AALTO-YLIOPISTON TEKNILLINEN KORKEAKOULU
AALTO UNIVERSITY SCHOOL OF SCIENCE AND TECHNOLOGY
Faculty of Electronics, Communications and Automation
Department of Micro- and Nanosciences

Kari Kokkinen

Implementation of Autocorrelation-based Feature Detector for Cognitive Radio

Thesis submitted for examination for the degree of
Master of Science in Technology.

Espoo, May 3^{rd} 2010

Supervisor	Professor Jussi Ryynänen
Instructor	D.Sc. Marko Kosunen

AALTO UNIVERSITY SCHOOL OF SCIENCE AND TECHNOLOGY	ABSTRACT OF THE MASTER'S THESIS

Author:	Kari Kokkinen		
Name of the thesis:	Implementation of Autocorrelation-based Feature Detector for Cognitive Radio		
Date:	3.5.2010	**Number of pages:**	57
Department:	Micro- and Nanosciences		
Professorship:	S-87 Electronic Circuit Design		
Supervisor:	Professor Jussi Ryynänen		
Instructor:	D.Sc. Marko Kosunen		

Emerging wireless systems demand more frequency bands in order to provide high data rate services. Most of the licensed frequency bands are underutilized, because of the rigid spectrum allocation. Cognitive radios aim to relieve the situation by identifying and exploiting the underutilized radio spectrum. A key task of the cognitive radio is spectrum sensing, which is intended to detect unoccupied frequency slots and licensed spectrum user transmissions.

This thesis presents an implementation of an autocorrelation-based feature detector for orthogonal frequency-division multiplexing (OFDM) based primary user signals. The autocorrelation-based detection algorithm is optimized in order to achieve power and area efficient hardware realization. The VHDL implementation is presented in detail and verified by simulations. After verification, the algorithm is implemented in a field-programmable gate array (FPGA) evaluation environment, and the performance is verified with measurements. An application-specific integrated circuit (ASIC) implementation is also realized in order to obtain comparable data of power consumption and area.

The algorithm implementation with DC offset compensation performed as predicted by simulations. The FPGA implementation requires 987 LUT flip-flop units, and the dynamic power consumption is 3.69 mW. The ASIC circuit implemented with 65 nm CMOS process occupies an area of 0.26 mm^2, and has power consumption of 1.02 mW.

Keywords: cognitive radio, spectrum sensing, autocorrelation

AALTO-YLIOPISTO	DIPLOMITYÖN
TEKNILLINEN KORKEAKOULU	TIIVISTELMÄ

Tekijä:	Kari Kokkinen		
Työn nimi:	Implementation of Autocorrelation-based Feature Detector for Cognitive Radio		
Päivämäärä:	3.5.2010	**Sivumäärä:**	57
Laitos:	Mikro- ja nanotekniikka		
Professuuri:	S-87 Piiritekniikka		
Työn valvoja:	Professori Jussi Ryynänen		
Työn ohjaaja:	TkT Marko Kosunen		

Nykyiset ja tulevat langattomat järjestelmät tarvitsevat yhä enemmän taajuuskaistoja uusille palveluille. Lähes koko käytettävissä oleva radiospektri on lisensoitu, mutta suurinta osaa lisensoiduista taajuuskaistoista ei hyödynnetä tehokkaasti tiukkojen käyttöehtojen vuoksi. Kognitiiviset radiot voivat helpottaa tätä ongelmaa, tunnistamalla vajaasti käytetyn kaistan ja ottamalla sen käyttöön häiritsemättä kaistan lisensoinutta käyttäjää. Kognitiivisten radioiden tärkein ominaisuus on löytää vapaat spektrin alueet sekä tunnistaa lisensoitujen käyttäjien lähetykset.
Tässä diplomityössä esitetään autokorrelaatioon perustuvan spektrinhavainnointialgoritmin suunnittelu ja toteutus. Algoritmi tunnistaa OFDM-signaaleihin perustuvia järjestelmiä. Toteutuksessa algoritmia on muokattu, ja laskentaa yksinkertaistettu tarvittavan pinta-alan ja tehonkulutuksen pienentämiseksi.
Implementaatio ja VHDL kuvaukset verifioidaan simulaatioilla. Algoritmi on toteutettu FPGA kehitysalustalle, ja sen toiminta on varmennettu mittauksilla. Algoritmi toteutettiin myös ASIC-piirinä tehonkulutus- ja pinta-alatietojen saamiseksi.
FPGA-toteutus tarvitsee 987 LUT flip-flop paria, ja sen tehonkulutus oli 3.69 mW. ASIC:na piiri toteutettiin 65 nm CMOS prosessilla pinta-alan ollesssa 0.26 mm^2 ja tehonkulutuksen 1.02 mW.

Avainsanat: kognitiivinen radio, spektrin havainnointi, autokorrelaatio

Preface

The work for this thesis was carried out at the Department of Micro- and Nanosciences of the Aalto University School of Science and Technology. This work has received funding from the European Community's Seventh Framework Programme under project named SENDORA.

I would like to thank especially D.Sc. Marko Kosunen and Professor Jussi Ryynänen for hiring me into the project, and for guiding me at each step. Special thanks to Vesa for always having the energy to explain everything to me from various subjects. Thank you to everyone else at the laboratory for helping me during the process. Thanks to William Martin for performing a language check and pointing out several inconsistencies.

My thanks also go to my friends, family, and relatives for encouraging me along the way and always asking of the latest process.

Espoo, May 3rd 2010

Kari Kokkinen

Table of Contents

Symbols and Abbreviations

P_d	Probability of detection
P_{fa}	False alarm rate
T_c	Cyclic prefix length
T_d	Delay length / useful symbol length
ADC	Analog-to-digital converter
AGC	Automatic gain control
ASIC	Application-specific integrated circuit
AWGN	Additive white gaussian noise
CAF	Cyclic autocorrelation function
CLB	Configurable logic block
CMOS	Complementary Metal Oxide Semiconductor
CP	Cyclic prefix
CR	Cognitive radio
CSD	Cyclic spectral density
DCR	Direct-conversion receiver
DFT	Discrete Fourier transform
DRC	Design rule check
DVB-T	Digital video broadcasting – terrestrial
FDM	Frequency-division multiplexing
FFT	Fast Fourier transform

FPGA	Field-programmable gate array
HDL	Hardware description language
I/O	Input/output
IF	Intermediate frequency
IFFT	Inverse fast Fourier transform
IMT-A	International mobile telecommunications advanced
ISI	Intersymbol interference
LAN	Local area network
LLR	Log-likelihood ratio
LTE	Long term evolution
LUT	Lookup table
OFDM	Orthogonal frequency-division multiplexing
PLL	Phase locked loop
PROM	Programmable read-only memory
PSK	Phase shift keying
PU	Primary user
QAM	Quadrature amplitude modulation
RAM	Random access memory
RF	Radio frequency
SD	Sequential detection
SNR	Signal-to-noise ratio
SU	Secondary user
VCO	Voltage-controlled oscillator
VHDL	VHSIC hardware description language
VHSIC	Very-high-speed integrated circuit
WiMAX	Worldwide interoperability for microwave access
WLAN	Wireless local area network

Chapter 1

Introduction

Today's wireless services have advanced significantly since the roll out of the conventional cellular systems. The demand for various kinds of mobile services have increased and higher data-rates are needed to meet users' requirements. However, there are no free frequency bands available as the whole spectrum is tightly allocated to existing systems. The radio spectrum is assigned to primary users by local authorities who restrict the spectrum users to specific bands of the spectrum, while prohibiting the usage by others. These frequency bands are even more limited by usage when they are divided into channels that use specific encoding and modulation schemes to prevent interference between users. The regulations on spectrum usage might work effectively for certain spectrum bands, but mostly they are only legacies of telecommunication history, where the spectrum has been allocated to the first comers, with television and radio broadcasts having the most preferable bands.

Primary users do not always use the spectrum reserved for them or it might be used inefficiently. Hence, the spectrum is underutilized, and yet, there is no free spectrum available for new users. A few spectrum bands have been released for new applications, as analog television broadcasts have been discontinued in various countries, but this will not solve the problem as a whole. According to extensive measurement campaigns, radio resources are utilized from 15 percent up to 85 percent depending on location, frequency band, and time of day [1]. Underutilized spectrum introduces fascinating possibilities for opportunistic usage. Cognitive radios, first introduced by Mitola [2], are proposed to solve the problem by exploiting this underutilized spectrum. In the United States, the unused broadcast television spectrum was opened for unlicensed use. However, in most countries the regulations on spectrum usage have to be altered in order to allow cognitive radio operation.

The cognitive type of radios are aware of their location, their network, and spectral environment. In addition, they can adapt to their environment to support more efficient spectrum utilization. Cognitive radios will be surveyed in more detail in Section 2.3. A key task of cognitive radios is to sense the spectrum, and then opportunistically exploit the free resources found. Several spectrum sensing techniques are introduced in Section 2.4.

In the work described in this thesis, an autocorrelation-based orthogonal frequency-division multiplexing (OFDM) signal detection algorithm, based on prior work by Chaudhari et al. [3], is implemented in a field-programmable gate array (FPGA) evaluation environment. OFDM-based signals are currently used in many current and emerging communication systems (e.g., WLAN, DVB-T, LTE, WiMAX, IMT-A), thus enabling the detection of OFDM signals is highly relevant. The WLAN stands for wireless local area network and is widely used by personal computers and smart phones for data transfer. The DVB-T is digital video broadcasting-terrestrial used in broadcasting the digital television signal on air. The LTE stands for long term evolution which is a project name for new high performance cellular mobile communication systems working as a stepping stone for fourth generation of radio technologies. WiMAX, meaning worldwide interoperability for microwave access, is a telecommunications technology providing wireless data transmission. IMT-A, international mobile telecommunications advanced, is the fourth generation of cellular wireless standards. It is the successor to 2G and 3G standards providing a wide range of data rates up to gigabit-speed.

The Chapter 2 is begun with mathematical derivation of common theory related to spectrum sensing algorithms. Then OFDM signaling system and cognitive radios itself are evaluated. Next the spectrum sensing challenge is considered, and common spectrum sensing algorithms are viewed. In the end of chapter 2 the design of digital implementation is introduced. The sensing algorithm itself and few modifications are presented in Chapter 3. The algorithm verification process is also described. Chapter 4 describes the sensing algorithm implementation, hardware description verification process, and implementation for an FPGA and an application-specific integrated circuit (ASIC). Finally, experimental measurements and results are shown in Chapter 5. In the end, whole work and results are concluded.

Chapter 2

Background

This chapter introduces the theory behind the detection algorithm, and derives equations needed to go further in the algorithm implementation. The algorithm is implemented in a field-programmable gate array (FPGA) development board, therefore digital design is covered in Section 2.1. Next, in Section 2.3, the concept of cognitive radio is introduced. Following this, the key challenge and topic of this work, the spectrum sensing is described in Section 2.4. The main issues and challenges that need to be considered in this work and in cognitive radios are presented in Section 2.3.2.

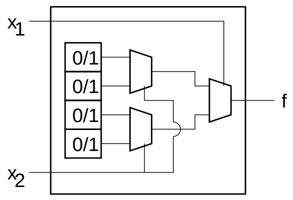
2.1 Digital implementation platforms used in this work

2.1.1 Field-programmable gate array

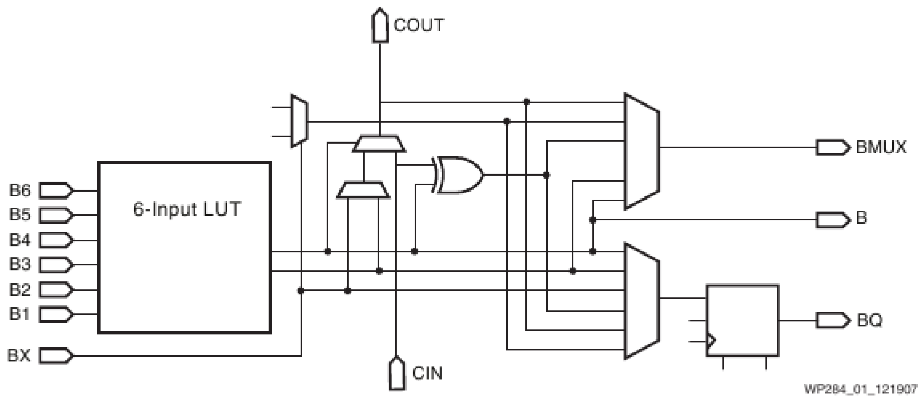
Field-programmable gate array (FPGA) is a programmable logic device that enables implementation of large logic circuits [4]. The general structure of FPGAs contains three main types of resources: configurable logic blocks (CLBs), input/output (I/O) blocks, and interconnection wires and switches. The logic blocks are arranged in a matrix, where interconnection wires are used as routing channels horizontally and vertically between rows and columns of CLBs. These routing channels contain wires and switches, thus allowing numerous ways to connect the CLBs. The I/O blocks enable connections between CLBs and pins of the package. Connections between interconnection wires and I/O blocks are also programmable.

Each configurable logic block in an FPGA contains digital logic, inputs, and outputs. The amount of logic used varies between manufacturers. The most commonly used CLB is a lookup table (LUT), which contains storage cells used to implement a small logic function. Each storage cell can hold a single logic value, either 0 or 1. This value can be obtained as an output of the logic cell. Many different sizes of LUTs can be used, where the amount of inputs determines the amount of variables used in a logic function. Figure 2.1(a) shows how a logic value f is obtained in a two-input LUT using input variables x_1 and x_2 as the multiplexer control signals.

In order to implement a design on the FPGA, the CLBs are individually configured and interconnection wires and switches are configured to connect or disconnect CLBs. The configurable logic block used in Xilinx Virtex-5 FPGAs contains a 6-input LUT, associated with muxes, logic, and a flip-flop, shown in Figure 2.1(b) [5]. Modern FPGAs can contain up to hundreds of thousands CLBs. Sophisticated development software is needed to configure the FPGAs. The manufacturer specific development software take logic design as an input and then output a bitstream for configuring the FPGA.



(a)



(b)

Figure 2.1: (a) Circuit for a two-input LUT (b) Xilinx Virtex-5 FPGA configurable logic block including a 6-input LUT associated with additional logic [5].

2.1.2 Application-specific integrated circuit

A digital application-specific integrated circuits (ASICs) consist of rows of logic gates connected by wires. The connecting wires are located between logic gate rows in specific routing channels. ASICs are commonly used for a full-custom design, where the circuit is created for a specific purpose and can be designed in a more detailed manner.

Various kind of logic gates can be used in the ASIC. They are pre-designed and collected in a library available to the designer. Standard cells are commonly used in situations where the designer does not need complete flexibility for the layout of each individual transistor, as part of the design effort can be avoided by using standard cells.

2.1.3 ASIC versus FPGA

Both ASIC and FPGA designs have their advantages and depending on the final product either one can be more suitable. FPGAs commonly have a faster time-to-market, as no layout, mask processing or other manufacturing steps are needed. Their design cycle is simpler due to the software that handles much of the routing, placement, and timing. FPGAs are reprogrammable, therefore, they can be reprogrammed on the field, if needed.

ASICs have full-custom capabilities as the devise is manufactured to design specifications. Accordingly, smaller form factor and higher internal clock speeds, compared to FPGAs, can be obtained. Additionally, lower unit costs can be obtained with very high volume designs.

2.1.4 Digital design flow

The digital design flow is similar for both an ASIC, and an FPGA, but the resulting circuit structures are different, and different tools are used for implementation. In this work, a Xilinx FPGA is used, and therefore, only Xilinx FPGA tools are discussed. A typical digital design flow, and the tools used in it are represented in Figure 2.2. The design process contains four main steps; problem defining, VHDL entry, synthesis, and layout. First, the designer defines the problem, and considers what operations the circuit needs to perform. The initial design process to characterize the problem, is the most important part of the design. The next step is to create a VHDL description of the circuit based on the theoretical design in the first step. Once the VHDL descriptions are ready, they need to be verified by simulations. Mentor Graphics' Modelsim is used in this work for simulating the VHDL descriptions. Onwards from this design phase, the ASIC and FPGA implementations branch off from each other.

Let us first consider the ASIC implementation. After verifying that the VHDL descriptions are functioning properly, they can be synthesized. VHDL descriptions are entered into Synopsys Design Compiler, as well as, design constraints for area, speed, and operating conditions. Operating conditions include used supply voltage levels and temperature conditions that both affect the speed of the circuit. The synthesis tool will generates a gate-level netlist of the circuit, using the logic cell library supplied by the process vendor. The netlist describes the connections between different cells to accomplish the desired functionality.

Subsequent to each process step, it is important to verify that the design functions as it was originally intended and that the design constraints are met. Functional and timing verification can be performed dynamically by simulating the synthesized netlist either statistically or with an estimated delay data or with means of static timing analysis. In static timing analysis, included in Design Compiler, the propagation delays for each path in the circuit are computed, and the path delays are compared to the given timing constraints. More advanced static timing analysis can be done using Synopsys Primetime. Static verification of functionality can be performed using Synopsys Formality, which compares the logical functionality of two netlists. Often static verification is considered to have better verification coverage

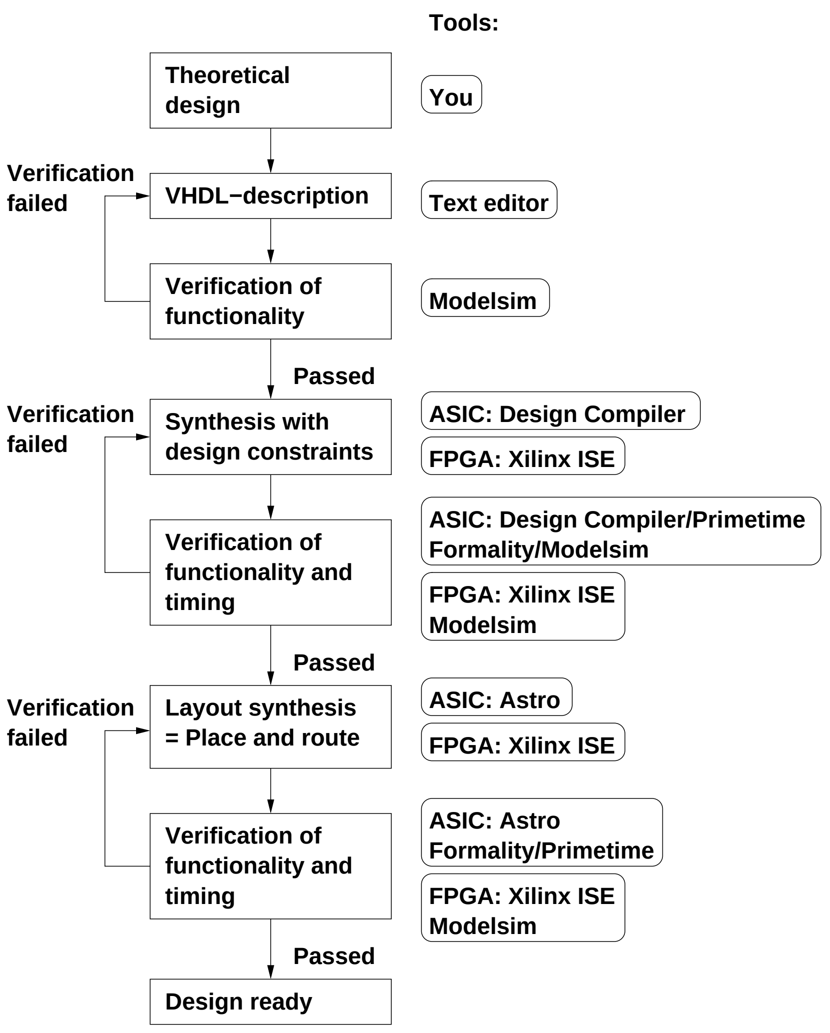

Tools:
Theoretical design
You
Verification failed
VHDL−description
Text editor
Verification of functionality
Modelsim
Passed
Verification failed
Synthesis with design constraints
ASIC: Design Compiler
FPGA: Xilinx ISE
Verification of functionality and timing
ASIC: Design Compiler/Primetime Formality/Modelsim
FPGA: Xilinx ISE Modelsim
Passed
Verification failed
Layout synthesis = Place and route
ASIC: Astro
FPGA: Xilinx ISE
Verification of functionality and timing
ASIC: Astro Formality/Primetime
FPGA: Xilinx ISE Modelsim
Passed
Design ready

Figure 2.2: A digital design flow.

compared to verification based on simulation, because the result of simulation depends on the simulation input, whereas the static functionality verification ensures logical equivalence.

After the synthesized circuit has passed the verification tests, it is ready for layout synthesis. The layout is generated using Synopsys Astro as the layout synthesizer tool. In this phase, the clock tree is generated. In clock tree generation the input clock signal is buffered in order to balance the clock delays for different blocks. The layout tool provides more accurate estimates of path delays compared to the logic synthesizer, because the load of the wires connecting the cells is known. This knowledge enables creating a clock tree with balanced delays that ensures accurate synchronization of synchronous logic elements. After the layout is completed, the obtained functionality and performance has to be verified. At this phase, a design rule check (DRC) should be performed. After the successful tests, the circuit is ready to be submitted to the circuit manufacturer.

In the FPGA approach, a netlist is generated by the Xilinx ISE synthesis tool. The netlist describes various logic gates and interconnections between them. The implementation tool maps logic gates and interconnections suitable for FPGA implementation. This is known as a mapping tool which combines the logic gates into groups fitting the lookup tables in the FPGA, and then generates the LUTs. The place and route is performed to assign the LUTs into specific CLBs and to program all the switches using routing matrices to connect the CLBs. Finally, the implementation process is complete, and the tool generates routing matrices describing the final status of switches. The routing matrices are used to generate an FPGA configuration bitstream, defining the ones and zeros corresponding to the closed and opened switches. This bitstream is used to configure the FPGA either by directly programming the FPGA through a connecting cable or by downloading the bitstream into a PROM which configures the FPGA. The FPGA is now ready to perform the operations specified in the design entry.

2.2 Mathematical theory related to autocorrelation-based sensing

A fast Fourier transform (FFT) algorithm and an autocorrelation function are covered in this section. Both are commonly used in digital signal processing applications.

2.2.1 Fast Fourier transform

The FFT algorithms are known as efficient algorithms to compute the discrete Fourier transform (DFT) by exploiting symmetry and periodicity properties of the phase factor [6]. The DFT is important algorithm used in many applications of digital signal processing, including linear filtering, correlation analysis, and spectrum analysis. The discrete Fourier transform of a sequence $x(n)$ of length N is calculated as

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-\frac{2\pi i}{N} nk}, \qquad k = 0,1,\ldots,N-1. \tag{2.1}$$

The DFT can also be calculated using a correlation method or by solving simultaneous linear equations but they are significantly slower approaches. The discrete Fourier transform separates a sequence of values in time into components of different frequencies. This operation is useful in different areas but computing it directly from the original definition is slow and impractical. Therefore, the FFT is used to compute the same result more efficiently.

One of the common FFT algorithms was introduced by Cooley and Tukey [7]. It is a recursive algorithm that breaks the DFT of size N into smaller DFTs of sizes N_1 and N_2. The simplest form of the algorithm, where the DFT is divided into two interleaved DFTs of size $N/2$ that are Fourier transforms computed from the even-indexed samples $x_{2m}(x_0,x_2,\ldots,x_{N-2})$ and odd-indexed samples $x_{2m+1}(x_1,x_3,\ldots,x_{N-1})$. These two results are combined to produce the Fourier transform of the entire sequence. The calculation of the

FFT is expressed as

$$X(k) = \sum_{m=0}^{\frac{N}{2}-1} x_{2m} e^{-\frac{2\pi i}{N}(2m)k} + \sum_{m=0}^{\frac{N}{2}-1} x_{2m+1} e^{-\frac{2\pi i}{N}(2m+1)k}. \tag{2.2}$$

Using the original definition, calculating the DFT of N samples takes $O(N^2)$ arithmetical operations, while the FFT can compute the same result in only $O(NlogN)$ operations. This results in substantial speed gains, especially, when long data sequences are used as N can be thousands or millions. In such cases, it is possible to reduce the computation time by several orders of magnitude, and the improvement is nearly proportional to $N/log(N)$.

2.2.2 Autocorrelation

Autocorrelation of a random signal is described as correlation between values of the same signal at different points in time. Therefore, autocorrelation can be used to distinguish repeating patterns in signals. A quite basic application of autocorrelation is determining tempo for musical beat or pitch detection.

Mathematically the autocorrelation can be described using a continuous autocorrelation function. The continuous autocorrelation function $R_{ff}(\tau)$ is a cross-correlation of function $f(t)$ with itself, at lag τ is shown in Equation 2.3.

$$R_{ff}(\tau) = f^*(-\tau) \otimes f(\tau) = \int_{-\infty}^{\infty} f(t+\tau) f^*(t) dt \tag{2.3}$$

in which f^* represents the complex conjugate and $\otimes$ represents convolution.

For a discrete process the autocorrelation function can be estimated as shown in equation 2.4.

$$\hat{R}(k) = \frac{1}{(n-k)\sigma^2} \sum_{t=1}^{n-k} [X_t - \mu][X_{t+k} - \mu] \qquad k < n \tag{2.4}$$

in which μ and σ^2 are the mean and the variance, respectively.

2.2.3 Orthogonal frequency-division multiplexing

Orthogonal frequency-division multiplexing (OFDM) is widely used for digital communication systems. The best known OFDM-based systems are IEEE 802.11 Wireless LAN, DVB-T, LTE and WiMAX. The key advantages of OFDM is that it enables high data rates over frequency selective communication channels of low complexity. Most common applications include wireless networking, broadband internet access, digital television and audio broadcasting.

OFDM is based on frequency-division multiplexing (FDM) as a digital multicarrier modulation. It enables data transfers to use multiple frequency channels simultaneously without mutual interference. It is accomplished by using closely-spaced orthogonal sub-carriers to carry the data. The data is divided in parallel data streams that are assigned to different sub-carriers.

The OFDM signal is created by inserting T_d symbols to an inverse fast Fourier transform (IFFT). If $C(0)$,$C(1)$,...,$C(T_d - 1)$ are T_d complex quadrature amplitude modulation (QAM) or phase shift keying (PSK) symbols, then the outputs of the IFFT are

$$c(n) = \frac{1}{\sqrt{T_d}} \sum_{k=0}^{T_d-1} C(k) e^{\frac{j2\pi nk}{T_d}}, \quad n = 0, \ldots, T_d - 1, \tag{2.5}$$

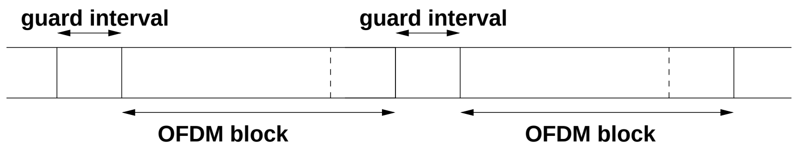
where n is a discrete time index, and k is a discrete frequency index. In consequence, T_d also denotes the number of symbols in an OFDM data block. Let T_c be the number of symbols in CP, now T_c symbols $c(T_d - T_c), \ldots, c(T_d - 1)$ are added in front of the block as CP and thus we have a complete OFDM block. There can be several OFDM blocks in one transmitted OFDM frame. The baseband OFDM signal is described in Equation 2.6.

$$v(n) = \sum_{k=0}^{N-1} X_k e^{j2\pi kn/T}, \qquad 0 \leq n < T \tag{2.6}$$

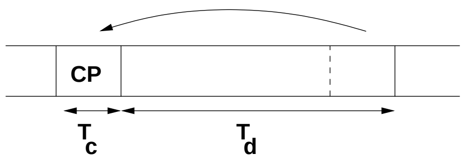
where N is the amount of sub-carriers, which are modulated using M alternative symbols, X_k are the data symbols of the sub-carriers and T is the OFDM symbol time. Individual sub-carrier is modulated using phase-shift keying (PSK) or quadrature amplitude modulation (QAM) at low symbol rates. In this way, it is possible to maintain equal data-rates compared to single-carrier modulation in the same bandwidth.

Multicarrier systems have robust performance against severe channel conditions. Single carrier systems are sensitive to attenuation on long transfer wires, narrowband interference and multipath fading. These non-idealities can be partially compensated with complex equalization filters. Because OFDM uses low symbol sample rate, a guard interval between symbols is affordable, therefore, intersymbol interference (ISI) can be averted. Also channel equalization is simpler as OFDM consists of many slowly-modulated narrowband signal compared to one rapidly-modulated wideband signal in single carrier case.

Intersymbol interference is avoided by adding a guard interval of length T_c between the OFDM blocks as illustrated in Figure 2.3(a). During this guard interval a cyclic prefix (CP), usually the length of T_c, is transmitted. CP is a repeat of the end of the OFDM symbol that is copied in front of the OFDM block as shown in Figure 2.3(b). This property of OFDM signals is quite convenient for detection algorithms and it is taken advantage of, for example, in cyclostationary feature detection [8].



(a)



(b)

Figure 2.3: OFDM data blocks: (a) A guard interval is in between of OFDM data blocks to prevent ISI. (b) OFDM data block, in which a cyclic prefix T_c copied from the end of the block is transmitted during the guard interval

2.3 Cognitive radio concept

Cognitive radio concept is developed to solve the problem of underutilization of almost completely allocated frequency space. This concept was introduced at 1999 by Mitola [2].

Cognitive radios are intended to have self-awareness and simple intelligence. A generic cognitive radio architecture is presented in Section 2.3.1. Cognitive radios are supposed to change their operating band, if the currently used band becomes too occupied or the primary user (PU) takes the band into use. The most important feature of the cognitive radios is the ability to sense the spectrum and find out, if some portion is left underutilized. After the spectrum is sensed, they can make an independent decision based on detection statistics [9] whether to take a certain band into use or not. Since cognitive radios are not the primary users or licensed users, they must ensure not to interfere with PUs' signals. This sets stringent sensitivity requirements for the spectrum sensing of cognitive radios.

Cognitive radios are not tied to certain signaling protocols and can adapt to their environment by changing their transmitter parameters [8] to different signaling systems. Depending on the network and cooperation with other cognitive devices, they can exchange information about their location and environment [10]. Cognitive radios can cooperate with other cognitive radios and share information between each other.

Prior to the introduction of cognitive radios, reconfigurability was used in radio development. A common radio communication system is implemented in hardware. In a software-defined radio most of the required hardware and required transmitter and receiver algorithms are implemented in software, thus high reconfigurability is achieved [11]. Cognitive radio is the next step where simple intelligence is added by allowing the radio to sense its environment, track changes and react upon its findings.

2.3.1 Hardware of cognitive radio

The architecture for a generic cognitive radio transceiver is shown in Figure 2.4(a) [8]. The cognitive radio transceiver unit consists of the radio frequency (RF) front-end and the baseband processing unit. A control bus is used in controlling each component to make the radio adaptive to the RF environment. The RF front-end first amplifies the received signal, then mixes it to a lower band and, finally, the analog signal is converted to a digital signal. The baseband processing unit modulates or demodulates and encodes or decodes the signal depending on whether a signal is transmitted or received. The baseband signal processing unit is similar to common transceivers, but the RF front-end is specifically designed to suit the need of the cognitive radio.

Cognitive radio transceiver is required to be capable of sensing over a wide spectrum range and preferably in real time. The wide spectrum range is accomplished by using RF hardware technologies like wideband antennas, power amplifiers, and adaptive filters [12]. The RF hardware is needed to be able to tune in to any part of the frequency spectrum. The main components of the cognitive radio RF front-end are shown in Figure 2.4(b) [8] and are as follows:

- *RF filter*: The RF filter selects the desired operating band by bandpass filtering the received RF signal.

- *Low noise amplifier*: The LNA amplifies the received signal without adding remarkable amount of noise.

- *Mixer*: The mixer is used to mix the received signal with locally generated RF frequency and then convert it to the baseband or the intermediate frequency (IF).

- *Voltage-controlled oscillator (VCO)*: The VCO generates a signal at a specific frequency depending on the control voltage. The generated signal is then used to convert the incoming signal frequency to the baseband or intermediate frequency.

- *Phase locked loop (PLL)*: The PLL makes sure that the signal of VCO is locked accurately on the specific reference frequency.

- *Channel selection filter*: The channel selection filter selects the desired channel and rejects adjacent channels.

- *Automatic gain control (AGC)*: The AGC is used to keep the gain or output power level of an amplifier constant over a wide range of input signal levels.

- *Analog-to-digital converter (ADC)*: The ADC converts the analog input signal to a digital signal.

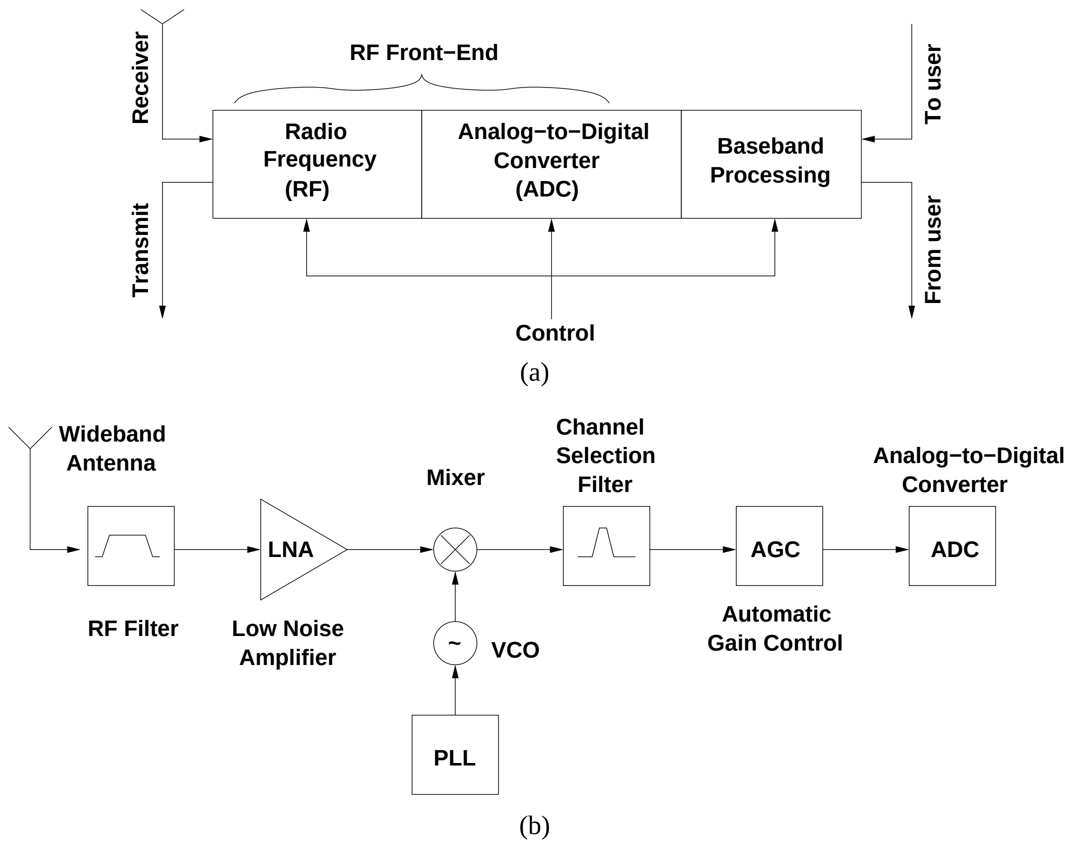


Figure 2.4: Physical architecture of the cognitive radio [8]: (a) Cognitive radio transceiver and (b) wideband RF/analog front-end architecture.

2.3.2 Cognitive radio challenges

Common receivers are capable of processing narrowband signals with low complexity and low power processors for digital signal processing. In order to utilize any opportunity, the cognitive radio terminals need to process significantly wider bands. PUs are entitled to claim their frequency bands anytime when cognitive radio is operating at that band. In order to prevent interference to and from PUs, cognitive radio needs to identify the presence of a PU as quickly as possible and vacate the spectrum immediately. Consequently, detection algorithms need to sense the PU during a certain time period. This sets stringent requirement for the sensing method, forming a design challenge for cognitive radios.

Challenges of cognitive radio can be listed as follows:

- designing an efficient spectrum sensing algorithm
- implementation complexity as cognitive radio has requirement of frequency and system flexibility
- operation in multiple secondary user environment to not compromise the signaling channel
- multipath fading and shadowing of user signals

- designing a resource efficient cooperative scheme for spectrum sensing and information sharing between cognitive radios
- robustness
- power consumption

2.4 Spectrum sensing theory

Cognitive radios have to sense the spectrum to detect opportunities, and reliably find out if PU signals are present. The spectrum has to be sensed accurately to find out even weaker primary user signals. At the same time, cognitive radios have to respect the needs of the PUs and not to interfere with them. Therefore, the spectrum sensing method has to be very sensitive and distinguish PU signals below the noise floor.

Many different spectrum sensing methods have been introduced [13]. Some methods work for specific signals while others are more generic. Depending on the knowledge of the signal under detection, better performance is usually obtained when detecting specific signals while more generic methods are good for rough estimates on channel usage.

One of the more generic spectrum sensing methods is the energy detection. Energy detectors have been introduced nearly half a century ago by Urkowitz [14], yet they are still researched and new ways to enhance their efficiency are published. Energy detectors do not need any information about the signal under detection, therefore they are able to detect wide variety of signals. However, they can not differentiate primary users' signals from noise. Other signal detection method exploits the statistical properties of PU signals to detect them [15]. One of these methods is cyclostationary spectrum sensing [16]. Cyclostationary feature detectors can differentiate noise from primary users' signals. Algorithms sensing even more specific signals by matched filtering have been studied from the 1960s, and Middleton introduced a generalized matched filter[17]. Matched filters deliver optimal detection performance, however, each signal under detections needs a specific matched filter. For this reason, matched filter is not widely used. Other way to enhance the detection probability is the cooperation between cognitive radios [18][19].

2.4.1 Spectrum sensing challenges

Reliable spectrum sensing has several issues that need to be taken into consideration. Some of the main problems include high hardware requirements needed to operate efficiently with large bandwidths and high resolution. Also signal propagation issues need to considered such as shadowing and severe multipath fading. Depending on the cognitive network in use, there can be weak PUs that may not be detected properly. This is known as a hidden node problem. The hidden node problem is depicted in Figure 2.5, it shows how node A is not aware of node C and vice versa. In consequence, nodes A and C might transmit simultaneously, and node

B would receive corrupt signal. Cooperation improves detection performance for users who are far away from each other, as the other secondary user (SU) might have a better chance of detecting the PU transmission than the other, since a single sensor might suffer from different kinds of interference. In addition, the primary users transmitting a spread spectrum signal or using frequency hopping, where the power of PU signal is distributed over a wide frequency, are difficult to detect.

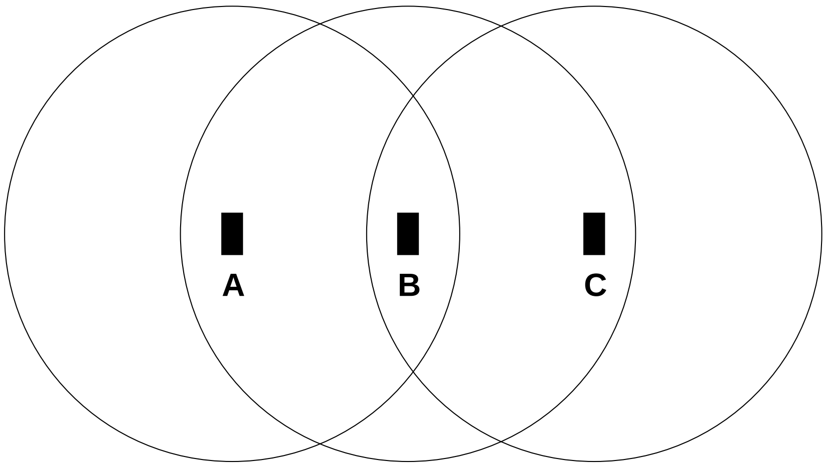


Figure 2.5: Hidden node problem.

The OFDM signal is very resistive against single channel fading, because multiple channels are used to transmit the signal. However, the fading caused by shadowing due to obstacles in the signal path affects the wave propagation. It has an effect on both primary user signals, which makes it harder to detect PU signals. As a solution, it is proposed to use multiple user cooperative cognitive radio networks [20] that can also alleviate the hidden node problem. Also high speed signal processors are needed for performing computationally demanding signal processing tasks with relatively low delay to minimize interference caused to PUs.

2.4.2 Statistical modeling of the signal

In order to make the decision whether a PU user is using the spectrum or not, a statistical model is needed for the PU signal for the detection, and then consider the situation without PU. Let us assume a simple received signal is modeled as

$$y(n) = s(n) + w(n), \tag{2.7}$$

where $s(n)$ is the signal under detection, $w(n)$ is the additive white Gaussian noise (AWGN) sample, and n is the sample index. When there is no PU signal present $s(n) = 0$. Detection algorithms calculate a detection statistic to be compared with a detection threshold:

$$\rho_x > \lambda_X \tag{2.8}$$

in which ρ_x and λ_X are the detection statistic and detection threshold, respectively. Now we can make a decision of the spectrum usage by comparing ρ_x to a fixed threshold value λ_X depending on the detection scheme in use. This can be expressed as a hypotheses comparison

$$\mathcal{H}_0 \quad : \quad y(n) = w(n), \tag{2.9}$$

$$\mathcal{H}_1 \quad : \quad y(n) = s(n) + w(n), \tag{2.10}$$

where hypothesis $\mathcal{H}_0$ denotes that no PU is present and hypothesis $\mathcal{H}_1$ that there is PU present.

The detection performance can be expressed with two probabilities: probability of detection P_d and false alarm rate P_{fa}. P_d means the probability of detecting a signal on a spectrum band when it really exists. Therefore, a higher P_d equals better performance. P_d can be obtained as

$$P_d = \Pr(\rho_X > \lambda_X | \mathcal{H}_1). \tag{2.11}$$

P_{fa} is the probability for the test to falsely indicate that the spectrum is in use when it really is not. P_{fa} can be expressed as

$$P_{fa} = \Pr(\rho_X > \lambda_X | \mathcal{H}_0). \tag{2.12}$$

P_{fa} should be kept as low as possible to prevent underutilization of spectrum.

2.4.3 Energy detection

Energy detector is simple to design and facilitates the use of low complexity hardware. It is a very generic sensing method as it does not need any knowledge on the primary users' signal. The signal is detected by comparing the output of the energy detector with a threshold defined by noise energy [14]. The energy detector can detect power levels at certain frequency bands, however it has no means to distinguish between signals from different systems or differentiate interference from a primary user signal and noise. It can only tell whether an energy on a signal band exceeds an estimate of noise energy. To achieve good detection performance, the noise power level has to be known accurately a priori which is difficult to achieve in practice.

Energy detectors do not work efficiently when detecting spread spectrum signals [21]. However, energy detectors are most suitable for making coarse estimates on channel usage or working side by side with other more advanced detection methods [22].

Using the model introduced in Equation 2.7, a decision metric for the energy detector can be expressed as

$$M = \sum_{n=0}^{N} |y(n)|^2. \tag{2.13}$$

The AWGN can be modeled as a zero-mean Gaussian random variable with variance σ_w^2, i.e. $w(n) = \mathcal{N}(0,\sigma_w^2)$, and the signal, as a zero-mean Gaussian variable $s(n) = \mathcal{N}(0,\sigma_s^2)$. Under these assumptions, the decision metric M follows chi-square χ^2 distribution with $2N$ degrees freedom χ_{2N}^2 and therefore, it can be modeled as

$$\mathcal{H}_0 \quad : \quad M = \frac{\sigma_w^2}{2}\chi_{2N}^2, \tag{2.14}$$

$$\mathcal{H}_1 \quad : \quad M = \frac{\sigma_w^2 + \sigma_s^2}{2}\chi_{2N}^2. \tag{2.15}$$

Now the detection probabilities P_{fa} and P_d for energy detector can be calculated as

$$P_{fa} \quad = \quad 1 - \Gamma\left(L_f L_t, \frac{\lambda_E}{\sigma_w^2}\right), \tag{2.16}$$

$$P_d \quad = \quad 1 - \Gamma\left(L_f L_t, \frac{\lambda_E}{\sigma_w^2 + \sigma_s^2}\right), \tag{2.17}$$

where λ_E is the decision threshold, and $\Gamma(a,x)$ is the incomplete gamma function as given in [23](see Section 6.5). The threshold in energy detector-based sensing algorithms depends on the noise variance, and even small noise estimation errors can cause significant performance loss [24].

2.4.4 Cyclostationary feature detection

Cyclostationary feature detectors [16] exploit periodicity in the signal or it's statistics like mean and autocorrelation or they can be intentionally induced to assist in spectrum sensing. These periodicities are called cyclostationary features. The cyclostationary feature detectors use a cyclic autocorrelation function to detect the signals. They can differentiate between different signaling systems and can also operate well on low SNR signals. The cyclostationary feature detectors' good performance makes them more attractive, but they are more complex to implement in hardware and require more computing than energy detectors.

Let us consider a cyclostationary process $x(t)$. It is second-order if its mean and autocorrelation are periodic in time [25]. For a cyclostationary process, the cyclic autocorrelation function (CAF) is nonzero for a set of cyclic frequencies $alpha \neq 0$. OFDM signal exhibits conjugate cyclostationarity, thus we can use the conjugate cyclic autocorrelation function at cyclic frequency α that is calculated as

$$R_x^\alpha = \frac{1}{M}\sum_{n=0}^{M-1} x(n)x^*(n-\tau)e^{\frac{-j2\pi\alpha n}{M}}, \tag{2.18}$$

in which, τ is the lag, and M is the sample length.

2.4.5 Matched filter detection

Matched filter detection is an optimum method for spectrum sensing [26]. When a matched filter has an impulse response matched to the input signal, it maximizes the output SNR and it does not matter if the input signal is corrupted with additive white Gaussian noise (AWGN). The main advantage is the short time needed to achieve certain probability of a false alarm. However, the matched filter input signal has to be demodulated. Therefore, complete knowledge of the primary user signal characteristics such as bandwidth, operating frequency, modulation type and order, pulse shaping, and frame format. The main drawback is the need for different receivers for different signal types, therefore, cognitive radio implementation is not feasible as the complexity of the sensing unit would be impractically large [21].

2.4.6 Autocorrelation-based detection

Detection is based on the value of the autocorrelation coefficient of the received signal. In this detection method, the system is identified by the time delay value (T_d), which should provide a nonzero autocorrelation value, if the received signal is from a particular radio system and about 0, if the received signal is noise. The decision making is based on the knowledge of statistical distribution of the autocorrelation coefficient. Once the value of the autocorrelation coefficient is computed, the decision can be performed so that a pre-defined false alarm rate specification of detection is fulfilled.

Autocorrelation-based detection can effectively detect PU signals under the noise floor, and is able to identify specific OFDM-based signaling systems. The implementation is relatively simple, since no FFT is done to the input signal. This limits the detection to the baseband frequency, but resource gain from omitting a FFT is significant for a low power implementation. This detection method was chosen for implementation, and is described in more detail in Section 3.1.

2.4.7 Cooperative detection

A single cognitive radio can obtain decent channel usage information by itself. However, in order to increase detection performance, cognitive radios can also share sensing information with each other. This is called cooperative detection. Cooperation alleviates problems due to noise uncertainty, fading and shadowing. It is also a way to alleviate the hidden node problem. There are three scenarios for cooperative sensing: centralized sensing, distributed sensing and external sensing.

In centralized sensing, a central unit collects all the sensing information from the cognitive radios. Then it analyzes the data and identifies the available spectrum. Finally, the unit sends the information to the cognitive radios or controls the cognitive traffic directly. Depending on the central unit, the data can be collected as hard or soft decisions. The hard decision is a binary result that tells whether a PU is present or not. The soft decision is a more accurate test statistic depending on the sensing algorithm in use. When a large number of users is collaborating, the transmitted soft decisions can require significant portion of the bandwidth. Therefore, it has to be chosen to use either hard decisions [27] or limit the soft decision accuracy accordingly to reduce the bandwidth required to transmit the decisions between cooperating radios.

In distributed sensing, cognitive radios share the sensing information among each other but they can still make independent decisions whether to utilize a spectrum band or not. Compared to centralized sensing this method is easier in practice since no central infrastructure is required. Several network topologies are proposed for cooperative sensing [28]. Two user cooperation is compared by Ganesan and Li in [18] and in [20], cooperation extended to multiple users is discussed. The benefits of cooperative sensing to shorter detection time and agility are shown in [29].

In the case of external sensing, external network performs spectrum sensing and then shares the spectrum occupancy information with the cognitive radios. The external sensor network solves the hidden primary user problem and reduces the performance loss due to fading and shadowing [30]. Various sensor network architectures can be used. The sensor network does not need to be mobile, reducing problems with power consumption. Also the sensor network, or a single sensor, could sense the spectrum continuously compared to a cognitive radio, which can only sense the spectrum for a short while between the data transmissions.

Chapter 3

Algorithm verification

This chapter introduces the autocorrelation-based algorithm used in OFDM signal detection. The algorithm is based on [3], and modified in order to obtain better suitability for hardware implementation. Correspondence of the original algorithm and modified version is verified by simulations.

The original algorithm has a few calculations, which can be simplified, in order to obtain a more efficient implementation in terms of area and power. Various simulations are made with varying detection times (M), varying time delays (T_d) and by using decimation. The algorithm is very vulnerable to the DC offset, and it's effects were simulated, and also a method of compensating the DC offset is developed. Results are presented in Section 3.6. Word length tests for integrators are executed in Section 4.2.3.

3.1 Detection algorithm for implementation

The algorithm is based on the maximum likelihood estimate of the autocorrelation coefficient of OFDM signals. The autocorrelation coefficient is calculated by correlating the input signal with a delayed version of itself. Presence of a cyclic prefix (CP) of OFDM signals results in autocorrelation coefficients at delays $\tau = \pm T_d$, where T_d is the number of samples corresponding to useful symbol length in an OFDM block. The autocorrelation coefficients corresponding to lags $\tau = \pm T_d$, are shown to be the log likelihood ratio test (LLRT) statistic in the low signal-to-noise ratio (SNR) regime. By determining the distributions of these decision statistics under both hypotheses, the specified performance of the detector in terms of false alarm and detection probabilities can be achieved. In this work, Neyman-Pearson detectors are employed [31].

The maximum likelihood estimate of the autocorrelation coefficient is defined as

$$\hat{\rho}_{ML} = \frac{\frac{1}{2M} \sum\limits_{t=0}^{M-1} \Re\{x(t)x^*(t+T_d)\}}{\hat{\sigma}^2}, \tag{3.1}$$

where $x(t)$ represents observations over several OFDM symbols. $x(t)$ has real and complex parts $x_r(t)$ and $x_i(t)$. $\Re$ denotes the real part of a complex number, and M is the detection time. The variance of $\hat{\rho}_{ML}$ can be calculated as

$$\hat{\sigma}^2 = \frac{1}{2M + T_d} \sum_{t=0}^{M+T_d-1} |x(t)|^2. \tag{3.2}$$

In order to decide whether a signal is present or not, the detection statistic is compared to a threshold value calculated as

$$\eta_l = \frac{1}{\sqrt{M}} \cdot erfc^{-1}(2\alpha) \tag{3.3}$$

in which $erfc(.)$ is the complementary error function and α the constant false alarm rate.

In simulations the detection ratio is shown as a function of SNR. The input signal used in this simulation is an OFDM-signal with 32 subcarriers, which are modulated with 16-QAM. The size of the FFT is 32 and, therefore, T_d is set to 32 and in this case the cyclic prefix $T_c = T_d/4 = 8$. The amount of samples taken from the OFDM-signal is 4000. The false alarm rate (P_{fa}) is set to 0.05. The simulation results are shown in Figure 3.1.

The Figure 3.1 shows how the detection probability increases as a function of SNR. Detection rate tends to P_{FA} when SNR is low enough. As SNR grows high the detection rate reaches 1. The number of samples affects the detection performance and is discussed in Section 3.3

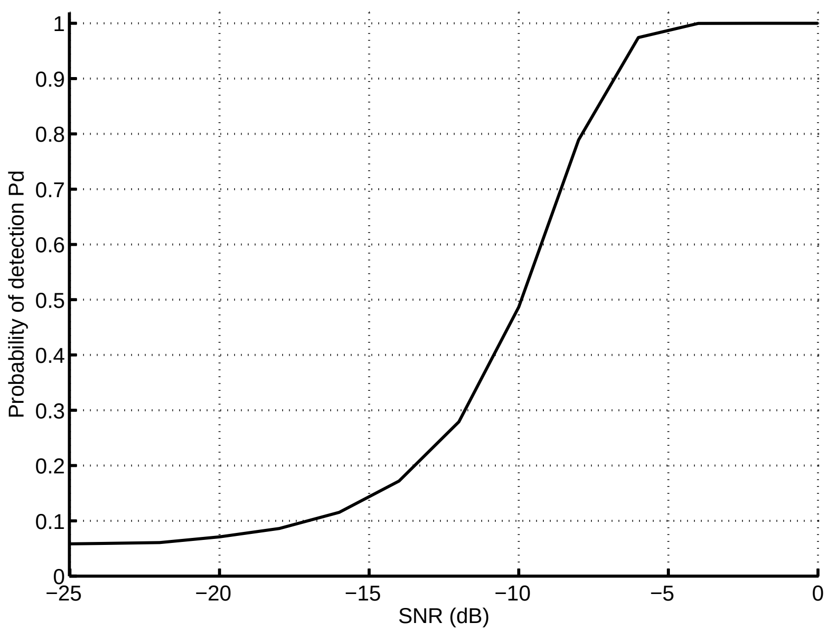


Figure 3.1: Probability of detection with the original algorithm, P_{fa}=0.05.

3.2 Approximations in algorithm

In order to attain the best implementation performance and support technology independent implementation, an approximation and a modification are applied to the detection algorithm. An estimate of the covariance is calculated over M signal samples instead of $(M + T_d)$ [3]. Assuming $M \gg T_d$, in implementation the number of samples varies from 512 samples to 4096 samples. Therefore, the change in the estimate of covariance is negligible. As a result, both autocorrelation and estimate of the covariance are summed over the identical number of samples and thus the factors in front of the summations cancel out thus eliminating one division computing unit. Another benefit is the equal signal path length for both autocorrelation and variance estimates, because no timing corrections are needed for proper hardware operation.

The approximated equation for the most likelihood estimate is

$$\hat{\rho}_{ML} = \frac{\sum\limits_{t=0}^{M-1} \Re\{x(t)x^*(t+T_d)\}}{\sum\limits_{t=0}^{M-1} |x(t)|^2}, \tag{3.4}$$

the obtained equation is simpler than their original equations represented in Equations 3.1 and 3.2, because there are no factors in front of autocorrelation and variance estimates.

Following modification is done, the places of absolute value operation and squaring of signal sample are switched, in order to simplify the DC offset calculation. We have an identity

$$|x^2| = |x|^2. \tag{3.5}$$

This identity is obtained by having a complex number

$$x = a + bj, \tag{3.6}$$

where a and b are real numbers and j is the imaginary unit, defined by $j^2 = -1$. Then the complex value is squared

$$x^2 = a^2 + 2abj - b^2. \tag{3.7}$$

Next, the absolute value of the complex equation is calculated by calculating a square root of the the complex number multiplied with its complex conjugate, and thus, we obtain

$$\begin{aligned} |x^2| &= \sqrt{(a^2 + 2abj - b^2)(a^2 - 2abj - b^2)} \qquad (3.8) \\ &= \sqrt{a^4 + 2a^2b^2 + b^4} \\ &= \sqrt{(a^2 + b^2)^2} = |x|^2. \end{aligned}$$

This modification simplifies implementing as taking the absolute value before squaring the signal sample is straightforward in hardware and is done with a multiplier, but implementing it in a reverse manner would require implementing a square root computation unit [32]. Calculating the square root would be as hardware consuming as implementing a division computation unit, see Section 4.2.4. Another method to calculate the absolute value is to use a coordinate rotation digital computer (CORDIC) [33] that can be implemented with multiple registers and adder/subtractors. However, both of these consume more resources compared to a complex multiplier.

3.3 Varying detection times

The detection time is an important factor for the performance and energy consumption of the implementation. The detection time is presented as a sample length is discrete systems. In order to attain reliable information about detection performance with certain hardware cost, the detection performance is simulated with varying sample lengths. Longer detection time increases detection performance, but a compromise has to formed between detection performance and time. The sample length has a direct impact on power consumption as the amount of data that needs to be processed is highly dependent on sample length. Therefore, sample length is an important factor for optimizing performance and power consumption.

Variance in sample length is simulated with MATLAB. The input signal models the 802.11g WLAN standard, which is an OFDM-signal with 64 subcarriers. The size of the IFFT is 64 and thus T_d is set to 64 and, in the case of a WLAN signal, the cyclic prefix $T_c = T_d/4 = 16$. The sample length varies from 128 to 32768. The false alarm rate (P_{fa}) is set to 0.05. Figure 3.2 presents how much the detection performance varies with different sample lengths as a function of SNR.

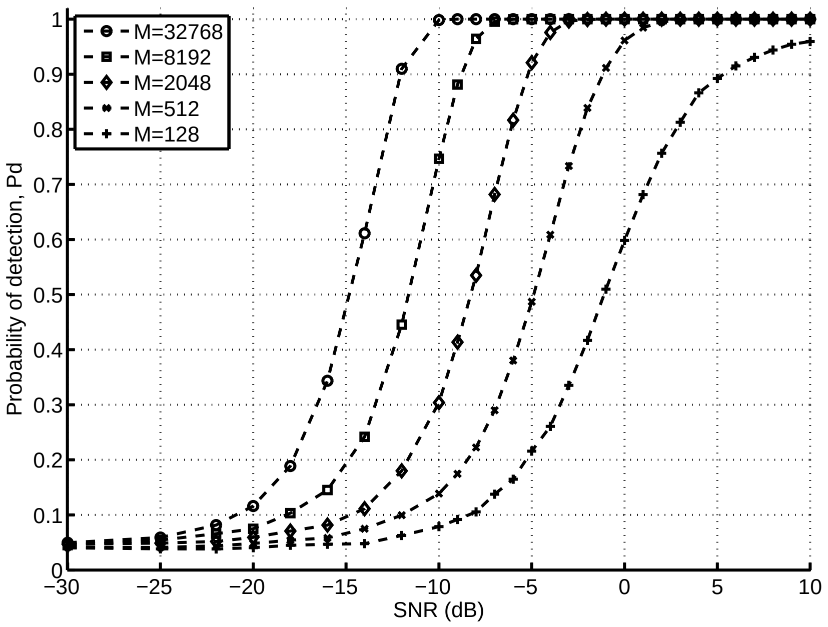


Figure 3.2: Probability of detection as sample length varies from 128 to 32768,P_{fa}=0.05. Probability of detection is higher with longer sample lengths.

3.4 Variance in autocorrelation time delay

Other important parameter of the detection algorithm is the autocorrelation time delay (T_d). The autocorrelation time delay is the factor used to identify the system. Incorrect time delays may result from timing or signal processing faults in the transmitting or receiving end.

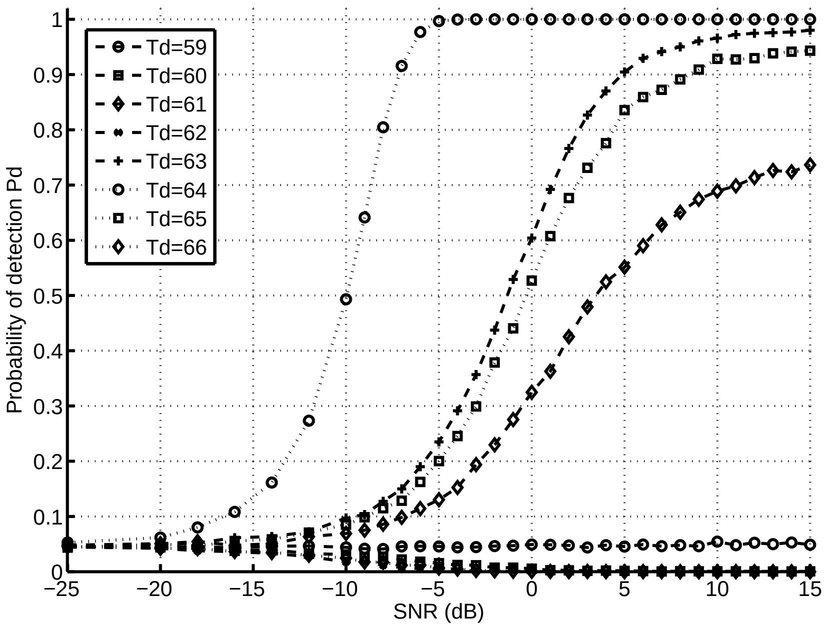


Figure 3.3: Probability of detection when autocorrelation time delay (T_d) used in the detection varies from 59 to 66,P_{fa}=0.05. The algorithm easily distinguishes the correct signal (T_d=64) from the other signals.

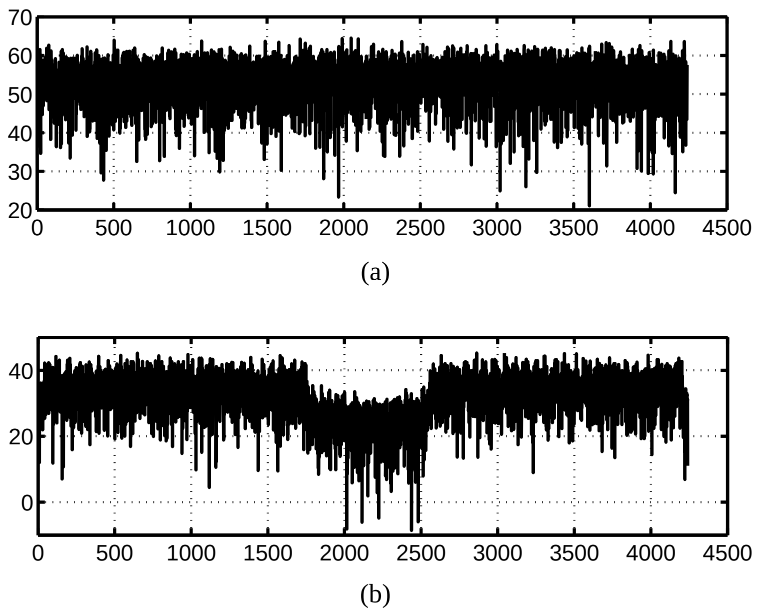


Figure 3.4: Spectrum of the signal with time delay of 64 samples: (a) Signal spectrum at SNR of -20dB and (b) signal spectrum at SNR of 10dB.

The algorithm behavior was simulated with the same input signal as in Section 3.3. With varying the autocorrelation time delay used in the detector was changed to find out how difference to correct time delays affect detection performance. The detection performance, shown in Figure 3.3, demonstrates how the algorithm detects signals with a difference of one, but with a greater difference the performance tends to P_{fa} as desired. However, the real signal is detected 10dB earlier compared to the false signal. One interesting feature is how all the curves depicting false signals do not tend to the set false alarm rate of 0.05, instead many of them tend to zero. It is due to the Neyman-Pearson test hypothesis [31], in which, the noise is estimated as AWGN. However, when the signal is stronger compared to noise the test fails as the signal does not have white noise characteristics. Figure 3.4(a) shows the spectrum of a signal with SNR of -20dB and the noise is AWGN. In Figure 3.4(b), the SNR is 10dB and the noise is not AWGN as there are a few zero sub-carriers in the signal.

3.5 The effect of DC offset

The DC offset is a typical front-end imperfection in direct-conversion receivers (DCR) affecting the received signal [34]. Direct conversion is the common approach to down convert a signal from RF to baseband. A DCR translates a specific band of interest directly to zero frequency, hence high offset voltages can corrupt the signal and saturate the following stages [12]. Other RF front-end imperfections include I/Q mismatch, even-order distortion, and flicker noise, but the DC offset affects to the algorithm's detection performance most severely. Therefore, it's effects on the signal path must be compensated. Otherwise, it may significantly degrade the performance of the detection algorithm as is shown in Figure 3.5. In the simulation the DC offset is added to the signal relative to noise power in dB. For example, we have a signal with signal-to-noise ratio of -5dB, and then, a significantly weaker DC offset of -18dB compared to signal noise level is added to the signal. The detection performance deteriorates even at DC offset levels of -21dB relative to noise power.

3.6 DC offset compensation

There are numerous methods to compensate the DC offset [34]. The DC offset can be measured, and then compensated with feedback loop. In certain cases, it can be removed by capacitive coupling. More feasible in our case is to modify the algorithm to be tolerant to the DC offset.

The DC offset can be compensated by calculating the mean value estimate from the input signal, and then removing it's squared value from both the dividend and divider. The approximate maximum likelihood estimate with DC compensation ready for implementation

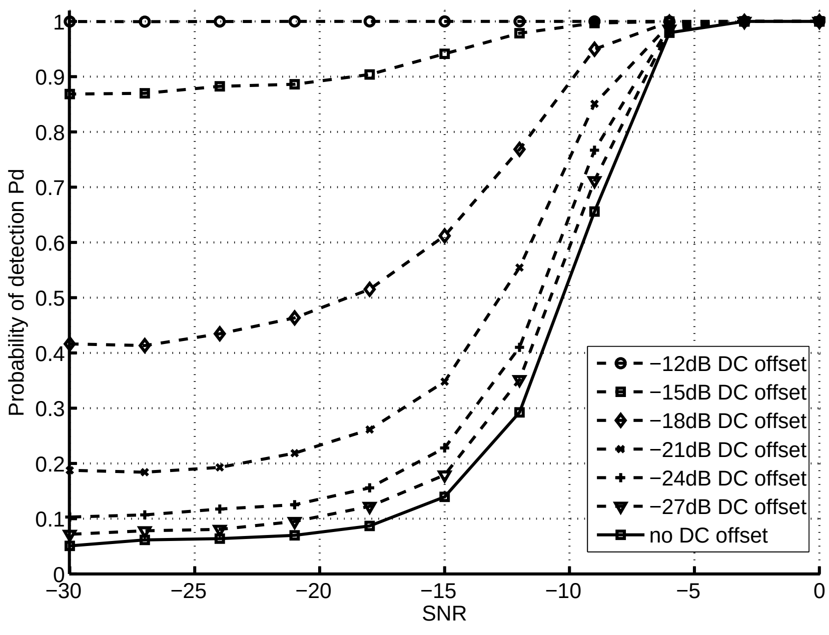


Figure 3.5: DC offset has a considerable effect on detection probability.

is as follows

$$\hat{\rho}_{ML} = \frac{\sum_{t=0}^{M-1} \Re\{x(t)x^*(t+T_d)\} - \hat{\mu}^2}{\sum_{t=0}^{M-1} |x(t)|^2 - \hat{\mu}^2}, \tag{3.9}$$

where $\hat{\mu}^2$ is calculated as

$$\hat{\mu}^2 = \left| \frac{1}{M} \sum_{t=0}^{M-1} x(t) \right|^2 . \tag{3.10}$$

DC offset compensation performance is simulated in similar manner as in Section 3.5. The DC offset was added to the input signal relative to noise power in dB. The DC offset compensation is evaluated in Figure 3.6, where two signal curves have an added DC offset of -18 dB power level relative to noise power in dB, and the third curve has only AWGN noise. Figure 3.6 presents how the DC offset compensation cancels the effect of DC offset.

3.7 Decimation

Decimation is a technique commonly used to reduce the number of samples in discrete signals. Decimation is a useful method to create more power-efficient implementation as the processed sample rate is lower and consumes less power. Since the sampling rate is reduced by downsampling, the resulting signal would be an aliased version of the original signal [6]. Therefore, the input signal bandwidth must be limited according to Nyquist sampling theorem. Decimation is accomplished by low-pass filtering the signal with an anti-aliasing filter prior to downsampling.

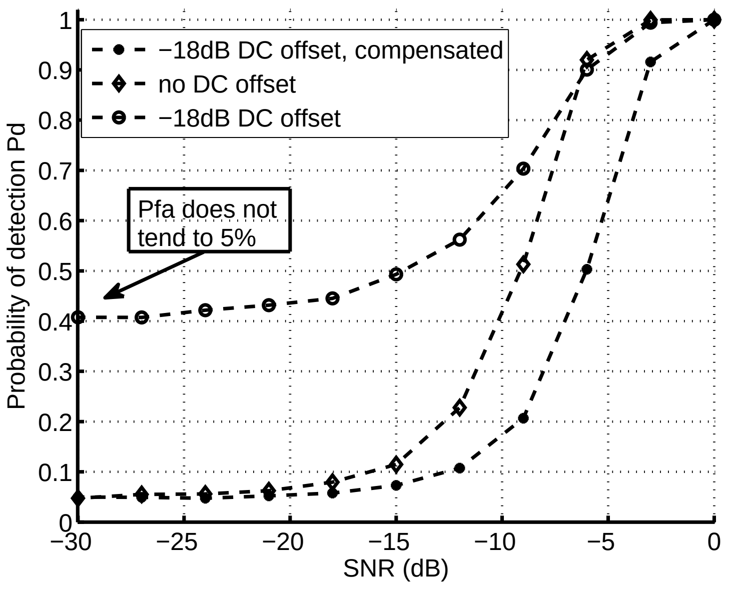


Figure 3.6: DC offset can be compensated without any effect on detection probability.

It was planned to use decimation in the algorithm implementation. In simulations, decimation improved detection probability similar to increasing the detection time. Simulations pointed out no clear advantage in using decimation, in addition, we are already using integrators to reduce the effects of the DC offset, shown in Section 3.6. As decimation employs a low pass filter that is equal to a more sophisticated integrator, it is too complicated to implement without any real benefit. In the end, it was not implemented in this work.

Chapter 4

Implementation of the algorithm

4.1 Introduction to implementation

The autocorrelation-based detection algorithm was implemented with simple blocks using standard logic in order to support platform independency and enabling a straightforward development process. As the implementation is hardware oriented, modifications to the algorithm were performed to produce a hardware efficient implementation. Section 4.2 describes the most fundamental blocks of the implementation including multiplication, multiplication with delay, integration, DC offset compensation, division, and control.

All the implementation blocks are described in VHDL (VHSIC hardware description language; VHSIC: very-high-speed integrated circuit) source code. VHDL codes describing the implementation are simulated and discussed in Section 4.4. After the verification of the VHDL codes, they are synthesized for both a field-programmable gate array (FPGA) and an application-specific integrated circuit (ASIC). These are described in Section 4.5. The FPGA implementation is verified with measurements in Chapter 5.

4.2 Fundamental design blocks

The final implementation consists of three main signal paths; autocorrelation, variance, and the DC offset compensation value calculations. The block diagram is shown in Figure 4.1[35]. The middle path calculates the autocorrelation estimate, and the lowest path the variance estimate. The top path calculates the DC offset compensation value that will be subtracted from the autocorrelation and variance estimates, before they are supplied to the division block. Following sections will describe the fundamental blocks: multiplication, multiplication with delay, integration, division, DC offset compensation and control logic.

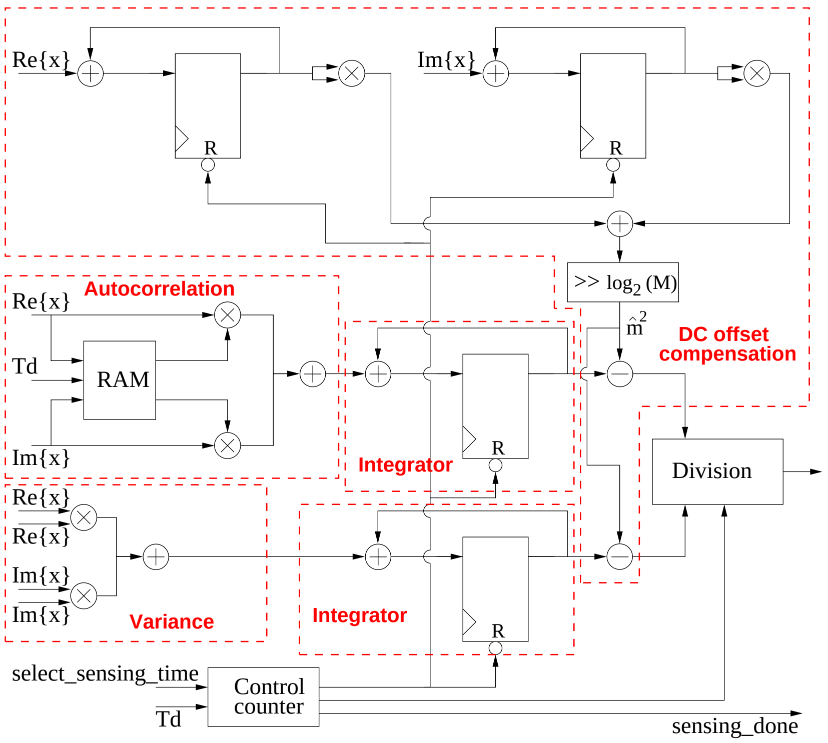


Figure 4.1: Final detection algorithm implementation.

4.2.1 Multiplication

A multiplication block simply multiplies two input values with each other. The simple structure consist of two complex multipliers and an adder. The structure of the multiplication block is shown in Figure 4.2.

The real $Re(x)$ and complex $Im(x)$ inputs depict the I- and Q-branches of the input radio signal. These input signals are squared, and the results are added together. The result is the squared absolute value of the input signal $x(n)$ sample. Afterwards this block's results are accumulated in the integrator described in Section 4.2.3. Finally, the obtained sum is the divider input for the division block that is shown in Figure 4.7.

In the implementation the input word length is 12 bits, therefore the word length after both multipliers is 24 bits. In order to prevent overflow, the output word length after final adder is 25 bits.

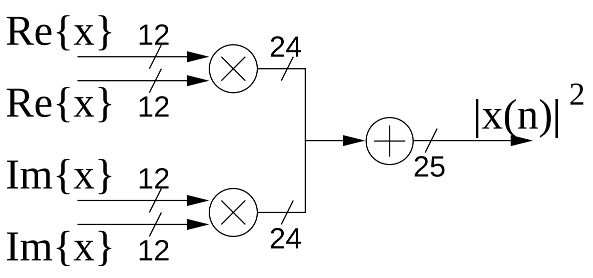


Figure 4.2: Multiplication.

4.2.2 Multiplication with delay

Multiplication with delay block is used for calculating the autocorrelation estimate. The output of this block is averaged and then used as the dividend for the algorithm. Multiplication with delay is primarily the same as the basic multiplication block introduced in Section 4.2.1.

The main difference is that this block uses a dual-port random access memory (RAM) to delay the values of the input signals in order to calculate the autocorrelation. When the detection cycle starts, the RAM writes the input values in the memory addresses. After the memory has values for the delay, the multiplication block reads the input values from the memory starting with the first address. Thus, we have both the present and delayed input values in the multiplier. The memory addressing with T_d is depicted in Figure 4.3.

The autocorrelation time delay is implemented with 8 bits, therefore the delay used in detection varies from 0 to 255 samples. Thus, 1.5 kbits of memory is required to implement delay of 255 samples. Word lengths in this block are equal to word lengths in the multiplication block. The delay could be implemented in a simple way with a register chain. A small fixed delay value is appropriate to implement with register chain, however implementing programmable delay with register chain and multiplexer would be a waste of resources. In addition, a more flexible implementation with a dual-port random access memory is achieved.

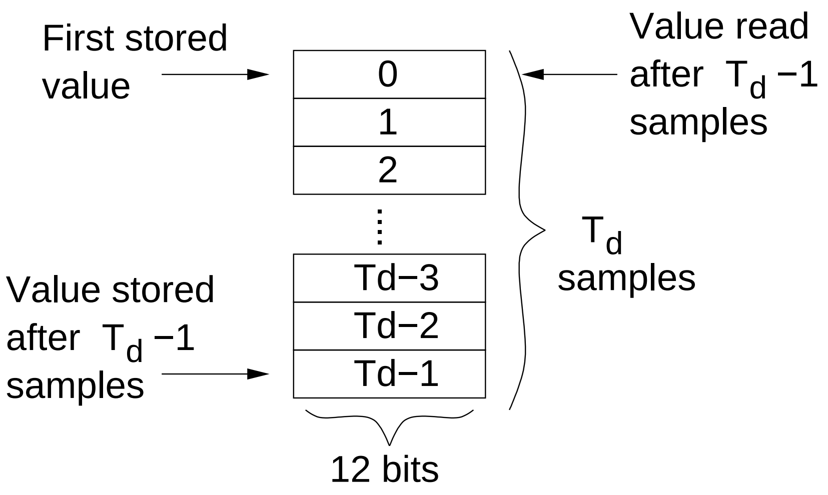

First stored value
0
1
2
Td−3
Td−2
Td−1
Value stored after T_d −1 samples
Value read after T_d −1 samples
T_d samples
12 bits

Figure 4.3: T_d samples are stored into RAM.

4.2.3 Integration

Integration in hardware is equal to calculating a cumulative sum. The integration block is used several times in implementing the algorithm. It is used in the calculation of variance, autocorrelation, and squared mean estimates. The integrator is a quite simple block, but due to the word lengths needed, it consumes a considerable amount of resources.

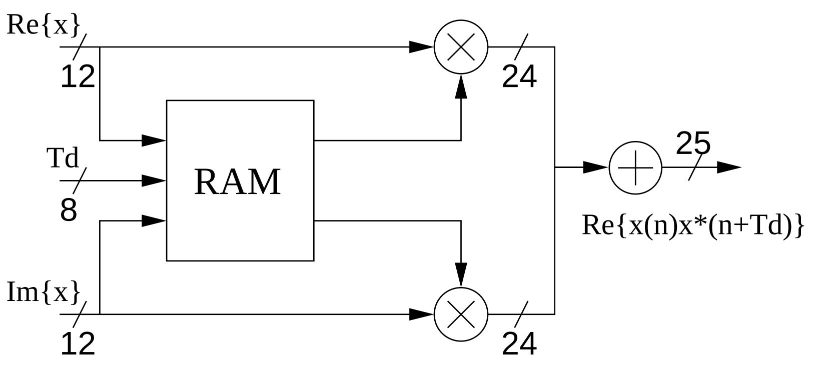


Figure 4.4: Multiplication with delay (T_d).

Figure 4.5 presents the integrator block, which is comprised of an adder and two n-bit registers. The other n-bit register has an enable port. The first register updates it's value constantly until an enable signal input in inputted. At that time, the second register places the current value after the adder to the output port. At the same time, the first register is reset, thus starting a new detection cycle.

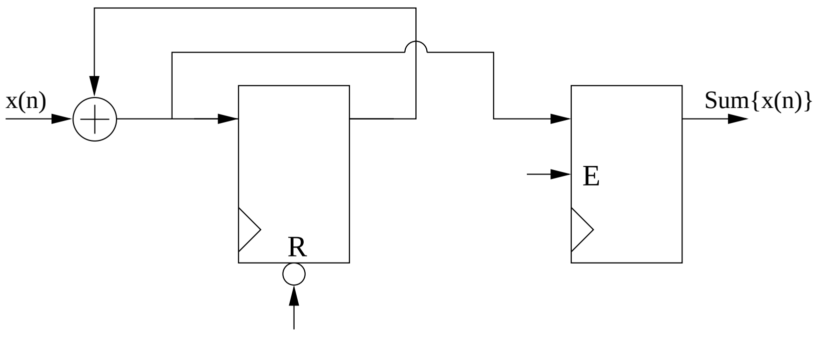


Figure 4.5: Integration.

The input word length for the integrators is 25 bits that is the same as the output word length of multiplication block. Determining the output word length of the integrator is important part of the implementation. Figure 4.6 shows how too short word length ruin the detection performance as the detection curve of 4096 samples never reaches the false alarm rate of 5 percent. The opposite case of too large integrator value guarantees detection performance, but consumes resources power and area wise. In addition, the attainable implementation operating clock frequency is lower when longer word lengths have to be processed at each following step.

The worst case estimate for integrator output word lengths is obtained when full dynamic scale signal is averaged for the sample length. In the worst case, for 4096 sample length an overhead of 13 bits is required. Thus, in the worst case the output word length would be 38 bits. However, with optimal word length fitting, power consumption and area requirements can be optimized without performance loss.

The optimal integrator word lengths were determined by executing simulations with var-

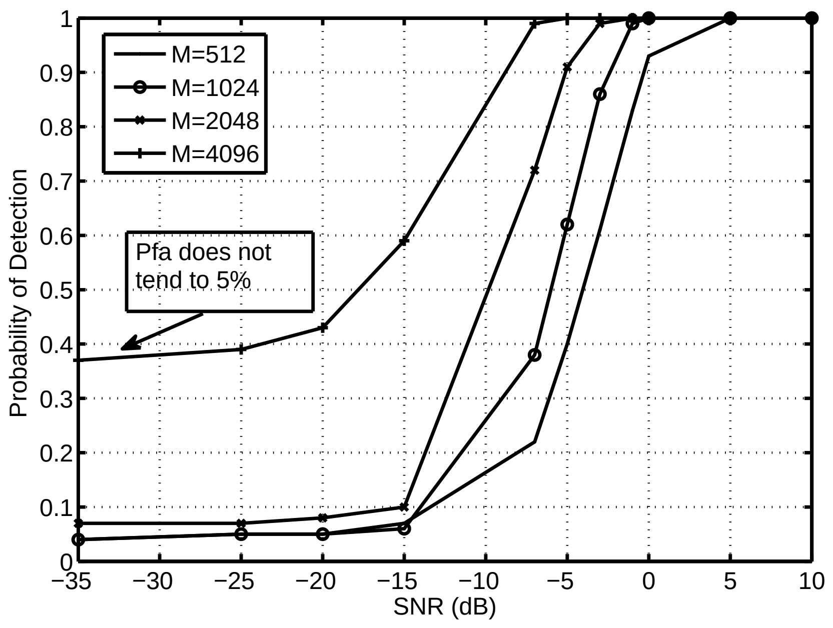


Figure 4.6: Incorrect integrator word lengths results in deteriorated performance.

ious word lengths used in the integrators. The word lengths required for various sample lengths were determined by simulations and the theoretical maximum value is calculated from the worst case assumption. The required amount of overhead bits are shown for various sample lengths in Table 4.1. Considerable amount of resources can be saved by utilizing only the required word lengths instead of theoretical maximum values. Table 4.1 shows how the variance and DC offset calculations need longer word lengths compared to autocorrelation. The values required for the sample length of 4096 samples with one extra bit will be used in the implementation.

4.2.4 Division

Division is an operation that is not well suited for binary calculations. In this design, the division is the most complex block of the implementation and has to be implemented as efficiently as possible. Good accuracy is important as the detection statistic relies on the quotient of division. Various division algorithms are reviewed in [36]. Division algorithms differ by the hardware operations used in their implementation, and are divided into five classes: digit recurrence, functional iteration, very high radix, table look-up, and variable latency.

The digit recurrence algorithms calculate a fixed number of bits of the quotient at a single iteration. It is simple to implement and utilizes small area, however, it has a relatively large latency. The latency can be reduced by using larger radices, but it results in increased complexity. In functional iteration, multiplication is the fundamental operation. Multiplications are used to converge to a result quadratically. Functional iteration has less latency compared to digit recurrence based algorithms, but in order to obtain accurate results, the number of

Table 4.1: Overhead bit requirements for integrators used in autocorrelation, variance, and DC offset calculations.

Autocorrelation sum		
Sample length	bits required	bits theor. max.
512	3	10
1024	4	11
2048	5	12
4096	6	13

Variance sum		
Sample length	bits required	bits theor. max.
512	4	10
1024	5	11
2048	6	12
4096	7	13

DC offset		
Sample length	bits required	bits theor. max.
512	4	10
1024	5	11
2048	6	12
4096	7	13

iterations and a rounding operation must be considered thoroughly. Both the functional iteration and digit recurrence with high radices can benefit from the use of look-up tables. They can be used to obtain a sufficiently accurate initial approximation. Look-up tables are fast as no arithmetic calculations are needed, however a trade-off exists between the precision of the table and its size. Variable latency algorithms introduce improvements to previous algorithms by exploiting the fact that the computation for certain operands can be completed sooner than others, or reused from a previous computation. Therefore, the overall system performance can be enhanced, as the result can be provided as soon as it is ready.[36]

In this design, the calculation of the division is performed with an algorithm based on the digit recurrence. Since the division is calculated concurrent to the integrators already processing the following detection, even large latencies are acceptable. Although the latency is large, it is only 10 percent of the shortest sample length in the implementation. The division algorithm and a more compact version are presented in [37].

The idea in division is that while the dividend and divider could be large, we know that the ratio is always small, or the large values are irrelevant in our case. Then the outcome can be presented with small number of bits. If the ratio is larger than can be presented with output bits, overflow is detected and maximum value, vector of 1's, is outputted. Second error in division can occur when the divider is zero, it is also detected by the overflow detection and handled in same manner.

To begin with the division, 32-bit words are assumed. 64-bit remainder register's left half is initialized with the dividend, and then a 32-bit divisor register is initialized with the divisor. The algorithm starts by subtracting the divisor from the left half of the remainder and placing the result in left half of the remainder. Now the sign of the remainder is tested. If it is negative, the original value is restored to the remainder by adding the divisor to it. The remainder is then shifted left, and the least significant bit is set to 0. When the result is positive the remainder is shifted left, and the rightmost bit is set to 1. This cycle is repeated 33 times, and at the end the quotient is in the right half of the remainder register. This division algorithm needs two registers, a subtractor, three multiplexers and control logic. The division hardware is shown in Figure 4.7.

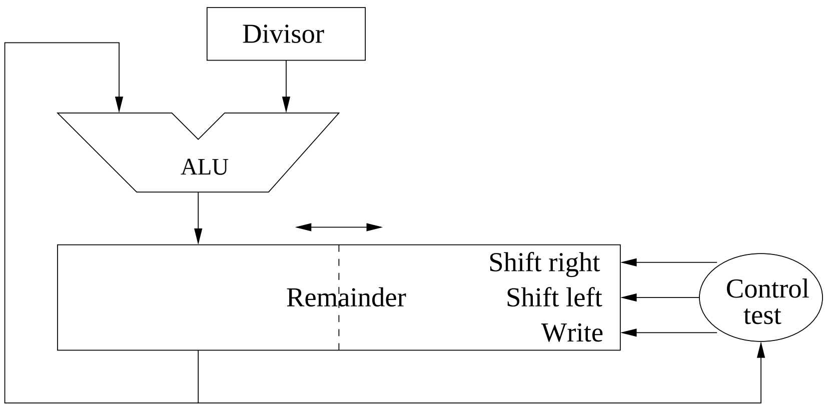


Figure 4.7: Implementation of division.

4.2.5 DC offset compensation

The detection algorithm is sensitive to the DC offset so a block to compensate for the DC offset is needed. The DC compensation value is obtained from a squared mean value calculated from both I and Q input signals, and it is subtracted from the dividend and divider values prior to division.

Both I and Q inputs are integrated and the integrators' outputs are then squared. The squared signals are then accumulated for the sample length, and the sum is then divided by the sample length to create the estimate of squared mean value used in DC compensation. In this case, the division can be done with a shift operation as the sample length is a power of two. DC offset compensation can be implemented in hardware with two integrators, two multipliers, an adder and a shift operation. The implemented block is shown in Figure 4.8.

The DC offset compensation can be disabled. The measured performance is presented in Chapter 5. The disable feature was implemented with a multiplexer, and therefore, even if the DC offset compensation is turned off, the DC offset compensation value is calculated normally, but the result is not subtracted from the autocorrelation and variance estimates.

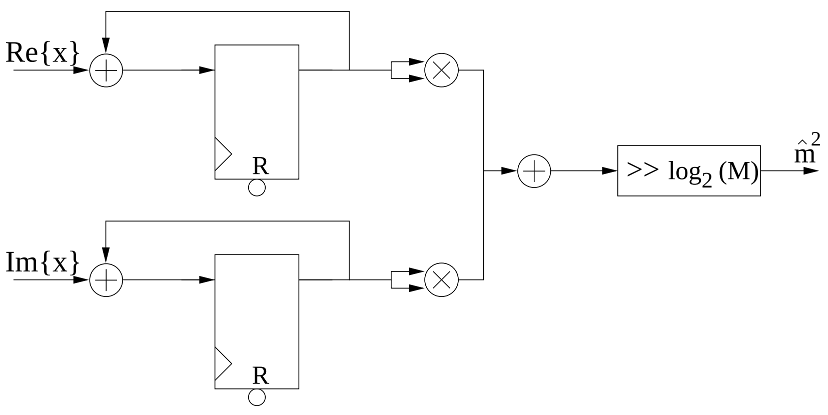


Figure 4.8: Implementation of DC offset compensation.

4.2.6 Control logic

Precise timing is needed to obtain correct operation throughout the implementation. Control logic is implemented in an own logic block monitoring and controlling the other blocks' event signals.

To achieve correct timing, a counter is used to count rising clock edges. Counter size is an open synthesis parameter based on the maximum sample length used in the detection. When the detection cycle is started, an active low reset is pulled high and the counter starts counting. It resets integrators calculating autocorrelation, variance and DC offset sums. After the counter attains the appropriate sample length value, a new detection cycle is started, and the current values in the integrators's registers are passed onward to the division block. Before division is started, the DC offset values are subtracted from the autocorrelation and variance, the autocorrelation value is divided by variance and the detection statistic is obtained. The division block sends the statistic value to the output port, and also sends an event to the control logic. The control logic is implemented with a counter. The flip-flops are used to delay event signals to time the event signals correctly to each block.

4.3 Detection time

Time consumed for a single detection is the detection time, and it depends on the sample length, autocorrelation time delay, and sample rate. The number of clock cycles needed for the detection, can be calculated as

$$\text{clock cycles} = M + T_d, \tag{4.1}$$

where M is the sample length and T_d is the autocorrelation time delay. Detection time can be expressed as

$$\text{detection time} = \frac{M + T_d}{f_{s,in}}, \tag{4.2}$$

where $f_{s,in}$ is the input sampling frequency.

The final implementation uses four specific detection times: 512, 1024, 2048, and 4096 samples. For example, a system using a 20 MHz sampling frequency, the detection time used in a single detection varies from 28.8 μs to 208 μs.

4.4 VHDL description verification

The algorithm is implemented using VHDL description. Various tests and simulations are performed in order to verify that the algorithm works as desired and all exceptions are considered. Especially possible overflows and exceptions in the multiplication and division operations need to be handled. It is very important to implement the division block with an exception control to make sure the division does not result in undefined states, for instance, division by zero. To ensure the functionality of the design, timing and functional simulations are essential.

4.4.1 Functional and timing verification

In order to ensure functionality, the design blocks were verified with functional and timing simulations. The design was verified with waveform simulations in Modelsim. In the waveform simulation, a test input is fed in the block and the inner signals are monitored by the simulator.

Figure 4.9 shows a waveform window from Modelsim. In Figure 4.9 the vertical line shows where the first detection cycle has just ended, and a detection statistic is placed in the output. By monitoring pulse widths, signal rise and fall times, as well as signal arrival times, optimal performance can be achieved. By functional simulations, timing errors and glitches can be discovered. In a similar manner, the main blocks have been verified to function properly.

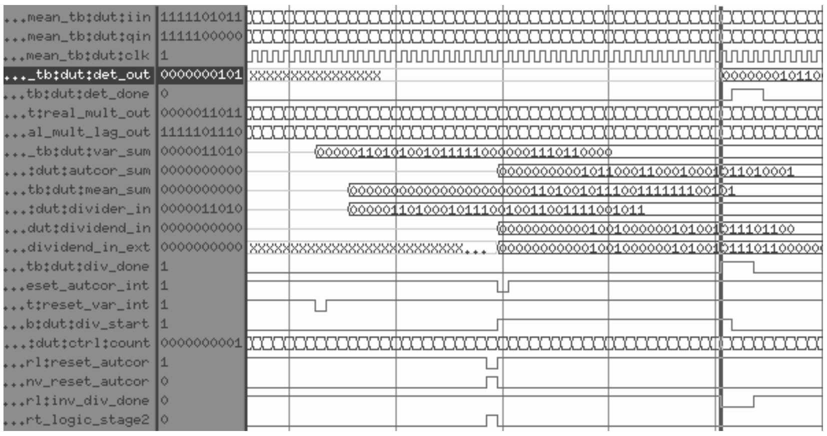

...mean_tb:dut:iin 1111101011
...mean_tb:dut:qin 1111100000
...mean_tb:dut:clk 1
..._tb:dut:det_out 0000000101
...tb:dut:det_done 0
...t:real_mult_out 0000011011
...al_mult_lag_out 1111101110
..._tb:dut:var_sum 0000011010
...:dut:autcor_sum 0000000000
...tb:dut:mean_sum 0000000000
...:dut:divider_in 0000011010
...dut:dividend_in 0000000000
...dividend_in_ext 0000000000
...tb:dut:div_done 1
...eset_autcor_int 1
...t:reset_var_int 1
...b:dut:div_start 1
...:dut:ctrl:count 0000000001
...rl:reset_autcor 1
...nv_reset_autcor 0
...rl:inv_div_done 0
...rt_logic_stage2 0

Figure 4.9: Top-level simulation with Modelsim. The vertical line points to the end first detection cycle.

4.4.2 VHDL simulations various detection times

The VHDL description are simulated with functional simulations to verify performance, and that no timing or other errors exist that might degrade detection performance. Functional simulations are performed with a test bench in Modelsim. The simulation itself is similar to the detection time simulation used with algorithm verification, in contrast to MATLAB simulations, the word length limitations are considered in VHDL descriptions. Therefore, new kind of problems may occur. The VHDL simulations were executed with detection times varying from 512 to 4096 samples with $T_d = 64$. Due to memory limitations in the simulation environment, the results, shown in Figure 4.10, are averaged over 400 measurements.

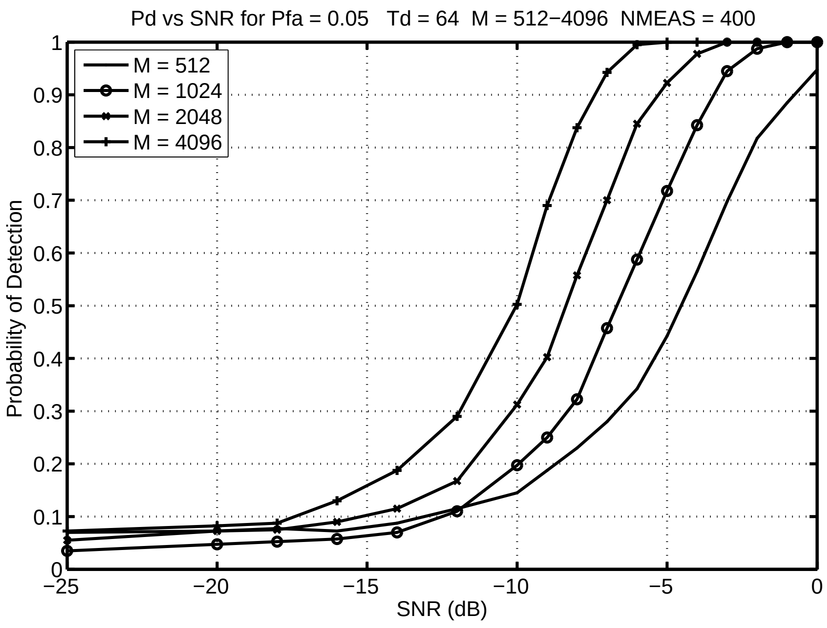


Figure 4.10: VHDL description simulation when detection time varies from 512 to 4096 samples,P_{fa}=0.05.

4.5 Implementation for FPGA and ASIC

The VHDL descriptions verified by functional simulations are ready to be implemented on hardware. In this section, an implementation for both an FPGA and an ASIC is performed. The implementation for FPGA is done on a Xilinx Virtex-5 evaluation board. Xilinx ISE software is used in the synthesis and layout.

The implementation for an ASIC is done using Synopsys Design Compiler for synthesis and Synopsys Astro for layout design. The VHDL descriptions are platform-independent, and the same implementation can be synthesized to ASIC and FPGA. Naturally, the ASIC implementation would be more resource efficient if RAM memory was used instead of flip-flops. Now the implementation uses registers to store the delayed input values in the autocorrelation calculation, and it consumes more power and area compared to an efficient memory block.

4.5.1 FPGA Synthesis and Layout

The target board for implementation is a Xilinx Virtex-5 LX50 evaluation board. The Virtex-5 FPGA uses a FF676 package and the speed grade is -1. The speed grade determines the switching characteristics of the FPGA, a higher value equals faster operation. The 802.11g WLAN standard uses a sampling frequency of 20 MHz, and it was used as a timing constraint for the design. The Xilinx ISE software tools are used to perform synthesis and layout of the design, based on the supplied FPGA details and timing constraints.

Due to the FPGA logic structure, synthesizing the design, and generating the layout for FPGA only takes a fraction of time compared to creating an ASIC circuit. Xilinx tools perform the necessary timing verification and design rule checks, but the functionality tests are left for the user. The layout generated by Xilinx tools utilizes 987 LUT flip-flop pairs. For comparison, a 64-point FFT processor was implemented in [38], and it used 33961 LUT flip-flop pairs in a Virtex-5 FPGA. The hierarchical layout complexity is presented in Table 4.2. The resource usage is presented by the number of LUT flip-flop pairs used. Single LUT flip-flop pair is shown in Figure 2.1(b). It can be noted that calculating the division, and compensating the DC offset consume most of the logic resources. The DC offset compensation resource usage is due to the long word lengths used in the multiply and integration calculations.

Table 4.2: Hierarchical resource usage of the FPGA implementation.

Hierarchical resource usage (Number of LUT flip-flop pairs used)		
Block	**LUTs used**	**%**
Division	307	31.1
DC offset compensation	297	30.1
Autocorrelation (dividend value)	158	16.0
Variance (divider value)	100	10.1
Control logic	92	9.3
Miscellaneous	33	3.4
Total	**987**	**100**

The maximum operating frequency reported by Xilinx tools was 95.3 MHz, but the obtained maximum frequency depends highly on the used word lengths. The 802.11g WLAN standard's sample frequency is 20 MHz, therefore, the obtained maximum operating frequency is more than sufficient.

Power consumption of the FPGA implementation was analyzed by a Xilinx XPower analyzer, and the dynamic power consumption is presented in Table 4.3. As expected, the power consumption of the main blocks is quite similar to resource usage, as the largest blocks commonly contain more active signals and registers changing state. In contrast to resource usage, the division block consumes only a fraction of total power, because it is only active over the time period calculating the result, and is idle the rest of the time.

Table 4.3: Power consumption of the FPGA implementation.

Hierarchical power consumption		
Block	**mW**	**%**
Autocorrelation	1.30	35.2
DC offset compensation	1.22	33.1
Detector (top level)	0.61	16.5
Variance	0.35	9.5
Division	0.13	3.5
Control logic	0.08	2.2
Total	**3.69**	**100**

4.5.2 ASIC Synthesis and Layout

ASIC synthesis is done using a ST 65 nm CMOS design kit. The nominal voltage is 1.0 V and the nominal temperature is 25°C. The standard-cell library is designed for low power and high threshold voltage. The high threshold voltage results in decreased leakage, but the circuit will function slower. The synthesis was performed in a worst case scenario with a voltage of 0.9 V and a temperature of 105°C. The 802.11g standard WLAN signal with 20 MHz sampling frequency is used as a timing constraint for design and clock tree generation. The design was synthesized with Synopsys Design Compiler.

Subsequent to synthesis, both area, and power estimations were obtained. The hierarchical area distribution of the main cells from synthesis report is shown in Table 4.4. The autocorrelation multiply cell consumes most of the area, because the delay was implemented with logic instead of memory, thus the area estimate will be lower when memory elements are used in the synthesis.

The power consumption estimate considers switching, internal cell, and leakage power. Switching power is dissipated when charging and discharging the load capacitance at the cell output. The load capacitance is composed of the interconnect capacitance and gate capacitances the net is connected to. Switching power consumption depends greatly on switching activity, and thus, is related to the operating frequency of the cell. Internal power is consumed withing a cell for charging and discharging internal cell capacitances. It also includes short-circuit power, as over logic transitions both P and N type transistors are both on simultaneously, thus causing a direct connection from the supply voltage to the ground voltage for a short time. Leakage power is consumed by sub threshold currents and by reverse biased diodes in a CMOS transistor. It does not depend on input transitions or load

Table 4.4: Hierarchical area distribution of the ASIC implementation after synthesis.

Hierarchical cell	**Cell area** [$\mu\mathrm{m}^2$]	**%**
Autcor. multiply	85959.3203	73.1
DC offset calculation	19399.4453	16.5
Divider	4984.6909	4.2
Variance multiply	3903.6387	3.3
Ctrl counter	1317.6818	1.1
Variance integrator	1003.0789	0.9
Autcor. integrator	1000.9989	0.9
Total	**117596.8672**	**100**

capacitance, hence it is constant for a logic cell. The power consumption of the synthesized circuit is presented in Table 4.5.

Table 4.5: Power consumption of the ASIC implementation after synthesis.

Hierarchical cell	**Switching Power [mW]**	**Internal Power [mW]**	**Leakage Power [mW]**	**Total Power [mW]**	**%**
Autcor. multiply	71.9e-03	0.713	2.78e-03	0.787	82.5
DC offset calculation	8.11e-03	47.9e-03	0.584e-03	56.6e-03	5.9
Variance multiply	14.6e-03	31.8e-03	0.123e-03	46.5e-03	4.9
Divider	0.894e-03	34.0e-03	0.163e-03	35.1e-03	3.7
Ctrl counter	0.982e-03	12.0e-03	43.5e-06	13.0e-03	1.4
Variance integrator	0.813e-03	6.38e-03	36.4e-06	7.23e-03	0.8
Autcor. integrator	0.527e-03	5.42e-03	37.9e-06	5.98e-03	0.6
Total	**0.100**	**0.850**	**3.77e-03**	**0.954**	**100.0**

After synthesis was verified successfully, the layout was generated with Synopsys Astro. The final version was also verified with functional and timing simulations. Area consumption and a more precise estimate of power consumption were obtained from layout. The obtained estimates for area and power consumption are presented in Table 4.6. The layout picture extracted from Synopsys Astro is shown in Figure A.1.

Table 4.6: Final area and power consumption of the ASIC implementation.

Final ASIC implementation values	
Total area	0.254 mm^2
Power consumption	1.019 mW

Chapter 5

Experimental measurements

FPGA is an optimal platform for algorithm performance evaluations, since it can be programmed multiple times, it is possible to debug the design by doing small updates at a time. In order to verify the results obtained in simulations, the algorithm is implemented on FPGA to obtain the measured results. The measurement configuration and necessary equipment will be described in Section 5.1. Measurements of the FPGA implementation are similar to VHDL description simulations because no analog signals are used. Therefore, the measurement results should be equal to the VHDL simulations.

The implementation was designed to use various detection times, and they are measured in Section 5.2. As expressed prior to algorithm implementation, the algorithm is sensitive to DC offset. Therefore, the effectiveness of DC offset compensation is evaluated with measurements in Section 5.3.

5.1 Measurement configuration

The FPGA implementation was measured using a simple configuration setup shown in Figure 5.1. The signal was generated with MATLAB modeling the 802.11g standard. The input signal is supplied to a pattern generator. The pattern generator transmits the input signal to the programmed FPGA. The input signal consist of I- and Q-branches both using 12 bits, hence the signal is transferred with 24 wires. An external waveform generator supplies the clock signal to both the pattern generator and the FPGA. After a detection cycle is done, the FPGA sends a detection done and detection statistic signals to the logic analyzer. The detection done signal is a single bit, which functions as a clock signal for the logic analyzer. This signal was also measured by a digital oscilloscope to verify correct operation of the detector. The detection statistic signal is a 16 bit signal represented in 2’s complement format. When measurement has ended the data from the logic analyzer is analyzed in MATLAB.

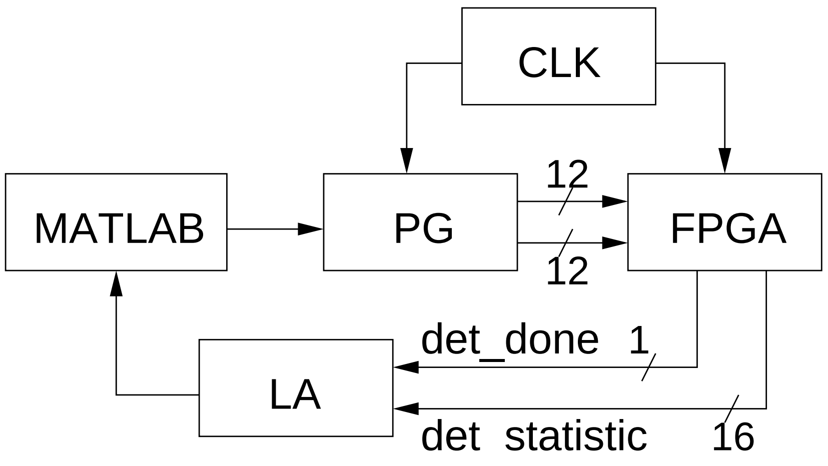


Figure 5.1: Measurement configuration.

Following FPGA and measurement equipment were used in the measurements:

- Avnet Xilinx Virtex-5 LX50 evaluation kit with expansion modules
  - EXP-to-P160 adapter module: Converts EXP style expansion slot to a P160 format
  - P160 analog module: Composed of four independent analog channels, two supporting analog inputs and two supporting analog outputs
    - Two Texas Instruments ADS807 12-bit, 53 Msps A/D converter
    - Two Texas Instruments DAC902 12-bit, 165 Msps D/A converter
- Tektronix TLA 720 Logic Analyzer with two modules
  - TLA 7N2: 68 Channel logic analyzer module with MagniVu acquisition
  - TLA 7PG2: 64 Channel pattern generator module
- Agilent 33250A
  - 80 MHz Function/Arbitrary Waveform Generator
- HP/Agilent 54825A infiniium Oscilloscope
  - 4 Channels, 500 MHz, 2 GSa/s

The expansion modules enable the usage of an analog input signal, although only digital input signals are used in the measurements to verify correct operation of the FPGA. The clock signal was supplied from the waveform generator to the FPGA through the analog expansion module, and it will be used to convert analog signals to digital in future work. The FPGA evaluation board with expansion modules is shown in Figure 5.2. In addition, the digital oscilloscope was used to verify correct detection times when various detection times were used, as well as for signal waveform verification.

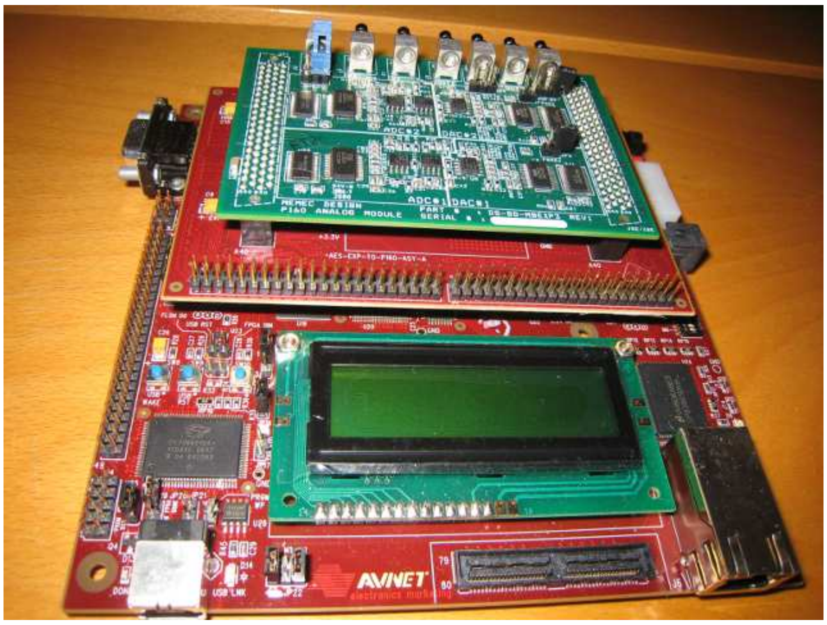


Figure 5.2: Avnet Xilinx Virtex-5 LX50 evaluation board with expansion modules.

5.2 Measurements with various detection times

To verify correct operation of the FPGA implementation, the implementation is measured using various detection times. The input signal generated with MATLAB consists of sequential detections. Due to the input vector size limit of the pattern generator, only 250 sequential detections are measured at a single point. The detection time of a single detection curve in this measurement varies from 512 to 4096 samples. The measured curves are represented in Figure 5.3. The results are effectively similar to the detection time simulation with the MATLAB and VHDL description simulations. The results verify that autocorrelation-based spectrum sensing algorithm is capable of sensing the primary user signal at 90 percent probability on SNRs as low as -7 dB, if a detection time of 4096 samples is used.

5.3 Measurements with and without DC offset compensation

The DC offset degrades the performance of the detection algorithm, therefore, it is important to compensate for its effects. The DC offset compensation performance is measured with three measurement sets, thus obtaining three specific curves. In the DC offset measurements, the DC offset is added to the generated signal relative to noise power. A DC offset of -18 dB relative to noise power was added to the second and third input signal curves, and the first curve is without any added DC offset. The DC offset compensation is turned off in the first two measurements, where in the first measurement no DC offset is added, and in the second, a DC offset of -18dB is added. In the third measurement the DC offset of -18dB is added,

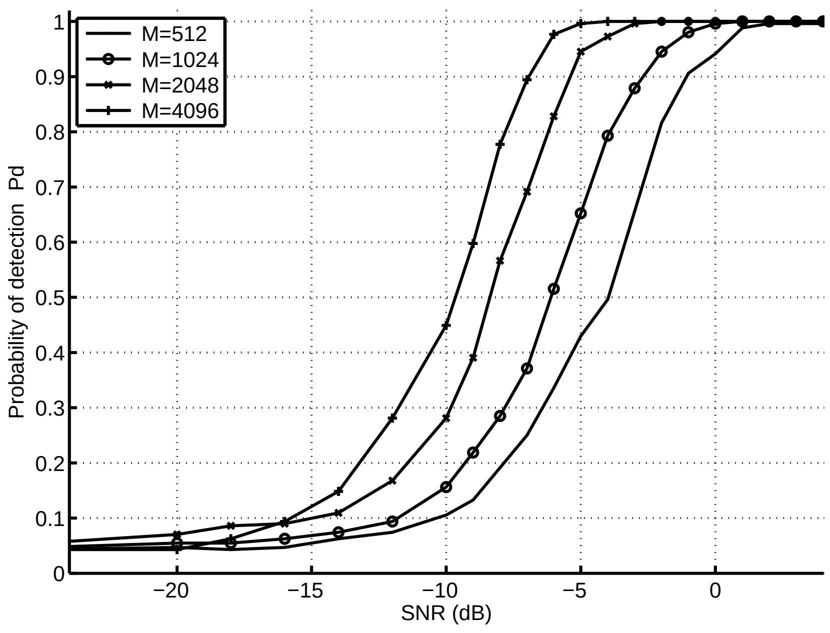


Figure 5.3: Measured probability of detection with detection time varying from 521 to 4096 samples, P_{fa}=0.05. Performance of the implementation is as effective as suggested by simulations.

and the DC offset compensation is enabled. The measured results are shown in Figure 5.4.

The resulting curves clearly show the performance of the DC offset compensation. DC offset compensation effectively compensates for the DC offset, but does not degrade detection performance. Actually, the performance seems to increase when DC offset is added but, on the contrary, due to the false detections initiated by the DC offset, the probability of detection does not tend to the theoretical constant false alarm rate of 5%. Therefore, the algorithm falsely assumes that the primary user is present when it actually is not, and significant amount of spectral opportunities are lost.

The DC offset compensation consumes a high percentage of implementation's total resources, but no DC offset cancellation is required from the analog hardware. Therefore, implementing DC offset compensation in the algorithm simplifies the hardware design of the cognitive radio, as the sensing algorithm is not affected by the DC offset.

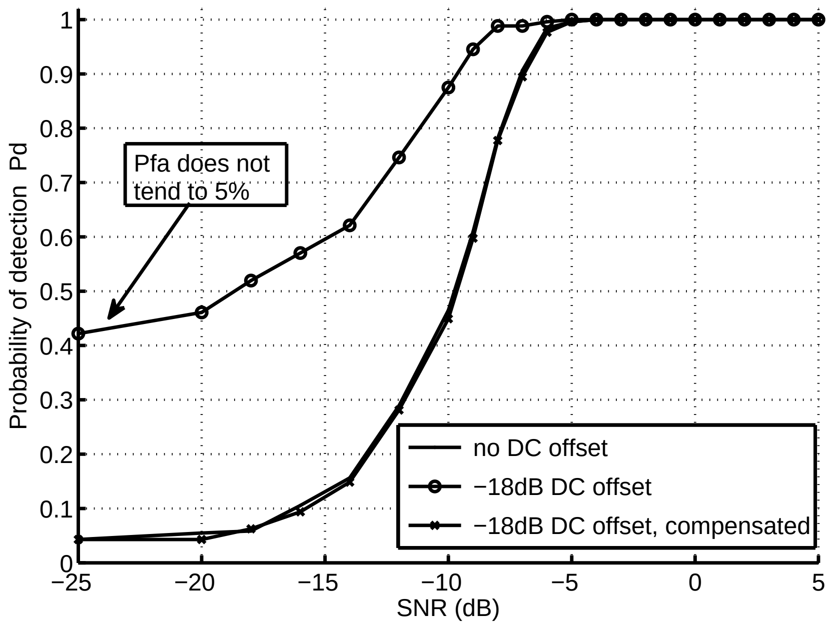


Figure 5.4: Measured probability of detection with and without DC compensation, P_{fa}=0.05. With compensation, the effect of DC offset is completely removed.

Discussion

As stated in the beginning of this thesis, the latest applications demand higher and higher data rates. Unfortunately, the spectrum is almost completely allocated, and only a few bands have been released for new applications, such as analog television broadcasts that have been discontinued in various countries. To solve the problem, cognitive radios were introduced. Cognitive radios are able to adapt to their environment, and can opportunistically take under-utilized spectrum for their own use, when licensed spectrum users are not present. Cognitive radio requirements and challenges were introduced to understand the difficulties which need to be addressed in cognitive radio development.

Spectrum sensing is the key task enabling cognitive radio operation. Common sensing algorithms were examined, and their requirements and feasibility for implementation were reviewed. The autocorrelation-based sensing algorithm was chosen for implementation. Simulations showed the performance of the algorithm. The algorithm is able to distinguish the signal under detection effectively under the noise floor. It has a relatively simple implementation compared to the most advanced sensing methods, yet it has high detection performance. Simulations also revealed how sensitive the algorithm was to the DC offset, therefore it's effects had to be compensated. The algorithm was modified to compensate for the DC offset, and simulations verified that the DC offset was effectively eliminated without performance degradation. Although, additional hardware is required to implement the DC offset compensation.

The autocorrelation-based spectrum sensing algorithm with DC offset compensation was implemented, and the building blocks were reviewed thoroughly. Subsequent to verification steps, the implementation blocks were described with VHDL descriptions, and they were simulated by functional and timing simulations. A few blocks needed high level adjustments to obtain exact timing. Nevertheless, the implementation functioned properly after specific fixes. The final VHDL descriptions were used to implement the design on an FPGA evaluation board to verify the performance with measurements. The measurements validated the performance suggested by the simulations, and DC offset compensation functioned effectively.

The implemented algorithm might be used as a part of the sensor node in a sensor network, therefore it was synthesized to an ASIC to obtain comparable data on area requirements and power consumption. The final ASIC circuit proved to be power efficient. Although, no special design aspects to minimize power consumption were used.

Bibliography

[1] D. Cabric, “Cognitive radios: System design perspective,” Ph.D. dissertation, University of California, Berkeley, 2007.

[2] I. Mitola, J., I. Mitola, J., and J. Maguire, G.Q., “Cognitive radio: making software radios more personal,” *IEEE Commun. Mag.*, vol. 6, no. 4, pp. 13–18, 1999.

[3] S. Chaudhari, V. Koivunen, and V. Poor, “Autocorrelation-based decentralized sequential detection of ofdm signals in cognitive radios,” *IEEE Trans. Signal Processing*, vol. 57, no. 7, pp. 2690–2700, July 2009.

[4] S. Brown and Z. Vranesic, *Fundamentals of Digital Logic with VHDL Design*. McGraw-Hill, 2009.

[5] A. Percey, “Advantages of the virtex-5 fpga 6-input lut architecture, wp284(v1.0),” Xilinx, Tech. Rep., 2007.

[6] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing - Principles, Algorithms, and Applications, 4th Ed.* McGraw-Hill, 2007.

[7] J. W. Cooley and J. W. Tukey, “An algorithm for the machine computation of the complex fourier series,” *Math. Comput.*, vol. 19, pp. 297–301, 1965.

[8] I. F. Akyildiz, W.-Y. Lee, M. C. Vuran, and S. Mohanty, “Next generation/dynamic spectrum access/cognitive radio wireless networks: a survey,” *Computer Networks*, vol. 50, no. 13, pp. 2127–2159, 2006.

[9] S. Haykin, “Cognitive radio: brain-empowered wireless communications,” *"IEEE J. Select. Areas Commun."*, vol. 23, no. 2, pp. 201–220, 2005.

[10] R. Thomas, L. DaSilva, and A. MacKenzie, “Cognitive networks,” in *in Proc. IEEE DySPAN 2005*, November 2005, pp. 352–360.

[11] F. K. Jondral, “Software-defined radio: basics and evolution to cognitive radio,” *EURASIP J. Wirel. Commun. Netw.*, vol. 2005, no. 3, pp. 275–283, 2005.

[12] B. Razavi, “Design considerations for direct-conversion receivers,” *IEEE Trans. Circuits Syst. II*, vol. 44, no. 6, pp. 428–435, 1997.

[13] T. Yucek and H. Arslan, “A survey of spectrum sensing algorithms for cognitive radio applications,” *IEEE Commun. Soc. Mag.*, vol. 11, no. 1, pp. 116–130, 2009.

[14] H. Urkowitz, “Energy detection of unknown deterministic signals,” *Proc. IEEE*, vol. 55, no. 4, pp. 523–531, April 1967.

[15] W. Gardner and L. Franks, “Characterization of cyclostationary random signal processes,” *IEEE Trans. Inform. Theory*, vol. 21, no. 1, pp. 4–14, Jan 1975.

[16] A. Dandawate and G. Giannakis, “Statistical tests for presence of cyclostationarity,” *IEEE Trans. Signal Processing*, vol. 42, no. 9, pp. 2355–2369, 1994.

[17] D. Middleton, “On new classes of matched filters and generalizations of the matched filter concept,” *Information Theory, IRE Transactions on*, vol. 6, no. 3, pp. 349–360, June 1960.

[18] G. Ganesan and Y. Li, “Cooperative spectrum sensing in cognitive radio, part i: Two user networks,” *"IEEE Trans. Wireless Commun."*, vol. 6, no. 6, pp. 2204–2213, 2007.

[19] S. Chaudhari, J. Lunden, and V. Koivunen, “Collaborative autocorrelation-based spectrum sensing of ofdm signals in cognitive radios,” in *Information Sciences and Systems, 2008. CISS 2008. 42nd Annual Conference on 19-21 March 2008 Page(s):191 - 196*, 2008.

[20] G. Ganesan and Y. Li, “Cooperative spectrum sensing in cognitive radio, part ii: Multiuser networks,” *"IEEE Trans. Wireless Commun."*, vol. 6, no. 6, pp. 2214–2222, 2007.

[21] D. Cabric, S. Mishra, and R. Brodersen, “Implementation issues in spectrum sensing for cognitive radios,” in *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Eighth Asilomar Conference on*, vol. 1, 2004, pp. 772–776 Vol.1.

[22] V. Turunen, M. Kosunen, S. Kallioinen, A. Parssinen, and J. Ryynanen, “Spectrum estimator and cyclostationary detector for cognitive radio,” in *Circuit Theory and Design, 2009. ECCTD 2009. European Conference on*, Aug. 2009, pp. 283–286.

[23] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, 1972.

[24] R. Tandra and A. Sahai, “Fundamental limits on detection in low snr under noise uncertainty,” in *Wireless Networks, Communications and Mobile Computing, 2005 International Conference on*, vol. 1, June 2005, pp. 464–469 vol.1.

[25] W. A. Gardner, A. Napolitano, and L. Paura, “Cyclostationarity: Half a century of research,” *Signal Processing*, vol. 86, no. 4, pp. 639–697, Apr. 2006. [Online]. Available: http://www.sciencedirect.com/science/article/B6V18-4H21H3X-1/2/13250f9017b0377f4e8083d8

[26] J. G. Proakis, *Digital Communications, 4th Ed.* Pearson Prentice Hall, 2001.

[27] J. Lunden, S. A. Kassam, and V. Koivunen, “Nonparametric cyclic correlation based detection for cognitive radio systems,” in *Proc. Int. Conf. on Cognitive Radio Oriented Wireless Networks and Communications*, 2008, pp. 1–6.

[28] R. Viswanathan and P. Varshney, “Distributed detection with multiple sensors i. fundamentals,” *Proceedings of the IEEE*, vol. 85, no. 1, pp. 54–63, 1997.

[29] G. Ganesan and Y. Li, “Cooperative spectrum sensing in cognitive radio networks,” in *Proc. First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks DySPAN 2005*, 2005, pp. 137–143.

[30] N. S. Shankar, C. Cordeiro, and K. Challapali, “Spectrum agile radios: utilization and sensing architectures,” in *Proc. First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks DySPAN 2005*, 8–11 Nov. 2005, pp. 160–169.

[31] J. Neyman and E. Pearson, “On the problem of the most efficient tests of statistical hypotheses,” *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, vol. 231, pp. 289–337, 1933.

[32] Y. Li and W. Chu, “Implementation of single precision floating point square root on fpgas,” in *FPGAs for Custom Computing Machines, 1997. Proceedings., The 5th Annual IEEE Symposium on*, apr 1997, pp. 226 –232.

[33] J. E. Volder, “The cordic trigonometric computing technique,” *Electronic Computers, IEEE Transactions on*, vol. EC-8, no. 3, pp. 330 –334, sept. 1959.

[34] A. Abidi, “Direct-conversion radio transceivers for digital communications,” *Solid-State Circuits, IEEE Journal of*, vol. 30, no. 12, pp. 1399–1410, Dec 1995.

[35] K. Kokkinen, V. Turunen, M. Kosunen, S. Chaudhari, V. Koivunen, and J. Ryynänen, “Fpga implementation of autocorrelation-based feature detector for cognitive radio,” in *NORCHIP, 2009*, Nov. 2009, pp. 1–4.

[36] S. Oberman and M. Flynn, “Division algorithms and implementations,” *Computers, IEEE Transactions on*, vol. 46, no. 8, pp. 833 –854, aug 1997.

[37] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design 3rd Ed.* Morgan Kaufmann Publishers, 2007.

[38] M. Jamali, J. Downey, N. Wilikins, C. Rehm, and J. Tipping, “Development of a fpga-based high speed fft processor for wideband direction of arrival applications,” in *Radar Conference, 2009 IEEE*, May 2009, pp. 1–4.

Appendix A

Implementation layout

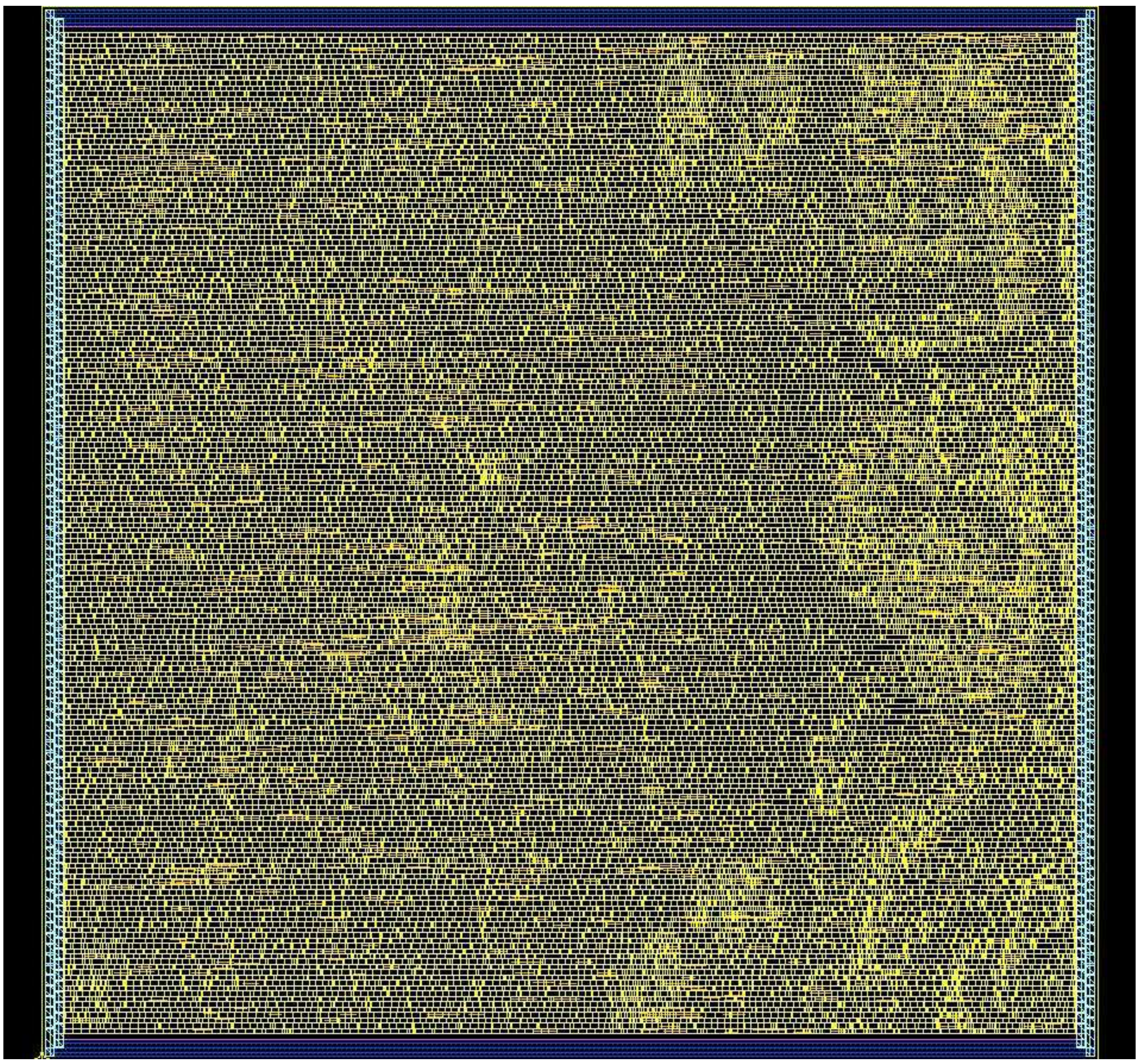


Figure A.1: IC implementation with 65 nm CMOS process. Layout size is 0.254 mm^2, and power consumption 1.019 mW.