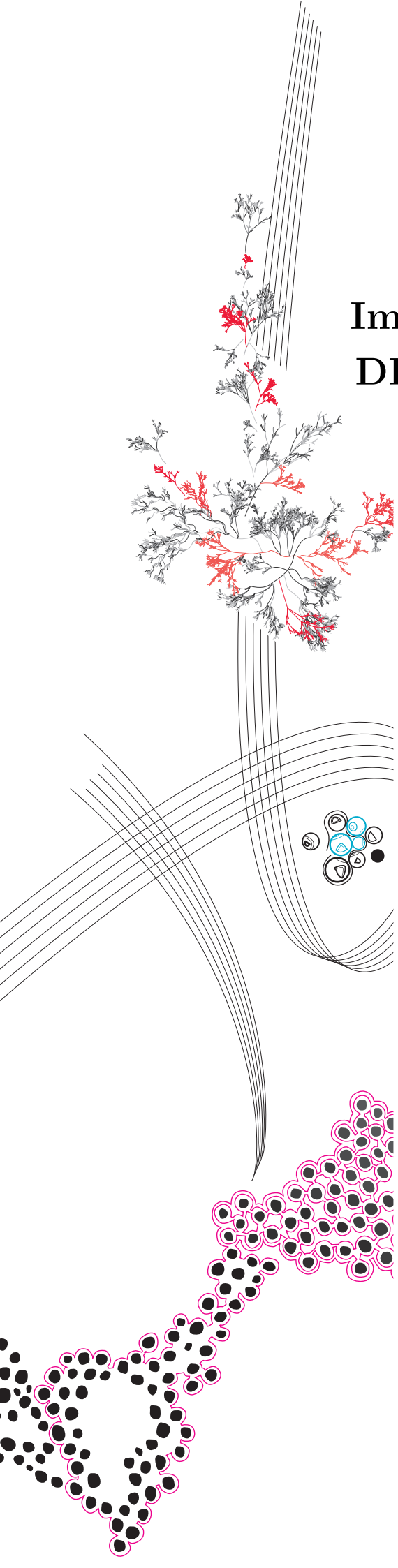

Implementation of a power-efficient DFT based demodulator for BFSK

Giovanni Meciani

M.Sc. Thesis

December 2018

Committee:
dr. ir. A. B. J. Kokkeler.
S. Safapurhajari M.Sc.
dr. ir. R. A. R. van der Zee

Computer Architecture for Embedded Systems Group
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Abstract

In wireless communication, the frequency offset is a problem that hinders the communication. It arises from the discrepancy between the frequencies of the oscillator of the transmitter and the one of the receiver. To solve this problem, oscillators with high frequency stability can be employed, at the expense of higher power requirements. As a consequence, this solution can be problematic in Wireless Sensors Network (WSN), where sensor nodes have limited power resources. Another solution is to correct the frequency offset within the demodulation algorithm, thus making it possible to use less power-hungry oscillators.

This thesis focuses on the hardware implementation of an existing offset tolerant demodulation algorithm for BFSK modulation. Particular attention was posed in making the demodulator as power efficient as possible, given its usage in a WSN. To reach this goal, the two Discrete Fourier Transforms were optimised for zero-padding, which is an operation used in the algorithm. These two modifications allowed to save power when compared with the standard radix-2 FFT.

Fixed-point is the chosen data representation. As the word length of the representation is a critical aspect that affects power consumption, an appropriate size was determined by using a procedure that makes use of MATLAB. The complete demodulator is implemented in VHDL and is used to characterise the system, when an FPGA is used in the synthesis process. Relevant results analysed are bit error ratio (BER) with different levels of noise, power consumption with different clock speeds, and resource usage.

Contents

List of Acronyms

AA	Anti-aliasing
ADC	Analog to Digital Converter
ALM	Adaptive Logic Module
ASIC	Application Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
BER	Bit Error Ratio
BFSK	Binary Frequency Shift Keying
BIBO	Bounded Input Bounded Output
CFO	Carrier Frequency Offset
DDC	Digital Down Converter
DFT	Discrete Fourier Transform
DSP	Digital Signal Processor
FFT-ZP	Fast Fourier Transform Zero Padding
FFT	Fast Fourier Transform
FP	Fixed-point
FPGA	Field Programmable Gate Array
IF	Intermediate Frequency
IoT	Internet of Things
IP	Intellectual Property
LNA	Low-Noise Amplifier
LUT	Look-Up Table
mSDFT	Modulated SDFT
OFDM	Orthogonal Frequency-Division Multiplexing
SDFT	Sliding DFT
SNR	Signal to Noise Ratio

SS Spread-spectrum

ST-DFT Short-Time Discrete Fourier Transform

UNB Ultra Narrowband

UWB Ultra Wideband

VHDL VHSIC Hardware Description Language

WSN Wireless Sensor Network

Chapter 1

Introduction

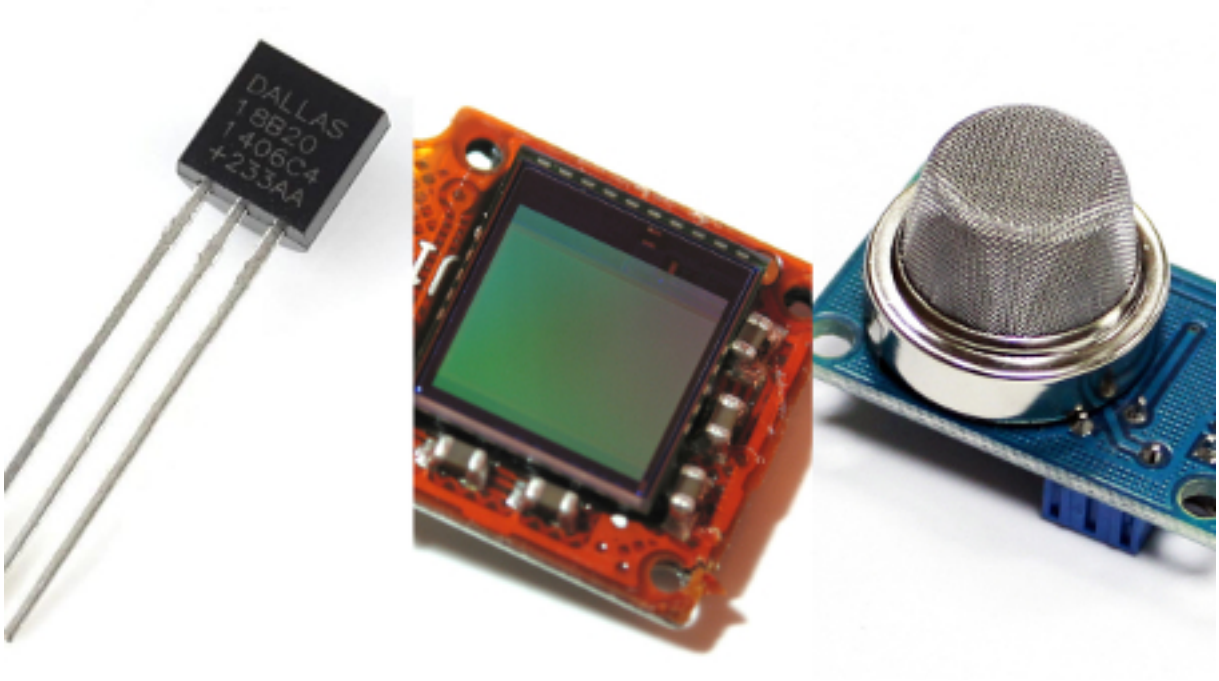

Figure 1.1: Example of sensors. From left to right: thermometer, image sensor, smoke sensor.

The acquisition of information is an important aspect in our lives. We use our five senses to obtain data from our surroundings, but they are neither precise nor reliable. We can tell if the weather today is hot or cold, however we cannot describe it using numbers, which makes the statement consequently subjective.

With the advent of science, which required precise measurements, analog sensors have been invented and improved since the 16th century, when the first thermometer was created by Cornelis Drebbel [1]. Other examples are the barometer to measure pressure, hygrometer for humidity, speedometers for speed, and many other exist to measure different physical variables.

Sensors have changed drastically with the coming of integrated circuits. They are now very small, highly precise, and most importantly, can gather information faster than a human could ever read. Imagine if we had to constantly monitor the vital signs of a patient, we would have to spend an excessive amount of time reading the thermometer, the ECG, blood pressure and so on. Thanks to electronic sensors, data can be continuously collected and stored for later use.

But what if we have a lot of these sensors, spread over a vast area? We can still place them here and there and check them from time to time. This idea would be feasible, although rather

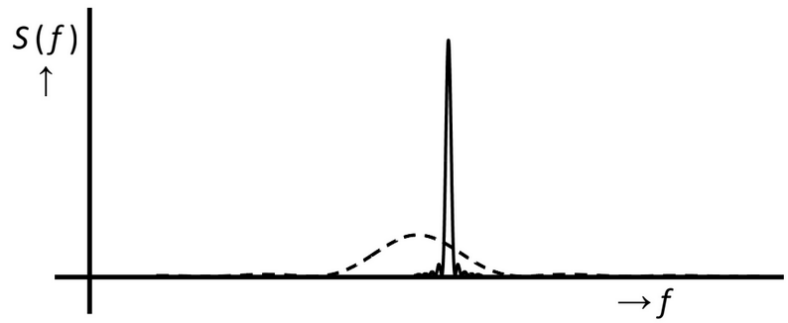

Figure 1.2: The dotted signal represents the interference which is wide spread, while the solid line depicts an ultra-narrowband signal.

tedious, when it comes again to patients or to monitor a vineyard or the temperature of each room in your house. However, when safety or security are involved, this option is not viable. In these cases, the information needs to be communicated as fast as possible. As a consequence, sensors have been arranged in what are now called Sensors Network. Each sensor node can communicate its retrieved data to the user, through wires or wirelessly. In the present context, we will focus on the second group which is referred to as Wireless Sensors Network (WSN).

Many WSN are characterised by a very low data rate, in the order of Bytes or kBytes per second. In fact, each sensor reading requires as many bit as the ADC bit resolution in use, which usually spans from a few bit up to 32 bit. Moreover, the transmission of this data occurs every now and then. Sensor nodes rely often on batteries and therefore it is necessary to use the available power as sparingly as possible. Because of that, two approaches have been adopted in order to maintain high power-efficiency in the sensor nodes: duty cycling [2], and wake-up radios [3]. With the first approach, the radio gets ready to receive data at a fraction of the time. This is already an improvement compared to leaving it continuously listening, but it is not optimal. In fact, as it may not be known when the next transmission will occur, the receiver might listen more than it is necessary in order not to miss messages. With the second approach, two radios are employed: a main radio and a wake-up radio. The first one takes care of demodulating the incoming signal and transmit the decoded bits to the microcontroller. Because this operation is power expensive, the wake-up radio, which is designed to be as power efficient as possible, is used to wake the main radio only when it detects a message being transmitted.

Interference is a disturbance produced by the communication between other devices. It is nowadays characterised by wideband signals, for example due to spread spectrum or OFDM techniques. These are usually employed in devices that use Wi-Fi or Bluetooth technologies, such as smartphones, PC, smart appliances, and so on.

In order to avoid it, ultra-wideband (UWB) techniques can be employed. The drawback is the short range that they reach. This is due to the regulations that impose a maximum amount of power that this techniques can use. The Slow Wireless Project proposes to utilise ultra-narrowband (UNB) techniques to realise robust, power-efficient, ad-hoc radio links for WSN that require low instantaneous data rates (Bytes/second). To understand how UNB signals can be useful in wideband interference such as spread spectrum (SS), an example is shown in Fig. 1.2. The dotted signal represents the interference which is characterised by a wideband signal. The solid line indicates an ultra-narrowband signal. Its power is concentrated in a very small bandwidth, while its power spectral density stands decisively above the interference. As a consequence, the signal can be better discriminated by the receiver.

One of the problems of using UNB techniques is the frequency offset [4], which will be illustrated in Section 1.2. This thesis focuses on the implementation of a power-efficient frequency

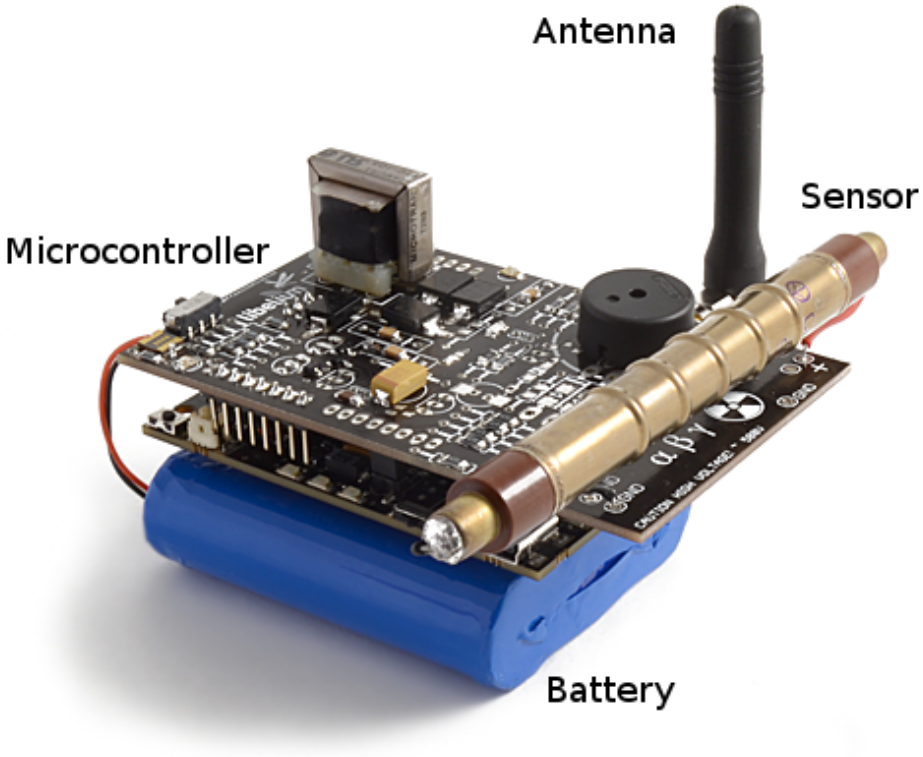

Figure 1.3: Example of a wireless sensor node.

offset tolerant demodulation algorithm for a BFSK modulation technique. Before presenting the algorithm, an overview of WSN is given in the next section.

1.1 Wireless Sensor Network

A Wireless Sensor Network (WSN) is a collection of sensors physically distant to one another that gather information and transmit them in a wireless manner. Usually, physical variables like temperature, humidity, air pressure, etc. is the data being collected. In a WSN there are usually two types of nodes. Sensor nodes obtain the relevant data from the environment and constitute the majority of the nodes. The other type is gateway node, which gathers the information transmitted by the sensor nodes and passes them to the user [5].

A node is usually composed by a radio transceiver, a microcontroller, at least one sensor, and a power supply. An example is shown in Fig. 1.3. The radio transceiver is required to send and receive data, although the latter functionality may not always be present. In fact, a sensor needs to receive data only if it can be controlled. The microcontroller is used to collect data from the sensor(s) and pass them to the transmitter, as well as receiving configuration instructions, for example the frequency at which the data should be transmitted. At least one sensor is connected to the node, although the system might incorporate multiple sensors.

Finally, a power supply is used to provide the energy to run the system. Depending on where the node is deployed, it may run connected to a power grid or use an energy storage like a battery or even harvest power on its own, for example by employing a small solar panel. Where the battery is the only available resource of power, it is of paramount importance to maintain the energy consumption as low as possible in order to use small batteries and/or guarantee a long term running period. As the radio communication system is the most power-hungry

component [6], modulation and demodulation techniques need to be designed as energy-efficient as possible.

WSN are becoming more adopted in many different fields. In [7], an extensive taxonomy of WSN applications is proposed. WSN can be characterised as single or multi-hop. In the former case, a message sent by a sensor node is directly dispatched to the user node. Instead, in multi-hop, a message is first sent to other nodes before reaching the user. Some of the applications where WSN are being used and belong to the multi-hop group are: metropolitan operation, military, civil engineering, environmental monitoring, logistics, position & animals tracking, transportation, automobile, sensor & robots, reconfigurable WSN, and nanoscopic sensors. Instead, single-hop WSN can find application in: industrial automation, health, mood-based services, entertainment, smart office, sports, building automation, home control.

Another classification for WSN takes into account whether nodes are static or mobile [8]. In the first case, the nodes are placed once and remain in the same position indefinitely, conversely, in the mobile WSN nodes can move around. In our work, only static WSN are taken into consideration.

In [7], WSN applications are grouped by data rate. Thirteen out of eighteen of the considered applications are medium or low bit rate, which means they use less than 115.2 kbps. Because of that, ultra-narrowband (UNB) modulation schemes are becoming more widespread [4]. The advantage resides in the longer range that this technique is able to achieve compared with wide-spectrum signals.

1.2 The frequency offset problem

The frequency offset is a non-ideality that arises from the discrepancy between the frequencies of the oscillator of the transmitter and the one of the receiver. This is due to physical limitations caused by the age of the oscillator, its temperature, mass transfer due to contamination, stress-strain in the resonator, and quartz defects [9]. This imperfection is expressed based on deviation from the nominal value of the frequency indicated by the manufacturer in terms of parts per million (ppm). For example if an oscillator is rated 50 MHz with $\pm 20 ppm$, then its actual frequency is between 50.001 MHz and 49.999 MHz.

If the oscillators were ideal, the two frequencies used to transmit bit “1” and bit “0” would remain unchanged. In reality, the frequency offset problem might occur. This produces a shift of these two frequencies, which makes the receiver unable to detect the incoming data. The lower the data rate, the higher the frequency stability of the oscillator has to be, being equal the carrier frequency. For example, for frequency offset in the order of 20% of data rate and data rate of 1 kbps, 2.4 GHz for the carrier wave, the frequency stability of the oscillator needs to be less than $\pm 1 ppm$. A crystal oscillator with low frequency stability is not only expensive, but also more power-hungry [4]. However, that is not desirable in sensor nodes powered by batteries.

This problem can be addressed in the digital or analog domain or both. A number of algorithms have been proposed for the digital domain [10–12], but they can only partially solve the problem. Another possible solution is to pass the burden of implementing frequency offset cancellation techniques to demodulation and detection schemes. This, in turn, helps to ease the constraints on the crystal by making it possible to adopt a less precise one, which ultimately makes the system more power-efficient [13].

The Doppler shift effect happens when the receiver and the transmitter have a non-zero relative speed. As a consequence, the signal acquired by the receiver results to have a different frequency to the originally transmitted one [14]. When the Doppler shift effect is constant or changes very slowly, it can be treated with the same method used for the frequency offset. The method depends on the used modulation as follows. For differentially encoded BFSK modulation, the Short-Time DFT (ST-DFT) is utilised [15], while for the Double Differential

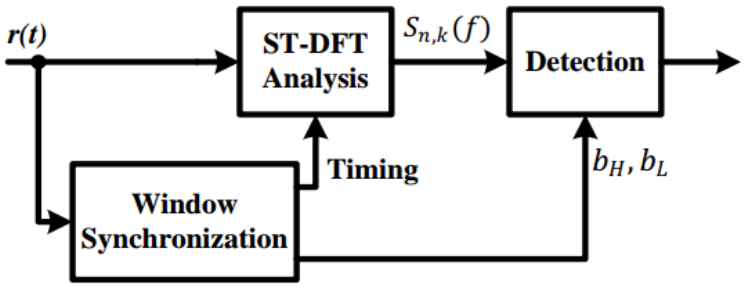

Figure 1.4: Diagram of the proposed demodulator [4].

BPSK the Auto-Correlation Demodulator is employed [16, 17]. The ST-DFT demodulation is the centre of this thesis and will be explained further in the next section.

1.3 ST-DFT demodulation scheme

In the previous section it was illustrated how the frequency offset problem may negatively affect the communication. In this section, an algorithm presented for BFSK modulation will be presented which is capable of tolerating large frequency offset by employing the Short-Time DFT [4]. In Fig. 1.4 its block diagram is shown. Initially, samples are acquired. A window is then aligned over a symbol, in the synchronisation phase. These samples are then processed by the ST-DFT block, which produces the power spectral density of the signal. By using this representation, it is possible to determine at which frequencies the transmitter is sending the data. Hence, a frequency offset would not hinder the correct data detection, because the frequencies are determined whenever a new data packet is received. As soon as the synchronisation is complete, the information regarding the detected frequencies is then passed to the detection block, indicated by b_L and b_H. At this point, the ST-DFT is used to determine the actual bits contained in the data packet. The algorithm will be explained in more details in Section 2.1.

1.4 Radio receiver

In this section, a radio receiver will be outlined which will give an idea of where the demodulator is placed. A generic schema for a radio receiver is presented in Fig. 1.5. A description of the flow of the signal from the antenna to the demodulator according to the previous figure is as follows.

1. An **antenna** is an electronic circuit that produces or captures radio waves. In our case it receives the electromagnetic waves and transform them in a signal.

2. The second element is a **bandpass filter**, which is used to allow a certain range of frequencies to pass through and reject those outside.

3. Next, the signal passes through a **low-noise amplifier** (LNA). It amplifies the signal, which is at the present stage very low on power. The characteristic of this type of amplifier is to enhance the useful signal, while adding as little noise as possible.

4. Up to this point the frequency of the signal is the same as the carrier wave. Processing such a high frequency signal is power consuming. Therefore a **mixer**, together with the

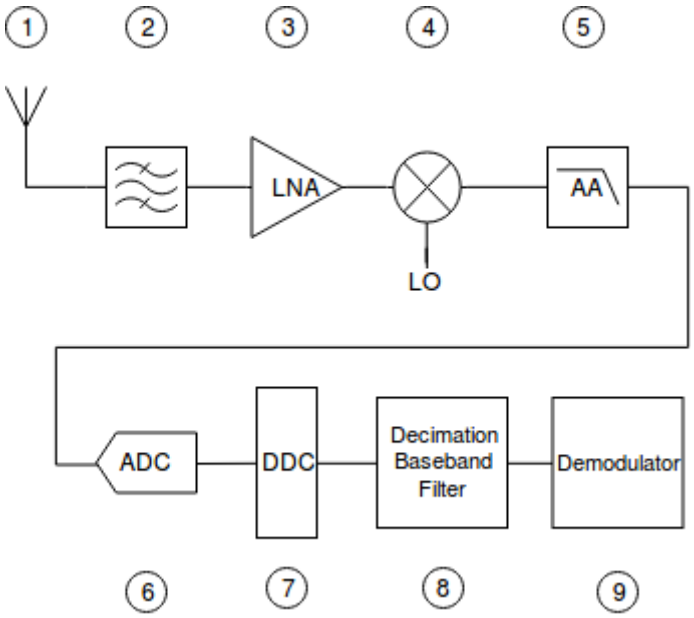

Figure 1.5: Schema of a generic radio receiver.

local oscillator, is used to shift the signal from the carrier frequency to an intermediate frequency (IF).

5. Subsequently, a **low-pass filter** is applied to counter the side effect of the previous operation. The mixer brings the signal to a certain frequency f_{IF}. As a result of the mixing operation, an additional frequency component located in $2f_c - f_{IF}$ is generated, where f_c is the carrier frequency. Hence, the low-pass filter is used to remove it.

6. At this point, the signal is still in the analog domain. As the demodulator is digital, an **ADC** is used to convert the signal to the same domain.

7. Next, a **digital down converter** (DDC) is employed. Recall that after the mixer, the signal is now located in an intermediate frequency. A DDC is employed to bring the signal in the base band, that is to centre it at the zero frequency. This operation is done to further reduce the frequency, which helps to process the signal at lower speed. This in turn, helps to reduce power consumption.

8. Before reaching the demodulator, the signal passes through a **decimation filter**. Its purpose is to reduce the sampling rate in order to accommodate slower components of the system. Additionally, it helps reduce the processing power, as less samples imply less memory is needed and a slower clock can be employed.

9. Finally, the digital signal reaches the **demodulator** and the relevant information can be extracted.

1.5 Research questions

The following objectives are defined in order to design and implement the proposed demodulator. While designing the system, power-efficiency is the first aspect that will be kept in mind. Previously, it was mentioned that the algorithm uses two DFT; in the synchronisation and in the detection blocks. This is usually considered a very power-hungry operation. Therefore, the first objective is to find a more power efficient implementation for the two DFTs.

Another important aspect while designing a system is the determination of an appropriate data representation, which is essential for two reasons. Firstly, an incorrect data representation may produce incorrect results, for example due to overflow as a consequence of too small registers. Secondly, it influences the power consumption as more registers and interconnections are needed.

Finally, the produced implementation of the system will be studied. As a power-efficient demodulator is designed, it is relevant to know its power consumption for the technology used to implement it. Additionally, its performance needs to be analysed in order to determine which ADC frequencies can be supported.

To summarise, the research questions are as follows:

RQ1. How can the two DFTs be implemented in a more power-efficient way?

RQ2. What is the smallest number representation so that the BER is the same as when full precision is used?

RQ3. What is the performance and the resource usage of the proposed implementation?

1.6 Thesis outline

In this chapter, the concept of WSN was introduced together with the frequency offset problem. Additionally, the demodulation algorithm capable of solving such problem was briefly described. In Chapter 2, the literature review on the subject is presented. In particular, the focus is on the improvement of the two DFTs used in the algorithm. Chapter 3 will present the design, implementation, and verification of the demodulator will be presented. After that, Chapter 4 includes the obtained results. It comprises the study of the BER while varying the SNR, together with power analysis of the system. Finally, in Chapter 5 the conclusion and future work will be presented.

Chapter 2

Background

In this chapter, the possibility to use more power-aware solutions for the two DFTs, in the synchronisation and the detection blocks will be investigated. As it will be shown, both cases include a special function that enables additional optimisation compared to the standard radix-r FFT.

In Section 2.1 the adopted demodulation algorithm will be further explained. In Section 2.2, the DFT in the synchronisation block will be analysed. Subsequently, in Section 2.3, a solution for the detection phase will be presented.

For the reminder of this text, the conversion from complex to real operation is as follows. It is assumed that when converting from a complex operation to a real one, a complex addition (C_A) results in two real additions (R_A), while a complex multiplication (C_M) is equivalent to two real additions and four real multiplications (R_M).

Additionally, the following notation is used unless otherwise specified

$$W_N^{kn} := e^{\frac{j2\pi kn}{N}}.$$

2.1 Demodulation algorithm

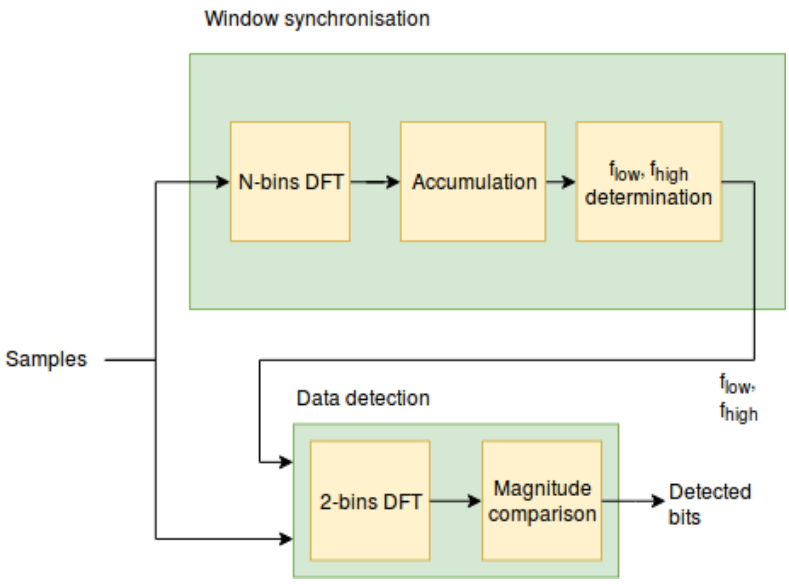

Figure 2.1: Schema of the demodulator.

The ST-DFT demodulation scheme is used to achieve a frequency offset tolerant communication. It was originally proposed in [15] and improved in [4]. It can be used for differentially encoded BFSK modulation. The algorithm is composed of two phases, namely window synchronisation and data detection. The diagram that shows their interconnections is presented in Fig. 2.1. Before describing the role of the two phases, the ST-DFT and zero-padding will be explained.

2.1.1 ST-DFT and zero-padding

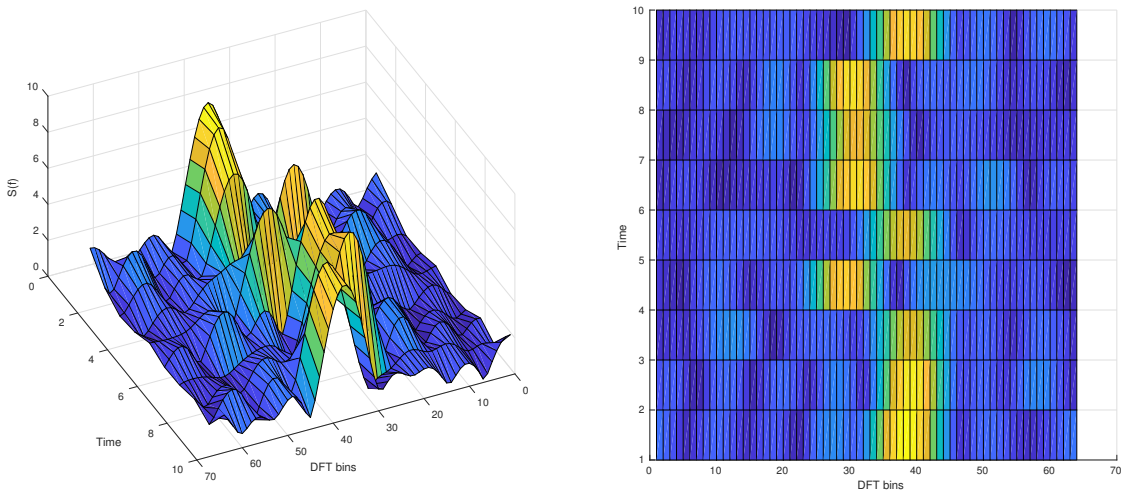

Figure 2.2: (left) Power spectral density obtained by using the ST-DFT. (right) Same representation viewed from above.

The ST-DFT is a linear transform that produces a time-frequency representation of a signal [18]. This information is useful for example with BFSK modulation. As data is encoded by varying the frequency of the transmitted signal over time, this frequency shift can be easily seen in the frequency domain. An example is shown in Fig. 2.2.

This operation is used both in the synchronisation and detection phase. In the first one, it is needed to detect the symbols present in the preamble of the transmitted packet. In this case, a full N-point DFT is required. In the detection phase, it serves to detect the bits transmitted. As this information is passed on two specific frequencies, in this case the algorithm needs only two bins of the calculated DFT.

The DFT is performed on zero-padded samples, both in the synchronisation and in the detection blocks. This operation is done to interpolate the bins in the spectrum. To understand the effect of this operation, an example is shown in Fig. 2.3. In this case, the window is composed of eight samples. The non-interpolated line was obtained by computing the 8-point DFT, without applying any zero-padding. The same DFT result is shown with the purple diamonds, but this time the magnitude of the bins have been spaced so that they are eight points apart from each other. Finally, the blue line with full points indicates the interpolated DFT of the same eight samples with a zero-padding factor of 8, which corresponds to a 64-point DFT.

The peak in the non-interpolated line corresponds to the one in the spaced line on bin 25. The same peak corresponds to the one on bin 28 on the interpolated curve. This shift even though it looks barely relevant, might compromise the detection. In fact, if bin_low and bin_high are not far enough apart, they may be detected incorrectly. This is the case with the non-interpolated line. As only eight points are available, it is more likely that two actually

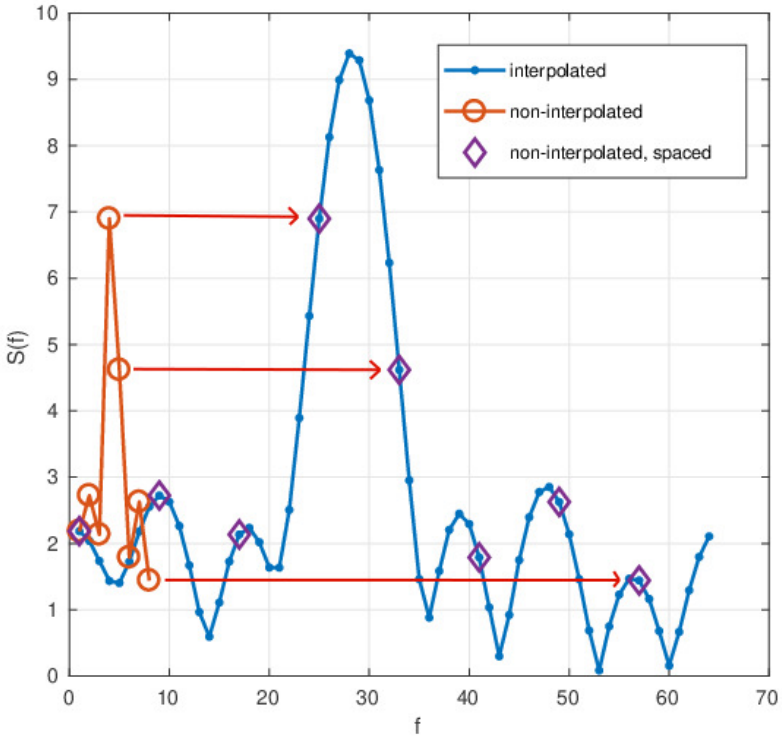

Figure 2.3: Comparison of different DFT.

different frequencies will fall into the same bin. Instead, thanks to the interpolation produced by the zero-padding, the power spectral density is well spread and the positions of the peaks can be better distinguished. In conclusion, the zero-padding was used to increase the resolution of the incoming signal.

2.1.2 Window synchronisation

The task of the window synchronisation phase is to align the sample window over a symbol. Moreover, this phase detects the bins corresponding to the frequencies used by BFSK, indicated by b_L and b_H in Fig. 2.1.

To obtain this information, a rectangular window is shifted over the initially arrived samples that belong to the preamble, which is a sequence of alternating zero and one symbols. Each symbol is composed of a fixed number of samples indicated with M. In Fig. 2.4 this shifting procedure is shown for three different delays. When the window exactly aligns with a symbol (delay = 0), its spectrum shows a single clear peak, either on the low frequency $flow$ or on the high frequency $fhigh$, which are unknown at this stage of the algorithm. When the window is not aligned, multiple peaks are present. Therefore, the idea is to slide the window over the whole preamble and understand which of the M delays is the best one.

In Fig. 2.5, the windowing system when M=8 is depicted and will be used to explain the procedure to determine the correct delay. The first window, window 1, contains the first eight samples, labelled from 0 to 7, that have arrived and will contribute for the delay 1. They are then zero-padded and a DFT is subsequently computed using them. The values resulting from the DFT are complex and their magnitudes are afterwards calculated and stored in register 1. The same procedure is then repeated for delay 2. Window 2 is produced, using samples from 1 to 8, they are again zero-padded, the DFT is computed, and finally the magnitudes

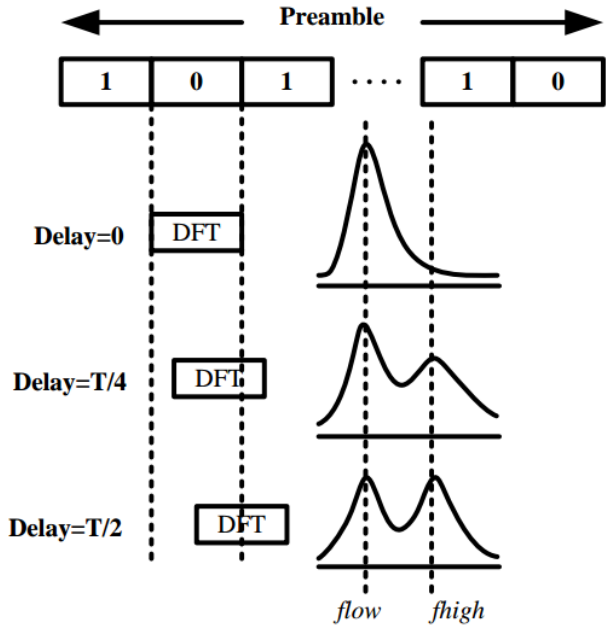

Figure 2.4: Variation of spectrum when the window is delayed [4].

are obtained and stored in register 2. This operation is repeated for M=8 times and involves samples labelled from 0 to 14. This was the first accumulation round, which corresponds to the first symbol. As there are multiple symbols in the preamble, an equal amount of rounds needs to be computed. In the subsequent rounds, the magnitudes are accumulated with the previously computed values. The accumulation is separate for odd and even numbered symbols.

At the end of all the symbols of the preamble, it is determined which register contains the biggest value, which indicates the delay that best alignes with the preamble symbols. Together with this information, the bins that represent f_{low} and f_{high} are determined (also referred to as bin_low and bin_high).

2.1.3 Data detection

The second phase receives timing information necessary to align the window on the incoming data. The ST-DFT produces a time-frequency representation of the incoming signal, so that the detection of the bits can be carried out. An example of this representation is shown in Fig. 2.2. Each axes that starts from the time line indicates a new ST-DFT. The frequency axis shows the bins of the DFT, while the vertical axis is the magnitude of each bin.

The DFT receives M samples, zero-pads them, and computes the result. Compared to the window synchronisation, in this case only two bins of the DFT are required, those indicated by bin_low and bin_high. The actual detection happens by comparing the magnitudes of the previously computed two bins. Whenever the first is greater than the second, a bit “1” is detected, otherwise it is a “0”.

2.2 DFT in synchronisation block

A DFT is used in the synchronisation phase. It is utilised to adjust the window over one symbol period. The windowing system processes group of M complex samples at a time. An example with M=8 is shown in Fig. 2.6.

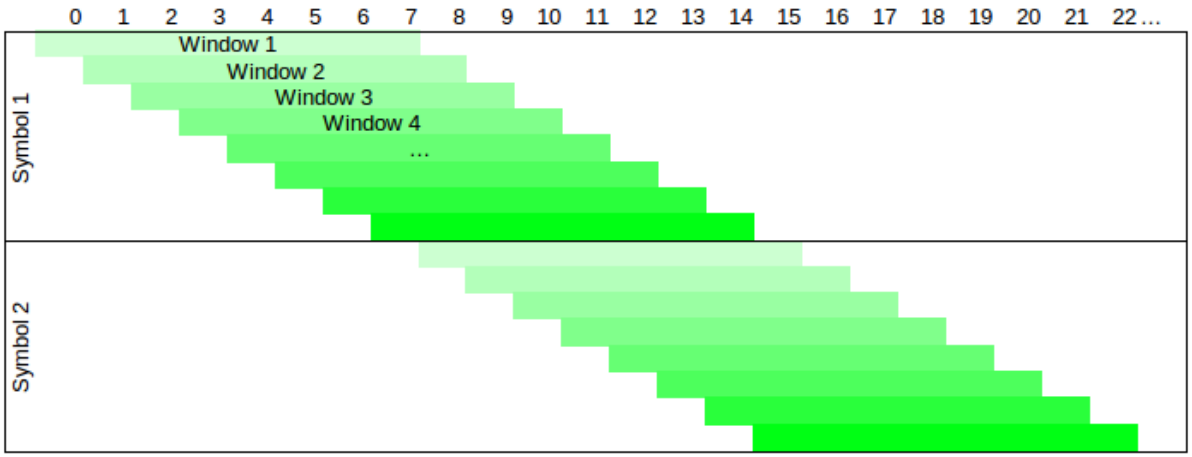

Figure 2.5: Shifting of the window when the number of samples per symbol is 8.

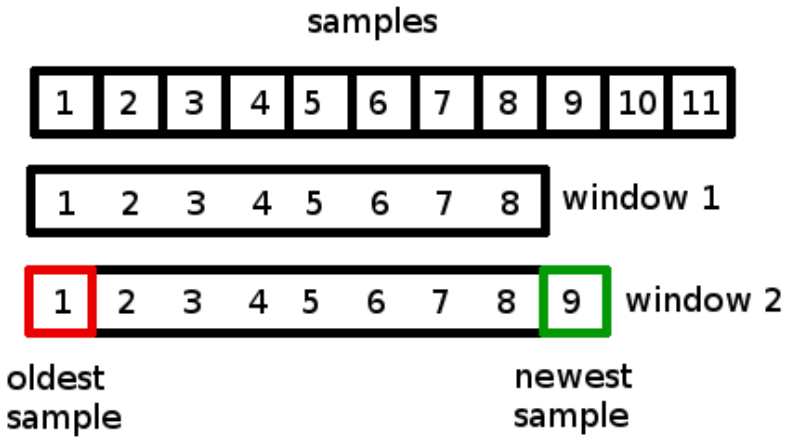

Figure 2.6: Window update process.

In the first window (window 1) the system has to wait for the first eight samples before it can start to process them. Subsequently, for window 2, 3, and so on, the oldest sample of the previous window is removed while all the other samples are shifted. This can be seen in window 2, where the second sample is not in the second slot of the window, but it is now in the first one. Then, the newest sample is inserted in the eighth slot. At this point they can be processed and as soon as the 10^{th} sample arrives, window 3 can be formed in the same manner. The explained procedure is in fact, sliding a window over a series of samples. This encourages us to investigate the possibility to use the sliding DFT.

2.2.1 Sliding DFT

As explained, each window overlaps with the previous one by the M-1 samples that they have in common. This suggests that the DFT bins obtained for the n^{th} window, can be reused for $(n+1)^{th}$ window. This is the idea behind the sliding DFT (SDFT) as presented by Jacobsen and Lyons in [19]. It is a reinterpretation of the DFT as a filtering operation. This method is useful when the variation of a DFT bin during a time interval needs to be tracked. A conventional DFT necessitates repetition of all computations for each new sample. However, using the sliding DFT, the computation decreases to a simpler "update" operation.

Different versions of the sliding DFT have been proposed. They differ in terms of computa-

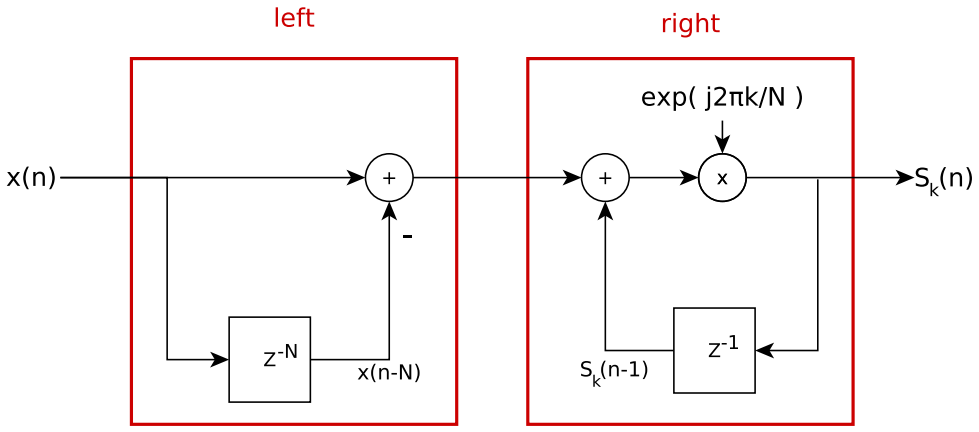

Figure 2.7: Resonator for a single bin for the original SDFT.

tional cost and numerical stability. In the next section the original work will be presented and will be followed by the modulated SDFT (mSDFT).

Original SDFT The first sliding DFT was proposed in [19] and updated in [20]. Its derivation is now briefly explained. Let $S_k(n)$ be the bin computed at time n for frequency index k for an N-point DFT. With a standard DFT it would be calculated as

$$S_k(n) = \sum_{l=n-N+1}^{n} x(l) e^{\frac{-j2\pi kl}{N}} \tag{2.1}$$

To calculate $S_k(n+1)$, the same operation needs to be computed. However, $S_k(n+1)$ and $S_k(n)$ can be related thanks to the circular shift property of the DFT. As mentioned before, the n^{th} window removes the oldest sample x(n-N) and adds the last arrived, x(n). Additionally, samples that are in common need to be shifted, which is done by multiplying $S_k(n-1)$ by $e^{j2\pi kn/N}$. The equation to compute bin k at time n from the previous one at time n-1 is [19]:

$$S_k(n) = S_k(n-1) e^{\frac{j2\pi k}{N}} - x(n-N) + x(n). \tag{2.2}$$

This formula is correct as long as only the magnitude of the bin is required. A better expression that corrects the phase is given by [20]:

$$S_k(n) = e^{\frac{j2\pi k}{N}} [S_k(n-1) - x(n-N) + x(n)]. \tag{2.3}$$

The cost of this solution can now be derived, with reference to Fig. 2.7. For K frequencies ($1 \leq K \leq N$), after L samples ($L \geq N$), when an N-point window is used and assuming complex inputs, the cost is as follows. The total cost is divided into two parts. The cost of sign inversion and first addition (left) and the rest of operations (right). The number of operations for the left part is $C_L = (L-N)C_A$, where C_A indicates complex addition. Since the delay register is empty for the first N samples, the addition and the multiplication do not contribute for the first N samples but only for the subsequent L-N samples. Additionally, notice that the left part is not affected by the k index and as such its results can be shared among all the resonators. The sign inversion was not counted as a standard operation. This is consistent with the approach that many authors take while evaluating the performance of the various FFT variations. The cost

for the right part is $C_R = K((L-1)C_A + LC_M)$, where C_M indicates complex multiplication. The final cost is:

$$C_{(tot)} = C_L + C_R = C_A(L - N + K(L-1)) + C_M(KL).$$

Note that additional simplifications can be obtained by taking into account the fact that for certain combinations of k and N, the complex factor $e^{\frac{j2\pi k}{N}}$ becomes a trivial multiplication by 1, -1, j or -j.

In terms of memory, the system requires N+1 registers for a single resonator. If multiple resonators are used ($K \leq N$), then N+K registers will be needed, as the left part is used for all the resonators.

The stability of this filter is now discussed. The Z-transform of the k-th bin is given by:

$$H(z) = \frac{1 - z^{-N}}{1 - e^{\frac{j2\pi kn}{N}} z^{-1}}$$

it has a single pole at $z = e^{\frac{j2\pi kn}{N}}$, which makes the filter marginally stable. In fact, the stability is affected by coefficient precision (number of bits used for representation in hardware), if the precision is not enough the pole moves outside the unit circle which leads to instability. However, if the numeric coefficient is not excessively rounded, the filter is BIBO stable (Bounded Input Bounded Output).

Modulated SDFT (mSDFT) The block diagram of the modulated SDFT is shown in Fig. 2.8 [21].

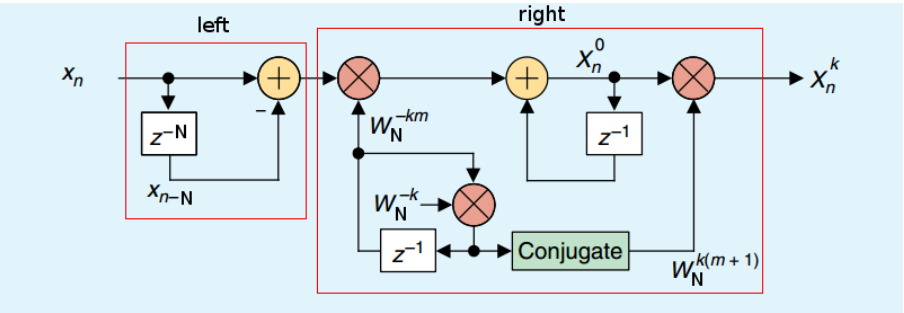

Figure 2.8: Resonator for a single bin for the mSDFT [21].

This variation of the SDFT takes advantage of the DFT modulation property. The k^{th} bin is shifted to the position zero. This removes the twiddle factor from the feedback, hence avoiding accumulated errors due to its finite representation. This can be seen by using $k = 0$ in Eq. 2.3, it becomes $S_0(n) = S_0(n-1) - x(n-M) + x(n)$ and the twiddle factor does not influence the computation anymore. In order to produce the desired shift, Eq. 2.3 needs to be multiplied by the factor W_N^{-km}, where m is the phase of the shift. When $m = 0$, the phase of the modulating sequence is 0, when the index increases, also the sequence increases by a factor of W_N^{-k}. In order to correct the phase, a multiplication by $W_N^{k(m+1)}$ is required to produce $X_n^k = W_N^{k(m+1)} X_n^0$.

Also for this SDFT, the left side is independent of the bin index and the result can be shared among multiple resonators. This can decrease hardware complexity by reusing the same component. The number of operations can be calculated similarly to the original SDFT. The cost is calculated for K frequencies ($1 \leq K \leq N$), L samples ($L \geq N$), when a N-point DFT is used and assuming complex inputs. For the left part the cost is $C_L = LC_A$. For the right part the cost is $C_R = K(3L-2)C_M + KLC_A$. The total cost of this solution is:

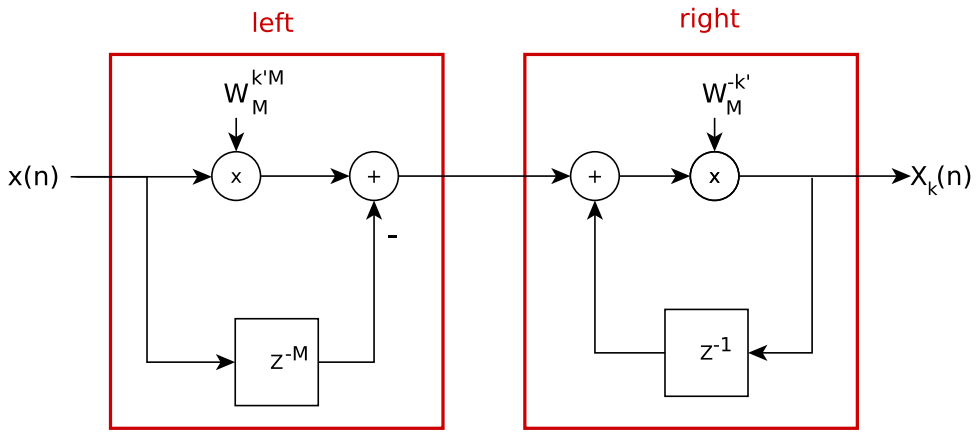

Figure 2.9: Block diagram of the original DFT when zero-padding is taken into account.

$$C_{(tot)} = C_L + C_R = K(3L-2)C_M + L(K+1)C_A. \tag{2.4}$$

In terms of memory, if K resonators are used, then N+2K registers are needed.

2.2.2 Sliding DFT in presence of zero-padding

The SDFT methods can be used to update previously computed bins whenever a new sample arrives. The last sample of the previous window is removed, all the old samples are shifted, and the new one is introduced. This operation though does not take into account the zero-padding operation. Such an operation, as it was illustrated in 2.2, is necessary in the present case and therefore has to be incorporated.

Currently, none of the existing SDFT methods can be utilised in the algorithm. In fact, the zero-padding makes the concept of sliding a window incorrect. This problem can be resolved by using the result proposed in [22]. The authors suggest that since the zero padding only adds zeroes, part of the computation of a single bin can be avoided.

The idea behind this optimisation is as follows. Let $N = Z + S$ be the number of point of the DFT, Z the number of zeros and S the number of non-zero samples. Then a generic bin is calculated, in the standard DFT, as:

$$X(k) = \sum_{n=0}^{N-1} x[n] e^{-\frac{j2\pi kn}{N}} \tag{2.5}$$

the summation can be divided in two parts, because of the tailing zeros:

$$X(k) = \sum_{n=0}^{S-1} x[n] e^{-\frac{j2\pi kn}{N}} + \sum_{n=S}^{N-1} x[n] e^{-\frac{j2\pi kn}{N}}$$

In the second term all values are zeroes and therefore, only the first term needs to be considered.

This idea was incorporated in the SDFT so that the zero-padding is maintained together

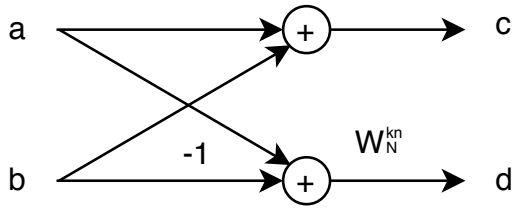

Figure 2.10: Butterfly.

with the sliding property. The derivation is as follows.

$$\begin{aligned} X_k &= \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi nk}{N}} \\ &= \sum_{n=0}^{M-1} x(n) e^{-j\frac{2\pi nk}{N}} \\ &= \sum_{n=0}^{M-1} x(n) e^{-j\frac{2\pi nk'}{M}} \end{aligned} \tag{2.6}$$

Where N indicates the DFT points, M the number of non-zero samples (N M), $k' = \frac{kM}{N}$. The first equality is the N-point DFT. The second equality is obtained by noting that $N - M$ samples are zero because of the zero-padding. The third equality is a rearrangement of the indices.

The original SDFT can now be revisited in light of this new achievement by starting with Eq. 2.3.

$$\begin{aligned} X_k(n) &= X_k(n-1) e^{j\frac{2\pi k'}{M}} - x(n-M) e^{j\frac{2\pi k'}{M}} + x(n) e^{-j\frac{2\pi (M-1)k'}{M}} \\ &= (X_k(n-1) - x(n-M) + x(n) W_M^{Mk'}) W_M^{-k'} \end{aligned} \tag{2.7}$$

The block diagram of the modified SDFT is shown in Fig. 2.9, with $k' = \frac{kM}{N}$. The trade-off here is that while the number of total operations is reduced, a correcting factor appears, which increases the complexity. Moreover, the new multiplication introduces the k factor on the left side, meaning that the computation of this first stage cannot be shared among all the resonators. Only the M registers and the sign inversion operation can be shared.

The cost of the revisited SDFT is as follows. On the left side the cost is $C_L = (L-M)C_A + KLC_M$, while on the right side $C_R = K((L-1)C_A + LC_M)$. The total is as follows.

$$C_{tot} = C_L + C_R = (2KL)C_M + (L - M + K(L-1))C_A \tag{2.8}$$

where M is the number of non-zero samples and N the order of the DFT, L is the number of total samples passed to the resonators, and K the number of frequency of interest.

2.2.3 FFT zero padding (FFT-ZP)

The second approach that was studied started by considering the standard FFT radix-2. The reason for choosing radix-2 instead of radix-4 or radix-8 is related to its simplicity. As the order increases, also the structure of each solution becomes more complex.

In the present case, the incoming samples are zero-padded and the position of the zeros are always fixed. By considering this information, a modification to the radix-2 FFT is proposed. It will be shown that a considerable amount of multiplications and additions become trivial.

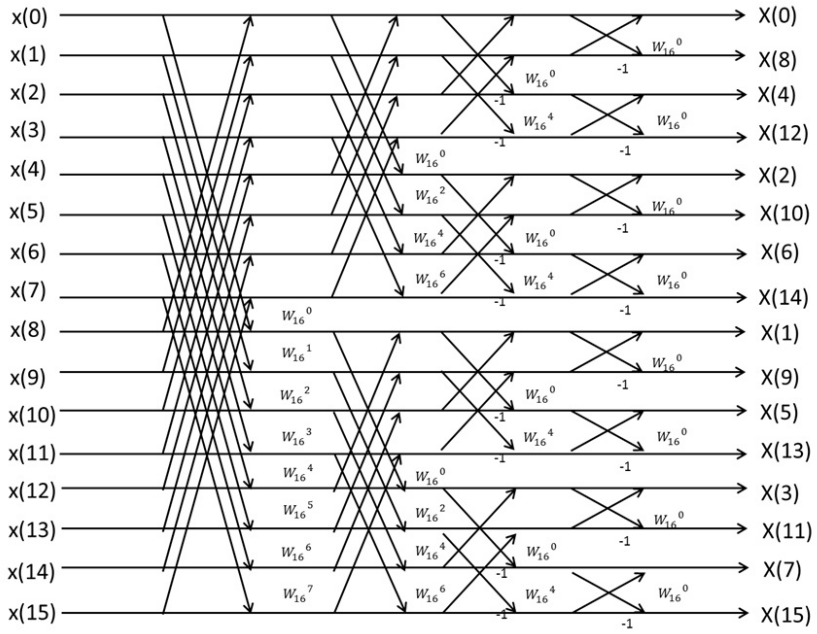

Figure 2.11: 16-points FFT radix-2 DIF.

In the standard radix-2 N-point FFT, there are l stages, where $l = \log_2 N$. An FFT stage is a collection of butterflies. An example is depicted in Fig. 2.11. The FFT in the picture is 16-point and the number of stages is 4. The structure of a butterfly is illustrated in 2.10. The interpretation of its flowgraph is as follows; its meaning extends as well to the previously shown FFT stage. The resulting values are $c = a + b$ and $d = (a - b)W_N^{kn}$. Usually the plus signs are omitted and the intersection of an arrow into another indicates that a sum is occurring. Instead a factor, like W_N^{kn} of the example, which lies above an arrow, indicates that it is multiplying the value carried by the arrow.

When M non-zero samples are given to an N-point FFT, with $M = 2^k$ and $N = 2^l$ and $M < N$, then there exist $l - k$ transfer stages and k normal stages. A transfer stage is a stage in which there are no butterflies, cfr. Figs. 2.11-2.13. Because of the presence of the input zeros, in the first $l - k$ stages there are no additions, as they result in passing through the samples. In this case though, the twiddle factors need to be multiplied with the non-zero samples. After $l - k$ stages, the following stage is a normal one because all its input are at this point non-zero.

The cost of this solution is calculated considering that all operations are complex. There are additions only in the last k stages and for each one there are N additions. Considering $M = 2^k$ and $N = 2^l$, the number of multiplications can be calculated. There are M multiplications in the first transfer stage, 2M in the second one, 4M in the third, and so on. The total number of transfer stages is l-k. Thus, there are $M \sum_{n=0}^{l-k-1} 2^n$ multiplications in all the transfer stages. In the normal stages there are twiddle factor multiplications which account for $\frac{N}{2}$ multiplications for each of the last k stages.

Overall, the total number of additions and multiplications are:

$$C_A = kN. \tag{2.9}$$

$$C_M = M \sum_{n=0}^{l-k-1} 2^n + \frac{N}{2}k, \tag{2.10}$$

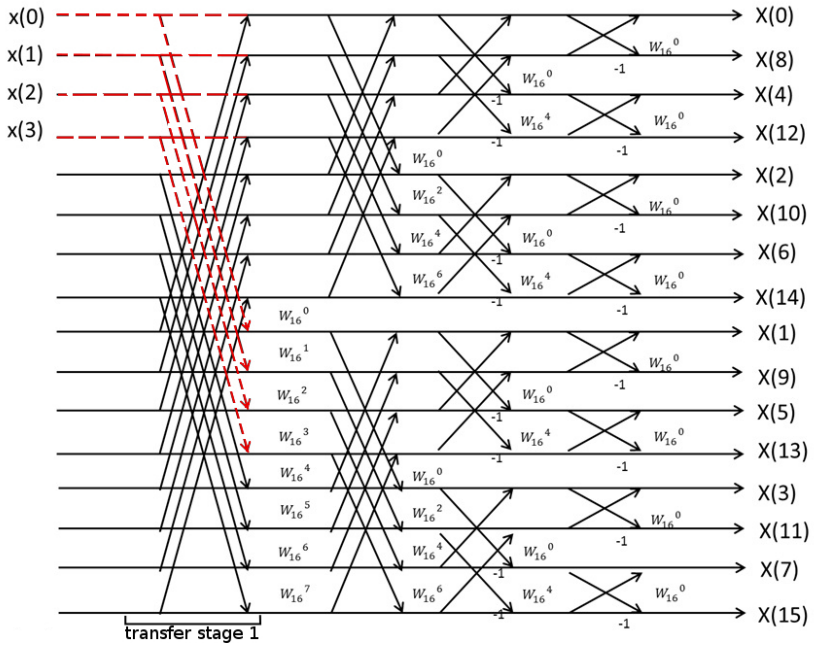

Figure 2.12: 16-points FFT radix-2 DIF. First stage transfers.

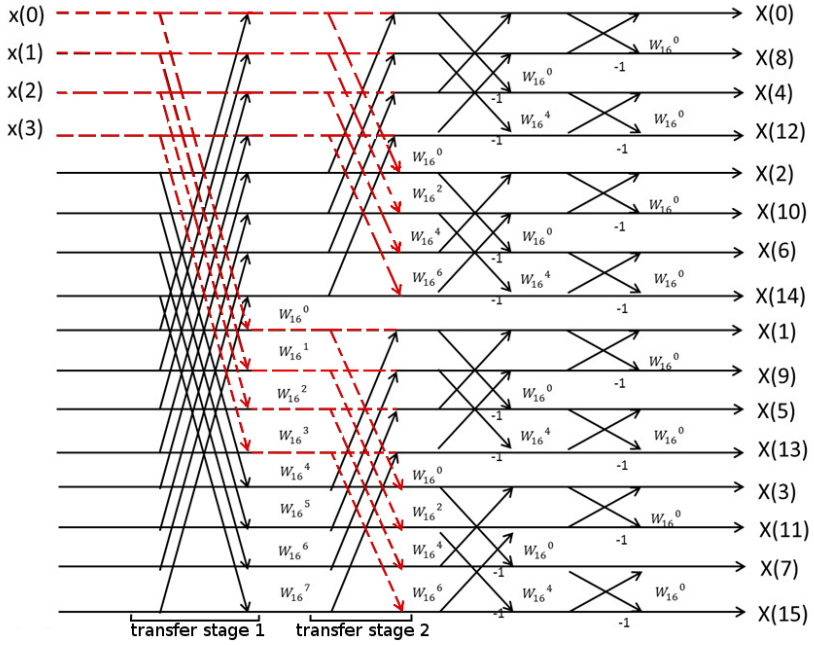

Figure 2.13: 16-points FFT radix-2 DIF. Second stage transfers.

2.2.4 Comparison

In Table 2.1 all the analysed methods for the synchronisation block are summarised. Note that none of the methods take into account the operation reduction due to trivial multiplications by the factors 1, -1, j, -j generated by twiddle factor simplifications.

In general, multiplications are regarded as more power intensive operations than additions. A number of algorithms have been implemented to reduce the complexity of both operations. For the addition, many adders have a logarithmic time. Namely, an n bit input Kogge-Stone parallel prefix graph has a delay of $\log_2 n$ as well as Ladner-Fischer, while Brent-Kung adder has $2\log_2 k - 2$ [23–25]. For the multiplication, the fastest algorithms reach a computational time proportional to $n \log n$. Mixed-level Toom-Cook implemented by D. Knuth reaches $O(n \log n 2^{\sqrt{2 \log n}})$, the Schönhage–Strassen algorithm improves to $O(n \log n \log \log n)$, and Fürer's algorithm takes $O(n \log n 2^{2 \log^* n})$ [26–28].

These two operations can be compared only when their implementation has been decided. Until that time, they need to be regarded as two incomparable costs. As a consequence, the cost of the DFTs cannot be expressed with a single number and therefore cannot be established unambiguously which one has a lower complexity.

Additionally, the complexity itself also depends, other than on the algorithm, on the technology used (7 nm, 10 nm, 14 nm and so on) and the target IC (FPGA, ASIC, CPU, GPU), among others. It is left to the designer to pick the best combinations depending on the circumstances. In this work, the actual implementation of adders and multipliers was left to the synthesis tool.

The decision of which of these DFTs best suit the present context is postponed to Section 3.1.1, when the parameters of the system will be known, i.e. variables N, M, L, J, etc.

	C_M	C_A	Note
Radix-2	$\frac{N}{2}lJ$	NlJ	[29]
Radix-4	$\frac{3}{8}NlJ$	$\frac{3}{2}NlJ$	[29]
Split-radix	$\frac{N}{4}(l-1)J$	NlJ	[30]
SDFT	KL	$L - N + K(L-1)$	2.2.1
SDFT revisited	$2KL$	$L - M + K(L-1)$	2.2.2
mSDFT	$K(3L+1)$	$L(K+1-N)$	2.2.1
FFT-ZP	$J(M\sum_{n=0}^{l-k-1} 2^n + \frac{N}{2}k)$	JNk	2.2.3

Table 2.1: Comparison of the DFT calculation methods studied that could potentially be used in the implementation. N is the DFT order and M is the number of non-zero samples, with $M = 2^k$ and $N = 2^l$. For the SDFTs, L is the total number of samples present in the preamble, for which the sliding operation is necessary; K is the number of bins (it may be less or equal to N). For the standard DFTs, J is the number of times the DFT needs to be repeated to complete the preamble.

2.3 DFT in data detection block

The DFT is also used in the data detection block. In this case it is used to identify whether the received symbol is 0 or 1. The decision is made by comparing the magnitude of the two bins determined in the synchronisation phase. M non-zero samples are zero-padded and sent to a DFT block. However, in this case only two bins are of interest, bin_low and bin_high. In this section, different methods that optimise the computation based on these conditions are compared.

In the remainder of this section, N denotes the length of the DFT and M is the window size.

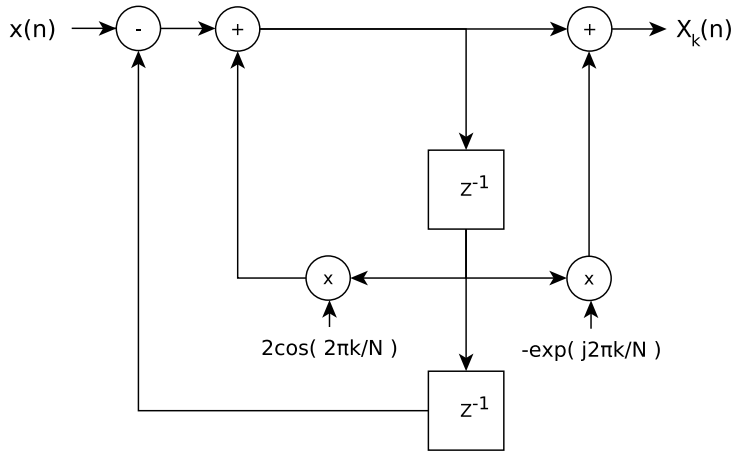

Figure 2.14: Schema of the Goertzel algorithm.

2.3.1 Single bin DFT

A single bin of an N-point DFT can be calculated using Eq. 2.5. It involves N complex multiplications and N-1 complex additions. Therefore, the computation cost in terms of real operations is $4NR_M + (4N-2)R_A$.

In Section 2.2.2, it was shown that the single bin DFT can be improved in presence of zero-padding. In this case the cost becomes equal to M complex multiplications and M-1 complex additions.

2.3.2 Goertzel algorithm

The Goertzel algorithm was firstly presented in [31]; its derivation can also be found in [32]. With this technique it is possible to obtain individual terms of an N-point DFT. The advantage is the reduced amount of operations. This is true as long as the number of bins of interest is equal or less than $\log_2 N$, otherwise the FFT outperforms this solution [32]. Only two bins out of the 64 produced by the DFT were required in the data detection, and because of that the previous condition is satisfied. Its schema is shown in Fig. 2.14. The algorithm is a two-stage filter. The first one is:

$$s[n] = x[n] + 2\cos\omega_0 s[n-1] - s[n-2],$$

which produces an intermediate result. In the second one, the output of the previous stage is used to compute the final value:

$$y[n] = s[n] - e^{-j\omega_0} s[n-1],$$

where $\omega_0 = \frac{2\pi k}{N}$ and s[n] is the output of the first stage.

The first stage is repeated N times. The multiplication by the cosine factor is real, while the input is complex, therefore two real multiplications are needed. The subtraction is equivalent to a complex addition, as both the input and the value in the bottom register are complex. Finally, the second addition is complex. Instead, the second stage requires a complex addition

and a complex multiplication. The total cost is as follows

$$C_{tot} = N(4R_A + 2R_M) + C_A + C_M = (4N + 4)R_A + (2N + 4)R_M. \tag{2.11}$$

The Goertzel algorithm was improved by using the technique described in 2.2.2. The derivation is as follows. The starting point is the same as for the SDFT Eq. 2.6, with $k' = kM/N$ and $W_M = e^{-j\frac{2\pi}{M}}$.

$$\begin{aligned} X_k &= \sum_{n=0}^{M-1} x(n) e^{-j\frac{2\pi n k'}{M}} \\ &= \sum_{n=0}^{M-1} x[n] W_M^{nk'} \\ &= W_M^{-k'N} \sum_{n=0}^{M-1} x[n] W_M^{nk'} \\ &= W_M^{-k'(N-M)} \sum_{n=0}^{M-1} x[n] W_M^{-k'(M-n)} \end{aligned} \tag{2.12}$$

We define

$$y_k[m] := \sum_{n=-\infty}^{+\infty} x[n] W_M^{-k'(m-n)} u[m-n] \tag{2.13}$$

By combining Eq. 2.12 and 2.13

$$X_k = W_M^{-k'(N-M)} y_k[m]|_{m=M} \tag{2.14}$$

Notice that

$$\begin{aligned} W_M^{-k'(N-M)} &= e^{j\frac{2\pi}{M}\frac{M}{N}k(N-M)} \\ &= e^{-j\frac{2\pi M}{N}k} \end{aligned} \tag{2.15}$$

Finally, from Eq. 2.14

$$X_k := e^{-j\frac{2\pi M k}{N}} y_k[m]|_{m=M} \tag{2.16}$$

In Eq. 2.16, it can be seen that the new multiplication factor appears just before the output. This operation can be avoided as it only affects the phase of the result and not the magnitude, which is the only information needed by the algorithm. Therefore, such operation is not included in the cost of the algorithm. Previously, the loop repeated N times, while now only M iterations are needed.

$$C_{tot} = M(4R_A + 2R_M) + C_A + C_M = (4M + 4)R_A + (2M + 4)R_M. \tag{2.17}$$

2.3.3 Comparison

In Table 2.2, a summary of the previously presented DFT is shown. In this case the operations are all real. This conversion was necessary because the Goertzel algorithm contains a real multiplication, which cannot be converted in term of complex operations.

For both the single bin DFT and the Goertzel algorithm, the improvement produced by the optimisation indicated in 2.2.2, makes the cost become linear in the number of non-zero input values (M) rather than the number of points of the DFT (N).

The decision of which of these DFTs best suit the present context is postponed to Section 3.1.2, when the parameters of the system will be known, i.e. variables N, M, L, J, etc.

	R_M	R_A	Note
Single bin DFT	4N	4N-2	2.3.1
Single bin DFT rev.	4M	4M-2	2.3.1
Goertzel	$2N+4$	$4N+4$	2.3.2
Goertzel modified	$2M+4$	$4M+4$	2.3.2

Table 2.2: Number of real operations for the data detection, for a single bin and a N-point DFT, where only the first M samples are non-zero. $M = 2^k$ and $N = 2^l$.

Chapter 3

Design, implementation, and verification

In this chapter, the proposed implementation of the demodulation algorithm described in Section 2.1 is presented. This chapter opens by presenting implementation considerations regarding the choice of the two DFTs 3.1. Following, the data representation will be discussed 3.2. Next, the system architecture is explained by elaborating on the design blocks that compose the demodulator 3.3. The chapter closes with a presentation of the tools used to verify the design and the methodology adopted for the verification 3.4.

3.1 Implementation

3.1.1 DFT in the synchronisation block

In Section 2.2, different alternatives for the DFT in the synchronisation block were considered. In the present section, their computational complexity will be revisited with defined parameters of the system. The number of samples per symbol, indicated with M, is 8, which is also the size of the window that is used in the SDFT and corresponds to the number of non-zero samples. The number of DFT points, indicated with N, is 64, while the number of times a DFT has to be repeated for a complete preamble, indicated with J, is 112. For the SDFTs, two additional parameters were defined. The number of samples in the preamble, indicated with L, is equal to 119, while the number of bins that need to be computed, indicated with K, is equal to 64.

Table 3.1: Comparison of the DFT calculation methods defined in Table 2.1 when evaluating for a **full preamble**. N=64, M=8, J=112, K=64, L=119, l=6, k=3. Input is a complex sequence, all the **operations are real**.

	R_M	R_A	Note
Radix-2	86,016	129,024	[29]
Radix-4	64,512	161,280	[29]
Split-radix	**35,840**	103,936	[30]
FFT-ZP	68,096	77,056	2.2.3
SDFT revisited	60,928	**45,790**	2.2.2

In Table 3.1 each implementation is reevaluated when considering the aforementioned parameters. Split-radix has the lowest number of real multiplications, while the revisited SDFT has the lowest count of real additions. From this table it is evident that the Split-radix and the SDFT are the best candidates to be implemented in the synchronisation block. To choose the best one, the complexity of their structures are now considered. The Split-radix is a combination of FFT radix-2 and radix-4 [33]. Instead, the SDFT has a straightforward structure which

comprises of only four operations, which is easier to implement. At this point, the SDFT is the best candidate to be implemented.

Before making a final decision, there is another aspect to be considered. In all the DFTs, there are twiddle factors that equal to 1, -1, j, or -j. These values can be efficiently computed by treating them as special cases. In Table 3.2 a summary is presented of the number of real operations required to compute a full preamble, when simplifications are taken into account. The FFT-ZP is the solution that best takes advantage of this simplification, while the SDFT results to be the worst. While the SDFT has the lowest count of additions, it has the worst count of multiplications, which is almost three times the number used by the split-radix or two times the FFT-ZP. Because of this, the SDFT was discarded. The split-radix and the FFT-ZP are the last two candidates. The second has one-third more multiplications than the split radix, but less than half of the additions of the split-radix. In numbers, the price to save 50k additions is roughly 6k multiplications. As it was explained before, comparing these two numbers is not possible until additional information is provided regarding the implementation of these operations. Therefore, their structures will be compared to make a final decision. The split-radix is a combination of radix-2 and radix-4, while the FFT-ZP is a simplified radix-2. In conclusion, the FFT-ZP was deemed the best compromise in terms of ease of implementation and computational cost.

Table 3.2: Comparison of the DFT calculation methods when evaluating for a **full preamble**. N=64, M=8, J=112, K=64, L=119, l=6, k=3. Input is a complex sequence, **operations are real** and trivial multiplications are removed.

	R_M	R_A	Note
Radix-2	29,568	115,584	[33]
Radix-4	23,296	109,312	[33]
Split-radix	**21,952**	107,968	[33]
FFT-ZP	27,328	56,672	2.2.3
SDFT revisited	57,120	**45,790**	2.2.2

3.1.2 DFT in data detection block

In Table 2.2, different methods to compute a single-bin DFT were presented. Similarly to what was done for the synchronisation block, these methods will now be revisited in light of the defined parameters. The number of samples per symbol, indicated with M, is 8, while the number of DFT point, indicated with N, is 64.

In Table 3.3 the cost for calculating a single bin DFT is reported in terms of real operations for each methods. The revisited Goertzel algorithm has 12 multiplications less than the revisited single bin DFT, while this second one has 6 less additions. Since multiplications are computationally more intensive than additions, the Goertzel algorithm was considered as the best solution between the two.

Table 3.3: Number of real operations for the data detection, for a single bin and a 64-point DFT, where only the first 8 samples are non-zero.

	R_M	R_A	Note
Single bin DFT	256	254	2.3.1
Single bin DFT rev.	32	**30**	2.3.1
Goertzel	132	260	2.3.2
Goertzel revisited	**20**	36	2.3.2

3.2 Data representation

3.2.1 Fixed-point representation

A fixed-point representation was chosen above floating-point. The latter has a lower power efficiency, which is due to the additional hardware required to carry out operations like addition and multiplication [34, 35]. Moreover, the dynamic range of the input data of the system was limited between -100 and 100. This is in contrast with the practice of using floating-point, which is more suitable when data with different order of magnitudes are present [34].

The fixed-point number representation is a data type used to represent real values. Its advantage resides in the hardware. Operations like addition, multiplication, square root algorithms and any other mathematical operation that can been used for integer arithmetic can also be used for fixed-point values. This is the case because fixed-point representation is equal to integer representation, except that the first is scaled by a predetermined factor. This factor though, does not affect the actual computation, but is only used to interpret the results. An example to clarify this concept will be presented later.

Similar to integers, fixed-point numbers can be signed or unsigned. Besides, signed values can be achieved either with two's complement or signed magnitude. Consider an unsigned N-bit integer value, its binary representation is given by

$$x = \sum_{i=0}^{N-1} b_i 2^i, \tag{3.1}$$

where b_i is the i^{th} bit. Each bit has a weight given by its position, i.e. bit i^{th} has weight 2^i. The weights in this case are only powers of two, between 1 and 2^{N-1}. With fixed-point, also negative powers of two are used, which make it possible to represent real values. For the remainder of this thesis, $(i|f)$ denotes a fixed-point number in which i indicates the number of bits used for the integer part and f for the fractional part. For example, given an unsigned number represented in binary as $(10010110)_2$ and FP representation $(5|3)$, then the bit weights are between 2^{-3} and 2^4, which gives the value $\sum_{i=-3}^{5-1} b_i 2^i = 18.75$. Another way to obtain the same result is by interpreting the binary representation as an integer in base 10, $(10010110)_2 = (150)_{10}$, and then multiplying it by the weight of the LSB, which is in this case 2^{-3}. As expected, the result is the same $150 * 2^{-3} = 18.75$.

When operating with two values that use different FP representations, the user needs to keep in mind the representations in order to interpret correctly the result. For example, let a have FP $(i_a|f_a)$ and b have FP $(i_b|f_b)$. When adding these two values, the one with the smallest fractional part needs to be adapted to the other. Assuming $f_a > f_b$, then the addition is obtained as $a + b * 2^{f_a - f_b}$. In this case the FP representation of the result is given by a because of the bigger fractional part.

The multiplication is instead easier. Let c be the result of $a * b$, then its FP representation is $(i_a + i_b|f_a + f_b)$. In this case the multiplication can be immediately carried out in integer arithmetic and later the result can be interpreted as a real value by dividing it by $2^{f_a + f_b}$.

This shows that, except for a shifting operation in the addition, the two operations are the same both in integer arithmetic and in fixed-point. Therefore, the hardware to compute the result can be the same. Similar reasoning applies for division and square root.

3.2.2 Fixed-point optimisation

There are three aspects in digital design that depends on the word length. Arithmetic units need to scale with the size of the operands. For example, the integer multiplication of the Altera IP for integer arithmetic requires 82 LUTs for 8 bit signed operands. The number becomes 294 and 1106 for 16 and 32 bit, respectively. This means that more interconnections inside the

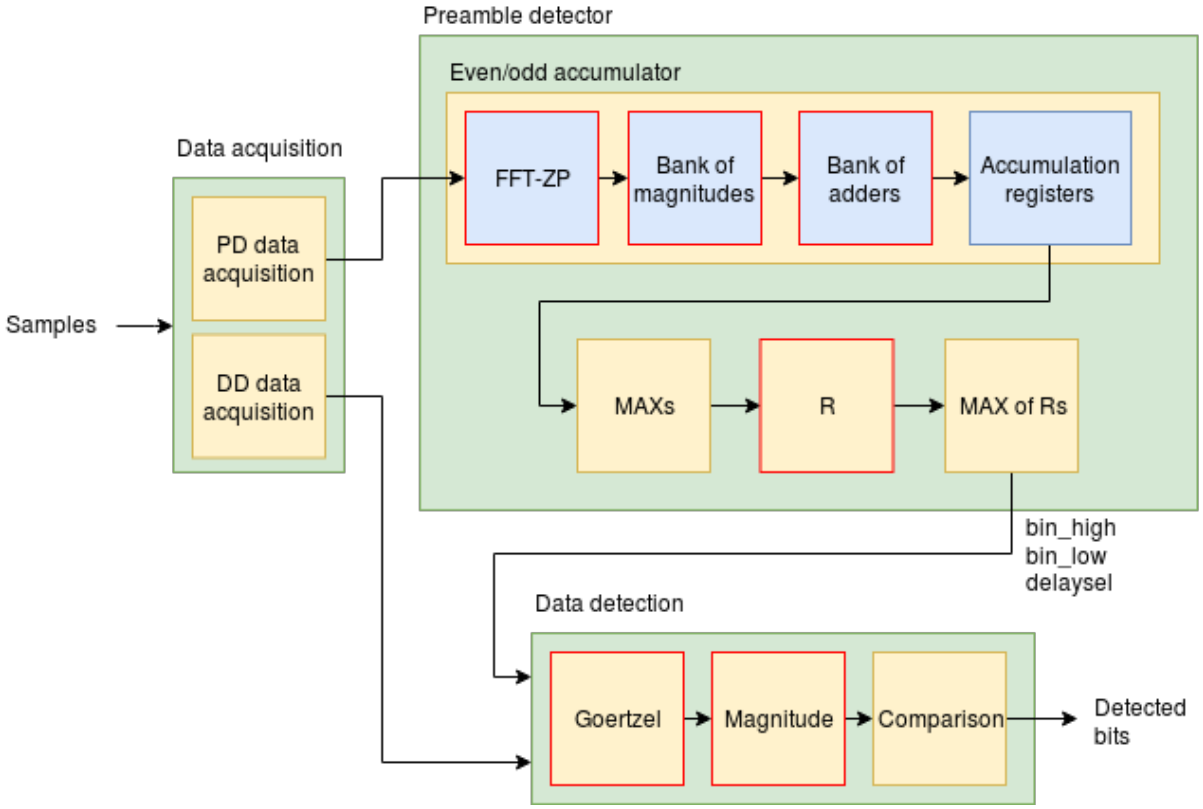

Figure 3.1: Block diagram of the demodulator.

processing unit need to be driven in order to obtain the result, which in turn leads to higher power consumption. Interconnections are not only present inside an arithmetic unit, but they serve to move data between different blocks. Each wire acts as a capacitor which drains energy when charged and discharged. As such, the total energy consumption is proportional to the number of interconnections, which is dependent on the data word length.

Registers are needed where intermediate results need to be stored. These elements retain information across subsequent clock cycles. It is in fact the system clock that dictates the frequency at which the content can change. Whether the content of a register changes or not, they still consume energy. Therefore, the higher the number of registers in use, the higher the energy consumption. However, the size of a register is strictly related to the data representation, hence another reason for keeping the word length as small as possible.

The third aspect is the precision of the computation. In the present algorithm there are many values that depend on the sine and cosine functions. The more precise the results need to be, the more the number of digits have to be stored. This translates to larger word lengths, with all the consequences described earlier. Therefore, it is important to maintain the precision just high enough to obtain correct results.

In conclusion, motivated by these three reasons, the next section will present the approach adopted to determine the FP representation. The procedure will strive to obtain word length as small as possible, while ensuring correct results.

3.2.3 Determination of the optimal fixed-point representation

The mathematical model of the demodulator was available as a MATLAB script. It randomly generates a data packet, simulates its transmission through an AWGN channel, then the demodulation algorithm takes place. Finally, a BER curve is produced by comparing the

transmitted bits with those actually decoded. The system was tested for 14 different levels of noise, $E_b/N_0 = 1, 2, ..14dB$, with *infinite* data precision, that is by using the MATLAB default data type which is 64 bit floating-point. The resulting curve was then used as the performance measure for fixed-point optimisation, which will be referred from now on as the reference curve.

Optimal fixed-point representation of a certain entity of the system means the $(i|f)$ combination that is as small as possible, while the results are still correct. To determine whether the results were correct or not, the BER curve generated by each FP representation was compared against the reference.

The entities for which the FP representation was determined, are shown in Fig. 3.1 with red borders (also for the ADC, which is not depicted). The determination was carried out as follows. The input and output values of the studied block were recorded while running the MATLAB script. The biggest absolute values were used to have a first, rough estimation of the required word lengths. Next, a formal reasoning was adopted to determine the correct representation necessary to avoid overflow. For example, squaring an n bit value requires 2n bit for the output; adding two n bit values require n+1 bit for the output, and so on. Gathering these information, input, output, and formal values, was vital because the number of suitable FP representations is infinite. Instead, with this approach a general idea of where to start searching for the best $(i|f)$ combination was obtained.

As the MATLAB data are in floating-point, a conversion function to fixed-point was needed. The conversion function is as follows.

```
1                value = sign(value)*mod(floor(abs(value)*2^f), 2^w)/2^f;
2
```

where f is the number of bits for the fractional part, w the word length, and *value* is the data that is being converted. This conversion function can be applied after every basic operation (addition, multiplication, square root). For example, if the following equation needs to be evaluated $f(x) = 2x^2 - 3x + 7$, the FP conversion function can be placed at different stages of the computation: on the input, on the output, after computing $2x^2$, or after $2x$ and then again after $-3x + 7$, after each multiplication, or any other combination. The trade off is between a very close representation, when a lot of conversions are executed, and a fast execution time of the script, when a few conversions are used.

The determination of the FP representation was done for each block separately. Initially, all blocks were set to use infinite precision. When a representation was obtained for a block, the following one were simulated while maintaining the FP of the previous equal to the one just determined, while the subsequent were still left with infinite precision. To clarify, an example is now presented. With reference to Fig. 3.1, let us take into consideration only FFT-ZP, bank of magnitudes (BOM), and bank of adders (BOA). Initially all three are given FP $+\infty$. Then the FFT-ZP FP is determined. At this point, BOM and BOA still have $+\infty$, while FFT-ZP has FP $(a|b)$. In the next step, the FP for BOM is determined while maintaining $(a|b)$ for FFT-ZP and $+\infty$ for BOA. After the determination, BOM has a defined FP $(c|d)$ and the procedure can finally be repeated once more for the BOA block.

This sequential determination was done for the synchronisation and detection block separately. All the FP were then put all together in (Section 3.2.12). In this final step, only some adjustments on the various FP representations were needed in order to obtain an acceptable BER.

In the next section, the ADC will be modelled. Following, for each block of the demodulator, the use of the conversion function will be explained and the determination of the smallest FP representation will be presented.

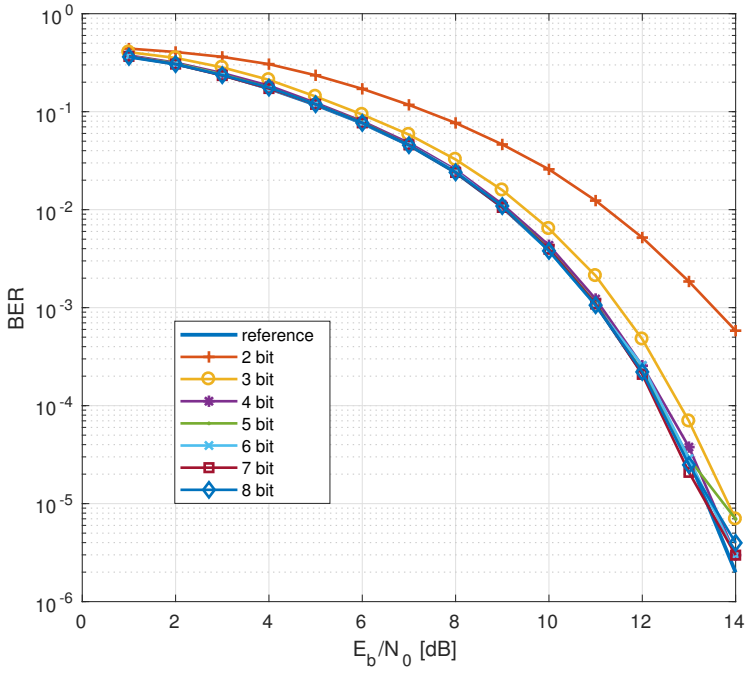

Figure 3.2: Comparison of different ADC resolutions.

3.2.4 Modelling the ADC

The precision of the digital input to the demodulator is determined by the ADC. It was necessary to model it in order to determine the minimum number of bits necessary to represent the input values, while maintaining the BER as close as possible to the reference. As the input was complex, it was assumed that two ADCs were present, for the real and imaginary parts. Similarly to fixed-point optimisation, the model of the ADC was tested in the MATLAB script.

The ADC used uniform quantisation with mid-riser characteristic. The values exceeding the threshold were clipped. The threshold, or saturation level, varied dynamically and was set equal to the square of the RMS of the incoming signal.

The results are shown in Fig. 3.2. For bit resolutions between 2 and 4, the distance from the reference curve is noticeable. Instead, for 5 and 6 the difference is relevant only by zooming in at the highest E_b/N_0 levels, nonetheless it is still relevant. Finally, the curves 7 and 8 bit are both very close to the reference. The latter was chosen as a conservative decision.

This part of the design was carried out for the sake of completeness. Additional research should be performed in order to determine whether a lower amount of bit could have been used.

3.2.5 FP data representation - ADC

Data produced by the ADC are integer values. Generally speaking, all the data processed by the hardware are integers. This is the case because, as it was explained earlier, fixed-point and integer values are two faces of the same coin. Their difference is a scaling factor used to interpret them.

The reason for viewing them as fixed-point, rather than as integer, is due to the fact that the MATLAB script produced results in floating-point. Therefore, when comparing the results from the hardware against those from the script, they had to be converted to integer or real values and in either case a conversion was needed.

In this section we will start by interpreting the output of the ADC and trying to find the

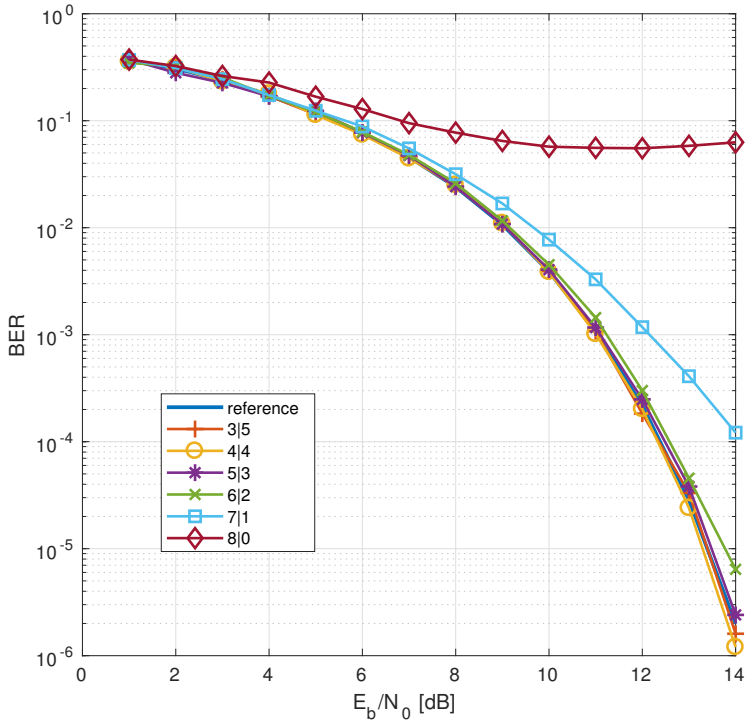

Figure 3.3: Comparison of BER results while varying word length and number of bits dedicated to the integer part for the ADC block.

smallest FP representation that also maintains an appropriate BER. According to simulations the maximum value of the ADC output was 9.0724 and therefore 4 bits were expected to be enough for the integer part. In Fig. 3.3 simulations with different representations are presented. Clearly, not dedicating enough bits for the fractional part leads to a loss of data, as curves (8|0), (7|1), and (6|2) show. Curves (5|3) and (3|5) slightly move away from the reference for high E_b/N_0. Therefore, representation (4|4) was chosen.

3.2.6 FP data representation - FFT-ZP

For the FFT-ZP block, the FP conversion function was placed in multiple positions as indicated with red blocks in Fig. 3.5. It was invoked on the input, for each twiddle factor and after each addition and multiplication operation; however, it was not placed after intermediate operations that arose with complex multiplications.

The maximum absolute value of all real and imaginary parts of the input was 7.25, and 26.95 for the output. Hence, at least 5 bits were necessary for the integer part. An upper bound on the number of bits for the integer part can be obtained by considering the worst case in which the inputs of the FFT-ZP all have the largest possible value, both for the real and imaginary parts. In this case, the output of the FFT-ZP is 73.61, which requires 7 bits.

In Fig. 3.4, the results for different FP combinations are shown. Curve (5|7) matches the reference curve completely. For (4|4), (5|5), and (6|6), the BER is noticeably higher than the reference. Representation (3|8) slightly deviates from reference curve for E_b/N_0 more than 7 dB. Finally, curves (4|8) and (5|7) are fairly close to the reference, except the first that is marginally higher at E_b/N_0 larger than 13 and 14 dB. Therefore, the combination (5|7) is chosen for the FFT-ZP block.

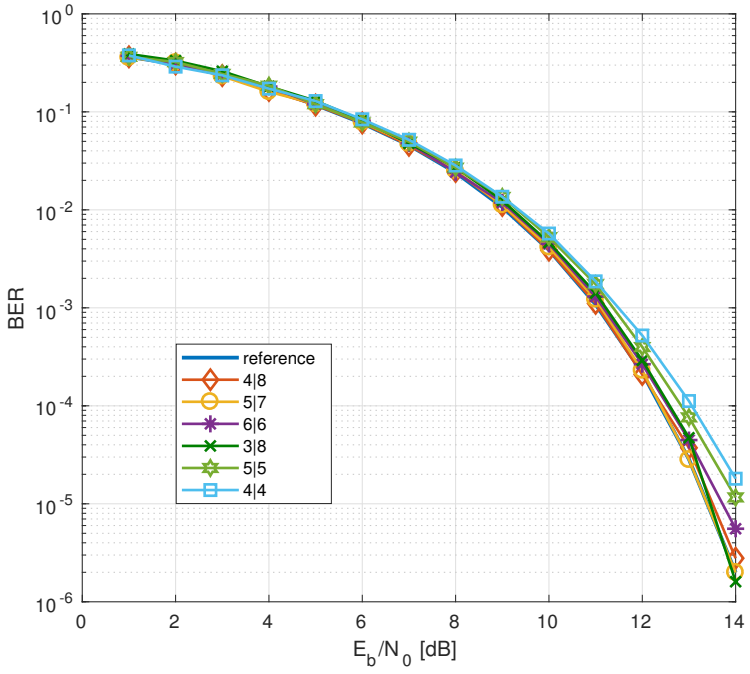

Figure 3.4: Comparison of BER results while varying word length and number of bits dedicated to the integer part for the FFT-ZP block.

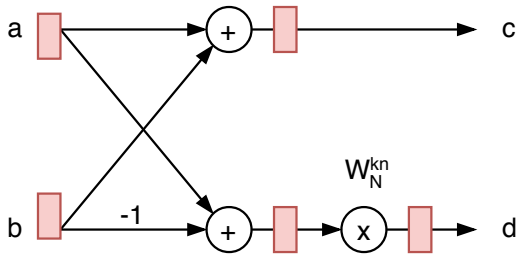

Figure 3.5: Butterfly used in the FFT-ZP. The red blocks indicate the position of the conversion function.

3.2.7 FP data representation - Magnitude (synchronisation)

The magnitude operation is composed by three steps; square of the operands, addition, and square root. If the operands are n bit, after the squaring, 2n bits are needed, 2n+1 after the addition and $ceil((2n+1)/2)$ after the square root.

The maximum absolute value recorded at the input and output were 26.95 and 26.07 respectively. Therefore, 5 bits for the integer part should suffice. However, as it was explained before, intermediate values may require up to $2*5+1=11$ bit dedicated to the integer part to avoid overflow. Therefore, it was expected that the final result may be contained in a register of size between 5 and 11.

When simulating in MATLAB, the magnitude of a complex value can be simply obtained by invoking the abs() function. The problem is that the intermediate operations are hidden, which makes the intermediate values impossible to approximate. Therefore, the built in function was not used and a dedicated one was made in which the squaring, addition, and square root operations were all separate. The FP conversion was then placed between each operation as well as on the input and output of the magnitude block.

In Fig. 3.6, the results are shown. Curve (10|0) is not suitable, which also suggests that the fractional part should be maintained. Curve (6|3) indicates that using less than 7 bit for the integer part is not correct. Finally, (8|3) and (7|2) are very close, and therefore the second is chosen for the reduced number of bits.

3.2.8 FP data representation - Accumulation

This block corresponds to Bank of adders and Accumulation registers, with reference to Fig. 3.1. The accumulation phase sums incoming data from the magnitude for seven times. This means that to avoid overflow three additional bits are required. The maximum absolute value on the input is 26.07 which requires 5 bits for the integer part. According to simulations, the maximum output value is actually 96.238, which suggests that 7 bit should suffice for the integer part. The FP conversion function was executed after each addition and on the incoming data.

In Fig. 3.7, simulation results are shown. Curve (5|7) shows that 5 bit for the integer part is not enough. Instead, no bit for the fractional part is also not a good decision, as shown by curve (7|0). Finally, the remaining curves are all very close to the expectation. Therefore, curve (6|2) was chosen.

3.2.9 FP data representation - Selection

This block corresponds to R, with reference to Fig. 3.1. In this block the delaysel and the two bins, bin_low and bin_high, are determined. The procedure is as follows. Let x and y be two arrays. The maximum value, and its position in the array, is searched for both. Let x_{max} and y_{max} be these maximum values and x_{pos} and y_{pos} be their respective index position in their corresponding arrays. Then, the value R_i for delay i calculated in this block is obtained as $R_i = x_{i,max} - x_i(y_{i,pos}) + y_{i,max} - y_i(x_{i,pos})$. The maximum between the R_i dictates the delay, i.e. $delaysel = i$.

Regarding data representation, if both maximum values, x_{max} and y_{max}, require n bits to be represented, then the final value R needs n+1 bits. This can be understood by noting that $x(y_{pos})$ and $y(x_{pos})$ are values smaller than x_{max} and y_{max} respectively. Therefore, the differences are always representable with n bit. Finally, only the addition of the differences may produce overflow and therefore n+1 bit are required.

The FP conversion was applied on the input, after the intermediate subtractions, and on the final R value. The maximum input and output were 96.238 and 102.56 respectively, for which 7 bits should suffice. In the worst case, as previously explained, 8 bits might be required.

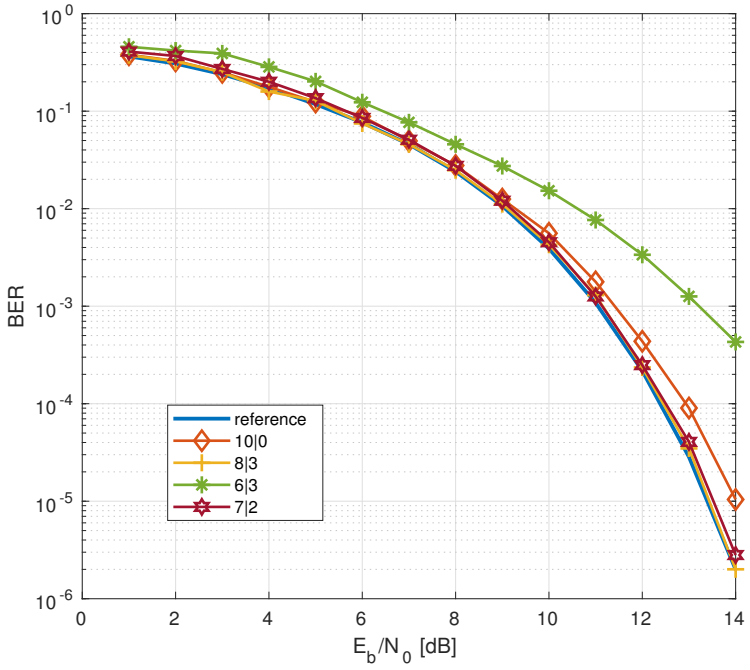

Figure 3.6: Comparison of BER results while varying word length and number of bits dedicated to the integer part for the magnitude (synchronisation) block.

In Fig. 3.8, the results are shown. Curves with integer bit 4, 5 and 6 are not suitable. Curve (5|0) performs well on high E_b/N_0, but bad on the low side, while curve (6|0) behaves in the opposite way. Nonetheless, both were discarded. Instead, curves with 7 bit for the integer part perform equally well suggesting that bit for the fractional part does not influence. Therefore, curve (7|0) was chosen.

3.2.10 FP data representation - Goertzel algorithm

In the Goertzel algorithm there are two additions and one multiplication that are repeated 8 times. Each of these operations require an additional bit. After this loop stage, three more additions are needed. In total there are $3 * 8 + 3 = 27$ operations in the algorithm, leading us to add $ceil(\log_2 27) = 5$ bits.

The maximum values recorded and the input and output of the Goertzel block were 8.9047 and 23.998 respectively. Therefore, 9 bits for the integer part should suffice when also considering the extra 5 bits previously indicated. The FP conversion function was invoked on all the twiddle factors, on the input, and after each arithmetic operation.

The results are shown in Fig. 3.9. The (7|0) curve suggests that leaving out the fractional part drastically hinders the correct detection of the data packet. Also 3 bit for the fractional part are not enough according to curves (7|3) and (5|3). Finally, (7|5) is picked as a conservative decision.

3.2.11 FP data representation - Magnitude (detection)

The discussion regarding the magnitude data representation presented for the one in the synchronisation block Section 3.2.7 applies for this case as well. The maximum values that were recorded were 23.998 and 25.572 for the input and output respectively. Therefore, considering intermediate operations it is expected that 11 bits will be enough for the integer part.

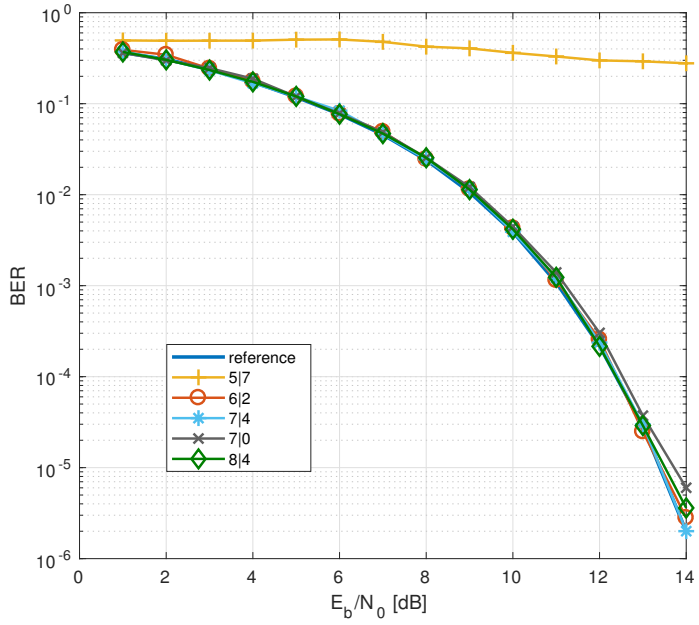

Figure 3.7: Comparison of BER results while varying word length and number of bits dedicated to the integer part for the accumulation block.

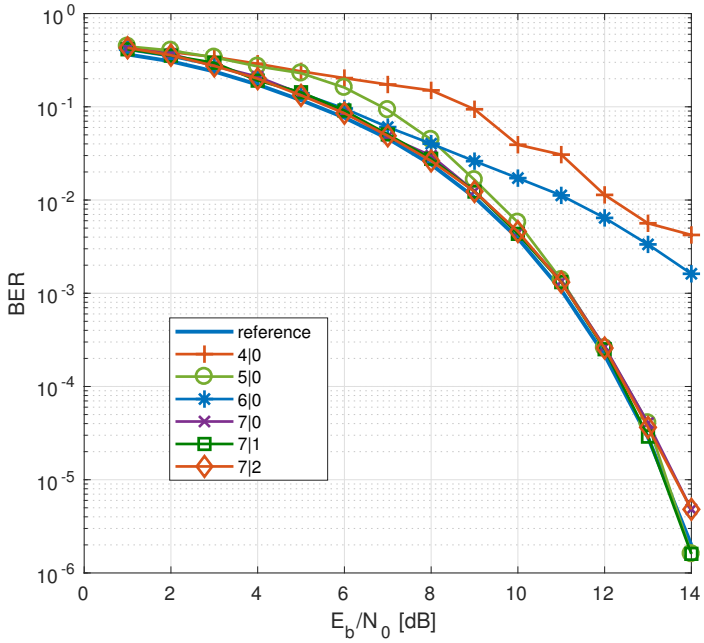

Figure 3.8: Comparison of BER results while varying word length and number of bits dedicated to the integer part for the selection block.

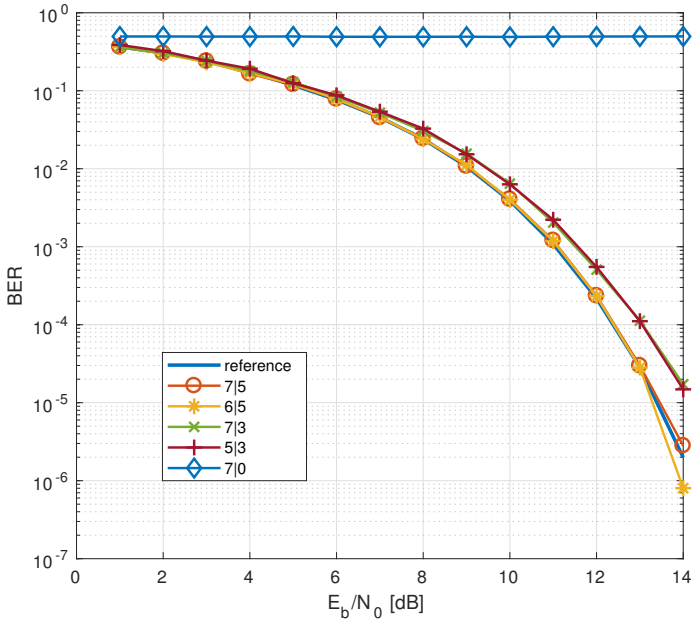

Figure 3.9: Comparison of BER results while varying word length and number of bits dedicated to the integer part for the Goertzel block.

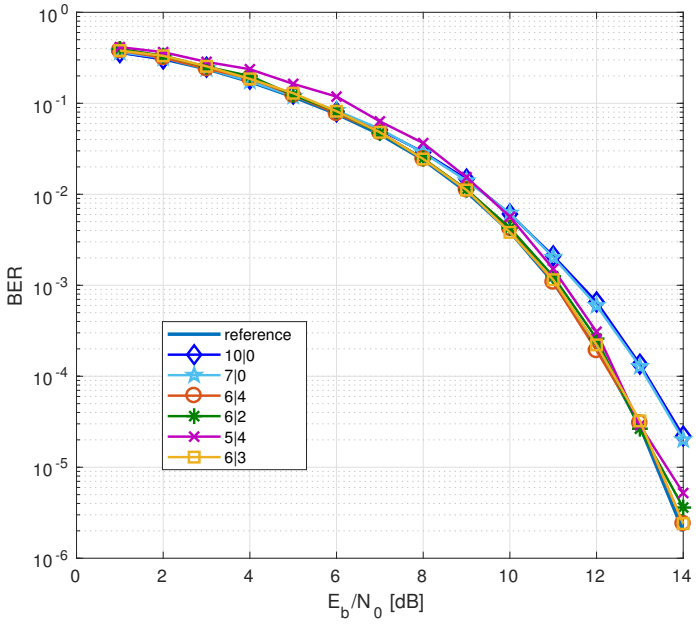

Figure 3.10: Comparison of BER results while varying word length and number of bits dedicated to the integer part for the magnitude (detection) block.

Results are presented in fig. 3.10. Removing the fractional part is not appropriate as shown by curves (7|0) and (10|0). FP (5|4) shows that overall the BER increases for all E_b/N_0 suggesting that 5 bit for the integer part are not enough. In contrast with the expectation, 6 bit seems to be enough. Combination (6|2) is therefore picked as it has the smallest number of bits in use.

3.2.12 FP data representation - Putting it all together

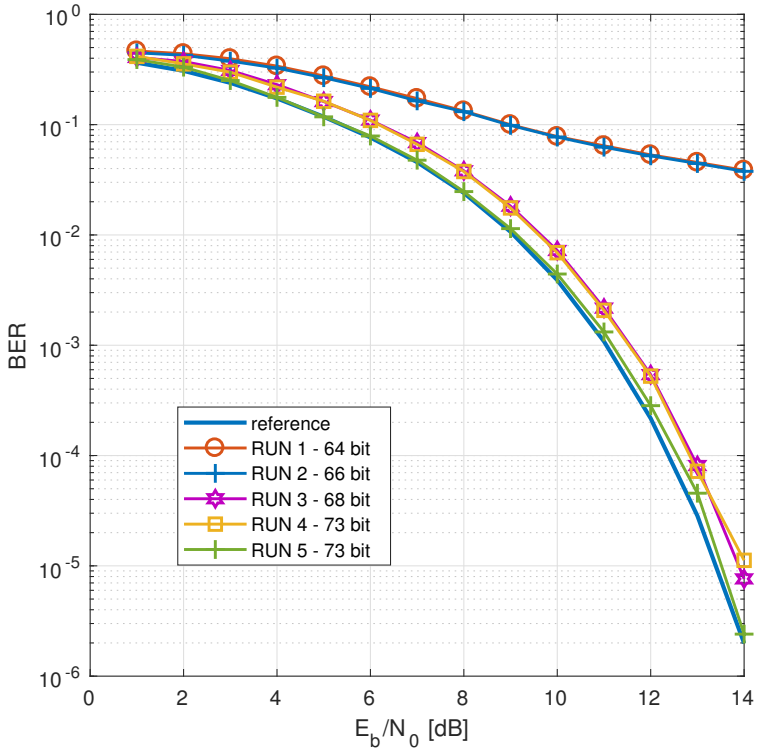

Figure 3.11: Comparison of BER results while varying FP representation for different blocks.

	FP1	FFT-ZP	Magn. (sync)	Accumulation	Selection	Goertzel	Magn. (det.)
RUN 1	(4|4)	(5|7)	(7|2)	(6|2)	(7|0)	(7|5)	(6|2)
RUN 2	(4|4)	(5|7)	(**8**|**3**)	(6|2)	(7|0)	(7|5)	(6|2)
RUN 3	(4|4)	(5|7)	(**9**|**3**)	(**7**|**2**)	(7|0)	(7|5)	(6|2)
RUN 4	(4|4)	(5|7)	(9|3)	(**8**|**3**)	(**8**|**1**)	(7|5)	(6|2)
RUN 5	(4|4)	(5|7)	(9|3)	(8|3)	(8|1)	(7|5)	(**7**|**2**)

Table 3.4: FP combination for the whole system used in Fig. 3.11.

To choose a proper FP representation for all blocks, the FP representations for the synchronisation and detection blocks were simulated together. The results are shown in Fig. 3.11. The FP combinations that were tested are listed in Table 3.4, bold text indicates which block FP representations were changed from the previous run.

Both the FFT-ZP and the Goertzel blocks never change. That is because their implementation did not allow for an easy modification. Therefore, when their FP representations were

determined, a conservative decision was made and kept unchanged. Instead, the ADC FP representation was deemed to be appropriate because of its simplicity.

The final search for the correct combination was rather difficult. This is because it was not easy to establish which block's FP representation needed to be changed. Therefore, a trial and error approach was adopted in which a few bits were changed between each run. Moreover, each simulation took a considerable amount of time, about an hour. Because of that, an exhaustive search was not possible.

RUN 1 corresponds to the combinations that were picked for each block. Its curve shows that it is not suitable, as well as for RUN 2. RUN 3 improves the previous two cases, but it is still noticeably far from the reference. RUN 4 and RUN 5 show appropriate results. Between the two, RUN 4 was finally adopted because it uses 1 bit less in the magnitude of the detection phase.

3.2.13 How many bits were saved?

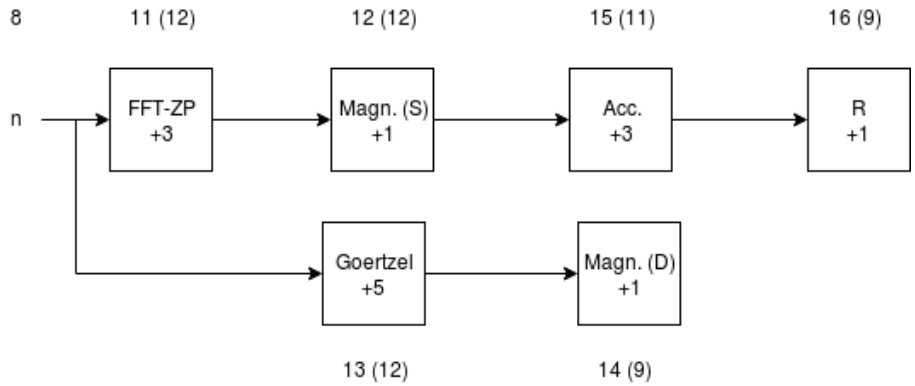

Figure 3.12: Comparison of the required number of bits and the actual amount that was used. The values outside the squares indicate the expected amount (out of parenthesis), values out of the squares, but in parenthesis indicate the amount obtained using the FP optimisation. The values inside the squares indicate the additional amount of bit that each block introduces.

This section will discuss the amount of bits that were saved by using the FP optimisation method. The comparison will be done against a naive approach. With this method it is only of interest to avoid overflow. The values will be obtained analytically, but will only account for the integer part, while the FP representations also include the fractional one.

Given to an N-point FFT block n bits input, in the worst case $\log_2 N$ bits need to be added on the output to avoid overflow, one bit for each stage. That is because of the additions that require and additional bit, while multiplications do not influence as the twiddle factor values are always between -1 and 1. For the FFT-ZP the number of addition stages are only three. Therefore, at the output of this block n+3 bit are needed.

The magnitude operation that follows is comprised by three internal operation. The squaring of the operands, which requires 2n bits, addition, which adds one bit, and square root, which reduces the number of bits to half (rounded up). Therefore, given n bits input, the output requires $ceil((2n+1)\frac{1}{2})$ bits, or in other words one additional bit compared to the input.

Following the magnitude, there is an accumulation phase, which is repeated 7 times. Therefore, $ceil(\log_2 7) = 3$ additional bits are introduced by this block. Finally, as explained in Section 3.2.9, an additional bit is required to avoid overflow in the R block.

For the detection part, the determination of the exact number of bits, when the input is n bit, is as follows. The number of bits added by the Goertzel algorithm is 5, which was explained

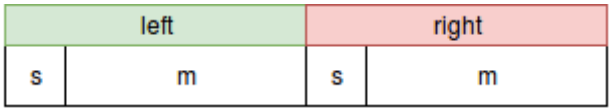

Figure 3.13: Schema of the representation of complex data in hardware.

in Section 3.2.10. Following this block, a magnitude is present which requires an additional bit for the same reason as indicated above.

In Fig. 3.12, a schema that recaps the expected amount of bits and the actual amount that were used is portrayed. The expected amount, i.e. values obtained analytically, are out of the squares and outside parenthesis. Instead, the number inside parenthesis indicates the word length obtained with the FP optimisation. The values inside the squares, preceded by a plus sign, indicate how many bits that block introduces. It can be seen that using the FP optimisation only the FFT-ZP waste 1 bit, while the following magnitude equals the expectation. Instead, all the other blocks use a reduced word length, which indicates that bits were saved compared to the naive approach.

3.3 System architecture

In this section, the architecture of the system will be presented. Firstly, the way complex data were encoded will be shown. Afterwards, the organisation of the blocks that compose the system will be presented. For each one, the state machine (if present and non trivial) and the block interface will be shown. In the interface, the term "slv x" indicates a std_logic_vector of x bits. With "complex12" it is indicated a complex data type of which its real and imaginary parts are 12 bits each, where for the FFT-ZP the FP is (5|7) while for Goertzel (7|5). Signals that expose just the name, e.g. "done" or "start", are intended as single bit wires. For all the state machines, state S_IDLE or IDLE indicates the initial state at reset time.

3.3.1 Complex data implementation

A complex value is made of four separate parts. The magnitude of the real and imaginary parts and their signs. The magnitude and sign can be represented together, by using two's complement, or separately, with sign magnitude. In this design, the left part contains the real part of the complex number, while the imaginary one is stored in the right side, as depicted in Fig. 3.13. From the scheme it is also clear that a signed magnitude representation was used. The reasoning behind this choice is as follows. In the FFT-ZP block, there are numerous multiplications by -1, j, and -j. Because signed magnitude is used, sign inversion is a power efficient operation as it is equivalent to inverting one single bit. Instead, if two's complement was used, the sign inversion would be obtained by inverting each bit and by adding one to the result, which is a more complex operation [1]. The downside of using signed magnitude is the additional logic required to compute the addition operation. In Section 4.3.1, it will be shown that from an energy point of view, the trade off between an easier multiplication by -1 surpasses the downside of a more complex signed addition.

3.3.2 Data acquisition

The Finite State Machine of this module and its interface are presented in Fig. 3.14. This module collects samples incoming from the ADC and passes them first to the synchronisation

[1] The idea of using the signed magnitude inside the FFT, has been firstly proposed by Oguz Meteer, who is a Ph.D. student at the same research group of the author.

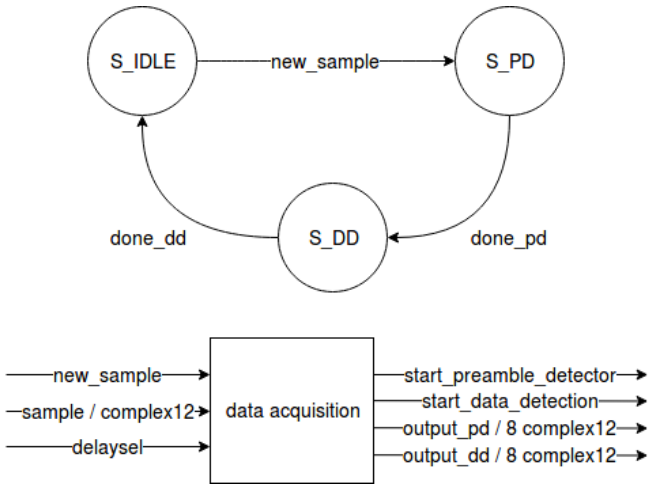

Figure 3.14: Finite state machine and interface of the Data acquisition module.

and then, to the Data detection. Its controller is straight forward. Initially, the Preamble detector (which corresponds to the window synchronisation phase) is enabled. As soon as the *delaysel* and bin_high and bin_low have been identified, the data detection block is enabled.

This module receives the *delaysel* signal generated by the preamble detector. It indicates how many samples were acquired, which do not belong to the preamble. This information is then passed to the Data detection - data acquisition block. It discards a certain amount of samples that lies between the preamble and the data packet, in order to balance those wrongfully acquired.

Preamble detector - data acquisition

The role of this block is to store eight samples at a time and pass them to the Preamble detector module. This operation is repeated 112 times, that is as many samples that constitute the preamble (eight for each of the fourteen symbols).

This module was implemented as a circular buffer of size 8. In fact, each window differs from the previous by one sample. Therefore, the oldest sample of the previous window is replaced with the newest. Two indices, head and tail, are maintained so that the samples can be passed in the correct order.

Data detection - data acquisition

Similarly to the previous module, this block also has to pass eight samples at a time to the Data detection unit. Its FSM and interface are depicted in Fig. 3.15. When the first sample arrives, the *delayselCounter* loads the value from the input port *delaysel*. As long as its value is different from zero, it will make the controller wait on S_DELAYSEL_WAIT. This way, the samples that were previously acquired, but not actually belong to the preamble, are compensated by discarding "guard samples" that lie between the last symbol of the preamble and the first one of the data packet.

In Fig. 3.16 an example is shown where the preamble is four symbols, each symbol is composed by four samples, one guard symbol is used, and two initial samples are wrongfully acquired. Window 3 is the one that properly overlaps, which indicates a *delaysel* of two samples. This information is only available when the last window, window 4, has been processed. When window 4 has been processed, it can be seen that $delaysel + 1 = 3$ samples need to be discarded before the actual payload is received. Instead, if three samples were wrongfully acquired at the beginning, then *delaysel* would be 3 and the number of samples to be skipped 4.

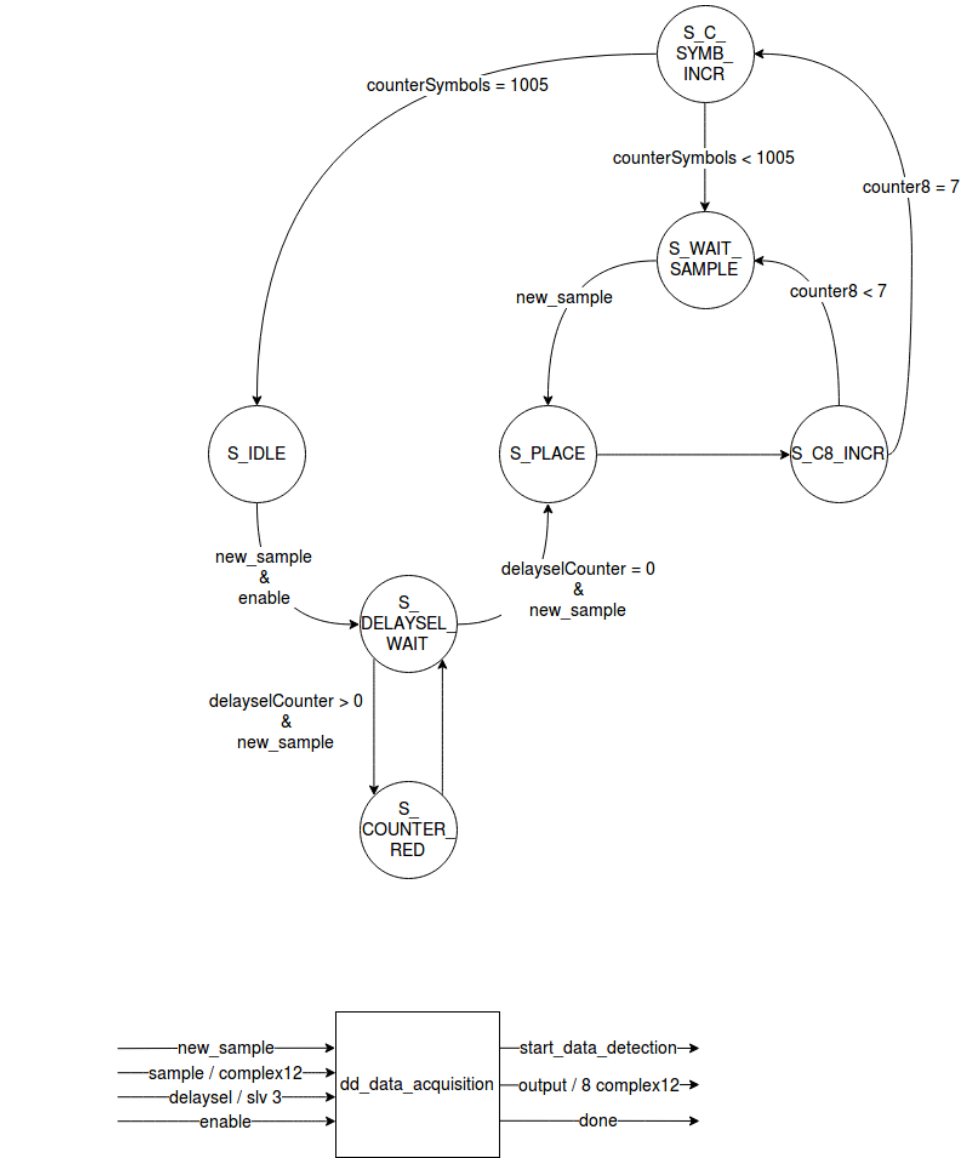

Figure 3.15: Finite state machine and interface of the Data detection - data acquisition module.

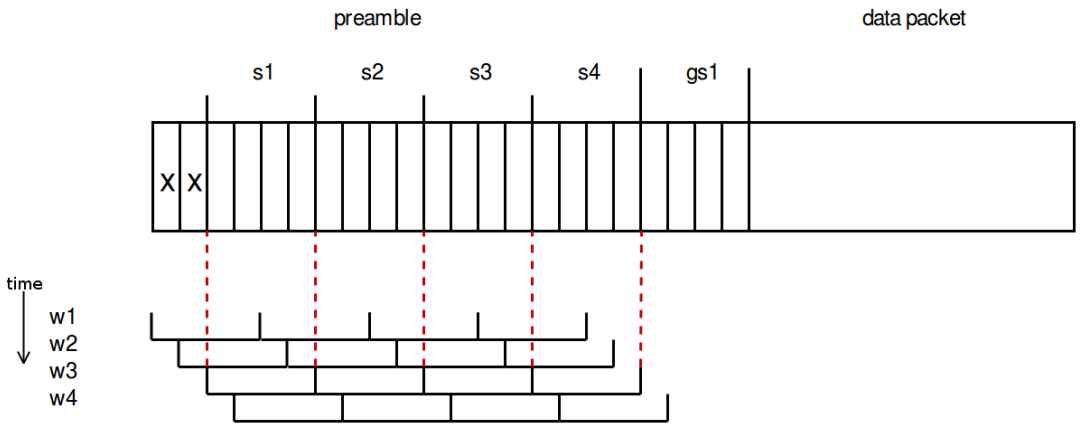

Figure 3.16: Packet composition when two samples are wrongfully acquired.

3.3.3 Preamble Detector

This entity is in charge to properly aligning the window to the samples stream. Its output are the *delaysel* and the indication of the two bins, bin_high and bin_low. As mentioned before, they are used in the detection phase.

There are five steps in the preamble detection: DFT, an accumulation phase (carried out by the even_odd_accumulator module), a maximum search (done by MAXs), a computation of a value R (of which the unit R takes care of), and finally finding the maximum of the R values which determines both the delay and the bins_ high and bin_low (the module responsible is Max of Rs or MOR).

Even/odd accumulator

The diagram of this unit is presented in Fig. 3.17. This unit is composed of four sub units: the FFT-ZP module (presented in 3.3.5), a bank of magnitude, a bank of adders, and a bank of registers. The initial eight samples are passed to the FFT-ZP, which produces a 64-point DFT. The results, which are complex values, are passed to a bank of magnitudes. It is a collection of 64 units that compute the magnitude of a complex number. Subsequently, the 64 magnitudes are accumulated in a two-dimensional array. The indices of this array are given by the parity of the current symbol number (even or odd), and by the current window delay, which spans between 0 and 7. Each register (eight for even and eight for odd symbols) accumulates values incoming from the bank of magnitudes for seven times, that is half of the number of symbols in the preamble (seven odd symbols and seven even symbols). When all the accumulations are done, these 16 registers are passed to the following unit, MAXs.

MAXs

There are eight instances of this unit, one for each delay. The input of each one, is a row of the array shown in Fig. 3.17, that is two arrays of 64 values each, one called "odd", the other "even". They are passed to the respective sub units present in each MAXs, called max_finder64, as shown in Fig. 3.18. The output of the max_finder64 are the maximum value of each array and their indices in their respective array. These indices are then passed to the other unit, which outputs the element corresponding to these indices in the other array. Finally, four values are the output of each MAXs unit.

R and Max of Rs

There are eight R modules, one for each delay. Each module receives the four values produced by the corresponding MAXs and produces the result as follows. With reference to Fig. 3.18, the result R is given by $(max\_odd - val\_index\_even) + (max\_even - val\_index\_odd)$.

The eight R values produced in the previous blocks are the input of the module Max of Rs (MOR). This unit simply indicates which of these eight values is the biggest. Because each R values correspond to a specific delay, the maximum among them indicates the *delaysel* signal, previously mentioned in Section 3.3.2. Additionally, the output of the corresponding MAXs indicates also the indices of the max_even and max_odd. These values correspond to bin_low and bin_high. The smallest of the two becomes bin_low, while the other bin_high.

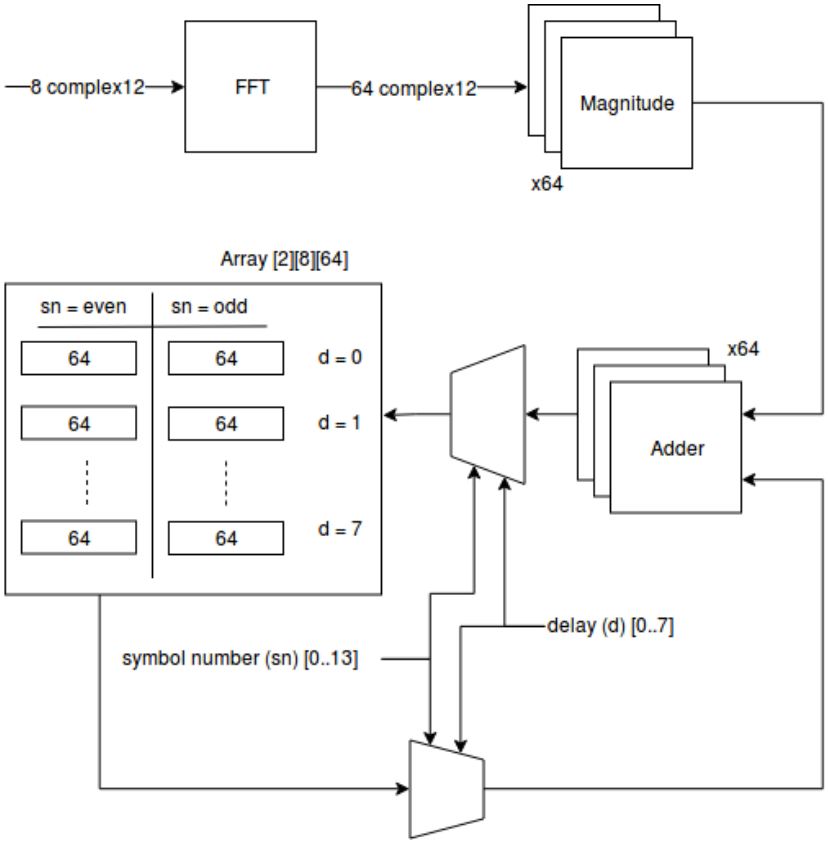

Figure 3.17: Schema of Even odd accumulator module.

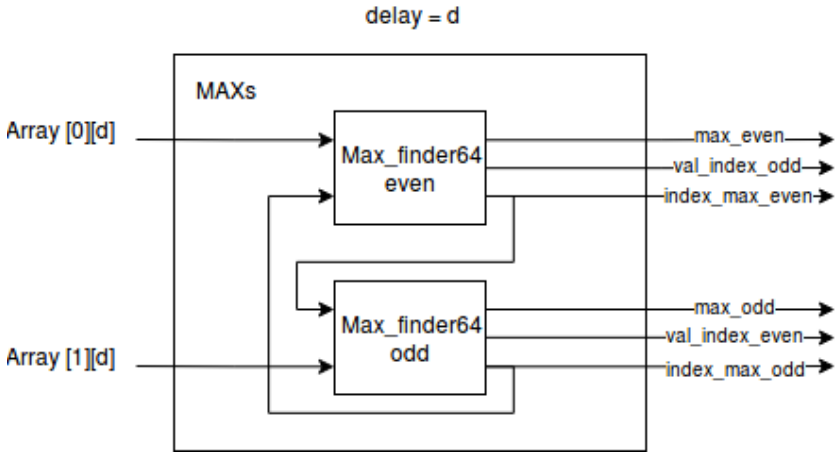

Figure 3.18: Schema of MAXs module.

3.3.4 Data detection

When the preamble has been detected and the delay determined, together with bin_low and bin_high, the data detection phase starts. There are three steps in order to detect the received symbol. Firstly, the Goertzel algorithm is used to determine the complex values of bin_low and bin_high. Two Goertzel blocks are in use, so that the computation of the two bins is done in parallel. Secondly, their magnitudes are determined. Follows a comparison of these two magnitudes, which determines whether the received bit is 0 or 1.

Goertzel

The input is the index of the bin that needs to be calculated. Moreover, the eight samples needed to compute the bin are passed one by one on the input port xn. The states of the FSM, shown in Fig. 3.19, are reported in the following listing as comments. They indicate which operations are carried out in each phase.

```
% bin is the input for which the algorithm is calculated
N           = 64;
M           = 8;
w0          = (2*pi*bin)/N;

cos_term    = 2*cos(w0);
WkN         = exp(-1i*w0);
factor      = exp(-2i*pi*bin/M);

s0 = 0;
s1 = 0;
s2 = 0;

for n = 1:N
    % SM1
    s0 = x(n) - s2;
    s2 = cos_term*s1;

    % SHIFT
    s0 = s0 + s2;
    s2 = s1;
    s1 = s0;
end

% MUL1
s0 = cos_term*s1;
s1 = s1*WkN;
% SUB1
s2 = s0 - s2;
% SUB2
s2 = s2 - s1;
% MUL2
% s2 = s2*factor;

output = s2;
```

The multiplicands, cos_term, WkN, and factor, are precomputed and stored in a look-up-table, which are indexed by the input. The multiplication by the factor is commented and its corresponding state in the FSM is cancelled. It was explained in Section 2.3.2 that such multiplication is not useful as long as the phase is not needed. In fact, in the present case, only the magnitude is necessary, therefore the operation is avoided.

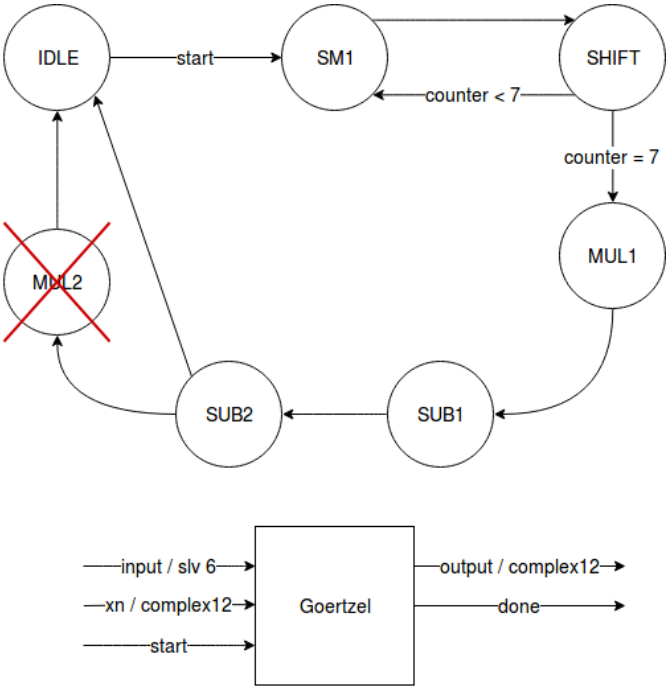

Figure 3.19: Finite state machine and interface of the Goertzel module.

3.3.5 FFT-ZP

The FFT-ZP module implements the algorithm described in Section 2.2.3. It is composed by three transfer stages and three normal stages, as shown in Fig. 3.20. The first stage is composed by transfers and multiplications by twiddle factors. The other three instead, implement the butterfly structures. Multiplication by the trivial factors 1, -1, j, -j, were treated separately.

The twiddle factors are precomputed as they remain always the same throughout different computations. Intermediate registers were used to reduce the occurrence of glitches [36], which are a cause of power waste. The FFT-ZP has been implemented as a fully parallel operation. This way, multiple operations can be carried out simultaneously. The advantage is the possibility to relax the system clock. This can be understood as follows. In Section 4.1 it will be shown that if the algorithm requires a large amount of cycles, then the system clock needs to be faster than the ADC clock by a factor proportional to the number of these cycles. Therefore, it is in the interest of a power-efficient solution to compute in as little number of cycles as possible. If the computation of a single butterfly can be executed in time T_1, then N butterflies need to be computed, using sequential computation, in N cycles of period T_1 each. Instead, if N butterflies are computed in parallel, it is likely that the period required is $T_2 \geq T_1$, because of the additional logic that might extends the longest combinational path. Nonetheless, it is unlikely that $T_2 > NT_1$. Therefore, with the parallel approach, the number of total cycles is less than the number used in a sequential solution. The downside of this choice is the additional resources that are required.

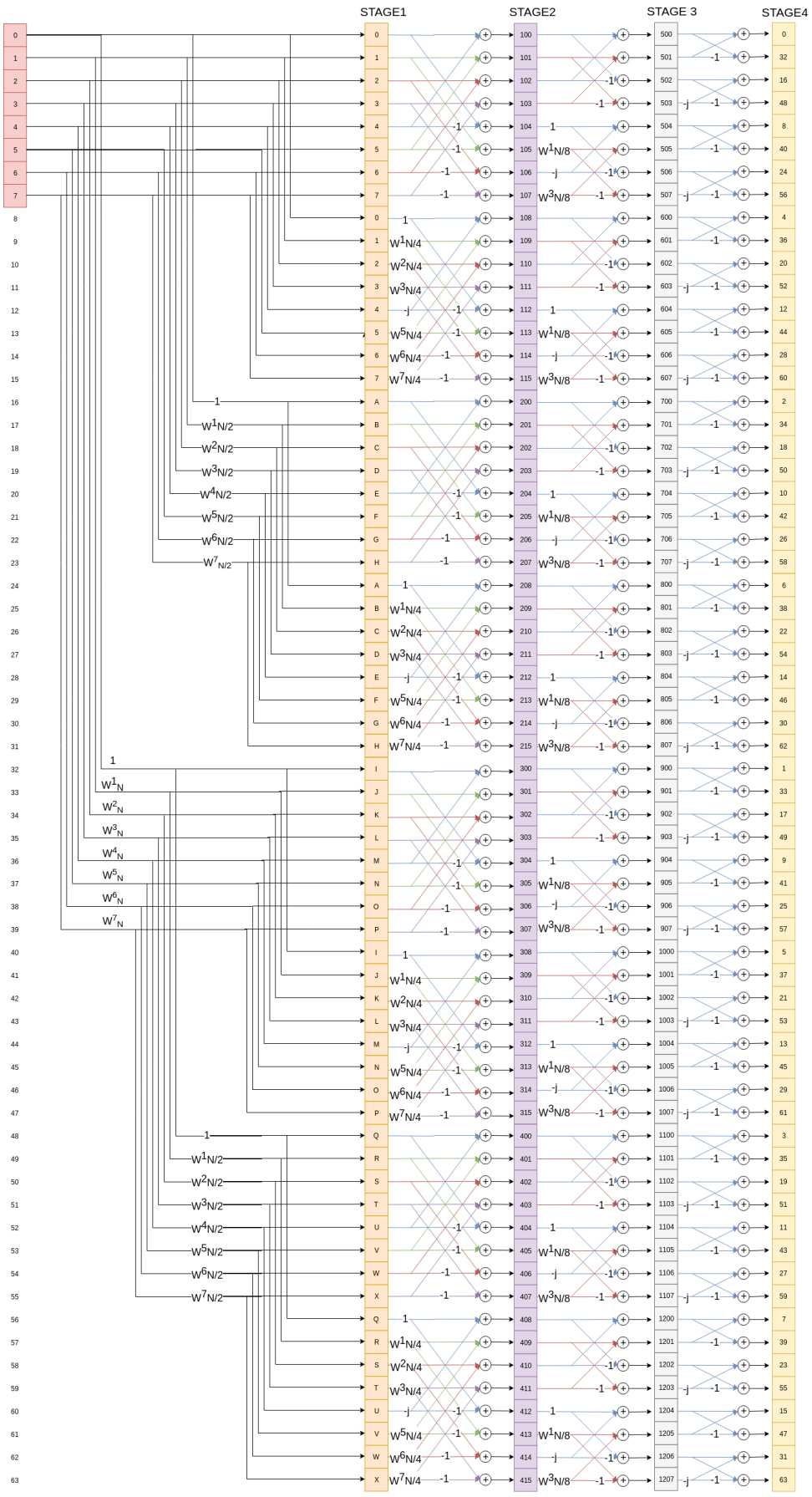

Figure 3.20: FFT-ZP schema.

3.4 Verification and tools

3.4.1 Verification

Two types of tests were used to verify the correctness of the design: functional tests through simple testbenches, and automated tests with input generation/output verification in MATLAB. Testbenches were manually written in order to verify the correctness of each module. The input provided covered a few cases, especially corner cases. Nonetheless, they were by no means intended for an extensive coverage. They were only supposed to provide a regression test to verify that new features did not break the existing code.

A more thorough verification was carried out using MATLAB. Recall that the demodulator was available as a script and each implemented block had a counterpart in that script. The input was generated using it and was then passed to the hardware implementation which was subsequently simulated with QuestaSim. Following, the obtained output were saved and passed back to the script. Finally, the script produced the BER curve. With this approach, had the hardware implementation been done incorrectly, the BER curve would have showed it. This procedure was repeated for the most important blocks, namely FFT-ZP, Goertzel, the complete synchronisation block, the complete data detection one, and finally for the whole preamble.

VUnit framework VUnit is an open source unit testing framework for VHDL and SystemVerilog. It helps to realise continuous and automated testing. In this context, VUnit was used to parallelise the execution of the most time-consuming testbenches. To give an idea, the simulation of the complete system for E_b/N_0 ranging in 1, 2.., 14 dB and for 50 packets, each one composed by 128 symbols, would have taken approximately 311 hours (or 13 days). Instead, by using VUnit, different testbenches can be dispatched to different cores of the CPU, which is a feature that QuestaSim does not offer. The verification of different packets is independent from one another, also with respect to different level of noise. Hence, it was possible to execute each simulation in parallel. With this approach, a test of roughly 800 packets, each composed of 128 symbols, and E_b/N_0 in 1, 2.., 14 dB, less then 24 hours were needed, which is thirteen times faster than the non-VUnit case.

3.4.2 Tools

For this project, VHDL version 2002 was used. The reason for the language choice was related to the author's previous experiences, while the version is due to the fact that few compilers fully support the 2008 standard. The chosen FPGA vendor was Intel Altera because also in this case the author had a preexisting knowledge on how to obtain power simulations.

All the arithmetic operations were implemented by using Altera's IP. Namely, addition, multiplication, and square root were the ones required for the algorithm. Of these operations, multiple instances were created to accommodate the different sizes of the operands. All the IP were generated without allowing the use of DSP. The reason is that the whole project is implemented for an FPGA, but only for prototyping reasons. It may actually be moved in the future to an ASIC for which DSPs are separate resources, in contrast with FPGA that already come equipped with them.

Chapter 4

Simulation and results

In this chapter, various results regarding the design are presented. Firstly, the admissible clock frequencies for the design are shown in Section. 4.1. Afterwards, the BER of the implemented system is reported in Section 4.2. Subsequently, the power results are given in 4.3. In this section, the advantage of using signed magnitude over two's complement is demonstrated, the power consumption of the whole demodulator is presented, and a power comparison of the proposed FFT-ZP, Goertzel algorithm, and a standard FFT is provided. Finally, the chapter concludes with Section 4.4 in which the resource usage of the design are elaborated.

4.1 Clock frequencies

The clock is a critical component of the design. If the frequency is higher or lower than the admissible range, the system does not operate as expected. When it is too high,, the clock signal arrives too quickly at the registers while the actual data that need to be stored may not have reached its input. To avoid this, it is necessary to identify the longest combinational path, also called critical path. The inverse of the time that a signal takes to cover this path indicates the maximum frequency supported by the circuit. This information is provided by the synthesis tool, which is Intel Altera Quartus in this case. The reported maximum frequency F_{max} for the whole demodulator is equal to 5.36 MHz.

Usually, the maximum frequency is the only constraint that the clock needs to meet. In our system though, it is also necessary to determine the minimum frequency F_{min}. In fact, the ADC acquires samples at a certain speed, which are then passed to the demodulator. This means that any possible operation that involves a sample needs to be completed before the next sample is received, otherwise extra registers must be added for data storage. The slowest operation of the algorithm occurs at the end of the synchronisation phase, when the two bins are determined together with the delay. These steps require a total of 35 cycles. Therefore, the relation between the ADC frequency (ADC_f) and the system clock frequency ($CLOCK_f$) is:

$$\frac{CLOCK_f}{ADC_f} \geq 35. \tag{4.1}$$

The system was tested with different ADC frequencies and the power results of each scenario is presented in Section 4.3.3. They are 800 Hz, 8 kHz, and 80 kHz. When considering the number of samples per symbol (M=8) these values correspond to data rates of 100 bps, 1 kbps, and 10 kbps, respectively. For each one, the slowest usable clock is 28 kHz, 280 kHz, and 2.8 MHz. The maximum allowed frequency of the ADC can be derived using Eq. 4.1. Knowing that the design supports a maximum clock of 5.36 MHz, then the highest ADC frequency is 153,142 Hz.

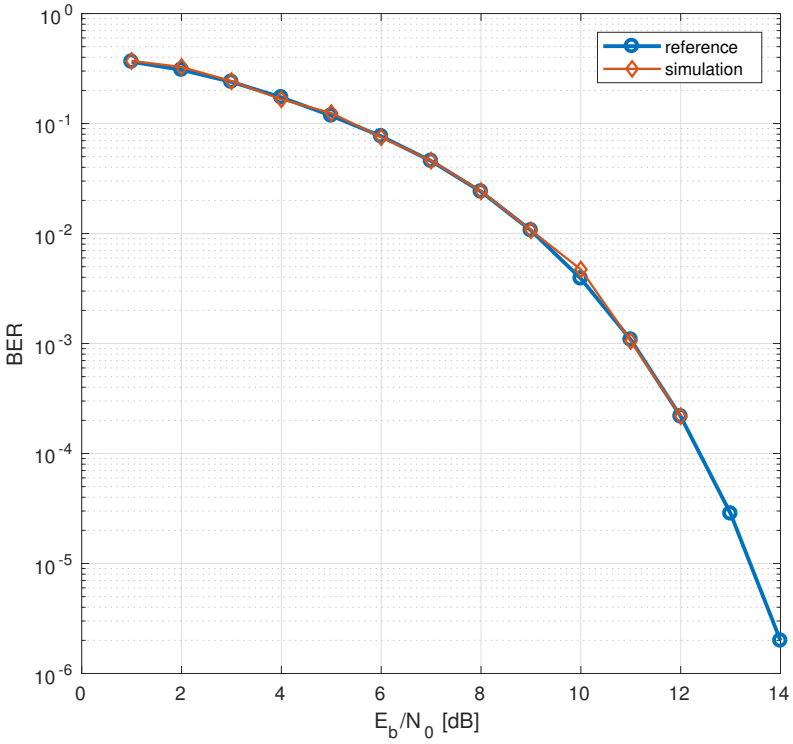

Figure 4.1: BER curve for the FFT-ZP block simulated in QuestaSim.

4.2 BER in hardware

The functional correctness of the system is now presented. The reference BER curve produced by the MATLAB script, described in Section 3.2.3, is compared against the BER curve obtained by simulating the hardware implementation, as it was explained in Section 3.4.

The tests carried out for the FFT-ZP, for the Goertzel algorithm, and for the whole system are here reported. The reason for simulating the FFT-ZP and the Goertzel algorithm is that their complexity did not allow for an easy modification of their designs. In other words, had the system been tested only when the whole system was complete and had it resulted in a non satisfactory BER, changing the FP representation of those two blocks would have proven challenging. Therefore, immediately after their implementation, an extensive amount of tests were carried out. The BER curves for the FFT-ZP and Goertzel algorithm are shown respectively in Fig. 4.1 and Fig. 4.2. Both tests show that the implementation and the choice of the FP were in fact correct. The FFT-ZP was tested for 50 packets, with a signal to noise level E_b/N_0 in range 1, 2,.. 14 dB, and each packet was 1024 symbols (preamble included). Instead, Goertzel was tested for 10 packets to decrease simulation time. Levels of noise and number of symbols were equal to the FFT-ZP test.

In Fig. 4.3, the BER result of the whole system simulated in QuestaSim is presented. Also in this case, the BER curve is very close to the reference, which shows that implementation and the FP representations were correct. The testing condition were as follows. The E_b/N_0 was between 1,2,.. 14 dB, the number of packets was 800, and the number of symbols per packet was equal to 128 (preamble included). It was necessary to reduce the number of symbols per packet because the simulation tool did not allow higher values.

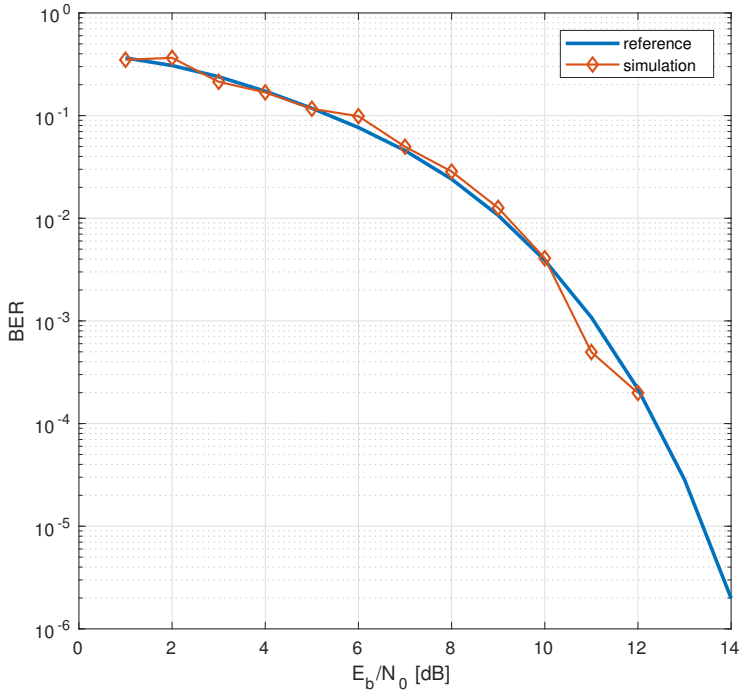

Figure 4.2: BER curve for the Goertzel algorithm block simulated in QuestaSim.

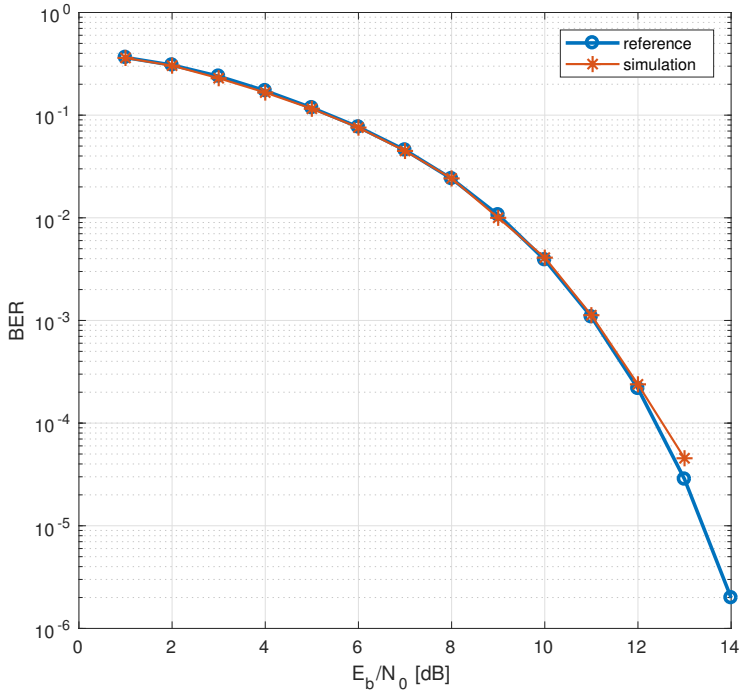

Figure 4.3: BER curve for the complete system block simulated in QuestaSim.

4.3 Power results

Power is either static or dynamic. Static power is due to MOS transistors leakage, mainly to sub-threshold current and to a reduced extent by gate-induced drain leakage, gate leakage, and diode leakage [37] (Chapter 11). Static power is determined by maintaining the input constant. The amount of current that flows during this period of time, multiplied by the input voltage, gives the static power.

Dynamic power instead, is characterised by the switching activity of CMOS gates. It is defined as:

$$P_D = \alpha C V^2 f \tag{4.2}$$

where α is the switching activity, f is the clock frequency, C is the capacitance of the system, and V is the input voltage. Dynamic power is obtained when a non-zero frequency input signal is present.

Power results were collected using the PowerPlay Power Analyzer available in Intel Altera Quartus. The FPGA used was a Cyclone V device 5CGTFD9E5F35C7, because it is part of a development board offered by Intel Altera and the only one big enough to incorporate the design. The software used for analysis, synthesis, fitting, and power estimation was Quartus Prime Version 17.0.0 Build 595 04/25/2017 SJ Standard Edition. Instead, for simulation QuestaSim-64 10.4a was used.

The workflow followed for the power simulation is the one proposed in [38]. Briefly, Quartus is used to generate a netlist file, which is a special description of the design which takes into consideration characteristics of the target device. This file is then used in QuestaSim, which simulates the design with user-defined input. During simulation, the switching activities of each component is registered and saved into a file. At the end of the simulation, the file is passed to PowerPlay Power Analyzer which studies the switching activities and estimates the power consumption.

The FPGA itself consumes a certain amount of power, even when it contains no logic, which corresponds to the static power. Therefore, it is important to determine this amount in order to understand approximately what the design actually consumes. To obtain this information, a very simple design was made in which a single bit input is transferred to the output. Then, the workflow previously explained was used to determine the power consumption. The results are shown in Table 4.1. Power thermal dissipation is the amount of power lost due to heat. It is characterised by static and dynamic power, which arise with different inputs, as previously described. Because the design contains almost no logic for this test, and most importantly does not use a clock, Core Dynamic Power Dissipation is 0. Therefore, when reading other tables, it should be kept in mind that these values are actually due to the use of the FPGA, while they would not be present in an ASIC design.

4.3.1 Power comparison - Signed vs. two's complement arithmetic

In Section 3.3.1 it was explained why signed magnitude was used. The claim was that multiplication by -1 is more power efficient with signed magnitude than two's complement. Instead, the added logic required to carry out the addition with signed magnitude, may outweigh the previous benefit. In this section it will be shown that multiplication by -1 is considerably more

Total Thermal Power Dissipation	Core Dynamic Thermal Power Dissipation	Core Static Thermal Power Dissipation	I/O Thermal Power Dissipation
524.91 mW	0.00 mW	518.59 mW	6.32 mW

Table 4.1: Minimum amount of power always consumed by the FPGA.

	Block type	Total thermal power	Thermal dynamic power	Thermal static power	Routing thermal dynamic power	Average toggle rate (million of transitions/sec)
Two's complement	Combinational cell	0.09 mW	0.01 mW	-	0.08 mW	5.200
	I/O	**17.68 mW**	15.42 mW	2.22 mW	0.04 mW	9.487
Signed magnitude	Combinational cell	-	-	-	-	-
	I/O	**2.78 mW**	0.64 mW	2.15 mW	0.00 mW	0.769

Table 4.2: Comparison of the thermal power dissipation for the multiplication by -1 test.

	Block type	Total thermal power	Thermal dynamic power	Thermal static power	Routing thermal dynamic power	Average toggle rate (million of transitions/sec)
Two's complement	Combinational cell	0.01 mW	-	-	-	1.230
	I/O	**3.03 mW**	0.67 mW	2.34 mW	0.01 mW	1.499
Signed magnitude	Combinational cell	0.01 mW	0.00 mW	-	0.00 mW	1.600
	I/O	**2.93 mW**	0.67 mW	2.25 mW	0.01 mW	1.538

Table 4.3: Thermal Power Dissipation for the addition test.

power efficient in signed magnitude than for two's complement. Moreover, the difference in power consumption of adding two numbers in signed magnitude and two's complement is only slightly in favour of the second representation. For both tests, the inputs were kept the same for both representation, therefore no randomisation was involved on the input data.

Multiplication by -1

The power results obtained for the multiplication by -1 test are shown in Table 4.2. It shows that most of the power is consumed by the I/O. This is because the test designs were made without using internal logic. That does not influence the comparison as the same policy was adopted for both. From the table it is clear that the two's complement is noticeably more power hungry than the signed magnitude. These values are heavily influenced by the toggle rates, shown in the last column. As it was expected, changing sign with two's complement requires a considerate amount of bit switching.

Addition

The results for the thermal power dissipation of the addition test are shown in Table 4.3. In this case, the results of the two representation are very close. Signed magnitude total thermal power for the I/O block slightly exceeds the two's complement by just 0.1 mW.

Conclusion

In the FFT-ZP there are 96 additions and 124 multiplications which switch the sign. Of those multiplications, 96 are actual multiplication by -1 from the butterflies, the other 28 are due to the twiddle factors that become -j. Taken into consideration the power results, it was decided that the trade off between a more power efficient multiplication by -1 was well worth the additional cost due to the more complex signed addition.

Module	Total Thermal Power Dissipation	Core Dynamic Thermal Power Dissipation	Core Static Thermal Power Dissipation	I/O Thermal Power Dissipation
FFT	528.62 mW	**3.74 mW**	518.64 mW	6.24 mW
FFT-ZP	526.92 mW	**1.98 mW**	518.62 mW	6.32 mW
Goertzel	525.16 mW	**0.32 mW**	518.60 mW	6.24 mW

Table 4.4: Power consumption summary for comparison between FFT-ZP, FFT, Goertzel algorithm.

ADC clock (system clock)	Total Thermal Power Dissipation	**Core Dynamic Thermal Power Dissipation**	Core Static Thermal Power Dissipation	I/O Thermal Power Dissipation
80 KHz (2.86 MHz)	530.22 mW	**5.31 mW**	518.66 mW	6.24 mW
8 KHz (286 kHz)	525.38 mW	**0.54 mW**	518.60 mW	6.24 mW
0.8 KHz (28.6 kHz)	524.89 mW	**0.06 mW**	518.59 mW	6.24 mW

Table 4.5: Power consumption summary for comparison between different ADC clock speeds.

4.3.2 Power results comparison for FFT, FFT-ZP, Goertzel

The DFT algorithm is the most power-hungry operation. As such, special attention was dedicated to find alternatives to make it more power-efficient. The proposed solutions were the FFT radix-2, the FFT-ZP described in Section 2.2.3, and the Goertzel algorithm presented in Section 2.3.2. In this section, the power results of these three alternatives will be presented and motivated.

In Table 4.4, the summary of the power consumption is shown. The FFT-ZP consumes almost half of the power of a standard FFT. This is due to the optimisation enabled by the zero padding. In fact, in the standard FFT, there are double the number of registers, double the number of additions, and 72 more multiplications. Clearly, these additional operations produce a significant impact.

Using Goertzel in the data detection instead shows that it allows to save 85% of the power of a standard FFT. For this module, the power simulation was obtained for a single Goertzel block, that is for a single bin, while the demodulator uses two of them in parallel.

4.3.3 Power results for the complete Demodulator

In Table 4.5, the power results for three different clock speeds are shown, when simulating the whole demodulator. By looking at the dynamic power, it can be seen that the power consumption decreases by the same factor by which the ADC is decreased, which is 10. From this, it can be concluded that the impact of the system clock is considerable.

In Table 4.6 the thermal power dissipation for each block is shown. It is read as follows. For example, for the whole Demodulator, Total thermal power (first column) can be calculated in two ways. By summing its immediate sub-entities, indicated with (a), (b), (c) (0.05 mW, 0.02 mW, 1.57 mW respectively), plus the power consumed by that entity alone, indicated in the first column, the value between parenthesis, which is 5.72 mW. The other way is to sum the values indicated out of parenthesis of the other three columns that lie on the same line: Block thermal dynamic power (1.65 mW), Block thermal static power (2.12 mW), Routing thermal dynamic power (3.59 mW). Similarly, any other block's power consumption can be obtained in the same way.

Data was obtained with an ADC clocked at 80 KHz, a system clock frequency of 2.86 MHz, and 10 packets of 128 symbols. In the first column, a sketched hierarchical view of the system

Compilation Hierarchy Node	Total Thermal Power by Hierarchy (1)	Block Thermal Dynamic Power (1)	Block Thermal Static Power (1)(2)	Routing Thermal Dynamic Power (1)
Demodulator	7.36 mW (5.72 mW)	1.65 mW (0.01 mW)	2.12 mW (2.12 mW)	3.59 mW (3.59 mW)
(a)Data acquisition	0.05 mW (0.00 mW)	0.05 mW (0.00 mW)	–	0.00 mW (0.00 mW)
——PD - DA	0.04 mW (0.04 mW)	0.04 mW (0.04 mW)	–	0.00 mW (0.00 mW)
——DD - DA	0.01 mW (0.01 mW)	0.01 mW (0.01 mW)	–	0.00 mW (0.00 mW)
(b)Data Detection	0.02 mW (0.00 mW)	0.02 mW (0.00 mW)	–	0.00 mW (0.00 mW)
——GoertzelH	0.01 mW (0.01 mW)	0.01 mW (0.01 mW)	–	0.00 mW (0.00 mW)
——GoertzelL	0.01 mW (0.01 mW)	0.00 mW (0.00 mW)	–	0.00 mW (0.00 mW)
——MagnitudeH	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	–	0.00 mW (0.00 mW)
——MagnitudeL	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	–	0.00 mW (0.00 mW)
(c)Preamble detector	1.57 mW (0.62 mW)	1.57 mW (0.62 mW)	–	0.00 mW (0.00 mW)
——Even/odd acc.	0.93 mW (0.37 mW)	0.93 mW (0.37 mW)	–	0.00 mW (0.00 mW)
———BOM	0.15 mW (0.00 mW)	0.15 mW (0.00 mW)	–	0.00 mW (0.00 mW)
———BOA	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	–	0.00 mW (0.00 mW)
———FFT-ZP	0.41 mW (0.14 mW)	0.41 mW (0.14 mW)	–	0.00 mW (0.00 mW)
————Stage1	0.02 mW (0.02 mW)	0.02 mW (0.02 mW)	–	0.00 mW (0.00 mW)
————Stage2	0.08 mW (0.00 mW)	0.08 mW (0.00 mW)	–	0.00 mW (0.00 mW)
————Stage3	0.09 mW (0.00 mW)	0.09 mW (0.00 mW)	–	0.00 mW (0.00 mW)
————Stage4	0.08 mW (0.00 mW)	0.08 mW (0.00 mW)	–	0.00 mW (0.00 mW)
——Max of Rs	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	–	0.00 mW (0.00 mW)
——MAX0	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	–	0.00 mW (0.00 mW)
——MAX1	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	–	0.00 mW (0.00 mW)
——MAX2	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	–	0.00 mW (0.00 mW)
——MAX3	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	–	0.00 mW (0.00 mW)
——MAX4	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	–	0.00 mW (0.00 mW)
——MAX5	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	–	0.00 mW (0.00 mW)
——MAX6	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	–	0.00 mW (0.00 mW)
——MAX7	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	–	0.00 mW (0.00 mW)
——R0	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	–	0.00 mW (0.00 mW)
——R1	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	–	0.00 mW (0.00 mW)
——R2	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	–	0.00 mW (0.00 mW)
——R3	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	–	0.00 mW (0.00 mW)
——R4	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	–	0.00 mW (0.00 mW)
——R5	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	–	0.00 mW (0.00 mW)
——R6	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	–	0.00 mW (0.00 mW)
——R7	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	–	0.00 mW (0.00 mW)

(1) Value in parentheses is the power consumed at that level of hierarchy. Value not in parentheses is the power consumed at that level of hierarchy plus the power consumed by all levels of hierarchy below it.
(2) The "Block Thermal Static Power" for all levels of hierarchy except the top-level hierarchy is part of the "Core Static Thermal Power Dissipation" value. The "Core Static Thermal Power Dissipation" also contains the thermal static power dissipated by the routing.

Table 4.6: Thermal Power Dissipation by Hierarchy for the Demodulator.

is shown, where more indentation to the right indicates that the module is a child of the first previously less indented parent (e.g. BOM is child of Even/odd acc.). The Preamble detector, together with its sub-modules, is the unit that consumes most of the power. Modules that have power consumption indicated to be zero are actually consuming an amount smaller $10\mu W$, which the tool does not take care to report.

4.4 Resources utilisation

In this section, the resource utilisation of the demodulator is presented. In Fig. 4.4, the number of Adaptive Logic Module (ALM) is shown for each node of the hierarchy. An ALM is a basic block in the Altera FPGAs.

The table is read as follows. For the whole demodulator, the total number of ALMs is given by the ALMs needed only by the demodulator alone (10.70) plus ALMs needed by its sub-entities (a), (b), and (c) (62,699.40, 452.40, 1,741.30 respectively). Roman numerals have indicated the blocks that belong to the Preamble detector. The Preamble detector requires 96% of the total design of said resource. In particular, the Even/odd accumulator takes 63% of the total used ALMs. The sum of the submodules resource utilisation (FFT-ZP, BOA, BOM) is roughly 46%. The remaining resources are used by the Even/odd accumulator itself. The whole system requires 64,904 ALMs, compared to the 113,560 ALMs available on the target FPGA. The number of used registers is 42,177, that is 19% of those available on the FPGA. Finally, 32 out of 616 available pins were used.

Compilation Hierarchy Node	ALMs needed (for entity and sub entities)	ALMs needed (only current entity)
DEMODULATOR	64,903.70	10.70
(a) PREAMBLE_DETECTOR:preamble_detector_unit	62,699.40	7,828.20
(I) even_odd_accumulator:EOA	41,089.10	10,615.40
FFT_ZP:fft	19,921.40	1,145.70
fft_stage_1:S1	3,259.10	145.60
fft_stage_2:S2	6,726.90	0.00
fft_stage_3:S3	5,344.50	0.00
fft_stage_4:S4	3,445.20	0.00
(II) bank_of_adders:BOA	247.00	1.50
(III) bank_of_magnitude:BOM	10,305.20	0.00
(IV) maxs:MAXSu0	1,692.90	0.00
max_finder64:max_even	839.20	839.20
max_finder64:max_odd	853.70	853.70
(V) maxs:MAXSu1	1,704.30	0.00
(VI) maxs:MAXSu2	1,710.20	0.00
(VII) maxs:MAXSu3	1,703.50	0.00
(VIII) maxs:MAXSu4	1,705.60	0.00
(IX) maxs:MAXSu5	1,710.30	0.00
(X) maxs:MAXSu6	1,712.00	0.00
(XI) maxs:MAXSu7	1,699.80	0.00
(XII) R:R0	13.60	13.60
(XIII) R:R1	12.50	12.50
(XIV) R:R2	11.80	11.80
(XV) R:R3	12.60	12.60
(XVI) R:R4	13.20	13.20
(XVII) R:R5	10.90	10.90
(XVIII) R:R6	13.40	13.40
(XIX) R:R7	13.70	13.70
(XX) max_of_Rs:MAX_OF_RSu	41.70	41.70
(b) data_acquisition:data_acquisition_unit	452.40	1.00
dd_data_acquisition:dd_da	59.00	59.00
pd_data_acquisition:pd_da	392.40	392.40
(c) data_detection:data_detection_unit	1,741.30	29.70
GOERTZEL:grzH	708.30	94.70
magnitude_grz:mag_grzH	119.20	17.50
GOERTZEL:grzL	766.00	147.40
magnitude_grz:mag_grzL	118.00	15.70

Chapter 5

Conclusion and future work

In this section the work produced in this thesis will be summarised. The research questions will be compared with the obtained results and recommendations for future work will be outlined afterwards.

5.1 Conclusion

The objective of this thesis was to implement a power-efficient demodulator that uses the algorithm described in [4] for BFSK modulation scheme. In Section 1.5, three research questions were defined in order to help achieving a power-efficient implementation. Firstly, the two DFTs used in the algorithm were studied.

> **RQ1.** How can the two DFT be implemented in a more power-efficient way?

It was shown in Section 2.2 and in Section 2.3 that the radix-2 FFT is not the best option in the presented context. In fact, the presence of zero-padding can be exploited in order to obtain a less computationally demanding algorithm. For the synchronisation block, the FFT-ZP was presented in Section 2.2.3 and implemented in Section 3.3.5. For the data detection, only two bins were required to be computed. Therefore, a complete FFT would have produced 64 bins of which 62 would have been discarded. Because of that, the Goertzel algorithm was presented in Section 2.3.2 and implemented in Section 3.3.4. The power comparison obtained in Section 4.3.2 confirmed that the two alternatives to the standard FFT are more power-efficient.

Data representation is an important aspect of every design and as such needed to be addressed.

> **RQ2.** What is the smallest number representation so that the BER is the same as when full precision is used?

This question does not have a single answer. It was shown in Section 3.2 that each block of the algorithm can be optimised for different fixed-point representations. This approach led us to save on the number of registers needed for the computation, as a consequence also on the power required.

> **RQ3.** What is the performance and the resource usage of the proposed implementation?

In Section 4.1 the relation between the ADC frequency and the clock frequency of the system was derived. This relation needs to hold that needs to hold, in order to have the system operates correctly. In Section 4.3, extensive data was collected regarding the power consumption of the system. Some of this information was used to motivate design choices, others could be used in the future to make quantitative comparison with different implementations.

In conclusion, all three research questions were addressed and the initially planned goal was achieved.

5.2 Future work

In this thesis, the implementation of a frequency tolerant algorithm for BFSK was proposed. As the designed system is fully digital, an ADC had to be modelled in absence of system requirements. This aspect was just borderline with the scope of this work, nonetheless a starting point needed to be defined. In Section 3.2.4, a mid-riser model with 8 bit precision was deemed as a good solution, based on simulation. However, additional research should be dedicated to this component to verify whether a less precise ADC could still produce correct results, while consuming less power.

Another aspect is the chosen platform. A decision had to be made regarding whether to use a certain vendor or another. In Section 3.4.2, the Intel Altera was chosen as a matter of experience. Because of that, part of the IPs are now specific to this vendor. It would be interesting to make the design vendor-independent and test it also with other FPGA manufacturers, like Xilinx or Lattice, and ultimately compare the results.

The major concern of this design was to make the system as power efficient as possible. Using an FPGA is in fact the opposite of that. This IC consumes a lot of static power, as simulation did show in section 4.3, and therefore would never be used in an actual device. That does not mean the work produced until now is a waste. Indeed, what is necessary now is to divide the development in the ASIC flow and the FPGA flow. The first one is needed in order to obtain more accurate power consumption estimation. Instead, the second should be used to produce an actual prototype. This split can be easily achieved because the produced implementation, written in VHDL, can be quickly ported to an ASIC design.

Another feature that could be tackled is the comparison with existing IPs for the FFT. In Section 4.3.2, a comparison of the FFT and FFT-ZP was presented. Their structures were voluntarily made similar in order to guarantee a fair comparison, while demonstrating that taking advantage of the zero-padding made the FFT-ZP a more power efficient solution. Nonetheless, it should be investigated whether different architectures could achieve better results. For example, in [39] radix-2^2 architecture is presented, while also reviewing existing ones like multi-path delay-commuter or single-path delay-feedback.

Bibliography

[1] "Cornelis drebbel (1572 - 1633)." `http://www.drebbel.net/Tierie.pdf`. Accessed: 2018-10-22.

[2] J. Hao, B. Zhang, and H. T. Mouftah, "Routing protocols for duty cycled wireless sensor networks: A survey," *IEEE Communications Magazine*, vol. 50, pp. 116–123, December 2012.

[3] I. Demirkol, C. Ersoy, and E. Onur, "Wake-up receivers for wireless sensor networks: benefits and challenges," *IEEE Wireless Communications*, vol. 16, pp. 88–96, Aug 2009.

[4] S. Safapourhajari and A. B. J. Kokkeler, "Frequency offset tolerant demodulation for low data rate and narrowband wireless sensor node," in *2017 11th International Conference on Signal Processing and Communication Systems (ICSPCS)*, pp. 1–8, Dec 2017.

[5] A. Forster, *Introduction to wireless sensor networks*. John Wiley & Sons, 2016.

[6] P. Rawat, K. D. Singh, H. Chaouchi, and J. M. Bonnin, "Wireless sensor networks: a survey on recent developments and potential synergies," *The Journal of Supercomputing*, vol. 68, pp. 1–48, Apr 2014.

[7] L. M. Borges, F. J. Velez, and A. S. Lebres, "Survey on the characterization and classification of wireless sensor network applications," *IEEE Communications Surveys Tutorials*, vol. 16, pp. 1860–1890, Fourthquarter 2014.

[8] H. M. A. Fahmy, *Wireless Sensor Networks Concepts, Applications, Experimentation and Analysis*. Springer, Singapore, 2016.

[9] A. Asiz, W. Zhang, and Y. Xi, "Analysis of aging of piezoelectric crystal resonators," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 50, pp. 1647–1655, Dec 2003.

[10] D. A. Gudovskiy, L. Chu, and S. Lee, "A novel nondata-aided synchronization algorithm for msk-type-modulated signals," *IEEE Communications Letters*, vol. 19, pp. 1552–1555, Sept 2015.

[11] H. B. Çelebi and H. Arslan, "A joint blind carrier frequency and phase offset detector and modulation order identifier for mpsk signals," in *2010 IEEE Radio and Wireless Symposium (RWS)*, pp. 348–351, Jan 2010.

[12] J. Sun and X. Li, "Carrier frequency offset synchronization algorithm for short burst communication system," in *2016 IEEE 13th International Conference on Signal Processing (ICSP)*, pp. 1231–1235, Nov 2016.

[13] E. Lopelli, J. Van der Tang, and A. Van Roermund, "A fsk demodulator comparison for ultra-low power, low data-rate wireless links in ism bands," in *Circuit Theory and Design, 2005. Proceedings of the 2005 European Conference on*, vol. 2, pp. II–259, IEEE, 2005.

[14] "The doppler effect." `https://www.physicsclassroom.com/class/waves/Lesson-3/The-Doppler-Effect`. Accessed: 2018-11-12.

[15] S. Hara, A. Wannasarnmaytha, Y. Tsuchida, and N. Morinaga, "A novel fsk demodulation method using short-time dft analysis for leo satellite communication systems," *IEEE Transactions on Vehicular Technology*, vol. 46, pp. 625–633, Aug 1997.

[16] M. K. Simon and D. Divsalar, "On the implementation and performance of single and double differential detection schemes," *IEEE Transactions on Communications*, vol. 40, no. 2, pp. 278–291, 1992.

[17] M. R. Yuce and W. Liu, "A low-power multirate differential psk receiver for space applications," *IEEE transactions on vehicular technology*, vol. 54, no. 6, pp. 2074–2084, 2005.

[18] F. Hlawatsch and G. F. Boudreaux-Bartels, "Linear and quadratic time-frequency signal representations," *IEEE signal processing magazine*, vol. 9, no. 2, pp. 21–67, 1992.

[19] E. Jacobsen and R. Lyons, "The sliding dft," *IEEE Signal Processing Magazine*, vol. 20, pp. 74–80, Mar 2003.

[20] E. Jacobsen and R. Lyons, "An update to the sliding dft," *IEEE Signal Processing Magazine*, vol. 21, no. 1, pp. 110–111, 2004.

[21] K. Duda, "Accurate, guaranteed stable, sliding discrete fourier transform [dsp tips amp; tricks]," *IEEE Signal Processing Magazine*, vol. 27, pp. 124–127, Nov 2010.

[22] C. Donciu and M. Temneanu, "An alternative method to zero-padded dft," *Measurement*, vol. 70, pp. 14 – 20, 2015.

[23] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE transactions on computers*, vol. 100, no. 8, pp. 786–793, 1973.

[24] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," *Journal of the ACM (JACM)*, vol. 27, no. 4, pp. 831–838, 1980.

[25] R. P. Brent and H.-T. Kung, "A regular layout for parallel adders," *IEEE transactions on Computers*, no. 3, pp. 260–264, 1982.

[26] D. Knuth, "The art of computer programming 1: Fundamental algorithms 2: Seminumerical algorithms 3: Sorting and searching," *MA: Addison-Wesley*, vol. 30, 1968.

[27] A. Schönhage and V. Strassen, "Schnelle multiplikation grosser zahlen," *Computing*, vol. 7, no. 3-4, pp. 281–292, 1971.

[28] M. Fürer, "Faster integer multiplication," *SIAM Journal on Computing*, vol. 39, no. 3, pp. 979–1005, 2009.

[29] E. d. R. Fabrizio Argenti, Lorenzo Mucchi, *Elaborazione numerica dei segnali*. McGraw-Hill, 2011.

[30] R. Yavne, "An economical method for calculating the discrete fourier transform," in *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pp. 115–125, ACM, 1968.

[31] G. Goertzel, "An algorithm for the evaluation of finite trigonometric series," *The American Mathematical Monthly*, vol. 65, no. 1, pp. 34–35, 1958.

[32] M. Proakis, *Digital signal processing: principles algorithms and applications*. Pearson Education India, 2007.

[33] P. Duhamel, "Implementation of "split-radix" fft algorithms for complex, real, and real-symmetric data," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 34, pp. 285–295, Apr 1986.

[34] C. Inacio and D. Ombres, "The dsp decision: fixed point or floating?," *IEEE Spectrum*, vol. 33, pp. 72–74, Sept 1996.

[35] J. Janhunen, T. Pitkanen, O. Silven, and M. Juntti, "Fixed- and floating-point processor comparison for mimo-ofdm detector," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, pp. 1588–1598, Dec 2011.

[36] B. Parhami, *Algorithms and design methods for digital computer arithmetic*. Oxford University Press, 2012.

[37] C. Piguet, *Low-power electronics design*. CRC Press, 2004.

[38] "Power analysis with quartus ii and modelsim." `http://wwwhome.cs.utwente.nl/~molenkam/ods/low_power_exercise/dds-power.pdf`. Accessed: 2018-09-05.

[39] S. He and M. Torkelson, "A new approach to pipeline fft processor," in *Parallel Processing Symposium, 1996., Proceedings of IPPS'96, The 10th International*, pp. 766–770, IEEE, 1996.