A comparison of FFT processor designs

Simon Dirlik
Computer Architecture for Embedded Systems
Department of EEMCS, University of Twente
P.O. Box 217, 7500AE Enschede, The Netherlands
s.dirlik@student.utwente.nl

December 2, 2013

Supervisors:
Dr. Ir. André Kokkeler
Ir. Bert Molenkamp
Dr. Ir. Sabih Gerez

ASTRON
Ir. André Gunst
Ing. Harm Jan Pepping

UNIVERSITY OF TWENTE.

Abstract

ASTRON is the Netherlands Institute for Radio Astronomy. They operate, among others, LOFAR (Low Frequency Array), which is a radio telescope using a concept based on a large array of omni-directional antennas. The signals from these antennas go through various processing units, one of which is an FFT processor.

In the current LOFAR design, FPGAs are used for this, since the numbers are too small to afford custom chips. For future astronomical applications, especially for the SKA telescope, a more specific chip solution is desired. SKA will be much larger than LOFAR and use many more processing elements. As power consumption is a major concern, the FPGAs are unsuitable and they need to be replaced with ASICs.

The energy consumption of the FPGAs is compared to the energy comsumption of the same FFT design implemented on an ASIC. For the FPGA synthesis and power calculation, Quartus is used. The ASIC was synthesized with Synopsys Design Compiler using 65nm technology. The energy usage is reduced from 0.84μJ per FFT on the FPGA to 0.41μJ per FFT on the ASIC.

Four new ASIC designs are compared to the existing one, in search of a better solution. An approach that uses the minimal amount of memory (SDF), and one that uses more memory for faster calculation (MDC) are implemented for both radix-2 and radix-4 designs. Different complex multipliers and different methods of storing the twiddle factors are also compared.

The fast calculating radix-2 design gives the best results. Combined with a complex multiplier that uses Gauss' complex multiplication algorithm and a twiddle factor component based on registers, the energy comsumption per FFT can be reduced to 0.33μJ.

Contents

1 Introduction **3**
- 1.1 Radio astronomy . . . 3
- 1.2 ASTRON & LOFAR . . . 3
- 1.3 Fast Fourier Transform (FFT) . . . 4
- 1.4 Goals . . . 4

2 Description of the FFT **5**
- 2.1 Decimation in time . . . 5
  - 2.1.1 Butterflies . . . 5
- 2.2 Decimation in frequency . . . 6
- 2.3 Bit-reversed order . . . 7
- 2.4 Radix-4 . . . 8
- 2.5 Split-radix . . . 9
- 2.6 Radix-2^n . . . 9
  - 2.6.1 Radix-2^2 . . . 10
  - 2.6.2 Radix-2^3 . . . 11

3 Architectures **12**
- 3.1 Single-memory architectures . . . 12
- 3.2 Dual-memory architectures . . . 12
- 3.3 Pipelined architectures . . . 12
- 3.4 Array architectures . . . 14

4 FFT implementations presented in literature **15**
- 4.1 ASIC Design of Low-power Reconfigurable FFT processor [1] . . . 15
- 4.2 A Low-Power and Domain-Specific Reconfigurable FFT Fabric for System-on-Chip Applications [2] . . . 16
- 4.3 ASIC implementation of a 512-point FFT/IFFT Processor for 2D CT Image Reconstruction Algorithm [3] . . . 16
- 4.4 An Efficient FFT/IFFT Architecture for Wireless communication [4] . . . 17
- 4.5 Design And Implementation of Low Power FFT/IFFT Processor For Wireless Communication [5] 18
- 4.6 Low-power digital ASIC for on-chip spectral analysis of low-frequency physiological signals [6] 18
- 4.7 Low Power Hardware Implementation of High Speed FFT Core [7] . . . 19
- 4.8 ASIC Implementation of High Speed Processor for Calculating Discrete Fourier Transformation using Circular Convolution Technique [8] . . . 19
- 4.9 Comparison . . . 20
- 4.10 Discussion of the results . . . 21

5 Description of the implemented designs **22**
- 5.1 ASTRON's implementation . . . 22
  - 5.1.1 Avoiding overflow . . . 22
- 5.2 New Radix-2 DIF implementations . . . 23
  - 5.2.1 Variant 1 (NEWv1) . . . 23
  - 5.2.2 Variant 2 (NEWv2) . . . 24
  - 5.2.3 Complex multipliers . . . 25
  - 5.2.4 Twiddle factors . . . 25
- 5.3 Radix-4 DIF implementation . . . 26
  - 5.3.1 Variant 1 (NEWR4v1) . . . 26
  - 5.3.2 Variant 2 (NEWR4v2) . . . 27
- 5.4 Synthesized combinations of components . . . 28

6 FPGA versus ASIC using ASTRON's design **29**
- 6.1 Area . . . 29
- 6.2 Power and Energy . . . 29

7 Comparison of ASTRON design with new designs **31**
7.1 Area . . . 31
7.2 Power and Energy . . . 31
7.3 Comparison using FOMs . . . 33

8 Discussion **34**
8.1 FPGA versus ASIC . . . 34
8.2 Design . . . 34
8.3 Components . . . 35

9 Conclusion **36**
9.1 Recommendations & Future Work . . . 36

List of abbreviations **37**

References **39**

1 Introduction

1.1 Radio astronomy

Radio astronomy is a subfield of astronomy that studies celestial objects by capturing the radio emission from these objects. The field has attributed much to the astronomical knowledge since the first detection of radio waves from an astronomical object in the 1930s. Most notably are the discovery of new classes of objects such as pulsars, quasars and radio galaxies.

1.2 ASTRON & LOFAR

ASTRON is the Netherlands Institute for Radio Astronomy. They operate, among others, LOFAR (Low Frequency Array), which is a radio telescope using a concept based on a large array of omni-directional antennas. The signals from these antennas are combined using beamforming, to make this a very sensitive telescope. LOFAR consists of about 7000 small antennas which are concentrated in 48 stations in total. 24 of these stations are grouped in the core area of the LOFAR, which is about 2-3km^2 and is located near Exloo in the Netherlands. There are 14 remote stations also in the Netherlands and there are 8 international stations, of which 5 are located in Germany, while France, Sweden and the UK each have 1 station. There are 2 more stations in the Netherlands which are not operational yet.
There are 2 types of antennas; Low Band Antennas (LBA) which are capable of observing the range between 10 and 90 MHz, but are optimized for the 30-80MHz range. Furthermore, there are High Band Antennas (HBA), which are capable of observing the range between 110MHz and 240MHz, but are optimized for the 120-240MHz range. The data from the antennas is digitized and processed at the station level before it is transferred to the BlueGene/P supercomputer at the University of Groningen where signals from all stations are combined and processed. Figure 1 shows the signal path.

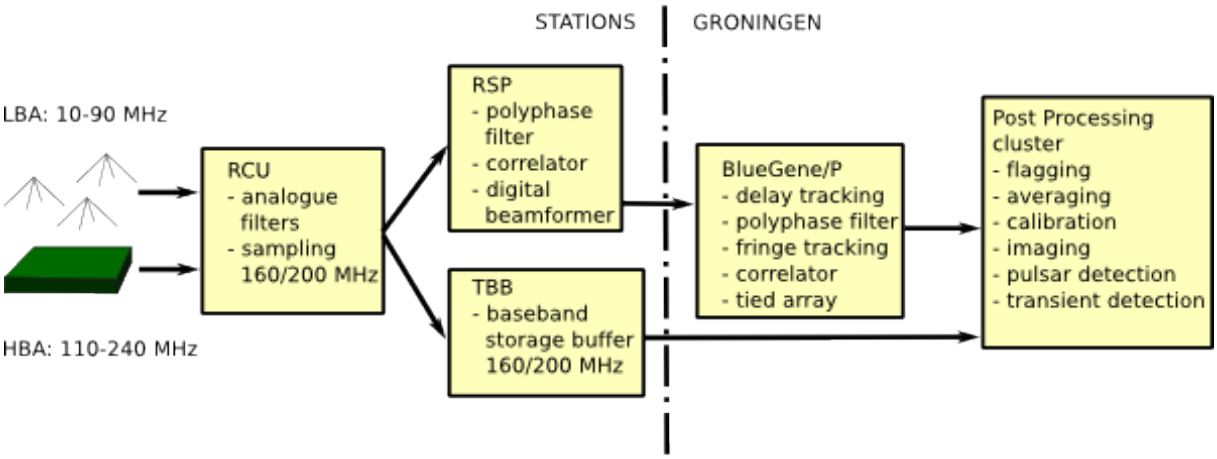


Figure 1: LOFAR signal path. On the left-hand side the station processing, on the right-hand side the processing at the supercomputer centre in Groningen. (this picture was taken from the ASTRON website)

The raw signals first pass the digital Receiver Units (RCU), where they go through some analogue filters to suppress unwanted radio signals. The filtered signals are digitized using a 12-bit ADC at a sampling frequency of either 160MHz (80MHz total bandwidth) or 200MHz (100MHz total bandwidth). The digital signal can go to 2 different types of boards, the Transient Buffer Boards (TBB) and the Remote Station Processing (RSP) boards. The TBB stores the last 1.3s of data in memory buffers. This data can be stored on a separate memory, if an algorithm running on a local FPGA fires a trigger or if an explicit command is given to the TBB. The saved data can then be analysed offline. The RSP splits the signal into 512 subbands using a polyphase filter (PPF) which is followed by a 1024-point FFT. The most common processing step on the separated signals is beamforming based on digital phase rotation. The beam-formed signals are then sent to the BlueGene/P over the wide area network (WAN). The BlueGene/P supercomputer does all further (online) processing, it can perform delay compensation, FFT, PPF etc. The results from the BlueGene/P and the TBBs are stored on the post-processing cluster, where more (offline) processing can be done like averaging, calibration, imaging etc.

1.3 Fast Fourier Transform (FFT)

The FFT is an algorithm introduced in 1965[9], which computes the Discrete Fourier Transform (DFT) in a fast way. The DFT, which is an adaptation of the original Fourier Transform (FT)[10], operates on discrete input signals, as opposed to the FT which is only defined for continuous input signals. The FT decomposes an input signal into a (infinite) list of sinusoïds of which the original signal consists. So the output of the FT, which are amplitudes of frequency components, can be used to process and manipulate the signal. One example is to reduce noise in an image or audio stream by filtering out the noisy frequencies. Another example is data compression; in some audio files for instance, inaudible frequencies are filtered out. But the applications in digital signal processing are many; from solving differential equations to wireless communication.

1.4 Goals

Within LOFAR, the FFT is done on a field-programmable gate array (FPGA). The intention is to investigate the implementation of the FFT on an application-specific integrated circuit (ASIC). An ASIC is an integrated circuit designed to perform one specific task very efficiently in terms of speed and power. This is opposed to a general purpose integrated circuit, which is designed to perform many tasks but does so much less efficiently. Though FPGAs are more flexible than ASICs, they are not as efficient. The next phased array, the Square Kilometer Array (SKA)[11], will be much larger than LOFAR and use many more FFT processing elements. As power consumption is a major concern, the FPGAs are unsuitable and they need to be replaced with ASICs. Currently, the FPGAs perform 1024-point FFTs on 16-bit data. Their clock speeds are 200MHz and with 1 FFT every 1584 clock cycles, they can perform more than 126k FFT's/second. The goal of this research is to find out what architectures and implementation techniques are most suitable for this specific case.

The first goal is to find out how much of a difference an ASIC will make compared to an FPGA. The main focus of this comparison will be the power consumption. To find out, the current implementation will be synthesized using Quartus for the Stratix IV FPGA it runs on now. Synopsys Design Compiler will be used to synthesize the same design for an ASIC.

The second goal is to find out what implementation techniques and architectures are most power efficient. To find out, four more implementations will be made based on different architectures. All designs will however be pipelined architectures, since they are most suitable for high throughput applications (chapter 3). Within these designs, different implementation techniques will be used to see how they affect power consumption. These designs will be synthesized for an ASIC using Synopsys Design Compiler. They will then be compared with each other and with ASTRON's implementation on ASIC.

2 Description of the FFT

Equation 1 shows the Discrete Fourier Transform. In this equation $x_0...x_{N-1}$ are the input samples.

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-2\pi i n \frac{k}{N}}, \ k = 0, 1...N-1 \tag{1}$$

The number of operations using a direct calculation would be in the order $O(N^2)$. By using a divide-and-conquer algorithm, the FFT requires $O(Nlog_r(N))$ operations. The radix, r, stands for the number of parts that the input signal will be divided into. The radix-2 algorithm is the simplest and most used form, it divides the input signal into 2 parts. The FFT of the two parts can be calculated separately and can then be combined to form the complete DFT. This dividing into smaller parts is done recursively, requiring the number of samples of the input, N, to be a power of 2[10][12].

2.1 Decimation in time

The input signal can be divided into 2 interleaved parts (odd and even n), this is called decimation in time (DIT). Equations 2a to 2d show the mathematical expressions behind dividing the input signal using the radix 2 DIT algorithm. The input $x_0...x_{N-1}$ will be divided into even and odd indices: $n = 2m$ and $n = 2m+1$. W_N^k is called the twiddle factor.

$$X_k = \sum_{n=0}^{N-1} x_n \cdot W_N^{kn} \tag{2a}$$

$$= \sum_{m=0}^{N/2-1} x_{2m} \cdot W_N^{k(2m)} + \sum_{m=0}^{N/2-1} x_{2m+1} \cdot W_N^{k(2m+1)} \tag{2b}$$

$$= \sum_{m=0}^{N/2-1} x_{2m} \cdot W_{N/2}^{km} + \sum_{m=0}^{N/2-1} x_{2m+1} \cdot W_{N/2}^{km} W_N^k \tag{2c}$$

$$= \sum_{m=0}^{N/2-1} x_{2m} \cdot W_{N/2}^{km} + W_N^k \sum_{m=0}^{N/2-1} x_{2m+1} \cdot W_{N/2}^{km} \tag{2d}$$

$$W_N^{kn} = e^{-2\pi i \frac{kn}{N}} \tag{2e}$$

Equation 2d shows that only N/2 length DFTs need to be computed. The DFT is periodic which is shown in Equation 3a and the same calculation can be done for the half-length DFTs in equation 2d. The twiddle factor is also periodic, equation 3c shows that the only difference is that the sign changes. This periodicity is exploited by the algorithm to gain speed; it re-uses the computations for outputs of $k = 0...(N/2)-1$, in the computations for the outputs of $k = N/2...N-1$.

$$\sum_{n=0}^{N-1} x_n \cdot e^{-2\pi i n \frac{k+N}{N}} = \sum_{n=0}^{N-1} x_n \cdot e^{-2\pi i n \frac{k}{N}} e^{-2\pi i n} = \sum_{n=0}^{N-1} x_n \cdot e^{-2\pi i n \frac{k}{N}} \tag{3a}$$

$$e^{-2\pi i n} = 1 \tag{3b}$$

$$e^{-2\pi i \frac{k+N/2}{N}} = e^{-2\pi i \frac{k}{N}} e^{-\pi i} = -e^{-2\pi i \frac{k}{N}} \tag{3c}$$

$$e^{-\pi i} = -1 \tag{3d}$$

2.1.1 Butterflies

The input is recursively divided into smaller DFTs. Size-2 DFTs are the smallest components of the FFT. The equations for a size-2 DFT are shown in (4a) and (4b).

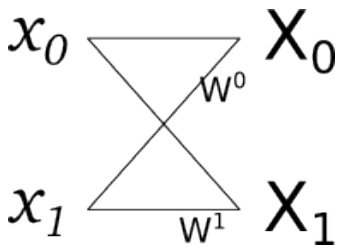
$$X_0 = x_0 + x_1 \cdot W^0 \tag{4a}$$

$$X_1 = x_0 + x_1 \cdot W^1 \tag{4b}$$

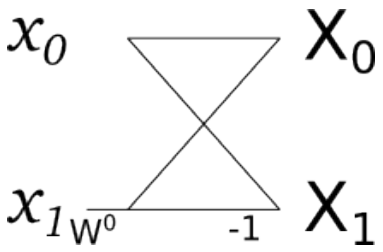
The data flow diagram of a size-2 DFT is presented in figure 2. This diagram is called a butterfly. Figure 2a shows a straightforward way of interpreting the formulas. Using equations 3c-3d, this can be rewritten into equations (5a) and (5b). Figure 2b shows the improved butterfly diagram.

$$X_0 = x_0 + x_1 \cdot W^0 \tag{5a}$$

$$X_1 = x_0 - x_1 \cdot W^0 \tag{5b}$$



(a) $X_0 = x_0 + x_1 \cdot W^0$ and $X_1 = x_0 + x_1 \cdot W^1$



(b) $X_0 = x_0 + x_1 \cdot W^0$ and $X_1 = x_0 - x_1 \cdot W^0$

Figure 2: Size-2 DFT butterfly

For larger FFT's this can be recursively extended, as shown in figure 3 for an 8-point FFT. This figure shows that the input values are not in order, this is explained in section 2.3. The figure also shows that there are 3 stages. Equation 6a shows that the number of stages depend on the size of the FFT, N, and the radix, r. The number of groups, g, in a stage can be calculated using Equation 6b, where s is the stage number, and the number of butterflies per group, b, can be calculated using equation 6c.

$$S = log_r(N) = log_2(8) = 3 \tag{6a}$$

$$g = N/r^s \tag{6b}$$

$$b = r^{s-1} \tag{6c}$$

Each stage has $N/2$ multiplications, $N/2$ sign inversions and N additions, so each stage can be done in $O(N)$ time. As explained before, there are $log_r(N)$ stages, making the order of the complete algorithm $O(Nlog_r(N))$.

2.2 Decimation in frequency

Another way to compute the DFT is to use the decimation in frequency (DIF) algorithm. This algorithm splits the DFT formula into two summations, one over the first half $(0...N/2-1)$ and one over the second half $(N/2...N-1)$ of the inputs. The derivation is shown in equations 7a-7d and equations 8a-8b.

$$X_k = \sum_{n=0}^{N/2-1} x_n \cdot W_N^{kn} + \sum_{n=N/2}^{N-1} x_n \cdot W_N^{kn} \tag{7a}$$

$$= \sum_{n=0}^{N/2-1} x_n \cdot W_N^{kn} + W_N^{Nk/2} \sum_{n=0}^{N/2-1} x_{n+\frac{N}{2}} \cdot W_N^{kn} \tag{7b}$$

$$= \sum_{n=0}^{N/2-1} \left(x_n + (-1)^k \cdot x_{n+\frac{N}{2}}\right) W_N^{kn} \tag{7c}$$

$$W_N^{Nk/2} = (-1)^k \tag{7d}$$

In equation 7c, the output, X_k, can now be split into interleaved parts, as opposed to DIT where the input was split.

$$X_{2k} = \sum_{n=0}^{N/2-1} \left(x_n + x_{n+\frac{N}{2}}\right) W_{N/2}^{kn}, \ \ k = 0, 1...\frac{N}{2} - 1 \tag{8a}$$

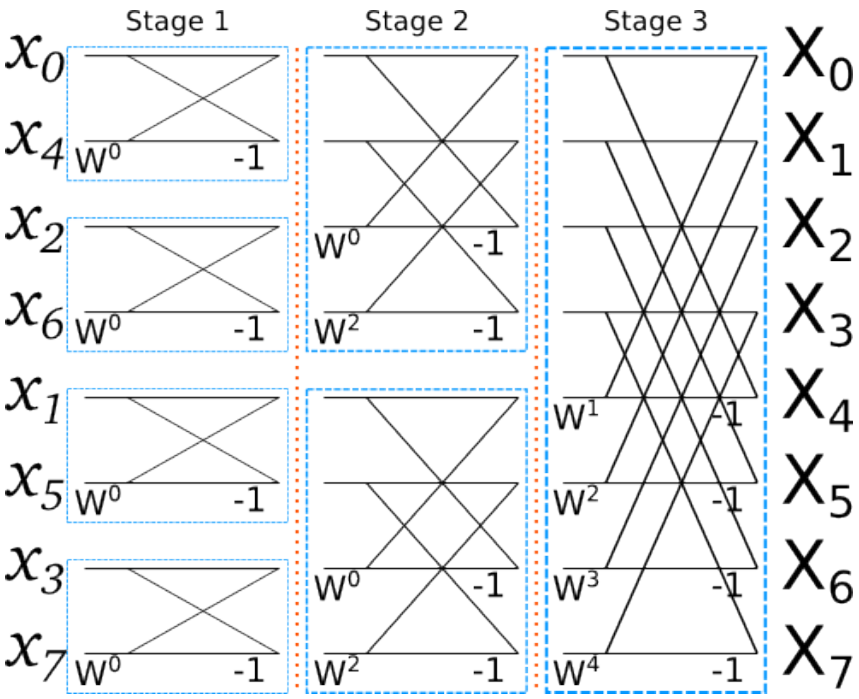


Figure 3: Size-8 DIT FFT; the red dotted lines separate the stages, the blue dashed lines separate the groups.

$$X_{2k+1} = \sum_{n=0}^{N/2-1} \left(x_n - x_{n+\frac{N}{2}}\right) W_N^n W_{N/2}^{kn}, \quad k = 0, 1 ... \frac{N}{2} - 1 \tag{8b}$$

The basic butterfly operation following from this, is shown in equations 9a-9b. Figure 4 shows that the data flow diagram is very similar to a DIT butterfly. The main difference is that the twiddle factor multiplication occurs at the end of the butterfly instead of at the beginning.

$$X_0 = x_0 + x_{\frac{N}{2}} \tag{9a}$$

$$X_1 = x_0 - x_{\frac{N}{2}} \cdot W_N^0 \tag{9b}$$

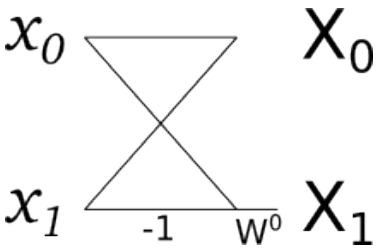


Figure 4: DIF butterfly

Figure 5 shows an 8-point DIF FFT. Equations 6a-6c still apply here, only the stage number, s, has to be reversed. The DIF algorithm requires the same amount of operations as the DIT algorithm.

2.3 Bit-reversed order

Figure 3 shows that in a DIT FFT, the inputs need to be rearranged, figure 5 shows that in a DIF FFT, the outputs need to be rearranged in the same order. Equation 10 shows that the correct order can be obtained

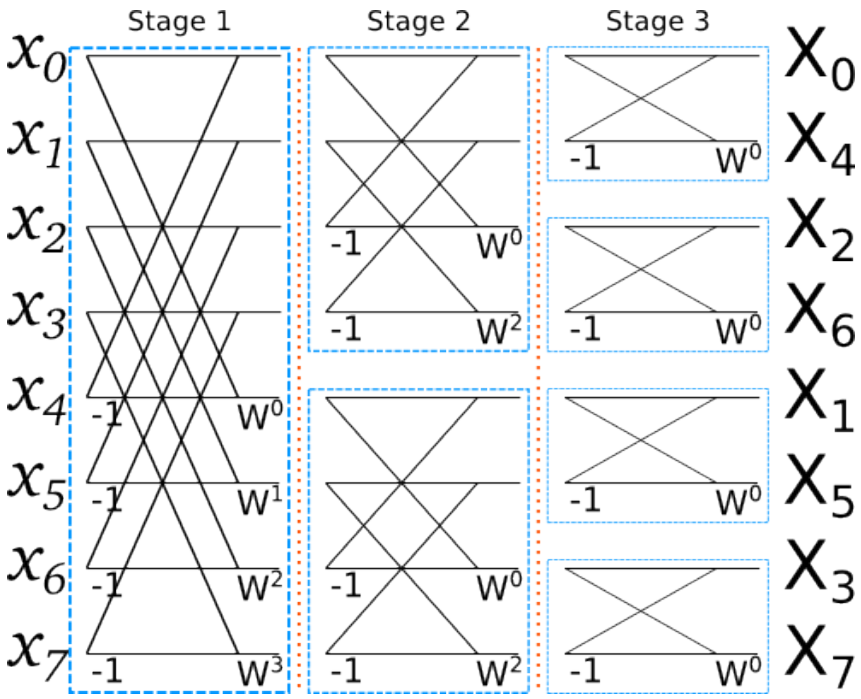


Figure 5: Size-8 DIF FFT; the red dotted lines separate the stages, the blue dashed lines separate the groups.

by reversing the bits in the binary representation of the index.

$$\begin{aligned}
0 \to (000) \ &< bit\text{-}reversal > \ (000) \to 0 \\
1 \to (001) \ &< bit\text{-}reversal > \ (100) \to 4 \\
2 \to (010) \ &< bit\text{-}reversal > \ (010) \to 2 \\
3 \to (011) \ &< bit\text{-}reversal > \ (110) \to 6 \\
4 \to (100) \ &< bit\text{-}reversal > \ (001) \to 1 \\
5 \to (101) \ &< bit\text{-}reversal > \ (101) \to 5 \\
6 \to (110) \ &< bit\text{-}reversal > \ (011) \to 3 \\
7 \to (111) \ &< bit\text{-}reversal > \ (111) \to 7
\end{aligned} \tag{10}$$

2.4 Radix-4

Using a higher radix to calculate the FFT has advantages and disadvantages. The radix-4 algorithm will be used to show the differences between radix-2 and higher radix FFTs.
The radix-4 algorithms split the DFT in equation 1 into 4 parts analogously to the radix-2 algorithms. The DIT algorithm is shown in equations 11a-11c.

$$X_k = \sum_{n=0}^{N-1} x_n \cdot W_N^{kn} \tag{11a}$$

$$= \sum_{m=0}^{N/4-1} x_{4m} \cdot W_{N/4}^{km} + \sum_{m=0}^{N/4-1} x_{4m+1} \cdot W_{N/4}^{km} W_N^{k} + \sum_{m=0}^{N/4-1} x_{4m+2} \cdot W_{N/4}^{km} W_N^{2k} + \sum_{m=0}^{N/4-1} x_{4m+3} \cdot W_{N/4}^{km} W_N^{3k} \tag{11b}$$

$$= \sum_{m=0}^{N/4-1} x_{4m} \cdot W_{N/4}^{km} + W_N^{k} \sum_{m=0}^{N/4-1} x_{4m+1} \cdot W_{N/4}^{km} + W_N^{2k} \sum_{m=0}^{N/4-1} x_{4m+2} \cdot W_{N/4}^{km} + W_N^{3k} \sum_{m=0}^{N/4-1} x_{4m+3} \cdot W_{N/4}^{km} \tag{11c}$$

Equations 12a-12d show the resulting equations for a butterfly and how they can be rewritten using equations 3b-3d. The butterfly itself is shown in figure 6.

$$X_0 = x_0 + x_1 + x_2 + x_3 \tag{12a}$$

$$X_1 = x_0 + x_1 \cdot W^1 + x_2 \cdot W^2 + x_3 \cdot W^3 = x_0 - x_1 \cdot jW^0 - x_2 \cdot W^0 + x_3 \cdot jW^0 \tag{12b}$$

$$X_2 = x_0 + x_1 \cdot W^2 + x_2 \cdot W^4 + x_3 \cdot W^6 = x_0 - x_1 \cdot W^0 + x_2 \cdot W^0 - x_3 \cdot W^0 \tag{12c}$$

$$X_3 = x_0 + x_1 \cdot W^3 + x_2 \cdot W^6 + x_3 \cdot W^9 = x_0 + x_1 \cdot jW^0 - x_2 \cdot W^0 - x_3 \cdot jW^0 \tag{12d}$$

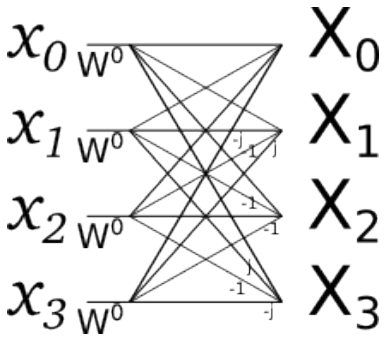


Figure 6: Radix-4 DIT butterfly.

The radix-4 butterfly requires 3 complex multiplications and 12 complex additions. For a N-point FFT that gives $(3N/4)log_4(N) = (3N/8)log_2(N)$ multiplications and $(3N)log_4(N) = (3N/2)log_2(N)$ additions. Compared to a radix-2 FFT, this reduces the number of multiplications by 25% and increases the number of additions with 50%. A disadvantage of the radix-4 algorithm is that it is only applicable for size 4^n FFTs.

2.5 Split-radix

The split-radix algorithm uses both radix-2 and radix-4 parts to compute an FFT. Equation 8a shows that the even part of the radix-2 DIF algorithm does not need any additional multiplications. The odd part does require multiplication by W_N^n. This makes the radix-2 more suitable for the even part and radix-4 for the odd part of the FFT. The FFT is therefore split into equations 13a-13c

$$X_{2k} = \sum_{n=0}^{N/2-1} \left(x_n + x_{n+\frac{N}{2}}\right) W_{N/2}^{kn} \tag{13a}$$

$$X_{4k+1} = \sum_{n=0}^{N/4-1} \left[\left(x_n - x_{n+\frac{N}{2}}\right) - j\left(x_{n+\frac{N}{4}} - x_{n+\frac{3N}{4}}\right)\right] W_N^n W_{N/4}^{kn} \tag{13b}$$

$$X_{4k+3} = \sum_{n=0}^{N/4-1} \left[\left(x_n - x_{n+\frac{N}{2}}\right) - j\left(x_{n+\frac{N}{4}} - x_{n+\frac{3N}{4}}\right)\right] W_N^{3n} W_{N/4}^{kn} \tag{13c}$$

This results in the L-shaped butterfly shown in figure 7, which can be recursively extended for larger N. The number of complex multiplications is $(N/3)log_2N$, which is less than radix-4. The number of complex additions is $(N)log_2N$, which is the same as radix-2. This means that the split-radix algorithm uses the lowest number of operations. Another advantage over high-radix algorithms is that it is applicable to FFTs of size 2^n. A disadvantage is that the structure is irregular, which makes it more difficult to implement[13][14].

2.6 Radix-2^n

The radix-2^n or cascade decomposition algorithms have the same number of complex multiplications as radix-4 (for radix-2^2), but it has the structure of a radix-2 FFT. The idea is to consider the first 2 steps of radix-2 decomposition together by applying a (n+1) dimensional map.

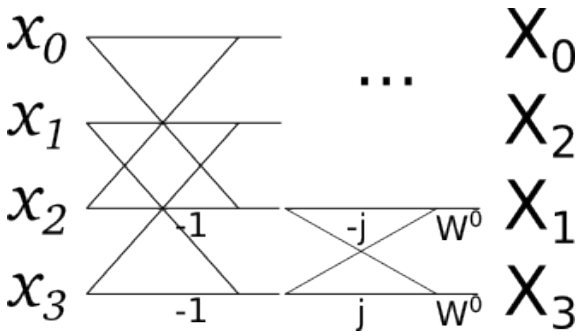


Figure 7: Split-radix DIF butterfly. One more radix-2 butterfly is needed for a 4-point FFT, but it was omitted to show the L-shape.

2.6.1 Radix-2^2

Equations 14a-14b show the 3-dimensional mapping for n=2. The decomposition using the Common Factor Algorithm [15][16], is shown in 15a-15b.

$$n =< \frac{N}{2}n_1 + \frac{N}{4}n_2 + n_3 > N \tag{14a}$$

$$k =< k_1 + 2k_2 + 4k_3 > N \tag{14b}$$

$$X(k_1 + 2k_2 + 4k_3) = \sum_{n_3=0}^{N/4-1}\sum_{n_2=0}^{1}\sum_{n_1=0}^{1} x(\frac{N}{2}n_1 + \frac{N}{4}n_2 + n_3)W_N^{(\frac{N}{2}n_1+\frac{N}{4}n_2+n_3)(k_1+2k_2+4k_3)} \tag{15a}$$

$$= \sum_{n_3=0}^{N/4-1}\sum_{n_2=0}^{1}\left[B_{N/2}^{k_1}(\frac{N}{4}n_2 + n_3)W_N^{(\frac{N}{4}n_2+n_3)k_1}\right] W_N^{(\frac{N}{4}n_2+n_3)(2k_2+4k_3)} \tag{15b}$$

$$B_{N/2}^{k_1}(\frac{N}{4}n_2 + n_3) = x(\frac{N}{4}n_2 + n_3) + (-1)^{k_1}x(\frac{N}{4}n_2 + n_3 + \frac{N}{2}) \tag{15c}$$

Equation 15c shows the structure of the butterfly. Computing the part between the square brackets in equation 15b before further decomposition, will result in an ordinary radix-2 DIF FFT. The idea of this algorithm is to decompose the FFT further, including the twiddle factor, so it is *cascaded* into the next step of decomposition. This exploits the easy values of the twiddle factor (1, -1, j, -j). Equations 16a-16b show the decompostion of $W_N^{(\frac{N}{4}n_2+n_3)k_1}$.

$$W_N^{(\frac{N}{4}n_2+n_3)k_1}W_N^{(\frac{N}{4}n_2+n_3)(2k_2+4k_3)} = W_N^{Nn_2n_3}W_N^{\frac{N}{4}n_2(k_1+2k_2)}W_N^{n_3(k_1+2k_2)}W_N^{4n_3k_3} \tag{16a}$$

$$= (-j)^{n_2(k_1+2k_2)}W_N^{n_3(k_1+2k_2)}W_N^{4n_3k_3} \tag{16b}$$

After equation 16b is subsituted in equation 15b and index n_2 is expanded, this results in a set of 4 FFTs of length $N/4$. This is shown in equations 17a-17b.

$$X(k_1 + 2k_2 + 4k_3) = \sum_{n_3=0}^{N/4-1}\left[H(k_1, k_2, n_3)W_N^{n_3(k_1+2k_2)}\right] W_{N/4}^{n_3k_3} \tag{17a}$$

$$H(k_1, k_2, n_3) = \left[x(n_3) + (-1)^{k_1}x(n_3 + \frac{N}{2})\right] + (-j)^{(k_1+2k_2)}\left[x(n_3 + \frac{N}{4}) + (-1)^{k_1}x(n_3 + \frac{3N}{4})\right] \tag{17b}$$

The parts between the square brackets correspond to the cascading of radix-2 butterfly stages[16][17]. This is shown in 8. The radix-2^2 algorithm requires $log_4(N)$ stages with N non-trivial multiplications, giving it a complexity of $Nlog_4(N) = N/2log_2(N)$. This is the same as the radix-2 algorithm.

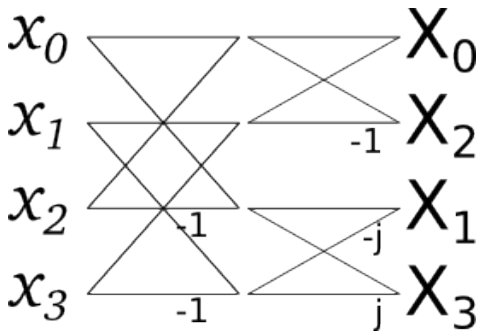


Figure 8: Radix-2^2 butterfly.

2.6.2 Radix-2^3

The equations for a radix-2^3 algorithm can be derived in a similar fashion, the results are shown in equations 18a-18d and in figure 9.

$$X(k_1 + 2k_2 + 4k_3 + 8k_4) = \sum_{n_4=0}^{N/8-1} \left[T(k_1, k_2, k_3, n_4) W_N^{n_4(k_1+2k_2+4k_3)} \right] W_{N/8}^{n_4 k_4} \tag{18a}$$

$$T(k_1, k_2, k_3, n_4) = H_{N/4}(k_1, k_2, n_4) + W_N^{\frac{N}{8}(k_1+2k_2+4k_3)} H_{N/4}(k_1, k_2, n_4 + \frac{N}{8}) \tag{18b}$$

$$H_{N/4}(k_1, k_2, n_4) = B_{N/2}(k_1, n_4) + (-j)^{(k_1+2k_2)} B_{N/2}(k_1, n_4 + \frac{N}{4}) \tag{18c}$$

$$B_{N/2}(k_1, n_4) = x(n_4) + (-1)^{k_1} x(n_4 + \frac{N}{2}) \tag{18d}$$

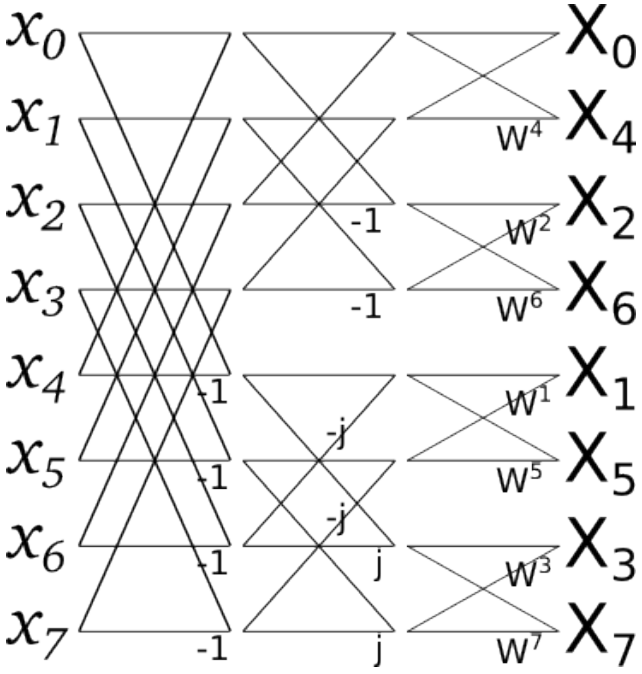


Figure 9: Radix-2^3 butterfly.

Equation 19 shows how the twiddle factor can be expanded to allow for a fixed-coefficient multiplier, which is more efficient than a general purpose multiplier. This makes the complexity of this algorithm $Nlog_8(N) = N/3log_2(N)$, which is the same as the split radix algorithm.

$$W_N^{\frac{N}{8}(k_1+2k_2+4k_3)} = (-1)^{k_3}(-j)^{k_2} W_N^{\frac{N}{8}k_1} = (-1)^{k_3}(-j)^{k_2} \left(\frac{\sqrt{2}}{2}(1-j) \right)^{k_1} \tag{19}$$

3 Architectures

There are many ways to implement the FFT algorithm. But when implementing the FFT in hardware (e.g. FPGA or ASIC), there are four main types of processing architectures[18]:

- Single-memory architectures
- Dual-memory architectures
- Pipelined architectures
- Array architectures

We will discuss these architectures shortly in this chapter[18].

3.1 Single-memory architectures

The single-memory approach is the simplest of the architectures. First the input values of an N-point FFT are loaded into memory, so the system needs a memory bank of at least N words. Then the first stage is calculated and its results stored back in memory, this can be done in-place. Those results are then used in the next stage and so on.

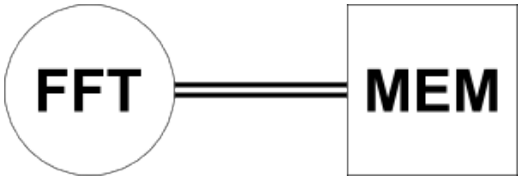


Figure 10: Simple diagram of a Single-memory architecture

3.2 Dual-memory architectures

The dual-memory approach is similar to the previous approach. However in this architecture the results of the first stage are stored in a second memory bank, which allows for reading, computing and writing to occur in one cycle. In the second stage the input is taken from the second memory bank and the results are stored in the first, this goes back and forth until all stages are completed.

Figure 11: Simple diagram of a Dual-memory architecture

3.3 Pipelined architectures

In a pipelined architecture there is not one (or two) big memory bank(s), but smaller pieces of memory located between stages in the FFT. There are several ways of implementing the pipelined architecture, the three most common ways are:

- Single-path delay feedback (SDF)
- Multi-path delay communicator (MDC)
- Single-path delay communicator (SDC)

In an MDC architecture, the input is broken into two (in case of radix-2) parallel data streams. The first half of the inputs is delayed in a buffer until the two inputs of the first butterfly have arrived. Figures 3 and 5 in chapter 2 show that input x_i is paired with $x_{i+N/2}$. The system uses delay buffers and a communicator to ensure that the correct pairs of input values arrive at the butterflies. The task of the communicator is to re-order the values before the next butterfly.

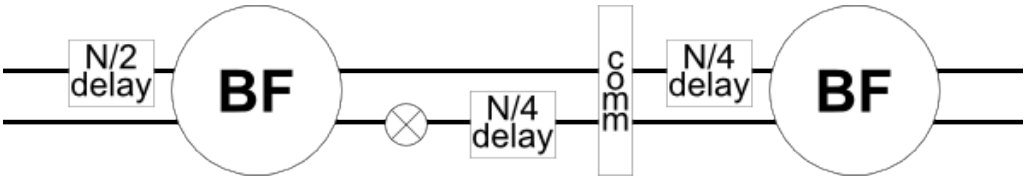


Figure 12: Simple diagram of part of a MDC architecture

In an SDF architecture there is only one stream of values, part of which is fed back into the butterfly, with the proper delay, to get the correct input values.

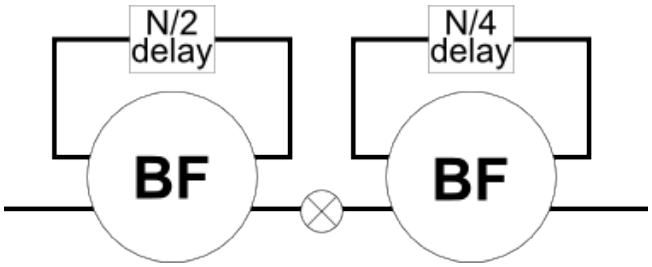


Figure 13: Simple diagram of part of a SDF architecture

Figure 5 in chapter 2 shows that for the first stage, input x_i is paired with $x_{i+N/2}$. For the second stage, input x_i is paired with $x_{i+N/4}$ and so on. Figures 12 and 13 show that the input is delayed in a buffer until the matching input arrives. This allows the pipelined architecture to start calculations before all inputs are read.
The architectures turn out differently when using a different radix. But generally, it can be said that SDF offers higher memory utilization than MDC and a higher radix offers higher multiplier utilization. Table 1 shows an overview of hardware utilization for the most common architectures. It shows that the radix-2 implementation using a MDC architecture (R2MDC) has a hardware utilization of 50%, however, this can be compensated for when 2 FFTs are calculated simultaneously. In case of a R4MDC, the same can be done to calculate 4 FFTs simultaneously[16]. The third type of pipelined architecture, Single-path Delay Communicator (SDC), uses a modified radix-4 algorithm as seen in [19]. It has higher hardware utilization than MDC and compared to SDF, it uses more memory and fewer additions. This architecture is however rarely used, mainly because the control logic is very complex.

Pipelined architectures generally have higher throughput than memory-based architectures because they have multiple butterfly units working at the same time[6]. This does require more complex control logic[18].

	#multiplications	#additions	memory size	multiplier utilization
R2MDC	$2log_4(N-1)$	$2log_4N$	$3N/2-2$	50%
R4MDC	$3log_4(N-1)$	$4log_4N$	$5N/2-4$	25%
R2SDF	$2log_4(N-1)$	$2log_4N$	$N-1$	50%
R4SDF	$log_4(N-1)$	$4log_4N$	$N-1$	75%
R4SDC	$log_4(N-1)$	$3log_4N$	$2N-2$	75%
R2^{2}SDF	$log_4(N-1)$	$4log_4N$	$N-1$	75%

Table 1: Overview of pipelined architectures. [16][18][19]

3.4 Array architectures

An array architecture consists of independent processing elements with local buffers. These processing elements are connected together in a network. To calculate the Fourier transform using an architecture like the one in figure 14, the one-dimensional input data is mapped onto a two-dimensional array. It is assumed that the length N is composite, $N = N_1 \cdot N_2$, where N_1 and N_2 are integers. Then an N point transform can be expressed as:

$$X(k_1, k_2) = \sum_{n_1}^{N_1-1} \sum_{n_2}^{N_2-1} x(n_1, n_2) W_{N_2}^{n_2 k_2} W_N^{n_2 k_1} W_{N_1}^{n_1 k_1}, \ k_1 = 0, 1...N_1 - 1, \ k_2 = 0, 1...N_2 - 1 \quad (20)$$

In equation 20, N_1 size-N_2 DFTs are computed. These DFTs, shown in equation 21, are transforms of the rows of the input. Each of these intermediate results are then multiplied by the twiddle factor $W_N^{n_2 k_1}$ and used in a second set of DFTs over the columns of the matrix $F(n_1, k_2)$[20].

$$F(n_1, k_2) = \sum_{n_2}^{N_2-1} x(n_1, n_2) W_{N_2}^{n_2 k_2}, \ n_1 = 0, 1...N_1 - 1, \ k_2 = 0, 1...N_2 - 1 \quad (21)$$

The biggest advantage of this type of architecture is that it has the flexibility to perform calculations other than the FFT. The final goal for ASTRON is to have a very efficient FFT. The ability to perform other types of calculations at the cost of efficiency is therefore unwanted. Designs using this architecture have therefore not been considered in this comparison.

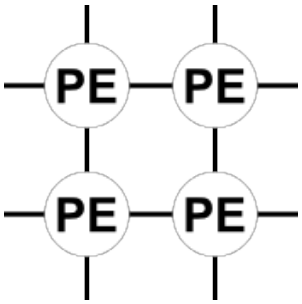


Figure 14: Simple diagram of a array architecture

4 FFT implementations presented in literature

The most common architecture is a pipelined architecture with Single-path Delay Feedback (SDF). This method is preferred by most because a pipelined architecture has higher throughput. The pipelined architectures require fewer clock cycles to finish an FFT calculation, so it can match the throughput of other architectures at a lower frequency. The SDF is preferred because it has higher hardware utilization than MDC and SDC. In the following designs, we will see pipelined and memory-based architectures.
The designs all use low radix butterflies, either 2 or 4, even though some are suitable for higher radices. The most important reason for the use of low radices is the complexity of the implementation of higher radix butterflies, as they require more non-trivial multiplications [21].
Two of the designs in this comparison are reconfigurable, meaning they can perform the FFT on variable length inputs. All designs work with fixed point values. For comparison one floating point architecture is added.
Some of the architectures also allow for inverse DFT computation, which is defined and rewritten as in equations 22a and 22b .

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k \cdot e^{2\pi i n \frac{k}{N}}, \quad n = 0, 1...N-1 \tag{22a}$$

$$= \frac{1}{N}\left(\sum_{k=0}^{N-1} X_k^* \cdot e^{2\pi i n \frac{k}{N}}\right)^* \tag{22b}$$

Because of the way it is rewritten, the IFFT can use the same hardware with the addition of a component that calculates the complex conjugate of the input at the beginning and a component that calculates the complex conjugate and divides the result by N at the end.

4.1 ASIC Design of Low-power Reconfigurable FFT processor [1]

The aim of this work is to make a low power and high speed reconfigurable FFT processor. The design consists of a radix-2 processing element (PE), two radix-4 PE's and two radix-8 PE's, which are put together in a pipeline SDF architecture (figure 15).

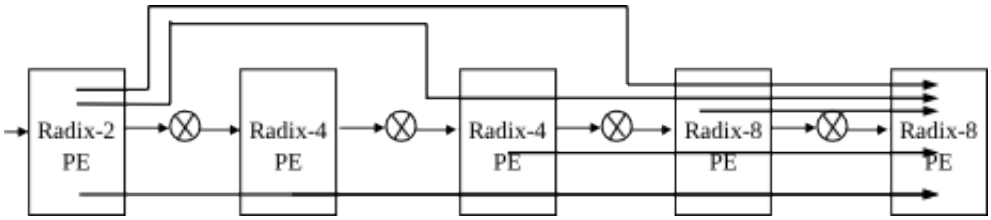


Figure 15: Pipelined architecture and data access. ([1])

Each of the PEs contain hardware to perform one stage of the FFT (figure 16). The complex multiplier produces 16-bit data from 12-bit input data, the compressing attenuator turns this into 14-bit data at the end of each stage[1]. Reconfigurability is achieved by turning blocks on or off, the two radix-8 blocks are fixed which gives a minimum of $N^S = 8^2 = 64$ points, this follows from equation 6. Using the same equations for the different radices, the design gets a maximum of $2^1 \cdot 4^2 \cdot 8^2 = 2048$ points.

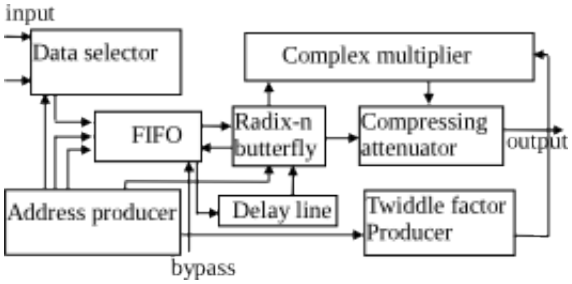


Figure 16: Architecture of the processing elements. ([1])

[1]This is confusing as apparently the PEs can get both 14-bit data from a previous PE, but also 12-bit data directly from the input

Power reduction is achieved using several methods. The first method is to cut off the power to unused blocks. The second method is providing memory with a voltage of 1.62V instead of traditional 1.98V[2]. The design uses a complex multiplier based on the CORDIC algorithm to reduce hardware costs and the amount of delay elements (compared to using ROM).
Although the authors claim to have made a low-power design, the results say something different. With 307.7mW, this is by far the biggest power consumer. With an average chip size of 2.1mm^2 and a high clockrate of 71.4MHz (for minimal power consumption), it is clear that to achieve reconfigurability the designers gave power the lowest priority.

4.2 A Low-Power and Domain-Specific Reconfigurable FFT Fabric for System-on-Chip Applications [2]

The goal of this paper is to get the optimal balance between low power and high flexibility. The system can be reconfigured to perform 16 to 1024 point FFTs using only one butterfly block. Figure 17 shows the memory-based design. Reconfigurability is achieved by masking bits in the Address Generation Block (AGB) and in memory. The AGB_1 generates addresses to select the correct twiddle factors, which are stored in the Coefficient Memory Cluster (CMC). AGB_2 generates addresses to select the correct input values for the butterfly block. The figure also shows that there are two Data Memory Clusters (DMC), making this a dual-memory based design. Section 3.2 explains that at each stage the data is read from one memory bank and written to the other. The Data Switch and the Address switch select the correct Memory Cluster to read from and write to.

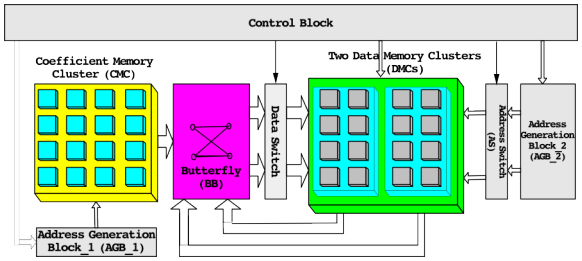


Figure 17: Memory based architecture. ([2])

There are only 15 configuration bits at the input, all other configuration data is encoded in the addresses generated by the AGBs. This is done so that the added flexibility has little effect on power consumption and size.
The results of the synthesis are compared to the same design without the reconfigurability and it shows only a slight increase in power (12-19%) and area(14%) compared to 1024 point FFT. However compared to the other designs in this study, this design uses more power than average: 68.7mW and 81.8mW for the non-reconfigurable and reconfigurable design respectively. The size is average in both cases: 2.51 & 2.86mm^2 and at 20MHz, this is one of the slowest processors especially considering it is memory-based.
The design was also compared to a Xilinx FFT Core generated by Xilinx Core Generator 6.1 and implemented on Virtex-2. The results show 30% less power consumption for 1024-point FFT on this design. The power savings are even higher when comparing smaller length FFT, up to 94% for 16-point. The Xilinx Core Generator gives many options to generate different types of FFT cores, unfortunately the authors do not describe what type of FFT core they used.

4.3 ASIC implementation of a 512-point FFT/IFFT Processor for 2D CT Image Reconstruction Algorithm [3]

The goal of this paper is to make an FFT processor with optimum hardware utilization. To reduce power consumption, the CORDIC algorithm is used to generate the twiddle factors.
The design has two RAMs, one reads and stores 512-point from the input, while the other serves as input for the butterfly. These two RAMs, RAM I and RAM II in figure 18, are synchronized to complete their tasks at the same time, after which they switch tasks. The input values have to be real numbers, the real parts of the intermediate results are stored in-place. RAM III is used to store the imaginary parts of the intermediate results. RAM IV and V are used to store the real and imaginary parts of the final result. The last step of this

[2]The authors of this paper claim 1.98V is traditional, there is however no reference to back up this claim.

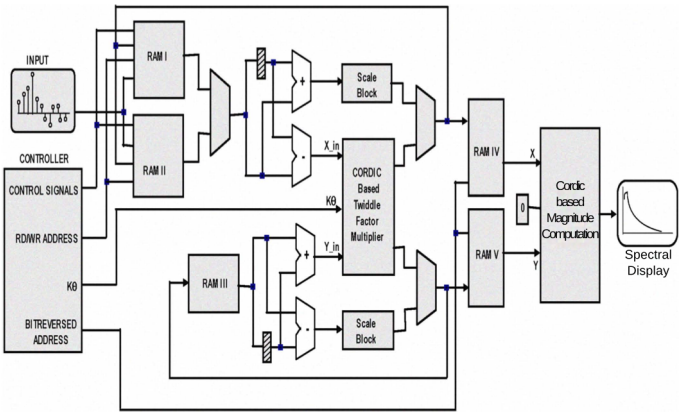


Figure 18: Memory based architecture. ([3])

design is the computation of the magnitude, this is also done based on the CORDIC algorithm.
Although the frequency is very high 220MHz, the throughput is average: 167.56μs[3]. This is caused by the CORDIC multiplier, which needs 16 clock cycles to perform one multiplication and the fact that it is a memory based architecture which requires higher frequency than the pipelined architectures. The power consumption and size of the chip are also average with 15mW and 3.16mm^2 respectively, which shows that the designers regarded each of the main characteristics (speed, power and area) as equally important.

4.4 An Efficient FFT/IFFT Architecture for Wireless communication [4]

In this paper the goal is to make a power-efficient architecture. This is done by using a reconfigurable complex constant multiplier and bit-parallel multipliers (using Booth's multiplication algorithm) to generate twiddle factors. This should also decrease hardware cost compared to a large ROM. By using the symmetry of the twiddle factors only a small ROM is needed containing 8 twiddle factors, other twiddle factors can be derived quickly from these values.

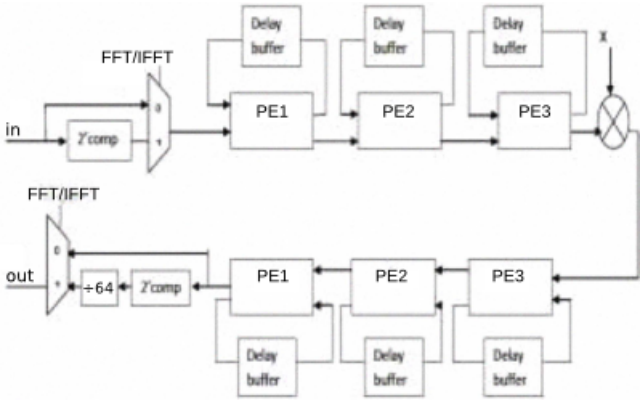


Figure 19: Radix-2 64-point pipelined SDF architecture. ([4])

Figure 19 shows the complete pipelined SDF architecture. Each of the processing elements (PE) in this architecture represents one stage in the FFT algorithm. PE3 is a simple radix-2 butterfly component without twiddle factor multiplication, and it is used as the basis for PE1 and PE2. In some stages the twiddle factor multiplications are more complex than in others and the different PEs are designed to fit the needs of each stage. PE1 performs computations, where the twiddle factors are of the form $-j$ or $W_N^{N/8}$, while PE2 only multiplies by -1 or i. The reconfigurable complex constant multiplier is shown in figure 20. It can generate all the other twiddle factors and is used to calculate those complex multiplications after the third stage. The results show a power consumption of 9.73mW, which is the second lowest value in this comparison and a gate count of 33590 which is the lowest in this comparison so it seems the designers achieved their goal, however no results are given about speed or technology.

[3] 9 stages · 256 butterfly operations · 16 clockcycles = 36864 clockcycles @ 220MHz = 167.56μs

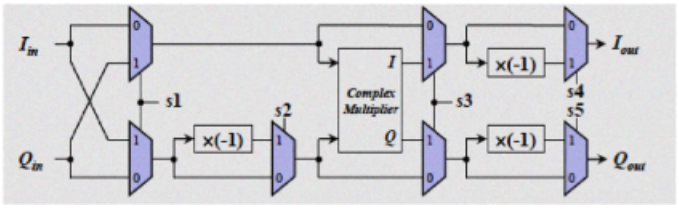


Figure 20: Reconfigurable complex constant Multiplier. ([4])

4.5 Design And Implementation of Low Power FFT/IFFT Processor For Wireless Communication [5]

The goal of this design is a low power 64 point FFT processor. To reduce chip size, a ROM-less architecture is used. This is achieved by using a reconfigurable complex multiplier using a modified Booth's multiplier. This algorithm was chosen because in [22] it was shown that it has a small truncation error compared to other implementations of Booth's algorithm. To increase speed it uses a radix-4 implementation.
This design is very similar to the previous one (section 4.4), differing only in radix. The papers show exactly the same structure, which is surprising since [4] states that it uses one PE per stage. A radix-4 design of a 64-point FFT uses 3 stages, not 6. The authors claim low cost and low power but no information is given about the synthesis results except that the design requires 33600 gates and runs at a frequency of 80MHz. The similarity between this design and [4] would be very interesting, to show the effect of using a different radix. Unfortunately, both designs do not give a lot of information about their synthesis results.

4.6 Low-power digital ASIC for on-chip spectral analysis of low-frequency physiological signals [6]

This paper describes a design to be used in a body sensor network, which means that low power and small area are required and speed is less important. The processor will be powered by battery and respond to physiological signals which do not exceed 1KHz. The processor is clocked at 1MHz. The design uses a hybrid architecture where most data is computed sequentially like in a memory-based architecture. But in the butterfly, the read, compute, write operations in the butterfly are pipelined. This allows for some speed, without sacrificing power consumption and area. The design uses a ROM to store twiddle factors.

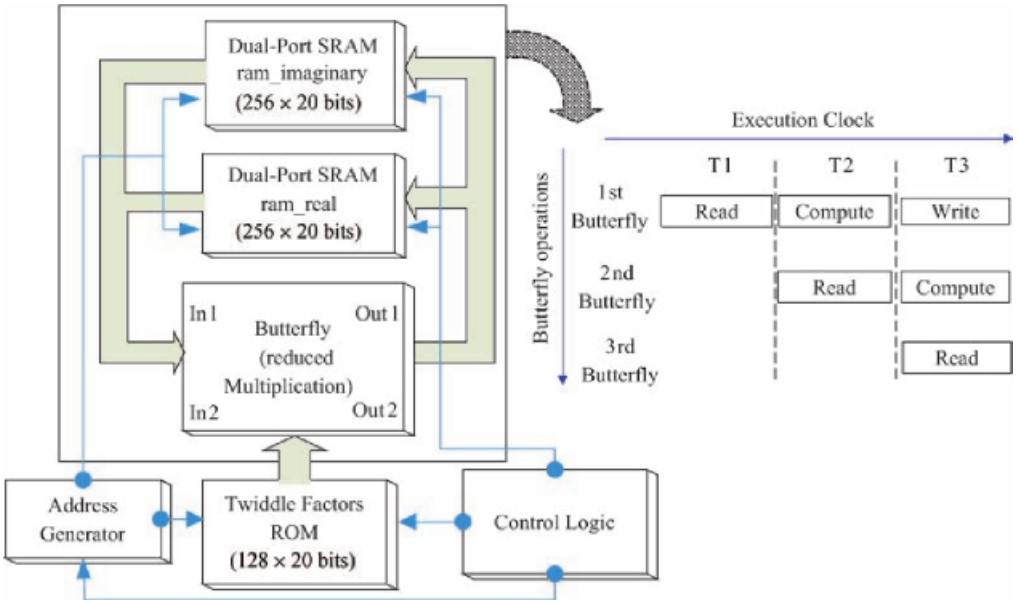


Figure 21: Hybrid architecture: memory based but with a pipeline in the butterfly. ([6])

Figure 21 shows the architecture and the pipelined operations in the butterfly. The twiddle factors are multiplied using a mathematical trick. Equations 23a and 23b show that the number of multiplications can be reduced

by 1 at the cost of 3 extra additions. In equation 23b, both the real and imaginary part contain the term $Y_r(W_r - W_i)$, and it only needs to be calculated once. This is more efficient as multiplication uses more computation resources than addition.

$$W \cdot Y = (W_r Y_r - W_i Y_i) + i(W_r Y_i + W_i Y_r) \tag{23a}$$

$$= [W_i(Y_r - Y_i) + Y_r(W_r - W_i)] + i[W_r(Y_r + Y_i) - Y_r(W_r - W_i)] \tag{23b}$$

The results meet the requirements: low power: 0.69mW, small area: 0.92mm^2, making this the smallest and most power efficient design is this comparison. It is also the slowest design. The authors also implemented this design on an FPGA, the results show that the FPGA implementation uses almost 6 times more power than the ASIC implementation.

4.7 Low Power Hardware Implementation of High Speed FFT Core [7]

This design uses a parallel pipelined architecture to achieve high throughput and low power. To reduce area the design uses Canonical Sign Digit (CSD) notation and a multiplier-less unit that does not store all the twiddle factors in ROM. Only a few twiddle factors are stored and the rest can be derived using only shift and add operations. To reduce power consumption the designers have taken into account that the inputs will be real, so the butterflies in stage 1 are modified to ignore the imaginary input.

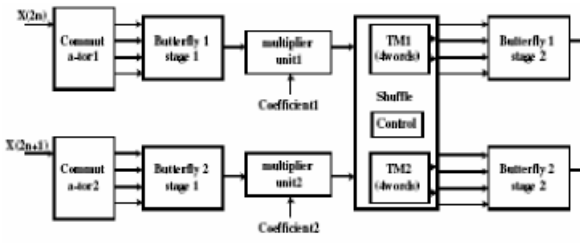


Figure 22: Parallel architecture. ([7])

Figure 22 shows the 2-parallel pipelined architecture. The input is split up in even and odd indexed values. It is a radix-4 16-point processor, which means there are only 2 stages and 4 butterflies per stage. There are 2 butterfly units running simultaneously in each stage.
The results show that this design leads to a very small area: 0.395mm^2 and very fast computation 28.8ns @ 833.33MHz. With a power consumption of 30.3mW this design achieves great speed and size, at a relatively small cost. A sidenote here is that this design probably does not scale well when it is adapted for i.e. 1024-point FFTs.

4.8 ASIC Implementation of High Speed Processor for Calculating Discrete Fourier Transformation using Circular Convolution Technique [8]

This design is aimed at making a high speed processor using circular convolution. This design is different from the others because it is made for floating point numbers. The speed of the design is supposed to be independent of the number of bits used and it uses CSD to improve the speed of multiplication and addition. It also uses radix-4 butterflies to increase the speed some more. To reduce chip size, this design only stores a few twiddle factors and it uses shift and add operations to calculate the others. Figure 23 shows the architecture. First, convolving matrices are generated using Matrix Vector Rotation (MVR). At the same time, twiddle factors are generated. The results of these components go into a multiply and accumulate (MAC) block. All arithmetic in the MAC is done using CSD to reduce area.
The results show that this design can perform a 16 point FFT in 23.79μs, the size of the processor is 12mm^2 and power consumption is 14.31mW. The design in section 4.7 is also a radix-4 16-point processor and is about 850 times faster, 30 times smaller and it uses more than twice the power compared to this design. This shows the result of the design choices made by [7], but also that floating point computations are terrible for performance.

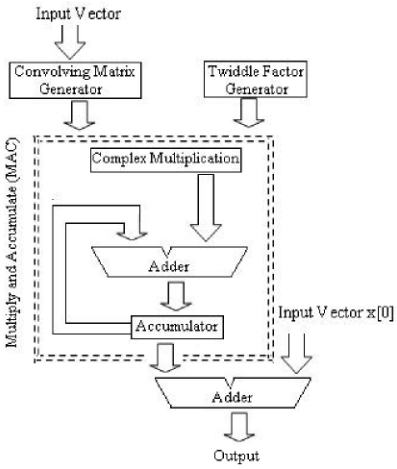


Figure 23: Floating point architecture using circular convolution. ([8])

4.9 Comparison

The designs use different length, precision and technology. To compensate for these differences, a number of figures of merit (FOM) are used.

When an architecture is synthesized using smaller technology, the result is, obviously, also smaller. To compensate for the differences in technology, the $Normalized\ Area$ is calculated using equation 24. This equation, presented in [18], normalizes the area to the smallest technology in the comparison, in this study the smallest technology is $90nm$.

$$Normalized\ Area = \frac{Area}{(Technology/90nm)^2} \tag{24}$$

To compare the power consumption, a FOM is introduced based on equation 25, from [23]. This equation factors out the effects of using different data width and technology for synthesis. The result of this equation represents the number of adjusted transforms per Joule.

$$Adjusted\ Transforms\ (FFTs/Joule) = \frac{Throughput \cdot Technology \cdot Data\ Width}{Power \cdot 10} \tag{25}$$

In [23] only 1024-point FFTs are compared and as a result, this equation does not consider different length FFTs. Since this study does compare different length FFTs, the FOM shown in equation 26 is introduced. It uses equation 25 as a starting point, but the throughput is multiplied by $N \cdot log_r(N)$. Without this alteration, the FOM would favour the 16-point FFTs in this study disproportionately. Then the scaling is removed because it produces more readable figures. This equation is used in table 3.

$$Adjusted\ Transforms\ (FFTs/Joule) = \frac{Throughput \cdot Technology \cdot Data\ Width \cdot N \cdot log_r(N)}{Power} \tag{26}$$

The last figure to be used, is the power consumption per butterfly (equation 27). In equation 26, the smaller length FFTs are still somewhat favoured, the power consumption per butterfly can put this in perspective. It is also an indication of how much more power the high speed cores use compared to the low speed cores.

$$Power\ Per\ Butterfly\ Operation\ (P/B) = \frac{Power}{Number\ of\ Butterflies} \tag{27}$$

Table 2 shows an overview of the designs. Unfortunately some authors do not specify all the necessary information leaving some blanks in the table. These are filled using common values or by making an educated guess using architectural analysis. Table 3 shows the results of the FOMs.

	N	Tech(nm)	Data Width	Radix	Max. Freq. (MHz)	Power (mW)	Area (mm^2)
[1]	1024	180	12	2-4-4-8-8[4]	86	307.7[5]	2.1
[2]	1024	180	16	2	20	81.8	2.68
[3]	512	130	16	2	220	15	3.16
[4]	64	180[6]	16	2	80[7]	9.73	0.4[8]
[5]	64	180	16[6]	4	80	4.9[7]	0.4[8]
[6]	256	180	40	2	1	0.69	0.092
[7]	16	180	64	4	833.33	30.3	0.395
[8]	16	90	16	2	-	14.32	12

Table 2: Summary of the designs

	Duration (μs)	Normalized Area (mm^2)	P/B (mW)	FFT's/second($\cdot 10^3$)	FFT's/Joule
[1]	143	0,525	0.4007	6.99	207
[2]	512	0,715	0.0160	1.95	704
[3]	168	1,513	0.0065	5.95	3803
[4]	3.2	0,1	0.0507	313	35519
[5]	3.2	0,1	0.1021	313	35265
[6]	2063	0,023	0.0007	0.485	5180
[7]	0.029	0,099	3.7875	34722	211220
[8]	23.79	12	0.4475	42	270

Table 3: Comparison of FOMs

4.10 Discussion of the results

What we see in table 3 is that one design stands out when it comes to throughput; with about 35 million FFTs per second, [7] is by far the fastest design and with about 211 thousand FFTs Joule, it is also the most efficient. The goal of this design was to make a high speed FFT core and that goal was achieved keeping the area small. But for a processor that only performs 16-point FFTs, it uses a lot of power. It has the highest power consumption of the non-reconfigurable designs and uses far more power per butterfly operation than the other designs. If this particular design was implemented for a 1024-point FFT, the power consumption of the processor would be approximately 4 times as high. This design also shows that using a higher radix, 4 in this case, can speed up the design without compromising the size of the chip.

When it comes to power consumption per butterfly operation and area, we see that [6] is the most efficient design. This design too achieved its goal, which was low power and small area, but it still managed to get a decent throughput using a hybrid architecture.

From table 2 it can also be concluded that reconfigurability comes at a high cost. In [1] the power consumption is very high, the highest of all designs, and in [2] the processor is very slow, the second slowest of all designs behind only [6]. These designs score the lowest FFTs per Joule together with [8], which performs floating point calculations.
Design [8] is the only design in this comparison that performs floating point calculations. The effects of that are visible most clearly in the size of the chip.

Designs [4] and [5] seem to find more of a balance in the tradeoff between power, area and speed. These designs end up in the middle in each list, but score very well in the number of FFT's per second. Design [2] also does not show remarkable figures in most area's, but has the second lowest power consumption per butterfly operation.

[4]For 1024-point FFT this design uses only the 4-4-8-8 blocks
[5]Power at optimal frequency of 71.4MHz.
[6]This value was not presented in the paper, so a common value is used.
[7]This value was not presented in the paper, but guessed based on the similarities between [4] and [5].
[8]This value was not presented in the paper, but guessed based on the number of gates and the technology.

5 Description of the implemented designs

ASTRON currently has an FFT implementation on a Stratix IV FPGA. Four more implementations were made for this research using different radices to see how this affects power consumption. The sizes of the FFT implementations are flexible (using VHDL generics). In this chapter, only the 1024-point designs will be discussed for simplicity. All implementations will use a pipelined architecture, since it is the most suitable for high throughput applications. The details of these implementations will be explained in this chapter.

5.1 ASTRON's implementation

This implementation uses the radix-2 pipelined SDF DIF algorihm. It was designed specifically for an FPGA. The size of the FFT depends on the VHDL generic `g_nof_points`, for which a value of 1024 will be used. The design consists of 10 stages (figure 24), each containing several components as shown in figure 25.

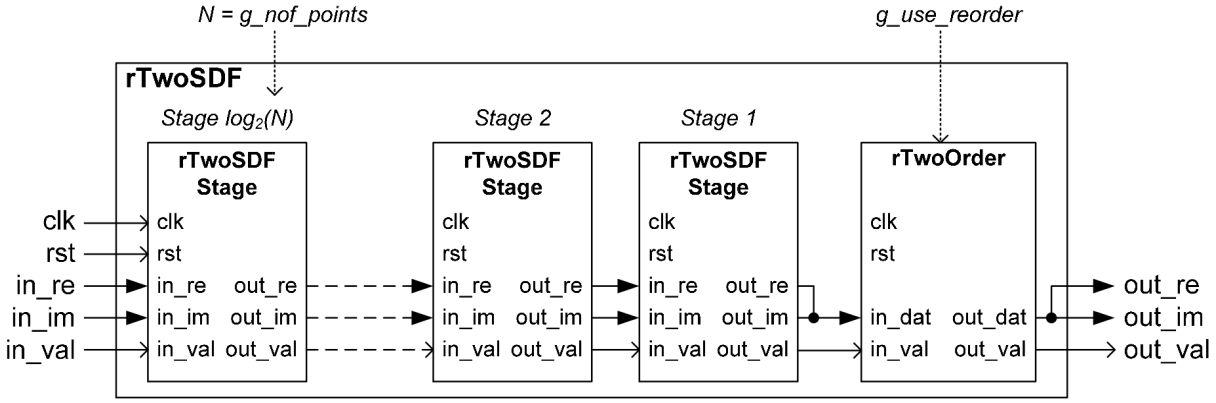


Figure 24: Schematic of the complete design.

The design can receive complex inputs. The real part of the input is put on the `in_re` signal, the imaginary part of the input is put on the `in_im` signal. The input values are only valid when the `in_val` signal is high. The `clk` and `rst` signals are the clock and reset signal, respectively.
At the output, the signals are similar: `out_re` and `out_im` are the real and imaginary parts of the output value. These values are only valid when the `out_val` signal is high.
In section 2.3 it is explained that the output signals of the DIF algorithm arrive in bit-reversed order. ASTRON's implementation has an optional component that reorders the values, it is only used when the VHDL generic `g_use_reorder` is set to `true`. The design will be synthesized without this component because it has a very large impact on the result, making it more difficult to compare the actual algorithm. It triples the total size of the design and requires a multitude of the power.
The main components of a stage are:

- `rTwoBFStage`, a butterfly component which performs the additions and subtractions of the butterfly. This component also contains the feedback delay.
- `rTwoWeights`, a component which selects the twiddle factors from a large memory containing all twiddle factors.
- `rTwoWMul`, a component which performs the multiplication. This component also performs truncation and resizing.
- `common counter`, this component keeps a counter so that the correct twiddle factor can be selected and the butterfly knows whether to delay the input or not.

Each stage can perform one butterfly operation at one time, so to perform the complete stage, 512 iterations are required. Every FFT implementation requires delays between stages as explained in chapter 3. To achieve a high clock speed in the FPGA, additional pipeline delays were added to this implementation.

5.1.1 Avoiding overflow

ASTRON's design uses unconditional block floating point scaling [24] to prevent data overflow. A value can potentially grow by two bits in one stage, therefore two guard bits are added for the input data to grow in. The

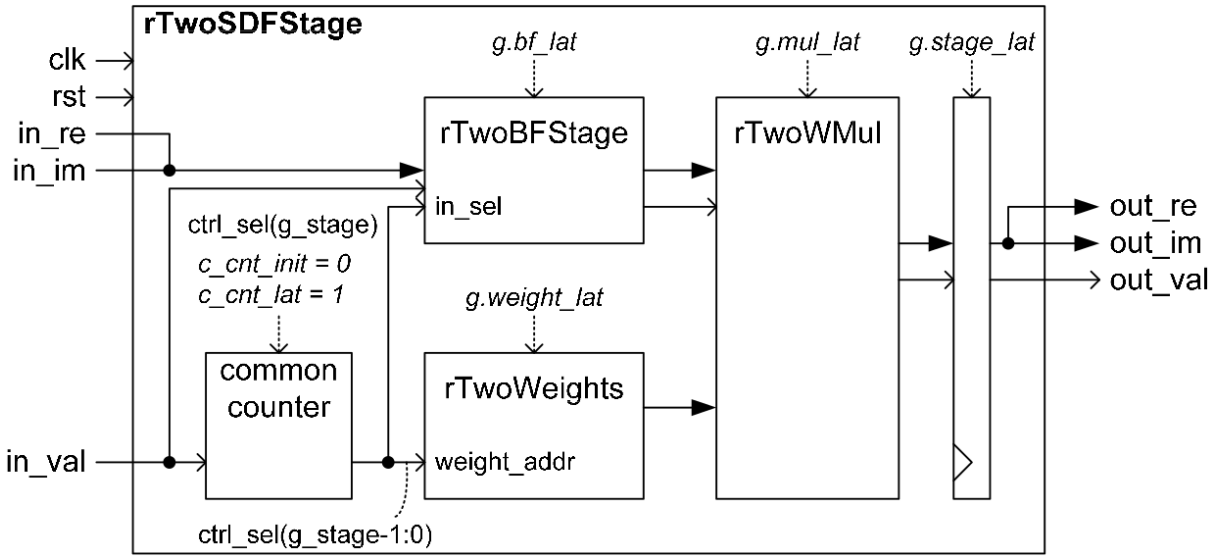


Figure 25: Schematic of a stage in the design.

data can, however, never grow by the maximum amount in two consecutive stages. Therefore this design with 10 stages uses 10 guard bits. After each stage the data is shifted to the right, unconditionally[9], to replace the guard bit. At the end of the calculation, the output is truncated and rounded to: $inputlength + log_2(N) + 1$ bits. ASTRON's implementation uses 18 bits for the internal signals, because the block multiplier of the Stratix IV has 18 bit inputs. At the output, the signals are truncated and rounded to 14 bit. This number was chosen to get an acceptable signal to noise ratio (SNR). From input to output, the design has a loss in SNR of about 2dB. The new designs are not bound by the 18 bit internal signals. Table 4 shows the effects of using different widths for the internal signals in the new radix-2 designs. The SNR loss is around 2dB when using 16 bit internal signals.

internal width	SNR loss (dB)
14	6.83
15	4.28
15 (15bit output)	2.98
16	2.09
17	1.30
18	0.97
ASTRON 18 bit	1.81

Table 4: SNR tests using different widths for the internal. One test uses 15 bit outputs instead of 14 bit.

5.2 New Radix-2 DIF implementations

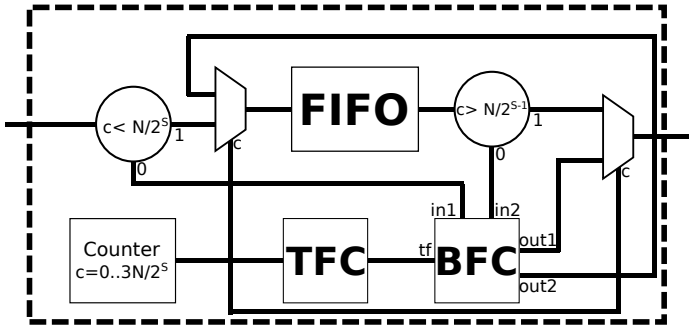
All radix-2 DIF implementations are similar. For this research, several variants of the implementation were tested. These variants show the effects of different components for the complex multiplier and twiddle factors. They also show the effects of different ways of buffering. The in- and outputs are identical to that ASTRON's design. Overflow prevention and truncation are also the same except for the widths of the internal signals, these are 16 bits.

5.2.1 Variant 1 (NEWv1)

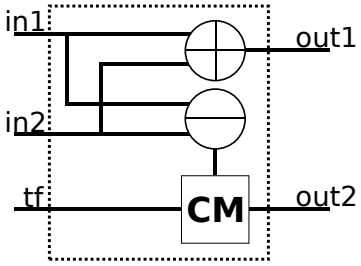
The first variant is very similar to that of ASTRON's, it is also an SDF implementation. It was, however, designed with the ASIC tooling in mind instead of the FPGA tooling, which should already make it a bit more efficient.
It consists of 10 stages, each containing a butterfly component (BFC) and a twiddle factor component (TFC). Figure 26a shows a schematic of the stages. In the real design the input and output values are represented by

[9] As opposed to conditional block floating point scaling, which is similar, but only shifts when the data grows.

two signals, a real and an imaginary part. This has been left out in the schematic to avoid clutter. The first input values are led into a FIFO until the counter reaches a threshold and the FIFO is full (section 3.3 explains the size of the FIFOs). The following input values, together with the output of the FIFO, go into the BFC together. The TFC produces a twiddle factor depending on the counter, this twiddle factor also goes to the BFC. The BFC calculates two output values at the same time. The first output value (`out1` in the figures) of the BFC goes directly to the output of the stage, the second output value (`out2` in the figures) is stored in the FIFO and led to the output after the BFC stops calculating new values.
Figure 26b shows a schematic of the BFC, the CM block is the complex multiplier. It does exactly what was explained in section 2.1.1 and shown in figure 2b.



(a) Schematic of a stage in the design.



(b) Butterfly component.

Figure 26

This variant uses the same buffer twice for one FFT; once at the input and once at the output. Figure 27 shows a schematic of the full design. The output of each stage goes directly to the input of the next stage.

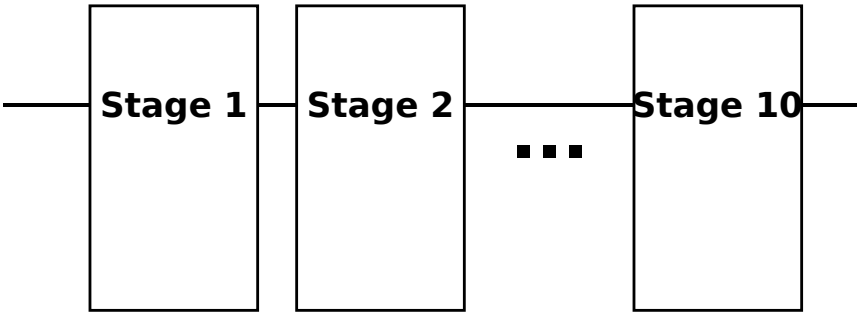


Figure 27: Schematic of full FFT design.

5.2.2 Variant 2 (NEWv2)

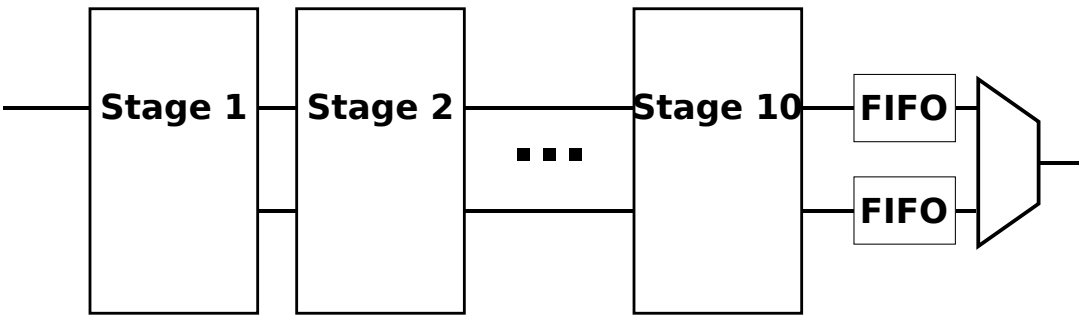
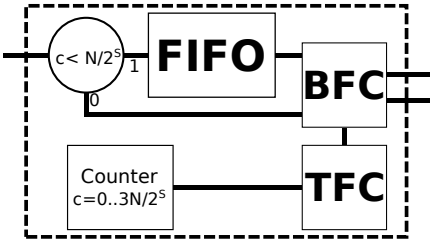


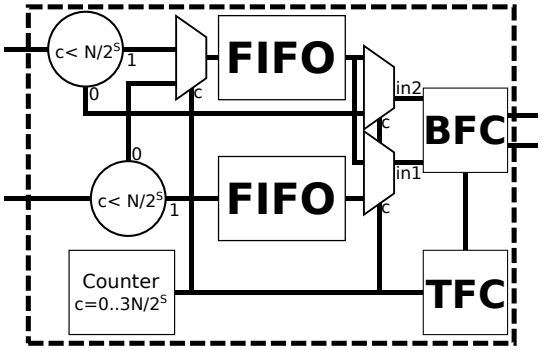
Figure 28: Schematic of second version of FFT design.

In the second variant, shown in figure 28, an MDC architecture is used. This allows for more parallel calculations and therefore fewer clock cycles to perform one FFT calculation.

The stages have 2 inputs and outputs, the outputs of a stage are fed directly into the next stage. This means an extra buffer is needed inside the stages. The first stage is different from the rest of the stages. It gets its single input directly from the FFT input. Figures 29a-29b show the schematics. Stage one works the same as a stage in NEWv1, except that the outputs of the butterfly are both directly connected to the output of the stage. In the other stages, both inputs need to be delayed. The first input line follows the same flow as the input in stage one. The first input values of the second input line are delayed in a seperate FIFO. When the counter reaches its threshold and the FIFOs are full, the values of the second input line are put into the FIFO of the first input line, which will then be outputting the values of the first input line into the BFC. When the flow of the first input line completed, both FIFOs will direct their output to the BFC.
This implementation performs one FFT operation in fewer clock cycles, because of the extra buffering. ASTRON's implementation, which requires 1584 clock cycles, can perform $200 \cdot 10^6/1584 = 126k$ FFT's/second at 200MHz. Because more calculations can be done in parallel, one full FFT calculation only requires 1041 clock cycles. So to match the number of FFT's/second of ASTRON's implementation, this implementation needs to run at a mere $126k \cdot 1041 = 131$MHz.



(a) Schematic of stage 1



(b) Schematic of a stage in the design. The multiplexers choose the inputs based on different thresholds of the counter value.

Figure 29

5.2.3 Complex multipliers

The complex multiplication can be done in several ways. Three methods were tested for this research:

- Straightforward implementation: $(A + Bi) \cdot (C + Di) = (AC - BD) + i(AD + BC)$.
- Gauss's complex multiplication: $(A + Bi) \cdot (C + Di) : k_1 = C \cdot (A + B), k_2 = A \cdot (D - C), k_3 = B \cdot (C + D) : (k_1 - k_3) + i(k_1 + k_2)$. This method requires fewer multiplications than the straightforward method, but more additions. Since multiplication is more expensive than addition, this could improve the design.
- Using a Synopsys Designware component. The ASIC synthesis tool comes with a component that, when given 4 inputs, calculates: $i_1 i_2 + i_3 i_4$. This matches well with the equation for the straightforward implementation.

5.2.4 Twiddle factors

Twiddle factors can be supplied in many ways. For this research, three ways were compared. The first way is a memory component containing all twiddle factors. Each stage requires a different set of twiddle factors. The first stage requires 512 twiddle factors, the second stage requires $512/2 = 256$, the third $512/4 = 128$ and so on. This makes a total of 1023 twiddle factors (Equation 28) and 2046 words of memory (real and imaginary parts). The memory component was synthesized using a constant array in VHDL, which results in registers and mulitplexers. And it was also synthesized using compiled RAM from CMP[10], which uses STMicroelectronics technology.

$$\sum_{s=1}^{10} 512/2^{s-1} = 1023 \tag{28}$$

[10] Circuit Multi-Projets : http://cmp.imag.fr

The second way, is to only store certain values in memory and deduce the rest using these values. The first stage requires 512 evenly spaced values on the bottom half of the unit circle (figure 30). A closer look at the unit circle shows that $\Re(W^k) = -\Re(W^{512-k})$ and $\Im(W^k) = \Im(W^{512-k})$, for $k = 0..256$. This means that in stage one, only 257 words of memory are required (instead of 1024) and some extra logic that selects the correct values at each index and multiplies by -1 if necessary.

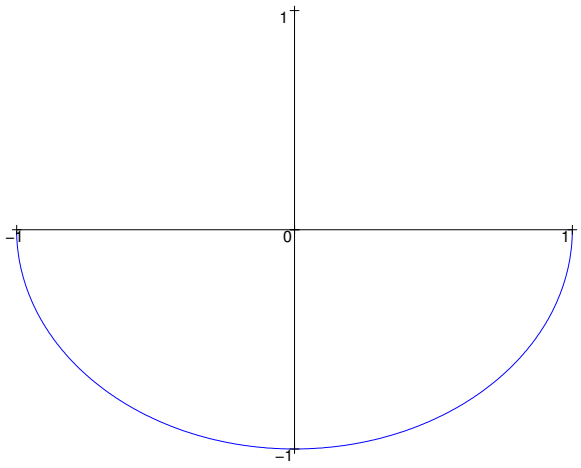


Figure 30: Bottom half of the unit circle.

The third way, is to calculate the values each time they are needed. Equation 2e shows the definition of the twiddle factors, equation 29 shows how this can be rewritten using Euler's formula. Synopsys supplies components that calculate sines and cosines. The real and imaginary parts can therefore easily be retrieved using these components.

$$e^{\frac{-2i\pi k}{N}} = cos(\frac{-2\pi k}{N}) + i \cdot sin(\frac{-2\pi k}{N}) \tag{29}$$

5.3 Radix-4 DIF implementation

The two radix-4 designs described below were made in the same style as the radix-2 designs. This will show whether a radix-2 or a radix-4 architecture is best suited for energy efficient FFT calculation. The overflow handling was adapted for a radix-4 butterfly, because input can now grow 3 bits in one stage. The designs use 9 guard bits, however, for the internal signals we keep using 16 bits. This gives less loss in SNR than for the radix-2 designs.

5.3.1 Variant 1 (NEWR4v1)

The NEWR4v1 implementation is an SDF architecture, implemented in the same style as the NEWv1 design. However because it is radix-4, there are now 5 stages. Every stage has a radix-4 BFC (which is more complex) and a TFC that can feed the butterfly correctly. A radix-4 butterfly performs three twiddle factor multiplications, so the TFC must be able to supply three twiddle factors in one clock cycle. Figure 32 shows a schematic of a radix-4 stage.
In a radix-4 butterfly the inputs are not just added or subtracted before the twiddle factor multiplication, they are also multiplied by $1, -i, -1, i$. These multiplications, however, can be reduced to sign changes and the switching of real and imaginary parts. This is shown in equations 30a to 30d. Equations 31a-31a show the resulting butterfly operations for the DIF algorithm.

$$1 \cdot (a + bi) = a + bi \tag{30a}$$

$$-i \cdot (a + bi) = b - ai \tag{30b}$$

$$-1 \cdot (a + bi) = -a - bi \tag{30c}$$

$$i \cdot (a + bi) = -b + ai \tag{30d}$$

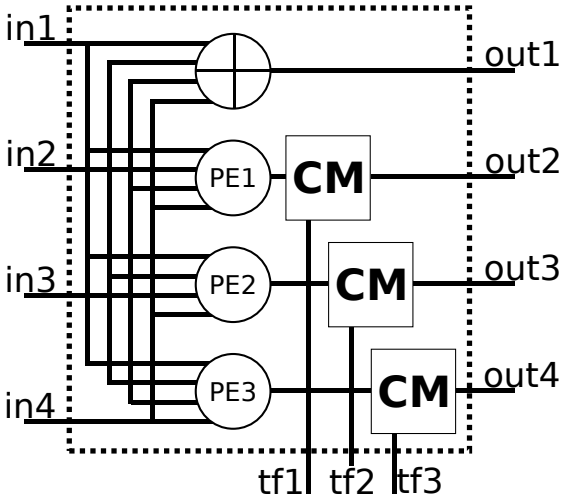


Figure 31: Butterfly component. The PE's correspond to equations 31b-31d.

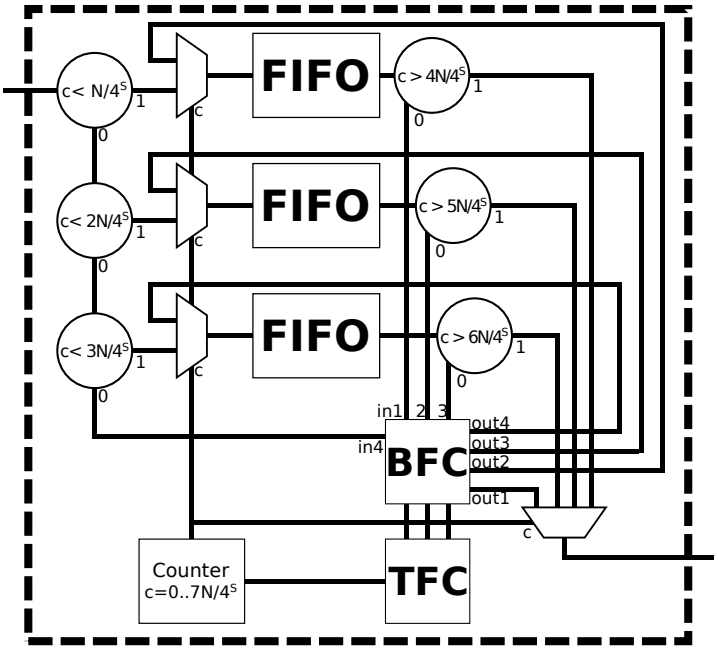


Figure 32: Schematic of a stage in the radix-4 design. The figure shows that the TFC has three outputs.

$$X_0 = x_0 + x_1 + x_2 + x_3 \tag{31a}$$

$$X_1 = ([\Re(x_0) + \Im(x_1) - \Re(x_2) + \Im(x_3)] + i[\Im(x_0) - \Re(x_1) - \Im(x_2) + \Re(x_3)]) \cdot W_N^n \tag{31b}$$

$$X_2 = ([\Re(x_0) - \Re(x_1) + \Re(x_2) - \Re(x_3)] + i[\Im(x_0) - \Im(x_1) + \Im(x_2) - \Im(x_3)]) \cdot W_N^{2n} \tag{31c}$$

$$X_3 = ([\Re(x_0) - \Im(x_1) - \Re(x_2) + \Im(x_3)] + i[\Im(x_0) + \Re(x_1) - \Im(x_2) + \Re(x_3)]) \cdot W_N^{3n} \tag{31d}$$

5.3.2 Variant 2 (NEWR4v2)

The NEWR4v2 implementation is an MDC architecture, implemented in the same style as the NEWv2 design. This design requires more memory than any of the other designs, and uses this memory to calculate the FFT in fewer clock cycles. It only takes 515 clock cycles to perform one FFT, which is about a third of the number of clock cycles in ASTRON's design.

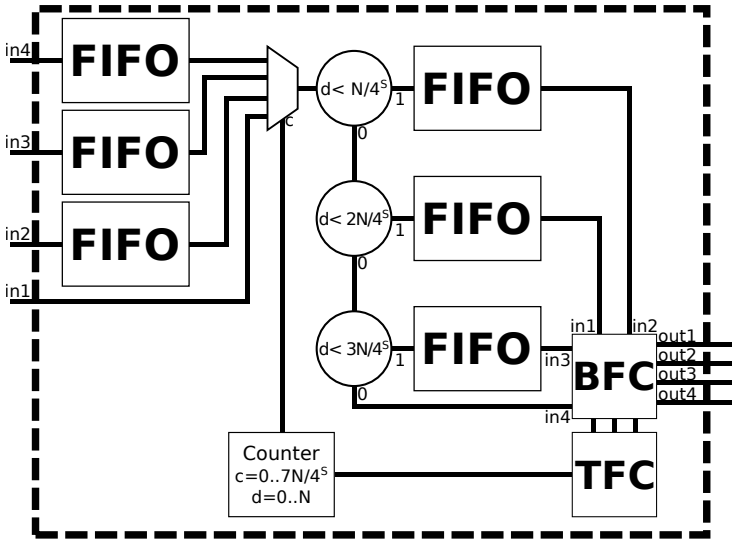


Figure 33: Schematic of a stage in the radix-4 design. The figure shows six FIFOs, which sum up to $15 \cdot (N/4^S)$ words of memory.

5.4 Synthesized combinations of components

There are 4 architectures, 3 complex multipliers and 3 twiddle factor components in the new designs. This gives 36 combinations which were not all implemented. Table 5 gives an overview of the implemented and synthesized combinations using NEWv2. NEWv1, R4NEWv1 and R4NEWv2 were only synthesized using the straightforward complex multiplier and the component where all the twiddle factors (TF) are stored.

CM \TFC	all TFs	selection of TFs	calculating TFs
Straightforward	X	X	X
Gauss	X		
Designware component	X		

Table 5: Overview of implemented and synthesized combinations using NEWv2.

6 FPGA versus ASIC using ASTRON's design

In this chapter, the FPGA and the ASIC results of ASTRON's implementation are compared with each other. The design was synthesized at 100MHz, the input signals are 8 bit wide and the output signals are 14 bits. The FPGA at ASTRON runs at 200MHz, but the timing analyzer showed that the timing requirements were not met. The clock frequency was lowered to make post-simulation possible.
Quartus was used for FPGA synthesis, the target device is the Stratix IV EP4SGX230KF40C3. This FPGA uses 40nm technology. For ASIC synthesis, Synopsys Design Compiler was used. The design was synthesized on 65nm technology.
The design uses a multiplier component[11] which is specific to the Stratix IV and cannot be synthesized for ASIC. Therefore the design was synthesized three times on ASIC; once using a similar component from the Designware library[12], and once without using a specific component but standard adders and multipliers. The design was also synthesized without using the additional pipeline delays (section 5.1).

6.1 Area

The area of an ASIC and an FPGA can barely be compared, because Quartus does not show the sizes of the components in squared millimeters, nor could this information be found in datasheets or elsewhere on the internet. Nevertheless, the results are shown here. The FPGA comes in a 40mm x 40mm package. A large part of the FPGA is however not used:

- Logic utilization : 2%
- Total block memory bits : 127,291 / 14,625,792 (<1%)
- DSP block 18-bit elements : 40 / 1,288 (3%)
- Total pins : 48/888 (5%)

The logic utilization reported by Quartus is an estimation of how full the device is. It covers everything except the block components.
At the input, 19 pins are used for the 8 bits real and imaginary input values and for the clock, reset and in_val signals. At the output, 29 pins are used for the 16 bits real and imaginary output values and the out_val signal. This gives a total of $19 + 29 = 48$ pins

The ASIC design is only 0.47mm^2 with the additional delays and 0.45mm^2 without them. The ASIC specific multiplier is about the same size as the FPGA specific multiplier.

Design:	power (mW)	energy / FFT (μJ)	area (mm^2)
FPGA	53.14	0.84	-
ASIC (with designware component)	34.10	0.54	0.47
ASIC (without designware component)	26.40	0.42	0.47
ASIC (without additional delays)	26.50	0.41	0.45

Table 6: Summary of FPGA v. ASIC using ASTRON's design.

6.2 Power and Energy

To compare the power requirements, the synthesized results were simulated with the same input values. These simulations resulted in Value Change Dumps (VCD), with which the synthesis tools can make accurate power estimations.
The FPGA tooling shows the power consumption of different parts of the FPGA. The 'Total Thermal Power Dissipation' is 940.33 mW, this includes static parts of FPGA. The actual power consumption of the FFT on the FPGA is 53.14mW. One FFT calculation is done after about 16μs, so the FPGA will have used 14.9μJ. The FFT itself will have used 0.84μJ.

[11] the altmult_add Megafunction : http://www.altera.com/literature/ug/ug_lpm_alt_mfug.pdf
[12] DW02_prod_sum : http://www.synopsys.com/dw/buildingblock.php

The ASIC power calculation is done before layout and floorplanning. Simulation results are more accurate after layout and floorplanning. However, for this research the only core dynamic power consumption is compared, which makes this approach sufficient.
The ASIC with the designware component requires 34.10mW on the same input data. That means 0.54μJ is required for one FFT. The version without the designware component requires 26.40mW which means 0.42μJ per FFT. The version without the delays requires 26.50mW, but takes only 1545 clock cycles to complete (instead of 1584). This makes 0.41μJ per FFT. This is a decrease in energy of about 50% by just switching to an ASIC from an FPGA.

To compare the area results in a more meaningful way and to factor out the static power inaccuracy, a wrapper was made that initialized several FFT components. The outputs of one FFT would become the input of the next. The idea was to fill the FPGA as much as possible. This approach failed, however, because of out of memory errors in both the FPGA and the ASIC tooling.

7 Comparison of ASTRON design with new designs

In this chapter, the ASIC results of ASTRON's implementation will be compared to the ASIC results of the new implementations which were discussed in sections 5.2-5.3. All of the designs were synthesized at 100MHz for consistency. In addition to that, the NEWv2 design was synthesized at 50MHz and the R4NEWv2 design at 25MHz. The input signals are 8 bit wide and the output signals are 14 bits. Synopsys Design Compiler was used to synthesize the designs, this was done on 65nm technology. All results are before layout and floorplanning, which is sufficient because the dynamic behaviour is compared.

7.1 Area

The area of the ASTRON's design is about 0.45mm^2 as shown in chapter 6. The NEWv1 design was synthesized with the straightforward multiplication and the TFC with all values stored. The area of this design was 0.62mm^2. The NEWv2 design was synthesized with many different components, but these components did not have a large affect on the total area. Using different combinations, the area of the FFT is between 0.94 and 0.97mm^2. The only significant change is seen when the compiled RAM component is used to store the twiddle factors. During a test where this RAM component was used in (only) the first stage, the area grew to 1.03mm^2. This is because the RAM component contains 512x16 bits and is of a fixed size. The register-based TFC in the first stage is also programmed with 512x16 bits, but this number is reduced to 64*16 bits[13] due to the compiler optimizations.
Table 7 shows the area's of the designs. For the NEWv2 design, the version with the straightforward CM and all of the TFs is used as a reference. In the table, the design names show which part deviates from the reference.
Two of the designs were also synthesized for 512-point FFTs. The ASTRON design (without delays) and the NEWv2 design become about 0.27mm^2 and 0.51mm^2 respectively.

7.2 Power and Energy

The power usage of ASTRON's FFT is a little over 26mW which results in a little over 0.4μJ per FFT. The NEWv1 design consumes only 21.78mW, which means 0.34μJ per FFT. What was said about the area of the NEWv2 design can also be said about the power consumption. The numbers stay between 32 and 33mW at 100MHz for all of the combinations of components. Table 7 shows the power usage and the energy needed for one FFT calculation. This table also shows the number of clock cycles required to calculate one FFT. This represents the number of clock cycles from when the first butterfly operation begins, until the last butterfly operations finishes.

Design:	area (mm^2)	Power (mW)	Energy (μJ)	clock cycles
ASTRON	0.45	26.50	0.41	1545
NEWv1	0.62	21.78	0.34	1553
NEWv2	0.97	32.54	0.34	1041
NEWv2 (compiled RAM)	1.03	32.78	0.34	1041
NEWv2 (small TFs)	0.97	32.94	0.34	1041
NEWv2 (calculated TFs)	0.97	32.75	0.34	1041
NEWv2 (Gauss CM)	0.96	32.05	0.33	1041
NEWv2 (designware CM)	0.97	32.87	0.34	1041
R4NEWv1	0.67	39.99	0.51	1287
R4NEWv2	1.30	80.73	0.42	515

Table 7: Results of the FFT designs. The components between the parentheses show where the designs deviate from the standard, which is straightforward multiplication and a TFC with all values stored.

The synthesis tool can show the power usage per component. This can show where the differences come from. Table 8 shows the results for some of the components. The numbers represent the usage of all of the instances of the components in the design. Most of the components occur 9 times in the design (in every stage except the last stage, which has no twiddle factor multiplication). The compiled TFC however, was only used in the first stage. So the results of the compiled RAM-based and the register-based TFC in the first stage are shown seperately.

[13]The exact number is hard to find in the synthesis tool. It is, however, possible to divide the area of the non-combinational part of the component by the area of the non-combinational part of another component, of which the content is known.

Design:	Power (mW)	Energy (μJ)
all TFs (registers)	0.22	0.002
small TFs	0.24	0.003
calculated TFs	0.31	0.003
CM for small TFs[14]	2.73	0.030
straightforward CM	2.33	0.024
Gauss CM	1.91	0.020
designware CM	3.58	0.037
Stage 1		
all TFs (registers)	0.04	0.000
all TFs (compiled)	0.59	0.006

Table 8: Power and energy comsumption of the components in the NEWv2 design

The FIFOs in the designs are Designware components which use RAM as memory. ASTRON's design uses register-based FIFOs. To find out how this affects the design, NEWv2 was also synthesized using register-based FIFOs. Table 9 shows the power usage per stage for RAM- and register based FIFOs. The tools unfortunately do not show the details of Designware components, so entire stages are used. The table shows that for memories smaller than or equal to $N/2^6 = 16$ words, registers are more efficient. For larger memories, RAM is more efficient.

Stage:	register-based (mW)	RAM-based (mW)
1	5.39	4.59
2	9.84	8.79
3	5.13	4.64
4	2.75	2.55
5	1.57	1.52
6	0.96	1.01
7	0.69	0.73
8	0.47	0.51
9	0.32	0.36

Table 9: Power and energy per stage in the NEWv2 design. Stage 10 does not use FIFOs.

Some of the designs were also synthesized at 50MHz and for 512-point FFTs and this gives some unexpected results. Table 10 gives an overview of the 512-point designs.

Design:	@ 100MHz :Power (mW)	Energy (μJ)	@ 50MHz : Power (mW)	Energy (μJ)
ASTRON	30.55	0.47	-	-
NEWv2 (Gauss CM)	34.36	0.36	17.26	0.36

Table 10: Power and energy comsumption of the 512-point designs

Table 11 gives an overview of the 1024-point designs that were synthesized at different clock speeds.

Design:	Power (mW)	Energy (μJ)
NEWv1 (50MHz)	21.43	0.67
NEWv2 (Gauss CM, 50MHz)	31.80	0.66
R4NEWv2 (25MHz)	20.59	0.42

Table 11: Power and energy comsumption of the 1024-point designs at lower frequencies

These numbers show that 512-point FFTs consume a bit more energy that 1024-point FFTs. And while a lower clock frequency gives about the same energy consumption for 512-point FFT, for 1024-point FFT it doubles.

7.3 Comparison using FOMs

Table 12 shows a comparison of the designs using the FOMs from chapter 4.

	Duration (μs)	P/B (mW)	FFT's/second($\cdot 10^3$)	FFT's/Joule
ASTRON	15,45	0,005	65	1626
NEWv1	15,53	0,004	64	1968
NEWv2 (Gauss CM)	10,41	0,006	96	1995
R4NEWv1	12,87	0,008	78	647
R4NEWv2	5,15	0,016	194	800
Designs from chapter 4				
[1]	143	0.4007	6.99	207
[2]	512	0.0160	1.95	704
[3]	168	0.0065	5.95	3803
[4]	3.2	0.0507	313	35519
[5]	3.2	0.1021	313	35265
[6]	2063	0.0007	0.485	5180
[7]	0.029	3.7875	34722	211220
[8]	23.79	0.4475	42	270

Table 12: Comparison of FOMs

[14] The CM component that is used in combination with the small TFs is based on the straightforward implementation, but has some additional logic to multiply with the correct value.

8 Discussion

In this chapter, the results from chapters 6 and 7 are discussed

8.1 FPGA versus ASIC

Earlier comparisons between FPGAs and ASICs have shown large differences in their outcome. In [25], a large number of designs are synthesized for FPGA and ASIC. Their results show that FPGAs use about 14 times more power than ASICs on average. However, depending on the design, the results vary from 5 to 52 times more power usage on an FPGA. A closer look at the results show, that when more Logic or RAM is used in the designs, the ratio's are generally smaller.

In [2], an FFT design is implemented for different lengths (16-64-256-1024-point FFTs). These designs were all synthesized on an FPGA and ASIC. They show reductions in power comsumption from 94% to 33%. So again we see; the larger their design the smaller the improvement in power consumption.

In [6], the 256-point FFT design was also synthesized on an FPGA and ASIC. This implementation requires about 6 times more power on the FPGA than on the ASIC.

Table 6 shows that the switch from FPGA to ASIC gives a reduction in power usage of about 50% for ASTRON's design. This is more than the 33% in [2] but less than the smaller design in [6]. Considering that the STRATIX IV FPGA uses 40nm technology and the ASIC uses 65nm technology, this is a very reasonable result.

8.2 Design

For the ASIC comparison, four new designs were made. NEWv1 is most similar to ASTRON's design and requires less power. This is mainly because the internal signals are 16 bits instead of 18 bits. But also because of the use of RAM-based FIFOs instead of register-based FIFOs. Table 9 shows that using RAM is more efficient for the larger memories, but using registers is more efficient for memories of 16 words and less.

Table 7 shows that the radix-4 designs consume more power than the radix-2 designs. When the NEWv1 and R4NEWv1 designs are compared, we see that even though they use the same amount of memory and about the same amount of arithmetic operations, R4NEWv1 uses far more power than the NEWv1. So much more, that the shorter calculation time of one FFT is not enough to match the energy consumption per FFT.

The NEWv2 and R4NEWv2 designs use the same technique to process more data in parallel. Because the radix-4 butterfly essentially already is a parallelism of the radix-2 butterfly, this effect is much larger in R4NEWv2 than in NEWv2. Table 7 shows that this gives R4NEWv4 the shortest calculation time, at the cost of a large area.

When we look at the differences between NEWv1 and NEWv2, we see that faster calculation can be exchanged for lower power consumption without it affecting the energy usage per FFT. However, when the faster calculation is performed at a lower clock speed, the power does not drop as much as expected. In table 10, we see that the 512-point designs do show a drop of about 50% when the clock frequency is halved.

When we look at the differences between the two radix-4 designs, we see that the faster calculation in R4NEWv2 makes the energy per FFT lower than in R4NEWv1. Table 11 shows that the R4NEWv2 also consumes about 25% of the power at 25% of the speed. It is unclear why the 1024-point radix-2 designs drop less than 1mW when the clock frequency is halved, when in other cases, the relation between power consumption and clock frequency is much closer.

When comparing all of these designs, the NEWv2 design gives best combination of low energy per FFT and fast calculation. The FOMs in table 12 also confirm this. When these FOMs are compared with the other works in table 4, we see that designs [4], [5] and [7] perform better. However these designs are all made for small sized FFTs, 64, 64 and 16 point respectively. The other 1024-point FFTs and even one of the smaller sized FFTs[15], are outperformed by the new designs.

[15] this design, [8], is made for 16-point, but uses floating point arithmetic.

8.3 Components

Tables 7 and 8 show that the choice in components is of less importance than the choice in architecture. Although the differences between the components are small, they do exist. The tables show that the TFC with all of the values is the best choice. The compiler performs optimizations and minimizes the number of registers in these components. Because the small TFC has extra control logic, it ends up being larger than the TFC with all the components. This small TFC approach might have worked better when only RAM components were used to store the values in, because memory in a RAM component will not be optimized out of the design.
The Designware components that were used for both the CM and the TFC turned out to be less effective than the more straightforward implementations of these components. In case of the TFC, the optimizations by the compiler make the Designware component in just one stage consume more power than all stages togethers using one of the other methods.
For the complex multiplier Gauss' trick works best. Reducing the number of multiplications does have its effect on power consumption.

9 Conclusion

This research had two goals. The first goal was to find out how much the difference would be between the implementation of the FFT on an FPGA and on an ASIC. The following conclusions can be made:

- Area is very difficult to compare, however, the ASIC design shows a large reduction in size compared to FPGA.
- The difference in dynamic power consumption depends greatly on which design is used. The difference seem larger for smaller designs.
- For ASTRON's design, there is a 50% reduction in power consumption, while the ASIC technology is one step behind.

The second goal was to find how the design could be improved to be more energy efficient. The following conclusions can be made:

- A radix-2 design is more energy efficient than a radix-4 design. The NEWv1 design consumes 33% less energy than R4NEWv1, NEWv2 requires about 20% less energy than R4NEWv2.
- By using parallelism the NEWv2 design can calculate FFTs quicker, without increasing the amount of energy per FFT.
- Using Gauss' complex multiplication trick reduces power consumption compared to standard complex multiplication by about 18%.
- Storing the twiddle factors in registers is more efficient than storing them in RAM. This is partly because of compiler optimizations.
- For memory smaller than or equal to 16 words, registers are more efficient than RAM. For larger memories, registers will become more inefficient compared to ram.
- The improvements of twiddle factor and complex multiplier components are trivial compared to improvements in the architecture.
- The design that requires the least amount of energy, is the NEWv2 design with the Gauss complex multiplier and the straighforward twiddle factor component.

9.1 Recommendations & Future Work

The conclusions leave some unanswered questions, which could be addressed in a future research:

- NEWv1 is more energy efficient than R4NEWv1, even though they use about the same amount of memory and about the same amount of arithmetic operations. R4NEWv1 also requires fewer clock cycles, so why is NEWv1 more energy efficient?
- In section 3.3, it was mentioned that MDC designs can compute multiple FFTs at the same time. Can NEWv2 and R4NEWv2 be made more energy efficient by applying this technique?
- For some of the designs, reducing the frequency by 50% also reduces the power consumption by about 50%, but not for all. Why?
- What would the effects of raising the frequency be?
- Would a split-radix, radix-2^2 or radix-2^3 architecture be any better?

List of abbreviations

AGB	Address Generation Block
ASIC	Application-Specific Integrated Circuit
BFC	Butterfly Component
CM	Complex Multiplier
CMC	Coefficient Memory Cluster
CMP	Circuit Multi-Projets
CORDIC	Coordinate Rotation Digital Computer
CSD	Canonical Sign Digit
DFT	Discrete Fourier Transform
DIF	Decimation In Frequency
DIT	Decimation In Time
DMC	Data Memory Clusters
FFT	Fast Fourier Transform
FIFO	First In, First Out
FOM	Figure Of Merit
FPGA	Field-Programmable Gate Array
FT	Fourier Transform
HBA	High Band Antennas
IFFT	Inverse FFT
LBA	Low Band Antennas
LOFAR	Low Frequency Array
MAC	Multiply And Accumulate
MDC	Multi-path delay communicator
MVR	Matrix Vector Rotation
NEWR4v1	New radix-4 SDF FFT design
NEWR4v2	New radix-4 MDC FFT design
NEWv1	New radix-2 SDF FFT design
NEWv2	New radix-2 MDC FFT design
PE	Processing Element
R2^2SDF	Radix-2^2 implementation using a SDF architecture
R2MDC	Radix-2 implementation using a MDC architecture
R2SDF	Radix-2 implementation using a SDF architecture
R4MDC	Radix-4 implementation using a MDC architecture
R4SDC	Radix-2 implementation using a SDC architecture
R4SDF	Radix-4 implementation using a SDF architecture

RAM	Random-access memory
RCU	Receiver Unit
ROM	Read-only memory
RSP	Remote Station Processing
SDC	Single-path delay communicator
SDF	Single-path delay feedback
SKA	Square Kilometer Array
SNR	Signal-to-noise ratio
TBB	Transient Buffer Boards
TF	Twiddle Factor
TFC	Twiddle Factor Component
VCD	Value Change Dumps

References

[1] Guihua Liu and Quanyuan Feng. ASIC design of low-power reconfigurable FFT processor. In *ASIC, 2007. ASICON '07. 7th International Conference on*, pages 44 –47, Oct. 2007.

[2] Y. Zhao, A.T. Erdogan, and T. Arslan. A low-power and domain-specific reconfigurable FFT fabric for system-on-chip applications. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, page 4 pp., April 2005.

[3] I. Hatai, R. Biswas, and S. Banerjee. Asic implementation of a 512-point FFT/IFFT processor for 2d ct image reconstruction algorithm. In *Students' Technology Symposium (TechSym), 2011 IEEE*, pages 220 –225, Jan. 2011.

[4] M.N. Khan, M. Mohamed Ismail, and P.K. Jawahar. An efficient FFT/IFFT architecture for wireless communication. In *Communications and Signal Processing (ICCSP), 2012 International Conference on*, pages 66 –71, April 2012.

[5] A. Anbarasan and K. Shankar. Design and implementation of low power FFT/IFFT processor for wireless communication. In *Pattern Recognition, Informatics and Medical Engineering (PRIME), 2012 International Conference on*, pages 152 –155, March 2012.

[6] Nie Zedong, Zhang Fengjuan, Li Jie, and Wang Lei. Low-power digital ASIC for on-chip spectral analysis of low-frequency physiological signals. *Journal of Semiconductors*, 33(6):065004, 2012.

[7] Akshay Sridharan and A. Viji. Low power hardware implementation of high speed FFT core. In *Advances in Computer Engineering (ACE), 2010 International Conference on*, pages 223 –227, June 2010.

[8] P. Saha, A. Banerjee, A. Dandapat, and P. Bhattacharyya. ASIC implementation of high speed processor for calculating discrete fourier transformation using circular convolution technique. *WSEAS Trans. Cir. and Sys.*, 10(8):278–288, August 2011.

[9] J.W. Cooley and J.W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19:297–301, 1965.

[10] J.G. Proakis and D.G. Manolakis. *Digital signal processing: Principles, Algorithms, and Applications*. Pearson Prentice Hall, 2007.

[11] Square kilometer array. `http://www.skatelescope.org`. accessed 20 Februari 2013.

[12] Wikipedia, Cooley-Tukey FFT algorithm. `http://en.wikipedia.org/wiki/Cooley-Tukey_FFT_algorithm`. accessed 10 December 2012.

[13] P. Duhamel and H. Hollmann. 'split radix' fft algorithm. *Electronics Letters*, 20(1):14–16, 1984.

[14] Split-radix FFT algorithms. `http://cnx.org/content/m12031/latest/`. accessed 15 April 2013.

[15] C.S. Burrus. Index mappings for multidimensional formulation of the DFT and convolution. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 25(3):239–242, 1977.

[16] Shousheng He and M. Torkelson. A new approach to pipeline FFT processor. In *Parallel Processing Symposium, 1996., Proceedings of IPPS '96, The 10th International*, pages 766–770, 1996.

[17] David Kampen van, Klaas L. Hofstra, Jordy Potman, and Sabih H. Gerez. Implementation of a combined OFDM-demodulation and WCDMA-equalization module. In *Proceedings of ProRISC 2006, 17th Annual Workshop on Circuits, Systems and Signal Processing*, pages 277–285, Veldhoven, The Netherlands, 2006.

[18] B.M. Baas. A low-power, high-performance, 1024-point FFT processor. *Solid-State Circuits, IEEE Journal of*, 34(3):380 –387, Mar 1999.

[19] Shousheng He and M. Torkelson. Design and implementation of a 1024-point pipeline FFT processor. In *Custom Integrated Circuits Conference, 1998. Proceedings of the IEEE 1998*, pages 131 –134, May 1998.

[20] C. Dick. Computing the discrete fourier transform on FPGA based systolic arrays. In *Field-Programmable Gate Arrays, 1996. FPGA '96. Proceedings of the 1996 ACM Fourth International Symposium on*, pages 129–135, 1996.

[21] Sergio Saponara, Massimo Rovini, Luca Fanucci, Athanasios Karachalios, George Lentaris, and Dionysios Reisis. Design and comparison of FFT VLSI architectures for SoC telecom applications with different flexibility, speed and complexity trade-offs. *Circuits, Systems, and Signal Processing*, 31:627–649, 2012.

[22] Min-An Song, Lan-Da Van, and Sy-Yen Kuo. Adaptive low-error fixed-width booth multipliers. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, E90-A(6):1180–1187, June 2007.

[23] B.M. Baas. An energy-efficient single-chip FFT processor. In *VLSI Circuits, 1996. Digest of Technical Papers., 1996 Symposium on*, pages 164 –165, Jun 1996.

[24] E. Lai. *Practical Digital Signal Processing*. Elsevier Science, 2003.

[25] Ian Kuon and Jonathan Rose. Measuring the gap between FPGAs and ASICs. In *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, FPGA '06, pages 21–30, New York, NY, USA, 2006. ACM.