Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in Electronic Engineering

Studio e realizzazione di un'architettura VLSI di un processore per l'implementazione dell'algoritmo FFT

Study and realization of a VLSI architecture of a processor implementing the FFT algorithm

Master Candidate

Lorenzo Lastrucci

Student ID 2018716

Supervisor

Prof. Daniele Vogrig

University of Padova

Academic Year
2021/2022

Abstract

Since the 5G connection standard is utilized by a rising number of devices and is evolving to meet new needs and requirements, it has become crucial to study and design new, faster, and more efficient transmitters and receivers. A fundamental role in the 5G connection is played by Orthogonal frequency-division multiplexing (OFDM), an encoding methodology. Since the demodulation is based on the Fourier Transform, the purpose of this thesis is to realize a processor capable of implementing FFT and DFT algorithms on variable length sequences that complies with the 5G standard criteria. In order to do so, first an analysis of the International Telecommunication Union report ITU-R M.2410-0 has been conducted to define the minimum requirements for the processor. Then, a study of the state of the art for similar devices led to the development of a VLSI architecture suitable for the application. An RTL version of the architecture has been implemented in VHDL and tested.

Contents

List of Acronyms

3GPP	3rd Generation Partnership Project
5G NR	5G New Radio
CORDIC	Coordinate Rotation Digital Computer
DFT	Discrete Fourier Transform
eMBB	enhanced Mobile Broadband
FD-SOI	Fully-Depleted Silicon-on-Insulator
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GSM	Global System for Mobile Communications
ICT	Information and communication technology
IEEE	Institute of Electrical and Electronics Engineers
IMT-2020	International Mobile Telecommunications - 2020
ITU	International Telecommunication Union
ITU-R	ITU Radiocommunication sector
MDF	Multi-Path Delay Feedback
MRA	Mixed-Radix Algorithm
No.Stage	number of stages

OFDM	Orthogonal frequency-division multiplexing
PEA	Processing Element A
PEB	Processing Element B
PEC	Processing Element C
PAL	Programmable Array Logic
PLD	Programmable Logic Device
RPK	Re-configurable Processing Kernel
RqOnLtcy	Requirement on latency
RqOnThrpt	Requirement on throughput
RTL	Register Transfer Level
SDF	Single-Path Delay Feedback
STA	Static Time Analysis
TFMUL	Twiddle Factor Multiplier
VHDL	VHSIC Hardware Description Language
VHSIC	Very High-Speed Integrated Circuit

1

Introduction

The fifth-generation technology for broadband cellular networks is referred to as "5G" in the telecommunication sector. The agency that takes care of the definition of these standards is the International Telecommunication Union (ITU). The ITU is a United Nations specialized agency, founded in 1865 to facilitate international connectivity in communications networks: it allocates global radio spectrum and satellite orbits, develops the technical standards that ensure networks and technologies seamlessly interconnect and works to improve access to Information and communication technologies (ICTs) to underserved communities worldwide. [1] The sector that works on the 5G standard is the ITU Radiocommunication sector (ITU-R): this is the sector that allocates global radio spectrum and satellite orbits and develops standards for radiocommunication systems with the objective of ensuring the effective use of the spectrum. In 2015 the ITU-R issued a set of requirements for 5G networks called International Mobile Telecommunications - 2020 (IMT-2020) standard. While the final version of the standard was published on February 1st, 2021, most of it was completed earlier. The report M.2410-0, published in November 2017, is of particular interest to this thesis because it describes key requirements related to the minimum technical performance of IMT-2020 candidate radio interface technologies.[1] The main requirements taken into account are:

- Latency up to 4 ms for enhanced Mobile Broadband (eMBB)
- Peak data rate 20 Gbit/s in Downlink and 10 Gbit/s in Uplink (eMBB)

[1]`https://www.itu.int/en/about/Pages/default.aspx`

- Peak spectral efficiency 30bit/s/Hz in Downlink and 15 bit/s/Hz in Uplink
- Bandwidth 100MHz

A description and standardization of the requirements for 5G systems have been carried out by the 3rd Generation Partnership Project (3GPP), a consortium of various national and regional telecommunications standard organizations that develops protocols for mobile telecommunications. The 3GPP was constituted in 1998 to first produce the 3G mobile phone system based on the 2G Global System for Mobile Communications (GSM) system. With its Release 15 of 2018, the 3GPP has defined the specifics of the 5G New Radio (5G NR) standard. The standard has been further described in Releases 16 and 17, with the latter published during the first half of 2022. The most relevant series of specifications for this work of thesis is the 38, which provides technical details about the standard. In particular, the 38.211 technical specification has been important since it describes the structure of physical channels and the modulation of the standard.[2] This report describes in detail the frame structure and the Orthogonal frequency-division multiplexing (OFDM) symbols used by the 5G NR standard and these specifics were used as a starting point to define a set of specifications for the processor realized since the OFDM requires both high-speed power-of-two Fast Fourier Transform (FFT) and non-power-of-two Discrete Fourier Transform (DFT) to support the high data rate of the 5G NR standard.

1.1 Orthogonal frequency-division multiplexing

The OFDM is an encoding methodology used in digital modulation to encode data on multiple carriers: in particular, the bit-stream of data to be sent is divided into multiple closely spaced orthogonal subcarrier signals with overlapping spectra and then transmitted. The main advantages of this modulation methodology are its ability to cope with non-optimal channel conditions and efficient implementation using FFT. In fact, the orthogonality ensures that the cross-talk is eliminated and inter-carrier bands are not required, allowing for efficient modulator and demodulator implementation using FFT algorithm on the receiver side and inverse FFT of the sender side. This type of implementation requires both a fast execution of the algorithm and high flexibility on the Fourier transform length. This means that an FFT processor, to be functional in this application, should have these characteristics:

- FFT size up to 4096
- Butterfly unit(s) supporting 2-,3-,5- and higher radices
- Twiddle Factor multiplication scheme with hardware efficiency
- Conflict-free data access scheme that supports multiple butterfly units for 2-, 3-, 5-, and higher radices as well as minimizes the memory usage for non-power-of-two DFTs

1.2 Fast Fourier Transform Algorithm

The Fast Fourier Transform (FFT) is an algorithm used to compute the Discrete Fourier Transform (DFT), the decomposition of the sequence of samples of a signal into its frequency components, defined as:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi nk}{N}}, \qquad k = 1, \dots, N \tag{1.1}$$

In particular, the FFT reduces the computation complexity of the DFT, and consequently the time required to carry out the computation, from $O(N^2)$ to $O(N \log N)$, where N is the data sequence length. While the fast algorithms for the DFT computation are diffused since 1805, the generic FFT algorithm is credited to James Cooley and John Tukey, who published it in 1965, after independent studies. This algorithm is based on the recursive breakdown of the DFT into smaller ones: an N-length sequence, with $N = N_1 N_2$, can be split into two smaller N_1 and N_2 DFTs and a series of complex multiplications (Fig. 1.1)[2] by the roots of unity, later called Twiddle Factors, reducing the complexity of the computation; this process can be repeated recursively, reducing at each step the length of the DFTs. The most common Cooley-Tukey algorithm consists in halving the DFT length at each step:

$$X[k] = \sum_{m=0}^{N/2-1} x[2m] e^{-j\frac{2\pi(2m)k}{N}} + \sum_{m=0}^{N/2-1} x[2m+1] e^{-j\frac{2\pi(2m+1)k}{N}} \tag{1.2}$$

[2]By Yangwenbo99 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=111271197

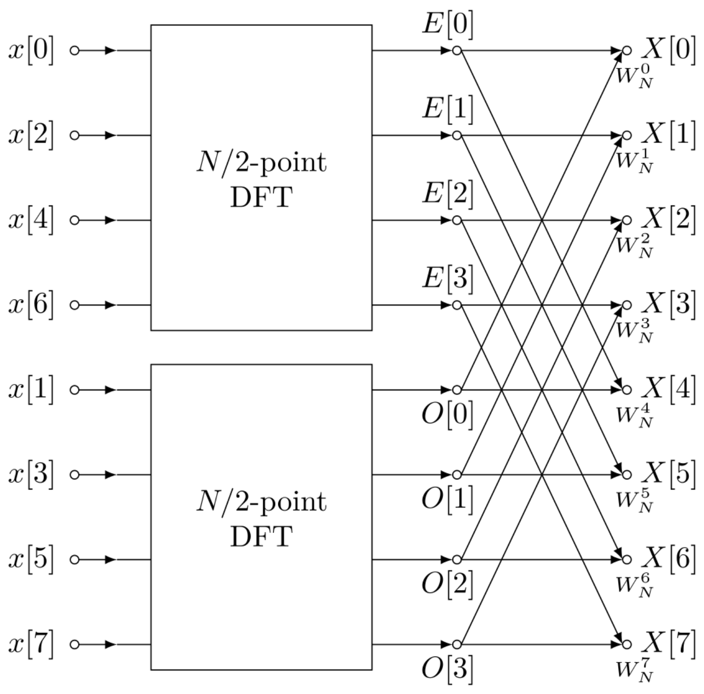


Figure 1.1: FFT algorithm scheme

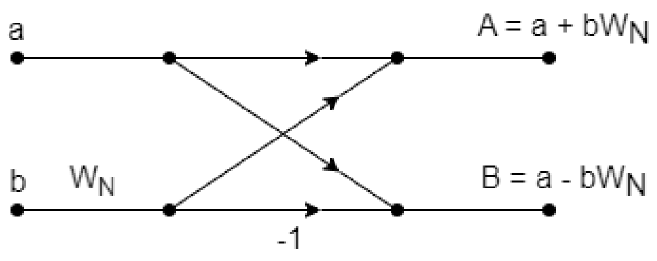


Figure 1.2: Radix-2 Butterfly

until they reach length two: this is known as the radix-2 (Fig. 1.2) algorithm and is possible only for power-of-two sizes. A more flexible algorithm is the Mixed-Radix: the DFT is factorized using various radix to extend the applicability to non-power-of-two sizes.

1.3 Mixed-Radix Algorithm

The processor's architecture is based on the Mixed-Radix Algorithm [3], an algorithm that divides the classic DFTs algorithm into a cascade of shorter DFTs.

If we consider the DFT definition:

$$\begin{cases} X[k] = \sum\limits_{n=0}^{N-1} x[n] W_N^{nk} \\ W_N^{nk} = e^{-j\frac{2\pi nk}{N}} \end{cases} \tag{1.3}$$

starting from the mapping of the mono-dimensional array $N = N_1 N_2$ into a two-dimensional array N_1 by N_2 [4], it's possible to map n and k, indices identifying input and output sequences of the DFT algorithm, as:

$$\begin{cases} n = (N_2 n_1 + A_2 n_2) \bmod N, & n_1, k_1 = 0, 1, \ldots, N_1 - 1 \\ k = (B_1 k_1 + N_1 k_2) \bmod N, & n_2, k_2 = 0, 1, \ldots, N_2 - 1 \end{cases} \tag{1.4}$$

where A_2 and B_1 are two coefficients that depend on the relation between N_1 and N_2. The DFT algorithm can be rewritten as:

$$\begin{aligned} X[k_1, k_2] &= \sum_{n_2} \sum_{n_1} x[n_1, n_2] W_{N_1}^{n_1 k_1} W_{N_2}^{n_2 k_2} W_N^{n_2 k_1} \\ &= \sum_{n_2} \left\{ W_N^{n_2 k_1} \sum_{n_1} x[n_1, n_2] W_{N_1}^{n_1 k_1} \right\} W_{N_2}^{n_2 k_2} \\ &= \sum_{n_2} W_N^{n_2 k_1} y[k_1, n_2] W_{N_2}^{n_2 k_2} \end{aligned} \tag{1.5}$$

where the term $W_N^{n_2 k_1}$ is the Twiddle Factor. If N_1 and N_2 are relatively prime, A_2 and B_1 are chosen to satisfy these conditions:

$$A_2 = p_1 N_1 \quad \text{and} \quad A_2 = q_1 N_2 + 1 \tag{1.6}$$

$$B_1 = p_2 N_2 \quad \text{and} \quad B_1 = q_2 N_1 + 1 \tag{1.7}$$

the Twiddle Factor can be simplified, and we obtain the relation:

$$\begin{aligned} X[k_1, k_2] &= \sum_{n_2} \sum_{n_1} x[n_1, n_2] W_{N_1}^{n_1 k_1} W_{N_2}^{n_2 k_2} \\ &= \sum_{n_2} \left\{ \sum_{n_1} x[n_1, n_2] W_{N_1}^{n_1 k_1} \right\} W_{N_2}^{n_2 k_2} \\ &= \sum_{n_2} y[k_1, n_2] W_{N_2}^{n_2 k_2} \end{aligned} \tag{1.8}$$

If we reiterate this process, splitting each time N_1 and N_2 into smaller length

DFT, we can map the indices of the DFT of length N to:

$$\begin{cases} N = \prod_{j=1}^{S} N_j \\ n = \sum_{m=1}^{S-1} \left(\prod_{n=m+1}^{S} (N_n) n_m \right) + n_S \\ k = k_1 + \sum_{m=2}^{S} \left(\prod_{n=1}^{m-1} (N_n) k_m \right) \end{cases} \tag{1.9}$$

an S stages sequence, where N_i, i=1,2,...,S, is the radix number of the i^{th} stage, and $n_i \in [0, 1, \ldots, N_i - 1]$ and $k_i \in [0, 1, \ldots, N_i - 1]$ are the corresponding sequence numbers of the input and output data in the i^{th} stage with i=1,2,...,S. At each stage we can compute the radix-N_i DFT as:

$$x_i\left[k_i\right] = \begin{cases} \sum_{n_i=0}^{N_i-1} x\left[n_i\right] W_{N_i}^{n_i k_i} & (i=1) \\ \sum_{n_i=0}^{N_i-1} \left(W_{\prod_{j=1}^{i} N_j}^{\left(k_1 + \sum_{m=2}^{i-1} \left(\prod_{n=1}^{m-1} (N_n) k_m \right) \right) n_i} x_{i-1}\left[n_i\right] W_{N_i}^{n_i k_i} \right) & (\text{else}). \end{cases} \tag{1.10}$$

With the exception of the first, each stage consists of computing the N_i-length DFT where each term of the sum is multiplied by a Twiddle Factor that is calculated over the product $N_{tf} = \prod_{j=0}^{i} N_j$, lengths of all previous stages plus the current one.

2

State of the Art

This chapter is dedicated to exploring possible implementation and architecture in literature, starting from deriving specifics for processor performances.

2.1 Specifics derivation

As was mentioned before, the specifics and characteristics of the processor are obtained starting from the requirements described in the technical report M.2410 and the technical specification report 38.211. The first consideration is about the throughput of the processor: considering the minimum bandwidth of 100 MHz as per requirements in [1], we have a Requirement on throughput ($RqOnThrpt$) of:

$$RqOnThrpt = 100MS/s \tag{2.1}$$

If we then consider a maximum FFT length of 4096, we can evaluate the Requirement on latency ($RqOnLtcy$) on a single data stream:

$$RqOnLtcy = \frac{4096}{RqOnThrpt} = 40.96\mu s \tag{2.2}$$

small enough to not be comparable with the maximum latency of 4 ms specified in the *Minimum requirements related to technical performance for IMT-2020 radio interface(s)* for enhanced Mobile Broadband (eMBB).

2.2 Architecture choice

The first design choice to be made is the architecture to be implemented: there are various efficient FFT architectures, capable of high data throughput, based on the Cooley-Turkey algorithm, but not all of them support multiple FFT sizes and some are not capable of supporting non-power-of-two DFTs. To implement the full Mixed-Radix algorithm and obtain a good versatility of the design is necessary to take into consideration three types of architecture:

- Single-Path Delay Feedback (SDF)
- Multi-Path Delay Feedback (MDF)
- Memory-Based architectures

In this section, a brief description of each type of architecture will be presented and their viability for the scope of this thesis will be considered.

2.2.1 Single-Path Delay Feedback

The Single-Path Delay Feedback (SDF) (Fig. 2.1) is a pipelined architecture based on a radix-r block, multipliers for the twiddle factors and N-1 registers (the minimum amount possible for an N-point FFT), making this architecture really efficient in terms of area occupation.

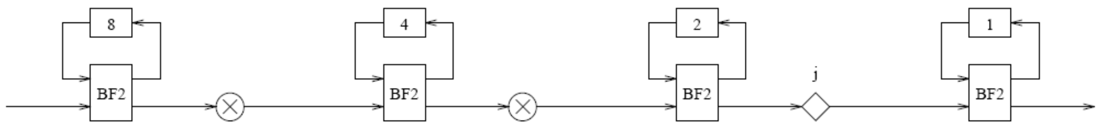


Figure 2.1: Simple Single-Path Delay Feedback architecture for an FFT of size 16

While this simple implementation is quite limited in versatility, it can be improved by substituting the radix-r block with a Re-configurable Processing Kernel (RPK) [5][6], a configurable block that can be controlled by the Control Unit: this means that a large number of FFT sizes can be supported by SDF architectures. Moreover, the versatility is greatly reduced if we consider non-power-of-two DFTs. Finally, the main problem of this implementation is the maximum throughput achievable: as the name self explains, the single path limits the throughput to a sample per clock rate. While this may be enough to reach the minimum requirements for 5G NR, it might not be future-proof as these requirements become more demanding.

2.2.2 Multi-Path Delay Feedback

The Multi-Path Delay Feedback (MDF) is a pipelined structure that divides the input sequence into multiple parallel streams: the structure, as for SDF structures, is based on a radix-r block, twiddle factors multipliers and registers to synchronize the data streams. This type of architecture solves the reduced throughput of the SDF designs, while still not solving the versatility problem: if we consider the MDF design in the paper “Power and Area Minimization of Reconfigurable FFT Processors: A 3GPP-LTE Example”[7], it supports only 1536-point DFTs and extending this support to other lengths it’s not easily doable.

2.2.3 Memory-Based architectures

Memory-Based architectures are mainly based on the utilization of memory: they are usually composed of a Memory block, a Control Unit and Processing Element or Butterfly Unit (Fig. 2.2).

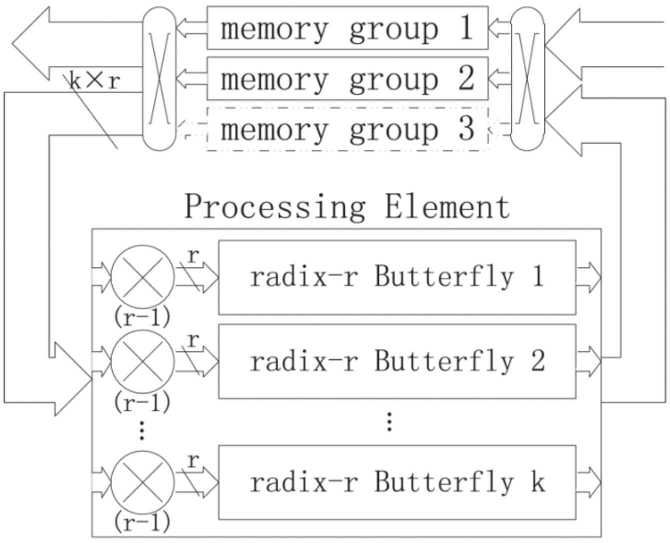


Figure 2.2: Memory-Based architecture [8]

Memory-Based architectures are of two types:

- Single Memory architecture
- Dual Memory architecture

The former only uses one memory block connected with the Processing Element by a bidirectional bus: at each stage is necessary both an output and input phase between two computation stages, increasing the latency of the total computation; the latter uses the two memories alternatively as input or output memory, reducing the latency. This type of implementation will solve both the throughput problem, as the processing element can be realized to work on multiple streams, and the versatility problem, as it's easier to implement radix-3 or radix-5 into General radix-r Butterfly Units. An example of this implementation is presented in "A High-Flexible Low-Latency Memory-Based FFT Processor for 4G, WLAN, and Future 5G"[8]: this paper describes in detail a memory-based FFT processor for telecommunication applications and it has been the main inspiration for this thesis. These are the reasons that lead to the decision of realising a Memory-Based architecture.

Finally, based on the analysis conducted in this paper [8] on the estimated hardware cost, while the throughput could be higher with a Continuous-flow processor, the choice of a Non-Continuous-flow processor with a radix-16 Butterfly Unit has been made, as it should result in a more efficient architecture while being able to satisfy the throughput and latency requirements.

2.3 Throughput estimation

If we consider a processor based on a radix-r Butterfly Unit, we can evaluate the number of stages ($No.Stage$) necessary to compute an N-point FFT as:

$$No.Stage = \frac{\log_2 N}{\log_2 r} \tag{2.3}$$

rounded up to the closer integer. Each stage will need a number of clocks:

$$Clk_{stage} = \frac{N}{k \cdot r} \tag{2.4}$$

where k is the number of Butterfly Units in the processor. In the case of a Non-Continuous-flow processor, to evaluate the throughput we need to take into account the non-computing clock cycle for I/O phases between two data streams. For an N-point FFT, we have to consider a number of clock cycles for I/O equal to:

$$Clk_{I/O} = 2\frac{N}{k \cdot r} \tag{2.5}$$

This value actually can be halved if the processor is used to compute in sequence multiple N-point FFTs Combining the equations (2.3), (2.4), (2.5), the total throughput can be estimated as:

$$Thrpt = No.Stage \cdot Clk_{stage} + Clk_{I/O} \tag{2.6}$$

that should be less or equal to $\frac{N}{RqOnThrpt} \cdot Clk_{freq}$, the number of clock cycles derived from the Requirement on throughput (RqOnThrpt). So, if we consider a 4096-point FFT on a one radix-16 Butterfly Unit architecture working at 500 MHz, this is the result:

$$\begin{aligned} Thrpt &= \frac{\log_2 4096}{\log_2 16} \cdot \frac{4096}{16} + 2 \cdot \frac{4096}{16} = \\ &= 1280\ Clk_{cycles} \quad \leq \quad 10240\ Clk_{cycles} \end{aligned} \tag{2.7}$$

3 Architecture design

The architecture of the processor is obtained starting from a memory-based architecture with a configurable radix-16 butterfly unit. As it can be seen in figure 3.1, the processor has a Control Unit that schedules and controls the processor operations, a couple of Addresses Generators that generate input and output addresses for the memory, a memory block that stores data during the operations, a Butterfly Unit and a Twiddle Factor Multiplier which execute the actual operations. This architecture grants the processor both versatility and high operation speed as it has a pipeline structure and the possibility to execute in parallel multiple radix-r operations.

3.1 RTL design of the FFT processor

This section describes the Register Transfer Level (RTL) design of the processor (Fig. 3.1), implemented using VHSIC Hardware Description Language (VHDL), a hardware description language developed by the United States Department of Defense in 1985 as part of the program Very High-Speed Integrated Circuit (VHSIC) and afterward standardized by the Institute of Electrical and Electronics Engineers (IEEE).

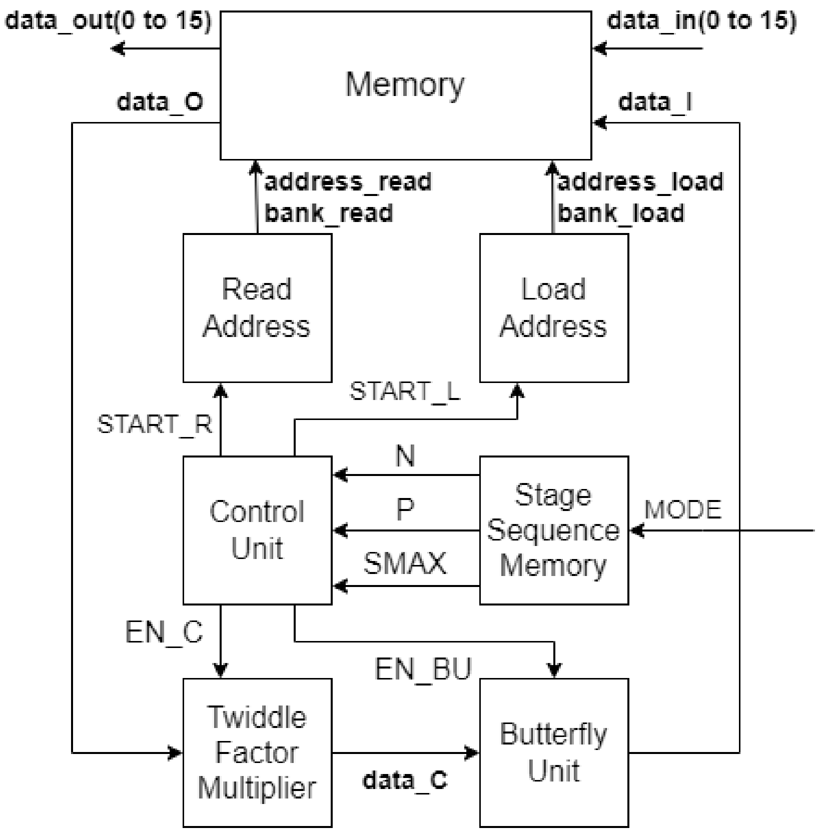


Figure 3.1: FFT processor block scheme

3.1.1 Control Unit

The Control Unit (Fig. 3.2) it's the central block of the processor that generates all the control signals needed by the other blocks. It is implemented as a Finite State Machine (Fig. 3.3).

Other than the Finite State Machine (FSM), the Control Unit is formed by a block that generates addresses for the Twiddle Factor Multiplier (TFMUL) memory and one that based on the current stage reorders the stage sequence.

Finite State Machine Operations

The FSM waiting state is the **IDLE** state: the Control Unit is waiting for an external **START** signal to start its operations while maintaining all the components of the processor deactivated through the control signals $\mathbf{START_L}$, $\mathbf{START_R}$, $\mathbf{EN_{BU}}$, $\mathbf{EN_C}$ and **SEL**, all active high; the **SETUP** state is used by the Control Unit as a transitional state to signal that the processor it's ready to receive the input data, while it activates the Load Address Generator through the $\mathbf{START_L}$

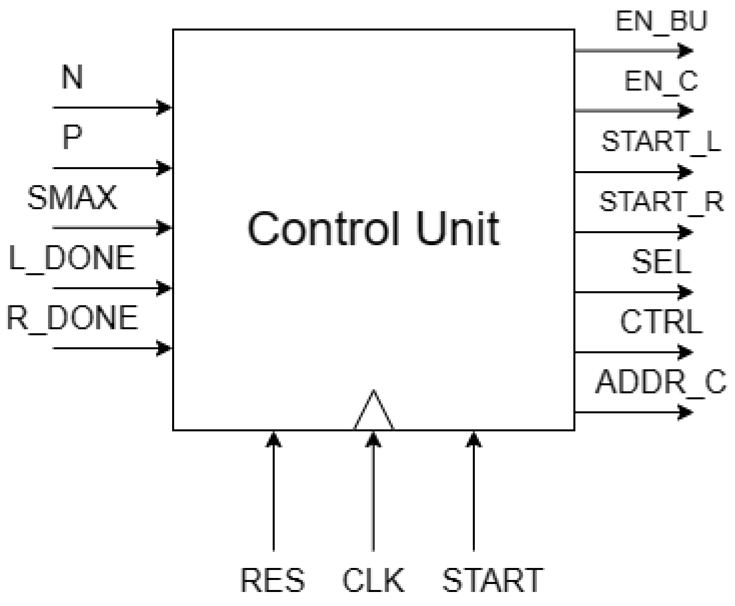


Figure 3.2: Control Unit Block

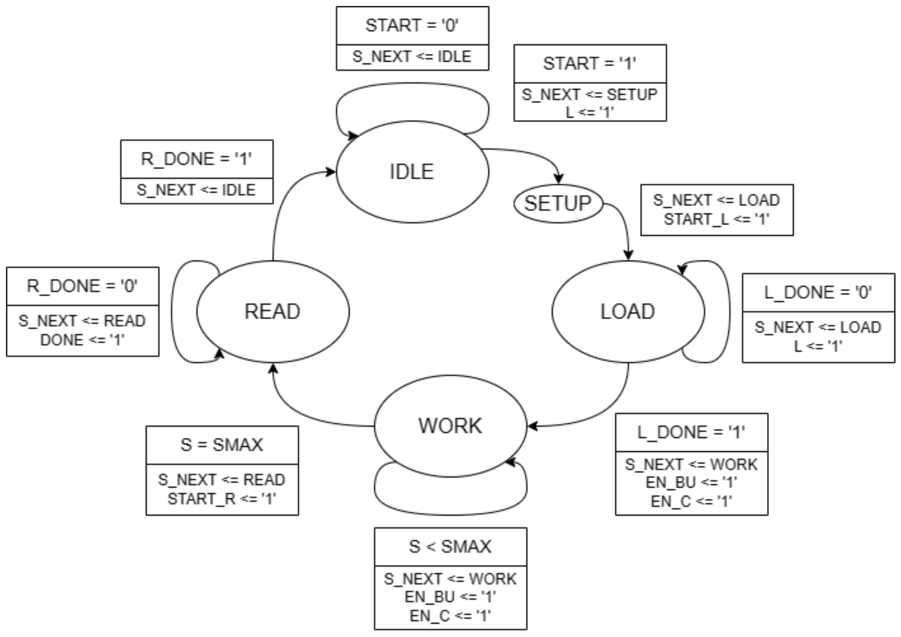


Figure 3.3: Control Unit Finite State Machine

signal; while in the **LOAD** state, the Control Unit waits for the Load Address Generator **DONE** signal: the processor is loading the input data in its memory; the **WORK** state is the core of the processor operations: while the Control Unit is in this state, the actual DFT computation is being carried out and the Control Unit uses the control signals to menage the other blocks; when all the stages DFTs are computed, the Control Unit enters the **READ** state and activates the Read Address Generator: the processor presents at its output the computed DFT sequence.

```vhdl
process(START,S_NOW,S,L_DONE,R_DONE)
begin
    S_NEXT <= IDLE;         --  FSM state
    L <= '0';               --
    EN_BU <= '0';           --
    EN_C <= '0';            --    Control signals
    START_L <= '0';         --
    START_R <= '0';         --
    DONE <= '0';            --
    case S_NOW is
        when IDLE =>
            if START = '1' then
                S_NEXT <= SETUP;
                .
            end if;
        when SETUP =>
            START_L <= '1';
            L <= '1';
            S_NEXT <= LOAD;
        when LOAD =>
            if L_DONE = '1' then
                S_NEXT <= WORK;
                START_R <= '1';
                EN_BU <= '1';
                EN_C <= '1';
            else
                L <= '1';
                S_NEXT <= LOAD;
            end if;
        when WORK =>
            if S < to_integer(SMAX) then
                S_NEXT <= WORK;
                .
            else
                if L_DONE = '1' then
                    S_NEXT <= READ;
                    DONE <= '1';
                else
                    S_NEXT <= WORK;
                    .
                end if;
            end if;
        when READ =>
            SO <= S;
            if R_DONE = '1' then
                S_NEXT <= IDLE;
                DONE <= '0';
            else
                S_NEXT <= READ;
                DONE <= '1';
            end if;
    end case;
end process;
```

Code 3.1: Address Generator FSM

3.1.2 Stages sequence memory

The Stages Sequence memory is a ROM containing information about the stage sequences for the 54 Fourier Transform lengths: each N-length DFT is in fact divided into a cascade of smaller $N_0, N_1, ..., N_S$ DFTs. Each stage length is associated with a P value that defines the parallelism used by the Butterfly Unit in that stage: for example, if $N_0 = 4$ and $\frac{N}{N_0} \mod 4 = 0$, its parallelism will be $P_0 = 4$. Finally the memory stores also the value SMAX for each length that gives the Control Unit information about the number of stages into which the DFT length is divided.

3.1.3 Address Generator

The Address Generator generates a set of 16 addresses for the memory. They are generated using the Conflict-Free Parallel Access scheme [9]:

$$\begin{cases} \mathbf{bank} = \begin{cases} \left(n_1^{br} + n_2\right) \bmod L & (S = 2) \\ \left(n_1^{br} + \sum\limits_{j=2}^{S-1} n_j + n_S^{br}\right) \bmod L & (S > 2) \end{cases} \\ \mathbf{address} = \prod\limits_{n=3}^{S} (N_n)\mathbf{n_2'} + \sum\limits_{m=3}^{S-1} \left(\prod\limits_{n=m+1}^{S} (N_n)\mathbf{n_m} \right) + \mathbf{n_S} \\ \mathbf{n_2'} = \mathbf{n_2} \gg \left\lfloor \log_2 \frac{L}{N_1} \right\rfloor \end{cases} \tag{3.1}$$

This Access scheme is designed to access simultaneously sixteen banks of the memory block without any conflict: it translates the indices $n_1, n_2 ... n_S$ that identify data in the Mixed-Radix algorithm (1.10) to a couple **bank** – **address** that addresses a memory location. In the computation of the value **bank** for values of S higher than two, it's used $(n_S)^{br}$, the bit-reverse of the last index, to improve the parallelism of the scheme. The access scheme can be used both in input and output without modifying it. To implement it, the Address Generator (Fig. 3.4) is realized as a FSM (Fig. 3.5), controlled by a **START** signal from the Control Unit, that generates the sequence of indices and produces as output sixteen couples of **bank** – **address**. The Address Generator also produces two logic signals, **W** and **DONE**, to communicate to the Control Unit when it's working and when it has finished generating the addresses for that stage. Finally, the Read Address Generator, the block that generates the output addresses for the memory, is used to provide the indices to the TFMUL.

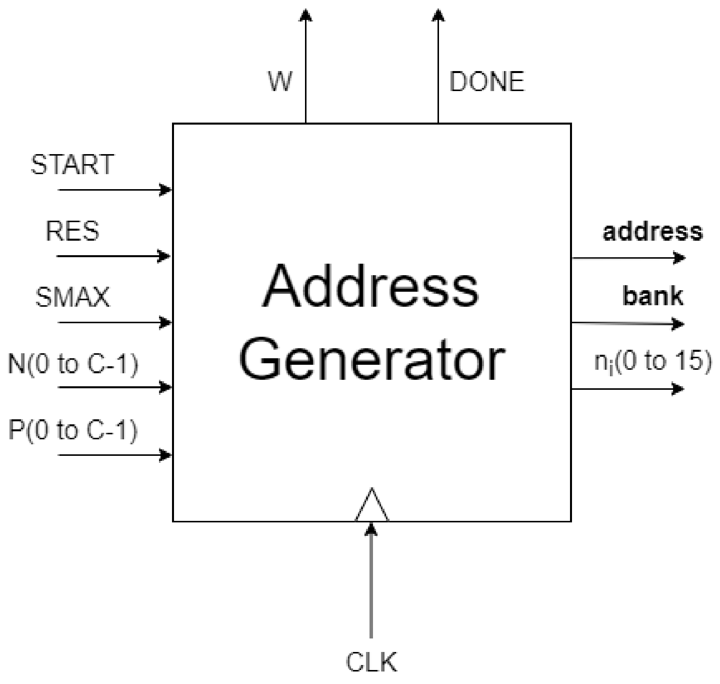


Figure 3.4: Address Generator Block

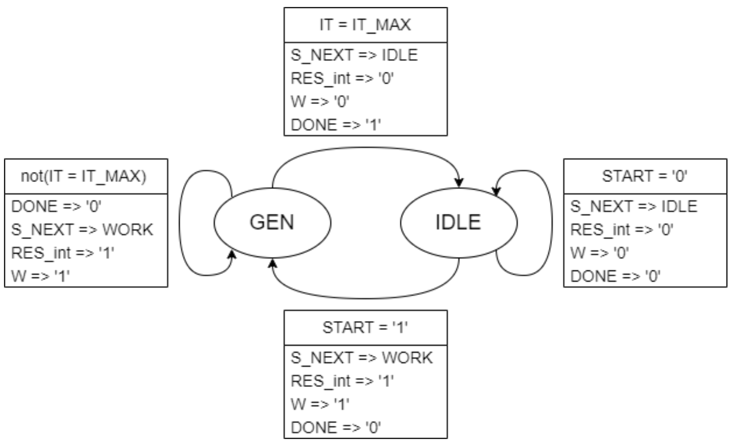


Figure 3.5: Address Generator Finite State Machine

Internal architecture

The Address Generator (Fig. 3.6) consists of the FSM, an Indices Generator, and an array of sixteen blocks named ADDGEN: these are the blocks that implement the Conflict-Free Access Scheme algorithm (3.1) to translate the indices into addresses for the memory.

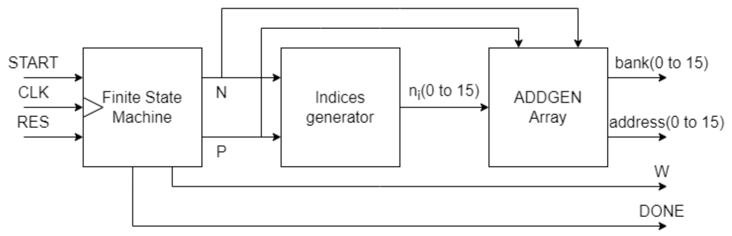


Figure 3.6: Address Generator Block Scheme

Finite State Machine Operations

The FSM has two states, **IDLE** and **GEN**; the first is the waiting state of the address generator: the generator waits for the **START** instruction from the Control Unit and maintains the internal reset signal at low state; the latter is the operational state of the address generator: the generator raises the internal reset signal, raises the **W** signal to high state and waits for the Indices Generator to end its iterations. Once it ends, the state is reverted to **IDLE** and the **DONE** signal is set to high to communicate to the Control Unit that the Address Generator has ended its operations. Code 3.2 presents the VHDL implementation of the FSM.

```vhdl
process(START,S_NOW,IT)
begin
    S_NEXT <= IDLE;
    IT0 <= 0;
    W <= '0';
    DONE <= '0';
    case S_NOW is
        when IDLE =>
            if START = '1' then
                S_NEXT <= GEN;
            end if;
        when GEN =>
            IT0 <= IT+1;
            RES_int <= '1';
            W <= '1';
            .
            .
            if IT = IT_MAX then
                IT0 <= 0;
                S_NEXT <= IDLE;
                DONE <= '1';
            else
                S_NEXT <= GEN;
            end if;
    end case;
end process;
```

Code 3.2: Address Generator FSM

3.1.4 Memory

The memory of the processor is realized as a "ping-pong" memory (Fig. 3.7), a couple of memory blocks used alternatively as writing or reading memory: at each stage, the processor reads data from one of the blocks, and writes the results in the other. Other than the input data signals and the addresses, the memory receives the **CTRL** signal from the Control Unit and the **RE**, **WR** signals from the address generators: the first one set which memory block will be the input one and which will be the output, the other two signals tell the memory when the address generators are providing valid addresses.

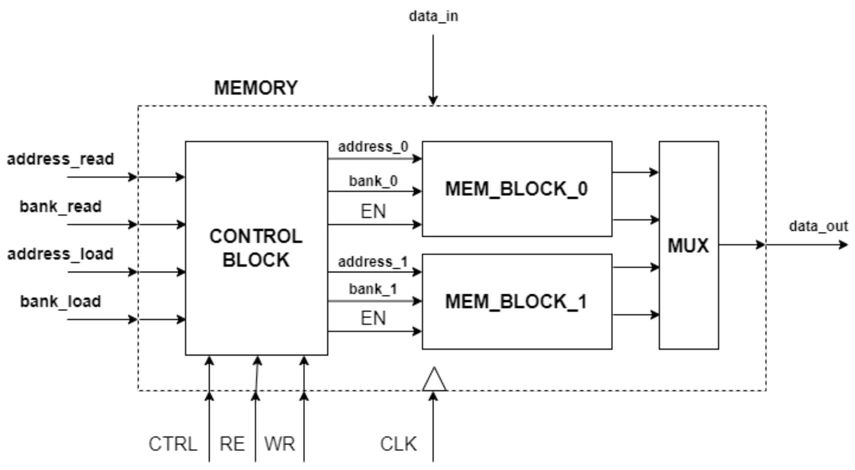


Figure 3.7: Memory Structure

Memory Block

Each memory block(Fig. 3.8) is realized as an array of 16 banks, each one with 256 addresses: this way each block is capable of writing/reading contemporaneously sixteen inputs/outputs without any conflict.

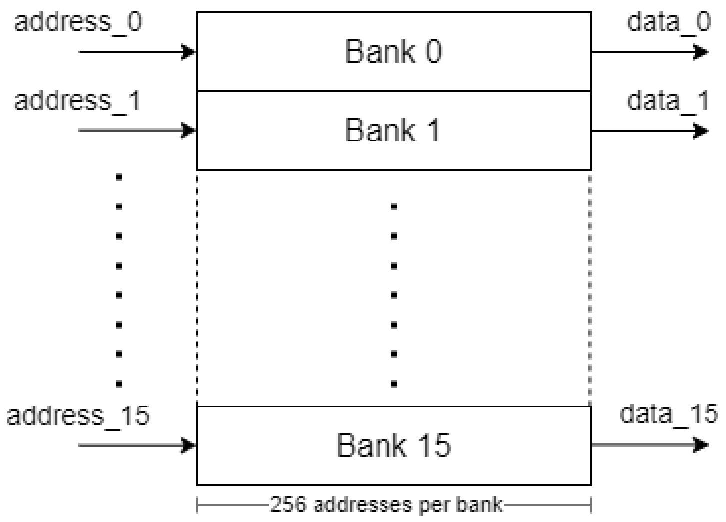


Figure 3.8: Memory Block

3.1.5 Butterfly Unit

The Butterfly Unit is the core block of the processor: it has a sixteen data input and the **SEL** signal that determines the operations to be carried out. It is a pipeline structure and it supports one radix-16, two radix-5/8, four radix-3/4, or eight radix-2 operations in parallel.

The internal structure (Fig. 3.9) is composed of five main blocks, two Processing Element A (PEA), one Processing Element B (PEB) and two Processing Element C (PEC), and a series of multiplexing blocks used to route the data in the correct way for the current radix-r operation.

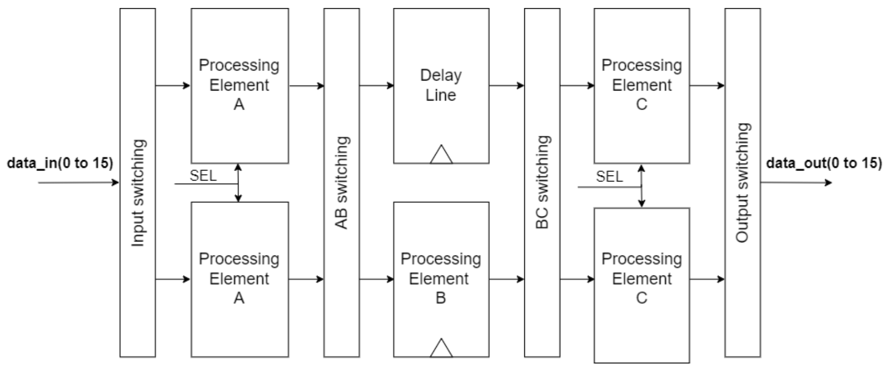


Figure 3.9: Butterfly Unit structure

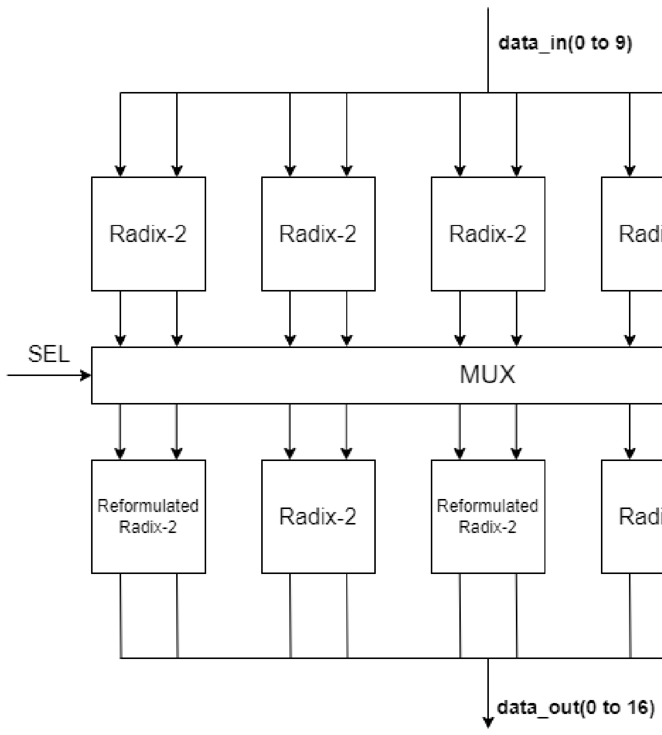
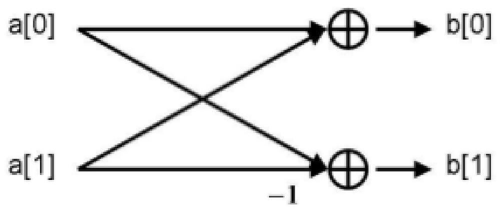


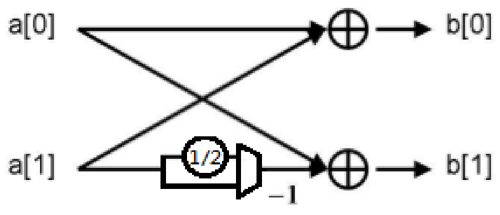
Figure 3.10: Processing Element A structure

Processing Element A

The PEA block (Fig. 3.10) is the first operational block of the Butterfly Unit, it has as input ten signals and as output sixteen signals. It supports two radix-4 or four radix-2 operations for radix-2, -4, -8, and -16 FFTs and also supports the initial part of the algorithms for one radix-5 DFT or two in parallel for the radix-3 DFT. It is made up of 6 radix-2 (Fig. 3.11a) units, 2 reformulated radix-2 (Fig. 3.11b) units, six multiplexers, and two trivial multipliers.



(a) Radix-2



(b) Reformulated Radix-2

Figure 3.11: Radix-2 and reformulated Radix-2 butterfly scheme

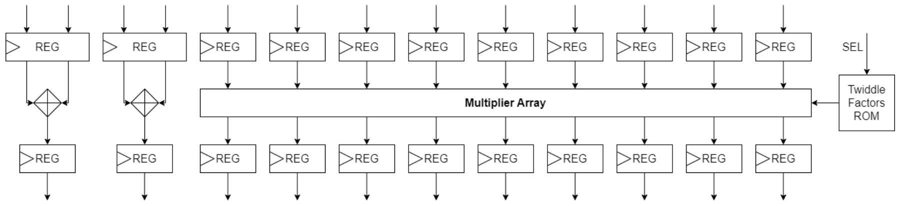


Figure 3.12: Processing Element B structure

Processing Element B

The PEB block (Fig. 3.12) contains all of the nontrivial multipliers in radix-3, -5, -8, and -16 DFT computation and contains the adders needed for radix-5 computation. It has as input thirteen signals and as output eleven signals.

Processing Element C

The PEC block (Fig. 3.13) supports two radix-4 operations for the radix-16 FFT or four radix-2 operations for the radix-8 FFT and also supports the final part of the radix-3 (two in parallel) and the radix-5 DFT computation. It is made up of 8 radix-2 butterfly units. It has as input ten signals and as output eighteen signals.

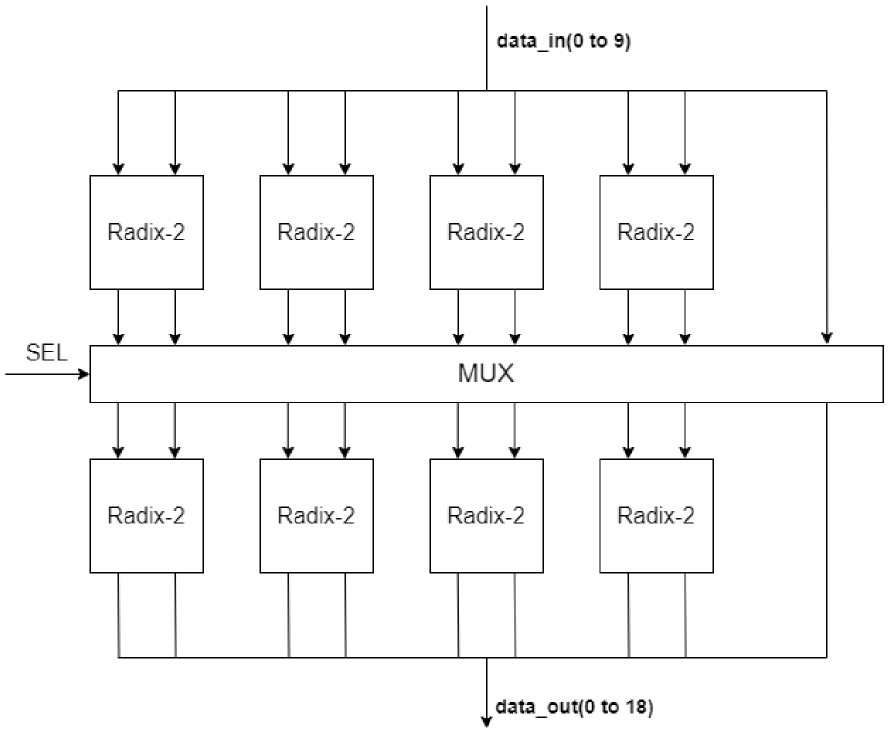


Figure 3.13: Processing Element C structure

3.1.6 Twiddle factor multiplier

The TFMUL (Fig. 3.14) is a module of the FFT processor used to execute the multiplications necessary to realize DFTs of a length higher than 16. Since this architecture should support a large number of different DFT sizes, to reduce the dimensions of the ROM containing the Twiddle Factors, the TFMUL is realized using a Rotation Angle Generator and a Coordinate Rotation Digital Computer (CORDIC) Rotator: this way the ROM needs only to be large enough to store the DFTs lengths to be implemented.

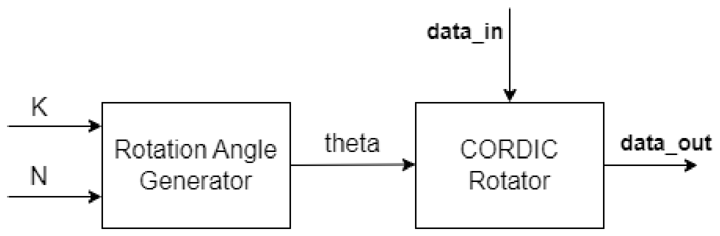


Figure 3.14: Twiddle Factor Multiplier structure

Rotation Angle Generator

The Rotation Angle Generator (Fig. 3.15) is used to obtain the angle θ used by the CORDIC Rotator, starting from the DFT length:

$$\theta = \frac{2\pi}{N}k \tag{3.2}$$

To reduce the occupied area by the ROM without losing too much precision, the lengths values $coef_i = \frac{2\pi}{N_i}$ are stored in fixed point format, 4 bit for the exponent $coef_{exp}$ and 15 bits for the mantissa $coef_{man}$. Storing the data this way means that the actual function that the block should realize is

$$\theta = coef_{man} \cdot k \cdot 2^{coef_{exp}} \tag{3.3}$$

So we can implement the Rotation Angle Generator using a multiplier and then a right shifter, since $-9 \leq coef_{exp} \leq 0$. The Rotation Angle Generator takes as input the address for the ROM and k and gives as output the value of θ.

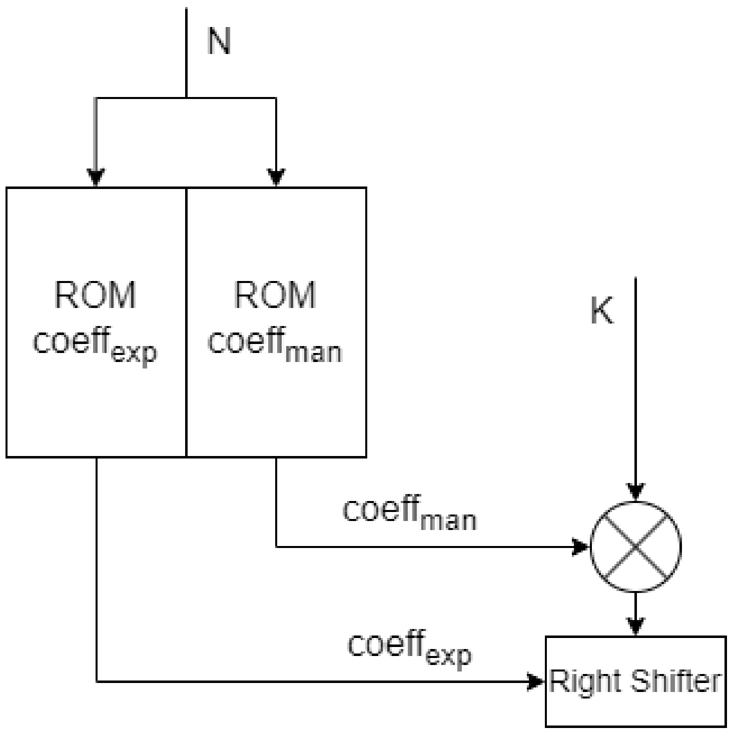


Figure 3.15: Rotation Angle Generator

CORDIC Rotator

The CORDIC Rotator is realized with a radix-4 pipelined architecture. It takes as inputs θ from the Rotation Angle Generator and $\mathbf{data_{in}}$, a complex value, and gives the rotated value $\mathbf{data_{out}}$.

While the radix-2 architecture needs a simple Selection Function:

$$\sigma_i = \begin{cases} -1, & z_i < 0 \\ +1, & z_i \geq 0 \end{cases} \tag{3.4}$$

and n rotation carried out as:

$$\begin{aligned} x_{i+1} &= x_i + \sigma_i y_i \\ y_{i+1} &= y_i - \sigma_i x_i \\ z_{i+1} &= z_i - \sigma_i \tan^{-1}(2^{-i}) \end{aligned} \tag{3.5}$$

to obtain n-bit precision, the radix-4 architecture, to reduce the number of micro-rotations needed to obtain the same precision, needs a more complex Selection Function. As described in the paper “High performance rotation architectures based on the radix-4 CORDIC algorithm”[10], this Selection Function:

$$\sigma_0 = \begin{cases} +2, & \text{if } 5/8 \geq w_0 \\ +1, & \text{if } 3/8 \geq w_0 < 5/8 \\ 0, & \text{if } -1/2 \geq w_0 < 3/8 \\ -1, & \text{if } -7/8 \geq w_0 < -1/2 \\ -2, & \text{if } w_0 < -7/8 \end{cases} \tag{3.6}$$

for i = 0, and:

$$\sigma_i = \begin{cases} +2, & \text{if } 3/2 \geq w_i \\ +1, & \text{if } 1/2 \geq w_i < 3/2 \\ 0, & \text{if } -1/2 \geq w_i < 1/2 \\ -1, & \text{if } -3/2 \geq w_i < -1/2 \\ -2, & \text{if } w_i < -3/2 \end{cases} \tag{3.7}$$

for i > 0, where $w_i = 4^i z_i$, and a this set of equations:

$$\begin{aligned} x_{i+1} &= x_i + \sigma_i 4^{-i} y_i \\ y_{i+1} &= y_i - \sigma_i 4^{-i} x_i \\ z_{i+1} &= z_i - \alpha_i[\sigma_i] \end{aligned} \tag{3.8}$$

where $\alpha_i[\sigma_i] = \tan^{-1}(\sigma_i 4^{-i})$can be used to reach a n-bit precision with only $n/2+1$ micro-rotations, half of what a classic radix-2 architecture needs. This means that the radix-4 architecture permits a significant reduction in computation time and area occupation when using a pipeline structure.

The pipeline structure used in this work is presented in Fig. 3.16: each couple x/y$_{cell}$ - w$_{cell}$ (Fig. 3.17) represents a micro-rotation; each σ_i value is also used to compute the scale factor: K is in fact not fixed at 1.64 as in radix-2 architecture, but needs to be evaluated each time. In order to do so, for the first $n/4-1$ micro-rotations, the scale factors k_i are pre-computed and stored in a ROM: the σ_i values are used to select the k_i value needed and then the final value of the scale factor K is obtained with a multiplier (Fig. 3.18).

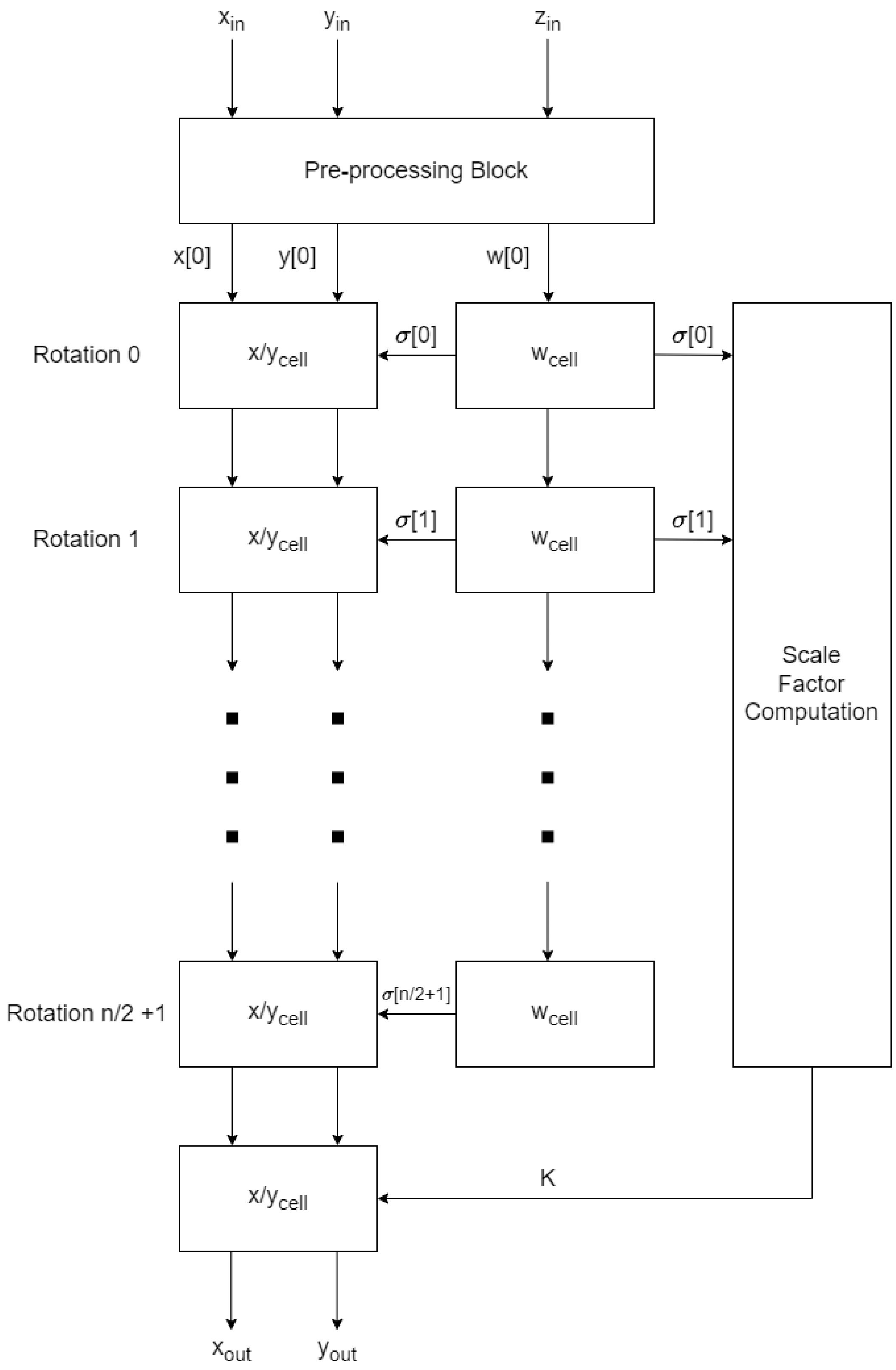
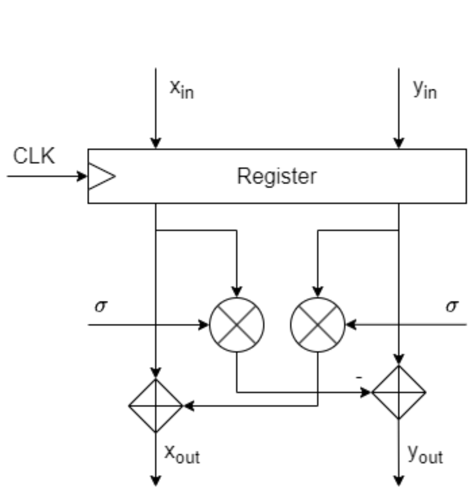
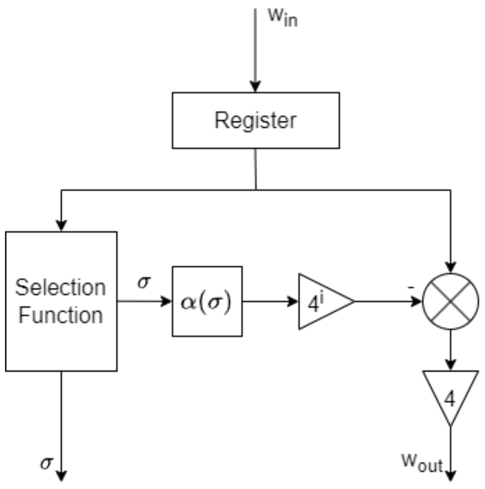


Figure 3.16: CORDIC Rotator pipelined structure



(a) x/y$_{cell}$ structure



(b) w$_{cell}$ structure

Figure 3.17: Blocks that implement a micro-rotation

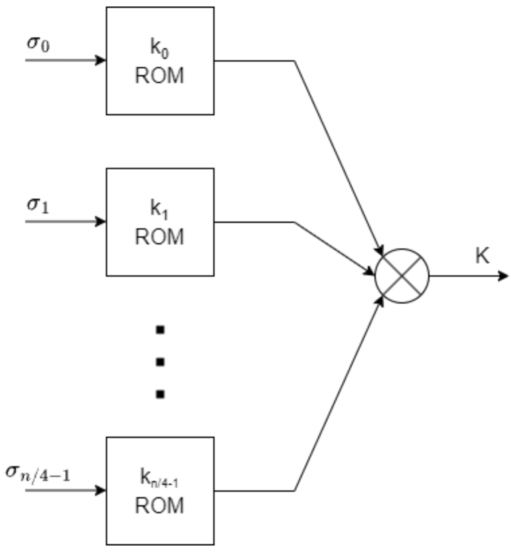


Figure 3.18: Scale Factor computation block

3.2 Debugging and simulations

During the RTL design, a test bench was realized to simulate the processor and its components' behaviour to ensure their correct functioning. The test bench is designed to load input data from a .txt file, sixteen samples per clock cycle, until it reaches the end of the file; the samples are converted in signed representation, form acceptable from the processor (Code 3.3).

```vhdl
process
    variable VLINE : line;
    variable V : real;
    file INP_FILE : text open read_mode is "./matlab/x_data.txt";
begin
    while not(endfile(INP_FILE)) loop
        if LOAD = '1' then
            for i in 0 to to_integer(N_out(0)*P_out(0))-1 loop
                    readline(INP_FILE, VLINE);
                    read(VLINE, V);
                    data_in(i)(0) <= to_signed(integer(V*2.0**(N-1)), N);
            end loop;
        end if;
        wait until CLK'event and CLK = '1';
    end loop;
    wait for TCK*50;
    wait;
end process;
```

Code 3.3: Test bench load process

Then the test bench waits for the processor to perform the computation and finally starts to save the output data to a .txt file, to later elaborate and display them(Code 3.4). The program used to carry out the simulations was Verdi Automated Debug System from Synopsis, and the data output from the test benches were elaborated using MATLAB.

```vhdl
process
    variable WLINE : line;
    variable W1,W2 : real;
    file OUT_FILE : text open write_mode is "./matlab/yv_data.txt";
begin
    wait until CLK'event and CLK = '1';
    if DONE = '1' then
      wait for OUTDEL;
    for i in 0 to to_integer(N_out(to_integer(S_out))*P_out(to_integer(S_out)))-1
    loop
            W1 := real(to_integer(data_out(i)(0)))*2.0**(-N+1+integer(ceil(log2(
    real(to_integer(MODE))))));
          write(WLINE, W1,left,15);
          W2 := real(to_integer(data_out(i)(1)))*2.0**(-N+1+integer(ceil(log2(real
    (to_integer(MODE))))));
          write(WLINE, W2,left,15);
          writeline(OUT_FILE, WLINE);
    end loop;
    end if;
end process;
```

Code 3.4: Test bench save process

Figure 3.19 shows the behaviour of the processor while computing a 20-point FFT: first, the processor loads the samples in its memory; after the FFT computing starts: in the case of 20-point FFT, the computation is divided in a cascade of a two 4-point FFT and a 5-point DFT with a Twiddle Factor multiplication in the middle. The output is then presented at the output. The total process is carried out in 37 clock cycles and the result has a maximum relative error of $\Delta = 0.052\%$ The graph in Fig. 3.20 shows a comparison of the processor results with the FFT

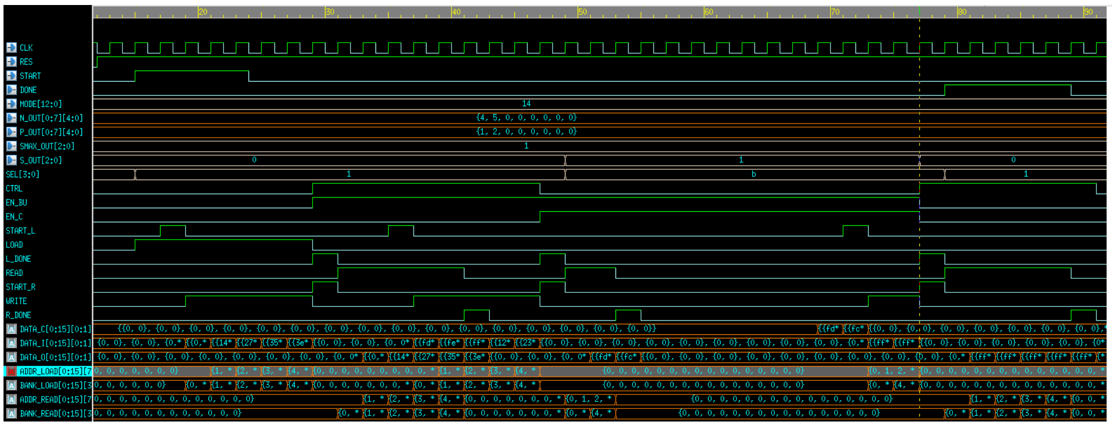


Figure 3.19: Waveform Graph of the simulation

computed in MATLAB: as expected the results are quite similar, with an absolute error of $\Delta = ...$, comparable with the quantization error caused by the number of bits used to represent the data.

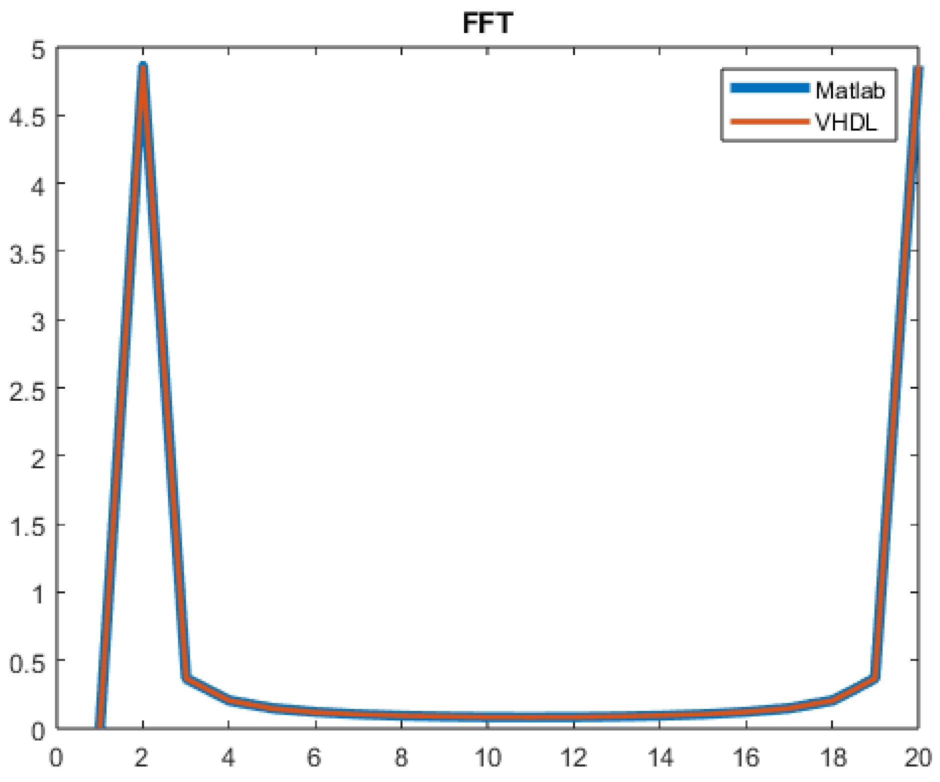


Figure 3.20: MATLAB - Processor comparison of the 20-point DFT

While the processor is able to compute DFTs implemented with a two-stage sequence, when the number of stages needed increases, the resulting DFT presents a series of artefacts: if we consider for example the case of 4096-point FFT in figure 3.21, the graph shows small peaks between the two main peaks. These errors are probably caused by a problem with the Conflict-Free Parallel Access Scheme implementation.

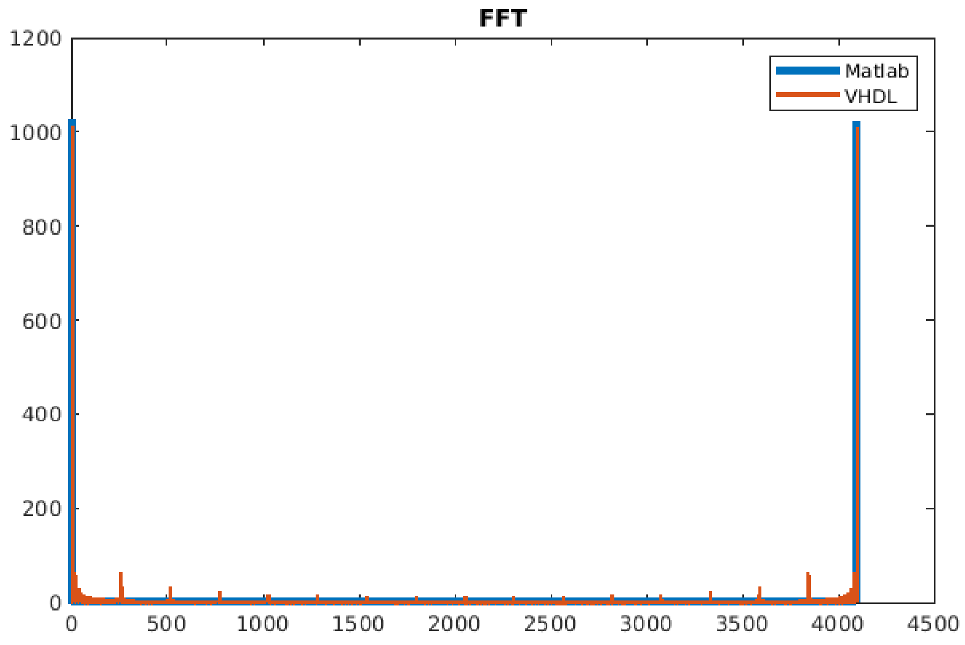


Figure 3.21: MATLAB - Processor comparison of the 4096-point FFT

4 Synthesis

The step following the RTL description of the processor is the Logic Synthesis. The Logic Synthesis is a process that turns the abstract description of the design behaviour, in our case the Register Transfer Level (RTL) description, into a netlist, a design implementation in terms of logic gates. This process is realized thanks to a synthesis tool: in this thesis, Design Compiler®[11] was used.

4.1 Logic Synthesis steps

The main steps of the Logic Synthesis are:

- Design preparation
- Definition of the environment
- Definition of Constraints
- Optimization

Design preparation

The RTL design should be realized following a set of guidelines aimed at improving the synthesis quality and reducing the compilation time: an example is the utilization of standard models for commonly used elements, like registers or multiplexers, or the use of output registers when possible to relax the time constraints. For this design, the main difficulties during synthesis came from the implementation of the Address Generator and the memory: this block presents a complex combinatorial logic, hard to properly implement and synchronize with the rest of the design.

Definition of the environment

By outlining the technological libraries to be employed, this stage defines the environment for carrying out the synthesis. For this thesis, it was used the Global Foundries 22 nm Fully-Depleted Silicon-on-Insulator (FD-SOI) technology, suitable for this type of implementation as it was developed for integrated, low-power RF designs. As the working conditions for the processor are very variable, in this work the default conditions defined by the library were used.

Definition of Constraints

During this phase the constraints for the optimization are set, both on timing and area. In our case, the main constraint was represented by the operating clock: the processor should be able to work at 250 MHz. As for the area, the main objective was to obtain the minimum area possible while respecting the constraint on timing. Another important optimization goal was the low power consumption of the device.

Optimization

This is the actual compilation of the netlist, in Design Compiler carried out using the Top-Down method: the design constraints are applied only to the top cell and the compilation is carried out simultaneously on all the designs. This is the simpler way to compile a netlist but it takes a long time and each change to the design needs a new compilation.

4.2 Logic Synthesis results

Once the compilation is ended, Design Compiler produces the netlist of the design and it can be analysed to verify if the design respects the optimization constraints and to do a preliminary evaluation on area occupation and power consumption.

Table 4.1 presents the report on the area realized by Design Compiler: after the optimization, the processor has an estimated area of 0.969 mm^2.

Table **??** presents the report on the power realized: after the optimization, the processor has a power consumption of 191 mW, mainly due to the registers present in the design. This power consumption however is probably largely overestimated due to a non-optimized implementation of the memory block.

Area report	
Number of ports	241956
Number of nets	1706235
Number of cells	1397709
Number of buf/inv	210474
Number of references	19
Cominational Area	534623 μm
Buf/Inv Area	37222 μm
Noncombinational Area	434305 μm
Total Cell Area	968928 μm

Table 4.1: Report on the Area occupied by the processor

Power report	
Cell Internal Power	180.7 mW
Net Switching Power	1.45 mW
Total Dynamic Power	182.15 mW
Cell Leackege Power	8.88 mW
Total Power Consumption	191.03 mW

Table 4.2: Report on the Power consumption of the processor

5

Physical Design

Once the Logic Synthesis is completed, the netlist can be exported and used to realize the Physical design. To do so, the optimized netlist is converted into a hardware language description file, for example, Verilog of VHDL, and along with a file containing information about the timing of the design, it's used as a starting point to obtain a geometrical representation of the design called layout [12]. Once this layout is obtained, it can be analysed through a Static Time Analysis (STA) to verify that it meets the timing requirements and finally, if it meets the specifics required, it can be physically realized into a foundry or implemented on a Programmable Logic Device (PLD), like a Programmable Array Logic (PAL) or a Field Programmable Gate Array (FPGA). The program used to realize the Physical design was Innovus Implementation System by Cadence®. The Physical Design is divided into a series of steps:

1. **Design Partitioning**: division of the design into blocks with the objective of minimizing interconnects between them.
2. **Floor Planning**: definition of the silicon area dedicated to the design implementation.
3. **Power Planning**: realization of the physical structure dedicate to power delivery.
4. **Placement**: geometric positioning of the standard cells.
5. **Clock Tree Synthesis**: generation of the clock tree distribution.
6. **Routing**: connection of the various standard cells and blocks.
7. **Design-for-Manufacturability enhancements**

8. **Verifications**: verifying the compliance of the layout with design rules.

The core steps of the Physical design are the **Placement** and the **Routing**.

5.1 Placement

The Placement step consists of the geometric placement of each cell of the design inside a specified area. Depending on the technology used, this could mean different things:

- **Standard Cells**: the technology is described by a group of predefined cells used to implement various designs; during the Place phase, the position X, and Y of each cell are defined.
- **Sea-of-gates**: the core element of the technology is the transistor itself. The Placement phase allocates for each cell to realise a series of pre-fabricated transistors and defines their geometrical interconnection.
- **PLD**: the design is not physically implemented but is realized by allocating its cells to logic blocks on the PLD.

The main objectives of the Placement are to minimize the interconnection lengths and to reduce to the minimum the interconnect congestion. To implement the Placement, two class of algorithm exists, Constructive placement and Iterative placement improvement. Usually, Placement starts from a Constructive placement method: using a prefixed set of rules, e.g. Min-Cut algorithm or Eigenvalue method, the constructive placement algorithm generates a first placement; then the placement is improved using an Iterative placement algorithm, a method that consists in moving cells of an existing placement trying to refine it. The result of Placement is a detailed description of the best positioning of each cell and block, based on various cost functions, such as wire length, routability and performance.

As can be seen in figure 5.1, the greater part of the processor area is occupied by the Memory block; a large portion is also dedicated to the Twiddle Factor Multiplier block as it is implemented with sixteen parallel CORDIC rotators with a pipelined architecture. The input/output PIN placement is automatically set to better suit the design.

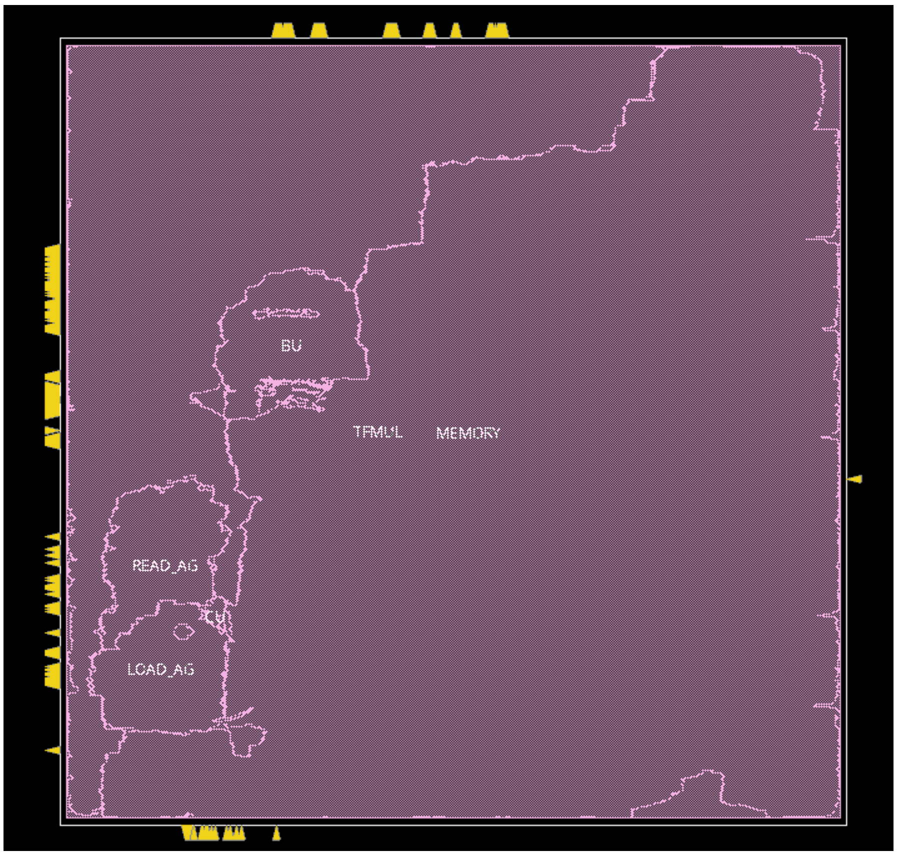


Figure 5.1: Placement of the processor cells and blocks

5.2 Routing

The Routing step consists of the realization of the interconnections between the standard cells, using wires. The objective is to connect 100% of a system while minimizing wire length and cross-talk. As for the Placement, Routing changes its meaning based on implementation technology:

- **Standard cells**: exact geometric description of the metal interconnections
- **PLD**: selection of the routing tracks to be used to connect the various blocks

While Routing can be realized with a single-phase approach called *Area Routing*, generally, it is divided into two phases:

Global Routing $\longrightarrow$ a coarse routing that has two scopes: generating an approximate route for each net and assigning to the correct routing regions each net without specifying the actual layout. This process consists of three steps:

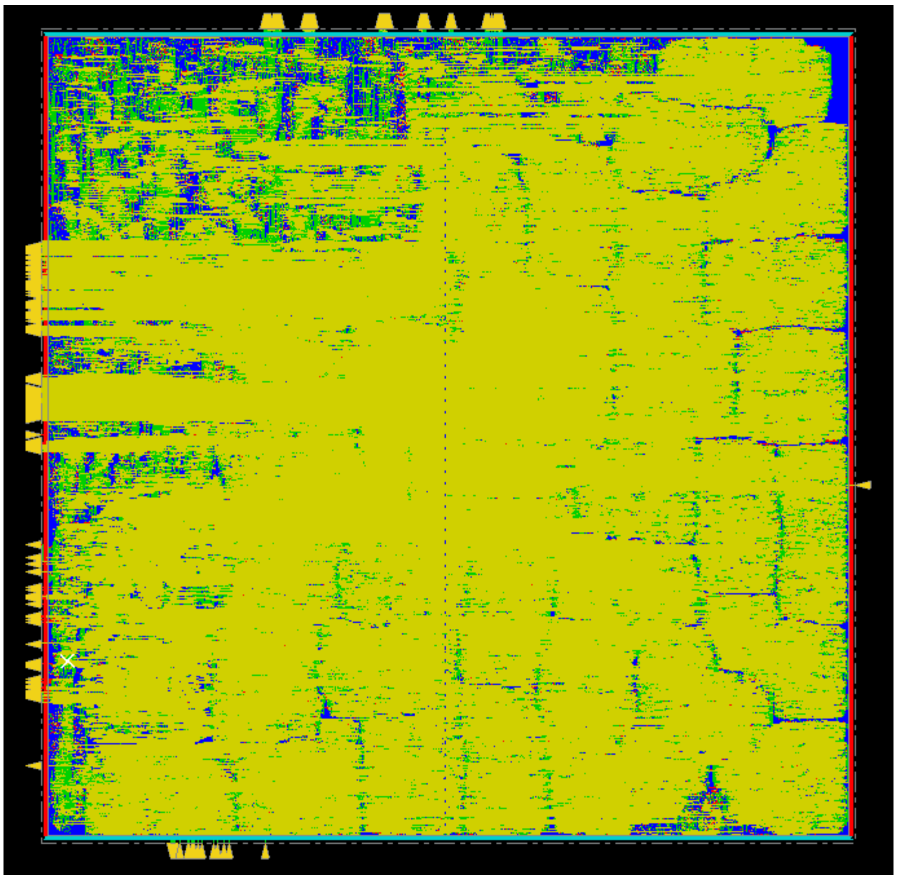


Figure 5.2: Final layout of the processor

Region Definition, partitioning of the design into routing regions, *Region Assignment*, identifying the series of routing regions that each net will route through, *Pin Assignment*, each net is assigned to a pin.

Detailed Routing $\longrightarrow$ the actual definition of the geometric layout of the nets. This process is carried out incrementally: each region is routed one at a time in a predefined order, based on the nets criticality and nets density of the region.

The result of the Routing phase is the geometric layout of each net connecting the standard cells: in figure 5.2 is presented the final layout of the processor.

6 Conclusions and Future Works

A study on the 5G standard was conducted to determine the specifics and requirements for a processor implementing the Fast Fourier Transform (FFT) algorithm. Starting from the various designs of processors present in literature, in particular, the one described in the paper "A High-Flexible Low-Latency Memory-Based FFT Processor for 4G, WLAN, and Future 5G"[8], a Register Transfer Level (RTL) design of the processor was realized and tested to verify its functioning. Using Design Compiler®, the synthesis of the design was carried out to obtain a netlist implementable using the Global Foundries 22 nm FD-SOI technology. Finally, the Physical Design of the processor layout was performed in Innovus Implementation System to obtain a geometric description of the design that could be used to realize a functioning chip. The main applications for this design come from the necessity of fast modulation and demodulation of complex data symbols in radiocommunication, in particular as part of Orthogonal frequency-division multiplexing (OFDM) for 5G, LTE, Wi-Fi and other communication systems.

As described at the end of chapter 3, while the processor is perfectly functional for each FFT or DFT size mapped in two stages using the Mixed-Radix Algorithm (MRA), it still presents some problems for FFT or DFT sizes mapped in a higher number of stages. This issue could be the object of future studies, starting from the Conflict-Free Access Scheme implementation which seems to be its main reason. Moreover, a more accurate study of memory architectures could lead to a better implementation of the RAM memory of the processor, to reduce both its power consumption and the area occupied on the die.

References

[1] Radiocommunication sector. *Minimum requirements related to technical performance for IMT-2020 radio interface(s)*. International Telecommunication Unity, 2017.

[2] Technical Specification Group Radio Access Network. *NR Physical channels and modulation*. 3rd Generation Partnership Program, 2022.

[3] Chen-Fong Hsiao, Yuan Chen, and Chen-Yi Lee. “A Generalized Mixed-Radix Algorithm for Memory-Based FFT Processors”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 57.1 (2010), pp. 26–30. DOI: 10.1109/TCSII.2009.2037262.

[4] C. Burrus. “Index mappings for multidimensional formulation of the DFT and convolution”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 25.3 (1977), pp. 239–242. DOI: 10.1109/TASSP.1977.1162938.

[5] Xin-Yu Shih, Hong-Ru Chou, and Yue-Qu Liu. “VLSI Design and Implementation of Reconfigurable 46-Mode Combined-Radix-Based FFT Hardware Architecture for 3GPP-LTE Applications”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 65.1 (2018), pp. 118–129. DOI: 10.1109/TCSI.2017.2725338.

[6] Xin-Yu Shih, Yue-Qu Liu, and Hong-Ru Chou. “48-Mode Reconfigurable Design of SDF FFT Hardware Architecture Using Radix-32 and Radix-23 Design Approaches”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 64.6 (2017), pp. 1456–1467. DOI: 10.1109/TCSI.2017.2654451.

[7] Chia-Hsiang Yang, Tsung-Han Yu, and Dejan Markovic. “Power and Area Minimization of Reconfigurable FFT Processors: A 3GPP-LTE Example”. In: *IEEE Journal of Solid-State Circuits* 47.3 (2012), pp. 757–768. DOI: 10.1109/JSSC.2011.2176163.

[8] Shaohan Liu and Dake Liu. “A High-Flexible Low-Latency Memory-Based FFT Processor for 4G, WLAN, and Future 5G”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27.3 (2019), pp. 511–523. DOI: 10.1109/TVLSI.2018.2879675.

[9] Chen-Fong Hsiao, Yuan Chen, and Chen-Yi Lee. “A Generalized Mixed-Radix Algorithm for Memory-Based FFT Processors”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 57.1 (2010), pp. 26–30. DOI: 10.1109/TCSII.2009.2037262.

[10] E. Antelo et al. “High performance rotation architectures based on the radix-4 CORDIC algorithm”. In: *IEEE Transactions on Computers* 46.8 (1997), pp. 855–870. DOI: 10.1109/12.609275.

[11] Synopsys. *Design CompilerőOptimization Reference Manual.* Version F-2011.09. 2011.

[12] N. Sherwani. *Algorithms for VLSI Physical Design Automation.* Kluwer, 1998.