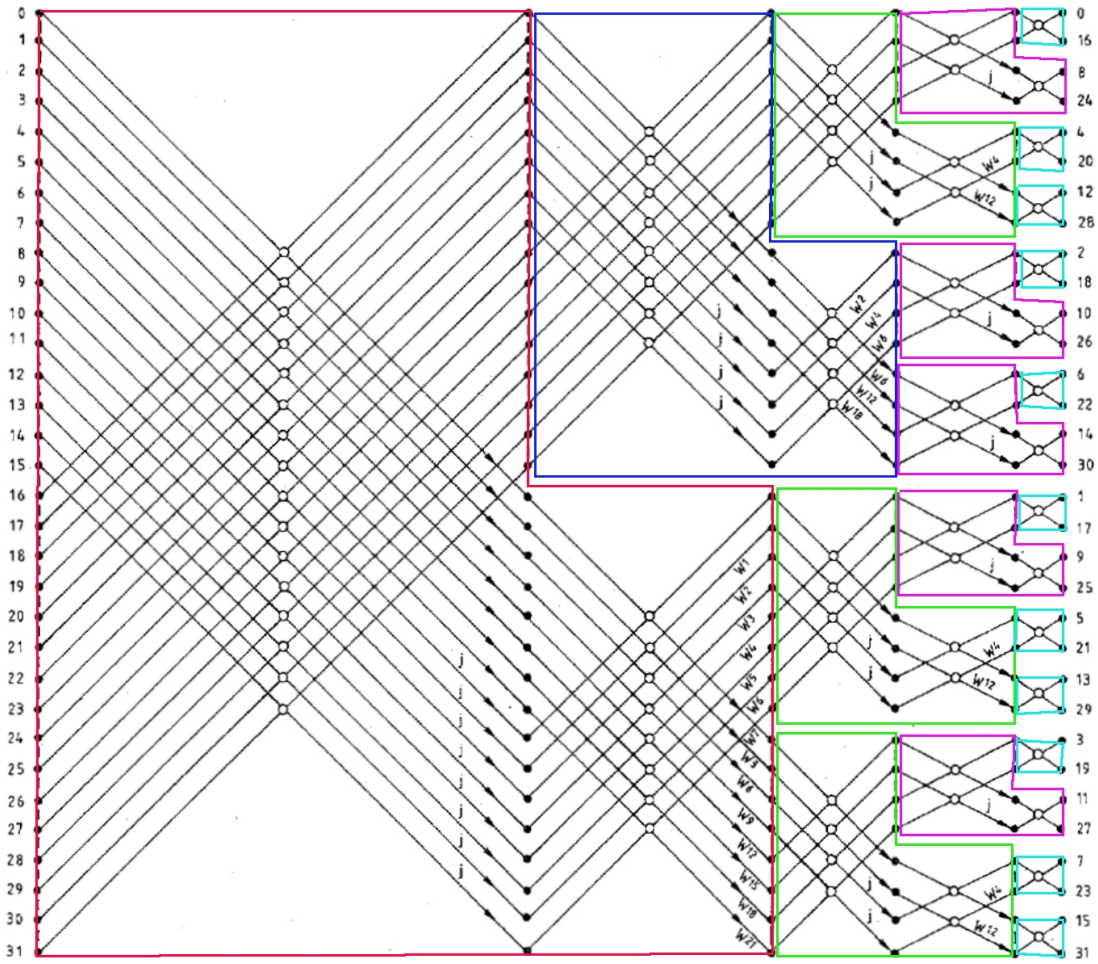

Figure 3.2: *Example of a 32 point SRFFT, modified from [Duhamel, 1986, p. 286]*

where P becomes the length of each sub-sequence and Q becomes the number of these sub-sequences. The exponential $T_N^{(n_1 \cdot Q + n_2) \cdot k}$ can be written as $T_N^{n1 \cdot Q \cdot k} \cdot T_N^{n_2 k}$ which allows for further splitting of the sequences, as n_1 and n_2 are the terms on which the summations depend.

$$X[k] \quad = \quad \sum_{n_2=0}^{Q-1} \left[\sum_{n_1=0}^{P-1} x[n_1 \cdot Q + n_2] \cdot T_N^{n_1 \cdot Q \cdot k} \right] \cdot T_N^{n_2 \cdot k} \tag{3.18}$$

It is seen that k is still present in both sequences. This is remedied by substituting $Q = N/P$, (3.14), into the innermost sequence twiddle factor of (3.18) which produces $T_N^{n_1 \cdot Q \cdot k} = T_P^{n_1 \cdot 1 \cdot k}$,

and usning the periodicity of k in P. This yields

$$X[k] = \sum_{n_2=0}^{Q-1} \left[\sum_{n_1=0}^{P-1} x[n_1 \cdot Q + n_2] \cdot T_P^{n_1 \cdot 1 \cdot k} \right] \cdot T_N^{n_2 \cdot k} \tag{3.19}$$

$$x_{n_2}[n_1] = x[n_1 \cdot Q + n_2]$$

$$X[k] = \sum_{n_2=0}^{Q-1} \underbrace{\left[\sum_{n_1=0}^{P-1} x_{n_2}[n_1] \cdot T^{n_1 \cdot \{k\}_p} \right]}_{X_{n_2}[r] = \sum_{n_1}^{P-1} x_{n_2}[n_1] T_p^{n_1 \cdot r}} \cdot T_N^{n_2 \cdot k} \tag{3.20}$$

$$r = 0 \ldots P-1, \quad r = \{k\}_p$$

where $\{k\}_p$ is k modulus p.

The innermost sequence is recognized as a length P DFT, as the values of k always can be mapped to a value of r via the modulus function. This is shown below (3.20) as function $X_{n_2}[r]$.

The derivation thus shows that (3.20) consists of Q numbers of DFTs where selected $r = \{k\}_P$ indices are subsequently multiplied by a twiddle factor and summed. Efficient computation of the DFTs can be performed by the use of any FFT algorithm which works for length P. The split-radix FFT of section 3.3 on page 20 is used as it is to be implemented for this project, and has the one of the lowest computational complexities for any power-of-two FFT-algorithm, see chapter 3.3 on page 20.

3.4.2 Graphical Example

An example of the SFFT transform for $N = 16$, $P = 8$ and $k = [3, 10]$ is given in figure 3.3 on the next page. As $Q = 2$ the selected samples in the "Transformed" column is selected by $n_1 \cdot Q + n_2$ which yields indexes 3 and $3 + 8$ for $n_2 = [0, 1]$ and $k = 3$. For $k = 10$ the indexes becomes 4 and 10.

The selected indices, one from each sub-DFT, are then multiplied by $T_N^{n_2 \cdot k}$ as in (3.20). Where n_2 signifies which sub-DFT the sample is originating from.

3.5 Complexity Analysis

Having examined the theoretical derivation of each algorithm, their respective computational are examined here to provide the basis for later comparison with the power performance.

The complexity analysis is also used in specifications of test-scenarios and in the case of the Sørensen FFT, to select optimal operating criteria.

3.5.1 DFT Complexity

The amount of calculations needed to compute L output samples with the DFT is found in (3.1) to be $L \cdot N$ complex multiplications and $L \cdot (N - 1)$ additions. The DFT is included here for completeness.

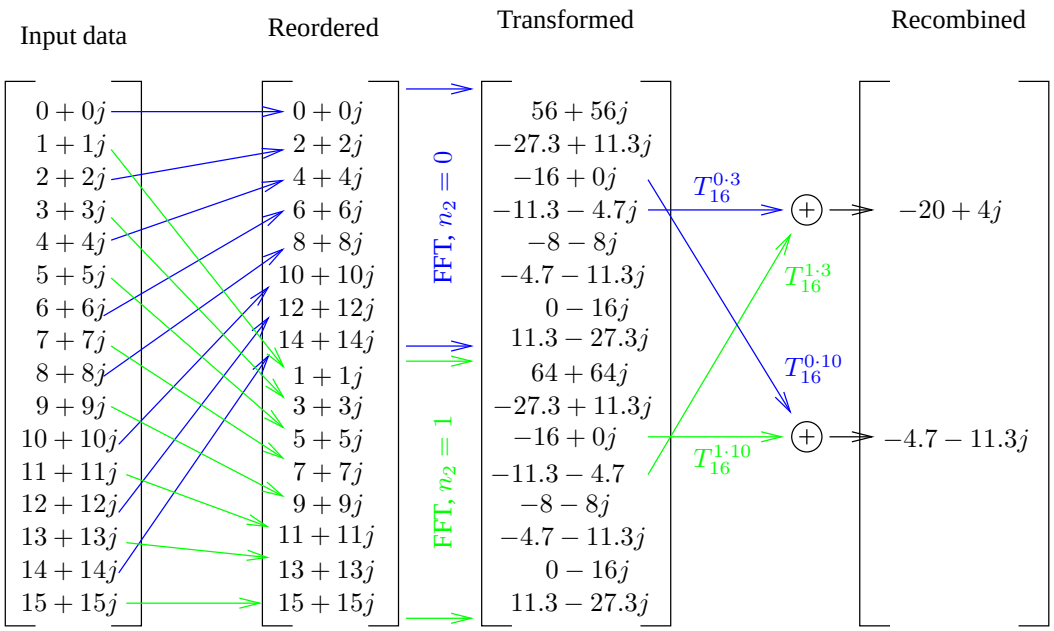

Figure 3.3: *SFFT example, where* $N = 16$*,* $P = 8 \Rightarrow Q = 2$ *and* $k = [3, 10]$*. See section 3.4.2 on the previous page for full explanation.*

3.5.2 Split-Radix FFT Complexity

The number of L-butterflies and two-point butterflies are calculated in [Skodras and Constantinides, 1992, p. 57 eq (10) and in section 8.2 p. 59]. For $N = 1024$ there will be 10 recursive stages, yielding 1593 L-butterflies and 341 two-point butterflies.

It can be seen from figure 3.1 on page 23 that each L-butterfly uses two complex multiplications, which yields 4 simple multiplications and 2 simple additions/subtractions per complex multiplication, and six complex additions/subtractions, which yields two simple additions/subtractions per complex addition/subtraction. The two-point-butterflies uses two complex additions/subtractions.

A MatLAB script which calculates the number of L-butterflies and two-point-butterflies for power-of-two values of N can be found on the accompanying CD [08gr1042, 2008, tools/matlab/SRFFToperations.m].

3.5.3 Sørensen FFT Complexity

The complexity of the Sørensen FFT depends on the chosen subdivision factor P, as this defines Q which is the number of sub-FFTs performed. For these sub-FFTs the split-radix FFT algorithm is used, making the Sørensen FFT dependent on the complexity of the split-radix FFT, with the

added computations in the recombination step. Calculation of the complexity yields

$$n_{complexadd/sub} = Q \cdot (n_{length(P)SRFFT}) + (Q-1) \cdot L \quad (3.21)$$

$$n_{complexmult} = Q \cdot (n_{length(P)SRFFT}) + Q \cdot L \quad (3.22)$$

$$(3.23)$$

3.5.4 Comparison

A calculation of the complexities for the DFT, SRFFT and SFFT algorithms are shown in figure 3.4, for simple additions and multiplications. The first axis shows the number of computed points L.

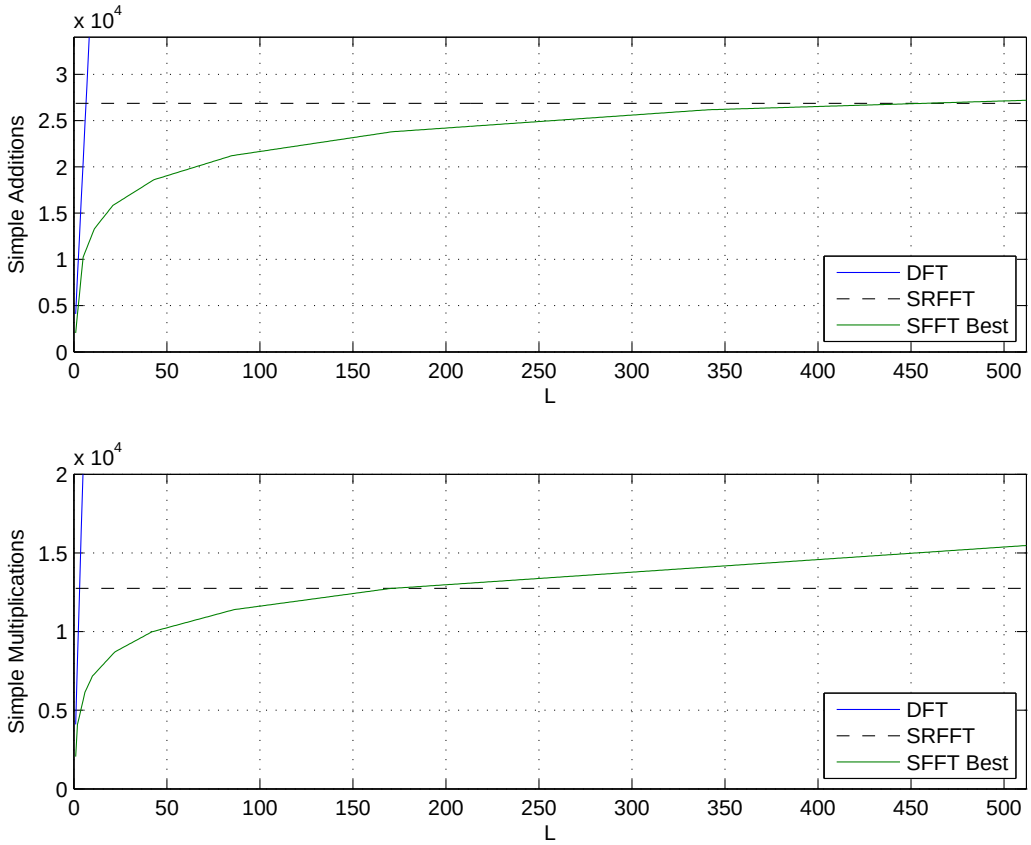

Figure 3.4: *Comparison of complexities. FFT length is 1024 and the first axis shows the number of computed points. The optimal setting for the SFFT is selected for each number of computed points.*

It is seen that the SRFFT is constant for all L, as it computes all points. The DFT quickly becomes infeasible for $L > 8$ while the SFFT performs better than the SRFFT up to $L < 400$. The SFFT has different performance characteristics depending on the selected subdivision factor

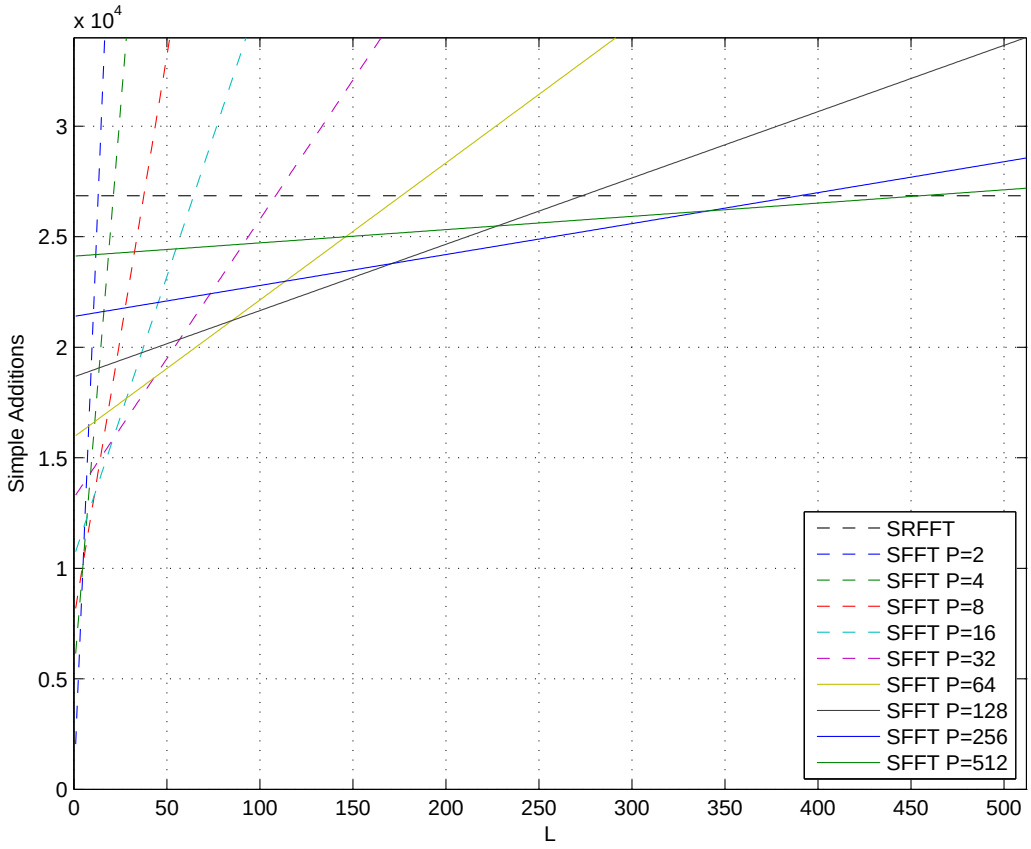

Figure 3.5: *Comparison of complexities, simple additions. FFT length is 1024 and the second axis shows the number of computed points.*

P and the resulting Q. Figure 3.5 and figure 3.6 on the next page shows the elaborated partition for additions and multiplications where the performance for the SFFT for different values of P is presented.

For later implementation of the SFFT a value of $P = 32$ and $Q = 32$ is selected. This is done to evaluate if complexity analysis is a fitting benchmark compared to power usage. With this selection the point where the SRFFT becomes more efficient than the SFFT is placed at $L = 50 \ldots 100$ according to figures 3.5 and 3.5.

A different value of P could be selected if power efficiency is the only goal, but it is also of interest to examine if the SFFT becomes worser than the SRFFT. With the selection of $P = 32$ the initial problem can be evaluated while the SFFT should still outperform the SRFFT for low values of L.

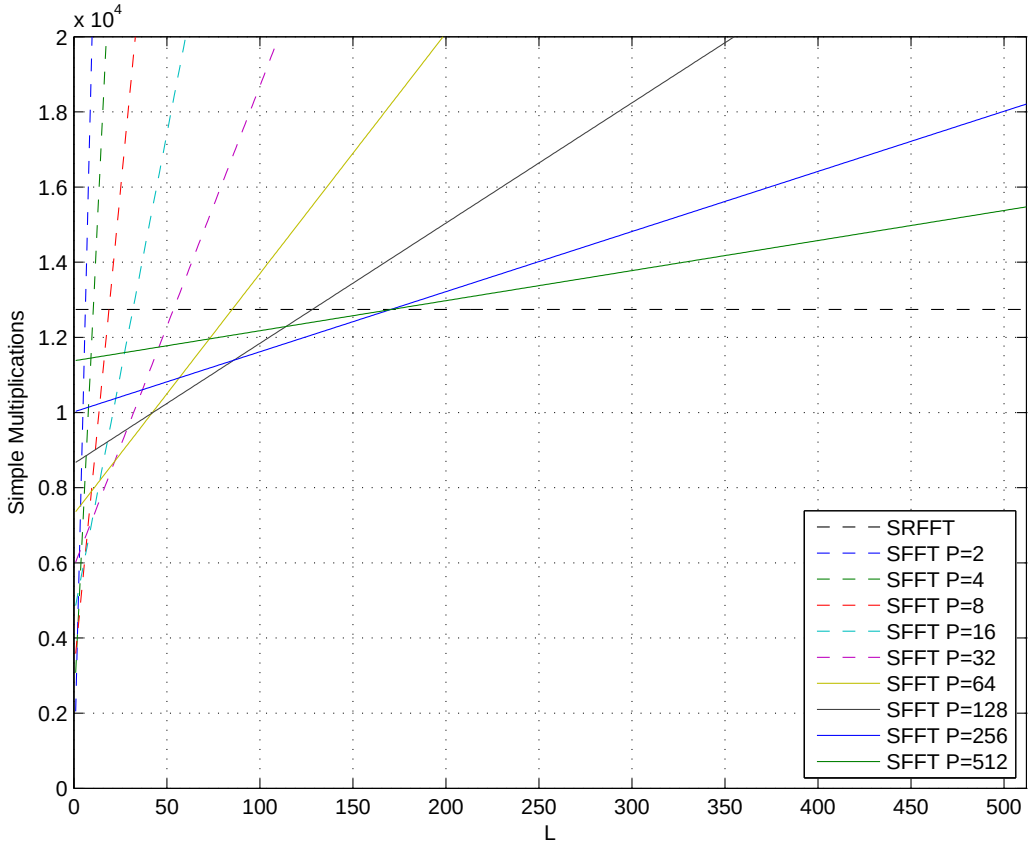

Figure 3.6: *Comparison of complexities, simple multiplications. FFT length is 1024 and the second axis shows the number of computed points.*

Chapter 4

Architecture Analysis

The purpose of the architecture analysis is to establish an overview of the available hardware and development tools used for mapping the FFT algorithms onto the FPGA architecture. In the first section the Cyclone III FPGA device is analyzed followed by a discussion of the Starter Kit development board used in the project. Finally, the Quartus software tools used in the algorithm mapping is presented and the resulting design flow using these tools is examined.

4.1 The Cyclone III FPGA

The general structure of a Cyclone III device is shown in figure 4.1. The FPGA core consists of programmable logical elements with memory and dedicated multiplier elements distributed across the device. Along the sides of the device are I/O interfaces to the device pins for external connections and in each corner is a Phase Locked Loop (PLL) for to be used for clock generation and management.

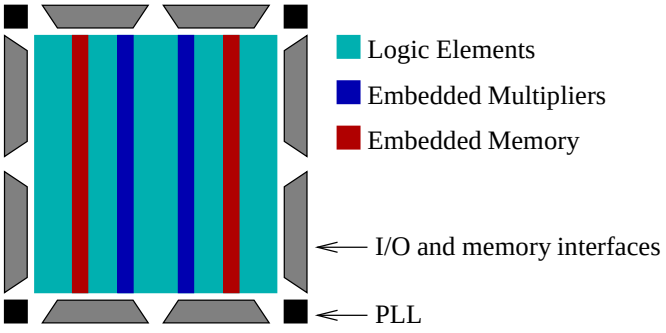

Figure 4.1: *Cyclone III floorplan showing the structural components of the FPGA device [Altera, 2007c, page 1-4].*

The integral part of the FPGA architecture is the programmable Logic Element (LE) which is discussed in further detail in the remainder of this section. The components of a single LE is shown in figure 4.2.

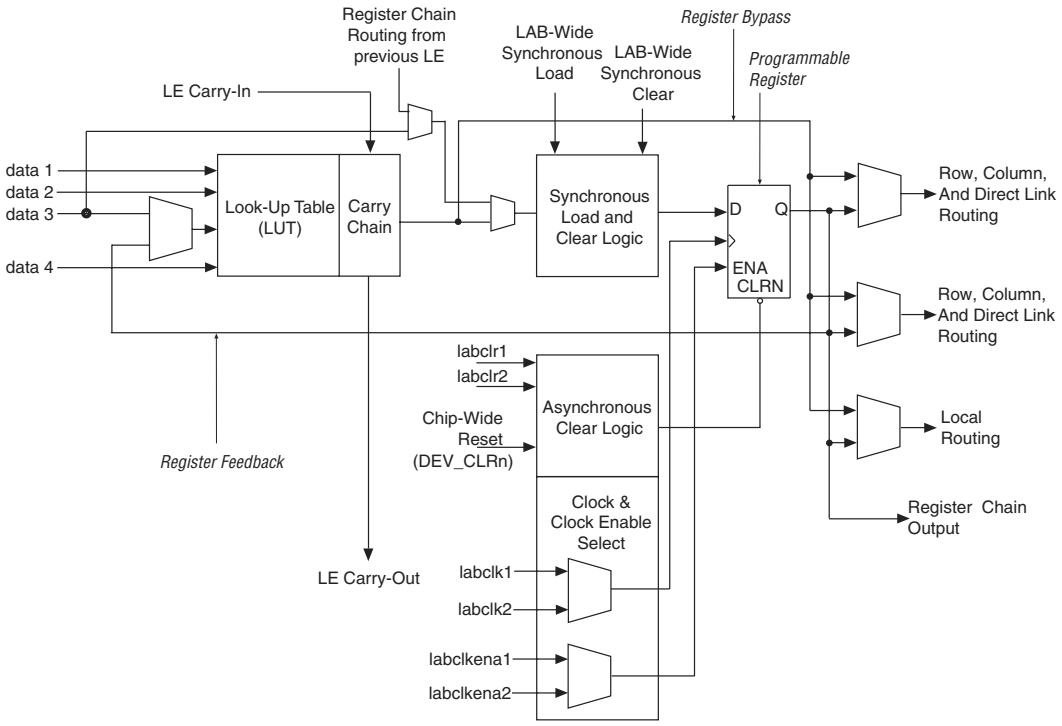

Figure 4.2: *Structure of a Cyclone III logic element, from [Altera, 2007c, page 2-2].*

Each LE has four data inputs and a carry in bit, used to access a look-up table. The output of this loop-up is a carry out signal and a signal used to set or reset the LE programmable register, which may be configured as either a D, T, JK or SR flip-flop. Based on the input from the LUT, the configuration of the register, and the signals generated from the clock selection and enable circuit, an output is produced. Both outputs from the LUT and Register may drive the LE outputs accessing the FPGA signal routing networks.

The LEs are ordered in columns of 16 in a Logic Array Block (LAB) . Within one LAB Carry-Out and Register Chain Outputs may be passed on from one LE to the next in the LAB without spending the FPGA general routing resources.

Combining the LEs enables construction of digital logic circuits, which again can be used to implement systems of higher abstraction. The LEs in the LABs thus provides the building blocks for construction adders, multipliers etc. See [Wakerly, 2001, chapters 5 to 10] for further information on this topic.

4.2 The Cyclone III Starter Kit

The Cyclone III Starter Kit is the platform used for evaluating the implementation of FFT algorithms on a FPGA architecture. An overview of the development board is shown in figure 4.3. In this section the board features of special interest to this project are examined.

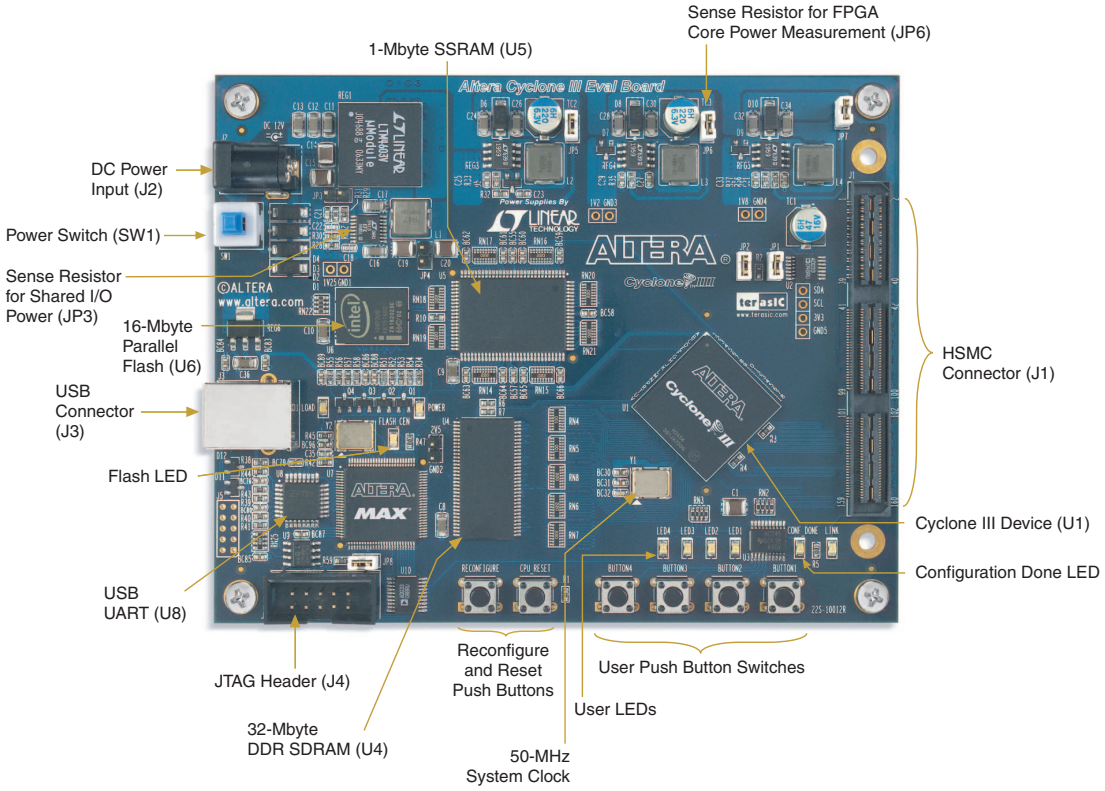

Figure 4.3: *Overview of the Cyclone III Starter Kit board, from [Altera, 2007b, page 2-1].*

The Cyclone III device itself (EP3C25F324C8) is from the middle of the range specification wise. A summary of the specifications is found in table 4.1.

Feature	Specification	Device family range	Approx. Requirement
Logic Elements	24,624	5,136 - 119,088	3,033
Memory [kBits]	504	414 - 3,888	102.4
Multipliers (18x18)	66	23 - 288	3

Table 4.1: *Cyclone III device specifications, specification range for device family and approximate requirements for full 1024 point FFT.*

The approximated system requirements in table 4.1 are based on compilation tests conducted in [Altera, 2007a, table 4], where a full 1024 point FFT core for an OFDM downlink has been

implemented on a Cyclone II device, which is the predecessor of the device used in chosen system. As seen from the table, even the smallest Cyclone III family device should be able to accommodate the requirements for a 1024 point FFT. Thus, there should be no HW resource shortages of the device available in the starter kit even though the transform recombination part of the SFFT probably will increase HW resource requirements.

Besides the FPGA device, the kit features an USB link for programming the FPGA using the USB-blaster setup provided with the Altera Quartus II software kit. This setup also allows for including virtual probes in the design and record signal activities in design nodes for functional verification. This is performed through the SignalTap interface in the Quartus II software.

Finally, 4 pushbuttons and 4 LED circuits are connected to user configurable I/O pins on the FPGA, allowing for user input and status output to and from the implemented system. Additional features such as DDR/flash memory and JTAG pinout are not utilized in the project.

4.3 Quartus II software tools and design flow

In order to map the FFT algorithms onto the FPGA, the Quartus II design software is used. The overall flow from design entry to device program along with the system verification flow is shown in figure 4.4.

4.3.1 Compilation Flow

The left part of figure 4.4 shows the compilation flow from design entry to device programming files.

Design Entry The design software allows for several possible methods of design entry. These types of design entries may be gathered in three principal groups:

- **Hardware Description Language (HDL) entries**
  Systems modelled using Verilog or VHDL may be used directly as design entry.

- **Altera design files**
  Predeveloped hardware functions in a Altera specific HDL may be adjusted to fit systems requirements for data types and function latency. The interconnection of the system design entities is specified in block design files as diagrams.

- **Third party netlists**
  EDIF and VQM netlists from third party development tools. This possibility for design entry is not utilized in the project.

Analysis and Synthesis The first part of system compilation is Analysis and Synthesis. This process optimizes and compiles the various design entries into one system netlist describing the setup of hardware resources and their interconnection.

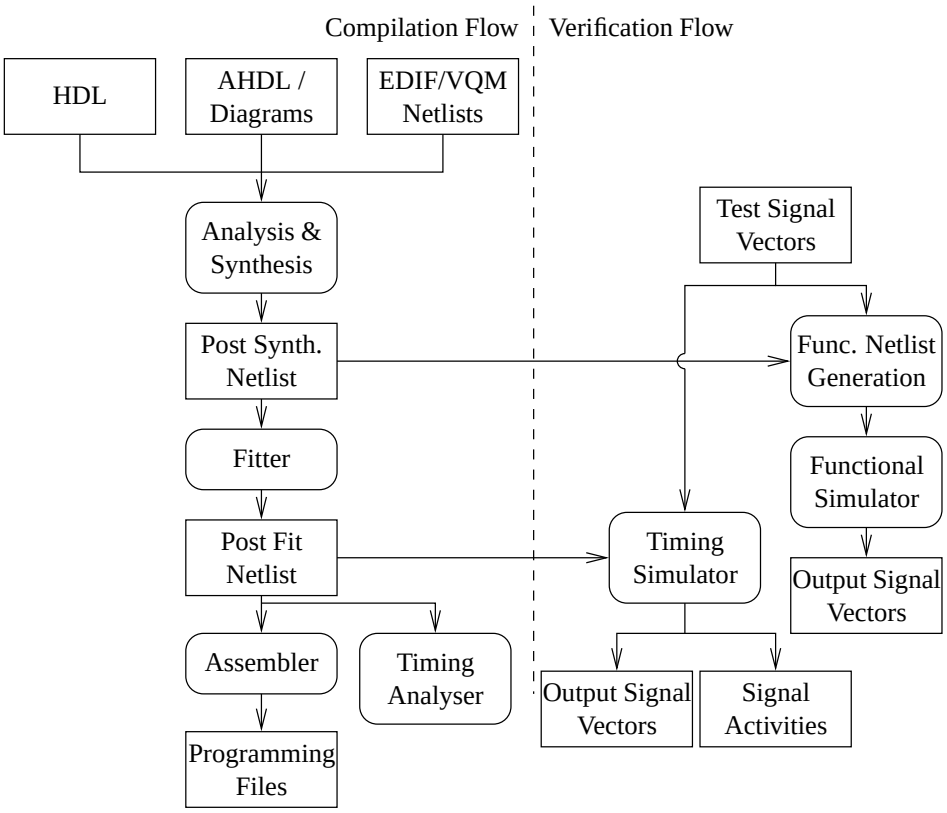

Figure 4.4: *Quatus II compilation flow from design entry to device program [Altera, 2007d, page 2-4] and verification flow, including functional and timing simulations.*

Fitter The fitter conducts the actual allocation of FPGA hardware resources, based on the generated netlist from the analysis and synthesis process and a model of the device. This results in a netlist describing the contents of figure 4.1 on page 31, specifying the setup of logic elements, multipliers, memory block, I/O pins and interconnection buses.

Assembler The final stage of the compilation generates the programming file from the fitter netlist. This is the final files to be loaded onto the device.

Timing analysis The compilation process includes a final timing analysis to verify, that the propagation delay of signals between the allocated device entities will not effect the functionality of the system.

4.3.2 Verification Flow

The verification flow shown on the right in figure 4.4 consists of two types of simulations.

Functional Simulation The functional simulation allows for principal verification of the designed system functionality. Based on a set of input signal vectors and the post synthesis netlist, vectors containing output signals are generated for comparison with expected results.

Timing Simulation The timing simulation is based on the same input signal vectors as the functional simulation, but makes use of the post fit netlist to simulate the timing of signals in the design as well as functional results. The results of this simulation is thus more elaborate and precise. Besides the output signal vectors for verification of the design a signal activity file is generated. For each internal signal of the design a probability of a signal change in each clock cycle as well as a static probability of the signal value is recorded. These signal activities a an integral part of the input for the power simulations, discussed in the next chapter.

With this overview of the available hardware and tools for mapping the FFT algorithms, the next step is to investigate the modelling and measurement of power consumption of the FPGA architecture. This analysis is conducted in the next chapter.

Chapter 5

Power Estimation and Measurement

In order to evaluate the DFT algorithms based on a power consumption performance measure, the available tools for simulation and measurement of system power consumption when using the Altera Quartus tools in connection with the Cyclone III starter kit are examined. The following sections present the functionality of the Powerplay simulator, the method used for measuring power and how power will be used to evaluate the algorithms.

5.1 Power Simulations

The following introduction to the Powerplay tool is based on an white paper on power modelling [Altera, 2007e] and the tool user manual [Altera, 2007d, chapter 10]. These materials give an introduction to the principles of the power modelling used in the Powerplay tool to simulate the power consumption of a FPGA design. The choice of the Powerplay tool for simulating system power consumption is based on the assumption, that the manufacturer of the FPGA device (and power estimation tool) has access to the most reliable device modelling data, in order to supply a reliable power estimate.

The flow for power simulation using Powerplay is shown in figure 5.1. As seen in the figure, the power analysis is based on the structure of the system implementation, described in the fitter output, and signal activities, derived from timing simulations of the system. The structure of these analysis input are described in more detail in chapter 4 on page 31. The data, that are key to the power analysis from each input source, are:

- **Fitter results**
  The fitter results is the map of the design onto the chip, describing which physical device resources are active in the design and how they are connected.

- **Simulation results**
  Based on the timing simulation toggle rates for each signal in the implementation are calculated and used in the power analysis. Toggle rates are measured as signal transitions per second and thus describe the signal activity.

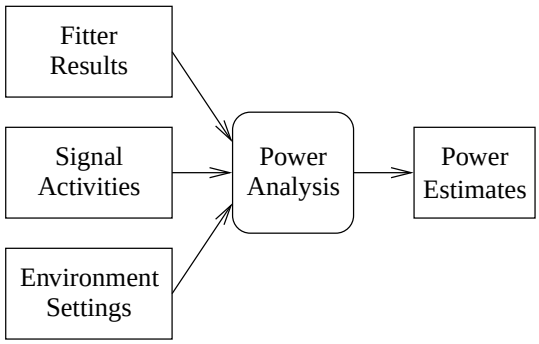

Figure 5.1: *Flow of Power Simulations using Altera Powerplay. The power analysis is based on the fitter and assembler output, describing the physical structure of the system, and signal activities derived from functional simulation results.*

- **Environmental settings**
  The environmental settings contain information on device operating conditions, including supply voltages, ambient temperature, and cooling aid characteristics, used by the analyzer to estimate the device temperature and how well thermal energy is routed away from the device.

The simulation output report contains the power consumption simulated at different abstraction levels, ranging from total device power consumption down to power consumed by each design entity defined. At each abstraction level both the static and dynamic power consumption is recorded for each entity.

The current drawn from the 1.2 V power supply, which is consumed by the device internal logic (V_{CCINT}) and the PLL circuits (V_{CCD}) on the chip, is what is possible to measure on the development board, and thus of primary interest for comparison of the simulations with the performed measurements in section 9.1 for the SRFFT and section 9.2 for the SFFT.

The detailed evaluation of the system design is discussed in the design space exploration section 10. In this section, power simulation reports, where power consumption is reported for each design block in the project hierarchy is accounted for, is used to determine where to turn focus when further optimizing the design.

5.1.1 Power models

The Powerplay tool divide the power consumption into two fundamental types of power consumption; dynamic and static power consumption. In this section the structure and contents of these models are described and the key points to managing system power consumption that may be derived from these models are discussed.

The dynamic power consumption includes power consumed due to circuit switching in the system. The general dynamic power estimation model used in the simulator is [Altera, 2007e,

page 2]:

$$P_{dyn} = \left[\frac{1}{2} \cdot C \cdot V^2 + Q_{sc} \cdot V\right] \cdot f \cdot a \tag{5.1}$$

where dynamic power, P_{dyn} $[W]$, consumed by a device entity, is a function of load capacitance, C, supply voltage, V, short circuit current when switching, Q_{sc}, system frequency, f, and signal activity, a, measured as a probability of a signal change in a cycle.

The output load capacitance, C shown in figure 5.2(a), of a design entity is determined from the wire network of the entity output, where properties such as wire length, thickness, and distance to neighboring wires are included. Each time a signal changes level, this parasitic capacitance is charged or discharged. In Powerplay, the dynamic power consumed due to the capacitance of the dedicated routing network, which interconnect device entities, is reported separately as routing power, whereas the parasitic capacitance in signal connections within a logic element, multiplier of memory block is included in the design entity dynamic power consumption.

The short circuit charge, Q_{sc}, occurs when signals are switched from low to high or vice versa. The principle is shown in figure 5.2(b), where a switch consisting of two transistor, will experience a short circuit of the supply, when both transistors are open for a short time during switching. This is due to switch not being instantaneous and the transistors cannot be assumed perfectly matched.

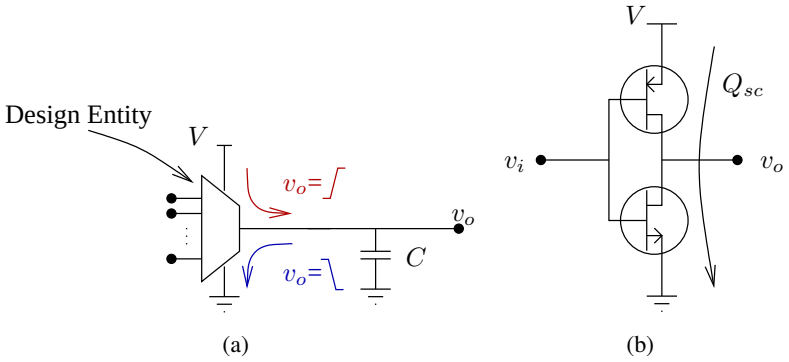

Figure 5.2: *Sources of dynamic power consumption.*
(a) Each time an entity switches output level, a parasitic capacitance, C, need to be charged (red) or discharged (blue) to change the level of the signal wire network.
(b) When the input signal, v_i, is switched, there is a charge, Q_{sc}, from V to ground during the time period in which the transistors are switching to change the level of v_o.

Where C and Q_{sq} are device specific and therefore usually predetermined variables. V is the supply voltage, f is the system frequency and a is the signal activity. V, f and partly a are set by the designer. As seen in equation (5.1) both V and f should be kept as low as possible to minimize the dynamic power consumption.

The signal activities, a, are partly determined by the system input signals and the algorithm structure. Still, managing the system by deactivating clocks subsystems when they are

not needed, in order to make certain that the subsystem logic is not toggled, will decrease signal activities and thus system dynamic power consumption.

The voltage supply, V, occurs at a power of two. Therefore, managing the supply voltage can have significant impact on system power consumption. However, lowering the supply voltage will increase the propagation delay of signals, due to capacitances, and thus lower the frequency at which the functionality of the design can be maintained.

System clock frequency, f will be dictated by requirements to completion time. If the algorithm to be mapped onto the architecture features inherent concurrency, the data path may be expanded to perform more calculations at once. This way the algorithm completion time is shortened and the system clock frequency may be lowered. However, the expansion of the data path will increase the device area utilization and add signal nodes to the design and thus add instances of equation (5.1) to the sum that constitutes the total dynamic power consumption.

The FPGA consumes energy regardless of signal activity. This static power consumption is a result of leakage currents in the device, and is mainly dependent on device area utilization and temperature. There are different approaches to modelling the static power consumption. Altera [2007e] outlines a model in which the static power consumption is an exponential function dependent on device temperature:

$$P_{static} = A \cdot e^{B \cdot T_j} + C \tag{5.2}$$

where T_j is the junction temperature or the temperature of the actual electronic device, and A, B and C are device specific constants. Another approach to expressing the static power consumption is to divide P_{static} into power dissipated from leakage and power dissipated due to a constant input value to a logic unit [Bellaouar and Elmasry, 1995, page 130-132]:

$$P_{static} = P_{leak} + P_{st} \tag{5.3}$$

P_{leak} occurs due to parasitic diodes in the gate junctions resulting in a current I_d from supply to ground. Thus P_{leak} may be expressed as the sum of leakage power dissipated in each junction:

$$P_{leak} = \sum_i I_{d_i} \cdot V \tag{5.4}$$

The second component of the static power consumption is a result of the transistors of the gate logic being held in a sub-threshold state where v_i in figure 5.2(a) is less than the threshold voltage V_T for either transistor. Thus the mean drain source current $I_{DS_{mean}}$ for the two transistors of the switch results in the following static power dissipation:

$$P_{st} = \sum_i I_{DS_{mean}i} \cdot V \tag{5.5}$$

The magnitude of these leakage and sub-threshold currents are highly temperature dependent and give rise to the exponential relationship between junction temperature in equation (5.2).

The discussion of the models used to estimate the system dynamic and static power consumption above may be summarized into the following generic guidelines for keeping system power consumption down:

1. Switch off subsystems when inactive
2. Minimize area utilization
3. Reduce clock frequency
4. Reduce supply voltage
5. Keep device temperature low

Point 1 of switching off inactive subsystems is directly applicable in the system design.

It is clear that points 2-3 may be conflicting tasks, since, as mentioned above, reduction of clock frequency may be obtained by exploiting algorithm concurrency and utilizing more device area. The optimal solution to this problem would require several iterations over possible solutions to find the optimal tradeoff.

To find the optimal voltage supply (point 4), a further investigation into the supply voltage versus maximum clock frequency relationship would need to be conducted for the specific device or device family. With the hardware available it is, however, not possible to change the supply voltages, thus for the specific system obtained in this project, the voltage is not a variable that may be changed to optimize power consumption.

Point 5 of reducing device temperature may be accomplished by applying passive (e.g. a heat sink) or active (e.g. a fan) cooling to the device to lead thermal energy away from the device. No further investigation into this subject will be conducted in order to keep focus on the project purpose of investigating power consumption in connection with mapping algorithms onto a FPGA architecture.

5.2 Power Measurements

A set of measurements are conducted to evaluate the Powerplay simulation results in comparison with actual system performance. The Cyclone III starter kit board allows for measuring the currents drawn from each of the voltage supplies, connected to the FPGA, by measuring voltage across a sense resistor, R_s, placed in series with each supply. The voltage supply of interest is the 1.2 V supply, feeding the internal logic, V_{CCINT}. This supply also feeds the digital part of the FPGA internal PLLs, V_{CCD}, so the simulated power consumed by this part of the chip must be included in the algorithm power consumption estimate to enable comparison between simulated and measured results.

Figure 5.3 shows the setup used for measuring the voltage drop across the sense resistor in the 1.2 V voltage supply circuit. The measurement setup consists of a DC voltmeter, connected to jumper $JP6$, and an oscilloscope probing the voltage driving the $LED1$ diode. This diode is toggled by the system to enable measurement of calculation completion time.

A DC-multimeter [Agilent, 2007] is used to measure the mean voltage drop across R_s. In order to get precise power estimates this approach requires the system to be able to be put in a constant idle state or constant active state, in order for no state changes to appear during measurements. To be able to acquire timing information from the calculation progress the external diode, $LED1$, is turned on, only when the system is in idle state. When the algorithm finishes

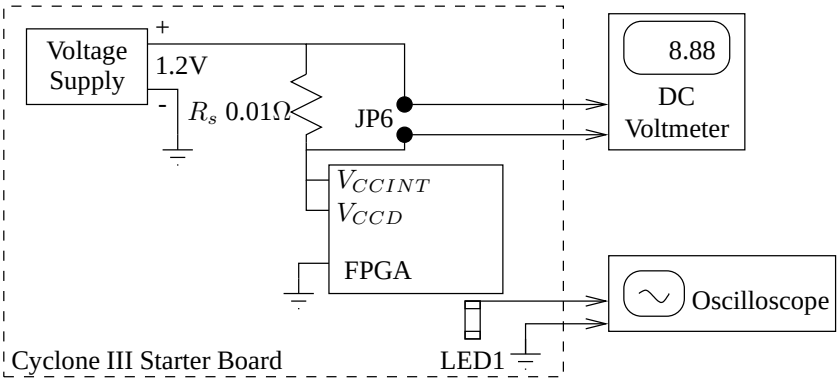

Figure 5.3: *Setup for measuring power consumed by the FPGA core. A voltmeter measures the voltage drop across a 0.01Ω sense resistor, R_s, in the voltage supply circuitry.*

the controller state machine passes through the idle state before calculating the next FFT. This results in a short pulse at $LED1$ and enables measuring the completion time for the algorithm under test.

Preliminary tests measuring the voltage across R_s with a differential probe and a oscilloscope where conducted to enable tracking of the power consumption changes through the calculation period. However, the results experienced relatively high noise, which hindered identifying changes in voltage across the sense resistor with changing system states. To reduce the noise level averaging the measurements is needed, which is the reason for using the voltmeter instead. A set of measurement records which exemplify this issue is found on the accompanying CD [08gr1042, 2008, testresults/differential/diff3.csv] The signal standard deviation of the example measurement record is 1.3 mV, and it will be clear in chapter 9 the changes in voltage across R_s is to be measured the a few hundred μV. Therefore an averaging of the measurements is needed and since the DC-voltmeter offers this feature directly along with a satisfying precision of $\pm$ 0.03 % of the reading $\pm$ 30μV [Agilent, 2007].

The test setup described above is used for all measurements conducted in chapter 9. A list of equipment used for the measurements is found on the CD [08gr1042, 2008, testresults/equipment.txt].

5.3 Power Performance Measure

In order to compare the results obtained from each algorithm test, a performance measure based on power need to be defined. Power, generally, is a measure of energy consumed per time unit (unit is $\left[W = \frac{J}{s}\right]$). The total energy, E_T, consumed by a system over a time interval, T, is thus:

$$E_T = \int_0^T P(t)dt \tag{5.6}$$

where $P(t)$ is the instantaneous power at time t, and the mean system power over the interval is

$$P_{mean} = \frac{1}{T} E_T \tag{5.7}$$

As mentioned in section 5.2, the Cyclone III board allows for measuring the voltage, V_{R_s}, across a sense resistor, R_s, thus determining the current drawn from the 1.2 V supply, $I(t)$. The mean power thus becomes:

$$\begin{aligned} P_{mean} &= \frac{1}{T} \int_0^T V_{supply} \cdot I(t) dt \\ &= \frac{1}{T} \int_0^T V_{supply} \cdot \frac{V_{R_s}(t)}{R_s} dt \end{aligned} \tag{5.8}$$

Since measuring $V_{R_s}(t)$ as a time series is not feasible, as described in section 5.2, the integral is solved and split into two intervals, one of length t_{active}, which is associated with the mean voltage measured over R_s when the system is kept active, and one of length $(T - t_{active})$, which is associated with the mean voltage measured when the system is idle:

$$P_{mean} = \frac{1}{T_s} \left(t_{active} \cdot V_{supply} \cdot \frac{V_{R_s\_active}}{R_s} + (T_s - t_a ctive) \cdot V_{supply} \cdot \frac{V_{R_s\_idle}}{R_s} \right) \tag{5.9}$$

where T is replaced with T_s, the OFDM symbol duration described in the system model, section 2.1, and set to 2 ms in the system specification in section 2.3. Equation (5.9) will be used when evaluating and comparing the implemented DFT algorithms in chapter 9 to determine which algorithm uses the least power when implemented.

Part II

Algorithm Mapping

This part concerns the implementation of the selected algorithms on the Cyclone III platform.

Initially an introductory chapter describes the abstraction scheme and the general properties of the system. Afterwards the split-radix and Sørensen FFTs are mapped onto the FPGA. Tests and further design-space exploration are performed in the following part.

Contents

Chapter 6

General Mapping

The mapping of the selected FFT algorithms to the FPGA platform is introduced in this chapter. The description is based on the abstraction-model introduced in figure 1.6 on page 7. Subsequently the different general decisions regarding the implementation is taken. These include topics as general control strategy, number representation and management and definition of the interfaces to the external components.

The following chapters describe the implementation of the specific FFT algorithms.

6.1 Environment Description

In the OFDM demodulation system shown in figure 2.1 on page 12 the data for the FFT are supplied by the entity removing the cyclic prefix. The Fourier transformed data is then moved to the subcarrier demapping. To provide easy access and emulate the FFT as part of a larger system, this data transfer is defined to happen in RAM blocks, as shown in figure 6.1 on the next page. These RAM blocks contains input and output data for the SRFFT. RAM # 2 and RAM # 3 are only present in the case of the SFFT and the indices of l which are to be computed and the results of the SFFT calculations.

6.1.1 RAM # 1

The interface between the system and the SRFFT is seen in figure 6.1 as RAM # 1, which is designed in this section. This RAM must hold 1024 real and imaginary values produced from the block which removes the cyclic prefix. After FFT processing the transformed data is placed in RAM # 1 to be accessed by the subcarrier demapping block. For the SFFT the RAM is also needed to contain initial data and the temporarily transformed data.

Requirements

Due to requirements from the L-butterfly in the SRFFT, explained in the coming section 7.2.1 on page 60, the RAM must be able to read and write 4 real and 4 complex values pr. clock cycle.

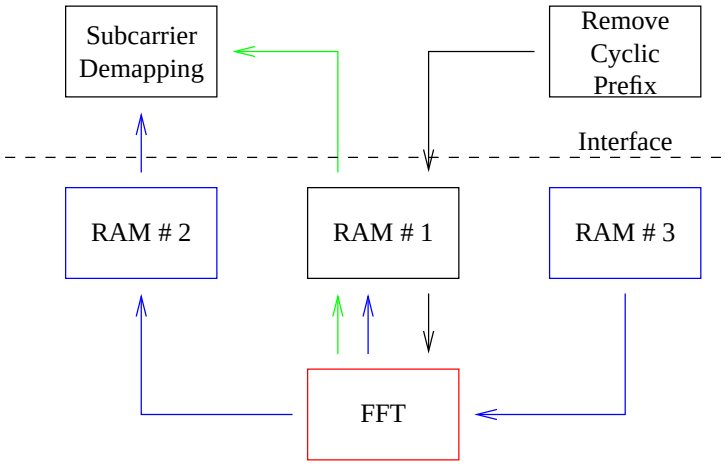

Figure 6.1: *Interface definition for the FFT. RAM # 1 receives the time-data from the received signal. In case of the SRFFT the FFT is performed in place and the green connections exists, whereas the SFFT places results in RAM # 2 and the blue connections exists. The blue connections are needed as the SFFT recombination cannot be done in place. RAM # 3 contains the indices of l which must computed. The FFT block is colored red to show placement in figure 2.1 on page 12.*

This stems from the nature of the L-butterfly which takes 4 real and imaginary input values pr. cycle and produces 4 real and imaginary output.

As the available RAM in the design tool is only able to produce 2 values per cycle (two-port RAM), another solution must be produced.

Design Choices

Initially it is decided to keep real and imaginary values in separate RAM blocks. The alternatives are to store the values interlaced or imaginary values sequentially after real values. These possibilities are disregarded as this would introduce additional multiplexers on the RAM output and encumber the addressing, compared to direct addressing of real and imaginary values in different blocks.

Defining the RAM as one or two blocks in the design makes no difference in the final implementation, as the memory of the FPGA are subdivided into blocks of 512×18 bit during compilation [Altera, 2007c, p. 4-1, table 4-1].

The problem of producing 4 values pr. clock cycle with two-port RAM can be solved by either subdividing the address space or make the RAM block run at a higher clock rate relative to the base.

Subdivision of the address space is not feasible for the current problem as the SRFFT accesses memory in an irregular fashion due to the L-butterfly. Instead it is decided to use a faster clock domain for the RAM blocks. This is possible as the datapath does not need to run at the same

clock as the RAM. As long as the data is ready and in synchronization with the datapath clock the solution is feasible.

As the real or imaginary RAM block must read and write 4 values pr. cycle while using two-port RAM, the RAM clock cycle is set to be four times faster than the datapath clock. This allows for two cycles of reading and two cycles of writing thus fulfilling the requirements.

Implementation

The structure of the RAM# 1 is shown in figure 6.2. The actual RAM block is dual port, which means that 2 addresses can be accessed in one clock cycle for either read or write operations.

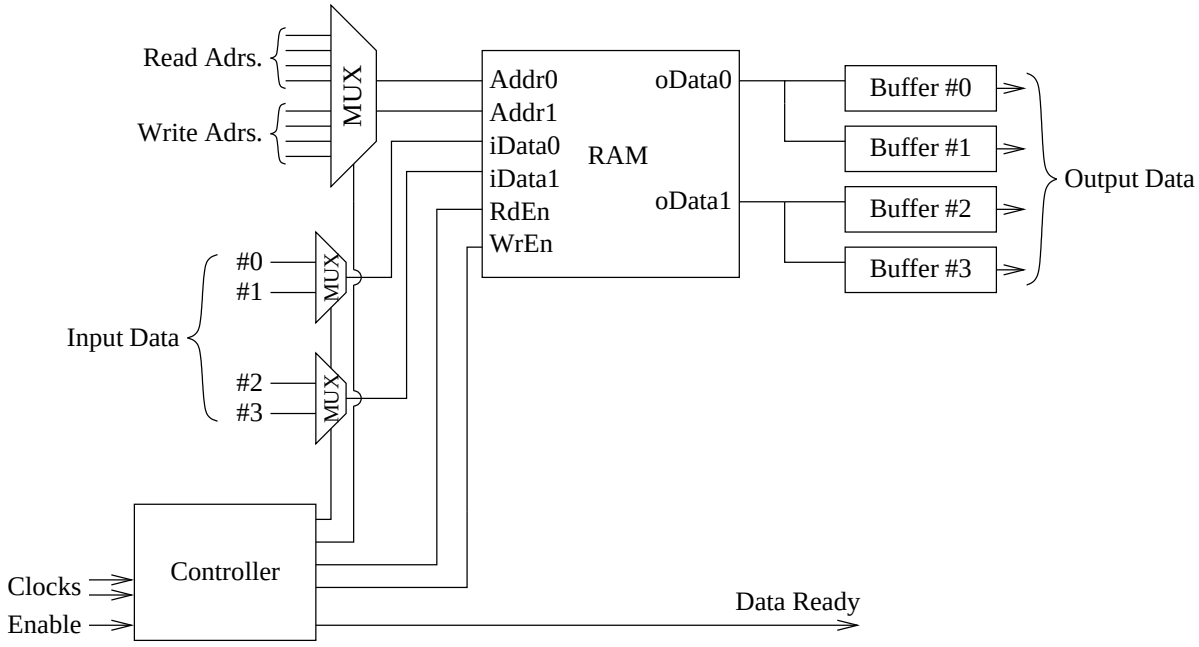

Figure 6.2: *Implementation structure of RAM#1. For overview, RAM block and data in- and output for imaginary data has been left out.*

As mentioned in the previous section one operation on the RAM include read and write of 4 complex data samples which is done in 4 RAM clock cycles, 0-3, in the following way:

- **Cycle 0:** Read address #0 and #2.
  The controller generates a read enable signal for the RAM block and set the address multiplexer to channel read addresses #0 and #2 through to the address input of the RAM block.

- **Cycle 1:** Read address #1 and #3.
  The controller generates a read enable signal for the RAM block and set the address multiplexer to channel read addresses #1 and #3 through to the address input of the RAM block.

On the RAM block output data from address #0 and #2 are ready and latched in output buffers #0 and #2.

- **Cycle 2:** Write input #0 and #2.
  The controller generates a write enable signal for the RAM block and sets up the address and data multiplexers to channel through address and data #0 and #2, respectively. On the RAM block output data from address #1 and #3 are ready from the previous cycle and latched in output buffers #1 and #3.

- **Cycle 3:** Write input #1 and #3.
  The controller generates a write enable signal for the RAM block and sets up the address and data multiplexers to channel through address and data #1 and #3, respectively.

The controller uses an enable input signal to determine is data is valid for a read and write operation and to generate a data ready output signal for the designated datapath. Furthermore, the fast system clock is used to drive the controller, RAM block and buffers, and the slow data path clock is used for synchronization.

6.2 General Control Strategy

The execution of the algorithms can either be controlled by dedicated logic or by software running on a NIOS-II softcore processor.

The NIOS-II softcore processor is a processing unit implemented in dedicated logic which reads and executes instructions from RAM on the FPGA board. Several different configurations of the NIOS-II processor are supplied from Altera, each with different complexity [Altera, 2008a, section I.1].

The alternative is to implement a state machine in dedicated logic. This solution forgoes the flexibility in reconfigurability of software but instead offers a very simple implementation.

For this project the solution with state machines in dedicated logic is selected. This is the most power-efficient possibility as the implementation is kept simple and with no overhead compared to the softcore processor.

In situations requiring counters and decisions there exists the possibility of writing sequential code in VHDL or constructing the control flow directly from logic. No design-possibilities is thus precluded.

6.3 Number Representation

The data format and wordlength (number of bits) used for the representation can be completely customized in the FPGA implementation, but the higher the number of bits and thus precision, the higher the power usage for each arithmetic operation. On the other hand, a signal demodulation must be performed after the FFTs, so number precision and introduced noise cannot be

disregarded.

The FPGA can synthesize IEEE floating point arithmetic operators, but these introduce an operational overhead and uses a large FPGA area and thus resulting power. Instead an implementation using fixed point is used as this is more power efficient due to simpler implementation. The drawback is the introduction of a limited dynamic range and noise in the calculations due to truncation. Both of these effects will be discussed next.

Initially it is seen that both negative and positive numbers are needed in the FFTs, so the twos complement number representation is selected. As numbers greater than 1.0 must be represented a Q$I.F$ number format is used . The I represents the number of bits used for the integer part, while the F represents the number of bits assigned to the fractional [Oberstar, 2007]. A positive fractional number is written as

$$x = \sum_{i=0}^{I-1} 2^i + \sum_{f=1}^{F} 2^{-f} \tag{6.1}$$

and the negative counterpart is constructed by taking the twos complement. The maximal negative number cannot be constructed using (6.1), but is instead constructed as $10\ldots0_b$.

The highest resolution/increment for a Q$I.F$ number is thus $\frac{1}{2^F}$ while the dynamic range of a Q$I.F$ number x is $\left[-(2)^I \leq x \leq 2^I - \frac{1}{2^F}\right]$ which needs $I + F + 1$ bits due to the sign bit. This means that a number representation using 3 bits for the integer part and 4 bits for the fractional part would be a Q3.4 number with a total length of 8 bits due to the sign bit.

6.3.1 Integer Word Length

In determination of the integer word length, it is of interest to determine the highest possible value which can be attained in the algorithm. An initial test of the maximum values attained in the SRFFT has been performed in the C implementation. The results are shown in table 6.1. It is seen that the maximum value is 0.8536 which would lead to the offhand choice of reserving no bits for representing integers.

Attribute	Value
Mean	0.7671
Variance	0.0034
Maximum	0.8536

Table 6.1: *Maximum achieved values in the SRFFT. The test has been performed 300 times with different seed values to the randomly generated input data.*

While this is a valid choice for the system under the current constraint of ideal channel and thus no noise, this cannot be assumed for the environment in which the system is to function. In

that case describing the algorithm input as a uniformly distributed random variable in the interval $[-1; 1]$ would be more appropriate.

The worst case scenario is in the SRFFT is then where calculation of output at index 0 equals the addition of 1024 numbers all of value 1. See section 3.3.3 on page 22 for graphical example of this addition.

As it is wasteful to use $\log_2(1024) = 10$ bits for storing an integer, as will be shown later, the input is defined as a stochastic process and the possible distribution of the sum is examined.

Assume the input data an uniformly distributed random variable with range equivalent to the possible input data. Adding this random variable N times and examining the resulting Probability Density Function (pdf) shows the probability of generating an overflow, compared to the bit sizes used to store the integer. It is thus possible to select an acceptable chance for overflow with regards to the integer word length.

Define $Z = \sum_{n=1}^{N} X_n$ where X_n is an uniformly distributed random variable in the range $[-a : a]$ with zero-mean and Z is the resulting random variable. The pdf $f_z(x)$ of the sum of random variables can be determined from the known pdfs of X_n by calculating the characteristic function (6.2) and performing the conjugate inverse Fourier transform the to get (6.3)

$$\Psi_Z(\omega) = E\left[\exp(j\omega Z)\right] = \int_{-\infty}^{\infty} f_z(x) \cdot \exp(j\omega x)\, dx \tag{6.2}$$

$$f_Z(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \Psi_Z(\omega) \exp(-j\omega x)\, d\omega \tag{6.3}$$

[Shanmugan and Breipohl, 1988, p. 39-40]. It is possible to solve the problem by convolving the variables X_n, but this solution is not investigated in this project.

Writing the characteristic function $\Psi_Z(\omega)$ for Z and assuming $X_{1\ldots N}$ independent produces

$$\Psi_Z(\omega) = E\left[\exp\left(j\omega \cdot \sum_{n=1}^{N} X_n\right)\right] = \prod_{n=1}^{N} E\left[\exp\left(j\omega X_n\right)\right] = \prod_{n=1}^{N} \Psi_{X_n}(\omega) \tag{6.4}$$

The characteristic function for X_n is found by inserting the definition of the uniform pdf [Shanmugan and Breipohl, 1988, p. 43] into (6.2) and using the Euler equality of $\frac{e^{jz} - e^{-jz}}{2j} = sin(z)$

$$f_{X_n}(x) = \begin{cases} \frac{1}{b-a} & \text{, for} \quad a \le x \le b \\ 0 & \text{, otherwise} \end{cases} \tag{6.5}$$

$$\Psi_{X_n}(\omega) = \int_a^b \frac{1}{b-a} \exp(j\omega x)\, dx = \left[\frac{1}{j\omega(b-a)} \exp(j\omega x)\right]_a^b \tag{6.6}$$

$$\Psi_{X_n}(\omega) = \frac{1}{\omega} \frac{e^{j\omega b} - e^{j\omega a}}{j(b-a)} \tag{6.7}$$

$$\Psi_{X_n}(\omega) = = \frac{1}{\omega} \cdot \sin(\omega), \quad a = -1,\, b = 1 \tag{6.8}$$

Determining $f_Z(x)$ is done by using (6.3), (6.4) and (6.8).

$$f_Z(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \left[\prod_N \frac{sin(\omega)}{\omega}\right] \cdot \exp(-j\omega x)\, d\omega \tag{6.9}$$

that has a solution which is beyond the scope of this project. The integral is solved in [Bradley and Gupta, 2007, p. 11] which produces the equation

$$f_Z(x) = \frac{1}{(N-1)!(2\cdot a)^N}\sum_{k=0}^{N}(-1)^k\binom{N}{k}\cdot(x+(N-2k)a)_+^{N-1} \tag{6.10}$$

where $(y)_+^{N-1}$ is defined in [Bradley and Gupta, 2007, p. 8] to be

$$(y)_x^{N-1} = \begin{cases} 1\cdot y^{N-1} & ,\text{for} \quad y>0 \\ \frac{1}{2}\cdot y^{N-1} & ,\text{for} \quad y=0 \\ 0 & ,\text{for} \quad y<0 \end{cases} \tag{6.11}$$

and define $a = b$ and with zero mean. The last definition is due to different definitions of the uniform distribution.

Thus the pdf for Z as a sum of uniform random variables is determined. The result for different N is plotted in figure 6.3 on the next page as pdf and Cumulative Distribution Function (cdf) . It is seen that the maximum possible value of the pdf expands as more and more summations are performed. This corresponds well to the expected result with increasing, but unlikely, maximum value.

Determining the required integer range is done by examining the value of the cdf at points $-2^I, I = 1\ldots(\max)$. This is half the probability that an overflow at the integer representation by I will occur.

Unfortunately the binomial distribution $\binom{N}{k}$ implementation in MatLAB gives inaccurate results for $N > 40$ in the project tests. Thus the pdf and cdf for $N > 40$ cannot be numerically determined based on (6.10). The exact solution to this problem is deemed outside the scope of this project and assumptions are used instead.

Examining figure 6.3 it is seen that the tendency for the values of the cdf (lower figure) is to approach the numerical extremes with increasing N. As the exact values of the cdf for $N = 1024$ are not determined, a value of $I = 4$ is deemed reasonable by the project group. This yields a dynamic range of approximately $\pm 2^4 = \pm 16$. The value of 4 bits as representative of the integer part is selected on the basis of low chance of overflow while keeping the number of bits as low as possible.

6.3.2 Fractional Word Length

Examining the FPGA platform it is seen that memory cells are able to operate in ranges of $(1, 2, 48, 9, 16, 18, 32, 36)$ bits [Altera, 2007c, p. 4-1]. Of these the 16- and 18-bit operations are selected by the project group as relevant, the lower bit numbers would leave to few fractional bits, with an integer length of $I = 4$, while the higher bit numbers would spend too much power.

The embedded multipliers either performs two 9×9 or one 18×18 bit multiplications. In keeping with the design of the FPGA, the 18×18 operation is selected, as this gives the highest precision and is natively supported by the platform [Altera, 2007c, p. 5-5].

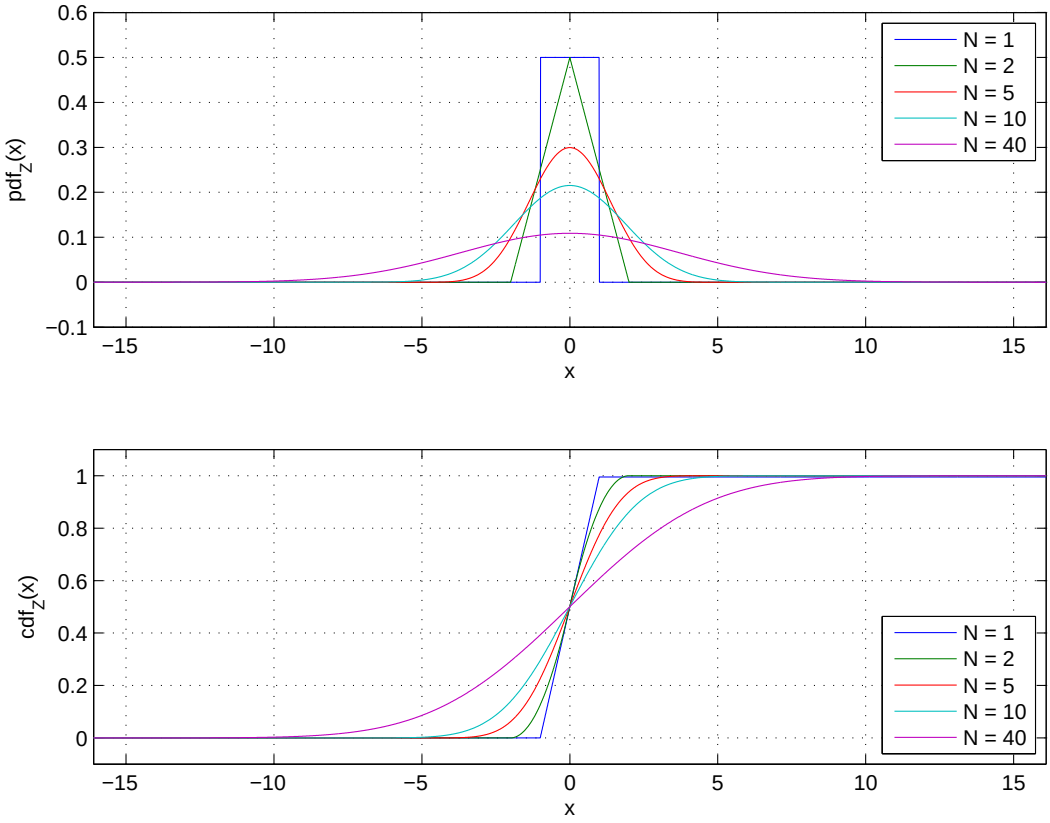

Figure 6.3: *The sum of uniform random variable $Z = \sum_{n=1}^{N} X_n$ shown as pdf (upper figure) and cdf (lower figure). The x-axis represents the FFT output while the y-axis represents probability.*

Accounting for the sign bit and selection of integer part yields a Q4.13 number format wherein thirteen bits are assigned to the fraction. This results in a number resolution of $R = \frac{1}{2^F} = 122.1 \cdot 10^{-6}$, which is the lowest number and increment the chosen number format can represent.

As the output are QPSK with values of $(\pm 1 \pm j) \cdot \frac{1}{\sqrt{(2)}}$, the maximum attainable SNR is

$$SNR_{db} = 20 \cdot \log_{10}\left(\frac{|X|}{|(1+j) \cdot R|}\right) = 115.3 \text{ dB} \tag{6.12}$$

Definition of the number format could also be performed by examining the requirements for numerical precision in the fractional representation, which could yield a different result than the one defined here.

This approach requires definition of the maximum allowed introduced noise in the FFT, which is typically set in conjunction with the overall system design. As this project does not cover a complete OFDM receiver system and corresponding definition of noise requirements, this approach is disregarded. The examination of achieved precision is thus left for later discussion.

It must also be noted that a lower number of bits in the numerical representation would yield a lower power usage. This possible optimization is also left for later examination.

6.4 Arithmetic Operations

Common for both algorithms is the arithmetic operations of addition, subtraction and multiplication. These operations and the effects on precision is discussed next.

The operation of addition and subtraction in twos complement is trivial and without loss of precision, as long as no overflow occurs. This should be ensured by the data format of Q4.13 and the investigations performed in the previous section.

Multiplication with binary numbers produces a result of double word length. This is impractical with a datapath that is used repetitively in the algorithm and would produce a very long word-representation. Instead the result can be truncated to the original number representation as shown in figure 6.4.

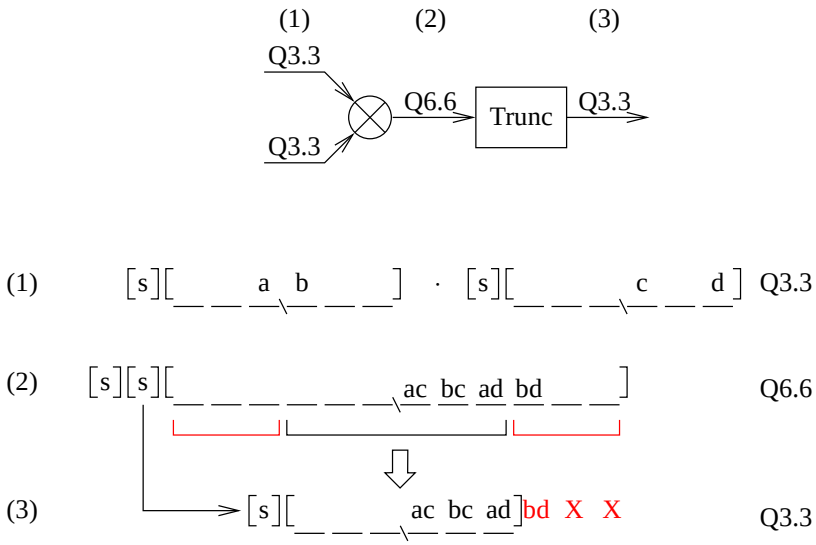

Figure 6.4: *Multiplication and truncation where a, b, c and d are binary variables. Bits marked with red are discarded and introduces an error.*

Part (1) of figure 6.4 shows two example Q3.3 words. The "s" represents sign bits and the Least Significant Bit (LSB) is to the far right. This example is for Q3.3 representations but is equally true for Q4.13.

Multiplying the two numbers yields part (2) which is double precision (Q6.6). The multiplication for the example can be calculated as multiplication of integers

$$
\begin{aligned}
&= (a \cdot 2^3 + b \cdot 2^2) \cdot (c \cdot 2^2 + d \cdot 2^0) && (6.13)\\
&= ac \cdot 2^{2+3} + bc \cdot 2^{2+2} + ad \cdot 2^{3+0} + bd \cdot 2^{2+0} && (6.14)\\
&= ac \cdot 2^5 + bc \cdot 2^4 + ad \cdot 2^3 + bd \cdot 2^2 && (6.15)
\end{aligned}
$$

The result is a double-length representation which must be truncated to the original format, part (3), to retain the original data precision. This is done by right shifting with the number of bits representing the fraction

$$
\begin{aligned}
&= \frac{ac \cdot 2^5 + bc \cdot 2^4 + ad \cdot 2^3 + bd \cdot 2^2}{2^{F=3}} && (6.16)\\
&= ac \cdot 2^2 + bc \cdot 2^1 + ad \cdot 2^0 + bd \cdot 2^{-1} && (6.17)
\end{aligned}
$$

and discarding numbers with representation $2^{(F<0)}$ as marked with red.

From part (3) of figure 6.4 it is seen that the uppermost integer bits and the lowermost fractional bits are discarded. The integer bits does not constitute a loss of precision, as long as the multiplication does not produce an overflow. Truncating the fractional bits introduces a variable noise in the result, due to lack of representation. This noise approaches $\frac{3}{4}$LSB in the worst case as the sum of the discarded bits. The actual noise introduced in the multiplication and subsequent truncation depends on the actual numbers.

An example with the Q3.3 numbers from figure 6.4 is the multiplication of 1.5 with 0.625. These are represented by integers in part (1) by $2^3+2^2 = 12$ and $2^2+2^0 = 5$ where $\{a, b, c, d\} = 1$. Multiplying and shifting yields

$$
\left\lfloor \frac{12 \cdot 5}{2^{(F=3)}} \right\rfloor = \lfloor 7.5 \rfloor = 7 \qquad (6.18)
$$

Converting the result, 7, to the Q3.3 representation yields $2^{-1} + 2^{-2} + 2^{-3} = 0.875$ whereas the correct result is $1.5 \cdot 0.625 = 0.9375$. The value of 0.5 which is truncated away in (6.18) is the value of bd in part (3) which yields the inaccurate result.

Relating this to the chosen number representation of Q4.13 it is seen that each multiplication approximately introduces a maximum noise of $\frac{3}{4}R = 91.6 \cdot 10^{-6}$ per multiplication.

The operation of truncation after multiplication in the design, is performed by a clocked VHDL block shown in figure 6.5 on the next page. This block is designed for accepting Q4.13 $\cdot$ Q4.13 = Q8.26 which is a 36 bit word with two sign bits. The input is moved to the correct output at the rising clock edge and is ready for processing by the next design entry. All multiplications on the FPGA is performed in this manner.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY shiftDown13 IS
5    PORT (
6      clock : IN std_logic;
7      input: IN BIT_VECTOR(35 downto 0);
8      output : OUT BIT_VECTOR(17 downto 0)
9    );
10 END shiftDown13;
11
12 ARCHITECTURE BEHAVIOR OF shiftDown13 IS
13
14 BEGIN
15   picker : PROCESS(clock)
16   BEGIN
17     IF (rising_edge(clock)) THEN
18       -- grab a sign bit
19       output( 17 ) <= input( 34 );
20       -- grab the lower QI integer bits
21       output(16 downto 13) <= input(29 downto 26);
22       -- grab the QF fractional bits
23       output( 12 downto 0 ) <= input( 25 downto 13 );
24     ELSE
25
26     END IF;
27   END PROCESS picker;
28 END BEHAVIOR;
```

Figure 6.5: *VHDL code for truncation after multiplication. The truncation is done by wire rerouting instead of a shift-register. This VHDL block is placed after multipliers as shown between stage (b) and (c) in figure 6.4 on page 55.*

6.5 Summary

The preceding three sections have defined basis for the overall structure of the FFT systems to be designed in the following chapters. The system environment has been designed in terms of the common RAM#1 block which interfaces the systems data paths enabling four sample read and write in each datapath clock cycle. Furthermore, a general approach for controller design based on finite state machine is chosen over the NIOS-II soft processor.

Finally, a discussion of the system data format was conducted leading to a choice of a Q4.13 fixed point number representation, based on a tradeoff between reducing the risk of overflow and ensuring sufficient decimal precision.

Chapter 7

Split-Radix FFT Mapping

The mapping of the SRFFT algorithm explains how the algorithm derived in section 3.3 is partitioned into datapath and controlpath and how these partitions are implemented on the FPGA system. Furthermore the necessary clock domains needed to drive the system are discussed. At last a summary is given along with examination of the hardware utilization.

7.1 Tasks

The overall task of the SRFFT to be implemented is to calculate a full 1024 point FFT. As described in section 3.3 the SRFFT consists of a series of L-shaped butterflies and a repeated subdivision of the transform into subtransforms of length N/2 and N/4. The resulting structure of the transform is shown in figure 7.1 for a 32 point example SRFFT.

Starting at level 0, a block of total length 32 consisting of 8 L-butterflies is calculated. Next at level 1, 4 L-butterflies are calculated giving a total block length of 16. At level 2, three blocks of length 8 starting at addresses 0, 16 and 24 are calculated. Level 3 consists of 5 4point L-butterflies, and level 4 of 11 2 point butterflies. As may be derived from the figure, the SRFFT consists of $log_2(N)$ levels at which a number of L-butterfly blocks (stages) need to be calculated.

The result of the SRFFT algorithms is the Fourier transform in bit-reversed order. A final stage of reordering the data could be included. However, the most efficient method for reordering the data is for the system, which is to read the result, to have its address bus connected to the RAM holding the results in bit-reversed order.

Thus two principal datapath structures are needed to perform the SRFFT are an L-butterfly and a 2 point DFT. The design and implementation of these two datapaths is examined in section 7.2. Furthermore, a control structure managing the levels and stages progress is needed to generate the addresses of the data to be calculated supply datapaths with the relevant data and storing the computed results is needed. This control structure is designed in section 7.3.

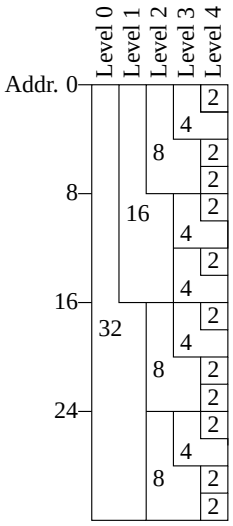

Figure 7.1: *Principal structure of a full 32 point SRFFT.*

Finally, the SRFFT environment consists of a RAM block and a clock controller. Since the SRFFT can be calculated in place, the RAM#1 designed in section 6.1.1 is used for both input and output. The tasks of the SRFFT are gathered in the system abstraction model in figure 7.2.

7.2 Datapath

This sections accounts for the mapping of the datapaths of the SRFFT, including the L-butterfly and 2 point butterfly needed to calculate the FFT.

7.2.1 L-Butterfly

The functionality of the L-butterfly is to calculate the 4 expressions in equations (7.1) to (7.4):

$$y[0] = x[0] + x[2] \tag{7.1}$$
$$y[1] = x[1] + x[3] \tag{7.2}$$
$$y[2] = (x[0] - x[2] + j\,(x[1] - x[3])) \cdot T_N^{nm} \tag{7.3}$$
$$y[3] = (x[0] - x[2] - j\,(x[1] - x[3])) \cdot T_N^{3nm} \tag{7.4}$$

where $x[0..3]$ are four complex inputs, $y[0..3]$ are the four complex outputs to be returned to the RAM and T_N^k are twiddle factors found from the current level, m, and counting variable, $n \in [0..N/4-1]$, where N is the length of the L-butterfly block at level m.

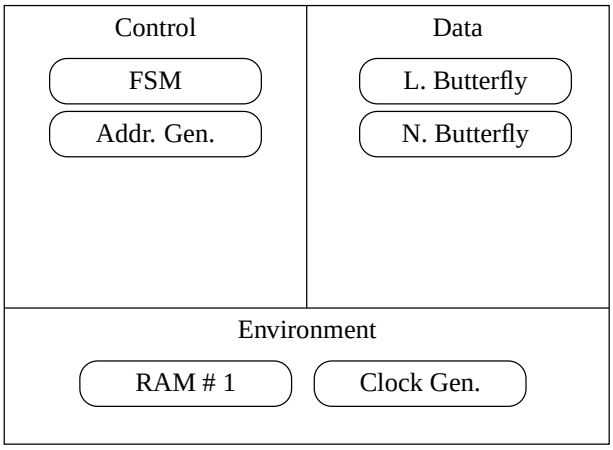

Figure 7.2: *Tasks for SRFFT*

Design Choices

To optimize the utilization of the calculation units constituting the L-butterfly datapath it is decided to pipeline the function in order for it to accept a set of input samples and produce a set of output samples at each clock cycle.

Furthermore, input and output data are kept aligned in time. The calculation of y[0] and y[1] is composed of only one summation, whereas y[2] and y[3] are both composed of two subtractions followed by a multiplication, introducing a inherent misalignment of output data. This decision leads to buffering of the calculated results of y[0] and y[1] to obtain equal latency of all results in the L-butterfly datapath.

Implementation

The structure of the implementation of the L-butterfly datapath is shown in figure 7.3, where each computation of the L-butterfly datapath defined in equations (7.1) to (7.4) is implemented as a separate calculation unit to enable the chosen full pipelining of the datapath.

The data and computations in the figure are complex, thus:

- complex addition is calculated as:

$$(a + jb) + (c + jd) \quad = \quad (a + c) + j(b + d) \tag{7.5}$$

- complex subtraction is calculated as:

$$(a + jb) - (c + jd) \quad = \quad (a - c) + j(b - d) \tag{7.6}$$

- complex multiplication is calculated as:

$$(a + jb) * (c + jd) \quad = \quad (ac - bd) + j(ad + bc) \tag{7.7}$$

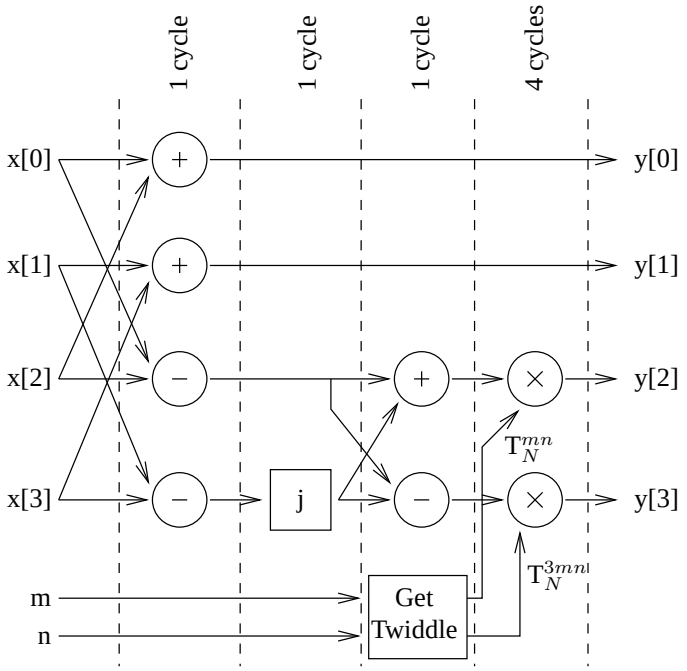

Figure 7.3: *Implementation of L-butterfly for SRFFT. The figure shows the scheduling of computations in the L-butterfly, resulting in a total latency of 7 clock cycles. All computations are complex and data not operated on during a cycle is buffered for alignment in time.*

Multiplication by j is done by explicitly mapping the connections of the subtractor output of ($r = y[1] - y[3]$) to $r\prime$:

$$\begin{aligned} Re\{r\prime\} &= -Im\{r\} \\ Im\{r\prime\} &= Re\{r\} \end{aligned}$$

where the negation of $Im\{r\}$ is the source of the one cycle delay of this block.

Finally, the generation of twiddle factors for the complex multiplication is implemented as two lookup-tables, one containing T^{mn} and one containing T_N^{3mn}, which are both complex. The splitting of T_N^{mn} and T_N^{3mn} into separate tables is a tradeoff between minimizing the table sizes and easing the look-up algorithm. At present, the first third of the T_N^{3mn} is already present in the T_N^{mn} table. This overhead could be eliminated by merging the two tables at the cost of a more complex derivation of the look-up instances for the last two thirds of the T_N^{3mn} values.

7.2.2 Two-Point Butterflies

The two-point butterfly data path is used to calculate the last level of the SRFFT where the sub-transform length is less than 4. The function that are calculated by the two-point butterfly

datapath was described in section 3.3:

$$y[0] = x[0] + x[1] \tag{7.8}$$
$$y[1] = x[0] - x[1] \tag{7.9}$$

Design Choices

In order for the 2-point butterfly datapath to fit the data format of the RAM#1 block, it is decided to copy the 2-point butterfly data path, defined above, and perform two in parallel. This, of course is done at the cost of utilizing additional LEs on the FPGA, but the computation time for the calculation of this level is effectively reduced by half, and this approach is estimated to be more efficient than to change the design of the RAM#1 block to be able to handle not reading and writing 4 values per clock.

Implementation

The principal structure of the 2-point butterfly implementation is shown in figure 7.4. The 2-point butterfly datapath consists of two complex adders and two complex subtractors to compute two instances of the equations (7.8) and 7.9.

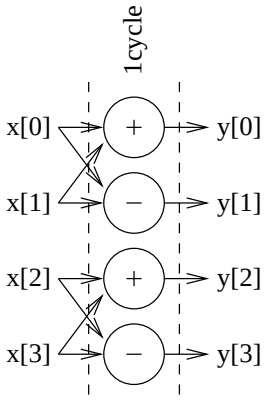

Figure 7.4: *Structure of datapath implementation for 2-point butterfly. The data and operations are complex valued.*

7.3 Control Path

The purpose of the control path is to manage the data paths by supplying input data and storing output data. In this section the structure of these tasks are divided into a general control path and address generators.

7.3.1 General Control Path

The purpose of the general control path is to manage the overall flow of the SRFFT algorithms execution. The main task of the controller is thus to activate and deactivate the L-butterfly and 2-point butterfly datapaths and corresponding address generators when required.

Design Choices

As discussed in section 6.2 the controller is implemented as a finite state machine. The basis for the FSM are one state for activating the L-butterfly address generator and datapath and one state for the 2-point butterfly datapath and address generator. The activation signal generated by the FSM is rippled through the address generator, RAM#1 block and data path pipelines and used to indicate the presence of valid data.

To enable detection of the completion of the address generator, where this need no longer be active, the address generator issues a completion signal inducing a state change in the control FSM. At this point, the L-butterfly datapath is still calculating due to the pipelining and the RAM# block is thus not available for the 2-point butterfly address generator to be activated.

A flush state is therefore inserted to wait for the L-butterfly datapath to empty. When entering the flush state, a signal is rippled through the L-butterfly datapath to detect, when the flush is complete, and the controller can continue to activate the 2-point butterfly address generator and datapath. For the 2-point butterfly states, the same procedure for detecting address generator completion and datapath flush.

Implementation

The resulting finite state machine for the general control path is shown in figure 7.5.

At system startup a reset state (`reset_s`) is added to ensure that all data paths and address generators are in a predefined state. Next, the system goes into an idle state (`idle_s`) waiting for a go input signal. For test purposes this signal is generated manually using one of the starter kit input buttons, instead of being produced periodically by a system process.

When the go signal is generated, the state machine moves into the `doLButterfly_s` state, activating the L-butterfly address generator and datapath. The rest of the SRFFT algorithm execution follows the procedure described in the previous section. Finally, when the 2-point butterfly pipeline is flushed the state machine returns to idle state and polls the go signal until this again is present.

7.3.2 Address Generators

The final components of the SRFFT are the address generators. The task of these generators is to supply the addresses of the datapath input to the RAM#1 block. Two address generators need to be implemented; one for the L-butterfly state and one for the 2-point butterfly state. Both are designed using the same principles and therefore examined together in this section.

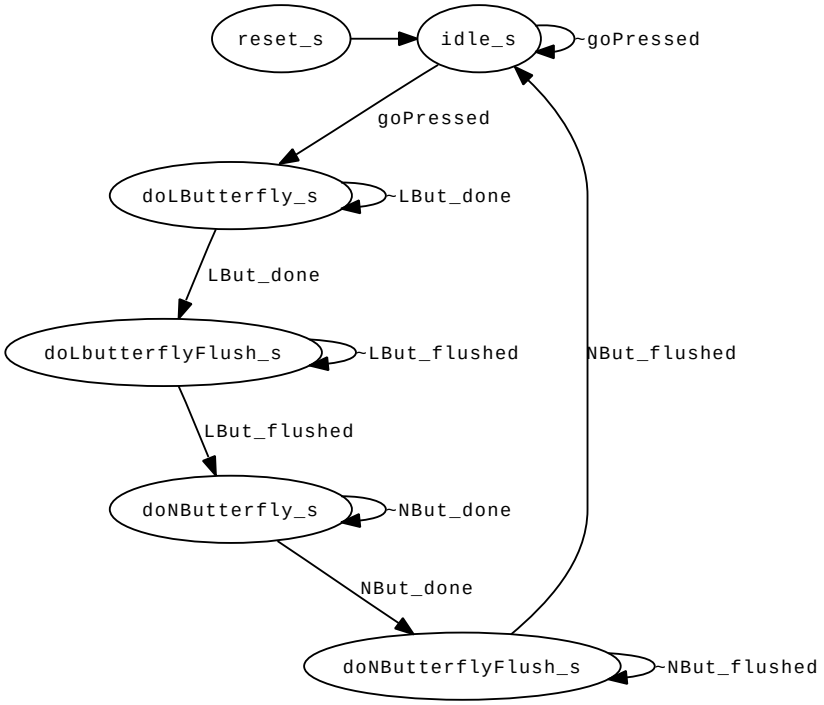

Figure 7.5: *SRFFT controlling state machine*

Design Choices

The design of the address generators for the SRFFT algorithm is based on [Skodras and Constantinides, 1992]. The main challenge for the L-butterfly state is to determine the location and length of each of the blocks of L-butterflies. Using the C-algorithm for generating look-up table containing the block start addresses, as suggested in [Skodras and Constantinides, 1992], along with a second LUT containing the number of L-butterflies to be calculated in the corresponding block, supplies the data needed for address generation.

For the 2-point butterfly state, a similar procedure may be used. In this state the length of the transform is always one, thus only a start address of each butterfly to be calculated is needed.

Alternatively, the LUT-generating algorithm could be implemented directly. This would reduce the memory requirements of the address generators significantly, since addresses and lengths then would be generated dynamically and not need storing. However, this approach would also require more calculations to be performed between each datapath clock cycle, and the address generators would require more area LE area utilization for dynamic start address and block length calculation. Therefore, the approach where LUTs containing the algorithm results is chosen for implementation in this system.

Implementation

The structures of the two address generators for the L-butterfly and 2-point butterfly states are shown in figures 7.6 and 7.7, respectively.

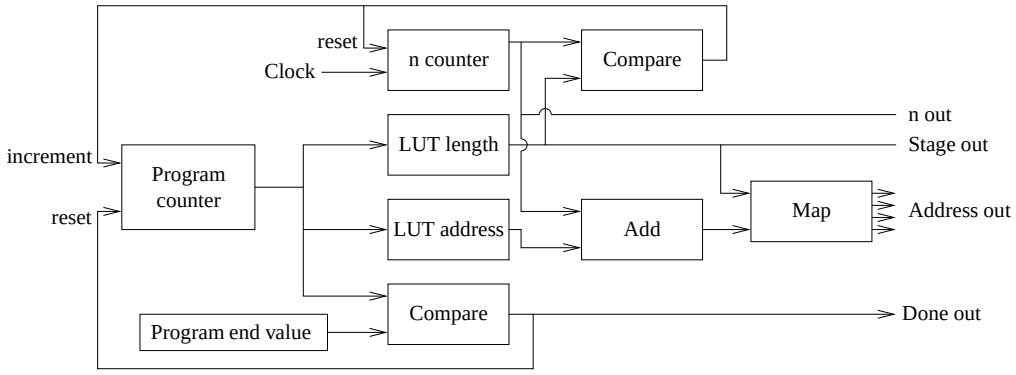

Figure 7.6: *Structure of L-butterfly address generator implementation*

The address generation for the L-butterfly state is composed of two counters and two look-up tables, which in effect constitute an outer loop counting through the L-butterfly blocks and an inner loop counting through the individual L-butterflies of each block.

A program counter defines the active entry in the address- and length look-up tables. The address entry defines the start address of the active L-butterfly block, and the length describes both the number of L-butterflies in the block and the input sample displacement. Thus the value read from the address LUT is used to compare to the n counter, which is incremented by the clock signal, to enable reset of the n counter and increment the program counter when n and the LUT entry are equal.

The first data addresses of each L-butterfly is found by adding the block start address from the address LUT with the n counter value. Relating this address to the input data described in figure 7.3, this address is a(x[0]). The final three addresses, a(x[1..3]), are found by adding multiples of the length to a(x[0]) to end up with:

$$a(x[0]) = \texttt{(LUT address)} + \texttt{(n counter)} \tag{7.10}$$
$$a(x[1]) = a(x[0]) + \texttt{(LUT length)} \tag{7.11}$$
$$a(x[2]) = a(x[0]) + 2 \cdot \texttt{(LUT length)} \tag{7.12}$$
$$a(x[3]) = a(x[0]) + 3 \cdot \texttt{(LUT length)} \tag{7.13}$$

Finally, the value of n and the length LUT is also sent to the L-butterfly datapath for twiddle factor look-up.

The address generator for the 2-point butterflies is significantly simpler, as shown in figure 7.7. Since the 2-point butterflies are always working on to successive data samples, the address generator need only read the start address and add one to have the necessary addresses for one 2-point butterfly. In order to accommodate the four sample structure of the RAM#1, two LUT

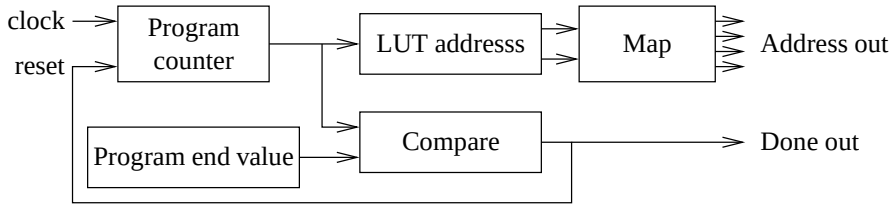

Figure 7.7: *Structure of 2-point butterfly address generator implementation*

values are read for each program count, generating the necessary four addresses. Relating the address generator output to the data input of the 2-point butterfly datapath described in figure 7.4, the four addresses, a(x[0..3]), generated in each datapath clock cycle are:

$$\begin{aligned} a(x[0]) &= \texttt{LUT}[\cdot\texttt{Program counter}] \\ a(x[1]) &= \texttt{LUT}[2 \cdot \texttt{Program counter}] + 1 \\ a(x[2]) &= \texttt{LUT}[2 \cdot \texttt{Program counter} + 1] \\ a(x[3]) &= \texttt{LUT}[2 \cdot \texttt{Program counter} + 1] + 1 \end{aligned} \tag{7.14}$$

7.4 Clock Domains Generation

As mentioned in the mapping sections, the design require two clock domains; one 4 times faster than the other. The down conversion from the fast clock may be accomplished either by utilizing one of the embedded Phase-Locked Loops in the device or, since the down conversion factor is a power of 2, by implementing a 2 bit counter and where the MSB will have a frequency of 1/4 of the clock driving the counter.

The global system clock in the development board is fixed at 50 MHz. However, a reconfigurable main clock is required for adjusting the system clock to investigate power consumptions dependency of clock frequency. Thus, the PLL solution is chosen. This way both clocks may be down converted with arbitrary values.

As will become clear in the test section 9.1, the SRFFT implementation is tested both with the full clock of 50 MHz resulting in the datapath being driven at 12.5 MHz and a clock reduced by 1/3, i.e. 16.7/4.17 MHz where the available time of 0.5 ms for calculating the FFT is fully utilized.

7.5 Summary

With the tasks for the SRFFT datapath, control path and environment implemented, the achieved design is summarized in this section before moving on to mapping the SFFT. An overview of the SRFFT system implementation is shown in figure 7.8.

In the figure, the FSM controller and clock signals are left out for overview. The multiplexers and demultiplexers are used to direct addresses and data to and from the RAM#1 block and active

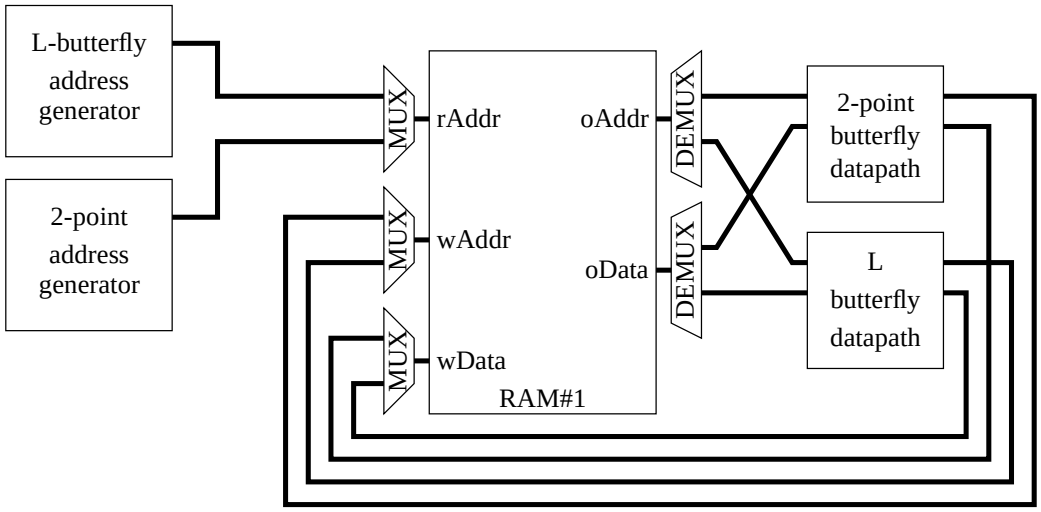

Figure 7.8: *Overview of SRFFT address generators, RAM#1 and datapath interconnection. For overview, the FSM controller and clock signals are left out.*

address generator and datapath. As seen in the figure the addresses generated are rippled through the RAM#1 block and active datapath to supply write addresses aligned with datapath output data to the RAM#1 block.

7.5.1 Hardware Utilization

To get an indication of the resulting hardware utilization of the achieved SRFFT system is reasonably comparable with other reference designs when measured on hardware utilization, the system is compared in table 7.1 to the same FFT reference design used in section 4.2 to indicate the hardware requirements.

Feature	Available	SRFFT utilization	Reference design
Logic Elements	24,624	3,299	3,033
Memory [kBits]	504	66.6	102.4
Multipliers (18x18)	66	16	3

Table 7.1: *SRFFT system hardware utilization compared to available resources and reference design used to estimate hardware requirements in section 4.2.*

As seen in the table the spend number of LEs are within 10% of each other, the memory usage approximately 35% lower in the SRFFT system, and the utilized number of embedded multipliers 5.3 times higher in the current implementation. It must be noticed that the functionality of the two systems are not exactly equal, and the reference may thus only be used as an indication. The relatively high utilization of multipliers is a consequence of the datapath structure where all computations of the L-butterfly and 2-point butterflies are assigned separate hardware resources.

Based on this condition the achieved hardware utilization for the SRFFT system is considered acceptable.

A further examination of the SRFFT system into the power consumption performance of the system is conducted in the test section 9.1 on page 87.

The SRFFT system serves as benchmark for the calculation of a full DFT for comparison with the calculation of only the subset of subcarriers of interest using the SFFT. Furthermore, the SRFFT composes a significant component of the SFFT, which is mapped in the next chapter.

Chapter 8

Sørensen FFT Mapping

The Sørensen FFT described in section 3.4 on page 23 is mapped to the FPGA platform in this chapter.

Initially the tasks needed to perform the SFFT are examined. This results in an extra datapath which performs the recombination. This datapath requires its own control mechanism which is designed afterwards. At last the overall system-clock is adjusted to achieve the correct time-performance and a summary including hardware utilization is given.

8.1 Tasks

Equation (3.20) on page 25 is repeated in equation (8.1):

$$X[k] = \sum_{n_2=0}^{Q-1} \underbrace{\left[\sum_{n_1=0}^{P-1} x_{n_2}[n_1] \cdot T^{n_1 \cdot \{k\}_p} \right]}_{X_{n_2}[r]=\sum_{n_1}^{P-1} x_{n_2}[n_1] T_p^{n_1 \cdot r}} \cdot T_N^{n_2 \cdot k} \tag{8.1}$$

Examining equation (8.1) it is seen that the SFFT is a sum of the outputs of smaller FFTs, thus the byname 'transform decomposition'. This is further elaborated in figure 3.3 on page 26 where the recombination steps are shown as additions.

A number of Q FFTs of length P are thus required to compute the SFFT. It is decided to use the split-radix FFT described in chapter 7 to perform these FFTs. This is done as the design and mapping already exists, and the SRFFT algorithm is an efficient method for calculating the FFT.

The project SRFFT requires changes to the address generator and lookup-tables used in the execution. These changes consists of performing a $P = 32$ length FFT $Q = 32$ times with increasing base offsets, instead of a full 1024-length FFT. These changes are trivial and are not discussed further. The values of P and Q are selected in section 3.5.4 on page 27.

Returning to (8.1) it is seen that the recombination step is a series of Multiply and ACcumulate

(MAC) operations. The abstraction model for the SFFT is thus extended, based on the SRFFT abstraction shown in figure 7.2, with a recombination part in the control- and datapath as seen in figure 8.1.

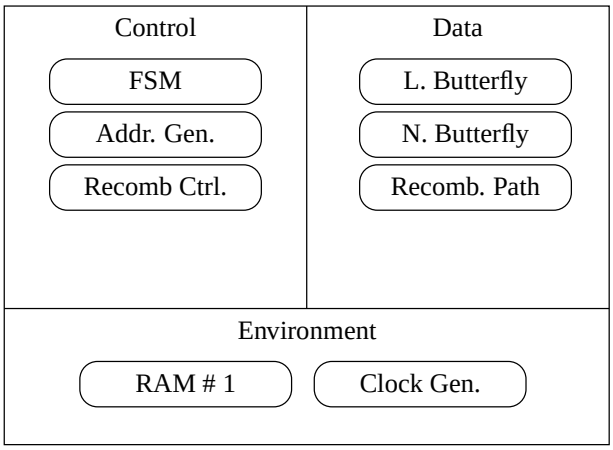

Figure 8.1: *Tasks for SFFT*

An extension of the interface definitions in figure 6.1 on page 48 is provided for the case of the SFFT in figure 8.2 on the facing page. A further explanation of the connections and entities are presented in the following section 8.3 on page 76.

Each of the control- and datapath entries are designed in the following.

8.2 Datapath

Requirements

The operations to be performed in the SFFT datapath is

$$X[k] = \sum_{n_2=0}^{Q-1} x_{n_2}[\{k\}_P] \cdot T_N^{n_2 \cdot k} \tag{8.2}$$

which is a rewritten equation (3.20) where it is assumed that $x_{n_2}[r], r = 0 \ldots P-1, \quad r = \{k\}_P$ is the FFT transformed data.

Design Choices

A possible optimization is presented in [Sørensen and Burrus, 1993, p. 1188], where (8.2) is rewritten as a convolution

$$X[k] \quad = \quad \sum_{m=0}^{Q-1} \left(x_{Q-m-1}[\{k\}_P]\right) \left(T_N^k\right)^{Q-m-1}, \quad n_2 = Q-m-1 \tag{8.3}$$

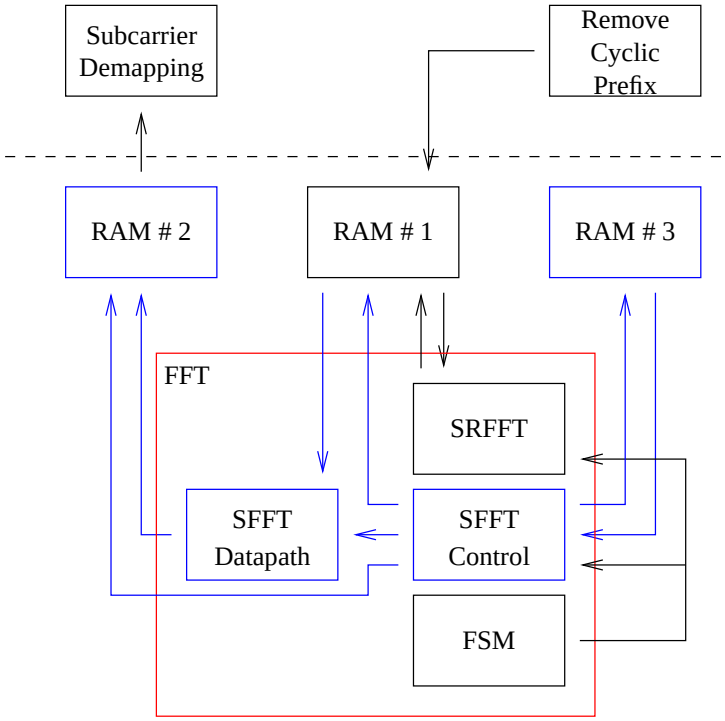

Figure 8.2: *The SFFT shown in conjunction with the interface definitions in figure 6.1 on page 48. The connections and entities are the only ones discussed in this section.*

Exchanging the upper boundary with j but keeping the input sequence yields

$$y_k[j] \quad = \quad \sum_{m=0}^{j-1} x_{Q-m-1}[\{k\}_P] \cdot T_N^{k\cdot(j-m-1)} \tag{8.4}$$

As j approaches Q the equation $X[k] = y_k[j]\Big|_{j=Q}$ is true. This is a convolution with a system

$$H_k(z) \quad = \quad \frac{z^{-1}}{1 - z^{-1} \cdot T_N^k} \tag{8.5}$$

and input $x_{Q-j-1}[\{k\}_P]$.

Rewriting the impulse response as

$$H_k(z) \quad = \quad \frac{z^{-1} \cdot (1 - z^{-1} T_N^{-k})}{(1 - z^{-1} \cdot T_N^k)(1 - z^{-1} T_N^{-k})} \tag{8.6}$$

$$H_k(z) \quad = \quad \frac{z^{-1} \cdot (1 - z^{-1} T_N^{-k})}{1 - 2 \cdot \cos\left(\frac{2\pi k}{N}\right) z^{-1} + z^{-2}} \tag{8.7}$$

and drawing the direct form II realization in figure 8.4 it is seen that the number of multiplications per iteration is reduced from 1 complex multiplication (2 real multiplications and 2 real additions) to one real and imaginary scaling (2 real multiplications). The operations right of the dotted line are only performed once to compute the output, as no results are dependent on the feed-forward network.

This saves a complex multiplication compared to figure 8.3, instead a real and imaginary scaling is performed.

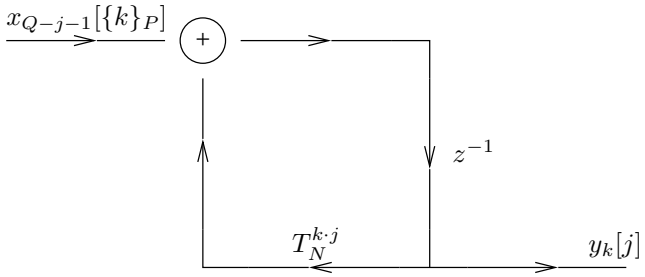

Figure 8.3: *Direct form II structure for MAC operation, see (8.5). Modified from [Sørensen and Burrus, 1993, figure 5.]*

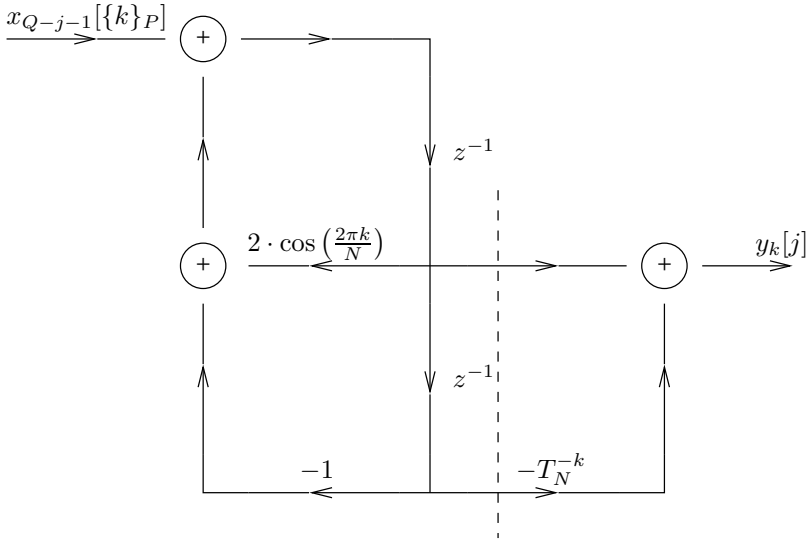

Figure 8.4: *Direct form II structure for modified MAC operation. Operations right of the dotted line are only performed once, thus saving multiplications. See (8.7). Modified from [Sørensen and Burrus, 1993, figure 6.]*

The downside to rewriting (8.7) (henceforth known as Method 2) is the presence of feedback

with an amplitude greater than 1, which can cause instability. A test of the achieved values is performed in C for 300 different FFT input sets. The results are shown in table 8.1.

	M2 Re	M2 Im	MAC Re	MAC Im
Mean	24.58	24.68	1.52	1.52
Variance	6.91	8.31	0.003	0.004
Maximum	32.60	34.81	1.67	1.70

Table 8.1: *Comparison of dynamic range in SFFT of Method 2 (M2) and standard Multiply-Accumulate (MAC). All possible values computed for 300 different seed-values. The table shows that Method 2 requires a higher dynamic range in the data representation, as a tradeoff it has a lower computational complexity.*

Of primary interest is the maximum achieved values for Method 2 compared to the maximum achieved values for the MAC-solution. The dynamic range of Method 2 is thus much greater and would require a data format of Q6.13 instead of the Q4.13 which is used in the rest of the design. See section 6.3 on page 50 for information about numerical representation. The decision between Method 2 and MAC is then based on the following

- Method 2 uses two more bits in the numerical representation than the MAC solution
- Method 2 uses two more registers for temporary storage than the MAC solution
- Method 2 uses fewer arithmetic operations than the MAC solution

Examining the itemized list, it is seen that the two extra needed bits and the two extra needed registers are outweighed by the complex multiplication in the MAC solution. Method 2 is then selected for implementation as it requires fewer arithmetic operations. The tradeoff is the use of more logic elements.

Implementation

The implementation is based on figure 8.4 on the facing page. The negation of the poles is implemented by subtraction instead of summation and the scaling by $2 \cdot \cos\left(\frac{2\pi k}{N}\right)$ is implemented directly as a multiplier. The forward path multiplication twiddle factors and addition is implemented as a complex multiplier and standard adder. The result is seen in figure 8.5 on the next page.

This figure also shows the conversion blocks between Q4.13 and Q6.13 and the ROMs used for the scaling factors $2 \cdot \cos$ and the twiddle factors. As the computational structure requires all summations and multiplications to be performed before data is latched, it is necessary to add intermediate cycles to the structure. The recombination datapath thus also uses the four times faster clock, which is used in RAM#1, but no design entities are clocked directly from the fast clock. The colored c_n shows in which sub-cycle the entity becomes active.

To stay in synchronization with the rest of the system, the methodology used in clock-assignment is that of doing the latching last and working backwards through the latches dependencies. The converter and simple multiplier is triggered in cycle 0, c_0 , and produces data

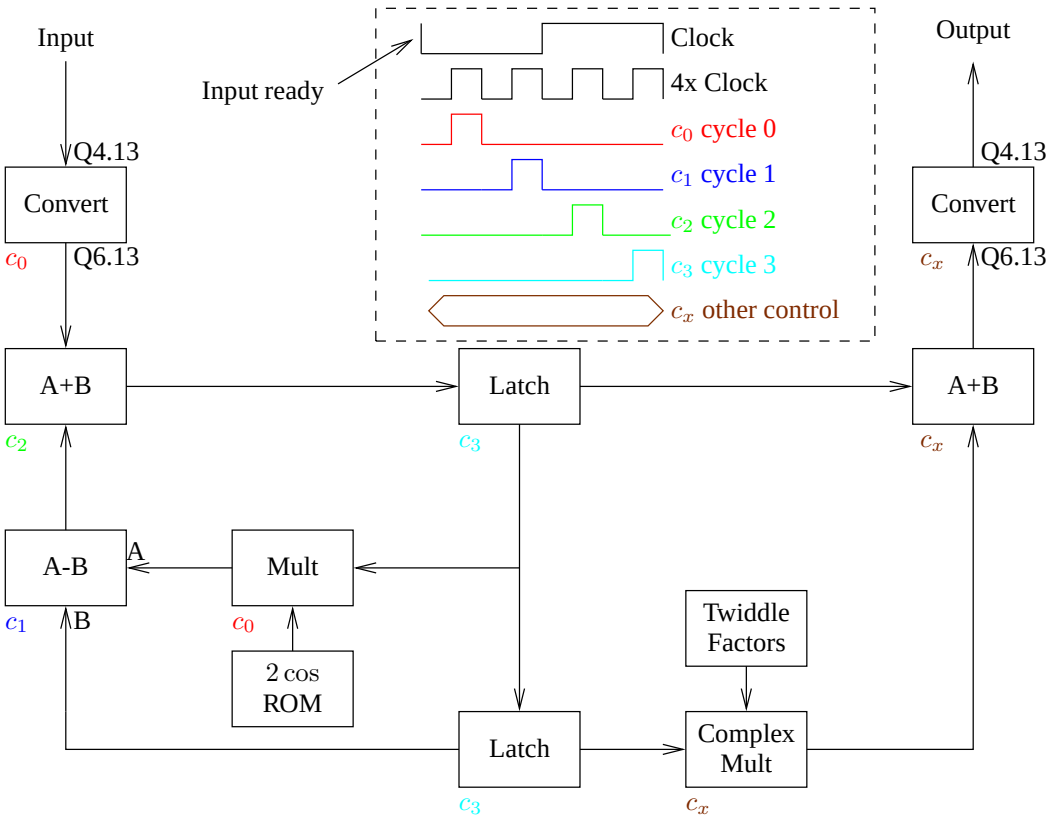

Figure 8.5: *SFFT recombination datapath. The colored numbers show which cycle the block becomes active.*

which is ready for the subtractor in cycle c_1 and later for the adder in cycle c_2. The complex multiplier and adder are controlled by external signals from the SFFT control path. The control path is also responsible for producing data at the Input at the start of c_0 and storing the produced data at the Output.

8.3 Control Path

Design of the control path is based on figure 8.2 on page 73. As discussed the overall execution is controlled by a FSM which executes the SRFFT and activates the SFFT when transformed data is ready in RAM # 1. The SRFFT FSM is thus extended from figure 7.5 on page 65 to figure 8.6, where the last recombination state is added.

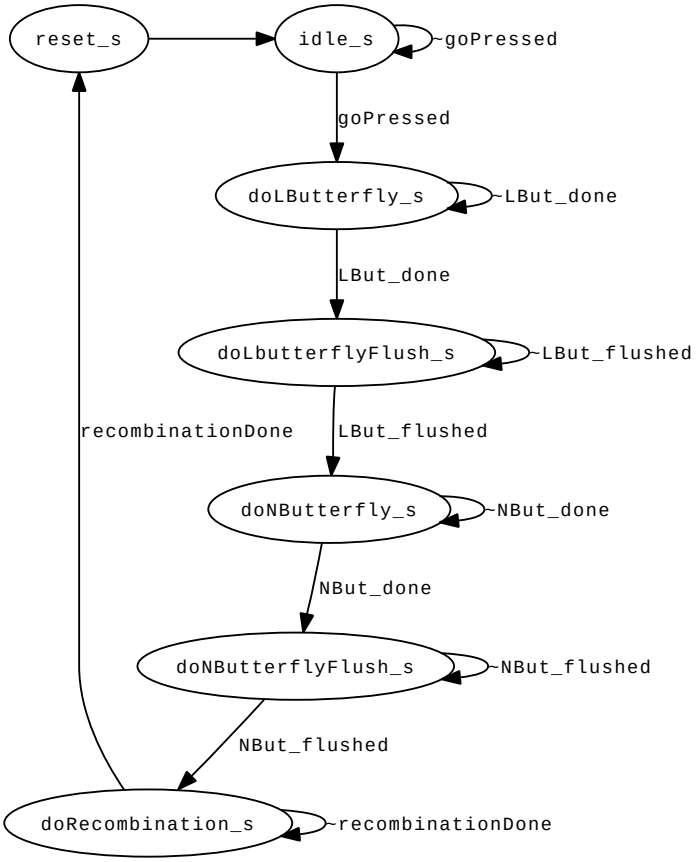

Figure 8.6: *SFFT controlling state machine, extended with an SFFT state (doRecombination_ s) from figure 7.5 on page 65.*

Requirements

Returning to figure 8.2 it is seen that the SFFT datapath writes its output to RAM # 2 with the address controlled from the SFFT control. The SFFT control also determines the source address for data movement from RAM # 1 to the input of the SFFT datapath. The addresses are determined from the values of l to be computed, which are stored in RAM # 3. The highest address in RAM # 3 contains the value of L, the total number of l-values to be computed.

The requirements for the SFFT control path can be summarized as

- Ensure the correct number and values of l is computed
- Issue addresses to RAM # 1 which are then moved to the input of the SFFT datapath
- Issue the address to store the result in RAM # 2
- Control the execution in the datapath

Design Choices

Initially the SFFT is designed to compute one value of $X[k]$ at a time. As RAM # 1 can read and write 4-data values pr. cycle, this SFFT design is to be instantiated 4 times to utilize RAM # 1 to full capability. This instantiation is trivial and the further design only deals with one SFFT control and datapath.

The two main objectives of the control path is that of controlling the datapath and handling the number of l-values computed. These are two distinct tasks which can be defined to run more or less independently. As this is possible the control path is split in two, an upper and a lower control, as shown in figure 8.7. The upper control path handles the values of l while the lower control path handles the datapath. The upper control path determines which values of l to be computed and orders the lower control path to do the computation for each l. A state machine for the upper control path is shown in figure 8.8 on page 81. A state machine for the lower controller, which controls the execution on the datapath and the movement of data between RAM # 1 and the datapath, is seen in figure 8.9 on page 82.

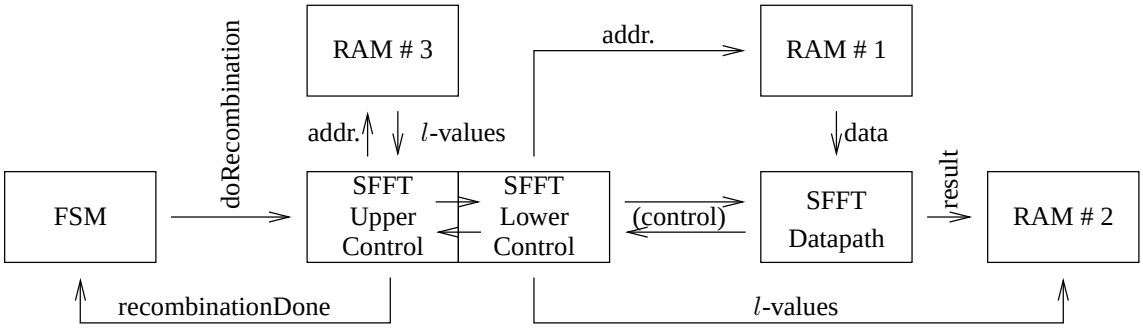

Figure 8.7: *The upper and lower control paths of the SFFT is shown here in conjunction with the datapath and RAM. The upper control path handles the values of l while the lower control path handles the datapath.*

Implementation

The implementation of the two controllers are performed in VHDL as a state machines which manipulate internal variables and determines state transitions based on these. All files regarding the implementation on the accompanying CD [08gr1042, 2008, algorithmStages/02bfgpaSFFT].

Of special note is that the addressing of RAM # 1, which contains the transformed data. As the SRFFT produces the output in bit reversed fashion for each 32 point FFT, the five lowermost bits of the address must be bit reversed. This is handled in VHDL by "wire reordering" as shown in figure 8.10 on page 83. This figure also shows an example state from figure 8.9. The states are implemented as switch-like statements in C.

8.4 Clock Adjustment

A simulation where the end-time is examined shows, that a standard clock of $8MHz$ and a faster clock of $32MHz$ is needed to compute the 250 values of l in the allotted 0.5 ms, as defined in the system specification, section 2.3 on page 15. The simulation is found on the accompanying CD [08gr1042, 2008, testresults/sfft/248samples.zip]. The clocks are generated by the PLL as described in section 7.4 on page 67.

A closer count of cycles from the simulation is shown in table 8.2. The resulting needed cycles is greater than the $4000\,\text{cycles} = 8MHz \cdot 0.5$ ms which is provided in the simulation. There is thus a discrepancy between the observed simulated behavior and the behavior expected from piecewise examination of the states. The requirement from the piecewise state examination is $\frac{4112\,\text{cycles}}{0.5\text{ ms}} = 8.224MHz$, an increase of 2.7% which is not investigated further for this project, as this change is deemed insignificant. All tests and simulations are performed with 8 and $32MHz$ clocks.

State	Cycles	Multiplicity	Result [cycles]
SRFFT	$23 + 11$	Q	1088
fetchData_ s	4	63	252
fetchDataAndCalc_ s"	28	63	1764
calc_ s	5	63	315
createOutput_ s"	3	63	189
done_ s"	2	63	126
idle_ s"	6	63	378
SUM			4112

Table 8.2: *Cycle count for simulation of SFFT. The cycles column shows the number of cycles required for one 32-point SRFFT or from one value of $X[k]$ in the case of the lower controller states. The values for the SRFFT is from 23 L-butterflies and 11 two-point butterflies. The multiplicity is the number of times the state is executed, thus $Q = 32$ for the SRFFT and $\lceil 250/4 \rceil$ for the SFFT states.*

8.5 Summary

The SFFT is implemented by extending the SRFFT with an extra state in which the recombination is performed. This recombination is performed in a dedicated datapath with its own controller and stores its results in the separate RAM # 2.

8.5.1 Hardware Utilization

The hardware utilization by the SFFT is seen in table 8.3 on the next page. This includes the 4 times instantiation of the data- and control paths.

Feature	Available	SRFFT Utilization	SFFT Utilization
Logic Elements	24,624	3,299	8,004
Memory [kBits]	504	66.6	332
Multipliers (18x18)	66	16	66

Table 8.3: *SFFT system hardware utilization compared to available resources and SRFFT design.*

Compared to the SRFFT the recombination data- and control path uses more memory and all of the available multipliers. The high memory usage is due to the instantiation where identical ROMs are created to contain the values of $2 \cdot \cos$ and the twiddle factors. The use of the maximum number of multipliers shows that the current design approaches the limit what is suitable for the chosen FPGA. There is a risk that a number of multipliers has been synthesized from logic elements and memory blocks as lookup-tables, instead of embedded multipliers [Altera, 2007c, p. 5-3]. This could cause suboptimal power performance. To remedy this the instantiation could be performed only 3-times or the MAC solution could be investigated instead. This is left for later investigation.

With the design of the mapping of the SRFFT and SFFT complete. The next part is concerned with evaluation of the achieved results. Initially, all tests are performed and analyzed. Afterwards, a design space exploration is performed to examine the possible changes which could improve the performance of the mappings. Finally the conclusion and further perspectives are discussed.

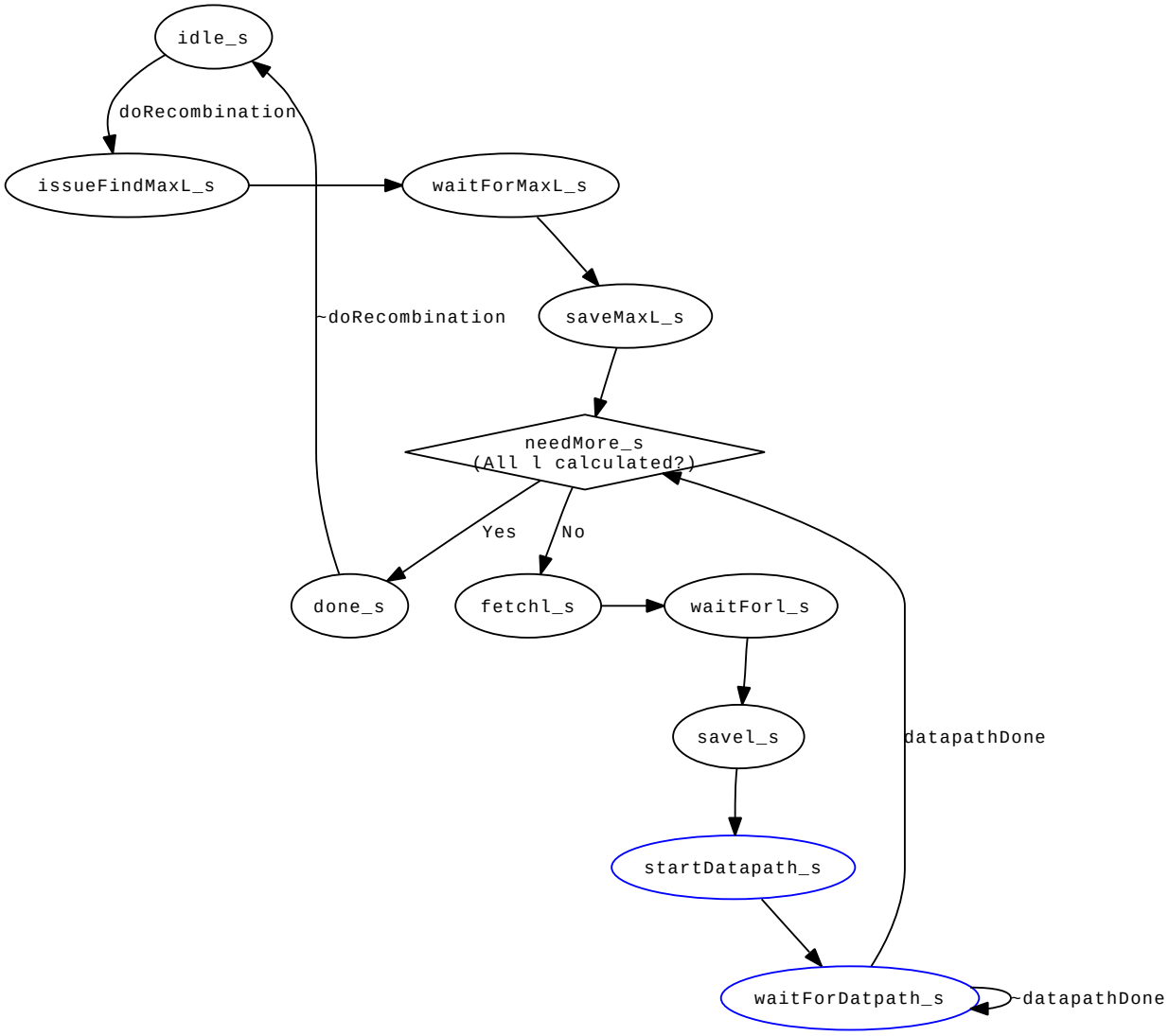

Figure 8.8: *SFFT upper control state machine. Initially the "doRecombination" signal is sent by the controlling FSM (see figure 8.7 on page 78) and the upper controller reads the highest address of RAM # 3 which contains the value of L. Afterwards a value of l is fetched from RAM # 3 and the lower controller (and thus the datapath) is started for each value of l. The lower controller is active in the states marked with blue. When all l is calculated the done_ s is entered and the controlling FSM is signalled with the output "recombinationDone". The upper controller returns to the "idle_ s" when "doRecombination" is set low from the controlling FSM. All variables and counters are reset in "idle_ s".*

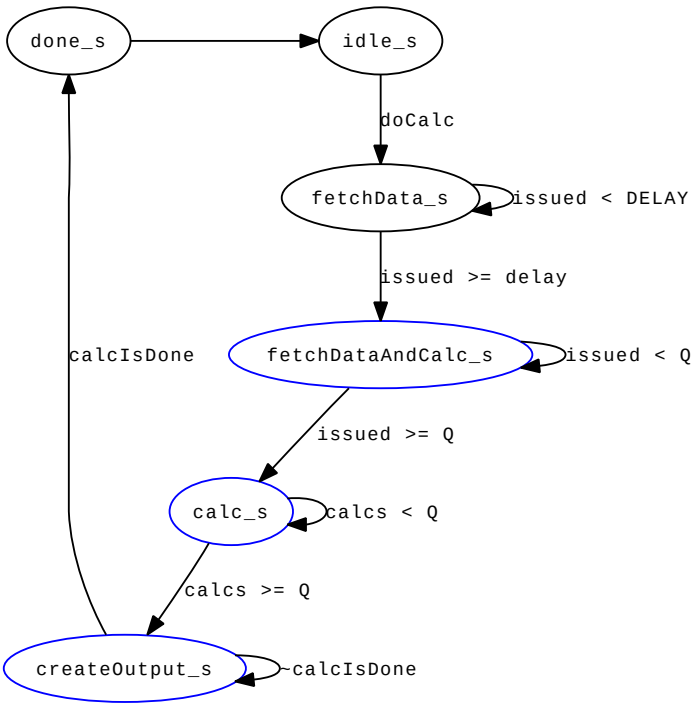

Figure 8.9: *SFFT lower control state machine. Initially the "doCalc" signal is issued from the upper controller along with a value of l (not shown). The controller then issues fetches of data to RAM # 1, see figure 8.7 on page 78. The delay in cycles between the data fetch issues and the arrival of data at the datapath is the constant "DELAY". When data arrives at the datapath, it is enabled in the "fetchDataAndCalc_ s". States shown in blue is where the datapath is active. When "Q" data fetches has been issued the state is changed to "calc_ s" where only calculations are performed. When "Q" calculations has been performed the rightmost part of figure 8.4 on page 74 is activated and the output is created.*

```
1  ------------------------------------------------------------------------
2  when fetchData_s =>                  -- fetch data state --
3    clear <='1';                        -- ensure datapath zeroed
4
5    -- fetch data
6    if fetchesIssued < Q then
7      -- convert variable tempAddr to std-logic-vector
8      bitRevTemp(9 downto 0) := std_logic_vector ( TO_UNSIGNED(tempAddr, ↘
           fetchAddr'Length) ) ;
9
10     -- do bit reverse, fetchAddr(9 downto 0) is mapped to an output and
11     --   address input of RAM #1
12     fetchAddr(9 downto 5) <= bitRevTemp(9 downto 5);
13     fetchAddr(4) <= bitRevTemp(0);
14     fetchAddr(3) <= bitRevTemp(1);
15     fetchAddr(2) <= bitRevTemp(2);
16     fetchAddr(1) <= bitRevTemp(3);
17     fetchAddr(0) <= bitRevTemp(4);
18
19     doFetchAddr <= '1';               -- enable read from RAM #1
20     fetchesIssued := fetchesIssued + 1; -- keep account of fetches
21     tempAddr := tempAddr - Q;         -- next address to get
22   else
23     fetchesIssued := fetchesIssued; -- keep value
24     tempAddr := tempAddr;           -- keep value
25     doFetchAddr <= '0';              -- disable RAM #1 read
26     fetchAddr(9 downto 0) <= "0000000000"; -- zero read address
27   end if;
28
29   -- next state logic
30   if (fetchesIssued >= DATAPATHDELAY) then
31     next_state <= fetchDataAndCalc_s; -- change state
32     enable <= '1';                    -- enable datapath
33     clear <= '0';                     -- do not clear datapath
34   else
35     next_state <= fetchData_s;        -- continue as usual
36   end if;
37 ------------------------------------------------------------------------
38 when fetchDataAndCalc_s =>          -- fetch data and calc state --
39   -- ... (not included in this example)
```

Figure 8.10: *VHDL code for an example state in the SFFT lower controller. First it is checked if the needed number of data fetches has been issued. If not, the variable "tempAddr" is converted to a standard logic vector format and then assigned bit-reversed to the address output which is connected to RAM # 1. Otherwise reading from RAM is disabled and the output address is zeroed. The next state logic examines if data is ready for the datapath and activates the datapath. If the data is not ready the state is kept and another read from RAM # 1 is issued in the next cycle.*

Part III

Evaluation

This final part is concerned with the test and evaluation of the FFT designs implemented in the previous part.

First, the two FFT systems are simulated and tested to verify the functionality and evaluate the power consumption of the systems. Next, a design space exploration is performed, where the details of the power simulations are used to determine possible design changes, which may achieve improved system power performance. Finally, the project conclusion recapitulate the project course and achieved results.

Contents

Chapter 9

Test

The test chapter is concerned with a functional verification of the SRFFT and SFFT systems. Furthermore, the simulations and measurements of system power consumtion for the two implemententations are documented and are evaluated. A deeper examination of the results to point out possible system improvements are found in the following chapter.

9.1 Split Radix FFT

The test of the Split Radix FFT is concerned with first of all to verify the functionality of the algorithm implementation on the Cyclone III System. This verification is done by estimation of the Signal to Quantization Noise Ratio (SQNR) for the system. Next the power consumption is evaluated by simulation, using the available tools and measurement options described in chapter 5.

The implementation for the FPGA is found on the accompanying CD [08gr1042, 2008, algorithmStages/02afgpaSRFFT].

9.1.1 Functional Verification

The functionality of the SRFFT implementation is verified by comparing the results calculated by the system to the results calculated by the C-model described in section 3.3. This allows estimation of a SQNR for the SRFFT implementation, which describes the obtained precision of the system. SQNR is given as:

$$SQNR = 10log_{10}\frac{\mu\left(|X_{ref}|^2\right)}{\mu\left(|X_{ref} - X_{sys}|^2\right)} \tag{9.1}$$

where $\mu()$ denotes the mean operator, X_{ref} is the C-model reference output, calculated using 32bit floating point data types, and X_{sys} is the SRFFT system output calculated using the Q4.13 signed integer data type defined in section 6.3 on page 50. As stated in equation (9.1), the quantization noise power is found from mean of differences between X_{ref} and X_{sys}, thus the 32 bit floating point reference is regarded as an exact reference.

A sample of 36 system results has been taken from the SignalTap analyzer, and the corresponding values have been taken from the C-model reference to calculate the SQNR for the SRFFT:

$$SQNR_{SRFFT} = 10log_{10}\left(\frac{1}{3.4398 \cdot 10^{-6}}\right) \quad (9.2)$$

$$= 54.63dB \quad (9.3)$$

The sample used for estimating $SQNR_{SRFFT}$ is found on the accompanying CD [08gr1042, 2008, testresults/srfft/veriSRFFT.mat]. From the SQNR, a number of effective decimal bits may be calculated:

$$SQNR = 20log_{10}(2^{b_{eff}}) \Leftrightarrow \quad (9.4)$$

$$b_{eff} = log_2\left(10^{\frac{SQNR}{20}}\right)$$

$$= 9.07 \approx 9bit \quad (9.5)$$

Thus the last four decimal bits of the system output contain the noise floor added by computing the implemented SRFFT algorithm.

9.1.2 Power Simulations

The power consumption of the SRFFT design has been simulated using the method described in chapter 5.

Full Clock System

Based on simulation results of a 2 ms system run, the total power dissipated by the system is estimated to 96.95 mW. In order to enable comparison with the power measurements on the system, table 9.1 gathers the simulated currents drawn from the FPGA core power supplies. V_{CCINT} supplies the internal logic, memories and embedded computation units, V_{CCIO} suppplies the I/O interfaces, and V_{CCA} and V_{CCD} supply the analog and digital part of the embedded PLLs.

The Cyclone III board allows for measuring the current drawn from the 1.2 V supply, feeding V_{CCINT} and V_{CCD} on the FPGA. With a supply voltage of 1.2 V, the power consumption from this supply is:

$$P_{sim_{full}} = U \cdot I$$

$$= 1.2 \cdot (I_{V_{CCINT}} + I_{V_{CCD}})$$

$$= 1.2 \cdot (13.28 + 14.24) = 33.0mW \quad (9.6)$$

Reduced Clock System

The results of a second simulation, where the system clocks are reduced to approximately fit algorithm completion time to the available 0.5 ms, are gathered in table 9.2. The total system power consumption is simulated to 89.75 mW.

Supply	Total [mA] (I)	Dynamic [mA]	Static [mA]
V_{CCINT} [1.2 V]	13.28	9.04	4.24
V_{CCIO} [2.5 V / 3.3 V]	4.18	2.59	1.59
V_{CCA} [2.5 V]	22.76	2.19	20.57
V_{CCD} [1.2 V]	14.24	5.45	8.79

Table 9.1: *Power simulation results for currents drawn from FPGA voltage supplies with system running at 50 Mhz/12.5Mhz.*

Supply	Total [mA]	Dynamic [mA]	Static [mA]
V_{CCINT} [1.2 V]	8.69	4.58	4.12
V_{CCIO} [2.5 V / 3.3 V]	2.46	0.87	1.59
V_{CCA} [2.5 V]	22.74	2.19	20.55
V_{CCD} [1.2 V]	14.08	5.33	8.75

Table 9.2: *Power simulation results for currents drawn from FPGA voltage supplies with system running at 16.7 MHz / 4.17 MHz.*

The power consumption from the 1.2 V supply is then:

$$\begin{aligned} P_{sim} &= U \cdot I \\ &= 1.2 \cdot (8.69 + 14.08) = 27.3mW \end{aligned} \tag{9.7}$$

9.1.3 Power Measurements

The Split Radix FFT calculates the entire 1024 points of the Fast Fourier Transform, thus only one test case is necessary. In order to estimate the mean power consumption of the SRFFT algorithm, measurements of the power consumption are needed for the FPGA design in the idle and active states, respectively, and the time spent in the active state.

As described in chapter 5 the Cyclone III kit allows for measuring the power consumption of the FPGA core using a 0.01Ω sense resistor on the 1.2 V power supply [Altera, 2007b, pp. 4-1 - 4-3]. In order to measure the power consumption in each state, the state machine described in figure 7.5 on page 65 is controlled by the board pushbuttons to stay in idle state or cycle the active states continuously. Finally, the output diode $LED0$ has been connected to the state machine and is switched on and off, when the system is in a active or idle state, respectively. Using a DC voltmeter across jumper $JP6$ the voltage across the sense resistor is measured. The time spent in an active state is measured by recording the voltage across the $LED0$ diode using a oscilloscope.

Full Clock System

For the system running at full system clock speed, 50 MHz / 12.5 MHz, the measured results are shown in table 9.3

State	Voltage across $JP6$ [mVDC]	Time [mus]
Idle	0.315	-
Active	0.449	156.4

Table 9.3: *Measurement Results of Power measurements for SRFFT. The voltages are measured across a 0.01Ω sense resistor. placed in series with the FPGA core voltage supply. The time for the active state is the completion time of the FFT algorithm.*

Using the results in table 9.3 the mean power consumption of the SRFFT algorithm for the scenario defined in section 2.3 on page 15 where one FFT needs to be calculated every 2 ms. The mean power consumption may then be found as:

$$\begin{aligned} P_{avg_{SRFFT}} &= \frac{1}{2ms}(\left(t_{active} \cdot U_{FPGA} \cdot \frac{U_{active}}{R_{JP6}}\right) + \ldots \\ &\quad \left((2ms - t_{active}) \cdot U_{FPGA} \cdot \frac{U_{idle}}{R_{JP6}}\right)) \qquad (9.8) \\ &= \frac{1}{2ms}(\left(156.4\mu s \cdot 1.2V \cdot \frac{0.449mV}{0.01\Omega} +\right) + \ldots \\ &\quad \left((2ms - 156.4\mu s) \cdot 1.2V \cdot \frac{0.313}{0.01\Omega}\right)) \qquad (9.9) \\ &= \frac{1}{2ms}(156.4\mu s \cdot 53.9mW + (2ms - 156.4\mu s) \cdot 37.6mW) \qquad (9.10) \\ &= 38.8mW \qquad (9.11) \end{aligned}$$

Reduced Clock System

Reducing the clock by one third to 16.67 MHz / 4.17 MHz to utilize the 0.5 ms available for calculating the SRFFT, results in the voltages and time gathered in table 9.4.

State	Voltage across $JP6$ [mVDC]	Time [μs]
Idle	0.241	-
Active	0.290	470,0

Table 9.4: *Measurement Results of Power measurements for SRFFT with clock reduced to 16.67 MHz / 4.17 MHz.*

Using the results in table 9.4 the mean power consumption over a 2 ms interval for the reduced clock SRFFT is found to be:

$$\begin{aligned} P_{avg_{SRFFT}} &= (\frac{1}{2ms}470\mu s \cdot 28.9mW + \ldots \qquad (9.12) \\ &\quad (2ms - 470\mu s) \cdot 34.8mW) \\ &= 33.4mW \qquad (9.13) \end{aligned}$$

9.1.4 Discussion

The results presented above are summed up in table 9.5.

	Simulated [mW]	Measured [mW]
Full Clock	33.0 mW	38.8
Reduced Clock	27.3 mW	33.4

Table 9.5: *Simulation and measurement results summary.*

Comparing the simulated power consumption with the measured results it is seen that the estimated mean power consumption of the SRFFT based on measurements is 5.8 mW higher for the full clock system and 6.1 mW higher for the reduced clock system compared to the simulation estimate. The possible source for this difference, which shows also to be present when testing the SFFT, is discussed in section 9.2.4.

Although, the simulation and measurement results deviate, they are still reasonably coherent, and the details provided from the simulation reports, will be used to evaluate the system design. One main point to be made is that a relatively high portion of the power consumed is used to drive the PLL circuit used to generate the multiple system clocks. For the 1.2 V supply in the full clock system, 14.24 mA of 27.52 mA (51.7 %) are drawn by at V_{CCD} as shown in table 9.1. When reducing the system clock the current drawn at V_{CCINT} is reduced, and the consumption by the PLL circuit becomes even more significant (61.8 %).

Thus, a design with only one clock, and no PLLs activated, may reduce system power consumption. This would require a redesign of the counter reset procedures in the address generators, and would probably increase execution time, since the datapath would need to be halted during these reset procedures.

A second issue is that of board system clock frequency. From both simulations and measurements it is clear that there are significant power consumption reductions to be achieved when reducing the system clock to fit timing constraints. By reducing the clock by a factor of 3, the power consumption is reduced by approximately 6 mW or 20 %. In addition, the simulations show that this reduction is almost entirely achieved by reducing the current drawn at V_{CCINT} by 4.6 mA (5.52 mW).

As seen in equation (9.3) the SQNR of both algorithms is approximately 54dB which leaves 9 decimal bits effectively representing the output signal. This representation was chosen in section 6.3 on page 50 as a tradeoff between reducing risk of overflow, decimal precision and power usage. Whether or not this signal precision and thus number representation is acceptable will depend on actual system requirements for SQNR and the input data SNR. This is important from a power consumption point of view as the data representation effectively scales the computational units of the data path, and the size of memories for storing data, which results in changed power usage.

This problem is more significant for decimal bits than integer bits, since bits only representing noise will toggle signal nodes in the system generating dynamic power consumption, whereas overhead integer bit will not be used and thus, ideally, not contribute to system dynamic power consumption.

Based on the simulations and measurements of the SRFFT implementation some power optimization has been achieved and potential for significant reductions in the power consumption have been discussed. The results obtained in the test of the SRFFT with a reduced system clock, provide a benchmark for power consumption in the SRFFT algorithm, where the full DFT is calculated. This benchmark is used to compare results of the SFFT implementation where only the subset of subcarriers of interest are calculated.

9.2 Sørensen FFT

The method for simulating and testing the SFFT in analogue to the approach used for testing the SRFFT in the previous section. In this section the results of these simulations and tests, performed with the SFFT calculating a range of of subcarriers, are presented.

The implementation for the FPGA is found on the accompanying CD [08gr1042, 2008, algorithmStages/02bfgpaSFFT].

9.2.1 Functional Verification

The results calculated by the SFFT in simulation and on the FPGA is shown in figure 9.1 on the facing page. This figure contains 56 samples of l for real and imaginary values.

The values of the FPGA are measured using the SignalTap-tool provided in Quartus II. The compilation has been performed without fitter- and power optimizations as these renders the design numerically inaccurate. The power simulations discussed in the next section are based on compilation with fitter- and power optimizations enabled. The values of the simulation are from a functional simulation whereas the simulation provided with the power estimation is a timing simulation. The measured values and simulation is found on the accompanying CD [08gr1042, 2008, testresults/sfft/sfftVerification.zip].

The data is split into two ranges, each targeting either the positive $\frac{1}{\sqrt{2}}$ or negative $-\frac{1}{\sqrt{2}}$ constellation according to expected result. This produces figure 9.2 on page 94 for the upper constellation point and 9.3 on page 95 for the lower constellation point.

An overview of the means and variances is provided in table 9.6. The table and previous figures 9.2 and 9.3 shows that the functional simulation produces accurate results, while the FPGA implementation does not produce satisfying results. It is furthermore seen in the figures that the FPGA implementation produces results which would be falsely classified.

	Mean	Variance
Simulated Upper	0.7077	$1.5594 \cdot 10^{-6}$
Simulated Lower	-0.7078	$4.1619 \cdot 10^{-6}$
FPGA Upper	0.7010	0.2112
FPGA Lower	-0.5555	0.3601

Table 9.6: *Mean and variances for the SFFT simulation and implementation.*

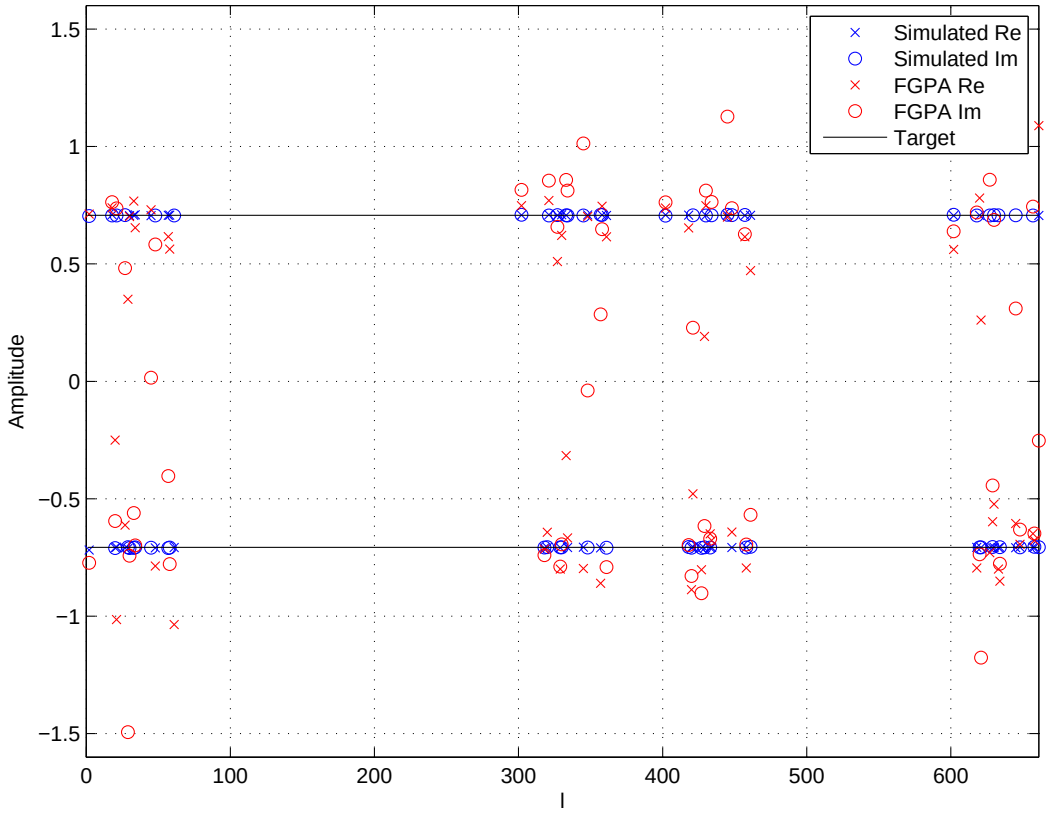

Figure 9.1: *The output of the SFFT plotted for simulated and FPGA measured values.*

This is a somewhat expected outcome as only the functional simulation shows accurate results. A timing simulation shows more inaccurate results, as it accounts for timing problems in the design. One cause of the inaccuracy could be a fragile design in the SFFT control- or datapath which makes it susceptible to changes in signal timing etc. incurred by the fitting to the FPGA. This is further supported by the observation that enabling improved fitting- and power optimization effort for compilation to the FPGA produces even worse results. This is a design error which would have to be corrected in future work.

Examining the simulation it is seen that the SFFT functionally performs as expected. This means that the implementation can be used for power measurements, even though the results are not correct.

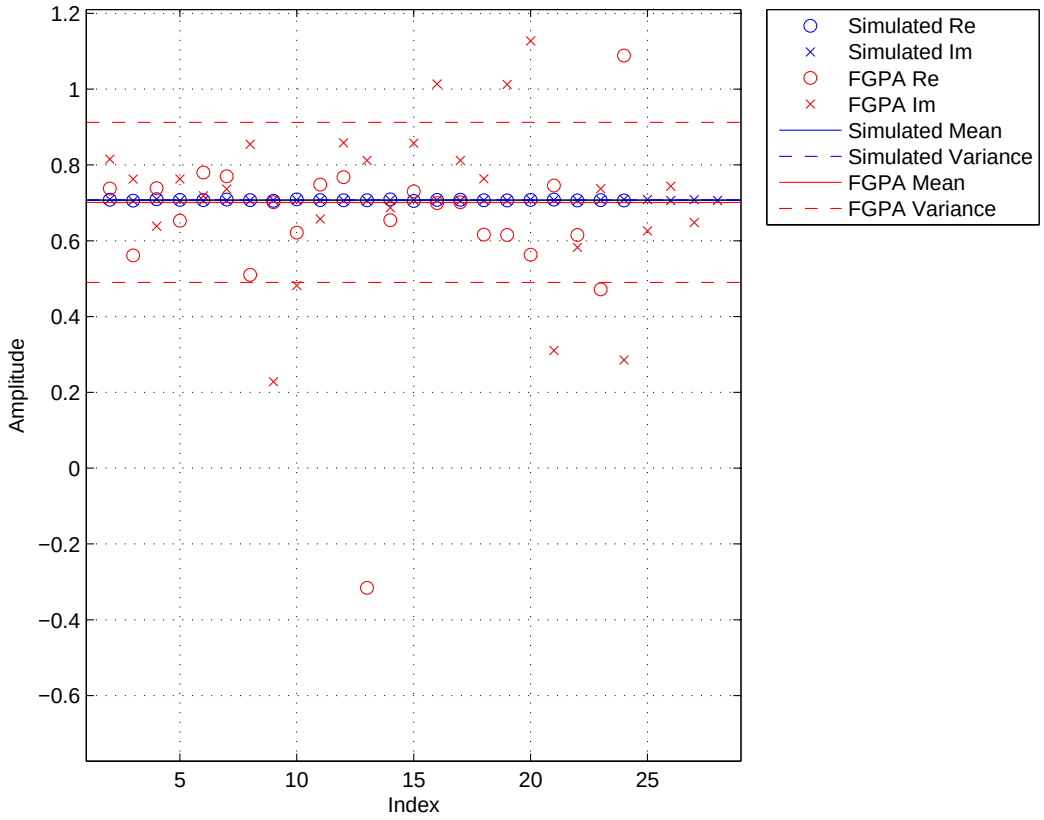

Figure 9.2: *Simulated and implemented SFFT output for upper constellation point. The mean and variance of the FPGA measurements are shown, while the simulated mean and variance of the simulation are indiscernible on the figure.*

A SQNR is defined in section 9.1.1 and is calculated here for the simulation only

$$SQNR_{SFFT,sim} = 10 \cdot \log_{10} \frac{\mu\left(|X_{ref}|^2\right)}{\mu\left(|X_{ref} - X_{sys}|^2\right)} = 10 \cdot \log_{10}\left(\frac{1}{3.8016 \cdot 10^{-6}}\right) \quad (9.14)$$

$$= 54.2 \text{ dB} \quad (9.15)$$

which is approximately the same as found in (9.3) on page 88.

9.2.2 Power Simulations

The simulated mean power consumptions of the SFFT algorithm is shown in table 9.7 for a range of number of demodulated subcarriers.

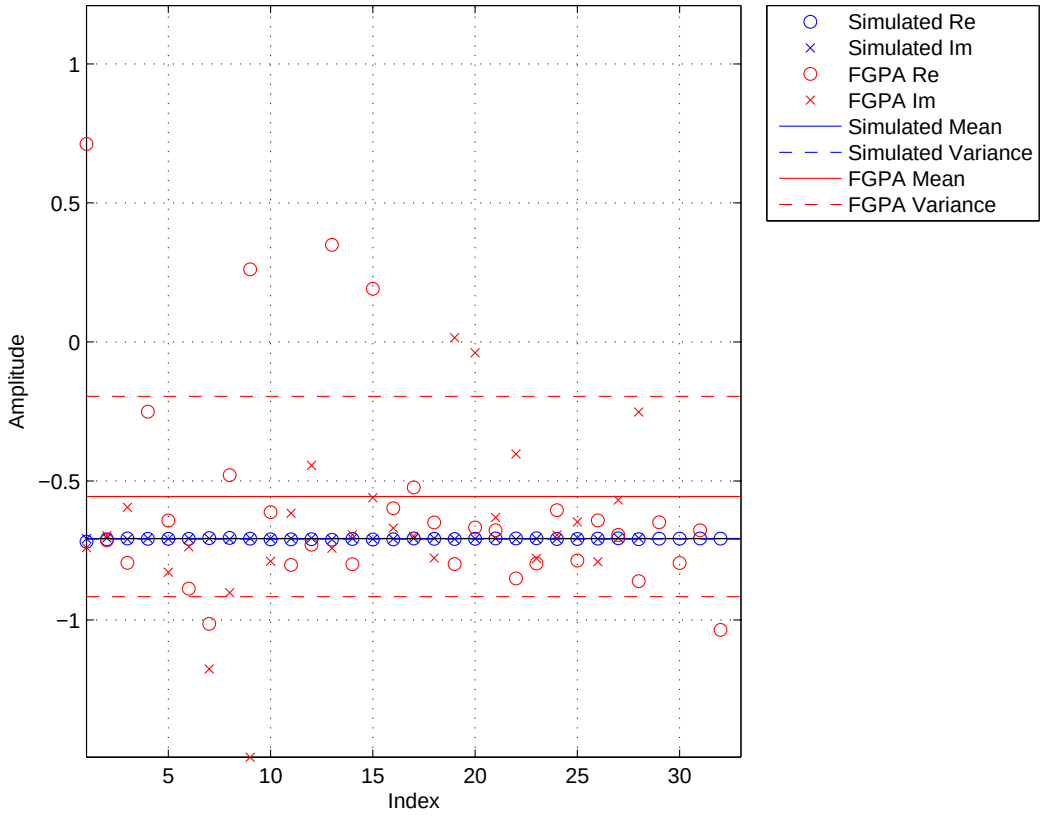

Figure 9.3: *Simulated and implemented SFFT output for lower constellation point. The mean and variance of the FPGA measurements are shown, while the simulated mean and variance of the simulation are indiscernible on the figure.*

9.2.3 Power Measurements

For the same range of demodulated subcarriers used in the simulations above, measurements of voltages across R_s have been measured in active and idle states, as well as calculation times. These results are gathered in the first three columns of table 9.8. Based on these measurements, the last three columns contain the calculated mean, idle and active power consumptions.

9.2.4 Discussion

The power consumption results gathered in section 9.1 for the SRFFT and this section for the SFFT are plotted together in figure 9.4.

n	P_{mean} [mW]
4	39.29
20	39.60
60	39.79
80	39.96
100	40.04
120	40.27
160	40.51
200	40.82
248	41.29

Table 9.7: *Simulation results for SFFT. Mean power consumption, P_{mean}, simulated for each number demodulated subcarriers, n.*

n	$V_{R_s\_idle}$	$V_{R_s\_active}$	t_{active}	P_{mean}	P_{idle}	P_{active}
4	0.375	0.473	123	45.7	45.0	56.7
20	0.376	0.477	143	45.9	45.1	57.2
60	0.373	0.481	210	46.1	44.7	57.7
80	0.375	0.483	240	46.5	45.0	57.9
100	0.374	0.487	267	46.6	44.8	58.4
120	0.371	0.485	300	46.5	44.5	58.2
160	0.373	0.489	363	47.2	44.7	58.6
200	0.373	0.490	423	47.7	44.7	58.8
248	0.375	0.490	493	48.4	45.0	58.8
unit	[mV]	[mV]	[μs]	[mW]	[mW]	[mW]

Table 9.8: *Measurements results for SFFT for a set of different number of demodulated subcarriers, n. Based on idle and active voltages across R_s and calculation times, mean, idle and active power consumptions have been calculated for each test.*

First of all, it is seen that the Powerplay analyzer is underestimating the power consumption of the SFFT as well. The mean difference between the simulation and the measured results is:

$$\begin{aligned} \Delta_P &= \mu(P_{mean_{meas}}) - \mu(P_{mean_{sim}}) \\ &= 46.79mW - 40.18mW \\ &= 6.61mW \end{aligned} \tag{9.16}$$

This difference is proportionally similar to the difference between simulations and measurements on the SRFFT system and as seen in figure 9.4 the difference is reasonably constant. This offset behavior of the difference could indicate that parameters for estimating the power consumption of the system are inaccurate. First of all, it is worth noticing that the power models for the Cyclone III device family using in the Powerplay analyzer are preliminary Altera [2008b]. This factor introduce some inaccurate power model parameters which may show in the simulation results.

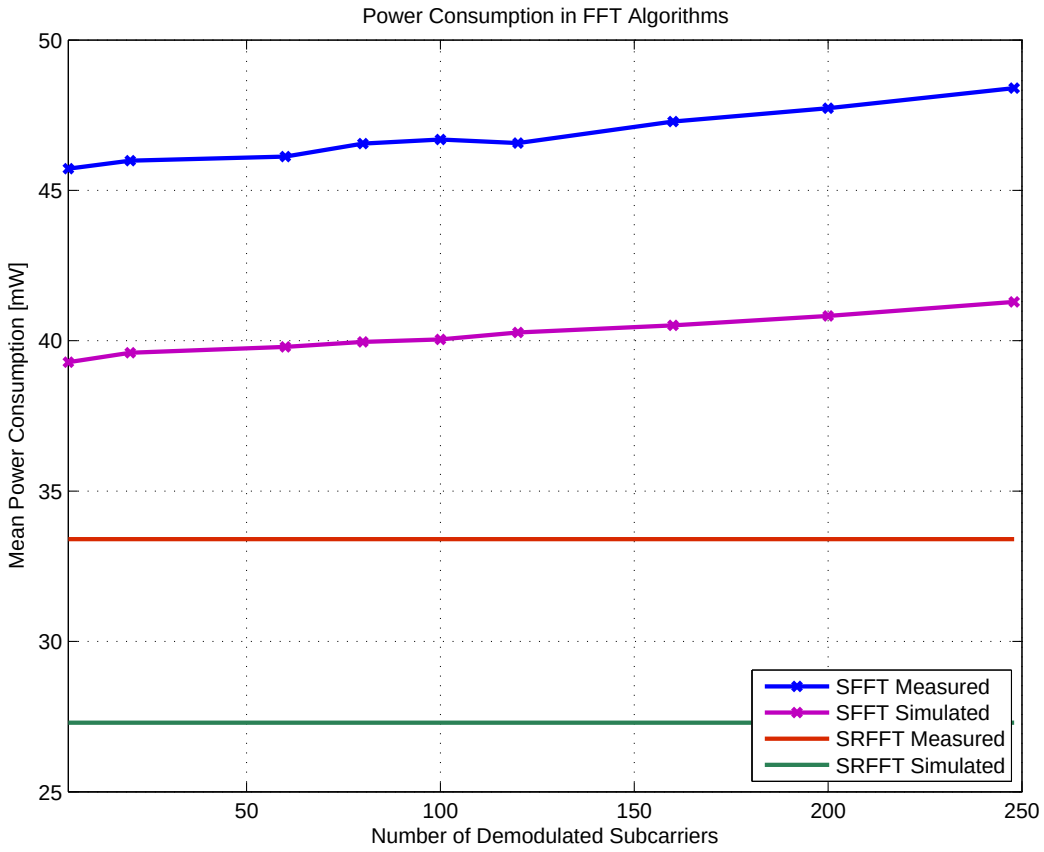

Figure 9.4: *Results of measurements and simulations of SRFFT and SFFT power consumption*

Furthermore, the settings of ambient temperature and cooling characteristic may be tuned by measurements of both the device environment (e.g. air temperature and humidity) and the device temperature changes during operation. Such measurements would add further confidence to the to the simulation results since ambient temperature a the ability to lead heat away from the device contributes to the power consumption characteristics of the system.

9.3 Summary

Based on the results gathered in figure 9.4 it would seem that the SFFT does not perform better than the SRFFT for any number of subcarriers demodulated, mainly since the idle power consumption of the SFFT is significantly higher than mean power consumption of the SRFFT.

However, it is clear from the results in equation (9.13) for the SRFFT and table 9.8 for the SFFT that the measured power consumption in idle state constitute a significant percentage of the total power consumption of the systems. The effects of this idle power consumption is discussed in further detail in the next chapter, along with a more detailed design space exploration into

which system optimizations that may accomplish improved power performance based on the simulation and measurement reports.

Chapter 10

Design Space Exploration

With the algorithms implemented and tested on the FPGA, the design process shown in figure 1.3 on page 4 has been completed. Depending on system requirements and tested performance, a further iteration of the design process can be performed to improve system performance. For this project a deeper examination of the power-wise performance of the implementation is desired. This is done to examine the implementation for power-wise flaws and to gain a better understanding of which low-level factors to manipulate to produce power optimized designs.

Complete exploration of the design space is for practical purposes nearly impossible, as there are many combinations of all possible design changes. For this project a focus area of power consumption is chosen and the implemented design is analyzed for possible design improvements which may reduce power consumption.

10.1 Basis for Analysis

The analysis is performed as if a further iteration of the mapping process of the FFT algorithms onto the FPGA archirtecture was to be carried out, and thus seeks to clarify which areas to focus on for improvements. The analysis itself is based on the simulated results, where the Power-analyzer tool provides in-depth information about circuit performance.

For this analysis only the SFFT is examined. This can be done as the SFFT performs a SRFTT as part of its execution. Both design are thus examined, but not directly compared.

The SFFT is set to calculate $L = 248$ points as this will expose the biggest difference between the SRFFT and SFFT. This is caused by the simulation output being an average of the entire simulation where if L is low the SFFT datapath extension would appear insignificant compared to the SRFFT. Finally the simulation length is set to 2 ms to keep evaluation the prerequisites consistent with the measurements and simulations performed in the previous chapter.

Another fact of interest is that the simulation includes the total power usage whereas the system measurements only includes FPGA core power. Thus the simulation results as a whole cannot be directly compared to the measured results. The simulations divide the total power consumption into static-, dynamic and routing power. These concepts, extensively used in the following discussion, are defined in section 5.1.1 on page 38.

The remainder of this chapter is concerned with an initial examination of the overall simulation summary followed by a top down examination of the implementation, where the design is split into different functional groups to determine where the most significant changes can be made. At last the SFFT implementation are examined in depth and a complete chapter summary is given.

10.2 Simulation Summary

An excerpt from the simulation results summary is given in table 10.1. It is seen that the static power usage is actually bigger than the dynamic power usage. This means that the idle power-usage represent will the main part of the active power-usage. It must be noted that idle-state power usage cannot be directly equated with static power usage, but for cases where the clock signal is nullified, a large part of the idle-power used is static power usage.

This could be caused by the large area spent on SFFT control and datapath, which is idle while the SRFFT is running. However, by comparing the static power consumption of the SFFT and SRFFT in table 10.1, it is clear that these are almost identical. Difference in area utilization thus seems to have a small effect on the leakage and sub threshold currents in the FPGA system.

Group	Power Usage SFFT [mW]	Power Usage SRFFT [mW]
Dynamic	28.45	16.24
Static	66.39	66.33
I/O	9.57	7.57
Total	104.4	90.14

Table 10.1: *Summary of the power analysis. The usage is divided into dynamic, static and I/O power usage.*

The unwanted idle-power usage could be removed by introducing a power-off state for the SRFFT and SFFT parts. Thus the circuits is completely turned off while inactive, instead of just having their clocks disabled as in the current design.

If a power-off state were introduced, the calculations, based on the results from the experiments, of power-usage, see equation (5.9) on page 43, would yield the results of figure 10.1 on the next page, which shows the SFFT method actually is feasible for approximately $L < 100$. This is quite contrary to the results of the SFFT test (chapter 9.2 on page 92) which shows that the SFFT has an infeasible performance compared to the SRFFT. A power-off state would thus be a highly-prioritized improvement in the further design and make the computational complexity as a performance measure on par with the power usage.

10.3 Performance by Hierarchy

Having examined the general summary in the previous section, the individual power usages of the design partitions in the system are of interest in this section. The power usage divided into dy-

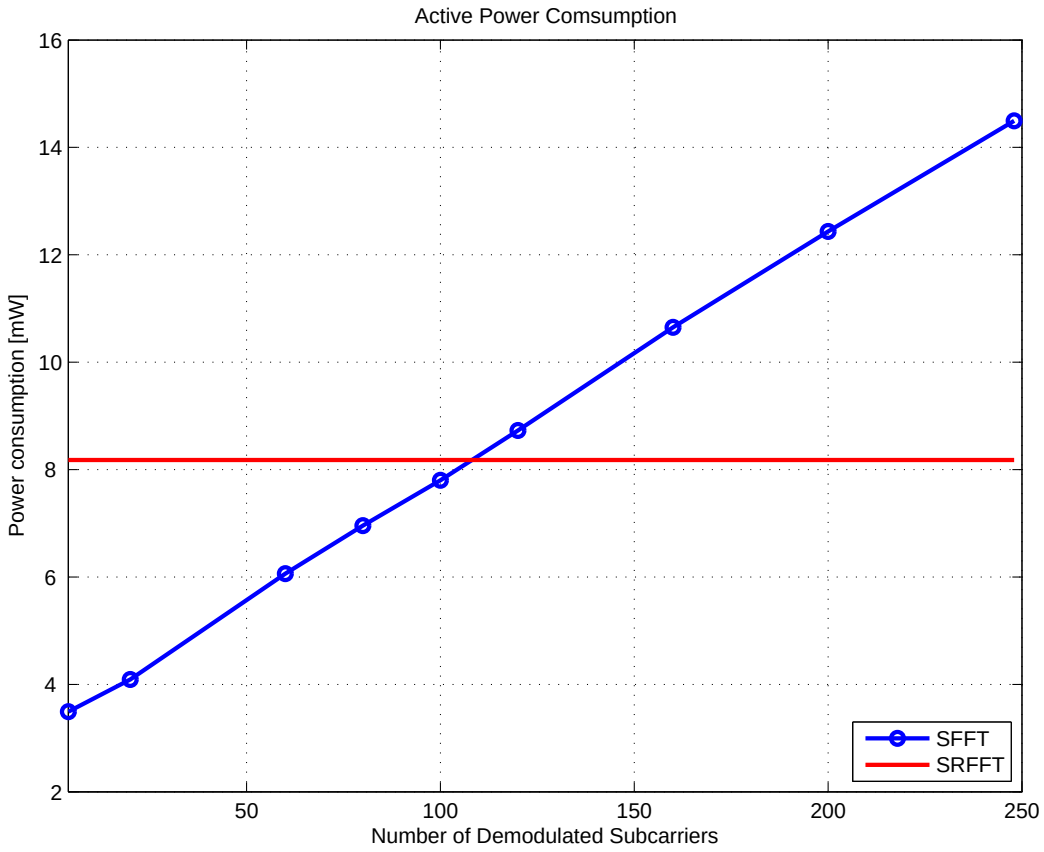

Figure 10.1: *Power consumption for FFT algorithms assuming zero idle power consumption. The data is from the results of the experiment as the simulation does not produce output in which idle- and active periods can be discerned.*

namic power and routing power is shown in table 10.2 on the next page. Static power usage is not included in these simulation results, but are instead included in table 10.1 in the previous section.

Each of the entities in table 10.2 are discussed in the following. All but the "Miscellaneous" entries are defined in the design chapters and shown in figure 8.2 on page 73.

The miscellaneous entry covers the connecting of design entities and simulation-taps which erroneously are connected to output pins. These erroneous pin connections covers at least 4.23mW which can be seen in the simulation and should be removed in a design revision.

A positive discovery is that the power consumption of the SRFFT and RAM # 1 entries are low, relative to the other blocks in the design. Together they consume approximately 14% of the total power spent. It is thus not in these design entries that major savings are found, instead the analysis should focus on the other parts.

As a side note, the top level controlling state machine has a negligible power usage. It is

Group	Total [mW]	Dynamic [mW]	Routing [mW]
RAM # 1	1.91	1.35	0.56
Clock Generator	12.98	6.43	6.55
SRFFT	2.63	2.51	0.12
SFFT	10.89	10.01	0.88
Miscellaneous	4.30	4.30	0.00
Total	32.71	24.60	8.11

Table 10.2: *Power usage by hierarchy*

not shown in table 10.2 but is listed as 0.00mW in the simulation. This should be compared to the power usage of the softcore processor NIOS-II which is an alternative control method, but probably fueatures higher power usage due to functionality overhead resulting from the reconfigurability of the NIOS-II. The NIOS-II softcore processor is not tested or investigated further in this project.

The two remaining entries are the clock generating circuit and the SFFT. The SFFT is discussed in section 10.4. The final entry is the clock generation circuit which consists of a PLL. It is seen that this entry has the largest power consumption of the design entities. The power usage is split evenly between dynamic and routing power, which is atypical compared to the other design entries. This could be caused by the PLL being implemented in dedicated hardware in a corner of the FPGA device, see figure 4.1 on page 31, and the output driving entire global clock networks covering the entire device. Changing the structure of the clock networks to reduce the network distribution across the device may prove difficult. This is due to the full utilization of the embedded multipliers in the SFFT design results in a design scattered across the device, which too gives rise to generally long signal lines between utilized device entities and thus increased dynamic power consumption.

Another approach to possibly decrease the power consumption of the clock generation task is to replace the external $50MHz$ clock crystal with a crystal which does not require clock-division, or implement the clock division by counters instead of in a PLL.

10.4 Examination of the SFFT Implementation

The second highest power consumption of table 10.2 is the SFFT block which primarily consists of four parallel SFFT datapaths. The multiple instantiations of the SFFT datapaths represents a tradeoff between area and clock-frequency. The instantiation allows for a slower overall clock frequency for the entire design (SRFFT and others included), which yields a lower dynamic power consumption. On the contrary the instantiation uses more area which should results in a higher static power usage and thus idle power usage. The multiplicity and power usage for the SFFT design entities are shown in table 10.3 on the next page.

Comparing the entries it is seen that the datapath uses a majority of the power for the SFFT. ROM # 3 which contains the values of l and L has a low power expenditure while the control-

Group	Multiplicity	Power Usage [mW]
Datapath	4	2.61
Control	4	0.09
8×18bit delay	1	0.07
ROM # 3	4	0.01

Table 10.3: *SFFT power usage by hierarchy and multiplicity. The power usage column is for one data- and control path, as shown in figure 8.7 on page 78. Values where the multiplicity is greater than one are average values.*

paths together uses 0.36mW. A small improvement could be gained by consolidating the control paths and improving the scheduling to e.g. avoid an 8×18 bit delay introduced during mapping to neutralize an one cycle latency of the controlling statie machine. ROM # 3 could be merged to one ROM bank, but this should only be done in conjunction with the merger of the control paths, as the improvement from joining the ROMs are insignificant.

Removal of 2 or 3 instances of the datapath has more complex effects. The overall clock-frequency would have to be increased which would give rise to a higher dynamic power usage. This must be compared to the gain achieved from instantiation of fewer SFFT datapaths. This comparison must also include the possible power improvements in the individual datapaths, which will be discussed next.

An overview of the power usage of a single SFFT datapath is seen in table 10.4.

Entity	Total [mW]	Dynamic [mW]	Routing [mW]
ROM $(2 \cdot cos)$	1.21	1.20	0.00
ROM Twiddle Real	0.60	0.60	0.00
ROM Twiddle Imag	0.30	0.30	0.00
Mult Complex	0.10	0.10	0.01
Mult 6Q13 Real	0.08	0.03	0.05
Mult 6Q13 Imag	0.04	0.03	0.01
FF20 bit (real)	0.02	0.00	0.02
subtractors	0.01	0.01	0.00
$\vdots$	< 0.01	< 0.01	< 0.01
Total	2.66	2.42	0.24

Table 10.4: *SFFT datapath power usage. Not all entities are shown.*

The three most power consuming entities are the ROM banks containing the values of $2 \cdot \cos(k)$ and the two twiddle factor ROMs. Compared to other ROMs, this usage is abnormally

high. A deeper examination of the implementation reveals that the ROMs are connected directly to the fast clock, without a signal to enable and disable the ROMs. This is a design error, as the ROMs in the current configuration are always read, even when everything else in the SFFT block is idle. With this error corrected the power usage should drop to levels seen with other ROMs. As an example ROM # 3 of table 10.3 on the previous page uses 0.01mW. The ROMs in the SFFT datapath should approach this value.

As a further improvement the ROM containing the $2 \cdot \cos(k)$ values are only needed to be read once for each value of l. An improvement of the control structure of the datapath to only read this value once and then store it in a latch could be investigated, but it must be considered that the gain could be insignificant.

The remaining components such as multipliers, subtractors and flip-flops are difficult to change as they require a complete redesign of the implementation. Additional alternatives include a test implementation of the MAC for the SFFT datapath solution shown in figure 8.3 on page 74, to see if it performs as well or better than the current SFFT method.

10.5 Summary

A resume of the main discoveries in the design space exploration is given below.

- Implement a power-off instead of idle, to avoid static static power expenditure. This would make the SFFT feasible for values of approximately $L < 100$ as seen in figure 10.1 on page 101.
- Correct design errors:
  - Simulation taps erroneously connected to output pins. This should lower the miscellaneous power consumtion in table 10.2 on page 102.
  - ROMs in the SFFT datapath should be controlled by a clock enable, which would lower the power expenditure. This should lower the power usage with approximately 1.5 to 2mW per datapath. This change would bring the SFFT power usage to a level comparable to that of the SRFFT in table 10.2 on page 102.
- The clock generating circuit output should be compared to the requirements to precision set by the overall design. The quality of the output, and thus the complexity of the clock generator, should be adjusted to match the requirements. Furthermore the placement of the clock generator, whether by PLL or alternative implementation, could have an impact on the routing power expenditure.
- Experiments with the number of instantiated datapaths in the SFFT and overall clock frequency, compared to the overall power expenditure of the system could be performed.

With these optimizations implemented a new power-analysis should be performed to further improve the design.

Chapter 11

Conclusion

The main purpose of this project was to investigate the power consumption of DFT algorithms suited for multiuser OFDM, when implemented on a FPGA platform, by comparison of the achieved power consumption with the theoretical computational complexity of the DFTs when only a subset of the OFDM subcarriers need to be demodulated. Through the work carried out to achieve this comparison and evaluation of performance measures, the project considered topics concerning algorithm mapping onto the FPGA platform, power simulation and measurement and design space exploration for power optimization as well.

Based on an analytical model of an OFDM downlink scenario, a DFT subsystem was specified for evaluation. A full length Split Radix FFT and a Transform Decomposition method (Sørensen FFT) for calculating only a subset of the FFT outputs was examined and mapped onto a Altera Cyclone III FPGA system using a Finite State Machine with Datapath approach.

Next, the DFT systems was evaluated with regards to power consumption by both simulations using a Altera Powerplay analyzer, and by measurements of the power supply feeding the internal logic of the FPGA. Comparing these simulations and measurements showed a consistent tendency for the power analyzer to underestimate the power consumption. This difference may be reduced with the maturing of the power model for the device family, which is preliminary, and by more elaborate setup of the device ambient characteristics used in the simulations. Still the comparisons of simulations and measurements showed good coherency in power consumption changes from test case to test case, and the elaborate structure of the simulation results was therefore used to determine potential optimizations of the system design.

The conclusion made from the measurements and simulations is, that the computational complexity is not directly comparable to the power usage for this project.

In order for the computational complexity to be a good performance measure compared to the power consumption for this project, is essential to keep idle power consumptions low or to remove it all together. This is due to the idle or static power consumption constituting a considerable percentage of the total power consumption, for the SRFFT the idle power consumption accounts for up to 62%. In addition to the disabling of clocks to inactive circuits used in the

design, an introduction of a power-off state to the system is recommended to minimize this idle power consumption.

Secondly, the evaluation of minimizing the system clock frequency to exploit the requirements for algorithm completion time showed significant mean power consumption reductions. A reduction of the system clock from 50 MHz to 16.7 MHz gave rise to power consumtion reductions of 35% and 23% for active and idle states, respectively.

When it comes to the design space exploration, the simulations show several potential power optimizations in the design. The addition of parallel datapaths in the SFFT to keep the system clock low proves to increase the power consumption significantly. The optimal tradeoff between utilized area and system clock frequency would require further iterations of the system design process.

An additional improvement in area utilization may be found from an evaluation of the data format used in the algorithms. The numerical precision should be no better than required in the system since widening the data buses also increase the size of the calculation units and thus the area utilization.

Finally, the circuitry for generating the system clocks, which is implemented using a PLL embedded in the FPGA, has shown to be the source of a significant power consumption as well. Therefore, a redesign of the clock generation circuit using e.g. counters instead of a PLL or a different crystal, should be tested to evaluate possible power consumption improvements.

Although, the design used for test in this project is not power-wise optimal, the analysis of power consumption in FPGAs and tests and simulations of the resulting system has given rise to some conditions for achieving consistence between algorithm computational complexity and implementation power consumption as well as general guidelines for improving power performance of FPGA designs:

- Minimize idle power consumption by disabling clocks and powering off.
- Minimize system clock frequencies to fit system time constraints.
- Minimize area utilization by fitting data formats to requirements.
- Optimize time vs. area tradeoff by iterating over algorithm concurrency utilization.

These conditions and guidelines have been achieved with a predefined hardware platform. If the system design includes defining the HW platform additional topics such as supply voltage and device cooling are important to consider, since these parameters affect the FPGA device power consumption as well.

This project has focused on investigating the coherency between computational complexity and power consumption, when inplementing FFT algorithms suited for OFDM system on an FPGA device. Thus, only two algorithms and one architecture has been examined. For further work, investigations of other algorithms and architectures, such as DSP and micro-controllers, are interesting in order to verify if the results obtained are generally applicable, and to obtain a set of guidelines into which algorithms are most power efficiently mapped onto which HW-platforms.