POLITECNICO DI TORINO

Master of Science in Electronic Engineering

Master Degree Thesis

Automatic signal detection and classification with software define radio

Advisors:
prof. Maurizio Martina
prof. François Quitin

Candidate:
Riccardo Pace

Conducted at Université Libre de Bruxelles

October 2017

To Michela

Acknowledgments

My thesis has been written down and it is ready to be printed, at last. Only the acknowledgements need to be written. In my university years in Turin and the last months spent in Brussels, I met a lot of people who helped me to learn and grow. I would like to thank all of them.
First, I would like to thank my second advisor, François Quitin, from BEAMS department at Université Libre de Bruxelles. He gave me the opportunity to develop my thesis work at the ULB and made my experience abroad wonderful. His office door was always open for my questions and he directed my research in the best possible way. Thank you to the whole BEAMS department for the lab support and their help.
I would also like to thank my advisor Maurizio Martina from DET at Politecnico di Torino. I am grateful to him for all his comments and suggestions on this thesis.
I would also like to acknowledge Ing. Wim Aerts at BIPT who showed great interest in my thesis, especially during the intermediate meetings, commenting my work and suggesting solutions.
Thanks to my colleagues and friends, Gabriele, Simone and Gabriele, for all the projects carried out during these academic years, and for creating a great method to work all together as a team.
I want and I must express all my gratitude to my parents, Michele and Carmen, for giving me the opportunity to study and for providing me with continuous encouragement during my studying process; to my brothers, Gabriele and Andrea, and my sister-in-law, Elsa, for providing me with unfailing support and continuous incentive. I would like to thank also my uncles, Piero and Carla, for helping me do my best.
I cannot forget to thank my fabulous roommates: Roberta, Marta, Noemi and Giacomo, for the wonderful time shared together in Turin. One special acknowledge is to all my friends: Eleonora, Andrea, Michela, Fabrizio, Luisa, Andrea, Francesca, Marta, Eliana, Andrea, Gabriele, Ilenia, Alberto, Anna and Paola for providing me with super encouragement and support during these last three years.
Finally, I would like to thank all the people who I forgot to mention, but helped me in all these years of study.

Summary

This master thesis describes how an automatic spectrum scanner and signal detector for radio frequency (RF) has been implemented using a software define radio platform. A first analysis of the various spectrum sensing techniques permits to understand which kind of resources are needed to implement the detection of signals (Chapter 2). Secondly, a study of the USRP-N210 device, used for the scanning implementation has been carried out in order to evaluate how and which of the previous techniques can be embedded in it (Chapter 3).
Thanks to the data/information acquired with the previous analysis two different algorithms are proposed: a manual and an automatic approach. On the basis of these two algorithms the respective hardware and software tasks have been defined (Chapter 4) creating a mixed FPGA/Software architecture. This permits to balance in a good way the work load of the device and the host CPU connected to the device efficiently.
The design flow for the hardware part follows a chain structure designing different hardware modules which are connected in a cascade mode (Chapter 5). Some main modules are shared among the two architectures (corresponding to the respective algorithm) and others are specific to the application.
The software part (Chapter 6) exploits some distinct features of the device used to implement the system. The main task executed by the software is related to control the SDR device especially to manage the tune operation of the receiving carrier frequency.
Different results have been analysed: from the FPGA resources usage side to the system detection performances (Chapter 7). Some experiments on the designed spectrum scanner system were performed and a real case of spectrum scan with the FM broadcasting in Brussels has been evaluated.

Table of contents

List of figures

Chapter 1

Introduction

1.1 Motivation

With the increased use of portable devices and the growing demand for greater data transmission rates, an increasing request of spectrum channels has been observed over the last decade.
All the available spectrum channels are licensed by a specific Institute in each country, and the licenses can change for different areas. Users buy expensive licenses to transmit on specific frequency bands and the regulator must guarantee a clean spectrum and make sure that there is no misuse of spectrum.
In Belgium the "BIPT" (*Belgian Institute for Postal services and Telecommunications*) has the task to control the correct use of spectrum frequency bands and detect abusive usage of radio frequency (RF). To carry out this control, the BIPT has some vehicles, with proper measurement instruments mounted inside, which travel along the whole Belgium to store data and also understand the appropriate spectrum usage by the transmitting companies.
Traditionally, spectrum analyzers are used to perform spectrum scanning, but spectrum analysers tend to be bulky, expensive and fragile.
In the context of ever-faster digital logics and increasing quality requirements, wireless transmitter and receiver are moving towards software-define platforms.
Recently, software-defined radio (SDR) has gained importance and has become an alternative solution for a wide variety of applications, for example the spectrum monitoring. These devices are cheaper and less bulky than the spectrum analysers, so that they could be mounted into the vans of the regulator technicians.
By exploiting the SDR features, an automatic spectrum scanner system can be implemented, whose aim is to continuously scan the RF spectrum and to detect whether any signals are present in a stand-alone mode.

1.2 Objective

The objective of this thesis is to implement an automatic spectrum monitoring system on a USRP-N210 software define radio, a popular and cheap model. The system needs to scan the spectrum for RF signals, detect if the bands are occupied or not, compare the signals detected with a database (also taking into account geographic location) and alert the user when an infraction is detected.

One common problem when using USRPs is that the CPU load of the host computer becomes unacceptable, due to the high sample rates. Taking advantage of the USRP-N210 features, a mixed FPGA/software approach, for implementing a spectrum monitoring system, could be used in order to decrease the host CPU workload.

Chapter 2

Theoretical background

2.1 Spectrum sensing

With the increasing usage of the Cognitive radio techniques, for realizing comfortable society by a wireless communication system, the request and usage of wireless resources has become the major issue in the world. Thanks to this context the spectrum sensing has gained an increasing interest by the researchers. This Section has the aim to give a brief overview of the spectrum sensing techniques available in literature.

The *Conventional spectrum sensing* can be modelled as a single primary system to decide between two hypotheses:

$$y[n] = \begin{cases} w[n] & H_0 \\ s[n] + w[n] & H_1 \end{cases} \tag{2.1}$$

where $y[n]$ is the complex signal received by the device, $s[n]$ the transmitted signal and $w[n]$ is the additive white Gaussian noise (AWGN), and n can be from *1* to N representing the observation interval. For a more detailed model the signal s[n] has to be multiplied by the channel gain.
H_0 represents the null hypothesis that no transmitted signal is present and H_1 represents the presence of a signal.
The main spectrum techniques can be listed as following:

- *Energy detection*;
- *Matched filter*;
- *Cyclostationary detection*;
- *Walvelet detection.*

2.1.1 Energy detection

If the signal to be detected is unknown, the energy detection method is the optimal solution to find the signals inside a spectrum.
In this approach the received energy signal is measured over an observation time to determine the occupancy of the spectrum. Two ways exist to evaluate the energy: in the time domain and in the frequency domain, thanks to the Parseval's Theorem. The energy in the time domain is defined as following:

$$E = \sum_{n=1}^{N} |y[n]|^2 \tag{2.2}$$

The received signal (*complex or real*) is squared and integrated over an observation interval T. The result of this operation is compared with a threshold to decide the presence of some signal. The model of the decision can be represented with the following system (2.3):

$$D = \begin{cases} H_0 & E \lessgtr \lambda \\ H_1 & otherwise \end{cases} \tag{2.3}$$

Where the λ factor is the threshold. If the frequency domain is selected, recalling the *Parseval's theorem* the energy can be evaluated and compared with the threshold.

The Figure 2.1 represents the frequency domain architecture, instead the Figure 2.2 the time domain one.

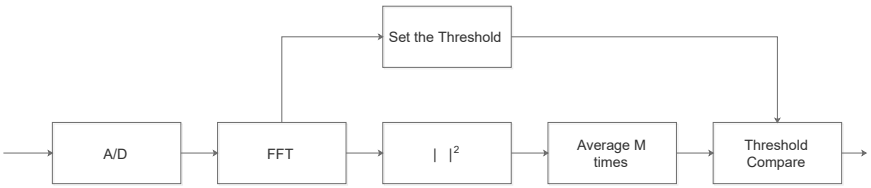

Figure 2.1: Energy detection architecture in frequency domain

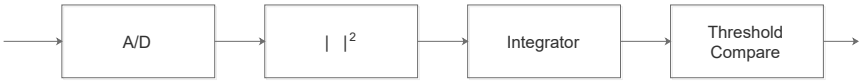

Figure 2.2: Energy detection architecture in time domain

This technique can be implemented without any a priori knowledge of the primary user signal. The threshold selection for energy detection can be a challenge

because it is very susceptible to the changing of the noise and especially to the interference level.

2.1.2 Matched filter

When the received signal is in the presence of additive Gaussian noise the Match filter approach can be an optimal detection method because it maximises the SNR of the received signal. Respect the previous method, the matched filter requires a priori knowledge of the signal received: pulse shaping, modulation type and packet format. This approach correlates a known signal with a unknown signal to evaluate the presence.
The mechanism is to perform a convolution of the received signal with a time-reversed version of the assumed signal and the result is compared with a threshold. Therefore the following binary decision can be done:

$$T = \sum_{n=1}^{N} y[n] * x[n] \tag{2.4}$$

$$D = \begin{cases} H_0 & T \lessgtr \lambda \\ H_1 & otherwise \end{cases} \tag{2.5}$$

The typical usage of the matched filter is the radar transmission. If partial information of the signal to be detected is known, such as pilots or preambles, the use of this method is still possible for a right detection. The advantage of using this technique lies on the time to achieve a good result. However this means that for each signal standard type a dedicated receiver is needed, which increases complexity as a significant challenge when the target system is out of multiple possible systems(Eq 2.5).

The Figure 2.3 shows a possible architecture based on the pilot frequency of the primary system.

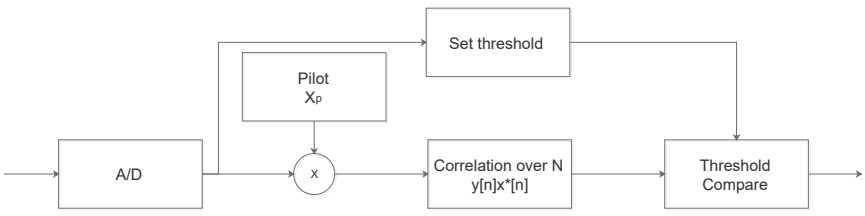

Figure 2.3: Matched filter architecture using pilot primary system information

2.1.3 Cyclostationary detection

Evaluating the cyclostationary features of the target signals, the transmitted signal can be detected. The idea is to exploit the periodicity of the modulated signal such as sine wave carries, pulse trains, repeating spreading, cyclic prefixes and hopping sequences. The cyclostationary signals manifest correlation between separated spectral components due to the spectral redundancy caused by the periodicity.
The spectral correlation is one important characteristic property of a cyclostationary random process. On the other hand the stationary processes have not this property, no pairs of distinct frequency components are correlated. Due to this difference, the cyclostationary detector can distinguish the noise energy from the modulated signal energy, because the noise is a generalized stationary process that it does not have periodicity and also no spectral correlation.

The cyclic autocorrelation function (CAF) of the observed signal $x(t)$ can be evaluated as following:

$$R_x^\alpha(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_{-T/2}^{T/2} x(t+\frac{\tau}{2})x(t-\frac{\tau}{2})e^{-j2\pi\alpha t}dt \tag{2.6}$$

where α is the cyclic frequency.
The spectral correlation function can be computed by the discrete Fourier transform of the Eq. 2.6 :

$$S_x^\alpha(f) = \int_{-\infty}^{\infty} R_x^\alpha(\tau)e^{-j2\pi f\tau}d\tau \tag{2.7}$$

Substituting the expression of CAF :

$$S_x^\alpha(f) = \lim_{T\to\infty}\lim_{Z\to\infty} \frac{1}{TZ} \int_{-Z/2}^{Z/2} X_T(t,f+\frac{\alpha}{2}\tau)X_T^*(t,f-\frac{\alpha}{2}\tau)dt \tag{2.8}$$

where

$$X_T(t,f) = \int_{t-T/2}^{t+T/2} x(u)e^{-j2\pi fu}du \tag{2.9}$$

The detection is performed by searching the unique cyclic frequency corresponding to the peak of the spectral correlation function plane.

A standard architecture to perform the cyclostationary detection is shown in the Figure 2.4. Usually this method is implemented in digital domain. First the spectral components of the data through FFT are evaluated, and then direct algorithms perform the spectral correlation on the spectral components. As mentioned above the main advantage of this technique for spectrum sensing is the ability to distinguish

the noise energy from the signal energy. For this aspect the method is more roboust respect the energy detector. Moreover it can work with low SNR, but the computational processing gain can be higher than the energy one. The implementation is more complicated and it requires more longer observation time.

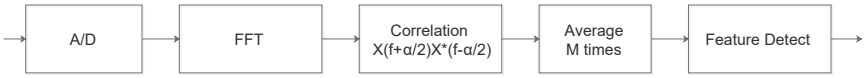

Figure 2.4: Cyclostationary detection architecture

2.1.4 Walvelet detection

The walvelet detection method treats the entire wideband as a sequence of consecutive frequency sub-bands, where the power spectral characteristic is smooth within each sub-band, but changes on the border of two neighbouring sub-bands. If the walvelet transform is applied on the power spectral density (PSD) of the received signal, the singularities can be located where there are holes in the spectrum. This method is suitable for wide band signals, the walvelet approach offers advantages in implementation cost but also in flexibility.

The Figure 2.5 shows the digital implementation of walvelet detection.

Figure 2.5: Walvelet detection architercture

2.1.5 Summary Spectrum Sensing Techniques

The analysis touched in this Section 2.1 permits to have main idea of the spectrum sensing techniques available in literature. In the Table 2.1 is reported a summary of the different techniques explained above.

The purpose of this thesis is to develop a spectrum scanner system able to detect several type of signals without know any priory information of these. The detection algorithm has to be embedded in the FPGA of the USRP N210, so a simple architecture with low computational cost has to be considered. Analysing the Table 2.1, the *Matched Filter* and the *Cyclostationary detection* techniques cannot be considered because they need priori information of the received signal even if they are more robust than the other techniques. *Walvelet detection* instead could be a

Spectrum Sensing Techniques	Pros	Cons
Energy detection	• Does not require any a priori knowledge of signal • Low computational cost	• Cannot work with low SNR • Cannot distinguish signals sharing the same channel
Matched Filter	• Optimal detection performance • Low computational cost • Maximize SNR	• Requires a priori knowledge of the signal • Each type of signal requires its design
Cyclostationary detection	• Roboust in low SNR • Roboust to interface	• It need partial information of the transmitted signal • High computational cost
Walvelet detection	• Effective for wideband signal	• Does not work for spread spectrum signals • High computational cost

Table 2.1: Summary Spectrum Sensing Techniques from [4]

good technique to sense signals but it is limited by the high computational cost. The best technique that fits the purpose of the thesis is the *Energy Detection*; it allows to detect signals without any priory information and with a low computational cost. In order to evaluate sub-bands in a slice of spectrum, wide as the sample frequency, the frequency domain architecture (Figure 2.1) has been used.
The following Section 2.2 explains methods to select the threshold for the Energy detection sensing technique.

2.2 Energy Detection: Threshold Study

To understand if a signal is present or not an appropriate threshold has to be selected. A lot of studies has been done on how to select the threshold for the energy detection technique. The energy detection of unknown deterministic signals has been developed by *Harry Urkowitz*[5] in the 1967. Starting from the mathematical model created by him, several studies have been done to find an optimal threshold for the detection of unknown signals.

In the energy detector system, the energy of the signal sampled is evaluated. If

we consider the time domain, the energy is defined as:

$$\Delta = \sum_{n=1}^{N} |y(n)|^2 \tag{2.10}$$

The variable Δ, also called *test statistic*, is compared with the threshold λ. Exploiting the Parseval's theorem or Rayleigh's energy theorem, the value of Δ is equivalent to $\Delta = \sum_{k=1}^{N} |Y(k)|^2$ where $Y(k)$ is the frequency domain representation of $y(n)$.

As reported in the Equation 2.3, two different scenarios can be distinguished. When there is only noise ($w_n[n]$) the detection result is *false* instead when there is the sum of signal and noise, the detector's result is true.
The noise is assumed as a random process of zero mean and variance σ_w^2 instead the signal is also assumed to be random process of zero mean and variance σ_s^2.

On both of the hypothesis H_0 and H_1, the test statistic Δ can be modelled as a random variable whose probability density function (PDF) is chi-square distributed. Exploiting the *Central Limit Theorem* when the number of samples N is sufficiently large, the hypothesis's PDF can be approximated using Gaussian distribution.

Knowing these distributions two performance metrics of spectrum sensing can be defined:

- *Probability of detection* (P_d)
- *Probability of false alarm* (P_f)

The detection probability P_d is the probability that a signal is correctly detected through sensing. P_d can be derived from the model of the hypothesis and assume the following expression(Equation 2.11):

$$P_d = P(\Delta > \lambda | H_1) = Q(\frac{\lambda - (\sigma_w^2 + \sigma_s^2)}{(\sigma_w^2 + \sigma_s^2)/\sqrt{N/2}}) \tag{2.11}$$

The false alarm probability P_f instead is defined as the probablity that a signal is detected inside the spectrum even if the signal is not present. P_f assumes the following expression(Equation 2.12):

$$P_f = P(\Delta > \lambda | H_0) = Q(\frac{\lambda - \sigma_w^2}{\sigma_w^2/\sqrt{N/2}}) \tag{2.12}$$

From the model of the signal and the noise the SNR can be defined as : $\gamma = \sigma_s^2/\sigma_w^2$. The two probabilities defined above are both dependent on the threshold λ.

Hence the decision threshold can be derived for a target P_d or P_f.
From the Equation 2.11 of P_d the threshold, λ_{P_d}, can be evaluated for a constant detection rate (CDR):

$$\lambda_{P_d} = (\sigma_w^2 + \sigma_s^2)(1 + \frac{Q^{-1}(P_d)}{\sqrt{N/2}}) \tag{2.13}$$

Similarly, using the P_f can be chosen a threshold for a target constant false alarm rate (CFAR) :

$$\lambda_{P_f} = \sigma_w^2(1 + \frac{Q^{-1}(P_f)}{\sqrt{N/2}}) \tag{2.14}$$

The CFAR threshold depends by the noise variance, the number of samples and the value of P_f; instead the CDR needs also the signal variance knowledge and obviously the P_d. Recalling the definition of SNR, the CDR threshold can be manipulated; the signal variance can be rewritten based on σ_w^2 and γ. For both the thresholds, the noise variance has to be estimated in order to evaluate the threshold.

As can be noticed in the Equations 2.11 and 2.12, the sensing performances are strongly affected by the selection of the threshold. A fixed threshold approach is very common in the conventional energy detection method. Moreover usually the CFAR threshold is applied in energy detection algorithms.
The threshold based on CDR can be used to increase the sensitivity of signal detection, the P_d as to be settled as high as possible. On the other hand if the idea is to have a robust detection, the threshold based on CFAR should be used. In this case the probability of false Alarm has to be settled as lower as possible. As mentioned above this kind of threshold depends strongly by the noise and guarantee the target probability false alarm is very difficult due to the noise power fluctuations.
The conventional threshold derivation has one main problem that it only considers one aspect at time, either respect to the P_d or P_f.

To avoid the problems related to a conventional fixed threshold sensing algorithm, a lot of researchers studied a new approach consisting to adapt the threshold with some measurements performed on the received signal. The adaptive threshold sensing algorithm allows to adjust the energy threshold according to the SNR, transmit power, sensing time.

For example in [6] an adaptive threshold is studied according the SNR. The objective of this type of threshold is to maximize the detection of bands with no signal using a new optimized policy function. The detection threshold is dynamically adapted using a linear increasing function of the SNR observed. Simulation results show that this new threshold approach, respect the conventional fixed threshold detector, achieves best performances and stability in terms of false alarm and missed

detection probabilities.

The approach followed in [7] [8] for the adaptive threshold detector is to find a trade-off between P_d and P_f defining a new error decision probability P_e and minimizing it (Equation 2.15). In this analysis the author also took into account the spectrum utilization of the transmitted signals with a parameter α.

$$minP_e(\lambda) = min\{(1-\alpha)P_f + \alpha(1-P_d)\}; \tag{2.15}$$

The final simplified expression (Equation 2.16) of the threshold, if the number of samples is very high, depends only by σ_w^2 and the SNR.

$$\lambda^* \approx \frac{2\sigma_w^2 \cdot (1+SNR)}{2+SNR} \; (N \to +\infty); \tag{2.16}$$

In the [9] a similar method has been used but instead to minimize the error decision probability, the intersection of two probability density functions (PDF) is evaluated. According to the central limit theorem, the authors modelled the energy of the received signal as a Gaussian random variable with different characteristics based on the case reported on the binary hypothesis 2.1 of the Conventional Energy Detector. Assuming $P_{H_0} = \frac{1}{3}$ and $P_{H_0} = \frac{2}{3}$ they searched the intersection of the (PDF) under the two hypothesis. The final expression of the threshold, in this case, depends only by the SNR and the number of samples taken into account.

In conclusion to have a good reference threshold, both threshold mechanisms have to estimate the noise power. For the case of a fixed threshold, the noise power is useful to know a base threshold level. Instead for the adaptive one, the noise power estimation has to be performed at run time to compute the correct SNR and so the threshold value.

2.3 Fast Fourier Transform review and architectures

The aim of this section is to give a brief overview of the differences between some FFT algorithms and architectures.

The Discrete Fourier Transform (DFT) is a discrete approach to perform a Fourier Transform. The concept of the DFT is to transform a discrete signal in the time-domain to the frequency domain. The DFT can be described as a sequence of N complex numbers x_0,x_1,...,x_{N-1} transformed into another sequence of complex

numbers X_0,X_1,...,X_{N-1}. The DFT is defined as following:

$$X_n = \sum_{k=0}^{N-1} x_k \cdot e^{-\frac{2\pi \cdot i \cdot n \cdot k}{N}} \tag{2.17}$$

The number of operations to perform DFT of a signal using direct DFT method is proportional to N^2. To reduce the number of operation for the Fourier Transform in the discrete domain, Cooley and Tukey developed the Fast Fourier Transform (FFT) method.
The DFT is a mathematical operation instead the FFT is a series of algorithms used to implement DFT in practise on a digital machine. Performing a certain sequences of operations (addition and multiplications), FFT eliminates redundancies existing in the DFT. The computational efficiency is achieved thanks to the additional reordering steps to determine the final results and it is proportional to $N \cdot log(N)$.

2.3.1 FFT Algorithms

In literature exist a large number of FFT algorithms and their aim is to reduce the complexity of the DFT. The most common FFT algorithms are:

- **Cooley-Tukey** algorithm uses the general principle of divide and conquer. It breaks the DFT to smaller sub problems which can be achieved in the time or in the frequency domain.
- **Bruun's FFT** algorithm is based on recursive factorization of polynomials. The idea is based on applying filters to derive the product of monomials.
- **Bluestein's FFT** algorithm re-elaborates the DFT as a convolution.

Only the Cooley-Tukey's will be touched more in detail because it has been used in the implementation of the system.

Cooley-Tukey

The basic idea of this algorithm is to split the DFT into smaller sizes DFTs in order to reduce the computational time. The DFT can be written as :

$$X_n = x_0 + x_1 \cdot W(1)^n + \cdots + x_k \cdot W(k)^n \tag{2.18}$$

The expression 2.18 can be re-expressed in composites, the N terms can be split in two composites odd and even parts. The twiddle factor ($W(k)^n$) on the odd part is taken outside the parenthesis in order to have the even expression:

$$X_n = [x_0 + x_2 \cdot W(2)^n + \cdots] + W(1) \cdot [x_1 + x_3 \cdot W(2)^n + \cdots] \tag{2.19}$$

Expressing it with series:

$$X_n = \sum_{k=0}^{\frac{N-1}{2}} x_{2k} \cdot W(2 \cdot k) + W(1) \cdot \sum_{k=0}^{\frac{N-1}{2}} x_{2k+1} \cdot W(2 \cdot k) \tag{2.20}$$

The divide and conquer approach can be repeated for $log(N)$ times and so the final complexity of the algorithm is $log(N) \cdot N$.

The Cooley-Tukey algorithm can be performed with *Decimation in Time* (DIT) and *Decimation in Frequency* (DIF). For both decimation types the total number of computations remains the same. The main difference is on the order of the output samples that in case of the DIF doesn't follow the natural order but proper permutations must happen to produce the bit-reversed ordered data.

Depending on the radix selected, *radix-2* or *radix-4*, two basis elements exist: the *two-points Butterfly* (Equation 2.21) and the *four-points Butterfly*(Equation 2.22).

$$\begin{aligned} X[1] &= x[0] + x[1] \\ X[0] &= x[0] + W[1] * x[1] \end{aligned} \tag{2.21}$$

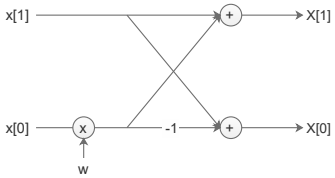

Figure 2.6: Two points Butterfly

$$\begin{aligned} X[0] &= x[0] + x[1] + x[2] + x[3] \\ X[1] &= (x[0] - jx[1] - x[2] + jx[3])W^1 \\ X[2] &= (x[0] - x[1] + x[2] - x[3])W^2 \\ X[3] &= (x[0] - jx[1] - x[2] + jx[3])W^3 \end{aligned} \tag{2.22}$$

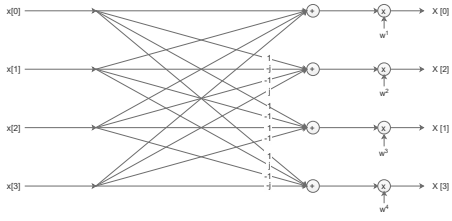

Figure 2.7: Four points Butterfly

2.3.2 Implementation Architectures

Several architectures to implement the FFT algorithms exist. The type of the input samples(timing and format) and the application type have to be considered to choose the appropriate architecture. The first subdivision concerns the application and can be distinguished into two different families:

- **Pipelined Architectures**
- **Burst I/O Architectures**

The *Pipelined* architecture is used when a continuous stream of input samples has to be evaluated, so it allows a continuous data processing.
The *Burst I/O* architecture instead stores into a memory the samples and evaluates the FFT on these separately. The main advantage of this architecture is the amount of resources needed to be implemented but on the other hand the processing time is longer.

The Pipelined architecture is implemented using several blocks in cascade mode. Each block, called processing unit, includes one butterfly unit and a memory bank used to store input data and intermediate values. This approach permits to send a stream of continuous input data and after a latency time unload the result.

The Burst I/O architecture can be used when the available resources could be a constraint. It uses only one butterfly unit, *radix-2* or *radix-4*, which computes the FFT algorithm with an iterative approach. When all the input samples are loaded, the input streaming has to be halted until the transform evaluation is finished. Then the FFT result can be unloaded.

A particular attention on the data format has to be considered. The samples can be expressed with fixed-point or floating-point representation. The floating point representation leads more computational resources than the fixed-point one and the case of a full floating point architecture could be very expensive, moreover the cases of *NaN*, ± *infinity* and Denormalized numbers have to be considered.
Always related to the data format, the fixed word length implies to use scaling factors or other methods. For example in case of a *radix-4* butterfly stage, in a DIT FFT algorithm, the result length can grow by a factor up to 3 bits. In the case of a *radix-2* butterfly stage the increasing factor is up to 2 bits. To handle this length growth several solutions can be used:

- Performing the transform using a no-scaling approach keeping all significant integer bits;

- Scaling each FFT transform stages using a pre-fixed scaling schedule approach;
- Exploiting the automatic scaling of the floating point representation.

A pre-fixed scaling schedule approach scales by a power of two each stage of the algorithm. The scaling factor exponent has to be considered such that no overflows appear in the Fourier transform. The scaling operation leads to use also a rounding operation. Several rounding approaches exist in literature but the ones with a zero bias value are preferred: for example the rounding to the nearest even.

Chapter 3

Software Defined Radio: USRP

3.1 Introduction

In the first part of this chapter will be touched what is *Software Define Radio*, which are the main advantages and the typical applications.
Instead the second part describes about the SDR device used for the purpose of this thesis, the *USRP-N210* device designed by *Ettus Research*. To develop the spectrum scanning system a first study of the device has been done to know which approach to use and which resources are available on the USRP.

3.2 Software define radio

Different definitions can be found to describe Software Defined Radio, also known as SDR. The SDR Forum working in collaboration with the Institute of Electrical and Electronic Engineers (*IEEE*) established a definition of SDR simply as:

"Radio in which some or all of the physical layer functions are software defined"

Traditional hardware based radio devices limit cross-functionality and can only be modified through physical intervention. By contrast, SDR technology provides an efficient and comparatively inexpensive solution, allowing multi-mode, multi-band and/or multi-functional wireless devices that can be enhanced using software upgrades.

SDR includes a set of hardware and software technologies where some or all the radio functions can be modified by software or firmware operating inside the

programmable part of the system. These devices include digital signal processors (*DSP*), field programmable gate arrays (*FPGA*), programmable System on Chip(*SoC*) or other programmable processors for specific applications.

A simple structure of a SDR receiver is shown in the Figure 3.1.

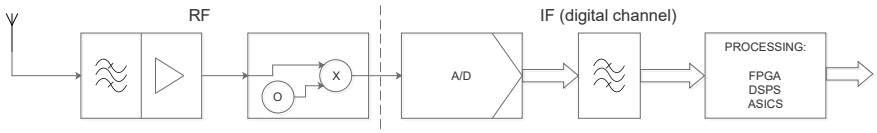

Figure 3.1: Simple structure of an SDR receiver

The Figure 3.1 shows how the receiving chain can be structured. The *RF* signal, coming from the antenna, is filtered by a wide band filter and then amplified with a LNA (*Low Noise Amplifier*). This amplifier usually is followed by an additional *Programmable Gain Amplifier* (PGA), which amplifies the received signal with a gain settled by software.
The amplified signal is shifted from the RF to IF band multiplying it by a local oscillator. The hardware used for this purpose is called *mixer*. The local oscillator signals usually are synthesised by a *Phase Lock Loop*(PLL) paired with a *Voltage Control Oscillator*(VCO). In this way the user has the availability to change the local oscillator frequency by software.
The analog to digital converter(ADC) converts the IF signal to a digital value, that can be filtered again and processed using some specific hardware.
Usually the RF signal is brought to baseband through the local oscillator and so it can be sampled by lower frequency, but there are also some specific applications where the amplified signal can be sampled directly by the AD without any tune stage. A lot of different architectures exist but the common type brings the signal into baseband frequency and splits it in: the *Quadrature* and *In-phase* components. A similar architecture can be derived from the transmitting chain where the AD converter is changed with a DA converter and the signal flows toward the antenna.

The flexibility of SDR lies in a programmable baseband, IF or RF section. Because the system can operate at multiple frequency bands, a set of dedicated RFs or tunable wideband RF are used (*wide band filter, LNA, mixer*).

3.3 USRP - Universal Software Radio Peripheral

In the previous section 3.2, the definition of SDR, its advantages, and the increasing usage of it have been described. For the project purpose, the *USRP-N210* device has been used.

The acronym USRP stands for *Universal Software Radio Peripheral*, this product was designed by Matt Ettus for RF applications from DC up to 6GHz, including the availability of multiple antenna system (MIMO). The platform is very flexible and can be used to implement real time applications. Most USRPs are connected to a host computer through a high-speed link, for example ethernet communication. The software-based host controls the USRP hardware to receive and transmit data using an open source UHD driver. Moreover, also other types of USRP exist where the host system functionalities are integrated in an embedded processor that allows the device to work in a stand-alone fashion.
One of the main advantages of this platform and the USRP family is the design for accessibility due to the use of an open source hardware. The following sub-sections have the aim to explain how the device work, how is structured and how can be used.

3.3.1 Hardware

The USRP's motherboard provides the following subsystems:

- *Base band processing*:
  - clock generation and synchronization;
  - FPGA;
  - ADCs;
  - DACs;
  - host processor interface;
  - power regulation.
- *RF analog processing*:
  - up/down conversion;
  - filtering;
  - other signal conditioning.

The *RF analog processing* is a modular front-end called *daughter board* which can be changed based-on the application. This modularity permits, as mentioned in the introduction, to handle application from DC to 6GHz.
The Figure 3.2 shows how the different parts listed before are connected.

The *USRP N210* specification are reported in the Table 3.1.

FPGA	Xilinx Spartan 3A-DSP3400
ADC	14-bits 100 Ms/s
DAC	16-bits 400 Ms/s
Connectivity	Gigabit Ethernet Interface
Optional	GPSDO module
	Fully Coherent MIMO capability

Table 3.1: USRP N210 device specification

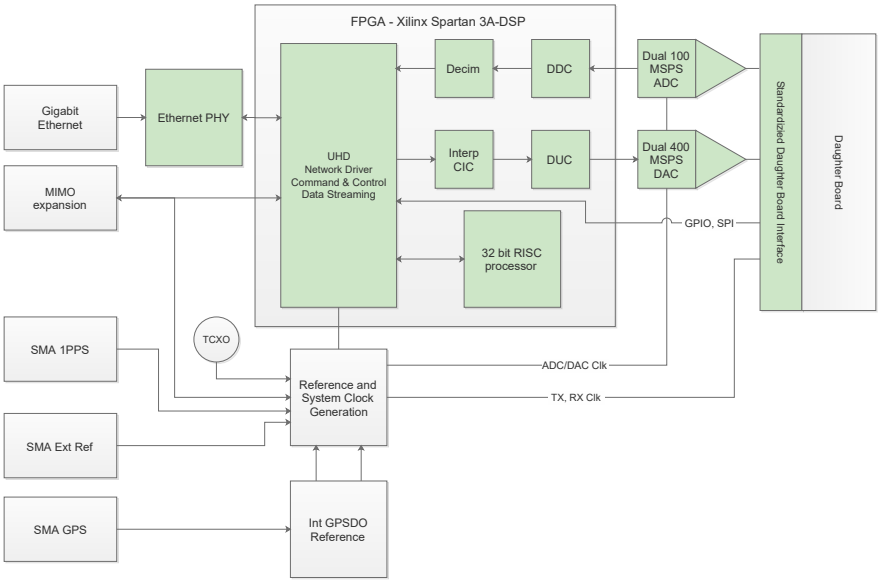

Figure 3.2: USRP N210 Block diagram from [1]

Analysing the receiver chain, the USRP down-converts the RF signal to base band, this is sent to the Host over a high-speed Ethernet link. While in the transmitting chain the USRP up-convert baseband samples, from the host system, to RF signal to be transmitted.

Several RF analog parameter inside the daughterboard are controlled by the software: local oscillator frequency, gain of the PGA and cut-off frequency of low pass filter.

Thanks to the open source issue of the device, the internal FPGA could be reprogrammed using the Xilinx ISE Design Suite. This allows in different applications to reduce the intensive workload of the host system.

3.3.2 Data Flow in USRP system

Figure 3.3 shows how the data received by the antenna is processed until to the host program. A similar analysis can be done for the transmitting sequence.

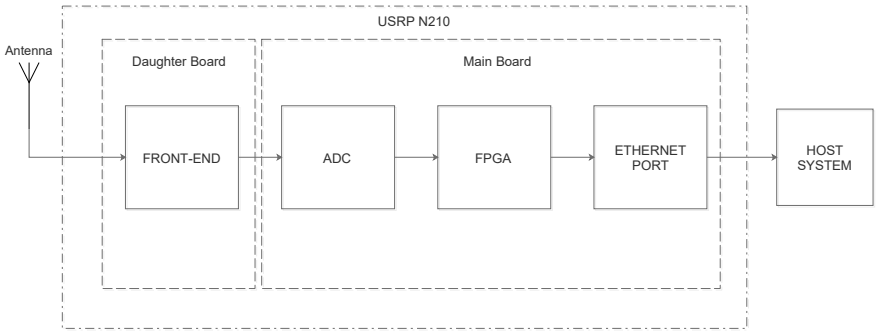

Figure 3.3: Data flow USRP host

Starting from the beginning of the chain each block will be discussed.

RF Front-end

The RF Front-end, in many cases, includes: the mixers, filters, oscillators and amplifiers lead to translate a signal from the RF domain to the complex baseband or IF signals. Based on the daughterboard type the translated signal can be complex or real. For example, the WBX daughterboard translates the RF signal to a complex extracting the quadrature and in-phase components. Instead the Basic RX shifts only the frequency components and treats the signal as a real one imposing the imaginary part equal to 0. Thanks to the flexibility feature of USRP, several parameters such as: *RF central frequency*, *gain of the PGA* can be changed by software intervention.

ADC

The baseband signals are then sampled by ADCs, one ADC for each component (I/Q). The ADC as mentioned in the Table 3.1 performs 100 MSps with the resolution of 14 bits. The digital samples are clocked into the FPGA.

FPGA DDC chain

The USRP N210 includes the Xilinx FPGA Spartan 3A-DSP3400, configured as a System on Chip (*SOC*) where different Intellectual Property (IP) cores of the Ethernet MAC, UART, I2C, Interrupt controller, SPI ecc. (Figure 3.5) are connected by the Wishbone bus. On this bus, a 32-bit soft-core processor ZPU acts as master while the other IP blocks are slaves.

The deafault stock FPGA image provides digital-down conversion and other functionality, such as a fine-frequency tuning and several filters for decimation (Figure 3.4).

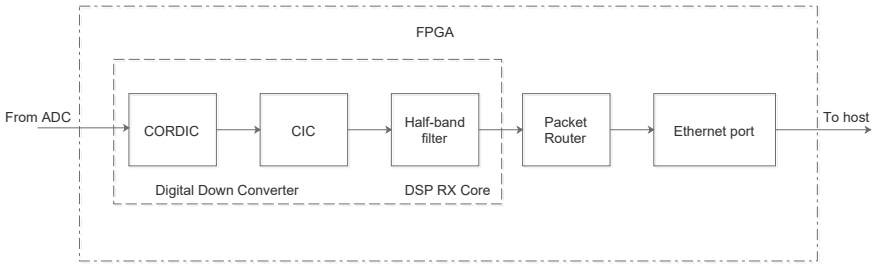

Figure 3.4: FPGA functionalities

The *CORDIC* unit as showed in Figure 3.4 is used for the fine frequency tuning. It should be thought as a DCO and Quadrature Mixer combined. The daughter board usually has a nominal useful bandwidth. The two half band FIR plus the CIC (Cascade Integrator Comb Filter) filter create a low pass filter with programmable decimation (Centred on DC). The CORDIC unit allows the nominal bandwidth to be arbitrarily repositioned with respect the DC of the downstream low-pass filter. So, in combination with the programmed decimation can be selected a narrower channel, that means a high precision tuning.
The *Packet Router* unit is composed by the VITA Radio Transport core that packs the samples decimated.

The packets are then sent to the Ethernet MAC to be transported to the host system.

3.3.3 FPGA resources

Compiling with Xilinx ISE suite the open source firmware the resources available inside the FPGA are found. The Table 3.2 shows percentage of resource usage.

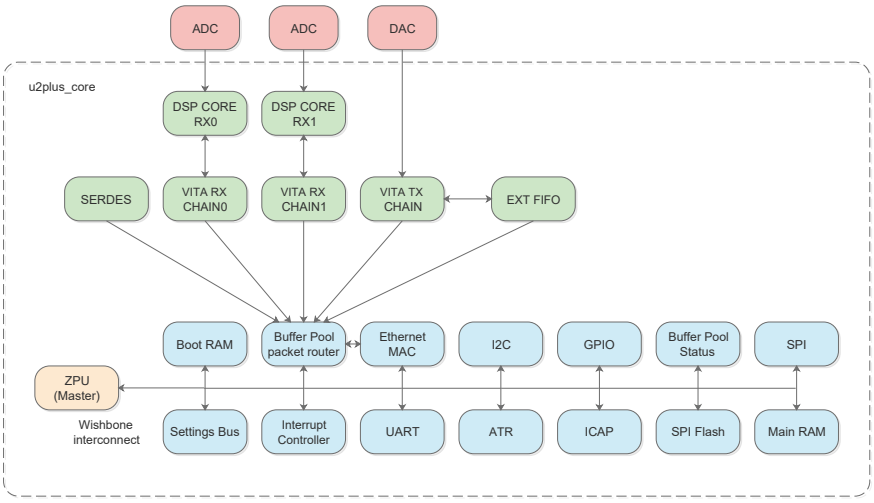

Figure 3.5: SOC in FPGA's USRP [2]

Design Summary Report				
Number of slice FFs	20,302	out of	47,744	42%
Number of 4 input LUTs	31,344	out of	47,744	42%
Number of BUFGMUXs	6	out of	24	25%
Number of DCMs	1	out of	8	12%
Number of DSP48As	31	out of	126	24%
Number of ICAPs	1	out of	1	100%
Number of RAMB16BWERs	41	out of	126	32%
Number of Slices	19682	out of	23872	82%
Number of SLICEMs	2360	out of	11936	19%

Table 3.2: FPGA resources available

FPGA custom part

The source code of the FPGA in the USRP is an open source code, in particular Ettus Research allows to modify it in predefined points simply adding a signal processing module in cascade mode. There are different points where the custom modules can be added(Figure 3.6):

- between the RF front-end and DDC (1);
- between the DDC and baseband processing (3);

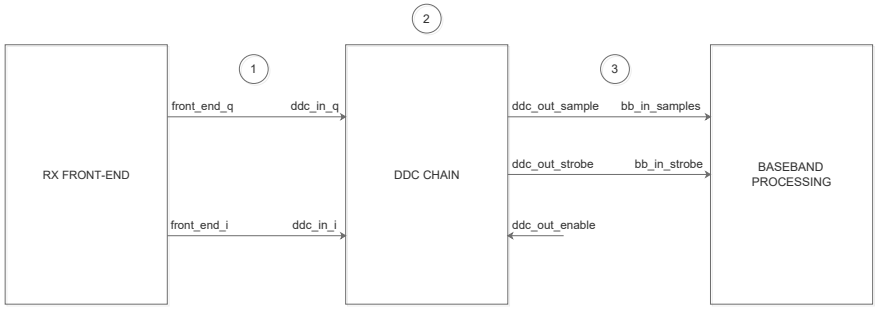

Figure 3.6: FPGA custom points

- replacing the DDC unit(2).

Using the Xilinx ISE 12.2 suite the modified source code can be synthesized and flashed into the FPGA.

3.3.4 Host Software

The host based software communicates with the USRP through a library called UHD(*Universal Hardware Library*). The C++ library provides the ability to control the device, abstracting away the underlying communication protocol. The host sends control messages, provides sources and sinks with transparent type conversion mechanism. The USRP device is completely compatible with: LabVIEW and Gnu Radio (a free and open-source Toolkit for Software Radio).
The UHD driver documentation for the classes and functions is almost complete on official web-site [11]. However a community based on mailing list exists where people can ask questions and the answers are available for all the users.

3.3.5 Practical Aspects

A lot of USRP based applications use a fully software approach to implement the signal processing algorithm. Users like to use the Gnu Radio tool, to implement software applications, for the user-friendly interface and for the easiness to process base-band signals. The software applications created are executed by the host CPU on a conventional operating system.
One major bottleneck of USRP is the limited CPU computation power, which does not allow to process RF signals at high sample rates continuously. The CPU workload of the host computer rapidly becomes intolerable when it processes a continuous sample stream.
Secondly the streaming performance depend also on the interface between USRP

and host computer. The USRP N210 should reach a maximum theoretical throughput of 25MS/s with a 16 bit I/Q format through a Gigabit Ethernet link. It is needed to underline that the actual throughput value depends on several factors: the processing capability of the host computer, the complexity of the application and other factors, so becomes hard to guarantee the theoretical value.
The typical idea followed to cope the limited CPU computation power is to embed part of the algorithm inside the customable FPGA side. Thanks to this approach the host CPU workload can be relived.

Some applications need to send several commands with a precise execution time, such as a spectrum scanner where the RF central frequency, of the daughterboard, needs to be re-tuned with specific period of time. When this time is very short, like *1ms*, it is difficult to guarantee that the host CPU will be able to send these commands every millisecond, especially when running software on conventional operating system. In contrast to a real-time system, when the software is executed by a conventional OS, it can suffer from OS related lags.
To solve this limitation, Ettus Research has introduced the concept of time commands. This allows to pair the command with a specific execution time. The advantage lies on the timing precision which is linked to the FPGA clock. Moreover, the USRP can handle up to 16 timed commands in advance and execute them in the future.

Chapter 4

Automatic Spectrum Scanning System

4.1 Introduction

The Chapter 4 has the aim to explain how the system is structured, and how works at high level. Two types of architectures have been implemented: a fixed and an adaptive threshold architecture. The system, in both of the cases, analyses the spectrum around a certain central frequency, f_c, to understand if there are signals in that frequency range. Changing the f_c of the receiver other frequencies can be analysed.

4.2 System Design

The Figure 4.1 shows the high level architecture of the *Automatic Spectrum System*. The system is composed by several parts: the spectrum scanning and the signal detection are performed inside the *USRP FPGA*; while the scanning coordination and the data logging are performed by the *Host CPU*.

The *front-end* part of the USRP samples data with sample rate frequency (f_s) at a certain central frequency f_c and using an appropriate ADC converts the analog signal into a digital one. The sampling operation, as explained in the Section 3.3, is performed into the base band thanks to a first stage composed by a mixer. The digital data is processed by the synthesized units, flashed into the FPGA. The processed samples are sent to the host CPU by an Ethernet communication, which handles the results storing them into a database.

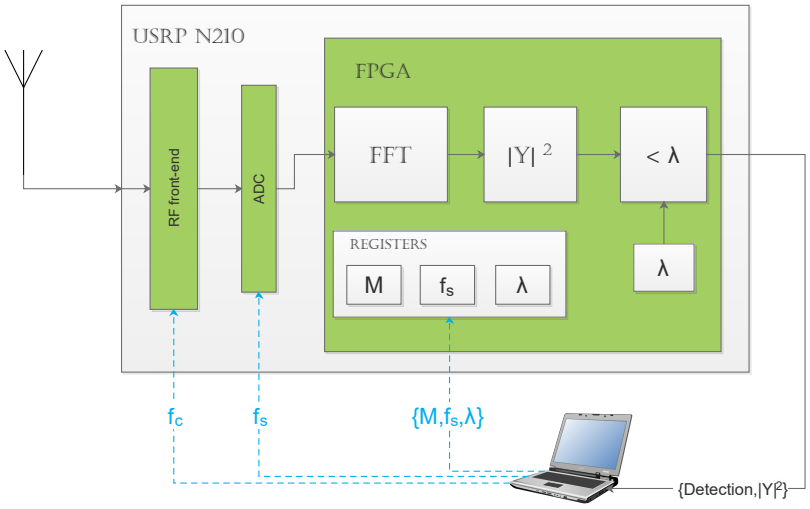

Figure 4.1: High level Architecture of Spectrum Scanning System

4.2.1 FPGA

By exploiting the features of the USRP device, explained in the section 3.3, a custom module, that includes several processing units, has been embedded into the original FPGA configuration.
As the Figure 4.1 shows, the FPGA architecture is composed by 3 main blocks.
The FFT of the digital received signal is performed. The FFT size is kept constant to 1024 so the frequency bin resolution is controlled by changing the USRP sample rate. Then Square Magnitude (SQM) of the complex FFT result is evaluated. Finally the comparison with the threshold (λ) is performed into the energy detection unit. The comparison with λ is evaluated over a sub-band of frequency bins (indicated by the register M), which can be configured from 1 to 1024 with values power of 2.
For the *Automatic Threshold* architecture some additional units are used to evaluate λ based on the overall SNR.
The signal, that is sent to the software, is defined over 32 bits; the first 31 bits represent the SQM(normalized in case of the adaptive architecture) of the FFT, and the last bit represents the detection for each frequency bin specifying if a signal is detected or not for the respective frequency bin.
Working inside the FPGA, all the operations are performed at frequency clock speed ($100MHz$), this allows the *Host CPU* to run a lighter program and so avoid lags that occur with conventional operating systems. The Chapter 5 explains in details each module implemented for both the architectures.

4.2.2 Software

The *Host CPU* controls and coordinates the *fc* retuning and all commands to receive the samples from the USRP. The received data is reordered and logged into a file. The main task is to retune the *fc* for covering a wide bandwidth of the spectrum. Each FFT can analyse a bandwidth equal to the sample rate frequency and so retuning the fc of the USRP receiver, a wideband of the spectrum can be analysed. The fc retuning operation, in the automatic mode, takes about $500\mu s$; so each retuning can be done approximately every 1ms to have more margin.
Moreover the *Host CPU* sends also commands to settle some user registers, the sample rate and the receive samples command.
The user registers that have to be settled are: the number of bins in the sub bands(M), the f_s value for re-adapt the samples to the sample rate and the level of the threshold λ. In the adaptive threshold architecture some more additional registers are used to remove the DC component caused by the RF front-end part.
More control commands are sent consequently exploiting the *time commands* feature of the *UHD driver*. Thanks to an input FIFO inside the USRP, the *Host CPU* can send up to 16 commands in advance with a precise future executing time. This allows to deal with inevitable software lags that occur on the *Host CPU* when using conventional operating systems.
The Chapter 6 explains in details the software implementation.

4.3 Spectrum Sensing Algorithm

The spectrum sensing algorithm implemented in the *Automatic Spectrum Scanning System* is based on the Energy Detection algorithm described in the Section 2.1.1. This sensing technique permits to detect signals without any priori info of the signal received and with low computational cost.
The idea is to perform the Fourier transform of 1024 samples and divide the frequency components into l sub-bands composed by M bins. The energy of each sub-band is compared with a threshold λ.
A deep study on the selection of the λ value has been conducted. As mentioned in the Section 2.2 many studies have been performed by researchers about the threshold selection, and in case of an adaptive threshold approach a measure of the SNR is necessary.
Several algorithms have been simulated in Matlab to find the best method for the system purpose. The SNR evaluation for example can be performed directly on the FFT result using an image processing approach based on Rank Order Filters (ROF)[12].

Using the Matlab simulation with floating point format, the algorithm detects precisely the signals estimating with good precision the SNR using the ROF method. Moreover the ROF has a good structure to be implemented in hardware. After the FFT module has been implemented and simulated, some more aspects on the threshold selection has to be considered. The FFT hardware module used, it has a resolution that does not permit measure the noise power directly on the FFT result. This fact is due to the fixed point data format and the scaled approach to avoid overflow on the FFT algorithm. So another way has been followed and it is to use a threshold based on the CFAR that needs only the noise power evaluated in the time domain directly on the Host CPU.

4.3.1 Fixed Threshold

The Fixed Threshold architecture evaluates the energy adding the SQM bin values on the sub-band selected, then the result is compared with the threshold adapted to the frequency domain. Following the sensing algorithm is shown as a pseudo Matlab code.

```
Y=fft(samples_t_domain , N);
SQM = abs(Y).^2;
threshold = threshold_evaluation(noise_power,N,M);
for(k=0 : 1 : m_window)
        signal_energy_window = sum(SQM([k*M+1 : k*M+M]));
        if(threshold < signal_energy_window)
                DETECTION=true;
        else
                DETECTION=false;
        end if
end for
```

The threshold selected as mentioned before is based on the constant false alarm probability (Equation 2.14). Because the expression is given on the time domain, recalling the Parseval's theorem [1] the version on the frequency domain is found simply multiplying it by the size of the FFT (N = 1024).
The expression to derive λ becomes the following

$$\lambda^* = (\sigma_w^2(1 + \frac{Q^{-1}(P_f)}{\sqrt{N/2}}) \cdot \frac{M}{N}) \cdot N) + \epsilon \tag{4.1}$$

[1] The Parseval's Theorem is often used to describe the unitarity of any Fourier transform, especially in physics. For the discrete Fourier transfor (DFT) the relation is: $\sum_{n=0}^{N-1} |x[n]|^2 = \frac{1}{N}\sum_{k=0}^{N-1} |X[k]|^2$ where $X[k]$ is the DFT of $x[n]$, both of length N [13].

Where σ_s is the noise power, P_f the probability of false alarm, M number of sub-band bins and N size of the FFT. The ϵ value is a correction factor due to the operations in fixed point format and the value is fixed to 10.

4.3.2 Adaptive Threshold

The idea of the Adaptive Threshold is to adapt a base threshold value with the mean energy of the samples received as explained in [14]. The normalized square magnitude of the spectrum, D factor, is evaluated dividing the SQM by the mean value of all FFT bins. Similar to the fixed one, then for each sub-band the sum of the bins is evaluated (D_{window}) and compared with the threshold. Before the comparison, λ value is adapted adding a portion (α) of the D_{window}.

The expression of the D factor is the following (Equation 4.2):

$$D[k] = \frac{|X[k]|^2}{\sum_{n=0}^{N-1} |X[n]|^2/N} \tag{4.2}$$

For the case of one sub-band of M bins the expression becomes:

$$D_{window} = \frac{\sum_{k=n_w}^{n_w+M-1} |X[k]|^2}{\sum_{n=0}^{N-1} |X[n]|^2/N} \tag{4.3}$$

where n_w stands for the starting bin of the sub-band.

The final threshold compared with D_{window} is defined as the sum of the CFAR threshold, used for the fixed case (Equation 4.1), and a percentage of D_{window} (Equation 4.3):

$$\lambda = \lambda^* + \alpha \cdot D_{window} \tag{4.4}$$

Following a pseudo Matlab code of the adaptive algorithm explained before is shown.

```
Y=fft(samples_t_domain , N);
SQM = abs(Y).^2;
D = SQM ./ mean(SQM);
th_b = threshold_evaluation(noise_power,N,M);
for(k=0 : 1 : m_window)
        D_window = sum(D[k*M+1 : k*M+M]);
        th = th_b+ alpha * D_window;
        if(th < D_window)
                DETECTION=true;
        else
                DETECTION=false;
        end if
end for
```

As can be noticed, the adaptive algorithm recalls some division operations. To avoid the implementation and the usage of some division modules on the FPGA, the algorithm is simplified to use only divisions by power of 2.
The threshold value, in both cases, has to be stored inside a user register of the FPGA. Since the data representation is always with fixed point format, the threshold value is converted on 32 bit fixed point data format and rounded to the nearest upper power of 2. In the conversion operation also the scale factor applied on the FFT stages to avoid overflows has to be considered, as explained in section 2.3.
The threshold expression based on CFAR (λ^*) becomes (Equation 4.5) :

$$\lambda = \lambda^* \cdot s \;\rightarrow\; s = 2^{(n_{bit}-1-scale)} \tag{4.5}$$

The result of λ can be stored directly to the user register dedicated for the threshold.

Chapter 5

Hardware implementation of Energy Detection

5.1 Introduction

The FPGA modules implemented to develop the algorithms explained previously (Chapter 4) are described in the following chapter. All the units have been described in Verilog and synthesized with the Xilinx ISE 12.2 suite. The approach followed is to create several modules with different scopes which can be put in cascade mode. In this way if some other units are needed, can be added simply inside the chain.
The Figure 5.1 shows the top view of the modules implemented. Each module has two input and output signals, one for the data and one that indicates if the data is valid.

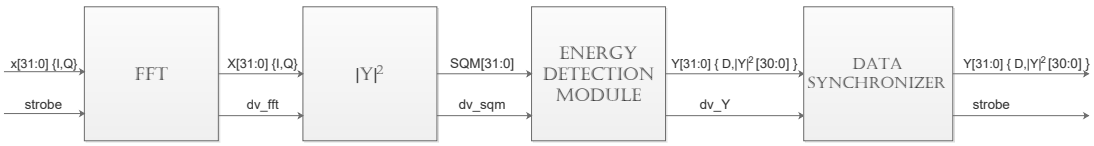

Figure 5.1: Top view of FPGA custom module

The FPGA architecture is divided in four main modules:

- FFT unit;
- SQM (Square Magnitude) unit;
- Energy Detection unit;

- Data Synchronizer unit;

To implement the two different algorithms, fixed and adaptive threshold, only the Energy Detection module in the chain changes; two different modules has been developed.
The chapter is organized with one section to describe each module shown in the Figure 5.1. First the units shared between the two architectures, FFT and SQM, will be explained. Then two different sections will describe the detection unit. The last section is dedicated to the Data Synchronizer unit which re-adapts the samples to the sample rate.

5.2 Fast Fourier Transform Module

The first module that is described is the Fast Fourier Transform Module. After a study of different architectures and algorithms to evaluate the FFT (Section 2.3), the module has been implemented.
The data coming from the USRP RF-frontend have fixed point format on 16 bit:

- 16 bit for the quadrature component in time domain (x_q);
- 16 bit for the in-phase component in time domain (x_i).

The FFT size (N) selected is 1024 and is fixed to this value; to refine the frequency resolution the f_s can be changed.
The suite used to synthesise the source modules is *Xilinx ISE 12.2* as suggested by the USRP manual. This tool provides IP cores for different applications. One IP core available is *LogiCORE IP Fast Fourier Transform v7.1* [15].
This IP core provides four different FFT architectures:

- Pipelined, Streaming I\O;
- Radix-4, Burst I\O;
- Radix-2, Burst I\O;
- Radix-2 Lite, Burst I\O.

Each architecture provides two different data format: floating point and fixed point representation. Moreover there is the possibility for all the architectures to use a convergent rounding to the data after the butterfly stage.
For the fixed point representation is present an input dedicated for the scale factor to be applied at each stage. The scale factor can be applied from the input port *SCALE_SCH* which has a pair of bits for each stage of the algorithm as following

[... N3 N2 N1 N0]. For the Radix-2 approach there are $\log_2 N$ stages, so for 1024 points are defined 10 stages and the bus length is of 20 bits. The suggested value, looking at the chapter dedicated to the scaling schedule in [15], to avoid overflow is *[01 01 01 01 01 01 01 01 01 10]*.
To be sure that the scaling schedule is appropriate, *Xilinx* provides a C model of the IP core to simulate it and look if some overflows occur. The scale factor mentioned before has been simulated with the C model and it does not give any overflows.

Two details have been taken into account among the different architectures: the resource usage and the processing time. The Pipelined architecture is the fastest one but comparing the resources available for the FPGA custom part with the requested by the core, result that are not enough. For the resource constraint, the Radix-2 Burst I\O architecture has been chosen. The Table 5.1 shows the performances and resources usage for the Spartan-3A DSP family of the architecture selected.

Slices	909
LUTs	1397
FFs	1273
18k Block RAMs	3
XtremeDSP slices	3
Max clock frequency [MHz]	203
Latency Clock Cycles	7385

Table 5.1: LogiCORE IP FFT v7.1, Radix-2 Burst I\O, performances and resources

The Figure 5.2 shows the top view of the FFT unit. The module has been implemented using an hierarchical approach. The top level includes several units which are:

- two RAMs to store the samples in the time domain;
- one FFT IP core;
- one Control Unit to handle the timing of the signals.

The module is characterized by a pair of input and output ports: the data bus, for both ports, is paired with a signal that indicates if the value is available or not.

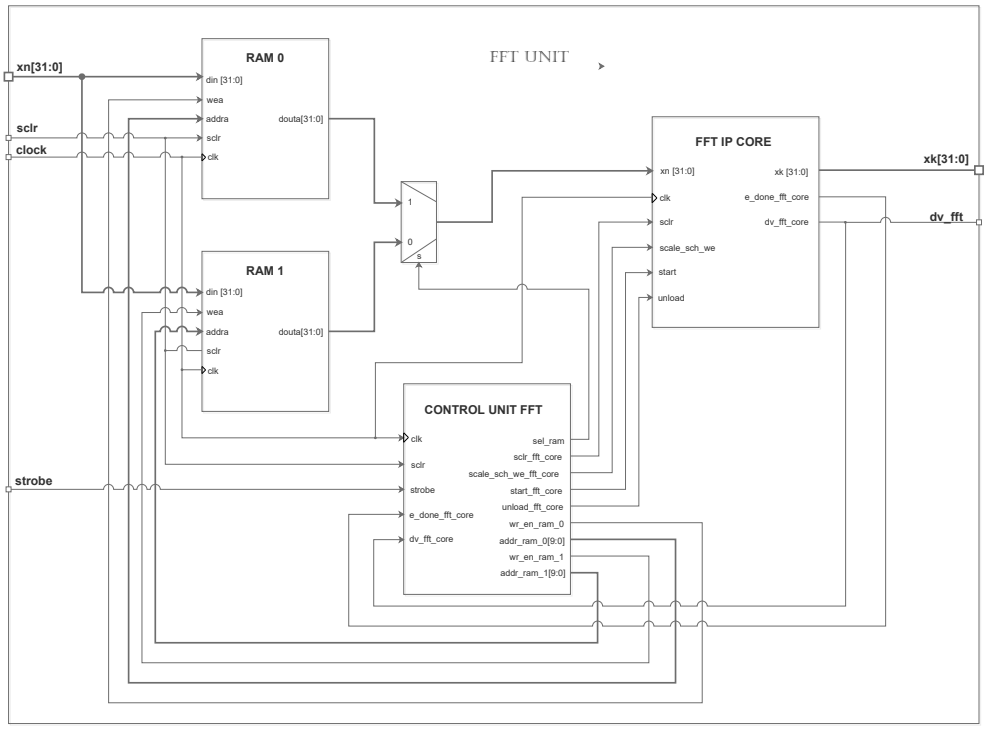

Figure 5.2: Top view of FFT module

Samples coming from the RF front-end has the behaviour as shown in the Figure 5.3. Every time when there is a new sample, the strobe signal changes from the binary value "0" to "1" for one clock cycle. The frequency of the strobe signal is equal to the sample rate (f_s).

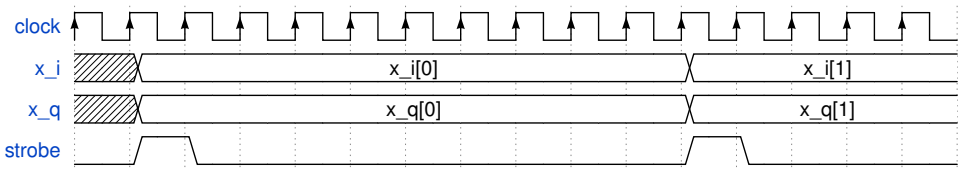

Figure 5.3: Timing diagram of time domain samples

For the N value, 1024 samples are needed to start the FFT IP core. Because the architecture selected is BURST I\O, two RAMs, of size 1024x32 bits, are used as buffer for samples. When one RAM is in write mode the other RAM is in read mode such that no sample will be lost. The idea is to write 1024 samples in one RAM and at the same time read the same amount of samples from the other RAM

sending them to the FFT IP core. When all the samples are read, the IP core can be run with the start signal.
Also the RAM verilog unit has been created through the *CORE Generator Graphical User Interface* from Xilinx ISE suite. The Control Unit FFT (Figure 5.4) generates timed signals to control: the read and write operations of RAMs and the FFT IP core. It is divided into two finite state machines (FSMs), which exchange signals among them:

- WR RAM Controller;
- FFT Controller;

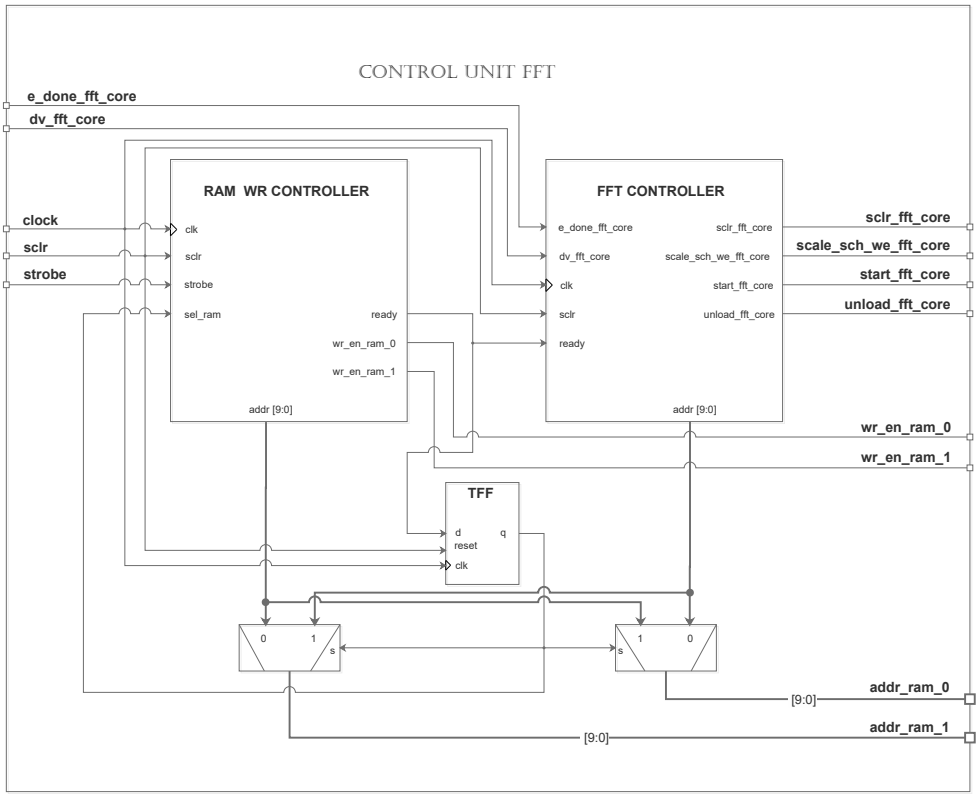

Figure 5.4: Control Unit FFT module

The WR RAM controller selects which RAM has to be written and generates a signal called *ready* used to inform the FFT Controller that samples are available to be transformed. Moreover this signal is an input of a T Flip Flop, the output of this FF selects the address for the respective RAM. When the binary value is "0": the RAM0 address comes from the RAM Controller and the the RAM1 address

comes from the FFT controller. In case the binary value is "1" the situation is complementary. An example of the switching mechanism is shown in the Figure 5.5.

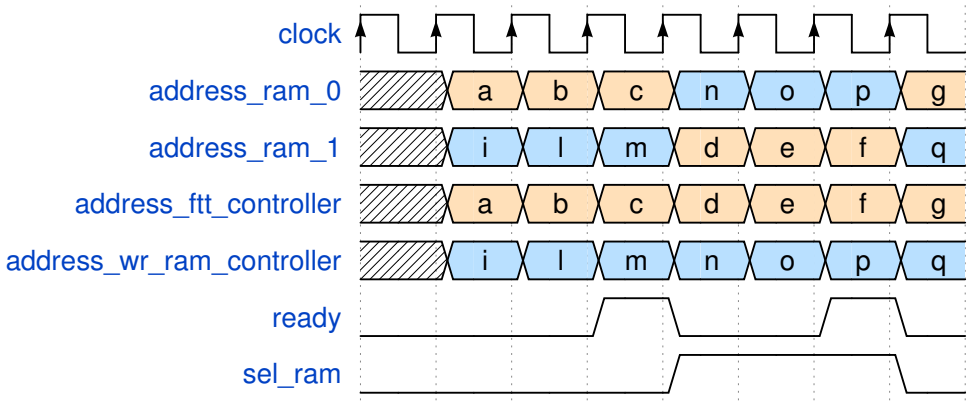

Figure 5.5: FFT Controller

5.2.1 WR RAM Controller

The WR RAM Controller is characterized by two inputs (*strobe* and *sel_ram*) and three outputs (*ready*, *wr_ram_0* and *wr_ram_1*). As can be noticed in the Figure 5.6 this unit is divided in two sub-units one that defines the FSM and the other the FSM's datapath.

The FSM's datapath is composed by a simple binary counter modulo 10 (Figure 5.7). This unit is enabled by the FSM when the strobe signal becomes high, and it counts the incoming samples. A terminal count, implemented with a simple AND gate, settled to 1023 is used to signal the switch to another RAM. When the output of the counter presents all ones then the terminal count transits from the binary value "0" to "1". Moreover the count value represents the address to point the memory. The binary counter is implemented with an IP Core available in the *Xilinx* suite.

The Control Unit instead represents the FSM description. A FSM with six states has been created: *RESET_STATE*, *WR_RAM_0*, *WR_RAM_1*, *COUNT_EN*, *RAM_FULL*, *WAIT_N_SAMPLE*. The Figure 5.8 shows the ASM chart of the state machine.
The reset signal has the higher priority respect all other signal. When is asserted the FSM changes is state, whatever it is, into the *RESET* state.
At the beginning the FSM remains in the RESET state waiting the signal *strobe* that transits from binary value "0" to "1". When a sample is available a test on the *select_ram* signal is performed. If it is "0" the left branch is selected otherwise the right one is followed. Based on the previous test, the next state will be WR_RAM_0

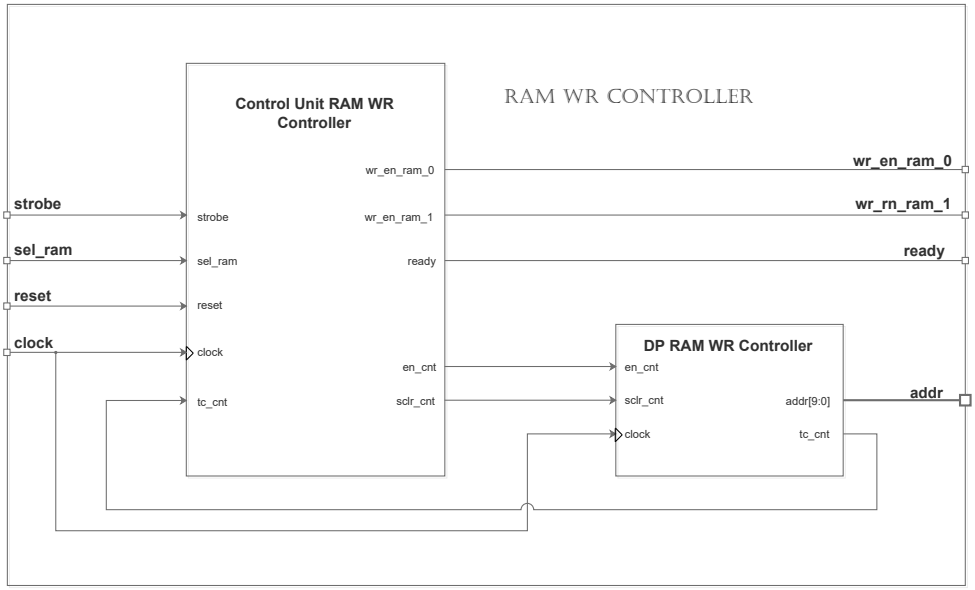

Figure 5.6: WR RAM Controller

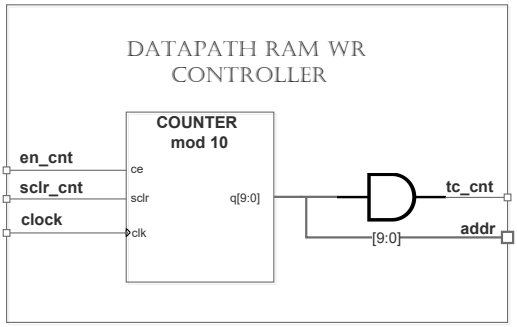

Figure 5.7: Datapath WR RAM Controller

or WR_RAM_1, in both of the cases the write control signal, of the RAM selected, is enabled.
When one sample is written, the binary counter is enabled for one clock cycle such that is incremented. If N samples have been counted, the *TC* is active high and so the *ready* command is settled to binary value "1" for one clock cycle. Instead if other samples have to be written the FSM remains in the WAIT_N_SAMPLES state waiting the next strobe signal. As mentioned previously, in the Control Unit FFT Module, the ready signal is used to inform the second FSM that data are available to be processed and at the same time through the T flip flop the selection of the

RAM is changed automatically.

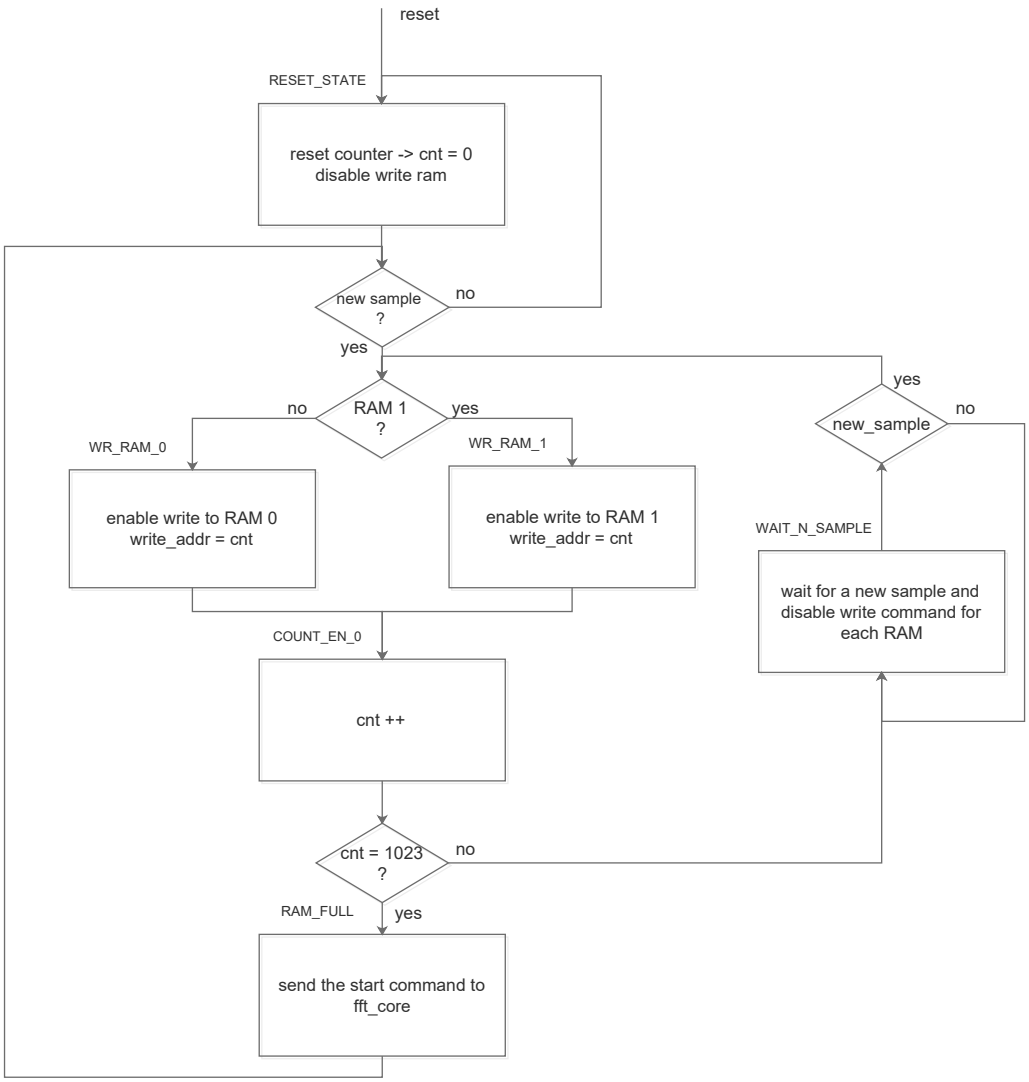

Figure 5.8: Control Unit WR RAM Controller ASM chart

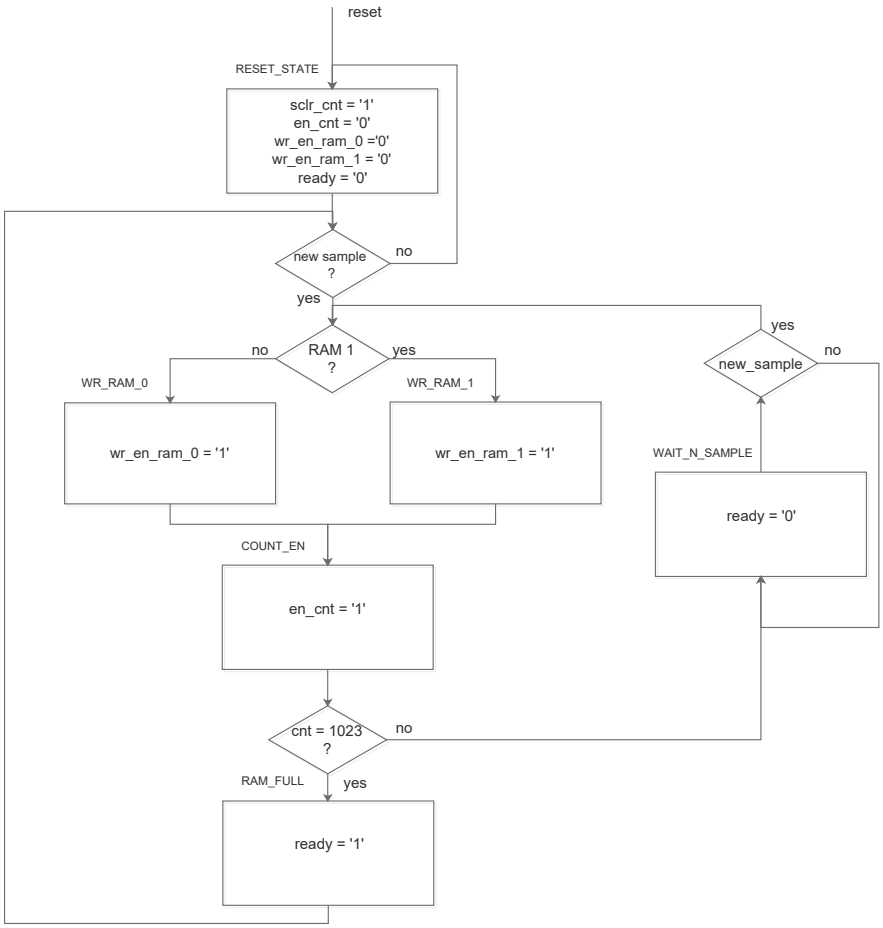

Figure 5.9: Control Unit WR RAM Controller Control ASM chart

5.2.2 FFT Controller

To generate the signals to control the FFT IP core a second FSM has been implemented. As the WR RAM Controller, the FFT Controller is divided in two blocks: the Control Unit and the datapath (Figure 5.10). The datapath of this FSM includes one counter as in the WR RAM Controller, the datapath figure is omitted because the similarity of Figure 5.8 . The aim of this unit is to count the samples and generate the address to point at each row of the RAM.

After a detailed analysis of the FFT IP core datasheet, a knowledge of the order and the timing of signals has been understood. The following signals have to be generated: *sclr_fft_core*, *scale_sch_we_fft_core*, *start_fft_core*, *unload_fft_core*, *en_cnt*

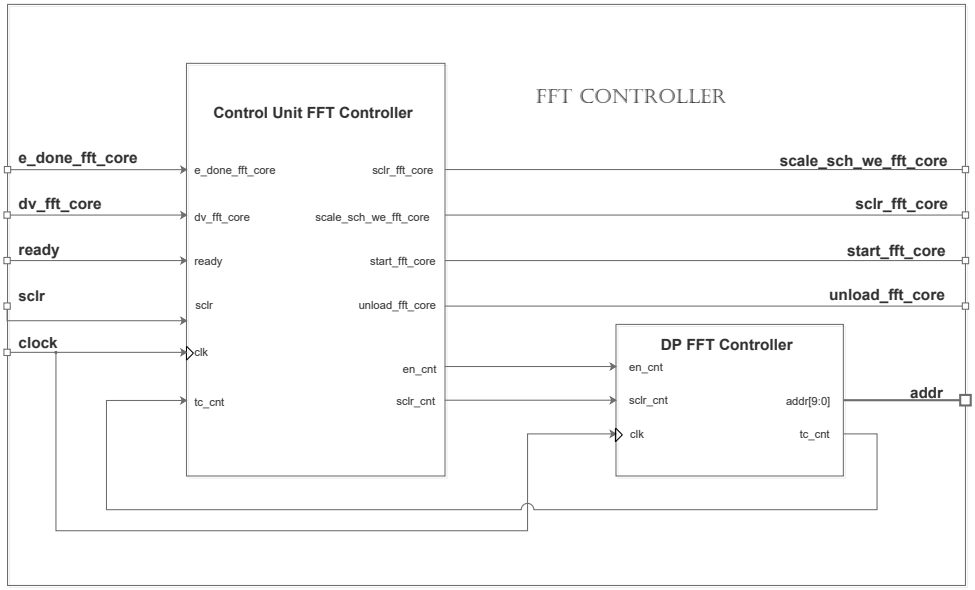

Figure 5.10: FFT Controller

and *sclr_cnt*. The input signals which have to be sensed are: *ready*, *TC*, *dv_fft_core*, *e_done_fft_core*.
To describe the FSM eight states has been used: *RESET_STATE*, *SET_SCALING*, *START_FFT*, *LOAD_FFT*, *WAITING_PROCESSING*, *UNLOAD_EN*, *WAITING_DV*, *UNLOAD*.
The Figure 5.11 shows the ASM chart of the FFT controller state machine, each steps, between states based on the inputs values, are described.
As on the previous FSM the *RESET_STATE* is the default state with the highest priority.
After the reset state the scale factor has been stored in the dedicated register in the IP Core through the *scale_sch_we_fft_core* control signal. Then the state machine remains in this state until the ready signal becomes the binary value "1".
When the *ready* signal is activated by the WR RAM Controller, the module knows that samples are available to be transformed. So the state changes to the *START_FFT* one.
The *start_fft_core* and *en_cnt* signals are enabled, from that moment 1024 samples one each clock cycles are sent to the IP core. The counter counts until 1023 starting from 0 and the output value represents the address to point at the RAM selected.
The FFT starts to transform the samples when all N samples have been sent to the core. From the starting time, a latency time has to be waited before the IP core output data is available. Through the value of *e_done_fft_core*, the end of the FFT

algorithm can be sensed one clock cycle before. As reported in Table 5.1 the latency clock cycles is of 7385.
When the *e_done_fft_core* signal is asserted, the new state becomes *UNLOAD* where the *unload_fft_core* signal is generated. After a latency time the output data of the FFT core starts to be available.
The state machine waits the end of all outputs data through the *dv_fft_core* signal. A new FFT is started only when the unload operation is ended and new set of samples are available.

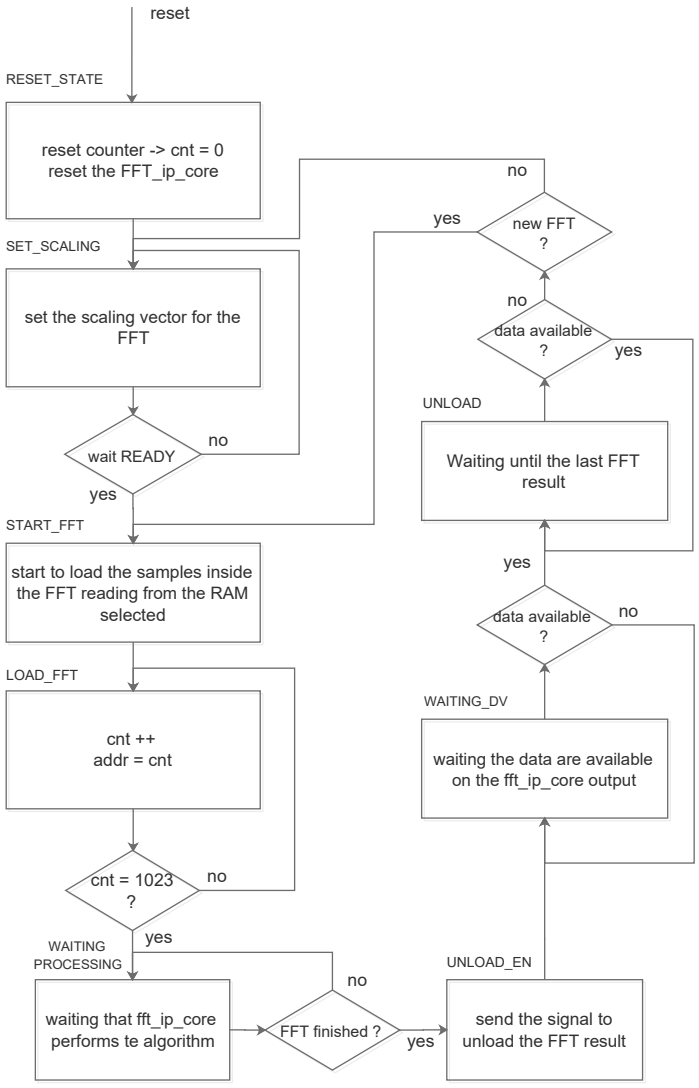

Figure 5.11: ASM chart FFT Controller

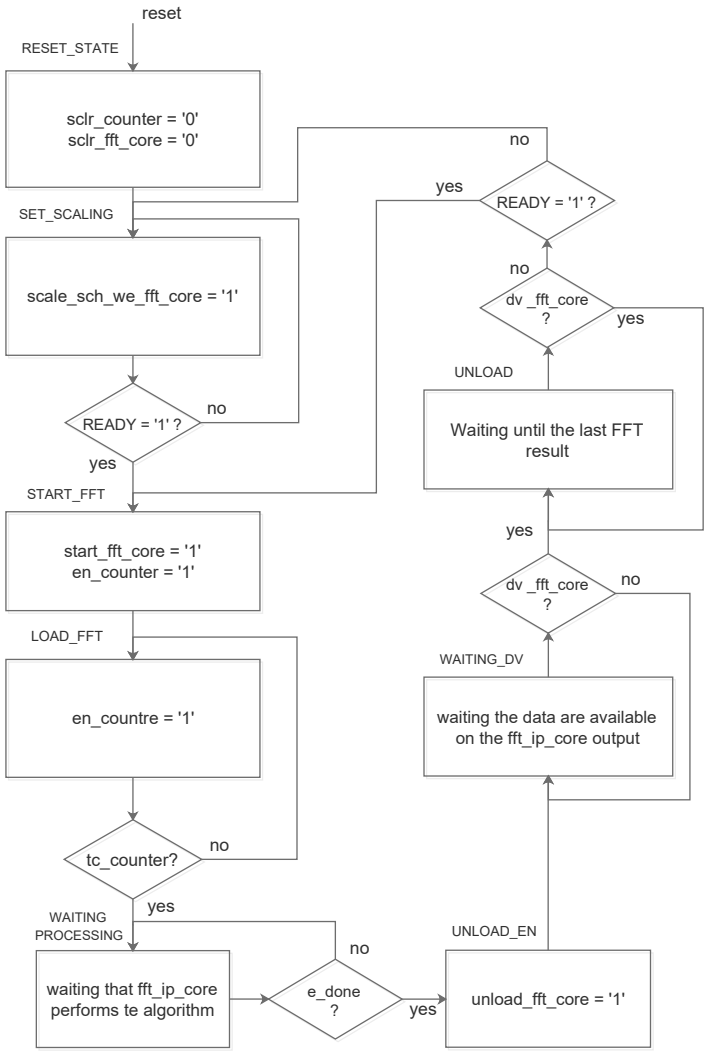

Figure 5.12: Control ASM Chart FFT Controller

5.3 Square Magnitude Module

The second shared module among the two architecture is the *Square Magintude Module* as mentioned in the Figure 5.1.
The magnitude of a complex number is defined as (Equation 5.1):

$$|Y| = \sqrt{Re[X]^2 + Im[X]^2} \tag{5.1}$$

Evaluating the square magnitude of a complex number the expression becomes(Equation

5.2):

$$|Y|^2 = Re[X]^2 + Im[X]^2 = Re[X] \cdot Re[x] + Im[X] \cdot Im[x] \tag{5.2}$$

As can be noticed the square root operator is not needed and the result can be evaluated with only multiplication and addition operations. The high level structure of this unit is shown in Figure 5.13. The output data from the FFT module is defined on 16 bits signed format, the multiplication stage moves the data parallelism to 32 bits. For the next addition stage the bus length is kept fixed to 32 bits. This because two signed binary data squared will be always positive and knowing that the data uses a fixed point representation with only the signed bit as integer part, 32 bits result enough without overflow.

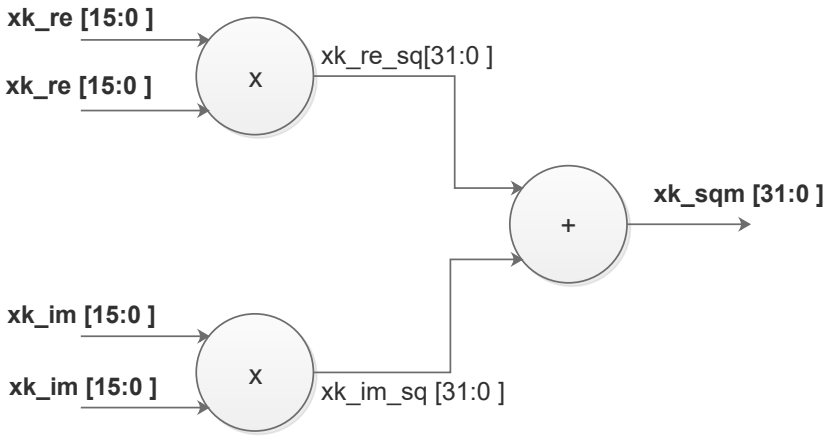

Figure 5.13: High Level architecture SQM unit

The Figure 5.14 represents the implemented module. The unit is characterized by five inputs:*xk_re*, *xk_im*, *dv_fft*, *clock*, *reset* and two outputs: *xk_sq_m* and *dv_sq_m*. Also for this scope IP cores from the *Xilinx ISE* suite hase been used.
The multiplier core is generated using one DSP48A unit available inside the *Spartan 3A-DSP 3400 FPGA*. The latency of this multiplier is of four clock cycles, for that reason the *dv_fft* is delayed of 4 cycles with 4 FFs. Two identical multipliers have been instantiated to handle the imaginary and real components.
The adder core is implemented also using the DSP48A unit and its latency is of 2 clock cycles.
Overall to have the correspondence between the bus data and the data available signal 6 FFs are needed. The Figure 5.15 shows the behaviour of the data available signal respect the data processing results.

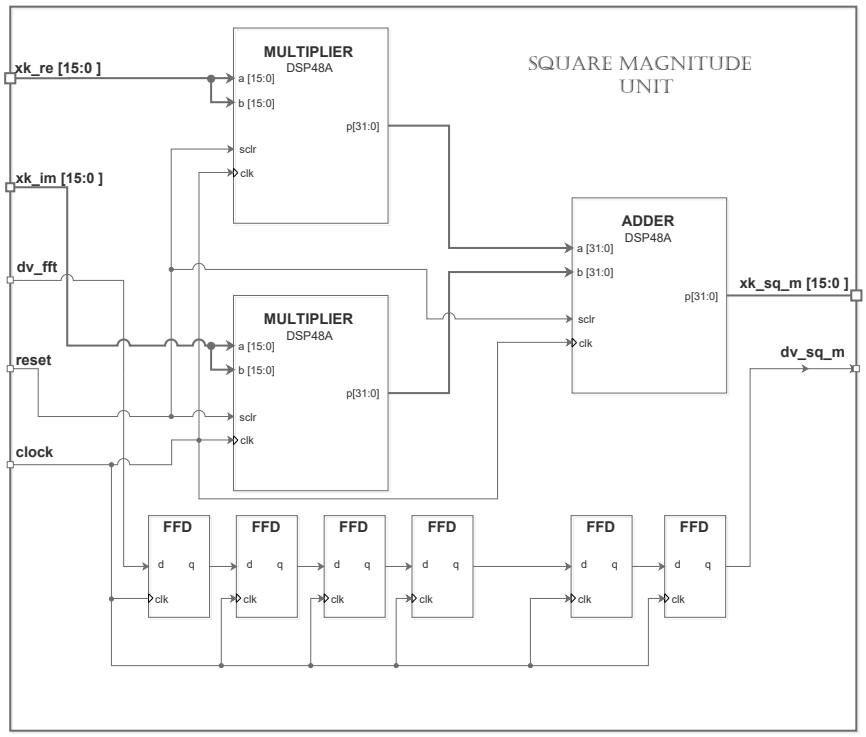

Figure 5.14: SQM implemented architecture

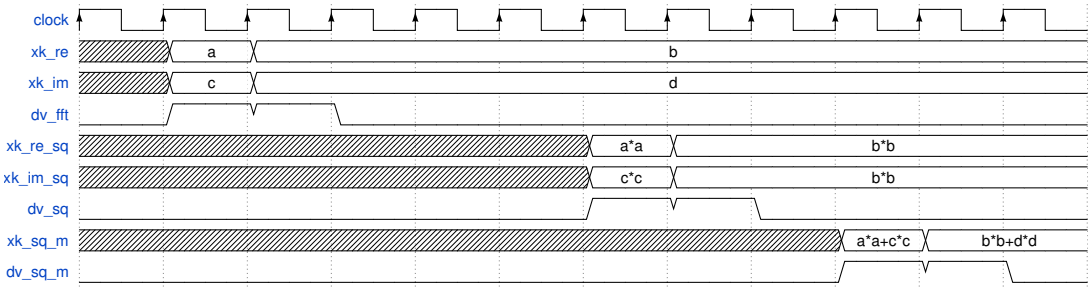

Figure 5.15: Timing diagram SQM Unit *dv* signal vs *data* bus

5.4 ED Module Fixed Threshold

The module, shown in Figure 5.16, represents the Energy Detection Unit for the fixed threshold algorithm explained in the section 4.3.1. The input ports of the unit are:

- *clock*;
- *reset*;

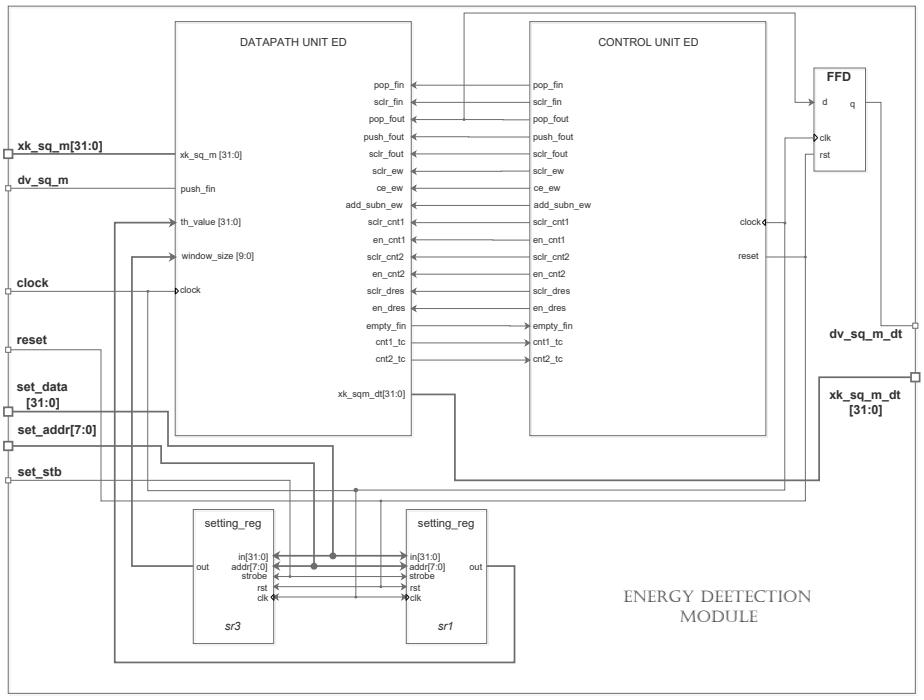

Figure 5.16: Energy Detector top level (*Fixed Threshold*)

- *xk_sq_m*, the *sqm* data of the spectrum analysed, defined on 32 bits;
- *dv_sq_m*, the signal for data available;
- *set_data*, a data bus of 32 bits for set values inside the user registers;
- *set_address*, an address bus of 8 bits to select the specific user register;
- *set_stb*, a strobe signal to inform the user register about the address.

The unit has instead just two output ports: *dv_sq_m_dt* and *xk_sq_m_dt*. This last one is defined on 32 bits and these are divided as following:

- MSB [31] bit represents the respective signal detection of the bin;
- other [30:0] bits represent the square magnitude spectrum value of the bin.

The module has been implemented with an hierarchical approach. The top level of the module is shown on the Figure 5.16. One user register is used to select the size of the sub-band (M) and the other one to settle the threshold (λ) value, evaluated with the Equation 4.1. The Control Unit ED is composed by two FSMs: one for the energy evaluation and comparison (*Control Unit Detection*) and the other one

for a packing operation of the result (*Control Unit Packing*).

The Datapath Unit ED receives control signals from the control unit and sends status signals that are:

- *empty_fin* active high when the input FIFO is empty;
- *cnt1_tc* active high for one clock cycle when the detection counter reaches the terminal count;
- *cnt2_tc* active high for one clock cycle when the repacking counter reaches the terminal count.

Figure 5.17 shows the Datapath Unit ED. Two FIFOs of the size 1024 buffer samples to be processed. The data coming from the SQM unit is a stream of 1024 samples each every clock cycle. Processing these samples with a pipelined structure in one clock cycle is very difficult and needs a lot of resources. The incoming data is stored in the first FIFO and then processed. The push operation is controlled by the *dv_sq_m* input signal.
Another FIFO is used to buffer the value of sub-band bins processed because these will be send to the *Host CPU* with the detection result. The push operation of this FIFO is controlled by the Control Unit Detection instead the pop operation is handled by the Control Unit Packing.

Exploiting as in the previous modules the IP core generator of *Xilinx* suite, one accumulator unit and the FIFOs have been generated.
The accumulator IP core issues one input port to select if the sample has to be added or subtracted. This feature can be exploited as a comparator operation. As reported in the algorithm (Section 4.3.1), the bins inside the sub-band has to be summed and then compared with the threshold. Comparison operation can be evaluated through a subtraction of the threshold value after the accumulation of all bins inside the sub-band. The sign bit of the result is the signal detection result; this value is stored inside a FF and then used to repack the sub-band bins value evaluated.
The counter labelled *counter detection* (Figure 5.17) is used to select the right number of bins in one sub-band. The terminal count value can be changed through the *window size* user register and is evaluated with a bitwise EX-NOR operation between the *window_size* and the *cnt1* values.
With the same approach the counter labelled *counter packing* stores the number of sub-band bins packed. In this case the terminal count TC is handled by the Control Unit Packing.

The two FSMs communicate among them, the Control Unit Detection when finishes to compare the threshold, with te signal *end_sig* inform the second one to

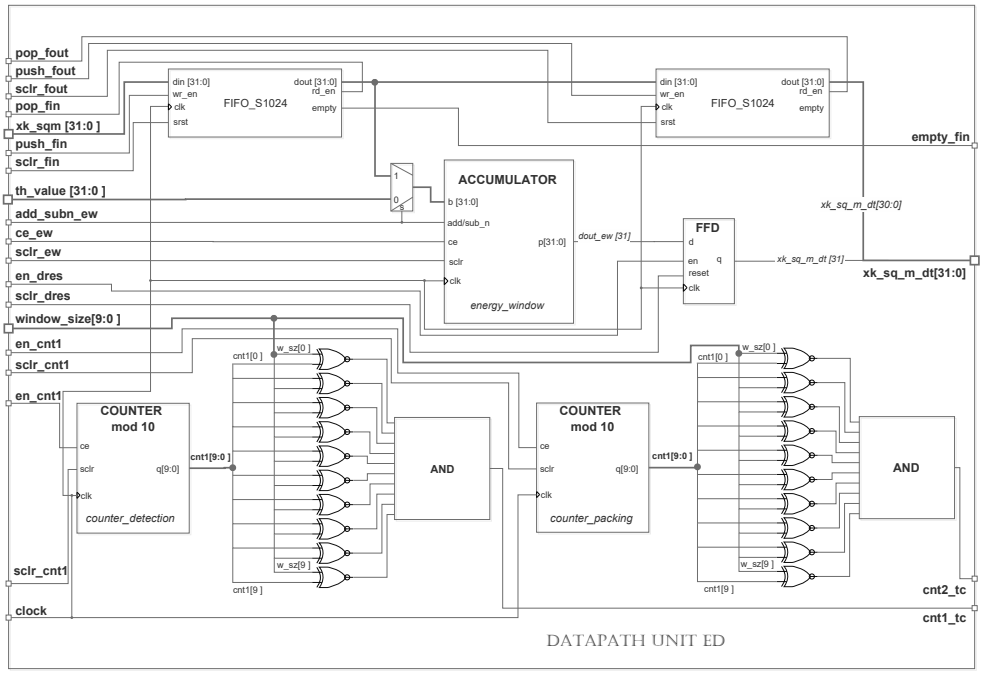

Figure 5.17: Datapath Energy Detector (*Fixed Threshold*)

start the packing of the result. Split the Control Unit ED into two FSMs allows to process and pack the samples concurrently and so save some clock cycles.

5.4.1 Control Unit Detection

The Control Unit Detection is a state machine with 9 states: *RESET*,*ACCUMULATE_1*, *ACCUMULATE_2*, *ACCUMULATE_3*, *COMPARE*, *WAIT_COMP*, *END_COMP* and *RESET_ACC*.
The aim of this FSM is to create a sequence of timed signals to control the Accumulator and so the comparison operation.
The *RESET* state has the higher priority respect all the other states. When the present state is *RESET* all the synchronous clear signals are enabled and so the FIFOs, the accumulator, the counter are reset.
The state machine, according to the ASM chart shown into Figure 5.20, remains into a wait state (*WAIT_DATA*) until one stream of samples from the SQM unit is available. The test condition of this situation is performed on the *empty_fin* status signal. When the first sample arrives, it is pushed into the FIFO, the *empty_fin* signals changes from the the binary value "0" to "1".

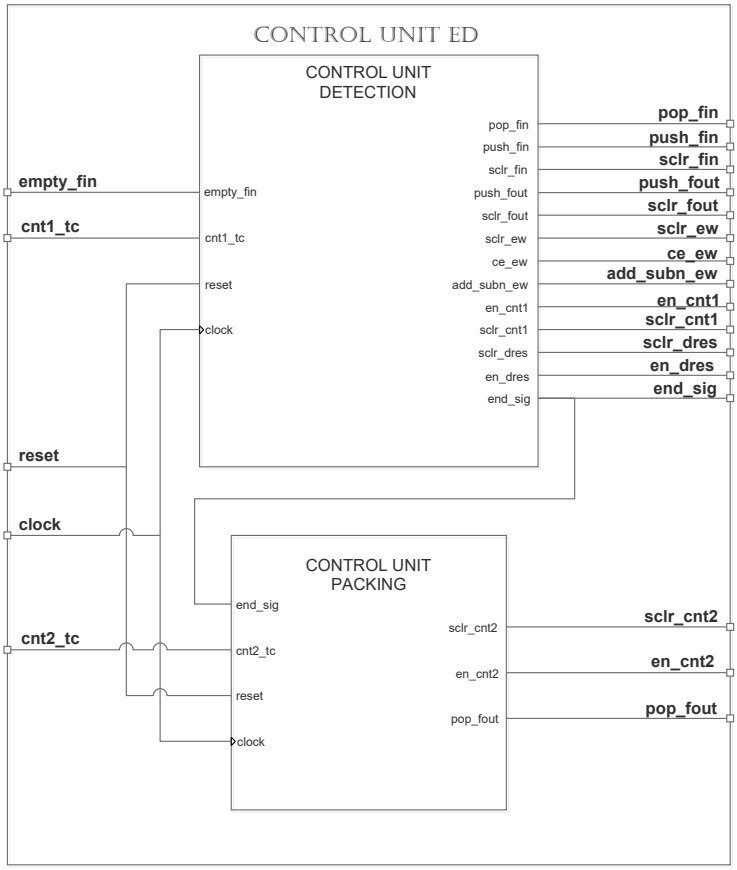

Figure 5.18: Control Unit ED Module (*Fixed Threshold*)

If the *empty_fin* binary value is "0", the state machine transit to the *ACCUMULATE_1* state. There are three consecutive states labelled ACCUMULATE. First *pop_fin* and *en_cnt1* signals are enabled and then, one clock cycle later, *ce_ew* and*push_fout* are activated. This because of the FIFO read operation latency which is equal to one clock cycle.
The FSM remains into the *ACCUMULATE_2* state until the terminal count (*cnt1_tc*) is sensed, where the state changes into *ACCUMULATE_3*. For these three previous states the accumulator mode is settled as addition.
When the machine enters in the *COMPARE* state, the accumulator mode changes into subtraction. The two consecutive states are needed to handle the latency equal to two of the accumulator. In the *WAIT_COMP* the counter is reset and the accumulator's clock enable remains active high.
The next state is *END_COMP* where the result of comparison is stored into the FF

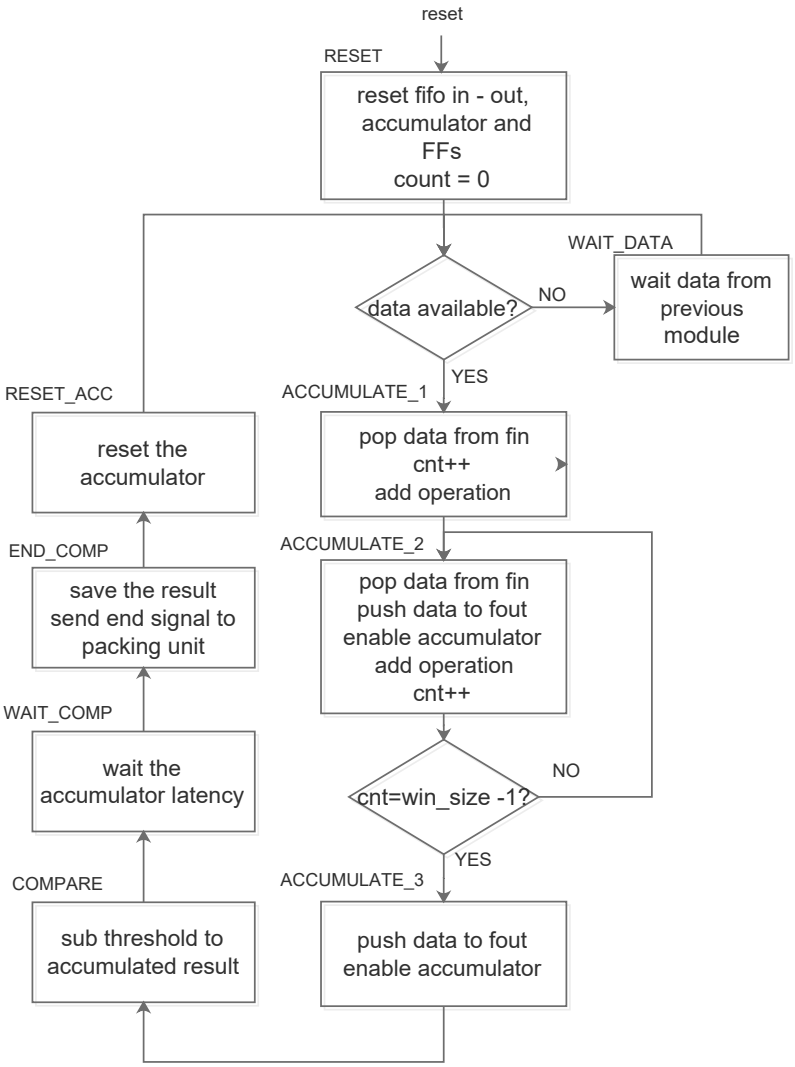

Figure 5.19: ASM chart Control Unit Detection (*Fixed Threshold*)

and the *end_sig* is activated to inform the other state machine.
Before restarting the other sub-band the accumulator is reset. The cycle starts again from checking the data from the FIFO until it will be empty.

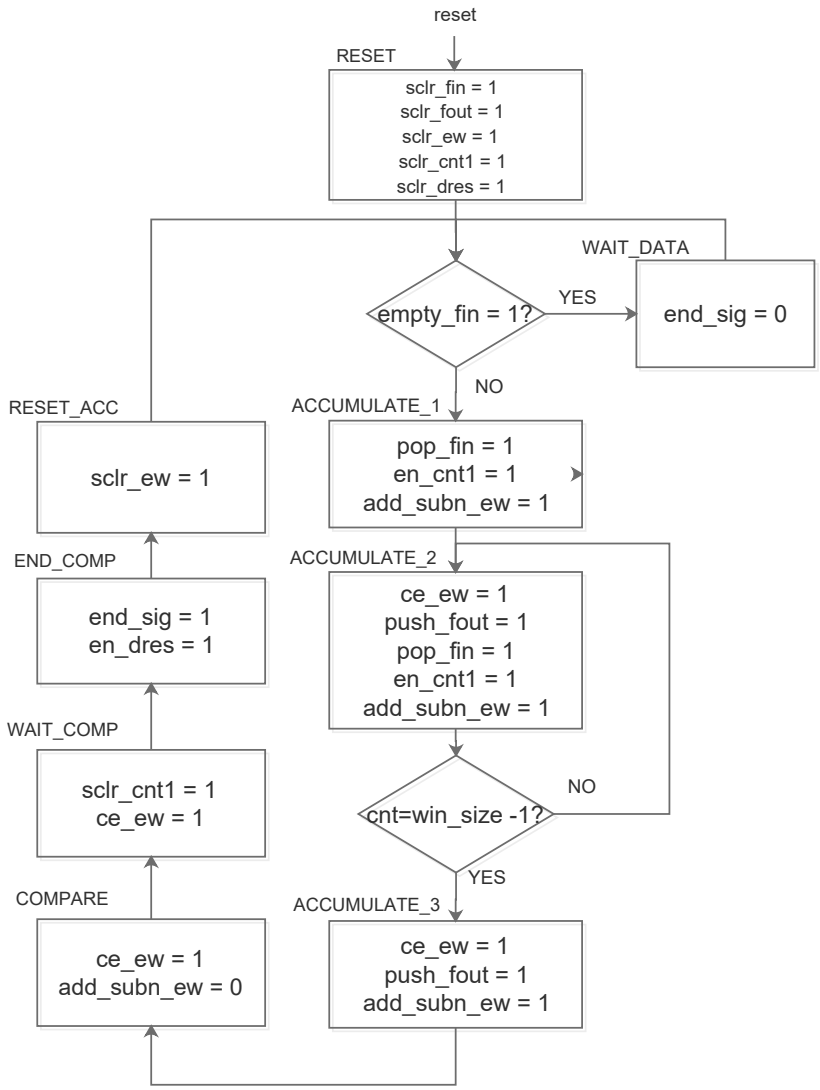

Figure 5.20: Control ASM chart Control Unit Detection (*Fixed Threshold*)

5.4.2 Control Unit Packing

The second FSM has the aim to pack the detection result with the SQM bin value. The detection result is expressed by one bit and its value is only ”0” or ”1”. Reducing of one bit the SQM value from 32 to 31 bits, the energy detection result is placed as the MSB bit. This packing operation is done by hardware link selecting the respective bits.

The control unit Detection analyses only one sub-band of m bins at time, the value of such m bins are stored into the second FIFO (Figure 5.17).
As reported in the ASM chart (Figure 5.22), the state machine waits the signal *end_sig* from the other FSM and then start to pop data from the FIFO until all bins of the sub-band are finished. The number of bins are counted by the *counter packing* presents in the datapath.

This FSM is characterized by only two states: *RESET* and *POP_DATA*. In the *RESET* state simply the counter is reset to "0", instead in the other state data is read from the FIFO and the counter is enabled.

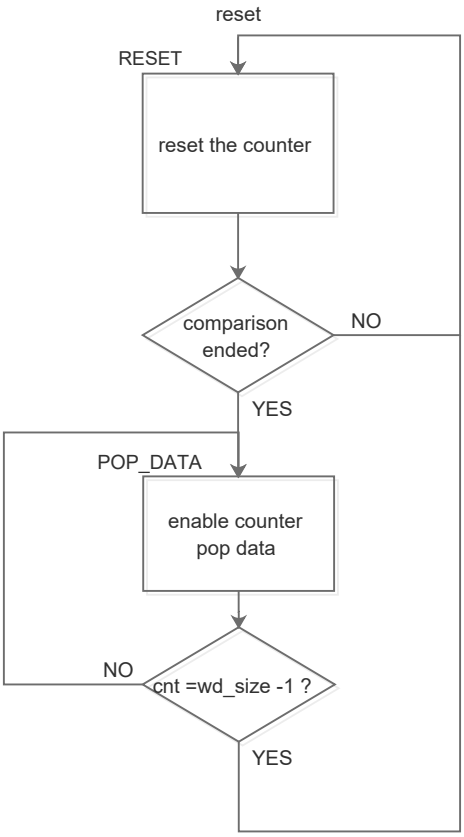

Figure 5.21: ASM chart Control Unit Packing

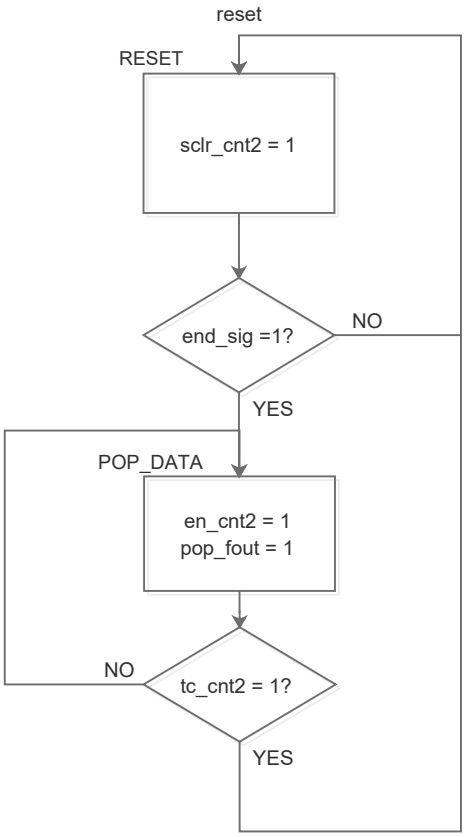

Figure 5.22: Control ASM chart Control Unit packing

5.5 ED Module Adaptive Threshold

The module for the adaptive threshold includes some more units respect the fixed one. As explained in the Section 4.3.2, the algorithm needs to evaluate the mean value of the all bins, performed in the D Factor module.
Because the WBX daughterboard, used inside the USRP device, induces a dc component and other components near the dc, the normalized spectrum values result smaller. To reduce this effect a DC elimination module has been implemented, simply, it put to zero some bins near the dc component. The position of these bins is evaluated by the software and the value is settled into three user registers.
The three modules mentioned before are connected in cascade mode starting from the DC elimination, then the D factor evaluation and finally the detection module (Figure 5.23). The following sub-sections explain in details each module how is implemented.

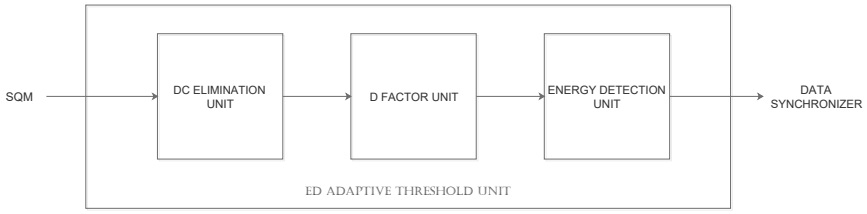

Figure 5.23: Energy Detector Adaptive Threshold

5.5.1 DC Elimination Module

The schematic of this module is shown in Figure (5.24). Three user registers are used to know the position of three bins to be reset. The dc component and other contributions around it are not positioned always to the central frequency requested. The receiving chain of the USRP is composed by a RF front-end part first and then a Digital Down Converter (DDC). The "noise" contribution comes from the first section of the chain.
The RF front-end includes a PLL that can't be locked to every frequency, but it is settled to the nearest frequency available to the target one. Then the DDC adapts the frequency range remained between the target and the RF front-end frequency.
Exploiting the results of the tuning central frequency command used by the software, the RF front-end frequency can be known precisely and so also the positions of these *"noise"* bins. These values are stored into three different user registers used as terminal counts to select the right bin on the 1024 bins of the FFT performed.
From Figure (5.24) the architecture can be easily understood. One counter counts

the bins and at the right position the terminal count changes its value from "0" to "1". Therefore also the multiplexer changes its state selecting zero instead of the incoming sample.
The registers and flip flops are used to divide the module respect the others in the chain in order to reduce the critical path.
To have a more readable figure the user registers has been neglected, just a data bus is used to recall the user register function.

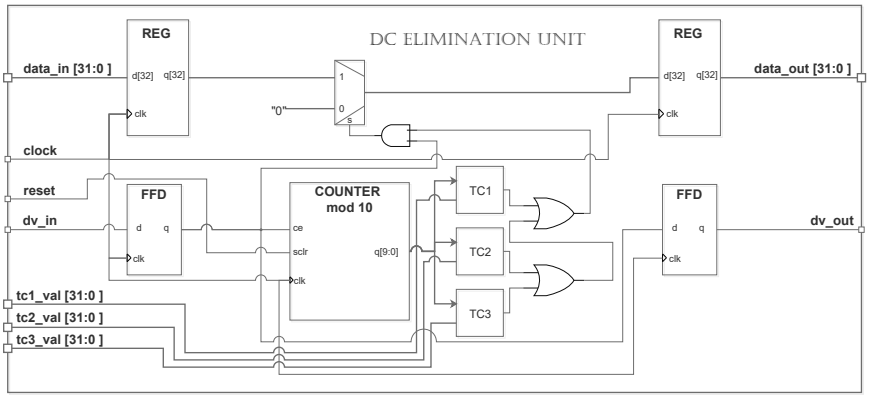

Figure 5.24: DC Elimination Module

5.5.2 D factor Module

Recalling the Algorithm explained in Sub-section 4.3.2, the expression of D factor is given by the Equation 4.2. Each bin of FFT performed has to be normalized by the mean value over all bins.
The Figure 5.25 shows the architecture of this module. Two steps are performed the first about the mean value evaluation and the second one the division of each bin by the nearest mean value power of 2. This allows to perform the division operation using a simple right shift.
The average is performed using an accumulator that adds each bin every clock cycle until the counter reaches the terminal count. Then a right shift by 10 is done to divide by 1024 to perform the mean operation.
The samples at the same time which are added, they are also stored into a FIFO of size 1024. When the mean value is evaluated, one sample each clock cycle is popped from the buffer and it is shifted by an amount power of two.

The average value of bins is rounded to the nearest power of two by a synchronous priority encoder described as following:

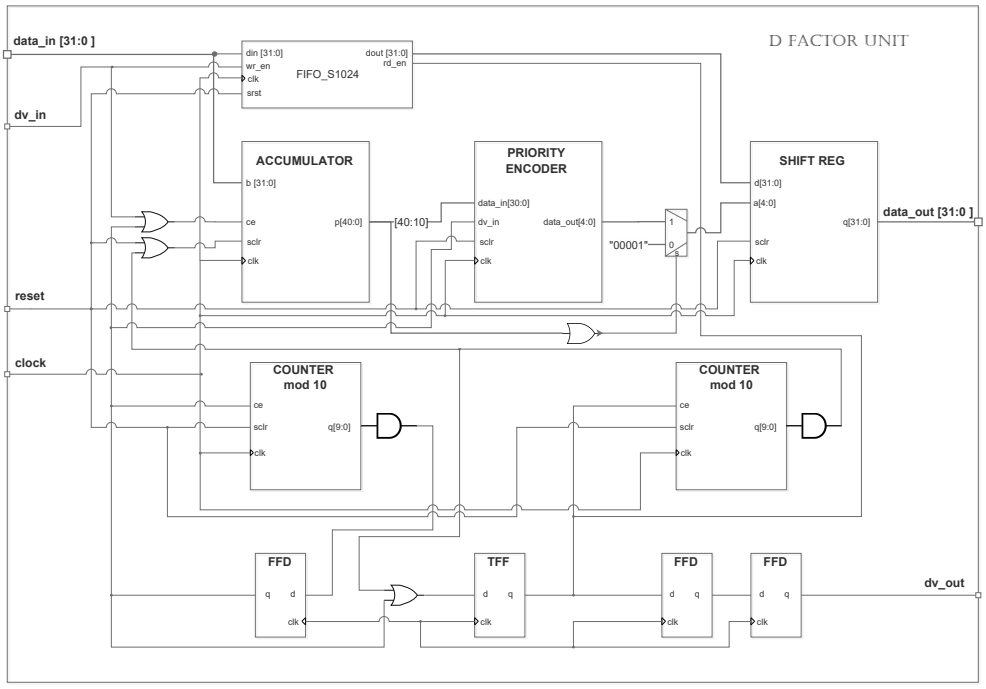

Figure 5.25: D factor module

```
module priority_encoder_32to6(input clock, input reset,
                              input [31:0]      data_in,
                              input             dv_in,
                              output reg [5:0] data_out);
always @ (posedge reset, posedge clock)
  begin
     if(reset == 1'b1) begin
        data_out = 6'b000000;
     end
     else begin
        if (dv_in) begin
           if (data_in[31] == 1'b1) begin
              data_out =32;
           end else if (data_in[30] == 1'b1) begin
              data_out =31;
           ...
           end else if (data_in[1] == 1'b1) begin
              data_out =2;
           end else if (data_in[0] == 1'b1) begin
              data_out =1;
           end else begin
              data_out =0;
           end
        end // if (enable)
     end // else: !if(reset == 1'b1)
  end // always @ (posedge reset, posedge clock)
endmodule
```

The output of this encoder is used to control the shift amount of a parallel shift register.
This shift register loads one bin value, shifts it by a amount and releases the result on the parallel output port (q).
A series of flip flops are used to generate the data available signal and to reset the counter. The terminal count of the left counter (Figure 5.25) toggles the T flip flop used to enable the second counter.
The count of bins to be divided is handled by this last counter, when it reaches the terminal count re-toggles the T-FF, resetting it.

About the data length, the port size of the accumulator is chosen as 40 bits to avoid the overflow over the sum of 1024 samples. Then for the right shift operations, the data length is brought again to 32 bits.

As in the module explained in the other sections there are a pair of signals for input and output: *data_in*(on 32 bits), *dv_in* and *data_out*(on 32 bits), *dv_out*.

5.5.3 Energy Detection Unit

Based on the fixed threshold architecture, the Energy Detection unit is updated to the adaptive case. The spectrum values, normalized by the D factor module and without the dc component, are analysed to understand the presence or not of transmitting signals.
The top level module is shown in the Figure 5.26, comparing it with Figure 5.18 there are few new signals. Instead of receiving directly data from the SQM unit, data comes from the D factor unit.
Two main units: *Datapath unit ED* and *Control unit ED* compose the module. The input and the output ports remains identical changing only the name. Two user registers are used as in the other architecture. The first indicates the number of bins of a sub-band and the second indicates the λ^* value.
The status signals sensed by the Control unit remains the same as the fixed architecture. The Control Unit ED has to handle some new control signals. The final output value does not change respect the fixed threshold architecture and it is defined on 32 bits, where:

- MSB [31] represents the detection result;
- other [30:0] bits represent the normalized bin value.

As all modules explained before the output signal is paired with a data valid signal: *dv_d_fct_dt*.

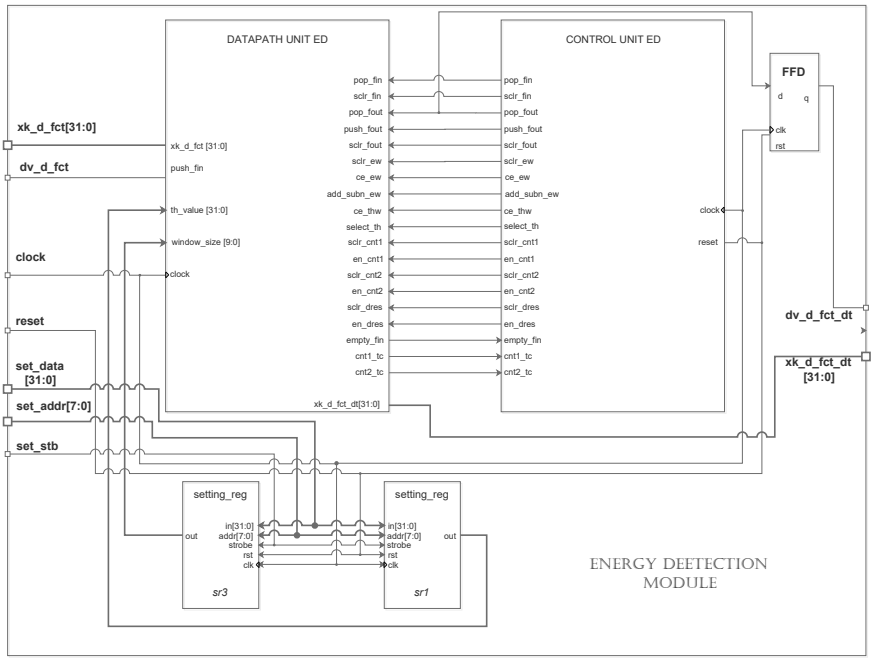

Figure 5.26: Energy Detector top level (*Adaptive Threshold*)

The Datapath schematic is shown in the Figure 5.27. The incoming data is stored in the input FIFO and then processed. Two accumulators are used instead of one. The first accumulator (*energy window accumulator*) is used to accumulate the sub-band bins and compare these with the threshold. The second accumulator (*adaptive threshold accumulator*) has the aim of evaluate the threshold accumulating the sub-band bins, shifted of a fixed position, and then adding the base threshold λ^* (Equation 4.4). The comparison operation is performed again using the subtraction, the threshold evaluated by the *adaptive threshold accumulator* is subtracted to the D_{window} value performed by the *energy window accumulator*. Two different multiplexers connected to the input ports of the accumulators allows to select the right input value. For the *energy window accumulator* the multiplexer selects between the input samples and the final threshold value. The second one instead selects between the λ^* value and the D factor bins shifted.
The sign bit of the subtraction represents the detection result and it is stored into a FF.
The number of sub-band bins is used to implement the terminal count for both the counters. The terminal count is represented by just a rectangle unit to have a lighter schematic but is implemented, as in the fixed architecture, using a bit-wise EX-NOR gate. When the value corresponds to the count value, the TC changes from "0" to "1". *Cnt1_tc* and *cnt2_tc* are status signal used by the two FSMs implemented in

the Control Unit ED. The second FIFO presented in the Figure 5.27 is used for the packing operation when the detection result is available. The normalized samples popped from the first FIFO are pushed directly in the second one and in the input multiplexer of *energy window accumulator*.

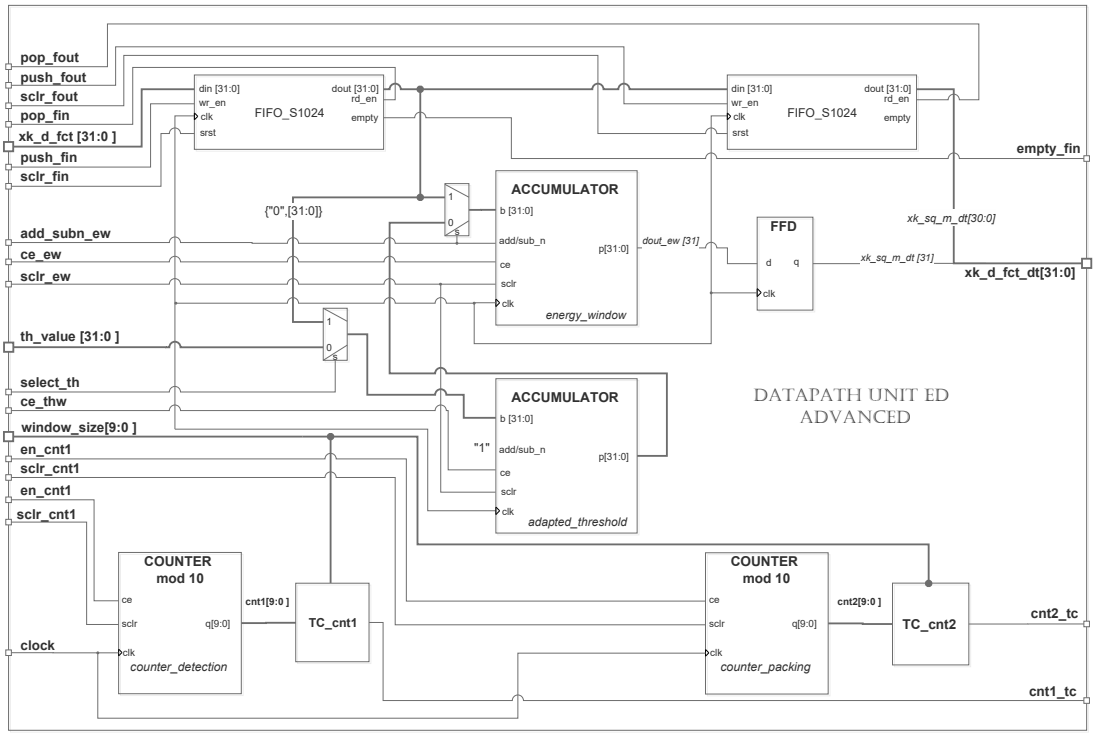

Figure 5.27: Datapath Energy Detector (*Adaptive Threshold*)

Figure 5.28 shows the Control Unit ED module. It includes two FSMs: one to handle the signals for the detection part (*Control Unit Detection*) and one to pack detection result with normalized spectrum values (*Control Unit Packing*). The two FSMs communicate together trough the *end_sig* signal: when the detection result is ready the Control Unit Detection activates this signal as a start signal for the other state machine. The following paragraph explains how is structured the Control Unit Detection. The Control Unit Packing remains the same as the fixed architecture, it is suggested to the reader of take a look at the Subsection 5.4.2.

Control unit Detection

The Control Unit Detection is a state machine which it controls all steps to evaluate the sub-band energy value, the threshold and compare the two values.
The FSM is characterized by eleven states as reported in the ASM chart Figure 5.30:

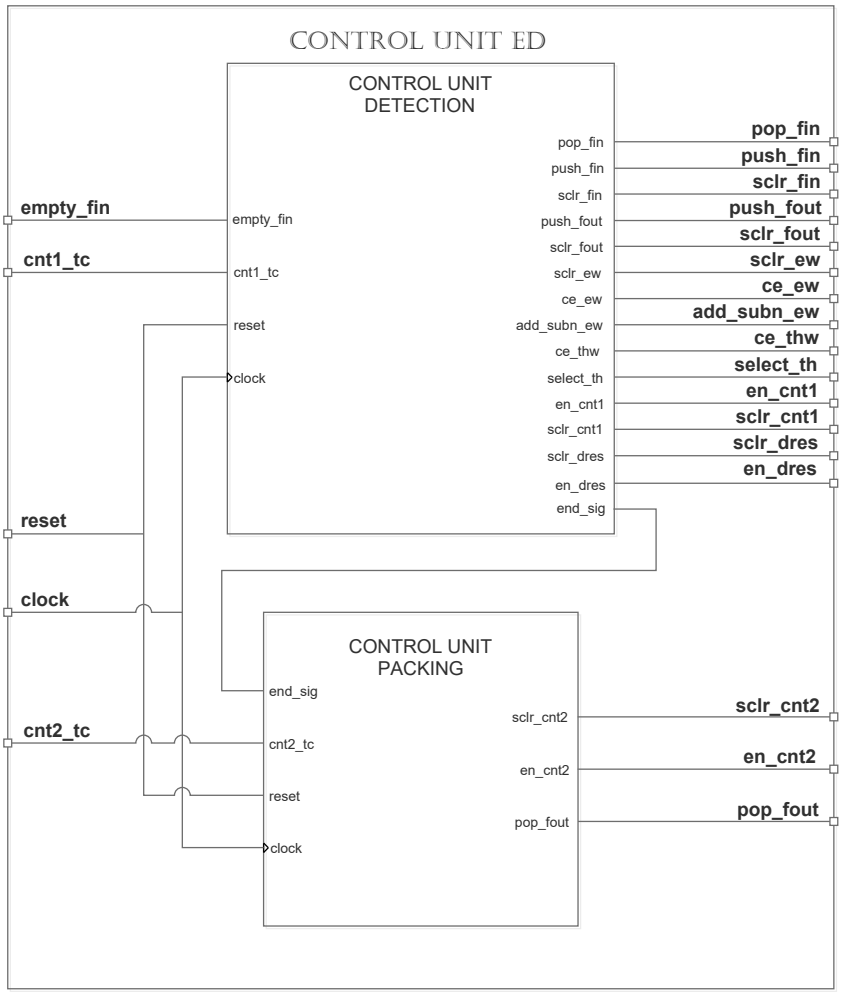

Figure 5.28: Control Unit (*Adaptive Threshold*)

RESET, *WAIT_DATA*, *ACCUMULATE_1*, *ACCUMULATE_2*, *ACCUMULATE_3*, *TH_EV1*, *TH_EV2*, *COMPARE*, *WAIT_COMP*, *END_COMP* and *RESET_ACC*.
Comparing the number of states with the fixed threshold one there are two additional states, these states are used to evaluate, with the *adaptive threshold accumulator*, the λ value.
The RESET state has the higher priority respects the other states and it enables all the synchronous clear or reset signals inside the datapath. When an incoming stream of data is available, the input FIFO changes the empty status signal from the binary value "1" to "0". If the FIFO remains empty, the state machine check continuously this status signal remaining into the *WAIT_DATA* state. When the

test condition is satisfied the state changes into *ACCUMULATE_1*. There are three states labelled *ACCUMULATE*.
ACCUMULATE_1 enables the read operation from the input FIFO and the counter. The next two states the two accumulators starts to accumulate (*add_subn_ew =1*) samples until the number of sub-band bins is reached. Moreover the data is stored at the same time into the output FIFO.
When the state becomes *TH_EV1*, the λ^* value is added to the *adaptive threshold accumulator*. Then another state *TH_EV2* is used to handle the latency of the accumulator, in this state the subtraction operation is selected for the *energy detection accumulator* which is disabled during the *TH_EVx* states.
Proceeding in sequence the next step is to compare the threshold value with the sub-band D factor value accumulated. States *COMPARE* and *WAIT_COMP* controls the compare operation; the second state is used to handle the accumulator latency. Then the last state of the cycle is *END_COMP* where the signal used to store the result in the FF (present in the data path) is activated. Moreover in this state the second state machine, *Control Unit Packing*, is informed that the result is available and the pack operation can be started.

5.6 Data Synchronizer

The last module of the FPGA system is the *Data Synchronizer*. Except before the Fourier Transform the data flows is a stream of 1024 samples one each clock cycle. This stream of samples has to be re-adapted to the format used before the custom module, where new sample comes at f_s frequency as shows the Figure 5.3.
The schematic of this unit is shown in the Figure 5.6.
The module uses a FIFO of size 1024, a modulo 4 counter and some logic units to evaluate the terminal count (EX-NOR gates and AND gates). As done for the energy detection units the terminal count value is stored into a user register that can be settled by software. In that way the sample frequency of the ADC can be changed; the only constraint to be respected is: the ratio between f_{clock} and f_s has to be an integer value.
The data incoming in the module is stored into the FIFO, the write signal is connected to the *dv_in* signal.
When the empty signal changes its binary value from "1" to "0" and the TC of the counter is active high, the FF used for the *dv_out* signal generation changes its value from "0" to "1". The Figure 5.32 shows an example of timing between the signals of the module.
The counter is in a free running condition and it recreates the f_s frequency dividing the f_{clock}. The pop operation is performed by the terminal count of the counter and so at f_s frequency.

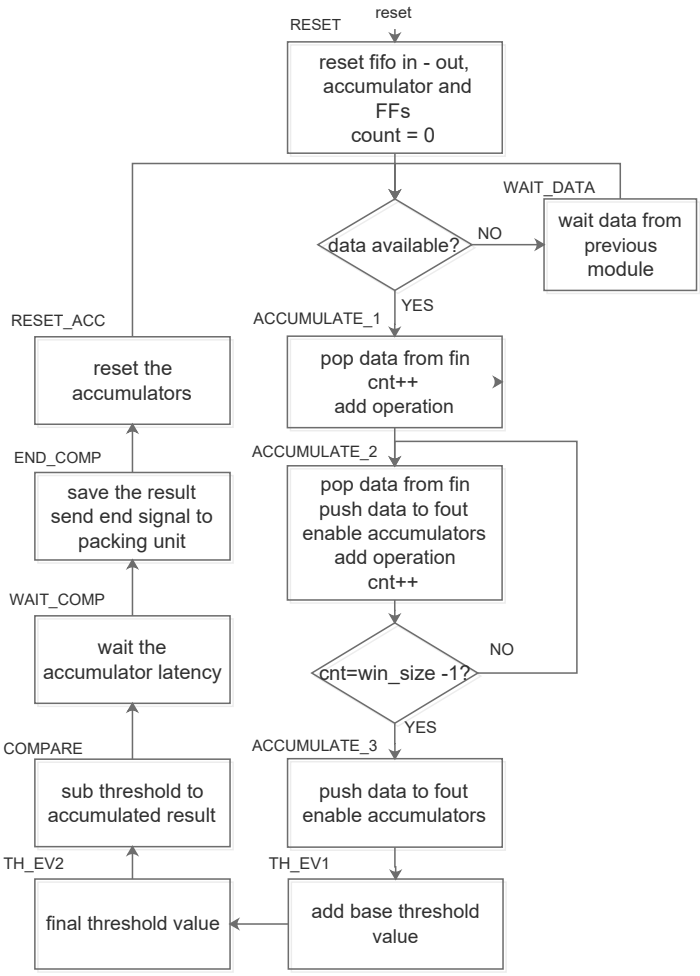

Figure 5.29: ASM chart control unit detection (*Adaptive Threshold*)

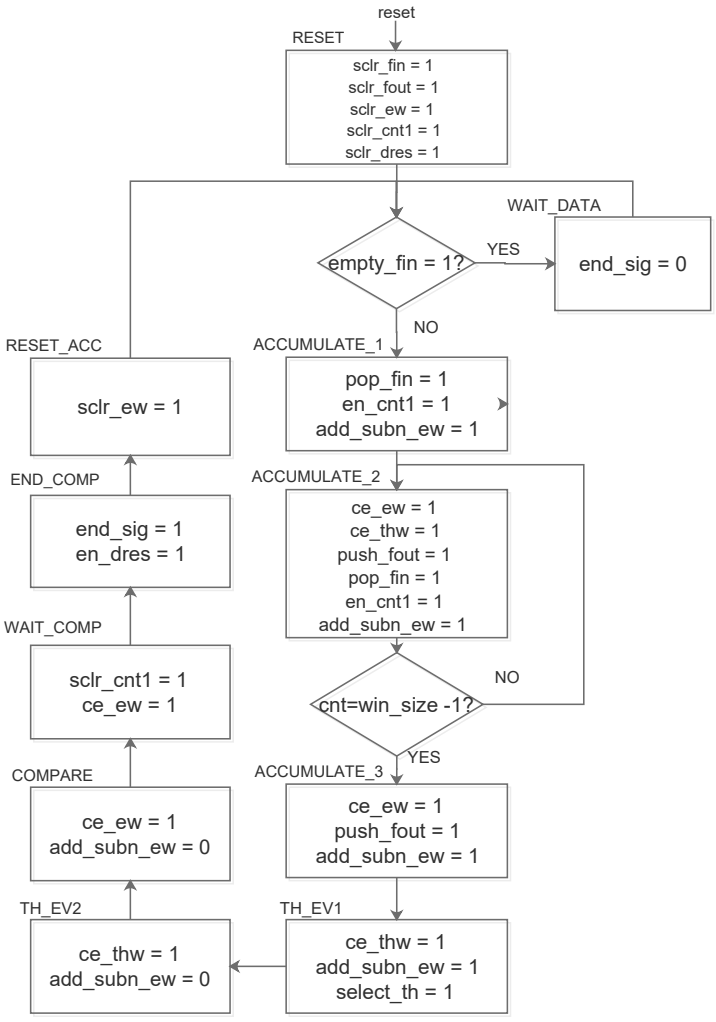

Figure 5.30: Control ASM chart control unit detection (*Adaptive Threshold*)

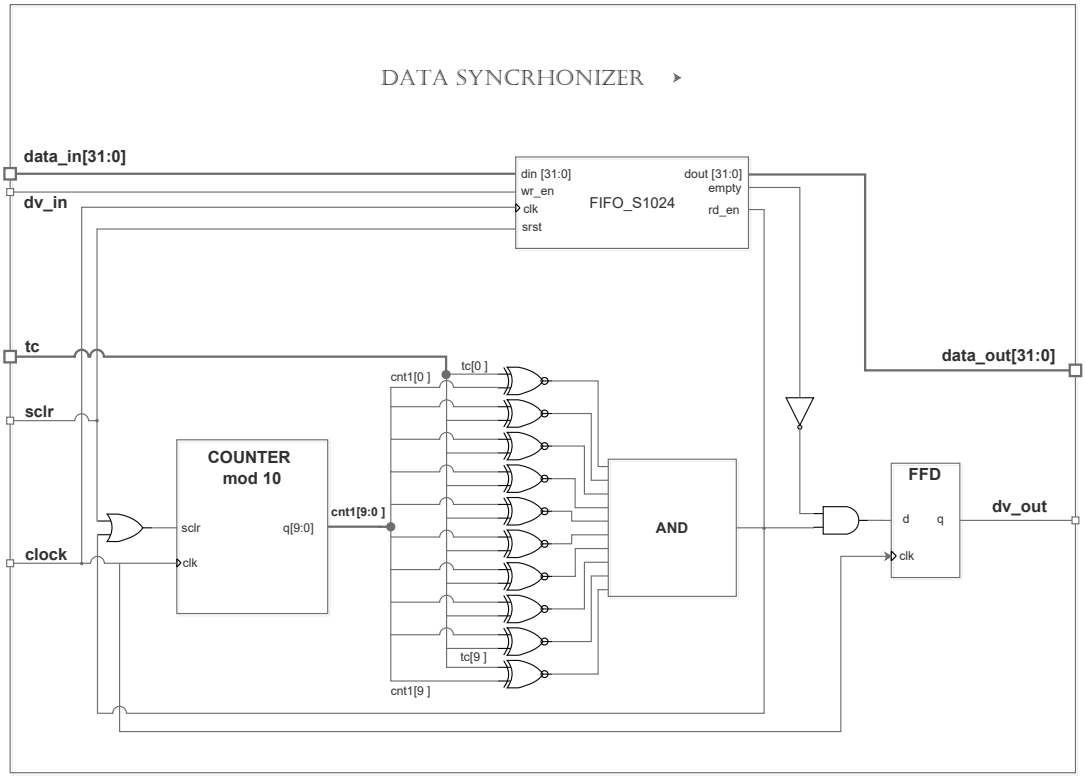

Figure 5.31: Data Synchronizer Module

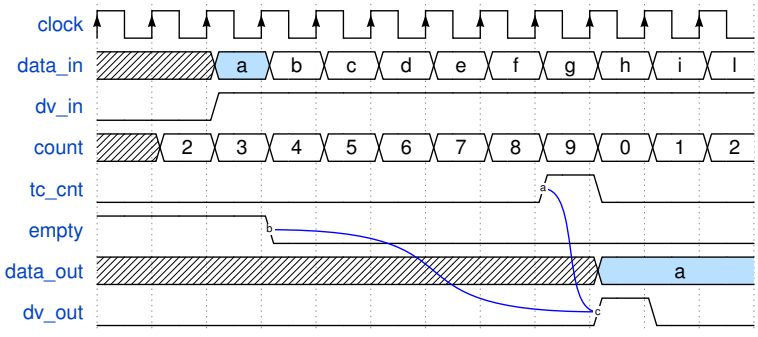

Figure 5.32: Data Synchronizer Timing Diagram

Chapter 6

Software Scanning Technique and Implementation

6.1 Introduction

The hardware implemented, synthesized and flashed into the FPGA has to be controlled by the Host CPU as explained in the Chapter 4.
This Chapter has the aim to explain how the software program is organized and how the retuning of the central frequency can be done in a reliable way.
Several environments can be used to control the USRP device and are based on a C++ library called *UHD Driver*. Developers gives some examples to use this library and a web page for the description of functions and classes.

6.1.1 Tuning Method and Policies

Figure 6.1 shows the receiver and transmitting chain differentiating the motherboard respect daughterboard.
The WBX daughterboard is based on two chips from Analog Devices, the ADL5387 quadrature demodulator and the ADL5385 quadrature modulator. The Analog Devices ADF4350 (*Wideband Synthesizer with Integrated VCO*), used to implement the Phase-Lock-Loop with a voltage controlled oscillator integrated, synthesises the local oscillator signals that drive the ADL538x.
Reading the datasheet of the PLL the typical lock time without any improvement is of $400\mu s$. If a particular register (CSR) of ADF4350 is enabled the lock time is improved up to $200\mu s$.

The tuning operation on the USRP-N210 is performed by two steps. First the WBX daughterboard (local oscillator) is settled as close as possible to the target frequency. The frequency generated by the local oscillator is called RF-frequency.
After the f_{RF} of the daughterboard is settled, the *Digital Down Converter* (DDC) or

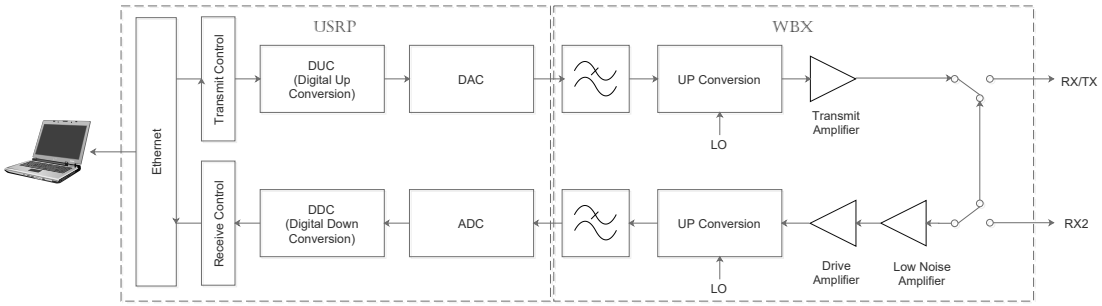

Figure 6.1: USRP and WBX block diagram

Digital Up Converted (DUC) of the USRP-N210, is set to refine the remain difference in frequency. This second step is performed only in case a frequency difference is present. The frequency generated by the DDC or DUC is called DSP Frequency.
When the *tune_request* command is sent to USRP, the result of the tuning request is available to the user. It gives info about both the frequencies mentioned before.
The UHD driver provides three different polices for frequency tuning [16]:

- Auto;
- Manual;
- None.

Each policy can be applied separately on the RF-frequency and the DSP frequency. Therefore 3^2 possible options can be used.

A deep analysis on the tuning performances of all possible configurations has been performed by Galal in [17]. The daughterboard used by this team was the RFX2400 but similar considarations can be done on the WBX daughterboard. The method followed by these researchers is to exploit the sensor lock value, available on the UHD driver, to measure the settle time for a tune request. All the different possibilities between the polices have been compared. Some of the 3^2 possibilities can be excluded because the difference between generated frequency and the target one is large. These pairs are *None-Auto* and *None-Manual*. The first policy is referred to the RF-frequency and the second to the DSP one.
Obviously the *None-None* configuration is an inappropriate combination because generates only the center frequency of the daughterboard.
Not all the combinations covers full frequency span of the board, so another group has to be discarded which is *Auto-None* and *Manual-None*. Among the remaining policies the *Auto-Auto* configuration remains the best compromise with the best

average lock time.
The study performed by Galal also shows the difference between the RF-frequency and the DSP frequency lock times. The DUC and DDC uses a NCO, therefore the lock time is very low. The RF-frequency lock time instead, because the presence of the PLL and VCO, is relatively high.

The system has to scan a wide frequency bandwidth continuously, an important question is: how often the retune request can be sent?
From the previous analysis the first constraint is due by the lock time of the local oscillator, the second one instead is about the processing time used by the FPGA to send the result to the Host CPU.
The first constraint implies an average lock time around $500\mu s$. The second constraint in the worst case is about $280\mu s$. To provide some margin, the delay between the retune command and the receive samples command is considered as 1 millisecond. For a conventional OS is difficult to guarantee sending commands every milliseconds. To manage this limitation, Ettus Reserch has introduced the concept of timed commands.
Timed commands permit to indicate the precise time when a command should be executed. The precision of this time is dictated by the FPGA clock (precise to microseconds or less). The device has a dedicated input FIFO for this feature which it allows to send up to 16 commands in advance before are executed. In case the Host CPU lags for some reason there are already other commands in queue which will be executed.

The Algorithm 1 shows the method used in the software program to retune and receive samples in a reliable way. As should be noticed some commands are sent in advance, then each time one stream of samples is received an additional pair of timed command is sent. In case a software lag is encountered the time t_0 of the FPGA is recorded again and the new timed commands are based respect this new value.

Algorithm 1 Re-tuning and receive samples

```
1: Initialize the USRP
2: Create a tune_request vector to cover the freq. band
3: Record the t_0 value from the USRP
4: Create a cmd_time vector with each value spaced by a span value
5: for k = 0 to k = 8 do
6:     set command time @ cmd_time[k]
7:     tune_request[k]
8:     rx stream command @ cmd_time[k]+delta
   end for
9: while (1) do
10:    receive samples
11:    if receiving time > span time then
12:        start from a new time t_0
       end if
13:    set command time @ cmd_time[k]
14:    tune_request[k]
15:    rx stream command @ cmd_time[k]+delta
16:    k++
17:    if k > k_max then
18:        k = 0
       end if
   end while
```

6.1.2 Software Enviroments

Different environments are available to control the USRP device, all of them are based on a *C++* library. The software can be written directly in *C++* language recalling the objects and methods available on the library, or in Phyton language.
Users writes often the programs in Phyton language because the USRP is completely integrated in the *Gnuradio*[1] platform.
To have the best control of the USRP a low level approach is followed: the program to be run by the Host CPU has been written into C++ program language.
The examples provided by the developers has been analysed in order to understand how to receive a stream of data, how to retune the central frequency etc. These examples are not very exhaustive such as the usage of timed commands. In order to complete the documentation of the device, there is available an open community based on a mailing list for support.
Another environment used to create a good user interface is *GnuPlot* . This graphic utility generates plots of mathematical function or a set of data. Different terminals selection permit to generate not only directly on the screen the plots but also pictures in different format, for example: jpg, eps, pdf ecc. For the scope of the program the terminal selected is wxt which permits to plot graphs directly on the screen.
Thanks to a library called *Gnuplot-iostream interface* the Gnuplot utility can be integrated in a C++ program. Basically this interface permits to create an iostream pipe between the program and Gnuplot to send setting values and data for the graph.

6.1.3 Software Program

The software part of the system has three major tasks:

- Re-Tuning the RF-frontend;
- Retrieving the data from the USRP;
- Logging the data.

The main program structure is as in the Algorithm 1 with: a first initialization, a section to send some commands in advance and the final receiving part.
In the program flow first of all an initialization procedure is performed. The sample frequency(f_s), the address of the device, the receiving port, the gain of the PGA (programmable gain amplifier) and other parameters are settled.

[1]Gnuradio is free and open-source tool which provides a graphical user-friendly interface and units to perform signal processing dedicated for Software Define Radio. It can be used with real external hardware board for SDR or simply without hardware in a simulation like environment[18]

Among the previous parameters there are also the user registers mentioned in the Hardware Implementation (Chapter 5). Depending on the FPGA image used, there is a different number of user registers to be settled.
The user registers, shared between the two architectures, store the:

- Number of FPGA clock cycles equivalent to f_s (*address* 0);
- Threshold value (λ) (*address* 1);
- Number of sub-band bins (*address* 3);

For the adaptive architecture three additional user registers for the dc elimination module are used, which have the following *addresses*:4,5 and 6.

The number of FPGA clock cycles equivalent to f_s are evaluated by a function where the sample frequency value settled is passed. In that way the user has just to indicate the sample frequency without care about this register.
A similar approach is done for the dc elimination user registers used in the adaptive threshold architecture. When a tune command is performed, the parameters settled for the RF-frequency and DSP frequency can be read. Because the DC component is due by the RF-frontend, reading the RF-frequency the DC position can be evaluated.
The RF-frequency result of the tuning operation and the target frequency are passed to a function that evaluates the number of the bin correspondent to the DC component. This value is then settled into the appropriate user register. To avoid some possible lags from the software when the timed commands are sent, all the DC position values are evaluated during the initialization routine. The RF-frontend is tuned to all the possible values for scan the range selected and the position values are stored into a vector to be reused by the program.
After the initialization an array for the timed commands, with the precise time when the tuning operation will be executed, is created. A first routine sends some commands in advance as in the Algorithm 1, the sequence on commands for the fixed threshold is:

```
for( i=0 ; i<N;i++){
 usrp->set_command_time(cmd_time[i]);
 usrp->set_rx_freq(tune_request[i]);
 stream_cmd.time_spec = cmd_time[i]+delta;
 rx_stream->issue_stream_cmd(stream_cmd);
}
```

In the case of the adaptive threshold instead three more commands has to be added in order to send the value for the DC elimination. The sequence becomes as following:

```
for( i=0 ; i<N;i++){
 usrp->set_command_time(cmd_time[i]);
 usrp->set_rx_freq(tune_request[i]);
 usrp->set_command_time(cmd_time[i]+delta);
 usrp->set_user_register(DC_EL_TC1_ADDRESS,dc_tc[3*i],0);
 usrp->set_command_time(cmd_time[i]+delta+delta);
 usrp->set_user_register(DC_EL_TC2_ADDRESS,dc_tc[3*i+1],0);
 usrp->set_command_time(cmd_time[i]+delta+delta+delta);
 usrp->set_user_register(DC_EL_TC3_ADDRESS,dc_tc[3*i+2],0);
 stream_cmd.time_spec = cmd_time[i]+delta+delta+delta+delta;
 rx_stream->issue_stream_cmd(stream_cmd);
}
```

For both the sequences a parameter *delta* has been used. The delta value is the time laps between two timed commands and it is less than the time between two tune requests.
The commands sent in the pieces of code presented are :

- *set_rx_freq(tune_request)*: to set the RX center frequency;
- *set_user_register(address,value,mboard)*: writes on the user configuration register bus;
- *issue_stream_cmd(stream_cmd)*: issues a stream command to the usrp device, this tells the usrp to send samples into the host. The stream_cmd variable contains all informations about streaming.

As should be noticed by the pieces of code written before, the timed command operation is performed into two steps. Before the *set_command_time(t_x)* is issued and then the command that has to be executed at time t_x is sent to the USRP.
For the case to retrieve a stream of data at particular time, the *stream_cmd* variable allows to set a parameter called *time_spec*, which has the same scope of the *set_command_time* method. This kind of parameter works only if the *stream_now* parameter is false. In case is true the stream starts ASAP without cares about the *time_spec*.
Moreover exist different streaming modes: *continuous*, *n. samples and done* and *n. samples and more*. In this case the mode selected is *n. samples and done*.

The third part of the software is defined by an infinite loop.
The data sent by the USRP, through the previous command to start the stream of samples, is collected into a vector. The received data is split into SQM values and

Detection values and then logged into two separate binary files. Before the reordering of the data routine new timed commands will be sent.
A test on the execution time is performed to understand if some software lags happen. This test evaluates if the difference between the execution time and the $t[i-1]$ of the previous tuning operation is bigger that the *span* time. In case the test gives a true result the $t[i]$ is reset. The following piece of code shows how this test is implemented.

```
if(usrp->get_time_now().get_real_secs()-cmd_time[i-1]>span)
 cmd_time[i]=usrp->get_time_now().get_real_secs()+0.001;
else
 cmd_time[i]=cmd_time[i-1]+span;
```

After the test mentioned the new timed commands for one tuning operation will be sent.
The received data are stored into two different files one for the SQM and one for the Detection result. Each value inside the file is paired with the respective frequency bin.

Referring to the Subsection 6.1.2 a GUI is used to plot the data. To have the infinite loop as light as possible and avoid a big amount of software lags, the Gnuplot utility runs a script loaded the first time when the program enter in the loop.
In the initialization part the Gnuplot script is written into a file with all the informations about the frequencies to scan and so the dimensions of the files. Using Gnuplot in this way has the advantage to use another process linked to the program through a pipe. This permits to unload the weight of the program.
The script simply reads the data files and plots in the same window two graphs, one for the detection result and one for the SQM values of the spectrum. The script replots the graphs with a lower rate: each 1 second. Following an example of GnuPlot script written by the program:

```
set term wxt 0 size 1366,768
while(1){
 set multiplot layout 2, 1 title "Energy␣Detection␣Frequency␣span" font ",14"
 set tmargin 2
 set title "|Y|^2"
 unset key
 set xrange [8.8e+07:1.08e+08]
 set format y "%e"
 set yrange [-0.1:5]
 set xtics rotate 90,500000
 set xlabel "Hz"
 plot "SQM.dat" binary format="%lf" with points title 'SQM'
 set title "Detection"
 set grid
 unset key
 set xrange [8.8e+07:1.08e+08]
 set yrange [-0.1:1.1]
 set xlabel "Hz"
```

```
 plot "Detection.dat" binary format="%lf" with lines title 'DETECTION'
 unset multiplot
 set grid
 pause 1
}
```

The user through the command line when runs the program can set the following parameters:

- Sample rate f_s[Hz];
- Gain of the receiving path[dB];
- Starting central frequency[Hz];
- Number of sub-band bins;
- Number of frequency spans;
- Span time[sec];
- Delta time[sec]

Chapter 7

Results

7.1 Introduction

In this chapter several results from the thesis will be explained: from the hardware aspects to the system performances in terms of *probability of detection* and scan speed.
The first part explains results about the FPGA custom modules synthesised and the simulation of some modules. After some tests performed to know some characteristics of the system have been explained.
The last section shows the result with a practical experiment with the scanning of the FM broadcast frequency range in Brussels.

7.2 Hardware simulation and synthesised resources comparison

7.2.1 Simulation

All hardware modules explained in the Chapter 5 has been simulated with the Xilinx simulator: ISim, integrated in the Xilinx ISE suite.
The test of this modules has been performed by a test-bench which provides precise control signals and data to the UUT (*Unit Under Test*) and it receives results from the UUT. The input data, created with a Matlab Script, are read from a file and the data results from UUT are written to a file. To control the correct behaviour a first comparison with Matlab and simulation results has been performed. A second deep inspection on the simulation timing diagrams has been done to verify the correct signal behaviour.
Figure 7.1 shows a graphical representation of the simulation configuration.

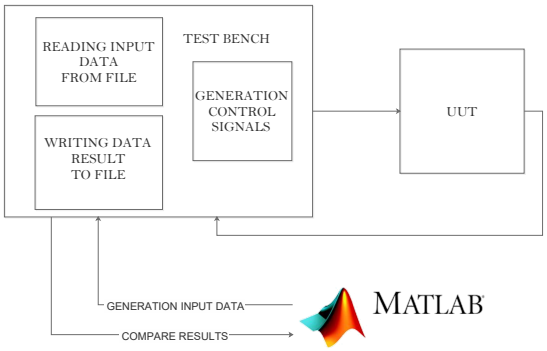

Figure 7.1: Simulation Configuration

All Ip cores generated with Xilinx have their own simulation model and a behavioural simulation can be performed. As example of one simulation result the Figure 7.2 shows the simulation of the FFT module.

The samples generated with Matlab, to simulate the FFT module, respect the following function:

$$y(t) = sin(2\pi f_2 t) \cdot e^{(i(2\pi f_1 t))} \tag{7.1}$$

where f_1 value is $1MHz$ and f_2 value is of $50kHz$. The t value in Matlab is described by an array of time values with a step of $t_0 = {f_s}^{-1}$. The sample frequency used has the value of 10MHz.

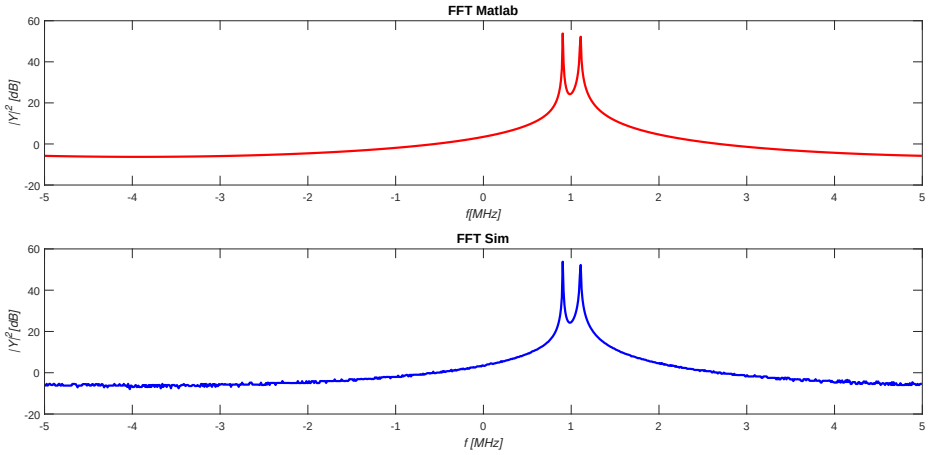

Figure 7.2: Comparison between Matlab FFT result and module simulation result

The two FFTs, as can be noticed, are very similar. These are not identical

because the FFT performed in Matlab uses a floating point data format for the Transform computation, instead the verilog implementation uses a fixed point data format on 16 bits for decimal part. Anyway the relative error respect the simulation result, where the signal is present (around $1MHz$), is less than 1%.

7.2.2 Synthesised resources comparison

All the FPGA modules created have been integrated in the FPGA source code [19] of the USRP and then synthesised with the Xilinx ISE suite. To check the correct behaviour of all modules, different FPGA images has been created. Exploiting the chain architecture used, only some module has been put in the chain and the synthesised image has been tested on the device. The following FPGA images has been created:

- image with the FFT module;
- image with the FFT and SQM module;
- image with the complete chain for the fixed threshold;
- image with the complete chain for the adaptive threshold;

All the FPGA images listed before includes as last module of the chain the *Data Synchronizer Unit*. The Table 7.1 shows the resource consumption by the different images synthesised. The synthesis tool provides a summary report with all the resources used and results concerned to the timing constraints.

	FFT	**SQM** [1]	**Fixed** λ	**Adaptive** λ
Flip Flops	1196(2.5%)	1280(3%)	1499(3%)	1745 (4%)
4-input LUT	1128(2%)	1150(2%)	1357(3%)	1737 (4%)
Slices	429(2%)	433(2%)	582(2%)	1021 (4%)
DSP48A	5(4%)	8(6%)	9(7%)	11 (9%)
RAM16BWER	11(9%)	11(9%)	15(12%)	17 (14%)

[1] SQM stands for the image with FFT and SQM units in the chain.

Table 7.1: Resource Consumption FPGA images

As expected the adaptive threshold architecture consumes more resources respect the others architecture, in particular respect the fixed threshold one. Overall the resources consumption remains very low and all the architectures uses less resources than those available.

7.2.3 Timing Performances

Simulating and analysing all the latencies of module used, the computational time has been evaluated in term of clock cycles. As mentioned in previous chapters the FPGA works at clock frequency of $100MHz$. The computational time of the entire system depends on the sample frequency and on the number of sub-band bins. The Table 7.2 shows time results considering a $f_s = 10MHz$ and $M = 8$.

Module	Clock Cycles	Time	Cycles Expression
FFT ip_core	7385	$73.85\mu s$	
FFT with RAMs	17625	$176.25\mu s$	$N\frac{f_Clk}{f_s} + FFT$
SQM unit	6144	$61.44\mu s$	$6 \cdot N$
Energy Detection Fixed	1674	$16.74\mu s$	$\frac{1024}{M}(5+M)+M+2$
DC elimination unit	1026	$10.26\mu s$	$N+2$
D factor unit	2051	$20.51\mu s$	$2 \cdot N + 3$
Energy Detection Adapted	1930	$1.93\mu s$	$\frac{1024}{M} \cdot (7+M)+M+2$
Data Synchronizer	1024	$10.24\mu s$	$N\frac{f_Clk}{f_s}$
Energy Detection system Fixed	29800	$298\mu s$	
Energy Detection system Adaptive	26467	$264.67\mu s$	
difference	3333	$33{,}33\mu s$	

Table 7.2: Computational time of hardware modules with $f_s = 10MHz$ and $M = 8$

The last column of the Table 7.2 indicates the expression to evaluate the clock cycles based on the f_s, M and the latency cycles due by IP cores and registers used in the modules. The N value stands for the size of the FFT which is 1024. Both the architecture for a sample rate of $10MHz$ the processing time for one FFT analysis around a certain f_c are less than $300\mu s$.

7.3 System test, characterization and performances

All FPGA images generated has been flashed into the USRP and tested feeding the URSP with real signals. To generate an appropriate real signal, the *Rohde&Schwartz SMATE200A* signal vector generator has been used [20].
The signal generator can generate two independent RF signals from a number of standards: Bluetooth, GSM, Tetra, CDMA, multicarrier. These two outputs has been added trough an RF adder and connected to the USRP (Figure 7.3).

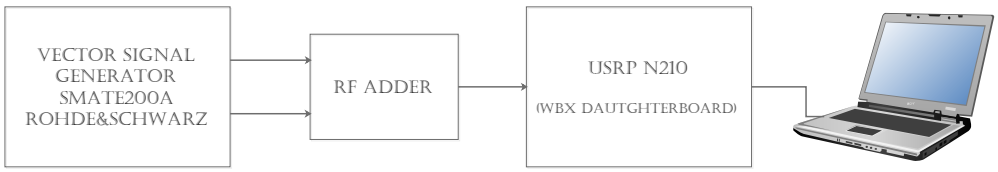

Figure 7.3: Laboratory test set-up

7.3.1 FFT Tests

The first test has been performed on the FFT module synthesised to evaluate the quality of the FFT. The signal generator has been settled to generate a multicarrier signal centered at $100MHz$ with 20 carriers spaced by $200kHz$.
The aim of the test is to compare the FFT evaluated by Matlab and the FFT obtained with the FPGA implementation.
The FFT performed in Matlab is obtained by recording samples in time-domain from the USRP and performing the FFT off-line. Figure 7.4 shows the comparison between the two approaches.

It should be noticed that most of the bin values for the FPGA implementation are zero. This is due to the limitation of fixed-point computations in the FFT. As explained in the selection of the algorithm for the adaptive threshold, this limitation does not permit to evaluate directly on the FFT the noise variance level. The Figure 7.5 shows the gap between the noise variance level and the minimum FFT level that can be sensed.

Other tests changing the standard signal has been done in order to check the correct operation with different type of signals.
A second test performed on this FPGA image is to understand which is the minimum sense level in dBm of the FFT module. Generating a Bluetooth standard signal and changing its transmission peak power, for a receiving sample rate of $10MHz$ the minimum level which the signal can be sensed is around of $-75dBm$.

7.3.2 Energy Detection system Tests

Using a similar approach as in the previous Section 7.3.1 both the two different architectures has been tested. Two types of environment test have been used: with the USRP connected to the signal generator and the USRP connected to an antenna. When an antenna is connected to the USRP the gain of the receiving chain has to be increased in order to polarize the antenna. This kind of experiment will be explained in the Section 7.4 where the real FM broadcasting in Brussel has been scanned.
Using the signal generator a first test has been performed in order to check the correct behaviour. Different combinations of signals has been tested. The Figure

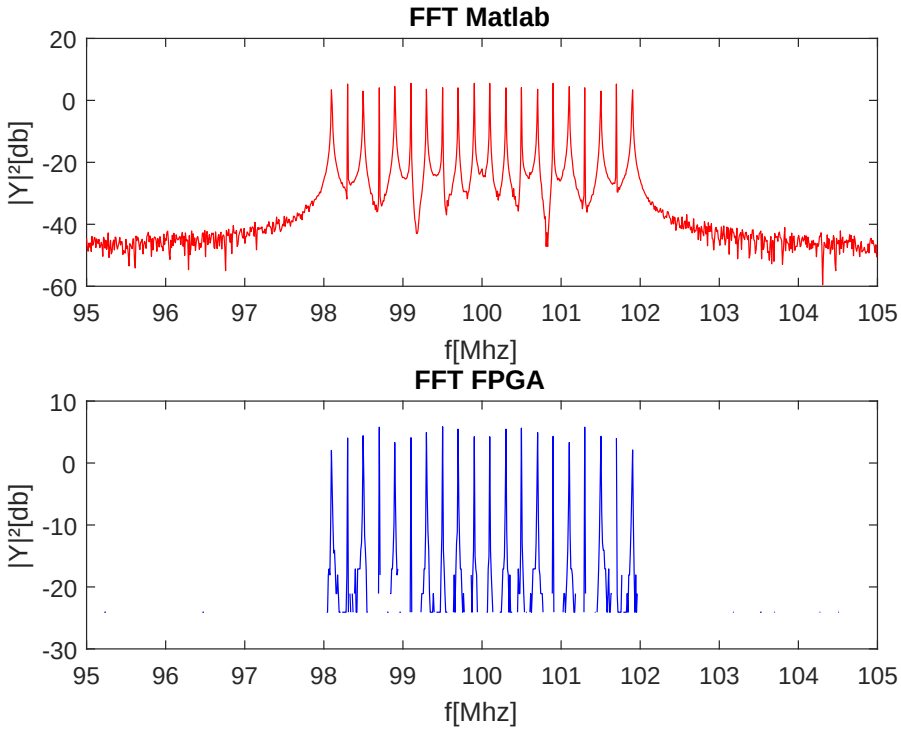

Figure 7.4: Comparison between FFT obtained from time-domain samples (on top) and FFT obtained with our FPGA implementation (bottom)

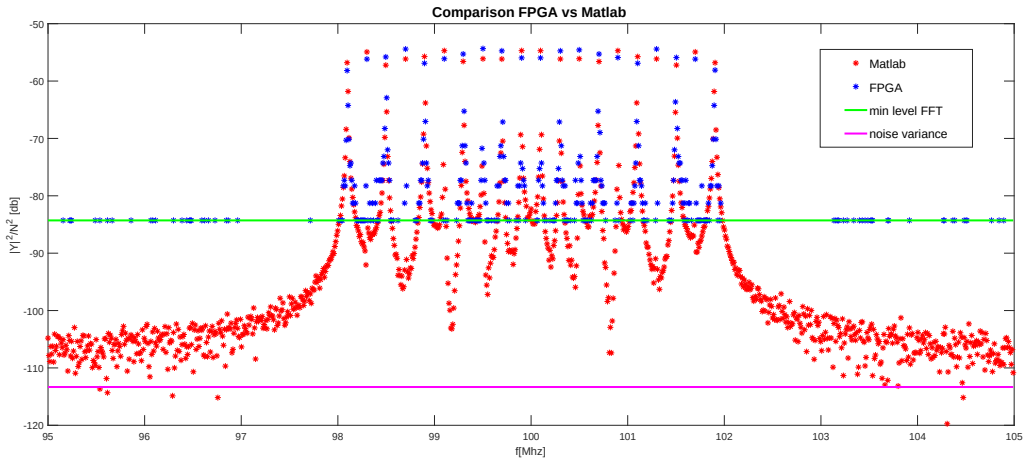

Figure 7.5: Comparison between FFT on Matlab, FFT obtained FPGA implementation and WBX noise variance level

7.6 shows a screenshot of the system output. The signal generator in this case has been settled to generate two different standard signals: Bluetooth and GSM.
The Bluetooth signal is transmitted at $402MHz$ and the GSM one roughly $50MHz$ apart. The USRP is retuning over a total bandwidth of 60 MHz. It can be seen that both signals have been detected. For this test the adaptive architecture and following system parameters have been used:

- $f_s = 10MHz$;
- $f_{start} = 400MHz$
- $M = 8$ (number of sub band bins);
- $n = 6$ (number of FFTs scanned starting from f_start);
- $\lambda^* = 8$;

The noise power to evaluate the λ^* value has been evaluated in the time domain at the RF-frequency of $400MHz$.

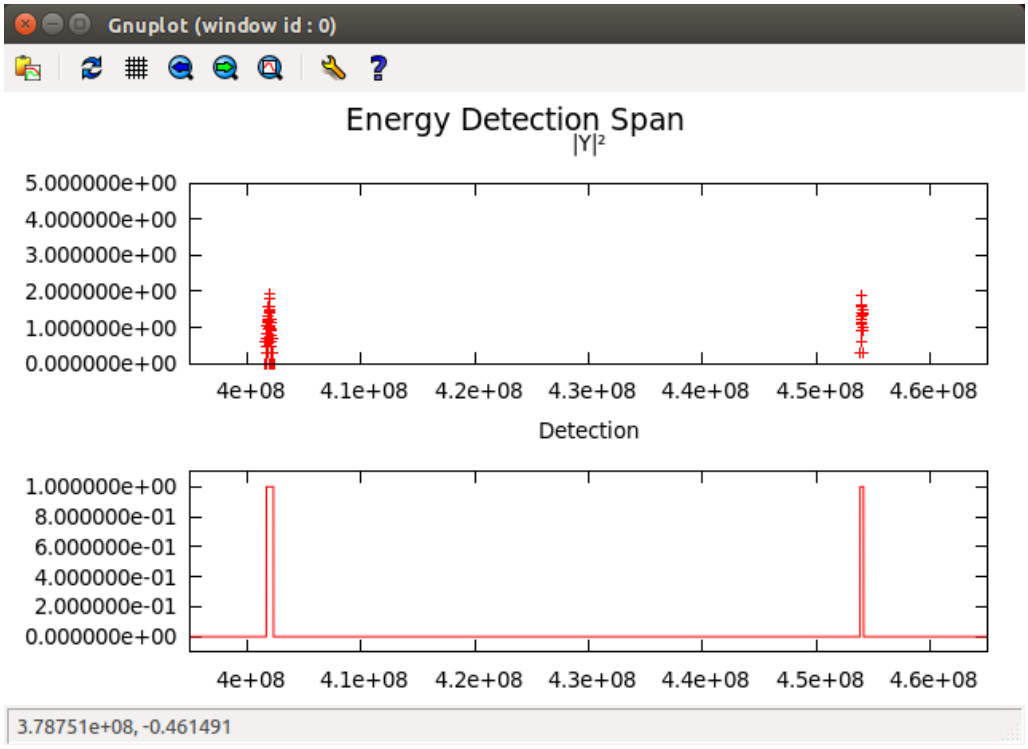

Figure 7.6: Spectrum scanning system output, (top) FFT result and (bottom) detection result

A second test performed for both architecture is sending one Bluetooth signal and a multi-carrier signal (21 carriers) over a bandwidth of $100MHz$. The Figure 7.7 shows the result of the test. The following parameters has been used for the experiment:

- $f_s = 10MHz$;
- $f_{start} = 400MHz$
- $M = 8$ (number of sub band bins);
- $n = 10$ (number of FFTs scanned starting from f_{start});
- $\lambda^* = 8$;

As can be noticed from Figure 7.7 both the Bluetooth and the single carriers have been detected.

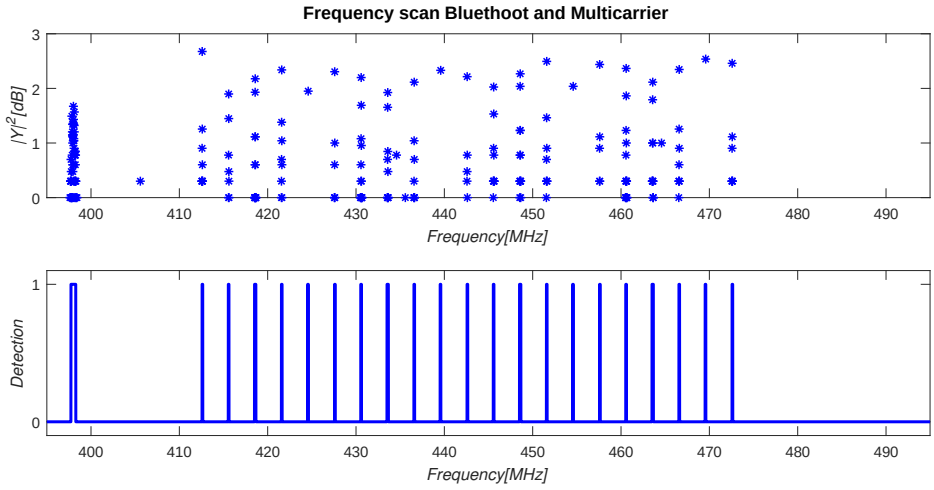

Figure 7.7: Spectrum scanning system output with Bluetooth and Multicarrier (21 carriers) signals, (top) FFT result ([dB]) and (bottom) detection result

7.3.3 Characterization of Energy Detection System

The spectrum sensing system performances are defined by two different probabilities:

- Probability of Detection: the probability that a signal, which is occupying a certain band, is correctly detected;
- Probability of False Alarm: the probability that a signal is detected inside a channel even if the signal is not present;

The two probabilities have been described in the Section 2.2 regarding the studies on the threshold methods. Some experiments have been performed on the two types of architectures to evaluate these kind of probabilities and characterize the system.

Probability of Detection

The RF signal generator has been settled to generate different standard signals changing the transmission peak power. From the USRP side the acquisition of samples and the evaluation of the result is repeated 100 times for each different peak power.
To have points enough to create a good P_d characteristic, the transmitting peak power step has been settled to $1dBm$.
Both the fixed and adaptive architectures have been tested in this way and the standard signals analysed are:

- Bluetooth $[BW = 1MHz]$;
- GSM $[BW = 270kHz]$;
- NADC $[BW = 24kHz]$;
- TETRA $[BW = 18kHz]$;
- WCDMA $[BW = 3.84MHz]$;

The signals have been transmitted at a central frequency equal to $402MHz$ because the frequencies around $400MHz$ is one particular case of study asked by BIPT, used for PMR transmissions.The sample frequency is settled to $10MHz$ and the receiving gain kept to $0dB$.
To compare the threshold selection the experiment has been performed changing the number of sub-band bins and the threshold value. Some results of these experiments are shown in the following figures:

- **Fixed Threshold**: Figure 7.8, Figure 7.9;
- **Adaptive Threshold**: Figure 7.10, Figure 7.11.

Other similar Figures representing other cases have been included in the Appendix A.

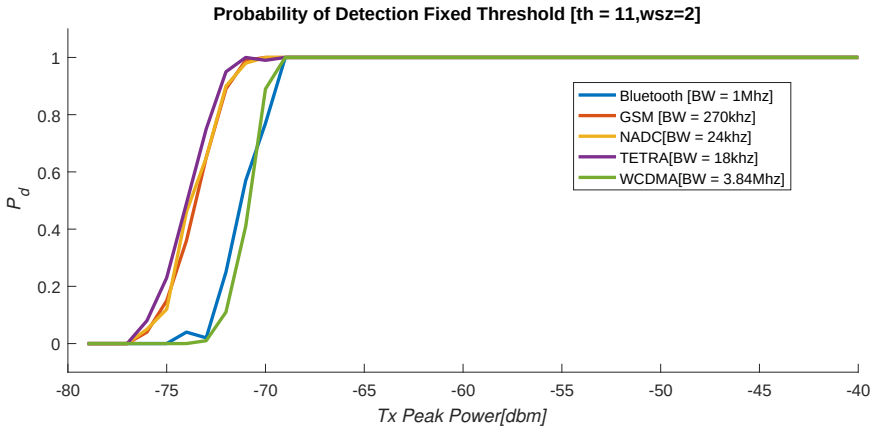

Figure 7.8: Probability of Detection ($M = 2$,$\lambda = 11$) - Fixed Threshold Architecture

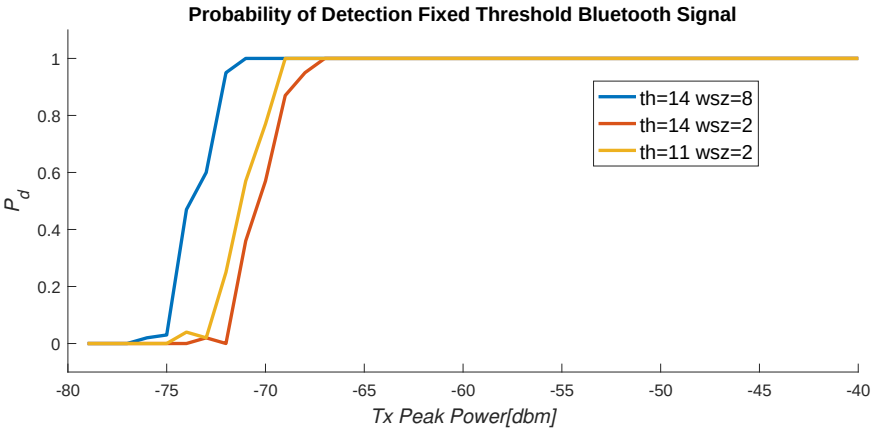

Figure 7.9: Comparison between Probabilities of Detection changing M and λ for a Bluetooth standard signal

The Figures 7.8 and 7.10 represents the identical settling for the fixed and the adaptive threshold cases. The overall peak power, where the two architectures show a P_d equal to 1, is around $-70dBm$. Both architecture detect signals more or less with the same peak power. One case which differentiates the two architectures is the

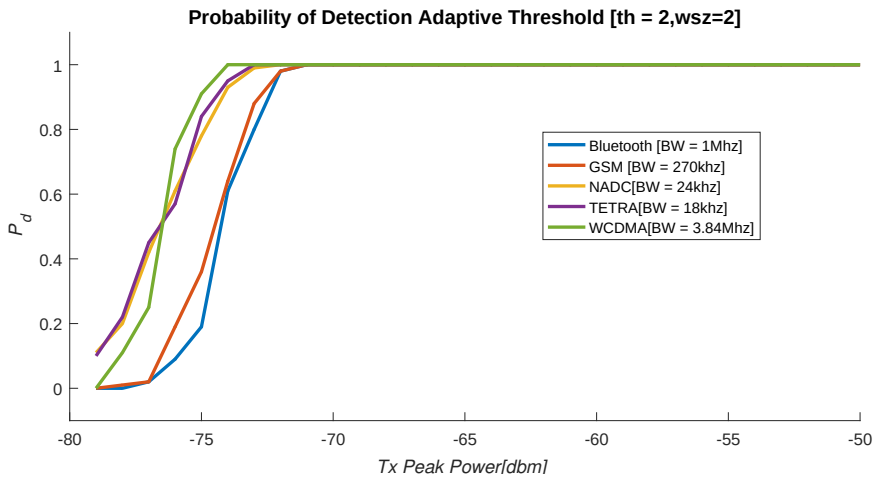

Figure 7.10: Probabilities of Detection ($M = 2$,$\lambda = 2$) - Fixed Threshold Architecture

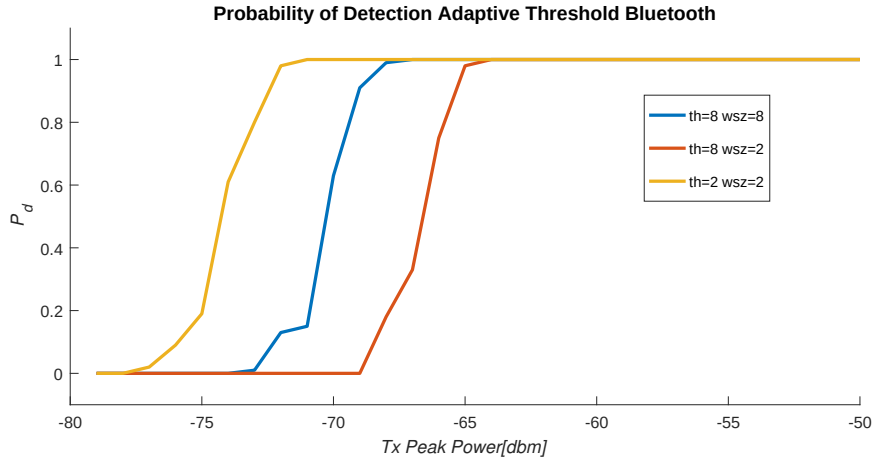

Figure 7.11: Comparison between Probabilities of Detection changing M and λ for a Bluetooth standard signal with Adaptive Threshold Architecture

P_d for the WCDMA signal. The adaptive threshold in this configuration can sense this standard signal up to $-75dBm$ respect around $-70dBm$ of the fixed threshold architecture. Figures 7.12 and 7.13 represent the comparison between the fixed and adaptive threshold with the same number of sub-band bins for the Bluetooth signal. When $M = 2$ the adaptive threshold architecture detects signal with less transmission peak power but in case of $M = 8$ the fixed threshold architecture has better performances than the adaptive one. Comparing also with the other standard signals the result is very similar. In particular for signal with lower bandwidth the

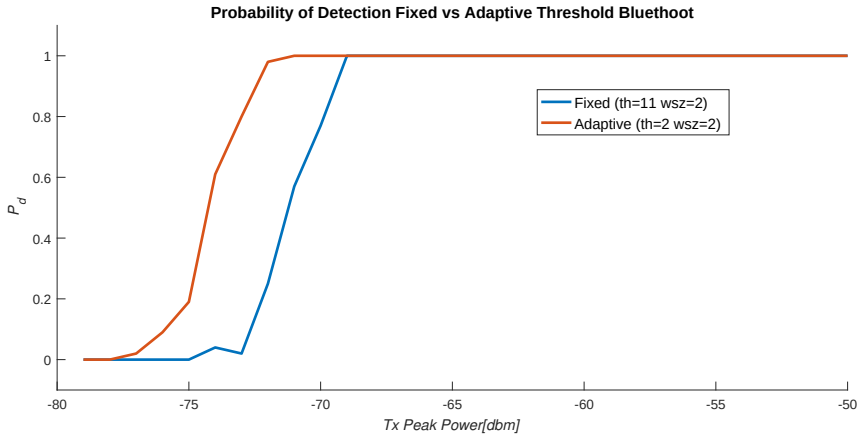

Figure 7.12: Comparison between Probabilities of Detection of Fixed vs Adaptive Threshold for a Bluetooth standard signal ($M = 2$)

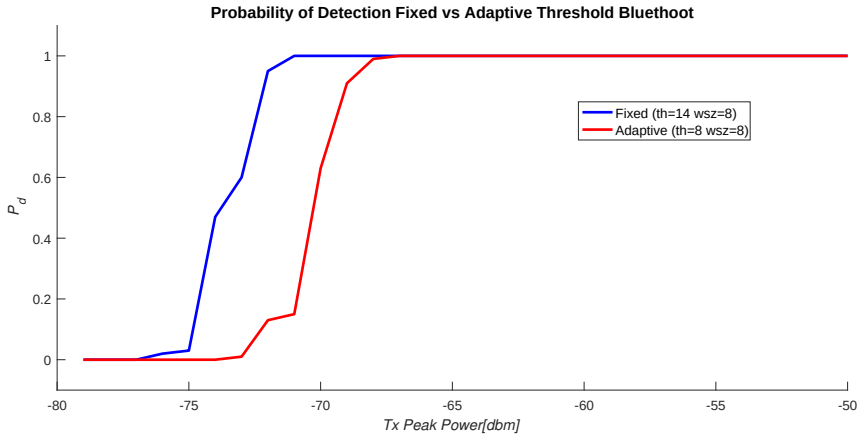

Figure 7.13: Comparison between Probabilities of Detection of Fixed vs Adaptive Threshold for a Bluetooth standard signal ($M = 8$)

difference is only of few dB.

Figures 7.9 and 7.11 show the P_d behaviour for a Bluehooth standard signal changing the threshold and the number of sub-band bins.
For the fixed threshold case can be noticed that if the number of sub-band bins is kept constant and the threshold is lowered the P_d becomes less than 1 at lower peak

power for a lower threshold. Increasing the threshold the characteristic is shifted to the right toward higher peak power.
On the other hand keeping constant the threshold and increasing the number of sub-band bins the graph is shifted toward left. This is obvious because the number of bins taken to compare with the threshold is greater.

For the adaptive threshold case a similar scenario happens. Keeping constant the number of sub-band bins and increasing the base threshold value the maximum peak power with $P_d = 1$ is lower, the characteristic is shifted toward right. Also for the adaptive threshold when the sub-band width increases also the performances will be increased.

The same comparison has been performed also with the other standard signals, the graphs of results obtained have been put on the Appendix A.

Probability of false alarm

The probability of false alarm has been evaluated using the received samples for the probability of detection. Based on the definition of P_f, the detection of others signals outside the bandwidth of the signal generated has been searched. Thanks to the minimum level that can be sensed by the FFT, the probability of false alarm is equal to zero for both the architectures.
One consideration, which has to be underlined, is that the P_f neglects the bin of the DC component and some other bins around it. This because is known that there is some noise induced by the WBX daughterboard and so there is the possibility to have some detections caused by these values. In particular when the gain of the receiving chain increases also these values become higher.

Gain vs Threshold

One case of study on the system performances is the gain versus the threshold. As mentioned before for the P_f, increasing the gain also the noise and the dc component change their values. This experiment had the aim to understand if was needed to increase the threshold value or adapt it based on the gain value.
For this test the signal vector generator was not used, and the single USRP without any antenna connected to the input port has been used. Some different frequencies among the WBX receiving frequency range are used and the gain is changed from 0 up to 30 dB.
The result of this experiment is that for both the architectures is not necessary to add some terms linked to the gain value. But in case of the adaptive threshold, increasing the gain more than $25dB$, the DC elimination module present on the chain

is not enough. This module eliminates only three bins around the DC component and the bins with a value different from zero are more than three. Overall for the adaptive threshold is suggested to use a gain less than $25dB$.

7.3.4 Spectrum Scan Speed

The Software part of the system has been tested scanning the whole bandwidth, allowed for the daughterboard, and finding the best *span* time which can be settled between two retuning commands. To perform this test the antenna has been connected to the device and it has been used a gain equal to $15dB$. The sample frequency instead has been changed to check the correct behaviour of the system. Reducing f_s the span time has to be increased in order to respect the latency time of the custom module implemented on the FPGA. These time values has been reported in the Table 7.2, Subsection 7.2.3.
If the f_s is greater than $1MHz$ the minimum values of *span* time which can be used are:

- **Fixed Threshold System**: **2ms**;
- **Adaptive Threshold System**: **2.5ms**.

If the sample frequency is less or equal to $1MHz$ the *span* time has to be increased. These results depend also on the Host CPU performance, in this case a laptop with 4Gb of RAM and a Linux OS (*Ubuntu 14.04 LTS*) has been used. Using these minimum values to scan the whole available WBX range of frequencies ($50MHz$ up to $2200MHz$), with a $f_s = 10MHz$ are needed about $535ms$ for one cycle. The detection data is refreshed every $535ms$.

7.4 FM broadcasting scan experiment

All the tests explained in the previous sections has been performed without connect an antenna to the USRP.
A practical experiment has been conducted on the FM broadcasting in Brussels. Using a database of all radio stations available in Brussels, the broadcastings detected have been compared with the transmission position.

The FM broadcast band usually is from 87.5 to 108.0 MHz and it is part of the VHF range of the radio frequency spectrum. Normally the FM frequency carriers are spaced of 200 kHz (0.2 MHz); so each channel is $200kHz$ wide and can pass audio and subcarrier frequencies up to 100 kHz[13].

The System parameters have been settled as following:

- $f_s = 1MHz$;
- $f_start = 89.5MHz$;
- $M = 8$ (number of sub band bins);
- $n = 9$ (number of FFTs scanned starting from f_start);
- $\lambda^* = 8$;
- $gain = 15dB$;

Setting the f_s to $1MHz$ the frequency resolution f_0 has been increased respect the previous experiments where the f_s was settled to $10MHz$.
The radio stations detected are shown on the Figure 7.14 and the respective transmission position is shown in the Figure7.15.

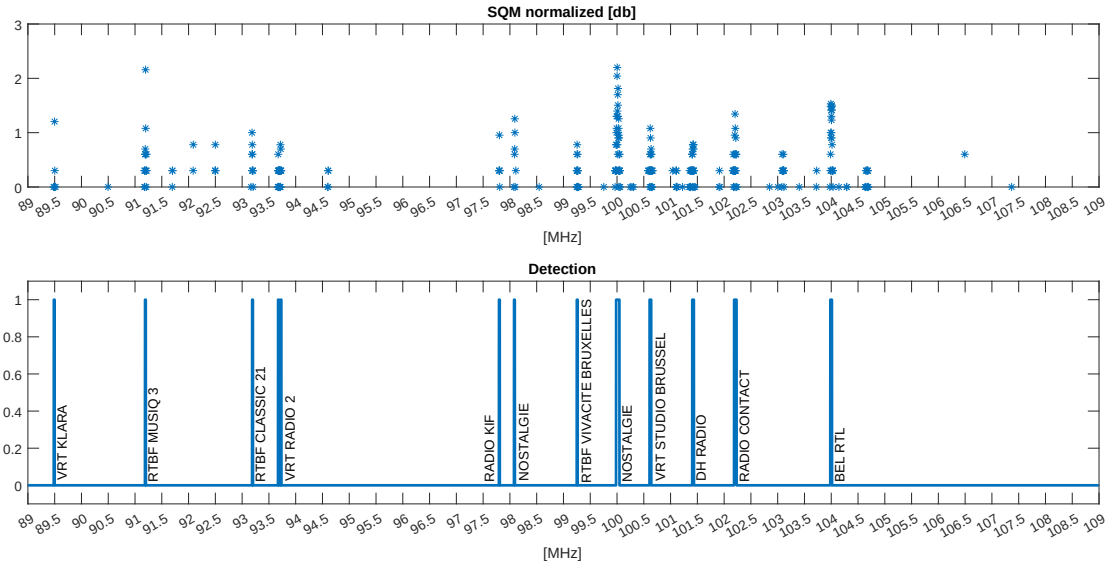

Figure 7.14: Spectrum scanning system output, (top) FM spectrum result and (bottom) detection result

Twelve radio stations have been detected from the BEAMS department at ULB in Brussels. From the Figure 7.14 can be noticed that in some parts of the spectrum there are some bins that are not detected, this is due by the fact there are few bins with values different from zero and moreover the antenna used to receive the FM radio is not designed for this range of frequencies. Only when the power transmission of the station is more strength the broadcast signal can be detected.

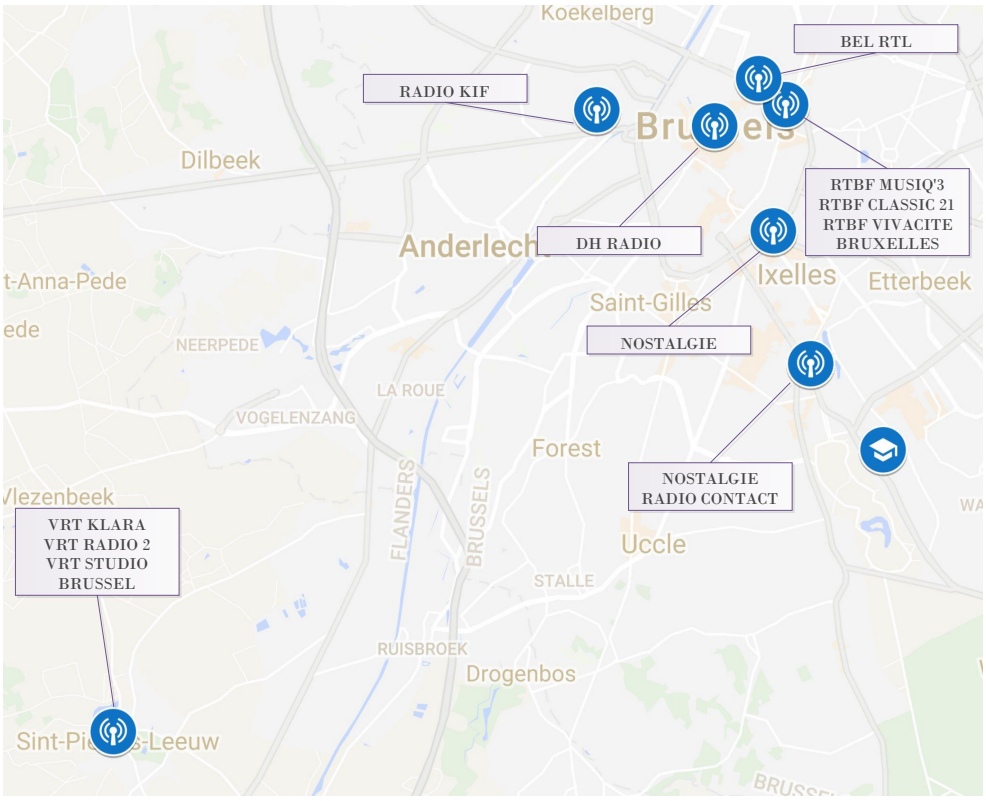

Figure 7.15: Map of FM stations positions detected respect the University position

Obviously the data plotted regards only samples of one entire spectrum scan. The test is performed with a continuous scan of the same piece of spectrum, the system has been left in running mode for several minutes and only some few software lags have been encountered, but the system continued work correctly.

Chapter 8

Conclusion

The thesis presented shows an opportunistic implementation of a spectrum scanner on a software define radio platform. By exploiting the USRP N210 features a mixed FPGA/software implementation has been designed, where the whole detection part is performed on the FPGA in order to alleviate the host CPU workload.
The system software part uses and takes advantage of the most advanced features of the USRP, making it possible to create a spectrum scanner at cost cheaper than traditional spectrum analyzers. Although some reduction of performances are present.
The overall features of the implementation presented are:

- *configurable total scanning bandwidth*;
- *a configurable USRP bandwidth (up to* $10MHz$*)*;
- *a configurable sub-bandwidth*;
- *an automatic or manual (fixed) threshold mechanism.*

The details of the system implementation performances and results have been explained in the Chapter 7, following a brief summary of most important results is given.
The automatic scanner system is able to re-tune its central frequency each ***2.5ms*** which means with about ***500ms*** the whole bandwidth of the WBX daughter board (*50MHz* up to *2200MHz*) can be scanned.
The overall minimum peak power of transmission which can be sensed is of *-70dBm* for different type of signals with a number of sub-band bins less than 8. The sample frequency of the USRP can be changed with also the threshold and M in order to refine the signal detection.
Some future improvements on the FPGA side could be performed, such as a module for the noise variance estimation in order to evaluate the base threshold value directly. Moreover, a complete FPGA image with both threshold mechanisms could

be created giving the possibility to select by software the threshold mechanism.
Future related work will be to integrate a GNSS chip on the USRP device, in order to retrieve the GNSS data and compare, by software, the detection result with a regulator provided database.
This spectrum scanner implementation was published as a "Temporary Document" in the COST IRACON (*Inclusive Radio Communication Networks for 5G and beyond*) Action and was presented in the *5th Technical meeting* of the COST IRACON Action in September 2017 at Graz (Austria).

Appendix A

System Characterization

A.1 Probabilities of Detection Fixed Threshold Architecture

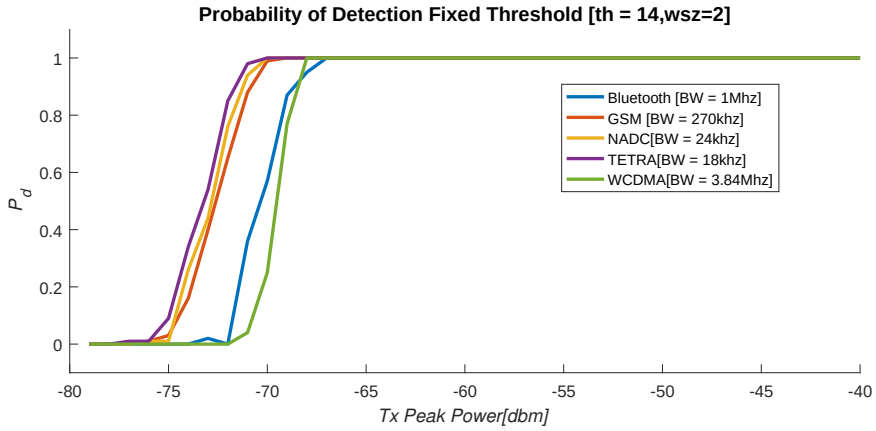

Figure A.1: Probabilities of Detection ($M = 2$,$\lambda = 14$) - Fixed Threshold Architecture

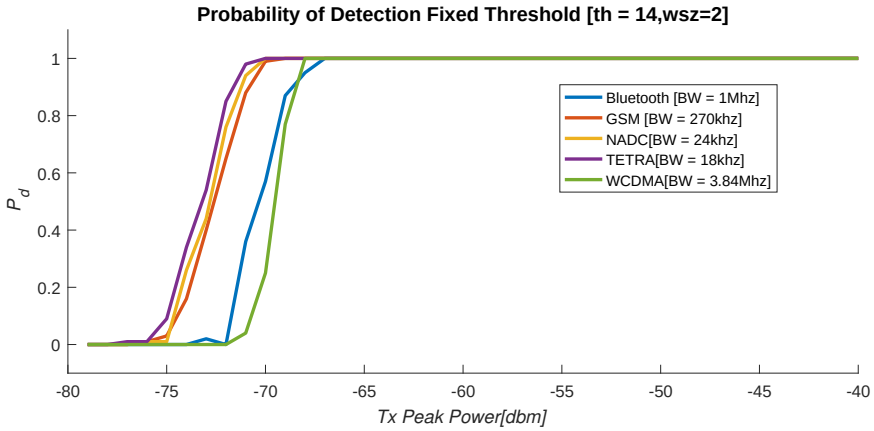

Figure A.2: Probabilities of Detection ($M = 8$,$\lambda = 14$) - Fixed Threshold Architecture

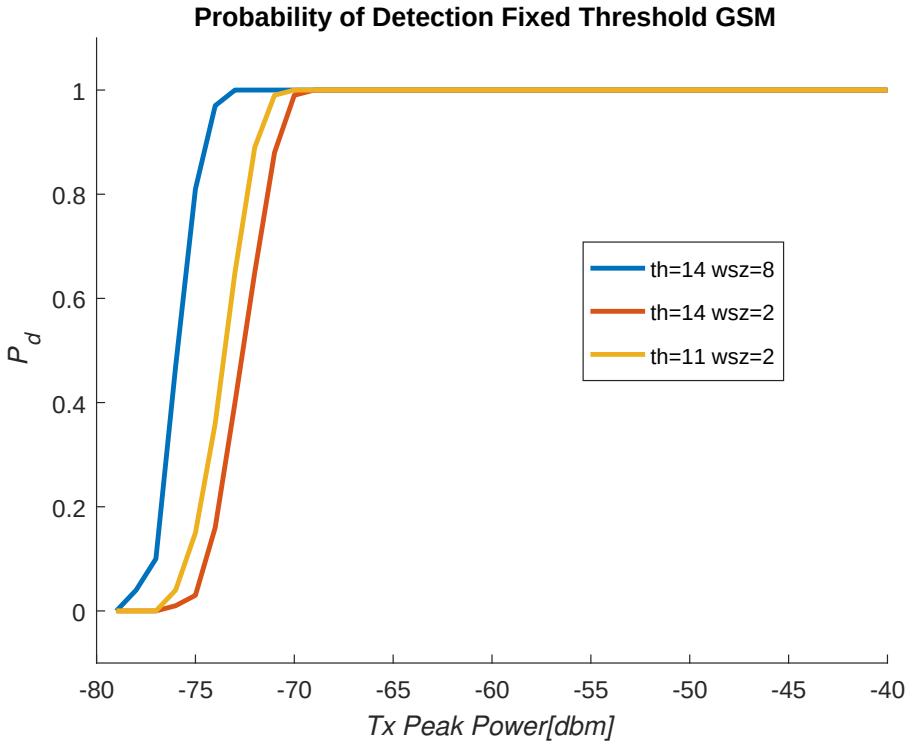

Figure A.3: Comparison between Probabilities of Detection changing M and λ for a GSM standard signal

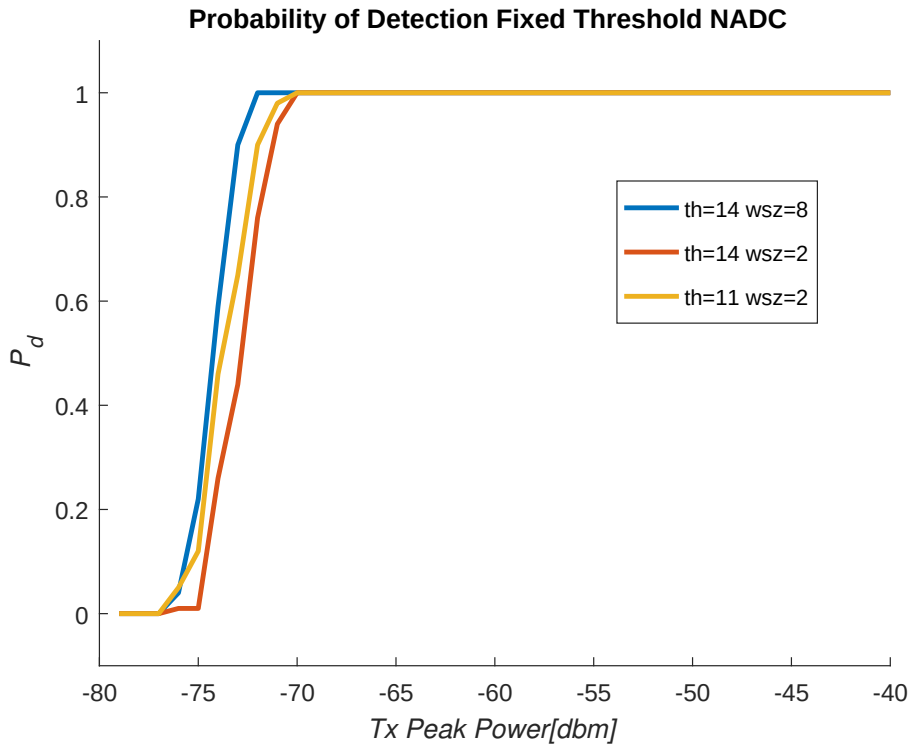

Figure A.4: Comparison between Probabilities of Detection changing M and λ for a NADC standard signal

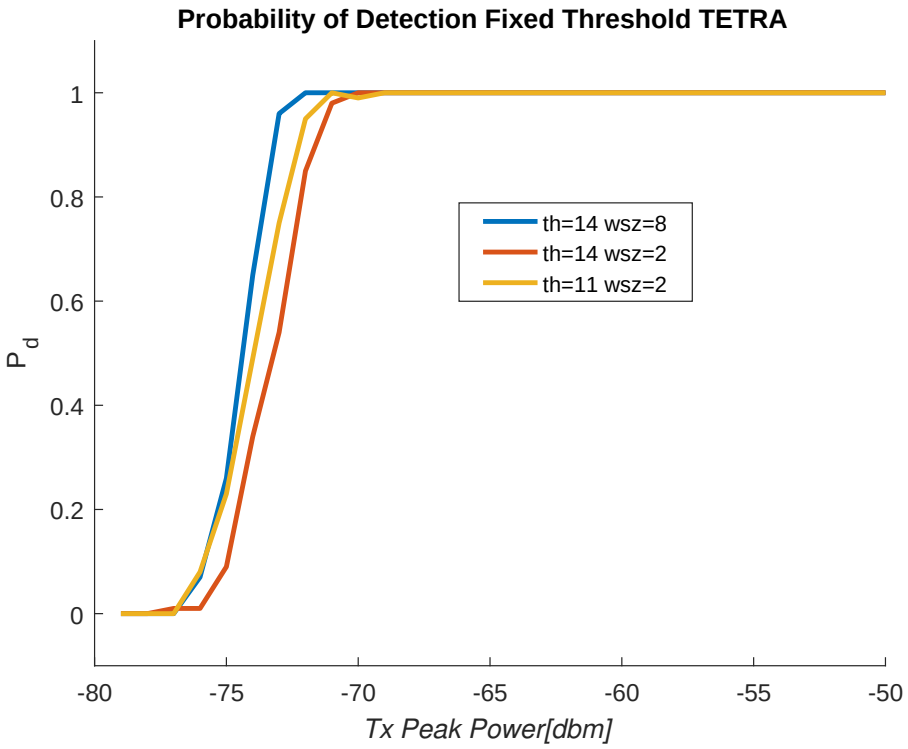

Figure A.5: Comparison between Probabilities of Detection changing M and λ for a TETRA standard signal

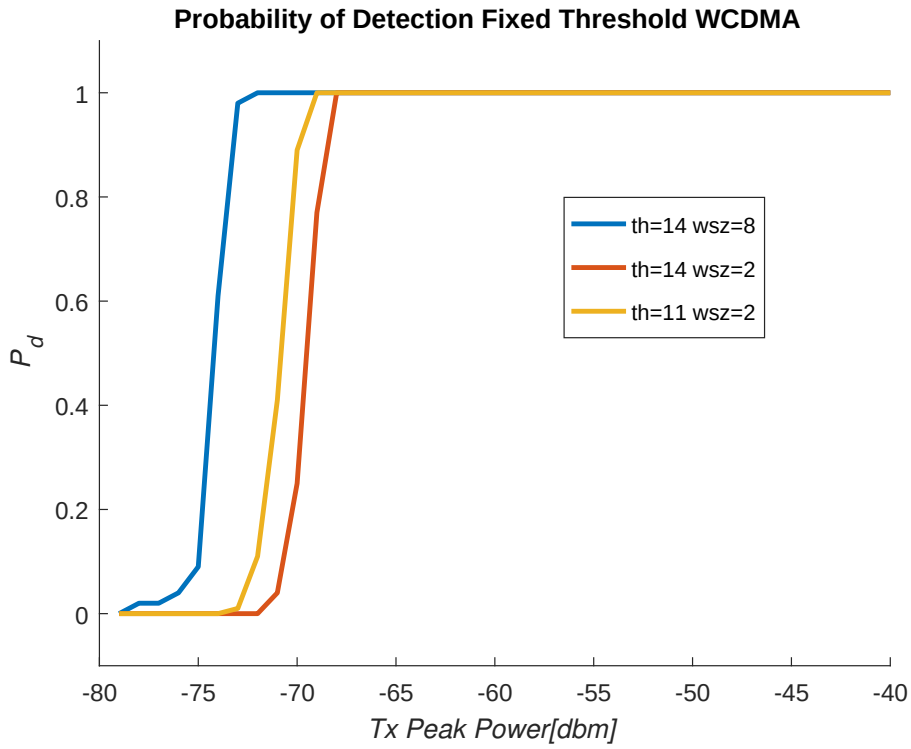

Figure A.6: Comparison between Probabilities of Detection changing M and λ for a WCDMA standard signal

A.2 Probabilities of Detection Adaptive Threshold Architecture

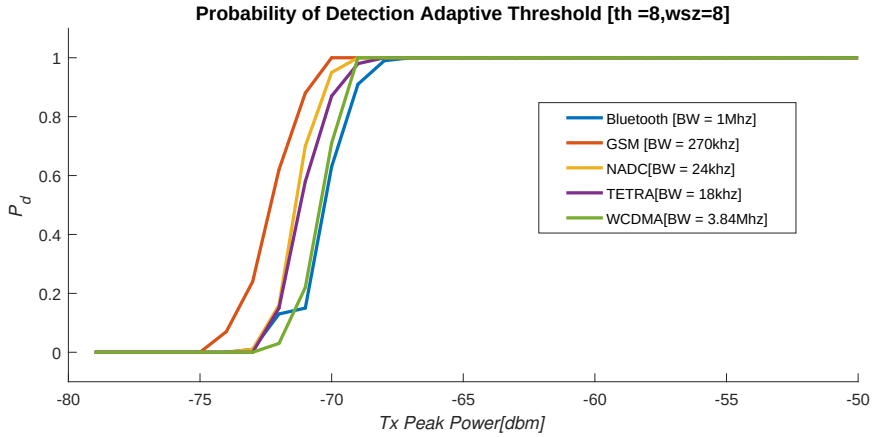

Figure A.7: Probabilities of Detection ($M = 8$,$\lambda = 8$) - Adaptive Threshold Architecture

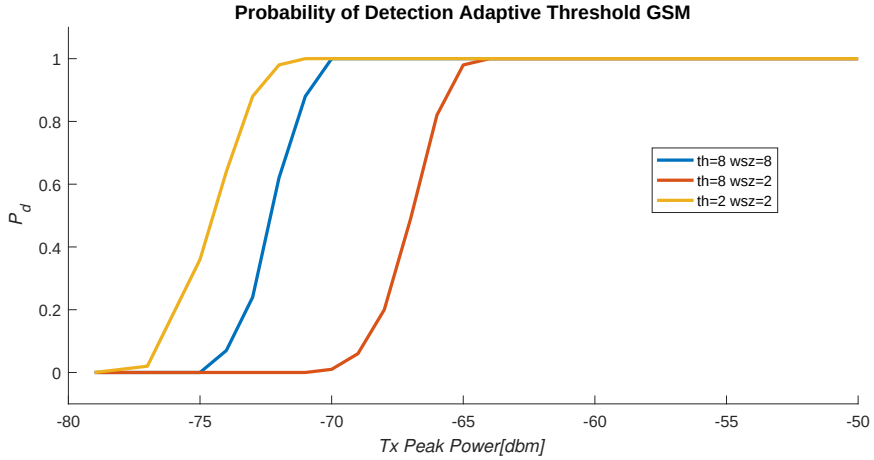

Figure A.8: Comparison between Probabilities of Detection changing M and λ for a GSM standard signal with Adaptive Threshold Architecture

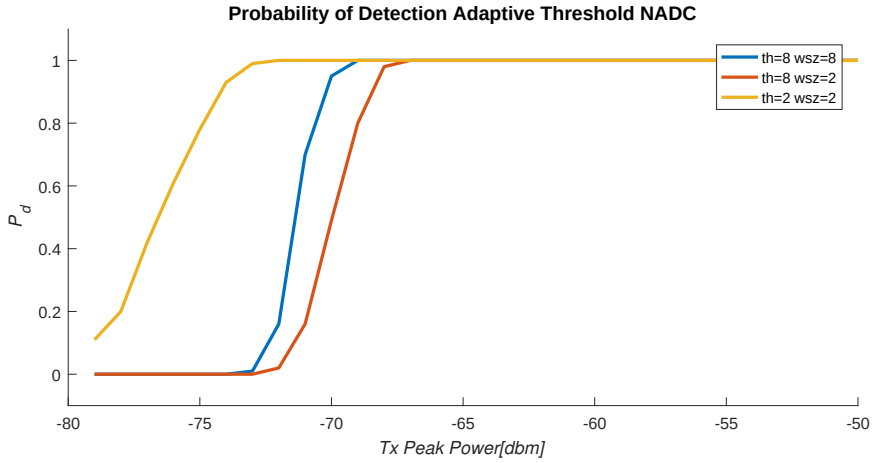

Figure A.9: Comparison between Probabilities of Detection changing M and λ for a NADC standard signal with Adaptive Threshold Architecture

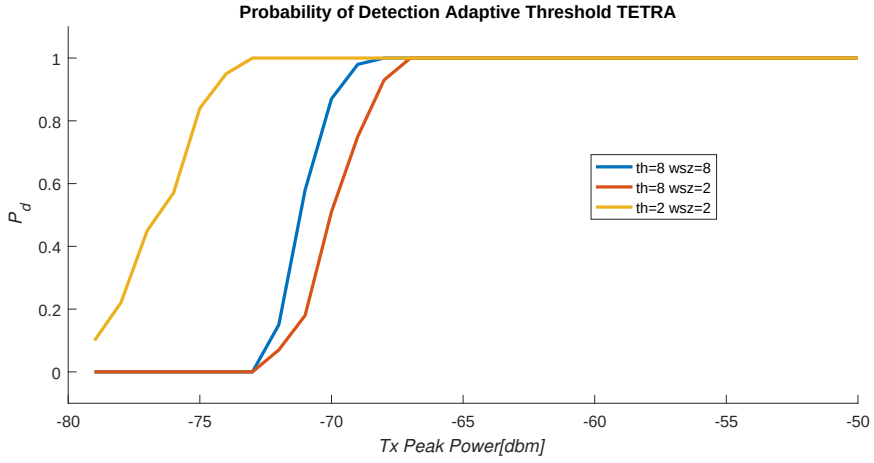

Figure A.10: Comparison between Probabilities of Detection changing M and λ for a TETRA standard signal with Adaptive Threshold Architecture

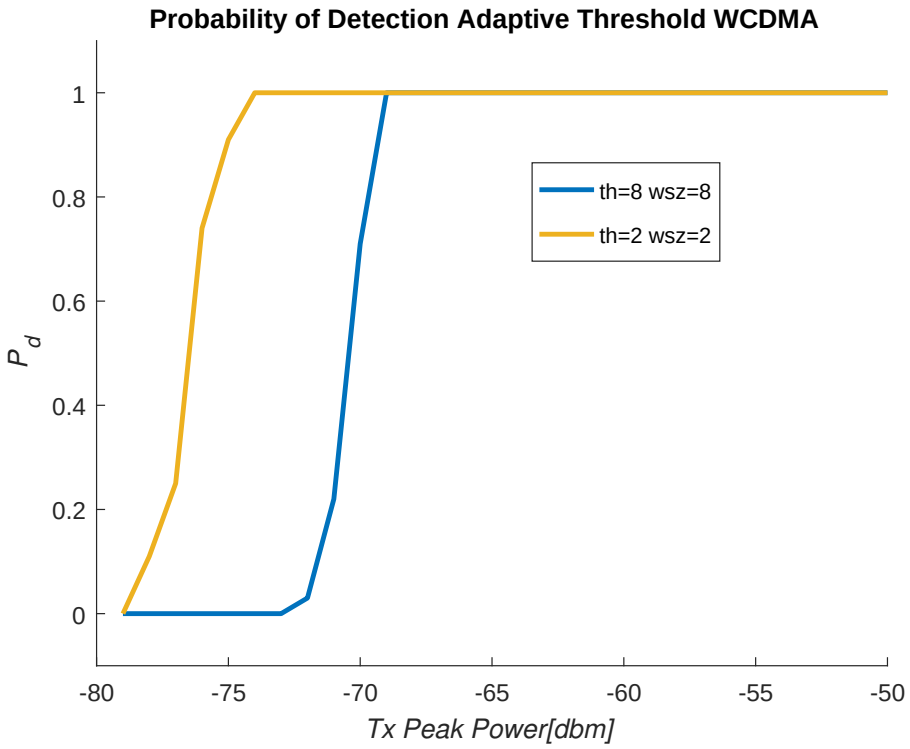

Figure A.11: Comparison between Probabilities of Detection changing M and λ for a WCDMA standard signal with Adaptive Threshold Architecture

Bibliography

[1] Ettus Research, "USRP n200/n210 networked series" 2012. [Online]: https://www.ettus.com/content/files/07495{_}Ettus{_}N200-210{_}DS{_}Flyer{_}HR{_}1.pdf

[2] J.-o. Jeong, "Hybrid FPGA and GPP Implementation of IEEE 802.15.4 Physical Layer" in *Master thesis*, p. 214, 2012.

[3] S. Atapattu, C. Tellambura, H. Jiang, "Energy Detection for Spectrum Sensing in Cognitive Radio" in *Energy Detection for Spectrum Sensing in Cognitive Radio*, 1st ed. Springer, 2014, capitolo 2, pp. 11–27. [Online]: http://link.springer.com/10.1007/978-1-4939-0494-5

[4] Kwang-Cheng Chen, P. Ramjee, *Cognitive Radio Networks*, 2009.

[5] H. Urkowitz, "Energy detection of unknown deterministic signals" in *Proceedings of the IEEE*, v. 55, n. 4, pp. 523–531, 1967. [Online]: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=1447503{%}5Cnhttp://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=1447503

[6] X. Ling, B. Wu, H. Wen, P.-h. Ho, Z. Bao, L. Pan, in "Adaptive Threshold Control for Energy Detection Based Spectrum Sensing in Cognitive Radios" v. 1, n. 5, pp. 448–451, 2012.

[7] N. Wang, Y. Gao, X. Zhang, "Adaptive spectrum sensing algorithm under different primary user utilizations" in *IEEE Communications Letters*, v. 17, n. 9, pp. 1838–1841, 2013.

[8] N. Wang, Y. Gao, "Optimal Threshold of Welch's Periodogram for Sensing OFDM Signals at Low SNR Levels" in *Wireless Conference (EW), Proceedings of the 2013 19th European*, pp. 1–5, 2013.

[9] S. Xie, L. Shen, J. Liu, "Optimal threshold of energy detection for spectrum sensing in cognitive radio" in *2009 International Conference on Wireless Communications & Signal Processing*, n. 2007, pp. 1–5, 2009. [Online]: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5371719

[10] M. Nasserbakht, in "Efficient Hardware Implementations" pp. 259–273, 2000.

[11] Ettus Research, "USRP Hardware Driver and USRP Manual" 2015. [Online]: http://files.ettus.com/manual/index.html

[12] S. Sequeira, R. R. Mahajan, P. Spasojević, "On the noise power estimation

in the presence of the signal for energy-based sensing" in *35th IEEE Sarnoff Symposium, SARNOFF 2012 - Conference Proceedings*, pp. 1–5, 2012.

[13] Wikimedia Foundation, in "WikipediA The Free Encyclopedia." [Online]: https://www.wikipedia.org/

[14] N. Wang, "Threshold Setting Algorithms for Spectrum Sensing in Cognitive Radio Networks by" Tesi di dottorato, Queen Mry University of London, 2014.

[15] Xilinx, "LogiCORE IP Fast Fourier Transform v7.1" Rapporto interno, 2011. [Online]: https://www.xilinx.com/support/documentation/ip{_}documentation/xfft{_}ds260.pdf

[16] Ettus Research, "UHD - General Application Notes." [Online]: https://files.ettus.com/manual/page{_}general.html

[17] I. Galal, M. E. A. Ibrahim, H. E. Ahmed, "Exploring Frequency Tuning Policies for USRP-N210 SDR Platform and GNU Radio" in *2013 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2013.

[18] GNU Radio Foundation, "GNURadio the free & open-source software radio eco-system." [Online]: https://www.gnuradio.org/

[19] Ettus Research, "Ettus Research USRP FPGA HDL Source" 2015. [Online]: https://github.com/EttusResearch/fpga

[20] Rohde & Schwarz, "R&S SMATE200A Vector Signal Generator." [Online]: https://www.rohde-schwarz.com/it/prodotto/smate200a-opzioni{_}63490-7556.html