RESOURCE USAGE OPTIMIZATION IN SOFTWARE DEFINED RADIO SYSTEMS

By

Sameh Yassin Rashad

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications Engineering

FACULTY OF ENGINEERING, CAIRO UNIVERSITY

GIZA, EGYPT

2014

RESOURCE USAGE OPTIMIZATION IN SOFTWARE DEFINED RADIO SYSTEMS

By

Sameh Yassin Rashad

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications Engineering

Prof. Dr. M. Hazim Tawfik

.....

Professor of Communications
Electronics and Communications
Department
Faculty of Engineering, Cairo University

Dr. Hossam. A. H. Fahmy

.....

Associate Professor of Computer
Electronics and Communications
Department Faculty of Engineering, Cairo
University

FACULTY OF ENGINEERING, CAIRO UNIVERSITY

GIZA, EGYPT

2014

RESOURCE USAGE OPTIMIZATION IN SOFTWARE DEFINED RADIO SYSTEMS

By

Sameh Yassin Rashad

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications Engineering

Approved by
The Examining Committee

_____
Prof. Dr. Ahmed Abu Auf

_____
Prof. Dr. Magdi Fikri Ragaey

_____
Prof. Dr. M. Hazim Tawfik

_____
Dr. Hossam A. H. Fahmy

FACULTY OF ENGINEERING, CAIRO UNIVERSITY

GIZA, EGYPT

2014

Engineer's Name:	Sameh Yassin Rashad
Date of Birth:	08/09/1985
Nationality:	Egyptian
E-mail:	syassin@eece.edu.eg sameh_yassin99@yahoo.com
Phone:	+201003781998
Address:	El Haram, El Galoly Street, Building 1
Registration Date:	01/10/2008
Degree:	Master of Science
Department:	Electronics and Communications
Supervisors:	Prof. Dr. M. Hazim Tawfik Dr. Hossam. A. H. Fahmy
Examiners:	Prof. Dr. Ahmed Abu Auf Prof. Dr. Magdi Fikri Ragaey Prof. Dr. M. Hazim Tawfik Dr. Hossam. A. H. Fahmy

Title of Thesis:
Resource Usage Optimization In Software Defined Radio Systems

Key Words:
Software Defined Radio; Resource Utilization; FPGA; Channel Equalization

Summary:

Software Defined Radio is a flexible platform that can provide dynamic reconfiguration for communication systems using software only. In other words the same hardware can be used to implement different transceiver functions such as modulation, detection and channel estimation.

With the ever-increasing need for higher data rates and stable performance, SDR systems importance is growing. In addition, it is vital to provide communication services globally including rural and poor areas. This thesis investigates SDR systems to provide the rural communities in the developing countries with cheap and basic telecom services such as voice communication. Many governments and agencies in the developing countries are focusing on extending telecommunications services into rural areas, as they seek to alleviate poverty, encourage economic and social growth, and overcome a perceived 'digital divide'.

SDR systems face three challenges; namely increased electric energy consumption, increased resource utilization, and increased processing time. There is a great opportunity to introduce a cost effective communication network, by overcoming these challenges.

This work is focused on reducing resource utilization, to increase the number of the offered services per the system deployed. Intuitively, this in turn will improve the power consumption. This is because any system module with same functionality and less resource usage, will consume less energy. However, the processing time for the SDR system may increase due to the optimized modules. Therefore, a trade off is needed to reduce the resource usage, while maintaining the real time requirements of the SDR system. The contribution of this work is to address this tradeoff. The results are verified using system simulation, and experimental results using the open source hardware Universal Software Radio Peripheral (USRP).

Acknowledgements

For my family; my father, my mother, my sister, and my brother.

For my teachers.

For my brilliant work mates.

For NTRA.

Table of Contents

List of Figures

List of Tables

List of Acronyms

SDR	Software Defined Radio
DFE	Decision Feedback Equalizer
DCD	Dichotomous Coordinate Descent
RLS	Recursive Least Squares
OpenBTS	Open source Base Transceiver Station project
USRP	Universal Software Radio Peripheral
DSP	Digital Signal Processing
DDC	Digital Down Conversion
DUC	Digital Up Conversion
GPMC	General Purpose Memory Controller
f_{wl}	Fractional part Word Length
i_{wl}	Integer part Word Length
BER	Bit Error Rate
SNR	Signal to Noise Ratio
MSE	Mean Square Error
GSM	Global System for Mobile Communicsations
ToA	Time of Arrival
VP	Valley Power
FSM	Finite State Machine
FPGA	Field Programmable Gate Array
FIR	Finite Impulse Response Filter
GMSK	Gaussian Minimum Shift Keying
LMS	Least Mean Squares
FTF	Fast Transversal Filter
FAEST	Fast Aposterior Error Technique
IDE	Integrated Development Environment
HDL	Hardware Description Language
BRAM	Block Random Access Memory
GCC	GNU Compiler Collection
UHD	Universal Hardware Driver
VRT	VITA Radio Transport Protocol

Chapter 1 Introduction

1.1 Motivation

Software Defined Radio is a flexible platform that can provide dynamic reconfiguration for communication systems using software only. In other words the same hardware can be used to implement different transceiver functions such as modulation, detection and channel estimation. One motivation for the rapid development of SDR systems, is the need to add new features to the radio equipment or to upgrade its functionality. This is needed by important sectors such as military and public safety [1]. In these sectors, special purpose radios were the norm rather than the exception. One radio device is needed for a few number of functions or wave forms, and it is not straightforward to alter the device functionality.

With the ever-increasing need for higher data rates and stable performance, SDR systems importance is growing. In addition, it is vital to provide communication services globally including rural and poor areas. This thesis investigates SDR systems to provide the rural communities in the developing countries with cheap and basic telecom services such as voice communication. Many governments and agencies in the developing countries are focusing on extending telecommunications services into rural areas, as they seek to alleviate poverty, encourage economic and social growth, and overcome a perceived 'digital divide'.

SDR systems face three challenges; namely increased electric energy consumption, increased resource utilization, and increased processing time [2]. There is a great opportunity to introduce a cost effective communication network, by overcoming these challenges. This work is focused on reducing resource utilization, to increase the number of the offered services per the system deployed. Intuitively, this in turn will improve the power consumption. Because any system module with same functionality and less resource usage, will consume less energy. However, the processing time for the SDR system may increase due to the optimized modules. Therefore, a trade off is needed to reduce the resource usage, while maintaining the real time requirements of the SDR system. The contribution of this work is to address this tradeoff.

1.2 Related Work

The related work can be classified into two streams. The first stream is related to SDR optimization under performance constraints, such as [3], [4], and [5]. The second stream, is related to implementation of low complexity channel equalization techniques such as [6], [7], and [8].

Now, the first stream of related work is presented. In [3], the main motivation was to reduce the energy consumption, and the complexity requirements to satisfy an embedded system needs. Therefore, fixed point arithmetic was used to present an optimized WCDMA receiver. The merit for this work was finding an analytical relation between the receiver performance, and the number of bits needed to satisfy energy consumption constraints. Using this approach, the minimum data width can be estimated with negligible performance degradation compared to floating point arithmetic. However, to deduce the minimum data width one cannot reach closed form expression for some equations. For SDR systems, it is impractical to update the performance equations each time the system functions are modified.

In [4], the work highlighted the need to implement complex signal processing algorithms in fixed point. The authors developed a metric to compensate for the loss in accuracy due to the conversion from floating point to fixed point. That metric is system dependent, and needs to be dimensioned carefully for each application.

In [5], the concept of scalable SDR was introduced for battery powered devices. To achieve the two contradicting requirements of fast time to market and minimum energy consumption, a new method was presented to deal with fixed point arithmetic. The new method considered the changes in data format such as modulation scheme, and number of antennas. Accordingly data width can be adjusted to save energy consumption of the battery. However, the savings in power consumption came at the expense of increased resource utilization.

Now we move to the second stream of related work. In [6], the complexity was reduced by observing that in communication systems the input data has shift structure. In [7], spectral factorization was used to calculate the Decision Feedback Equalizer (DFE) coefficients in fast and efficient way. In [8], a new Dichotomous Coordinate Descent (DCD) algorithm was developed to solve the Recursive Least Squares (RLS) equations.

Related work proposed different solutions, but they all have one thing in common, which is the reduction of the total number of required mathematical operations. Hence, it is not straightforward to compare between different DFE algorithms. In this work, we will introduce a new metric to compare fairly between DFE algorithms, by considering the implementation of each mathematical operation.

1.3 Thesis Scope

The focus of this work is to optimize the resource usage in a SDR system, taking into consideration the real time requirements of embedded system. All simulations and experiments are investigated for a generic SDR system. Without loss of generality, the proposed enhancements are validated by a case study of a recent SDR system; namely the Open BTS project.

1.3.1 Generic block diagram of SDR

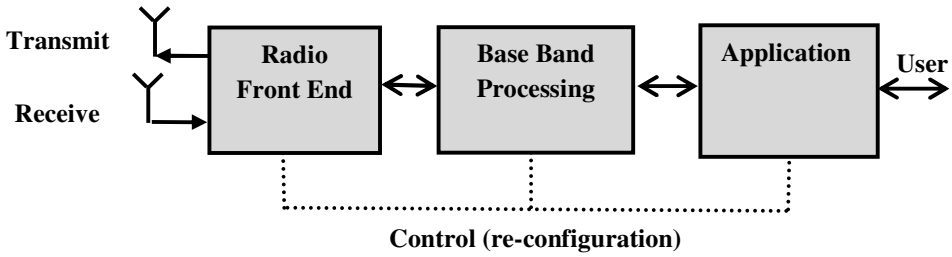


Figure 1-1 - Software Defined Radio Aspects

A SDR system is a generic communication system starting with user data layer and ending with physical data layer [9]. The generality of a SDR system is due to its ability to reconfigure the system modules without changing the hardware. This is obtained by adding control to a communication system as shown in Figure 1-1. A SDR system consists of radio front end, base band processing, control bus, and application. These modules are combined to map the required data from the application into physical signal at arbitrary carrier frequency. The application is modified during design and/or execution time by sending reconfiguration messages through the control bus. Hence, to modify the functionality of the system, only the application is updated using software configuration.

Without loss of generality, one SDR project will be studied in this work to show the effectiveness of our proposed methods. This project is entitled "OpenBTS" which is an open source implementation of a low cost mobile network. A complete mobile network can be implemented using one Universal Software Radio Peripheral (USRP), with a personal computer, or using one Embedded USRP. The embedded USRP is preferred because it can be deployed in practically, not only for development purposes.

1.3.2 The OpenBTS Project

The OpenBTS is based on the embedded family of USRP, which is called E1x0[1] family. The embedded USRP family is generic and can be customized to perform the BTS functions. As was described in Section 1.3.1, the OpenBTS project consists of three parts; namely the application, the

[1] E1x0 family has two members, the E100 and the E110

base band processing, and the radio front end. The application is the OpenBTS source C++ project that can be downloaded and modified. The base band processing part is implemented on the FPGA, as well as the ADC/DAC operating at a sample frequency of 52 MHz. The front end is fully customized and can be flexibly modified by only changing a daughter board. Each daughter board has a range of frequency operation. In this work any daughter board may be used. As an example the Wide Bandwidth Transceiver (WBX) with range of operation 50MHz to 2.2 GHz is used in our work. The OpenBTS project can be projected to generic SDR block diagram as shown in Figure 1-2.

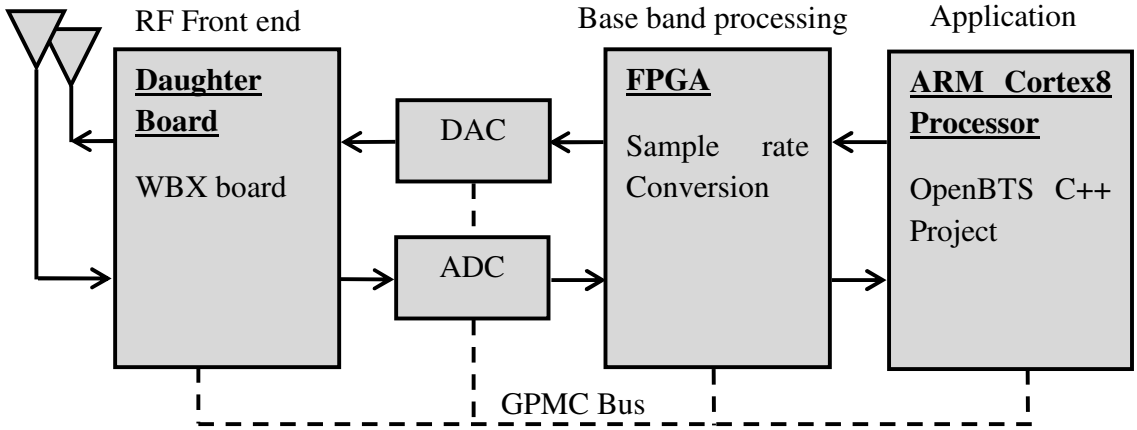


Figure 1-2: OpenBTS block diagram as a SDR system

When the OpenBTS project is run, it was observed that the processing power of the ARM CORTEX 8 can only support a single RF carrier with one call at a time instead of 7 calls. Since there is room inside the FPGA for user expansion, some software functions are moved from CPU into FPGA to reduce the CPU utilization. The functions that are assigned to the FPGA in the current OpenBTS project can be summarized as follows:

- Digital Up Conversion (DUC),
- Digital Down Conversion (DDC),
- Bus Interfacing using General Purpose Memory Controller (GPMC), and
- Control functions
    - Daughterboard Control
    - Testing and monitoring

Channel equalization, and its pre-processing functions will be moved to FPGA, because they have the highest execution time. Since the CPU and FPGA communicate through the VITA Radio Transport (VRT) protocol, it should be noted that communication will not be changed in order to avoid communication overhead. The communication overhead can increase the processor utilization and outweigh the benefit of moving functions to the FPGA.

1.3.3 BTS Performance Improvement

A new process is proposed to design any new SDR system, or add features to an existing one. This process will be applied to the OpenBTS project to show its effectiveness. A simulation based approach is chosen because it is more adequate to the nature of programmable SDR systems in terms of supporting new features in short design cycle, and coping with fast market changes. The added value of this process is the link between system performance and computational accuracy. A similar method was proposed in [10] using an analytical approach. The analytical approach was chosen there because it needed less execution time. However, obtaining closed form expressions for each application is not straightforward, and may have to be solved numerically. Therefore, the increase in problem complexity may outweigh the decrease in execution time. Moreover, a SDR system needs to be flexible to changes in system functions. It is impractical to update the performance equations each time the system functions are modified.

Moreover, the design process enables both the system architect, and the designer to discover modules that affect system performance in terms of resource usage, such as channel equalization. Therefore, a great attention is paid for selection and implementation of channel equalization module. A novel metric is developed to compare fairly between different channel equalization algorithms. Significant resource savings can be accomplished by using the proposed metric.

Finally, a new method is developed to implement channel equalization algorithms efficiently. It is usually advised to avoid using algorithms containing division operations [11]. This is advised to save resource usage when the algorithms are implemented in programmable devices. However, using the proposed method, it is permitted to use the division process for implementing equalization algorithms. Hence, it is now permitted to use advanced equalization algorithms in terms of channel tracking capability, containing division operations under reduced complexity constraints.

The contribution of this thesis is summarized by the following three points

- A new simulation based process to design any SDR system under development, or add features to an existing one.
- A novel metric to evaluate the complexity of different DFE is introduced which can significantly reduce the effort to choose a suitable algorithm for implementation.
- A new method is proposed to implement the division process with minimum resource usage.

1.4 Thesis Organization

The rest of this thesis is organized as follows: In Chapter 1, a case study of SDR systems is presented; namely Open BTS project. Open BTS is explained briefly to highlight the available opportunities to optimize the current system performance, using the proposed design process. In Chapter 2, a new process is proposed to design any new SDR system, or add features to an existing one. In Chapter 3, a comparison is accomplished for an important module of SDR communication system. This module is the channel equalizer. The importance of this module comes from the fact that it consumes high resource utilization relative to other modules in the system. In Chapter 4, simulation and experimental results of the methods developed for SDR and channel equalization are detailed. Finally, the conclusion of this work is presented in Chapter 5.

1.5 Published Papers

Two research papers have been published. The first was published in AICT 2013 in Rome, Italy:

- S. Yassin, and H. Tawfik, "Reduced Complexity Decision Feedback Channel Equalizer using Series Expansion Division", The Ninth Advanced International Conference on Telecommunications, June 2013, pp. 219-223.
- http://www.thinkmind.org/index.php?view=article&articleid=aict_2013_10_10_10064

The second was published in ICONS 2014 in Nice, France:

- S. Yassin, I. R. Kamel, and H. Tawfik " A New Design Process to Reduce Resource Usage in SDR Systems", The Ninth International Conference on Systems, Feb. 2014, pp.1-5.
- http://www.thinkmind.org/index.php?view=article&articleid=icons_2014_1_10_40017

Chapter 2 SDR Resource Usage Optimization

One of the important properties of a SDR is the flexibility in adding new, or modifying current features of a communication system. In order to make use of this property, any added features should not alter the performance of a communication system. Hence, the system performance should be monitored before and after the addition of new features. This chapter presents a design process to achieve this requirement.

The aim of the design process is to calculate an optimum data width that can be used without affecting the system performance. A metric will be developed to compare between system performance in floating, and fixed point representation. The proposed design process is outlined in the flowchart shown in Figure 2-1. The flowchart is similar to the data width optimization performed by Digital Signal Processing (DSP) engineers[3]. The added value of this work, is the application of the optimization process into SDR systems. Moreover, the proposed design process provides the link between system performance and resource usage.

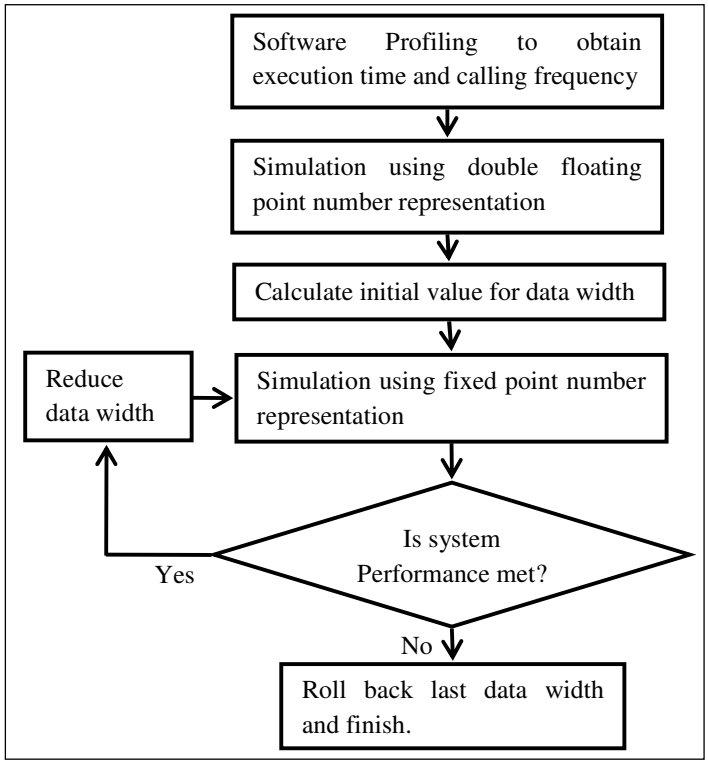


Figure 2-1: Flow chart of the proposed design process

2.1 Proposed design process

Usually, the data width is calculated once for any system under development [9], [11]. In the design process it is proposed to calculate the output data width for each block within the system. The data width is optimized by allocating proper number of bits to the integer part i_{wl}, and the fraction part f_{wl}. The design process contains five steps as shown in Figure 2-1, which will be described next.

The first step is to determine the utilization of processing power by each system function. This is achieved by software profiling[2]. Software profiling is used to measure two factors for each system function; namely its execution time, and the number of times it is called. Both factors are needed to avoid optimizing one function with high execution time when it is only called few times. This can be illustrated by a simple example as shown in Figure 2-2.

At the top of Figure 2-2, the execution time and calling frequency are plotted for two functions. Function 1 and Function 2 have the same execution time, and different calling frequency. In the bottom of Figure 2-2, the multiplication between execution time percentage, and calling frequency is plotted for the two functions. It can be observed that Function 1 has 28% of CPU resources, while Function 2 has only 8% of CPU resources.

The second step is to simulate the system performance using double floating point number. It is efficient to use floating point numbers, when a large dynamic range is required [10]. The dynamic range is an important factor in a digital receiver, and is defined as the ratio between the largest and the lowest signal amplitudes. During the system simulation, the performance metric can be chosen to be either Bit Error Rate (BER) [10], or Signal to Noise Ratio (SNR). In this work, BER is used for comparison, because it can be accurately measured.

The third step, is to determine an initial data width for fixed point simulation as will be discussed in Section 2.2. This initial value will be optimized in the following steps.

The fourth step, is to simulate the system and obtain performance curves using fixed point data width similar to Figure 2-3. The curve for BER versus SNR will be used to compare the performance of both the fixed point, and the floating point systems.

[2] The details of software profiling are deferred to Appendix A.

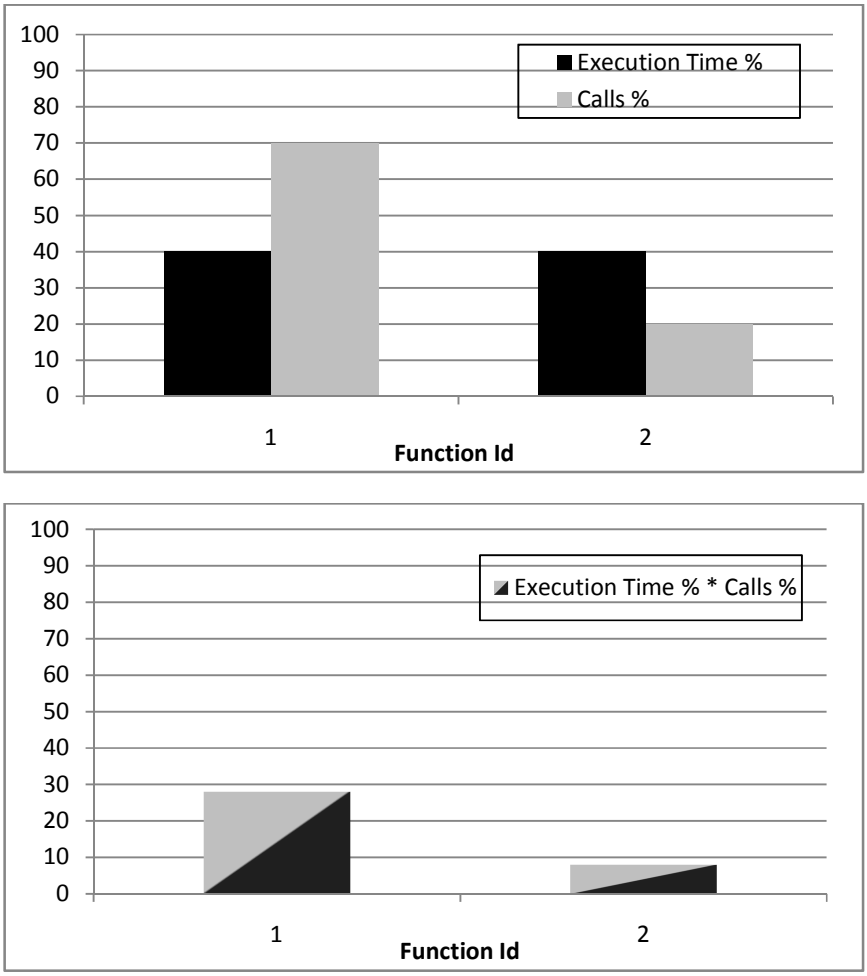


Figure 2-2: Example on software profiling to start the design process

The fifth step, is to compare between the BER curves obtained in steps two and four, as shown in Figure 2-3. By comparing both BER curves, the required increase in SNR to obtain the same BER value can be calculated. The required increase Ɛ in the SNR can be calculated using Equation (2.1).

$$\text{Ɛ} = SNR_{fp} - SNR_0, \tag{2.1}$$

where SNR_0, SNR_{fp} are the SNR for floating point, and fixed point, respectively. The value of Ɛ is a design parameter that is chosen arbitrarily. When the system is desired to have the same BER value for both fixed point and floating point, Ɛ is set to minimum value close to 0 dB. For further savings in resource usage, Ɛ is increased. The fifth step proceeds by decreasing $SNR_{fp} - SNR_0$. If the difference $SNR_{fp} - SNR_0$ is still smaller than Ɛ, then reduce the data width (i_{wl}, f_{wl}). Otherwise, roll back previous value of data width and proceed to the end of the process.

Note that the data width is reduced gradually by reducing the integer part i_{wl} by one bit at a time, until its minimum value is obtained. Afterwards, the fraction part f_{wl} is reduced by one bit at a time until its minimum value is obtained.

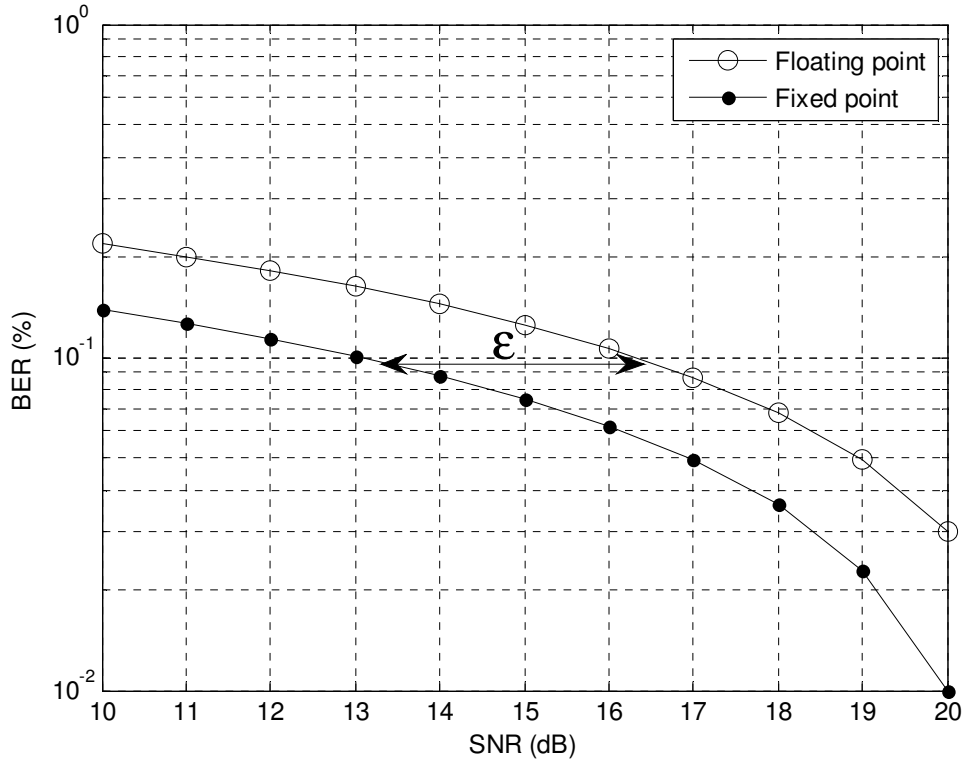


Figure 2-3: Illustrative example for step 5 of the design process

2.2 Dynamic range estimation

To estimate the initial value of data width a signal flow graph for the SDR system under development should be constructed, as shown in Figure 2-4. Afterwards, the dynamic range can be estimated for both receiver and transmitter chains. The initial value for data width will be calculated for receiver chain because it is more complex than the transmitter in terms of the required number of functions. Without loss of generality, the same method can be applied to transmitter chain to calculate its initial data width.

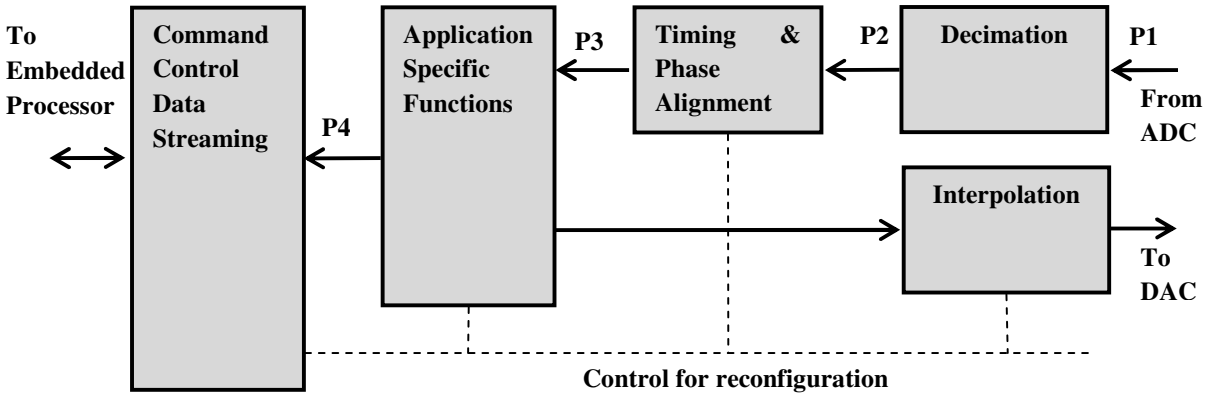


Figure 2-4: General block diagram for SDR systems

For the receiver chain, the points of interest are highlighted and marked as Pi where 1< i < N, where N is the number of blocks with different data width. Next, the initial value of data width can be estimated by observing each point in the receiver chain. At point P1, the data width is the precision of the front end Analog to Digital Converter (ADC), such that

$$wl_{p1} = Accuracy_{ADC}. \tag{2.2}$$

At point P2, the data is prepared for base band processing by reducing its sampling rate using the decimation process. The input stream sampling frequency is reduced by an integer factor called decimation factor *R*, even though only the sample rate is changed, and not the bandwidth of the signal. The input signal bandwidth must be filtered to avoid aliasing. Therefore, the decimation process requires an increase in the data width to maintain proper number of bits per sample. The increase in data width can be calculated, depending on the decimation method of choice. One example is the Cascaded Integrated Comb (CIC) filter. The input signal is fed through one or more cascaded integrator sections, then a down sampler, followed by one or more comb sections [12]. The increase in the data width will be dependent on the differential delay *M* of the comb section, and the number of blocks *N* as in Equation (2.3), where Ceil(A) rounds to the nearest integer greater than or equal to A.

$$wl_{P2} = wl_{P1} + Ceil(N * \log_2(M * R)). \tag{2.3}$$

At point P3, the timing and phase changes relative to the original transmitted signal are estimated. This includes correlation operation, which is composed of addition, shift, and multiplication operations. The multiplication operation particularly leads to the most significant increase in the input data width, which is proportional to the multiplier width n_1, and the multiplicand width n_2 as shown in Equation (2.4).

$$wl_{P3} = wl_{P2} + \sum(n_1 + n_2 - 1). \tag{2.4}$$

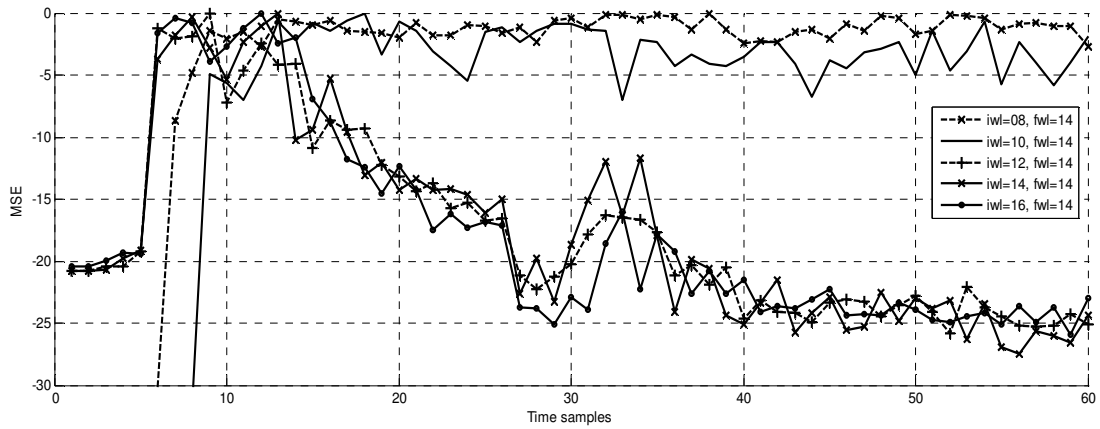
At point P4, application specific functions can also increase the input data width, and may alter the numerical stability due to the use of fixed point such as channel equalization. The data width for this block must maintain numerical stability of the SDR system. In case of channel equalization, the data width should result in quantization noise power that will not alter the computation of the Mean Square Error (MSE) of the equalization algorithm [11]. To illustrate how to maintain numerical stability of equalization algorithm, one channel equalization algorithm, namely the Recursive Least Squares (RLS) is considered.

To maintain numerical stability of RLS, data width should be increased as shown in Figure 2-5(a) . A family of MSE curves is obtained for different data widths ranging from (i_{wl} = 8, f_{wl} = 14) bits, to (i_{wl} = 16, f_{wl} = 14) bits. This family of curves can be used to conclude that the minimum value is i_{wl} = 12, because it keeps MSE decreasing as the time samples advance. To obtain the minimum value for fractional part f_{wl} = 14, a similar family of curves is developed in Figure 2-5(b), but with fixed integer part, and variable fractional part. Finally, the chosen values can be compared to the floating performance as shown in Figure 2-5(c).

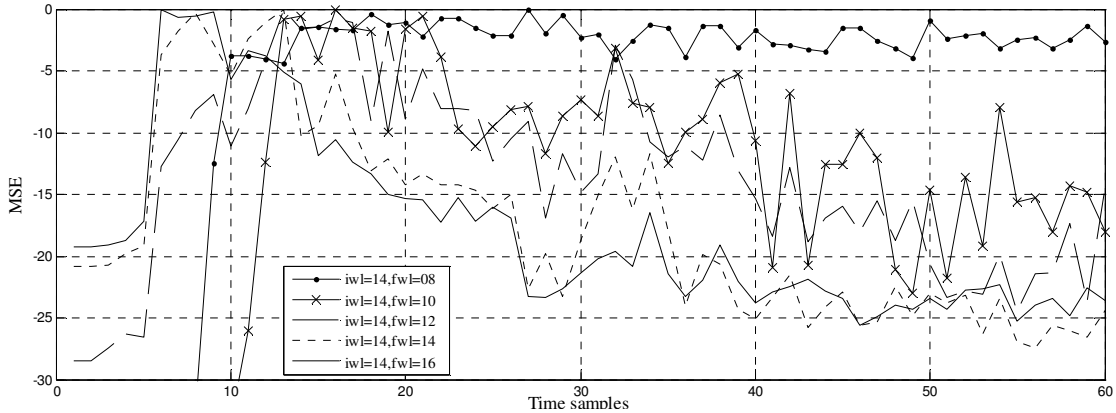
Note that stabilizing the algorithm will increase the data width. Therefore, fast fixed order filters can be used to reduce the data width [22]. To choose an equalizer algorithm with the lowest resource utilization, a fair comparison between different equalizer algorithms was proposed in [23]. In [12], the authors proposed to weigh the resource usage of equalizers by mapping the algorithm in terms of mathematical operations. This fair choice will compensate for the increase in data width. The data width for this block can be calculated as in Equation (2.5), where δ is the required increase in the data width to maintain the numerical stability.

$$wl_{P4} = wl_{P3} + \delta. \tag{2.5}$$

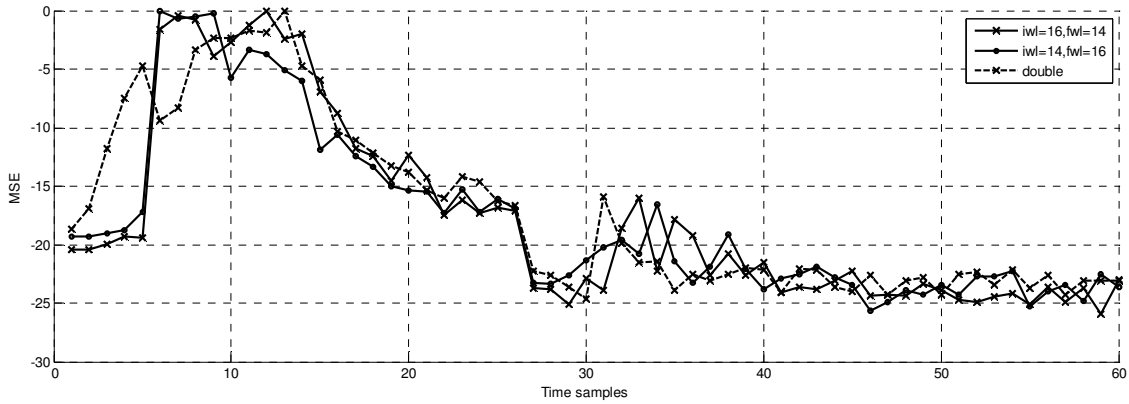
The value wl_{P4} can be used initially for the design process. To validate the design process, a case study will be considered in Section 2.3.



(a)



(b)



(c)

Figure 2-5: Finite precision effects on channel equalization

2.3 Case Study: Open BTS receiver chain

The case study is part of the Open Base Transceiver Station (OpenBTS) project [14] [15], which is based on ETTUS research platform named Universal Software Radio Peripheral (USRP). This platform provides a cheap alternative to standard BTS [16] [17]. Hence, rural communities can enjoy cheap and basic telecommunication services using Global System for Mobile communications (GSM) [32]. The five steps of the design process will be applied to transfer the functions of synchronization from software to FPGA.

The first step is to perform software profiling for the OpenBTS system as shown in Figure 2-6. The function identities of the OpenBTS system appear on the x axis, while the y axis shows both the percentage of processor execution time, and the normalized number of calls of each function. As previously mentioned, only those functions with high execution time and high calling frequency should be optimized.

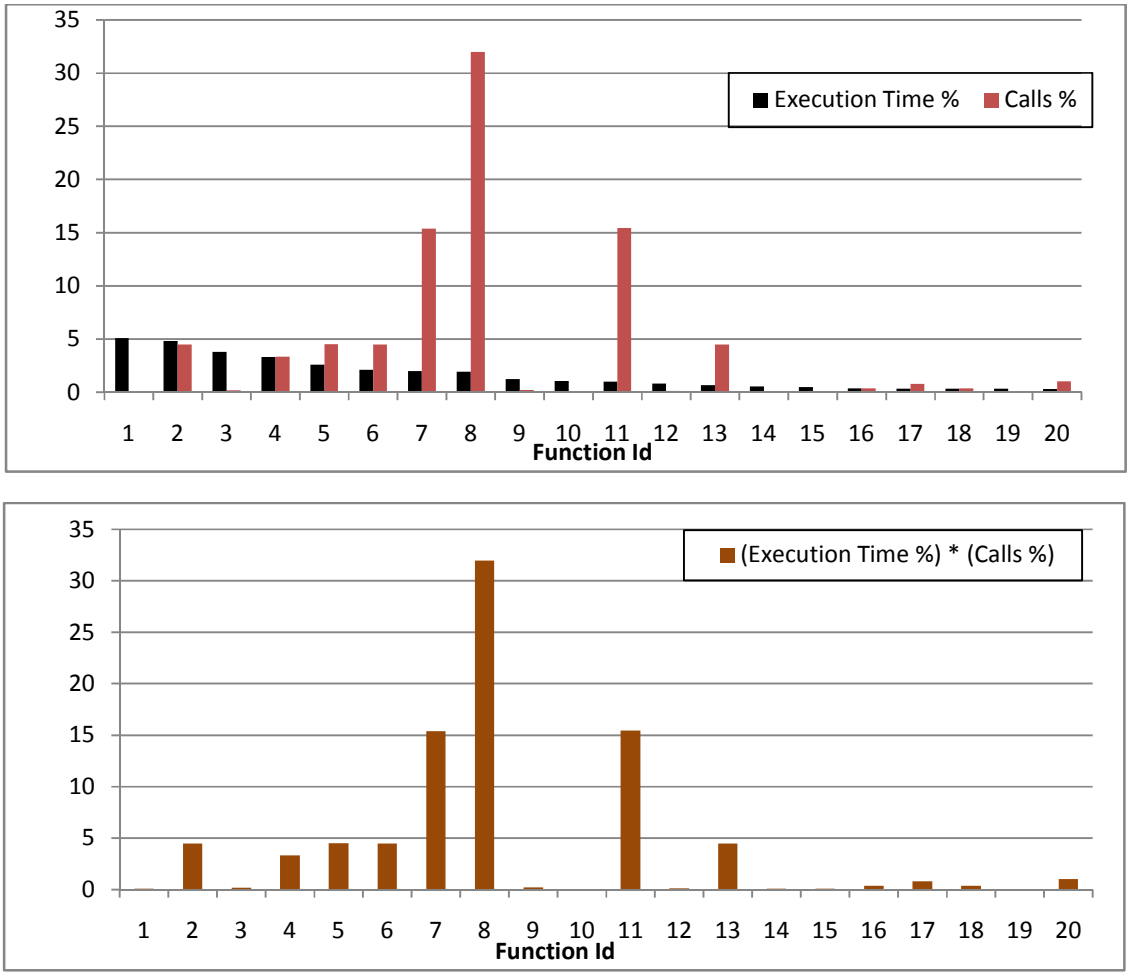


Figure 2-6: Software profiling result for OpenBTS

It can be observed from Figure 2-6 that the functions with identities (7, 8, and 11) have the highest utilization. These functions are grouped into one large function named “Analyze Traffic Burst”. The purpose of the function is to calculate important parameters of the receiver, namely Time of Arrival (ToA), Valley Power (VP), and channel estimation coefficients. The two parameters ToA, and VP are used for synchronization between mobile station and base station.

The second step, is to simulate system performance using double floating point representation as shown in Figure 2-7.

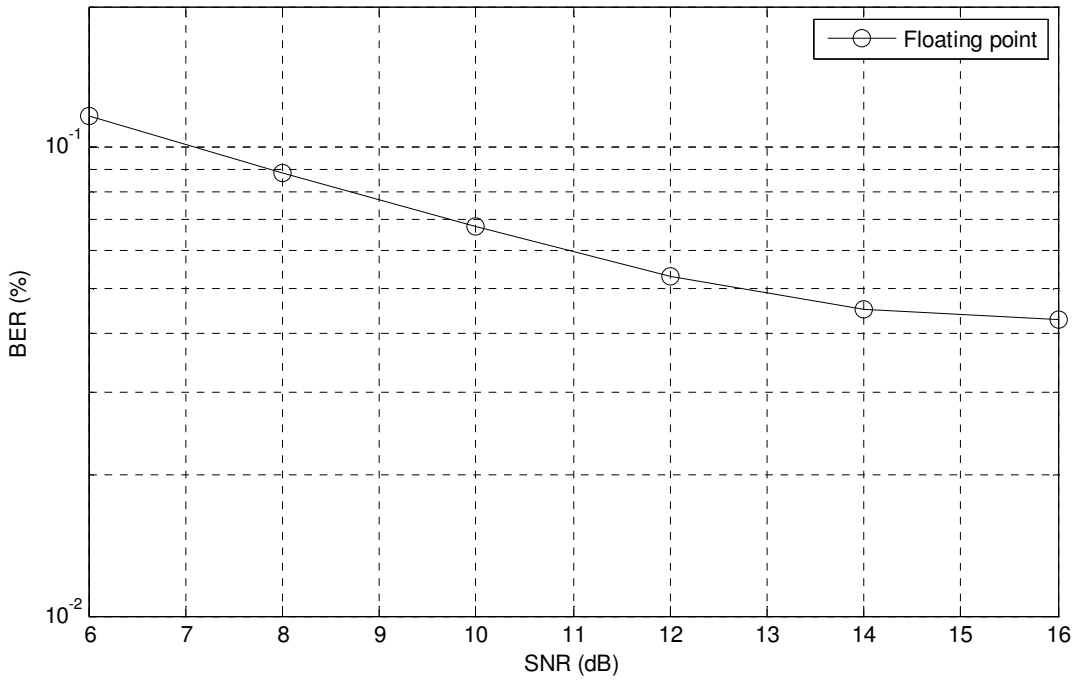


Figure 2-7: Simulation of system performance using floating point

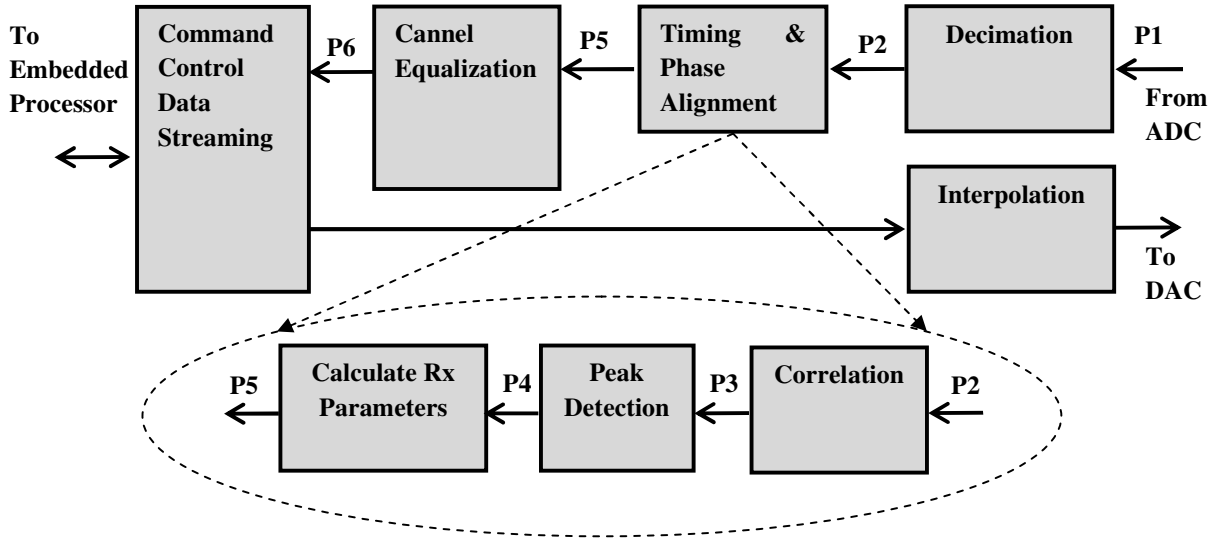


Figure 2-8: Block diagram for the function Analyze Traffic Burst

The third step is to calculate an initial data width for simulation. A block diagram is constructed for the "Analyze Traffic Burst" function as shown in Figure 2-8. At P1, the accuracy of the ADC component in USRP is 12 bits. Then the input data width equals the ADC accuracy $wl_{P1} = 12$.

At P2, decimation is implemented using CIC filter with N = 4 stages and a variable decimation rate log2(M * R) = 7 resulting in $wl_{P2} = (12 + 28)$. However, the original OpenBTS design chose to truncate this value to be $wl_{P2} = (12 + 12)$ to save resource usage. This truncation will not affect the design process, and can be ignored without affecting the obtained results.

At P3, the data width will be doubled due to the multiplication in the correlation function resulting in a data width of $wl_{P3} = (24 + 24 - 1)$.

- **Correlation:** This block has two input vectors with length of 16 and 27 samples respectively. Each sample is represented by data width $wl_{P2} = 24$ bits. This block is implemented as a Systolic Finite Impulse Response (FIR) with Adder Cascade as shown in Figure 2-10. The Systolic FIR is implemented using internal blocks of Spartan-3A DSP FPGA named DSP48A blocks using sixteen blocks. In general, the baseband signal is represented in complex number notation. Therefore, the correlation block has to be able to accept complex inputs and produce complex outputs. To achieve this, complex multipliers should be implemented. The complex multiplication is a resource greedy operation. It has been optimized by using off the shelf ready Xilinx components as will be discussed in Chapter 4.

At P4, the peak detection is divided into two sub blocks, namely Coordinate Rotation Digital Computer (CORDIC) and interpolation. Peak detection can be described by a Finite State Machine (FSM) as in Figure 2-11.

- **CORDIC:** The basic operation is to rotate the x-y axes to eliminate the y component. Hence, the magnitude of the input vector is stored in the x component. The CORDIC block consists of twelve stages. It is implemented by using ready blocks from original OpenBTS FPGA code. However, in this work, the CORDIC is used differently to calculate the 2-norm of each sample. This block is re-used from the original project with modifications to perform this new task. For example, the gain of CORDIC block is compensated to obtain correct outputs.
- **Interpolation:** Interpolating at fractional spacing around an initially determined integer peak index can enhance the accuracy of peak location as illustrated in Figure 2-9. Two early and late points are evaluated around the initially estimated peak. The two points are initially separated by one sample period. The early, late and peak locations are shifted towards the index of the larger of the early/late samples by half the previous shift, and 3 new values; peak, early and late, are interpolated using *sinc* function lookup table. The shift/interpolate process is successively repeated, with the shift halved every repetition, till we get a peak location when the early and late values are almost equal. After 8 repetitions, the peak location is determined with a resolution of 1/256 of one sample period.

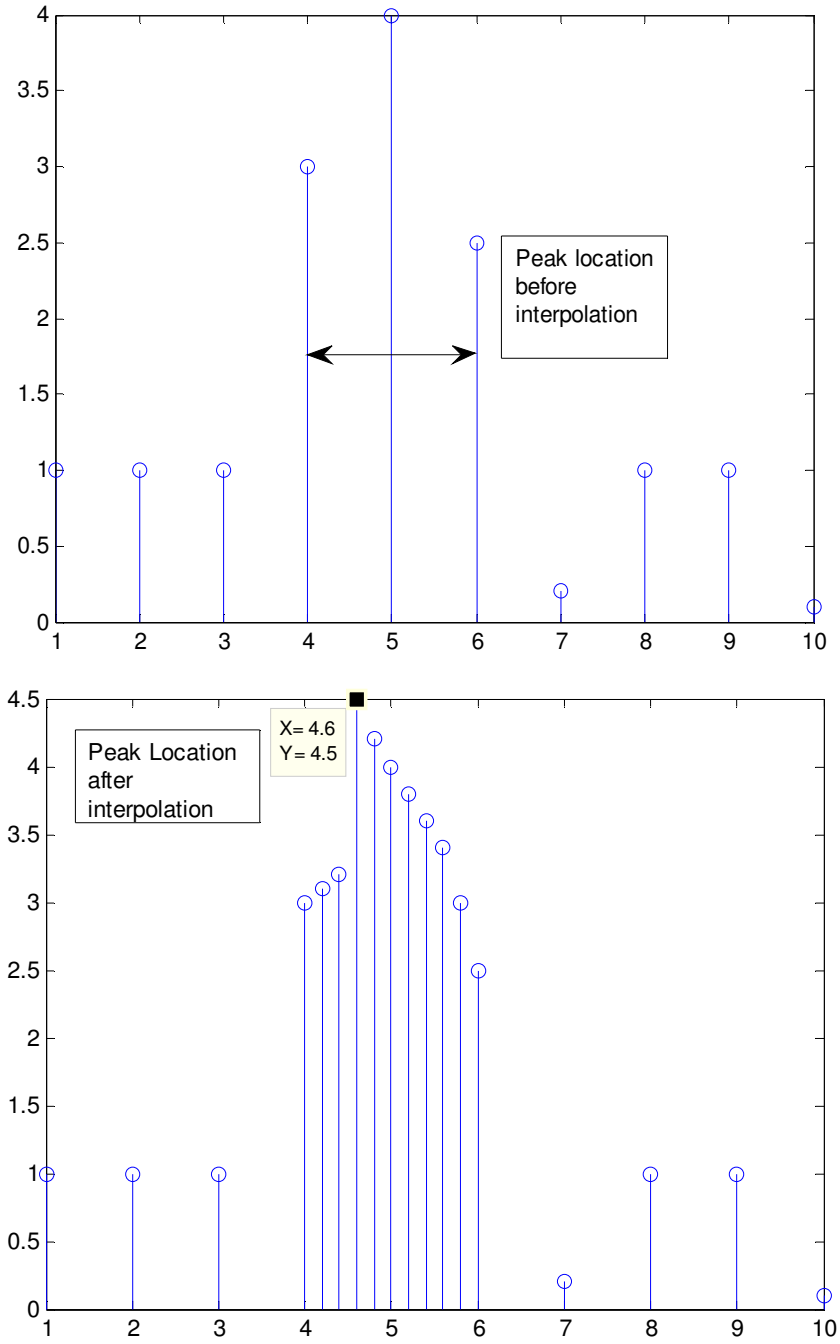


Figure 2-9: Accuracy improvement of peak location

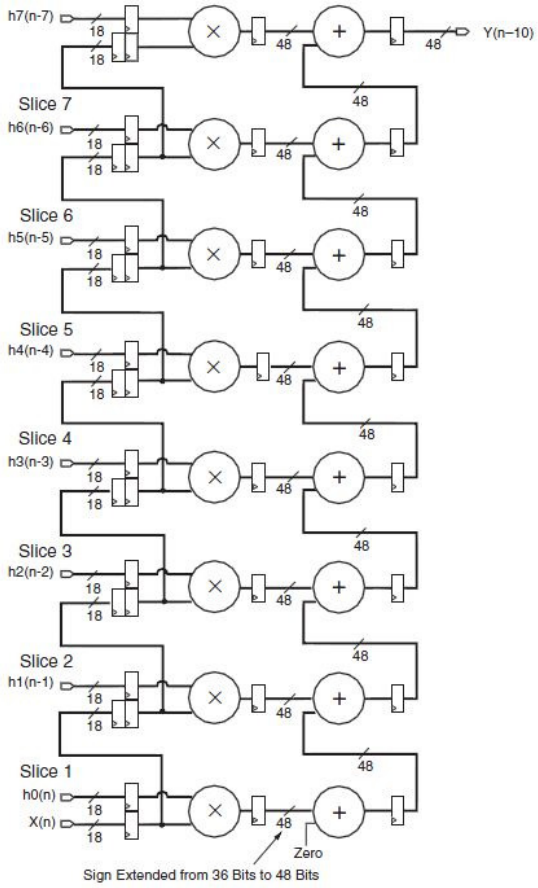


Figure 2-10: Systolic FIR with Adder Cascade [27]

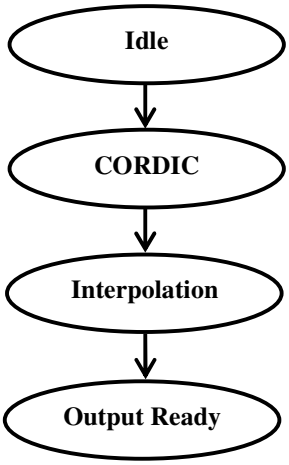


Figure 2-11: FSM describing the Peak Detect block

A SINC function with 1/256 resolution is required by GSM system [32]. This SINC function has maximum integer value of 1, and minimum value of 1/256. Hence, a SINC can be represented with only 8 bits without increasing the data width. Yet, we choose to represent the SINC function by the same data width as $wl_{P2} = 24$ to reuse components from the correlation module. Now, the data width of multiplier equals $wl_{P3} = 47$, and the data width of multiplicand is 24. Therefore, the initial value for data width in the simulation will be $wl_{P4} = (47 + 24) = 71$ bits. The initial value will be optimized by repeating steps four and five until the performance condition is violated as was detailed in Section 2.1.

The fourth and fifth steps are performed by simulation. The simulation is used to calculate BER curve for different values of SNR as shown in Figure 2-12. The graph with solid line and 'o' marker is calculated when all variables are represented in floating point. The design process starts by choosing $\mathcal{E}$= 0.2 dB to obtain system performance similar to the case of floating point arithmetic. This results in a minimum data width of ($i_{wl} = 62$, $f_{wl} = 4$) bits, which is shown in the curve with the solid line and 'x' marker. It can be observed that for the same BER value, the SNR difference between the two curves is less than 1 dB. This was expected because $\mathcal{E}$ is small.

If the system performance is relaxed to reduce resource utilization, the same process can be run again with $\mathcal{E}$ = 2 dB, resulting in data width of ($i_{wl} = 52$, $f_{wl} = 2$) bits. It should be mentioned that the advantage of using fixed point is twofold. First, the data width can be minimized at different points of the receiver chain. Second, all fixed point mathematical operators will consume less resources than floating point operators [13].

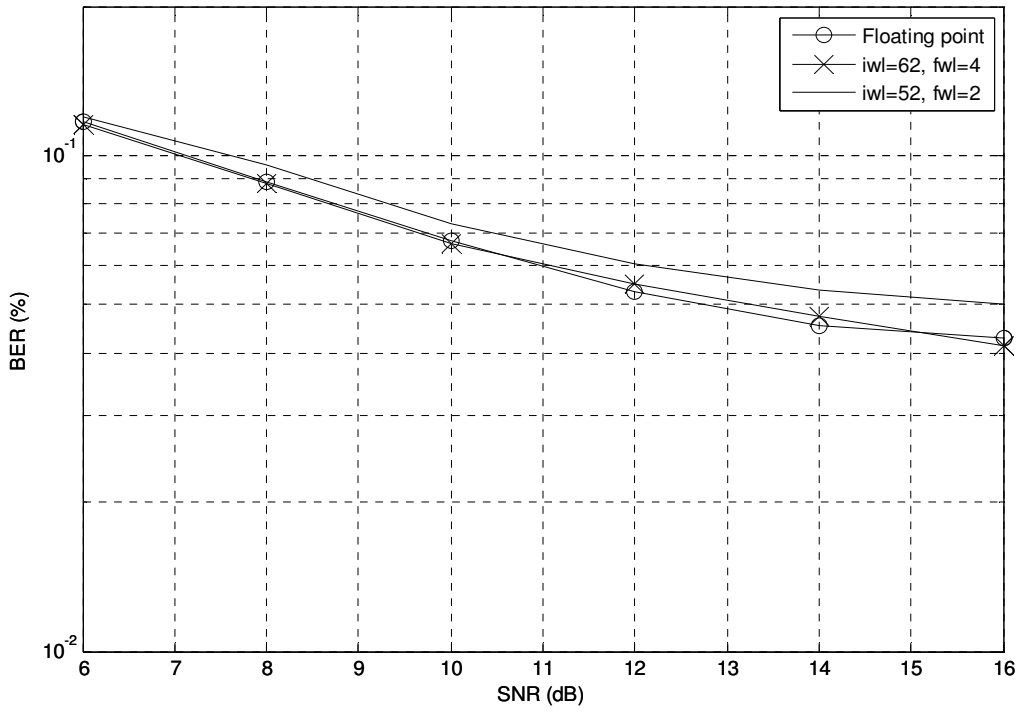


Figure 2-12: Simulation results for different values of $\mathcal{E}$

2.4 Simulation Parameters

The results obtained in Section 2.3 are validated by developing a simulation model using Mathworks tool Simulink Matlab v7:12 [29]. The model consists of transmitter, channel, and receiver.

- **The transmitter:** consists of random data source generator, GSM burst formatter, and digital up converter.
- **The channel:** is a Rayleigh fading channel, with variable fractional delay, and Additive White Gaussian Noise (AWGN). This channel model [14] can be used to simulate the system for the cases of typical urban, and rural areas as shown in Figure 2-13.

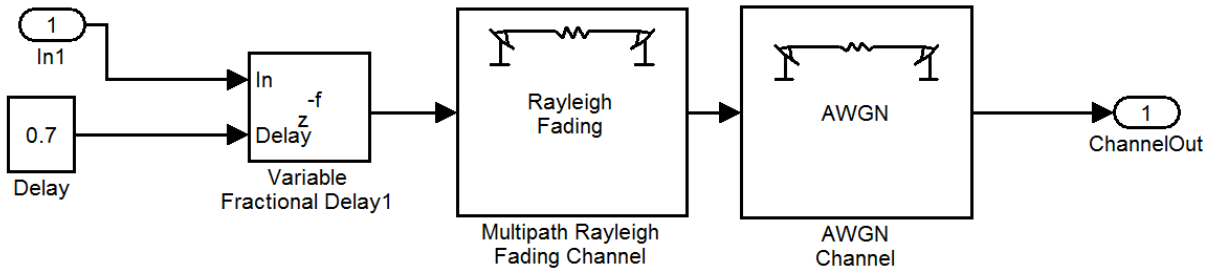


Figure 2-13: Channel model

- **The receiver:** consists of digital down conversion, timing and phase alignment, channel equalization, burst de-formatter, and bit error rate calculator.

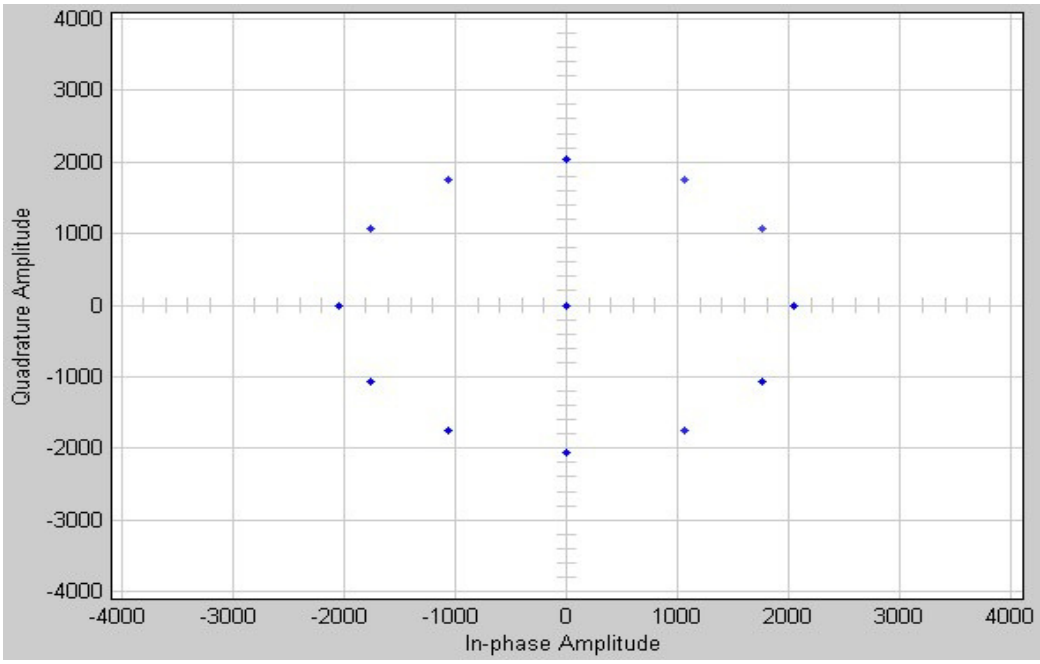
One of the advantages for using the simulation based approach, is the ability to track the system behavior at different stages of the system. For the system under test, OpenBTS, it is important to monitor the variations to the transmitted symbols during their journey to the receiver side as tracked in Figure 2-14.

In Figure 2-14(a) the constellation of the transmitted signal is shown. Each symbol has three points instead of one due to the Gaussian pulse shaping filtering of GSM. Therefore, instead of four points only of Minimum Shift Keying (MSK) , there exist 12 points of Gaussian Minimum Shift Keying(GMSK). In Figure 2-14(b), the constellation has been changed due to the channel. In Figure 2-14(c), the analyze traffic has extracted the receiver parameters only, but the receiver parameters have not been used yet. Therefore, the constellation is still not useful to detect the original transmitted symbols.

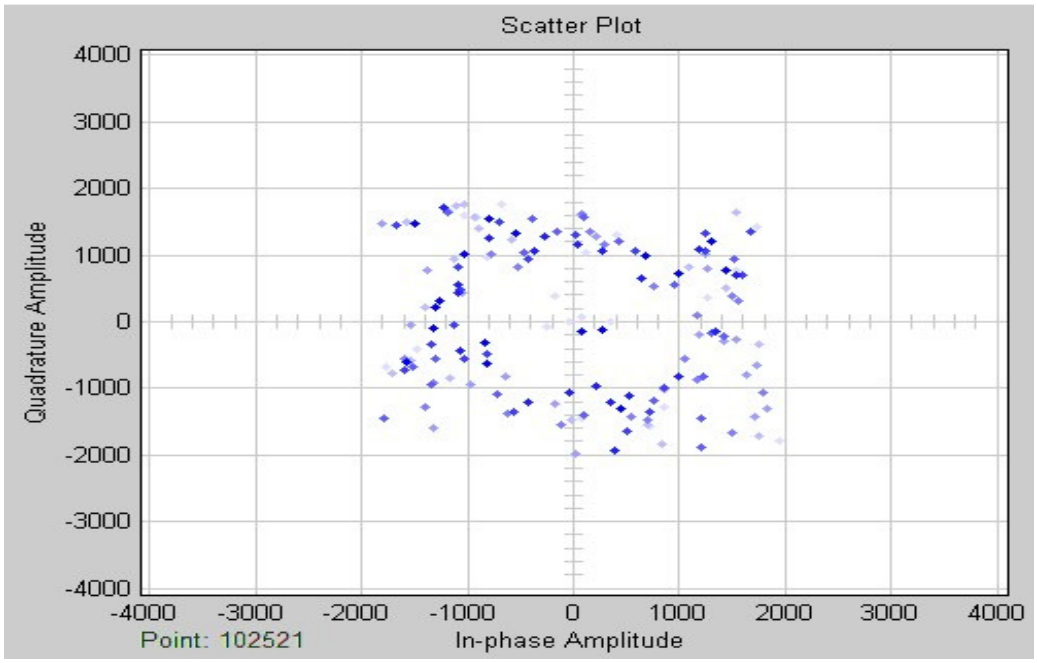
In Figure 2-14(d), a serial receiver is used to enable the detection of GMSK by a Binary Shift Keying (BPSK) demodulator. This idea is attributed to Proakis [21]. It can be observed that now, the constellation points have been grouped into almost two groups, one to the right of the y-axis and the other is to the left of the y-axis as in Figure 2-14(d).

In Figure 2-14(e), to improve the error rate, the two groups should be sent away from the y-axis. In this last graph, the constellation points are moved away from the y-axis, and located into exactly two groups. The two groups will be demodulated using a simple BPSK modulator.

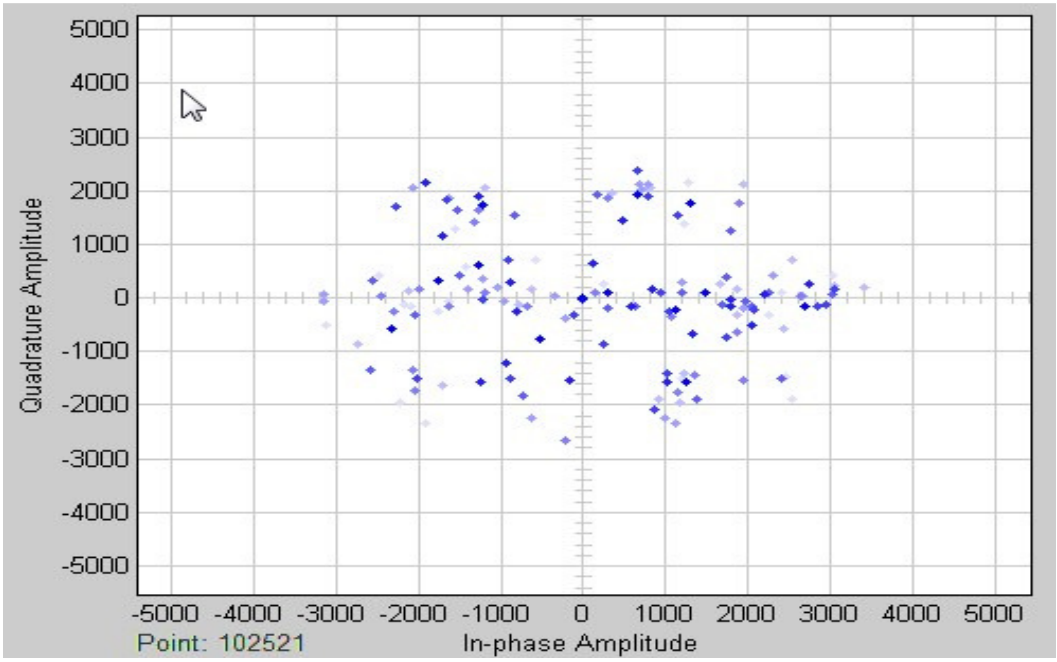
By the end of this chapter, we have applied the SDR design process in a real application, which is the OpenBTS system. Functions that have the largest processor utilization was identified, and implemented in fixed point representation. The fixed point presentation was based on optimum data widths from resource usage point of view. During our investigation in SDR systems, we found that the channel equalization is a critical system component from the implementation point of view. A separate study will be presented in the next chapter to enable smooth integration of channel equalization inside SDR system with optimum resource usage.



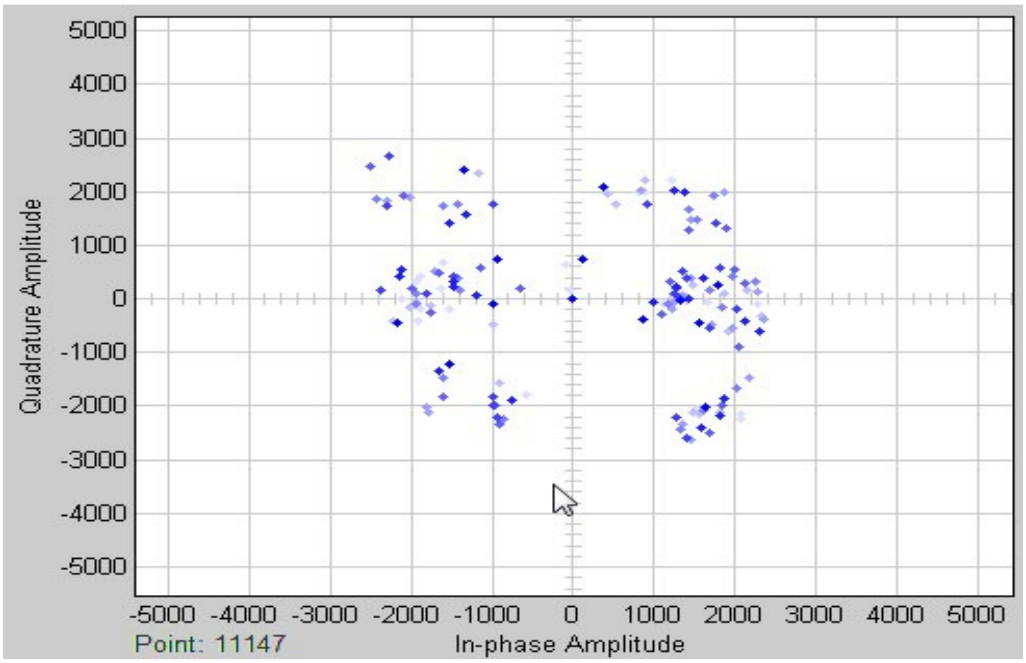
(a) Transmitted signal constellation



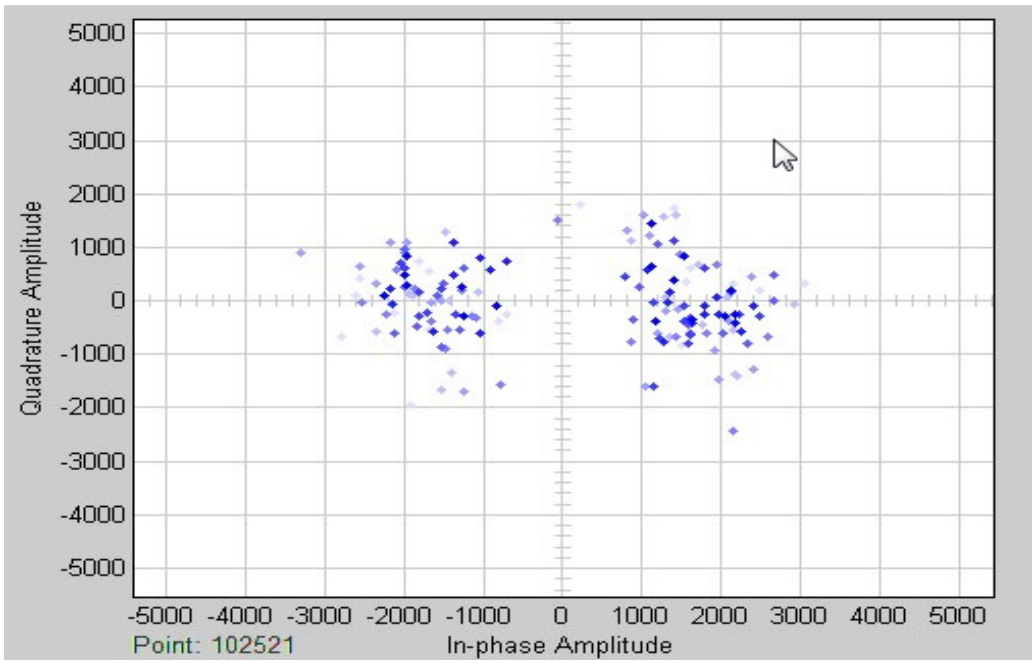
(b) Signal received at channel output



(c) After performing "Analyze Traffic Burst"



(d) After rotation



(e) At the output of channel equalization

Figure 2-14: The signal constellation at different receiver stages

Chapter 3 Reduced Complexity Channel Equalization

In the previous chapter, the functions that have the largest utilization were identified. These functions are the correlation, CORDIC, and channel equalization. The implementation of channel equalization in either software or on FPGA is not straightforward [8]. Therefore, this chapter is dedicated to make a survey of channel equalization techniques, and to highlight the challenges in their implementation. The most important challenge is to evaluate the complexity of a chosen equalization technique. The main focus of this chapter is to evaluate the computational cost or complexity for DFE that is generic enough to be used with any algorithm and structure.

The work in this chapter is twofold. First, a new metric to evaluate the complexity of different DFE is developed. This would decrease the effort to choose a suitable algorithm to be implemented. Second, DFE algorithms containing division operations consume large chip area, and it is advised to avoid them for implementation in programmable devices [11]. This work proposes a new method to implement division process. Hence, it is now permitted to use DFE containing division operations under reduced complexity constraints.

3.1 Why Channel Equalization

Inter Symbol Interference (ISI) is a common phenomenon encountered when recovering band limited channels. ISI occurs if modulation bandwidth increases beyond the channel coherence bandwidth [19]. Channel equalization is used to compensate for ISI at the receiver to decrease the bit errors. In the family of DFE equalizers the previous output decisions influence the current estimated symbol [20]. Therefore, DFE has better tracking performance than the family of linear equalizers when the channel has severe distortion and many nulls in the pass band. Both linear, and non-linear equalizers are identified by a structure, and an algorithm as shown in Figure 3-1.

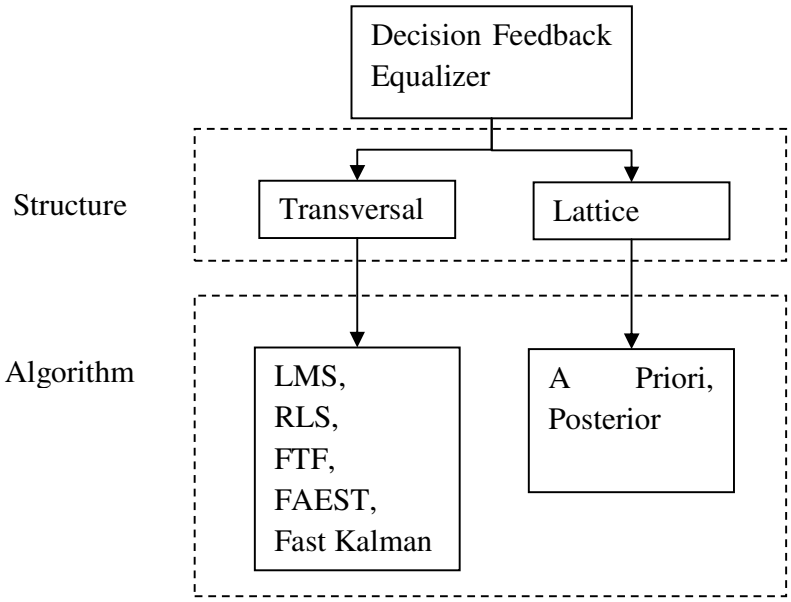


Figure 3-1: Channel equalization classification

The structure of DFE may be linear transversal or lattice. Both types will be treated in a similar approach to evaluate their complexity. The choice of an algorithm to update the equalizer weights is of great importance. Rate of convergence, misadjustment, computational complexity, and numerical properties are used to evaluate different algorithms [19]. A comprehensive survey on DFE can be found in [20].

3.2 Complexity as Number of operations

Let N be the total number of forward and backward taps of DFE and consider two channel equalization algorithms A and B. Algorithm A has O(7N) multiplications and two divisions, while algorithm B has O(8N) multiplications and one division. Note that O(7N) represents that the total number of an operation is dominated by a term that is seven times the number of taps in DFE. It is not straightforward to determine which algorithm has lower computational cost. Hence, it is required to find a metric or weighting score for each basic arithmetic operation to be able to determine the overall computational cost. The value of the weight score will represent the overall complexity of DFE. A lower weight score corresponds to reduced complexity.

In previous work where the complexity of DFE is addressed, the weight score is reported as the number of basic mathematical operations [7], [8], and [15]. The complexity of DFE is usually reported for real data and linear filtering. These results are extended to include complex data, which is the general case for communication systems [21] as shown in Table 3-1. It is important to know that the computational cost of DFE is the same as linear filtering for the same total number of taps [22]. Hence, the calculations are not only applied for linear filtering but also for the DFE problem.

Algorithm	×	+	÷
Conventional [20]	$4N^2 + 16N + 1$	$4N^2 + 12N - 1$	1
A priori lattice [11]	64N	32N	32N
FAEST [11]	28N + 6	28N + 2	5
FTF [11]	28N + 10	28N + 1	3
Fast Kalman [11]	36N + 2	32N + 1	2
ERLS-DCD-16 [8]	12N	134N	0
LMS [20]	8N + 2	8N	0

Table 3-1: Mathematical operations for different equalization techniques

Candidate algorithms will be chosen to represent the families of conventional Least Mean Square (LMS), conventional RLS, fast fixed order, lattice, and a recent study [8] of RLS based on Decent Coordinate Descent (DCD) iterations. Each candidate algorithm is generic enough to represent its family. The same procedure can be used to calculate the weight score for any other algorithm. It will be shown in Section 3.5 how the proposed metric simplifies the comparison between different families.

(LMS) algorithm has the lowest computational cost [20]. Although LMS has the slowest convergence rate, it will be used as a reference scenario for the proposed metric.

3.2.1 Fast fixed order family

Due to the desirable feature of fast fixed order family [22] with the complexity in the order of O(N), three members of this family will be evaluated; namely Fast Transversal Filter (FTF), Fast Aposterior Error Sequential Technique (FAEST) and Fast Kalman. Fast fixed order algorithms are based on RLS; namely FTF, FAEST and Fast Kalman. Their complexity is in the order of O(N) and their rate of convergence is similar to the conventional RLS, which is considered fast. Therefore, they will be suitable for systems needing short iterations to reach optimum weights of the channel to reduce transmission overhead [20].

3.2.2 Lattice RLS algorithms

The lattice RLS algorithms have a lot of desirable features such as improved numerical properties and modularity [11]. However, lattice filters have higher computational requirements and cannot be used in all applications [8]. They will be compared here with other families, only to show the effectiveness of using the proposed metric.

3.2.3 Enhanced RLS with DCD

A recent study to reduce the complexity of RLS algorithm is based on DCD iterations [8]. The RLS is expressed in terms of auxiliary normal equations with respect to increments of the filter weights. Auxiliary equations are solved using line search methods. These methods have more than

one solution for conventional RLS problem. One of these solutions is chosen, which has the least computational cost among its family namely; Exponentially Weighted RLS with 16 iterations per sample (ERLS-DCD-16). However, the ERLS-DCD offer reduced complexity at the expense of increased convergence time.

3.3 Complexity Weight Score

After describing the candidate algorithms briefly, it can be observed from Table 3-1 that it is not straightforward to determine which algorithm has the least computational cost. Moreover, it is impossible to arrange the rows in Table 3-1 in a descending order according to computational cost. Therefore, we found a need to develop a new metric to evaluate the weight score of DFE. To accomplish this goal, it is proposed to normalize all operations to a weight score as will be discussed in this section.

As stated in [23] the simplest mathematical operation is the binary addition. Therefore, all operations should be mapped to a finite number of additions. Then, the overall weight score of DFE will be a linear summation of the weight score of all used operations. In order to accomplish the normalization, we propose to analyze the low level details of each operation involved in DFE. In the following, the normalization will be developed for the multiplication operation. This normalization enables one to compare fairly between different families of algorithms. The three major categories for multiplication [23] will be discussed:

- Shift and add multiplication
- Wallace tree multiplications
- Iterative array multiplication

3.3.1 Shift and add Multiplication

As described in Table 3-2 the binary multiplication may be performed by adding the multiplier to the multiplicand and storing to a temporary result. Then the multiplier is shifted one bit to the left and added to the previous temporary result [23]. To illustrate the multiplication of X and Y Let M_l and M_c be the data width for multiplier and multiplicand respectively. The first partial product is the binary multiplication of first bit in the multiplier y_0 and each bit in the multiplicand x_i where the index " *i* " is in the range 0 to Mc-1. Recalling that binary multiplication is the logical "AND" operation, the first bit of the first partial product equals x_0y_0, and the second bit of the first partial product equals x_1y_0, up to $x_{Mc-1}y_0$. The same procedure is repeated for each bit in the multiplier until all of the partial products are generated.

Multiplicand Multiplier	.....x_3 x_2 x_1 x_0 .....y_3 y_2 y_1 y_0
Partial Products	...x_3y_0 x_2y_0 x_1y_0 x_0y_0 ...x_3y_1 x_2y_1 x_1y_1 x_0y_1 ...x_3y_2 x_2y_2 x_1y_2 x_0y_2 .....
Final Product	$Z_{Mc+Ml-1}$ ..... Z_2 Z_1 Z_0

Table 3-2: Multiplication using add and shift method

This method is the simplest from complexity point of view because it can be implemented using only one adder and one shifter. However, from processing time point of view it is considered the slowest to store the final result.

3.3.2 Wallace Tree Multiplication

Another multiplication method is the Wallace Tree [23]. It is based on parallel generation of the required number of partial products. Afterwards this number of partial products is reduced. It is the fastest multiplier scheme at the expense of increased computational cost.

3.3.3 Iterative array Multiplication

A method that is considered a good compromise between processing time and computational cost is the iterative array of cells [23]. Hence, the iterative array multiplication will be used in the rest of this chapter. However, the new proposed metric can be calculated for all multiplication methods as will be discussed in the following paragraph. In general, the iterative array method is used for short data lengths, which is the case for communication receiver [24], because its delay increases with operand length.

Multiplication consists of a finite number of cells or building blocks. To perform one multiplication, a finite number of building blocks are needed. let G be the number of building blocks needed to perform one multiplication. In order to calculate the number of required building blocks G, the following relation is used [23]

$$G = \left\lceil \frac{M_l \times M_c}{m_l \times m_c} \right\rceil, \tag{3.1}$$

Where and m_l and m_c are the data width of the building block used. For example, using $M_l = M_c = 12$ and $m_l = m_c = 4$ will result in G = 9. Therefore, nine building blocks will construct one multiplier, and each building block has two operands with data widths m_c, m_l respectively.

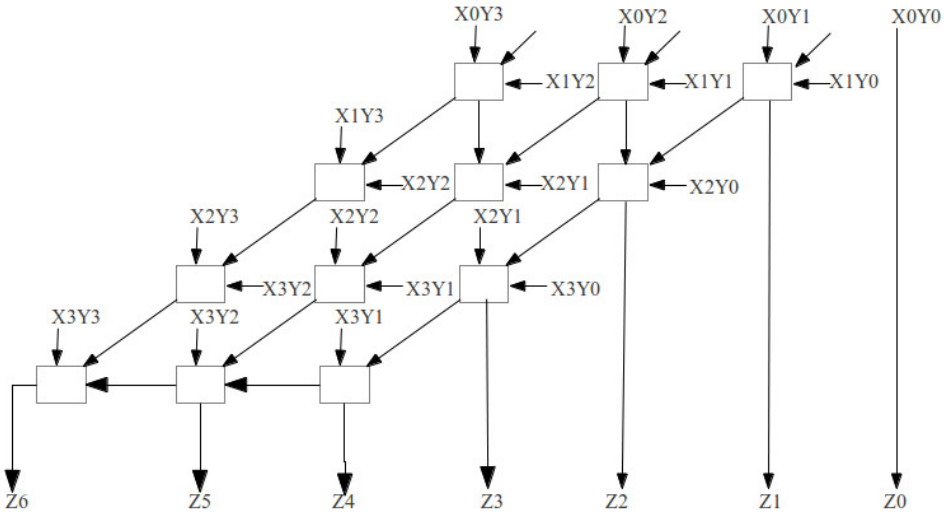


Figure 3-2: Structure of 4 x 4 multiplier using full adder cells

To complete the normalization, each building block is composed of finite number of 1-bit full adder cell as shown in Figure 3-2. Hence, one multiplier building block is composed of [(m_l - 1) (m_c) + 1] 1-bit full adder blocks. Therefore, the weight cost of multiplication M_{score} can be rewritten as

$$Mul_{score} = \big((m_l - 1) \times m_c + 1\big) \times G. \quad (3.2)$$

Equation (3.2) explains how to calculate the weight score of multiplication operation in terms of 1-bit full adder cells. It can be observed that there exists an inverse proportional relation between the data width of the building block and the number of building blocks G. As ml x mc increases, the number of required building blocks G decreases as shown in Figure 3-3.

From complexity point of view, the weight score of multiplier is independent of the number of used building blocks G. Intuitively it does not matter whether to use one large building block or many small building blocks. In both situations the total number of 1-bit full adder cells are the same as shown in Figure 3-3. However, from processing time point of view it is favored to use smaller building blocks. This is due to the fact that the processing time to obtain a result is proportional to the width of the building block [25].

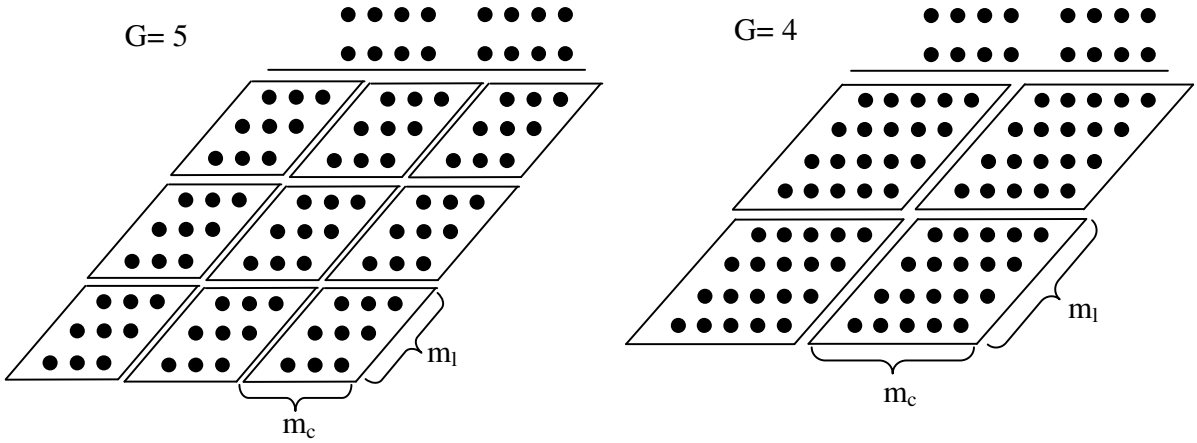


Figure 3-3: Compromise between number of blocks G and operands width ml, mc

It can be observed, from Figure 3-3, that using 3*3 building block generates an extra partial product because the data width is not integer multiple of the building block data width. This extra partial product will be zeros and may be skipped using a multiplexer [25]. In general, for any Ml*Mc data width, and any ml*mc building block width the extra partial products will not affect the final weight score. This is because the weight score is used for comparison based on the number of used 1-bit full adder cells and not on the number of building blocks.

3.4 Division Using Series Expansion

In the literature of adaptive filtering such as [7] [20], the computational cost of DFE is reported as the number of four basic mathematical operations in addition to the square root one. To reduce complexity it is advised neither to use divisions nor square root, and as few multiplications as possible [8]. In this work division will be permitted by reducing its weight score.

Division algorithms can be divided into two categories; the first category is based on iteration of subtraction and the second is based on iteration of multiplication [9]. The first category is performed as the normal pencil and paper division. For each iteration there is a remainder R, divisor D, and quotient Q. The $i^{\underline{th}}$ bit of quotient q_i can be calculated using the following equation

$$R(i) = R(i+1) - q_i \times D \times 10^i. \qquad (3.3)$$

For example, the first iteration of division of 4000 over 3 will be $4000 - 1 \times 3 \times 1000$, where $R(i+1) = 4000$, $q_3 = 1$, $D = 3$, and $10^i = 10^3$. Each iteration q_i is chosen to be 0 or 1 according to the negative or positive value of $R(i)$ respectively. This method is considered slow because the delay is proportional to the ratio between the divisor and the dividend. Therefore, we will consider another category of division algorithms, which has less processing time. The second category of division algorithms is based on the use of Maclaurin series expansion [23]. The division a/b will be obtained as the multiplication of a and 1/b . Note that, according to the floating point standard [26],

numbers are represented in either 32 or 64 bits with the format 1.x*radix^{exponent} where x is a fraction. Therefore, one can use b = 1 + x and ignore higher orders of x, depending on the required accuracy, in the familiar series expansion

$$\frac{1}{b} = \frac{1}{1+x} = \underbrace{(1-x)(1+x^2)(1+x^4)(1+x^8)}_{Memory\ location\ address}. \tag{3.4}$$

All possible values of the reciprocal are stored in a memory element with its length proportional to the data width. This would replace the need to true division with a simple memory allocation. The memory score ($Memory_{score}$) is the size of the table to store the values of 1/b. To estimate the table size we recall that in typical communication systems [24] the data widths are in the range 8, 10, 12 or 14 bits. A memory block of size 2^c is needed to store all values of a digital word with data width c bits. Hence, the memory size is approximately 0.25, 1, 4 or 16 kbits.

One divider consists of a finite number of adders and multipliers to form the memory address. This number varies according to the desired data accuracy. For 8-bit data accuracy, six multipliers and four adders are needed to implement one divider and the calculation of the memory address is shown in Figure 3-4. To locate the memory element address we need to calculate the powers x^2; x^4; and x^8 depending on the desired accuracy. For 8-bit accuracy, the weight score of one division D_{score} can be calculated according to the following relation

$$D_{score} = 6 * Mul_{score} + 4 * Add_{score} + Memory_{score}. \tag{3.5}$$

To increase the accuracy of the division operation, higher powers of x should be considered. For increased accuracy, such as 12-bit, x^{12} can be calculated by multiplying x^4 by x^8, which have been calculated earlier as shown in Figure 3-4. Consequently, one extra multiplier and two adders are needed. The weight score can be modified accordingly such that

$$D_{score} = 7 * Mul_{score} + 9 * Add_{score} + Memory_{score}. \tag{3.6}$$

According to equations (3.5) and (3.6) the division is normalized into a finite number of 1-bit full adder cells. In the following section results of weight score for different algorithms are presented.

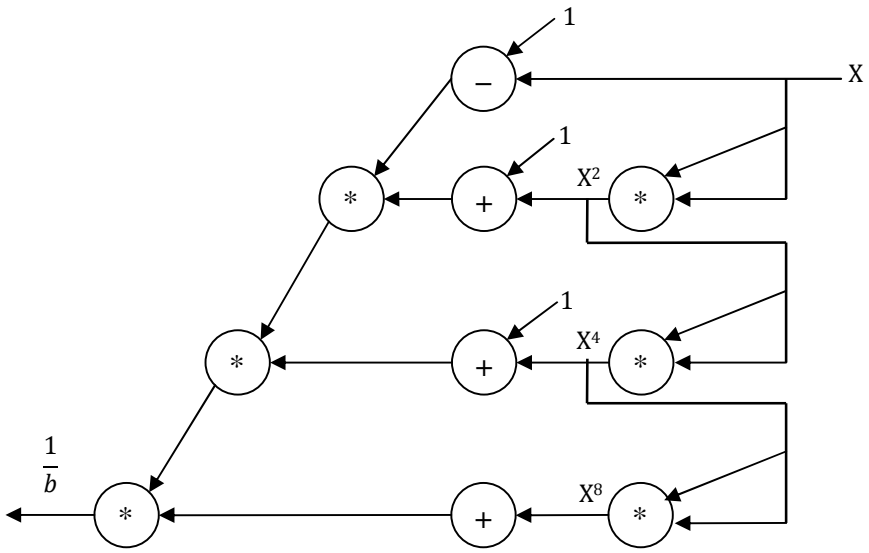


Figure 3-4: Division using series expansion for 8-bit accuracy

3.5 Weight score simulation results

In this section, results obtained by computer simulation are presented. All equations are plotted using Mathworks tool Simulink Matlab v7.12 [29]. The complexity weight score of different algorithms is plotted against filter order N. First, the weight score of each operation is calculated according to equations (3.2) and (3.5). For each algorithm the total computational cost is calculated according to Table 3-1. Then the number of multiplications, and divisions in Table 3-1 is multiplied by Mul_{score} and D_{score} respectively. In this manner the computational cost is compared fairly. The total weight score of each algorithm is the linear summation between the weight score of additions, multiplications and divisions required for that algorithm. This procedure is repeated for different values of equalizer order N as shown in Figure 3-5.

It can be observed from Figure 3-5 that the conventional RLS has much higher weight score compared to all other graphs, which are located at the bottom of the y-axis. The weight score of conventional RLS is almost 5,500 at N = 10. However, its complexity increases exponentially afterwards until it reaches 20,000 at N = 20. It is observed that the LMS has the least weight score over all values of N. This result was expected as was mentioned in Section 3.2. To be able to zoom into different algorithms, the RLS graph will be removed as shown in Figure 3-6.

The ERLS-DCD-16 has slightly higher complexity than LMS. This result conforms with the results in [5], which states that LMS complexity is proportional to 2N, while ERLS-DCD-16 complexity is proportional to 3N. Both FTF and FAEST have almost the same weight score over N. It is important to observe that fast Kalman has higher weight score than FASET. This result is not obvious without using the weight score metric because FAEST has 5 divisions while Fast Kalman has 2 divisions only.

By the end of this chapter, we will consider two FPGA experiments to verify the simulation results experimentally. The two experiments are evidences of the real resource usage of the FPGA. The first experiment is related to the new design process of Chapter 2, while the other is related to channel equalization.

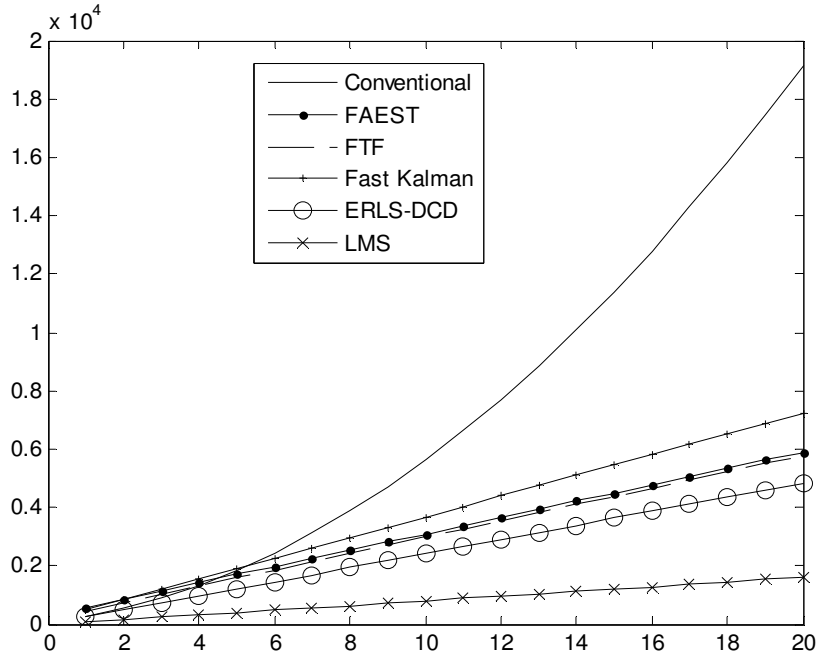


Figure 3-5: Complexity of DFE versus the total number of taps (with RLS)

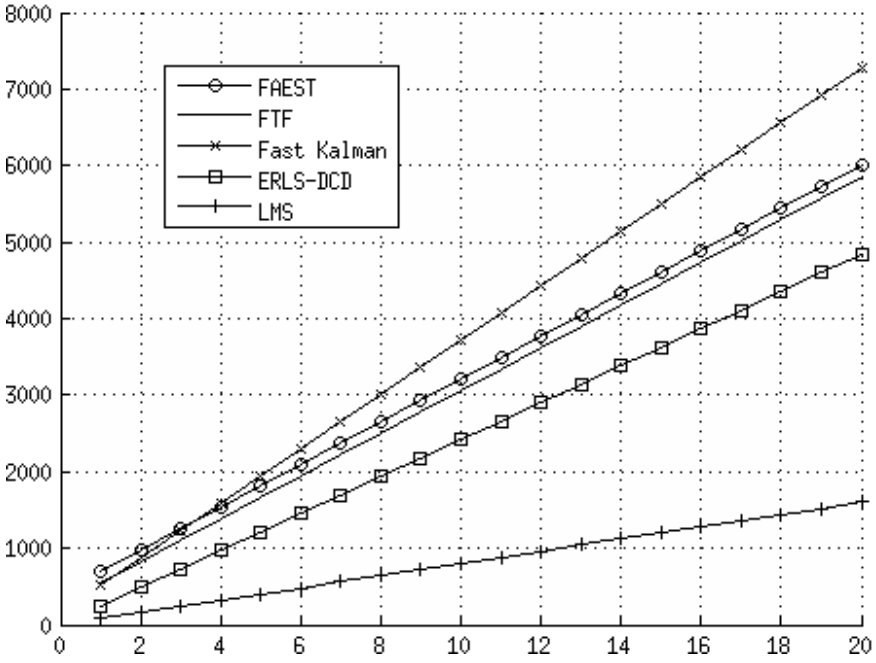


Figure 3-6: Complexity of DFE versus the total number of taps (without RLS)

Chapter 4 Experimental Results and Guidelines

The proposed methods in Chapter 2, and Chapter 3 are validated by implementing their designs on FPGA. The implementation results is collected and summarized in tables, to show the effectiveness of the proposed methods. The implemented designs are simulated twice using functional, and timing simulations. A functional simulation is performed on the input text files only. The Integrated Development Environment (IDE) extracts the architecture and creates files responsible to simulate the circuits, without mapping them into physical gates. On the contrary, a timing simulation is performed on the design after it has been translated, placed, and routed into physical gates.

Before the two Hardware Description Language (HDL) experiments are presented, implementation guidelines are highlighted. These guidelines were discovered during the design time, and can be used as general guidelines when one is programming for resource minimization.

4.1 Implementation Guidelines

4.1.1 Complex multiplication

In general, the baseband signal is represented in complex number notation. Therefore, the correlation block has to be able to accept complex inputs and outputs. To achieve this, complex multipliers should be implemented. This can be implemented using four real input multipliers as shown in Equation (4.1).

$$\begin{aligned} re(c) &= re(a)re(b) - im(a)im(b) \\ im(c) &= re(a)im(b) + im(a)re(b) \end{aligned} \quad (4.1)$$

To improve the resource usage one complex multiplier will be implemented using only three multipliers as in Equation (4.2). This implementation will increase the number of used adders. However, the resource usage of an adder is very small compared to one real multiplier.

$$\begin{aligned} re(c) &= \big(re(a) - im(a)\big)im(b) + \big(re(b) - im(b)\big)re(a) \\ im(c) &= \big(re(a) - im(a)\big)im(b) + \big(re(b) + im(b)\big)im(a) \end{aligned} \quad (4.2)$$

4.1.2 Forming wide operands multipliers

To implement multipliers with MxM-bit operands, the built-in NxN-bit multiplier of the FPGA is used. For instance, a 35x35 multiplier is composed of four 18x18-bit multipliers, as shown in Figure 4-1. It should be noted that the optimization can be written manually, or automatically be setting the proper synthesis options.

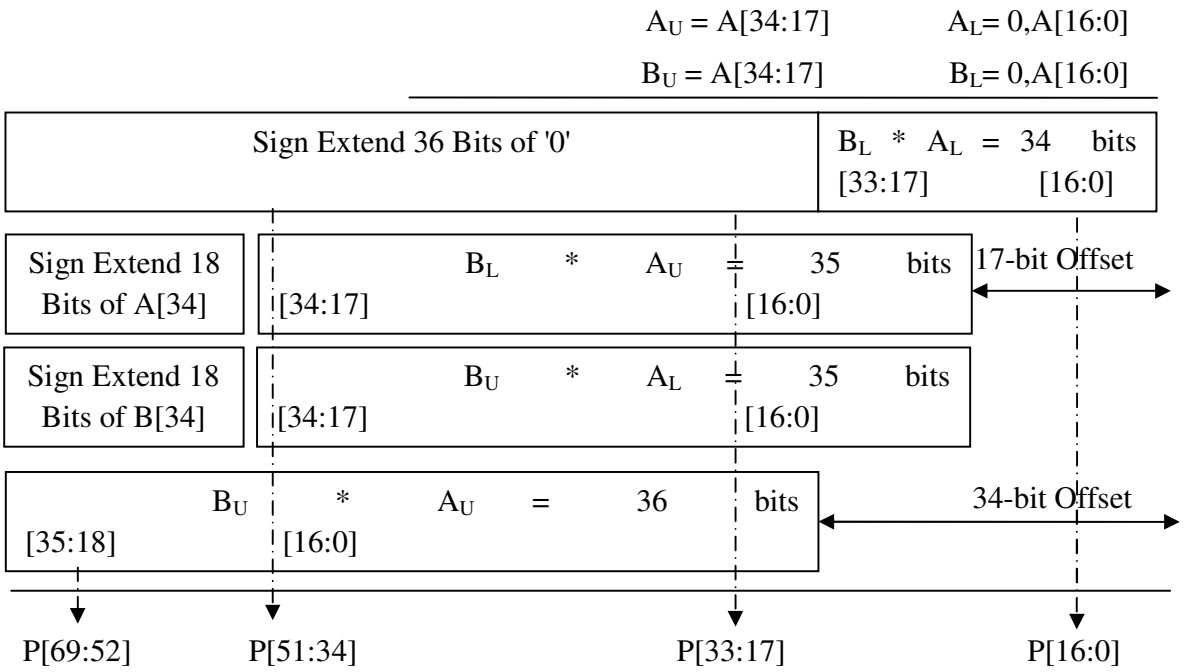


Figure 4-1: 35x35-bit Multiplication from 18x18-bit Multipliers

4.1.3 FIR Filters

The Systolic FIR is implemented using internal blocks of Spartan-3A DSP FPGA named DSP48A blocks. Sixteen blocks are needed for the current software version of OpenBTS. This will be used for correlation, and convolution operations.

It is worth mentioning that the 16 DSP48A blocks are programmed to have three groups as shown in Figure 4-2. The first group is the bottom DSP48A block that accepts two external inputs, and its output is shifted internally. The second group consists of the next 14 DSP48A blocks which accepts only one external input, one internal input from the previous block, and one internal output shifted to the next block. The third group consists of the last DSP48A block which is similar to the previous group but with one external output. This is the actual final output of the correlation process.

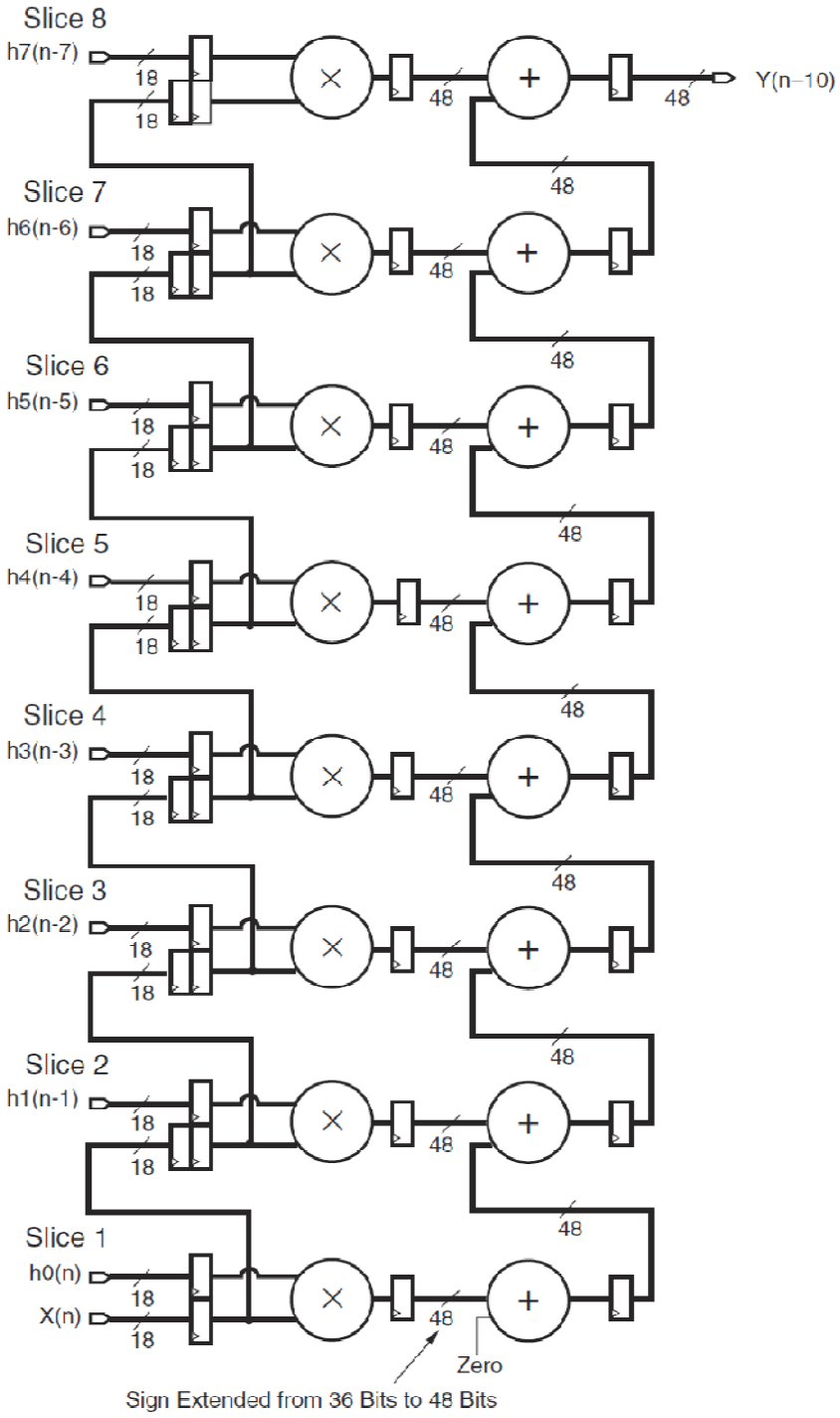


Figure 4-2: Systolic FIR with Adder Cascade [2]

4.2 HDL implementation of Analyze Traffic Burst

4.2.1 Experiment Setup

In this experiment, the steps to move functions from the CPU to the FPGA are detailed. The experiment is performed on USRP E110 device.

Step 1: Get the source files of the Universal Hardware Driver (UHD)

The UHD contains all software drivers that control the USRP device. The UHD enables to modify the FPGA image, which comes as a pre-built binary in the OpenBTS. To download the latest UHD release, use the "git" shell command

```
git clone git://github.com/EttusResearch/uhd.git
```

The current version is of UHD is Mirror-release_003_007_001. The downloaded source files will have the structure as shown in Figure 4-3. Following the arrows in Figure 4-3 starting from "fpga" folder, any part of the FPGA code can be modified. In this experiment, we will modify the top module of E110, which is located at the "top" folder. Strictly speaking, this file will be modified "~/fpga/usrp2/top/E1x0/u1e_core.v", where "~" denotes the path where the UHD is installed.

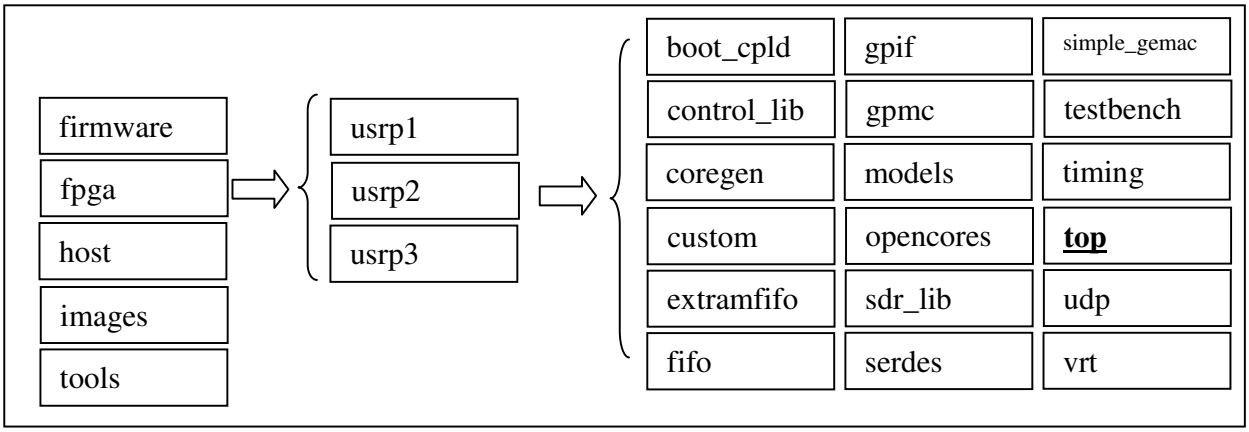


Figure 4-3: The folder structure of UHD code

Step 2: Locate the functions with the highest CPU utilization

Now we need to install OpenBTS project either on Windows or Linux Operating system. The source code can be downloaded from the following web link "http://openbts.org/get-the-code/". The functions with the highest execution time are shown in Table 4-1. The two functions "equalizeBurst" and "convolve" will be moved according to software profiling. These two functions are located in the file "~/ openbts/Transceiver52M/sigProcLib.cpp".

Function Name	Execution time (us)	Comment
USRPifyVector	11.66	It cannot be moved because it is needed for interface.
equalizeBurst	7.9	It can be implemented
convolve	5.28	It consists the major part of "Analyze Traffic Burst"
unUSRPifyVector	1.81	It cannot be moved because it is needed for interface.

Table 4-1: Function execution times in OpenBTS project

Step 3: Determine the FPGA module to be modified

Before we determine where to place the moved function into the Verilog code, we need to understand the general architecture of the FPGA code. Therefore, a low level block diagram was constructed highlighting the main modules, and the connections between them as shown in Figure 4-4. Each block indicates the file name where a function is defined, such as "rx_frontend.v", and the instance name which uses that file such as "RxFrontEnd".

From the left of the diagram the signal is received from ADC and transmitted to the processor at the right of the diagram. Data from ADC is concatenated with 4 zeros, then decimated through the module ddc_chain0. Afterwards, the data is transmitted via VRT protocol [16], which is handled by the block vita_rx_chain. Finally, the multiplexer is used to combine the two DDC chain data (second chain is not drawn for simplicity) and forward them to the wishbone master (GPMC) towards the processor.

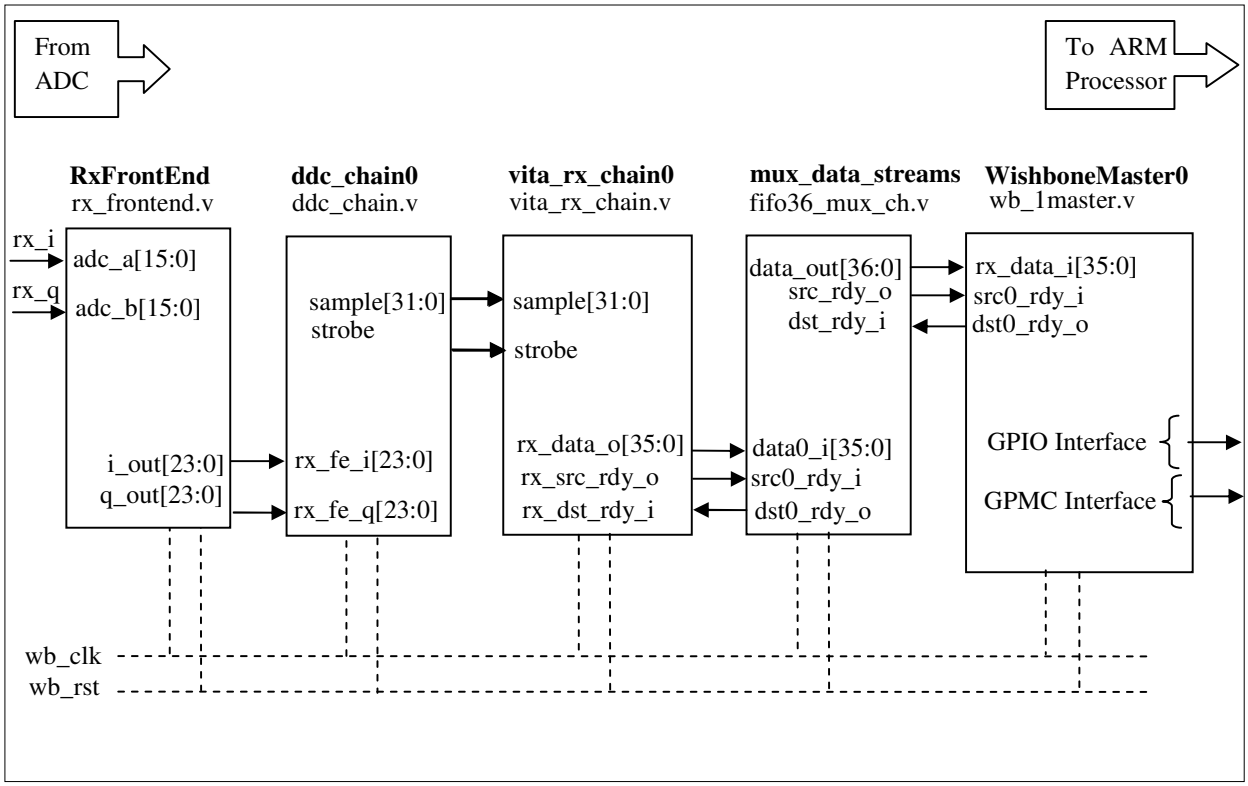


Figure 4-4: The code hiearchyinside the FPGA

Step 4: Check the FPGA project Hierarchy

After the code hierarchy has been explored, the new module can be added into the proper location. The code snippet in Table 4-2 shows the added module "AnalyzeTrafficBurst". By observing the clear mapping between the code hierarchy and the source code, the location to insert "AnalyzeTrafficBurst" is determined. It is inserted before the last module in the receiver chain; namely the VRT module,

```
////////////////////////////////////////////////////////////////////
// DSP RX 0

wire [31:0]  sample_rx0;
wire         strobe_rx0, clear_rx0;
wire [35:0] vita_rx_data0;
wire         vita_rx_src_rdy0, vita_rx_dst_rdy0;

ddc_chain #(.BASE(SR_RX_DSP0), .DSPNO(0)) ddc_chain0

    (.clk(wb_clk), .rst(wb_rst), .clr(clear_rx0),
     .set_stb(set_stb),.set_addr(set_addr),.set_data(set_data),
     .set_stb_user(set_stb_user), .set_addr_user(set_addr_user), .set_data_user(set_data_user)
     .rx_fe_i(rx_fe_i),.rx_fe_q(rx_fe_q),
     .sample(sample_rx0), .run(run_rx0), .strobe(strobe_rx0),
    .debug() );

AnalyzeTrafficBurst #( .WordWidth(16),.SamplesPerSymbol(1)) A0

          (.clk(wb_clk),.rst(wb_rst),.sampleIn(sample_rx0),.peakIndex(),.toa()     );

vita_rx_chain     #(.BASE(SR_RX_CTRL0),     .UNIT(0),     .FIFOSIZE(10),     .PROT_ENG_FLAGS(0),
.DSP_NUMBER(0)) vita_rx_chain0

    (.clk(wb_clk),.reset(wb_rst),
     .set_stb(set_stb),.set_addr(set_addr),.set_data(set_data),
     .set_stb_user(set_stb_user), .set_addr_user(set_addr_user), .set_data_user(set_data_user),
     .vita_time(vita_time), .overrun(rx_overrun_dsp0),
     .sample(sample_rx0), .run(run_rx0), .strobe(strobe_rx0), .clear_o(clear_rx0),
      .rx_data_o(vita_rx_data0), .rx_dst_rdy_i(vita_rx_dst_rdy0), .rx_src_rdy_o(vita_rx_src_rdy0),
      .debug() );
```

Table 4-2: A code snippet illustrating the location of the added module

After "AnalyzeTrafficBurst" is added to the OpenBTS project, the Xilinx tools indicate the instance among other modules. The instance is named A0 as show in Figure 4-5. Finally, generate the FPGA image and report the resource usage in the next section.

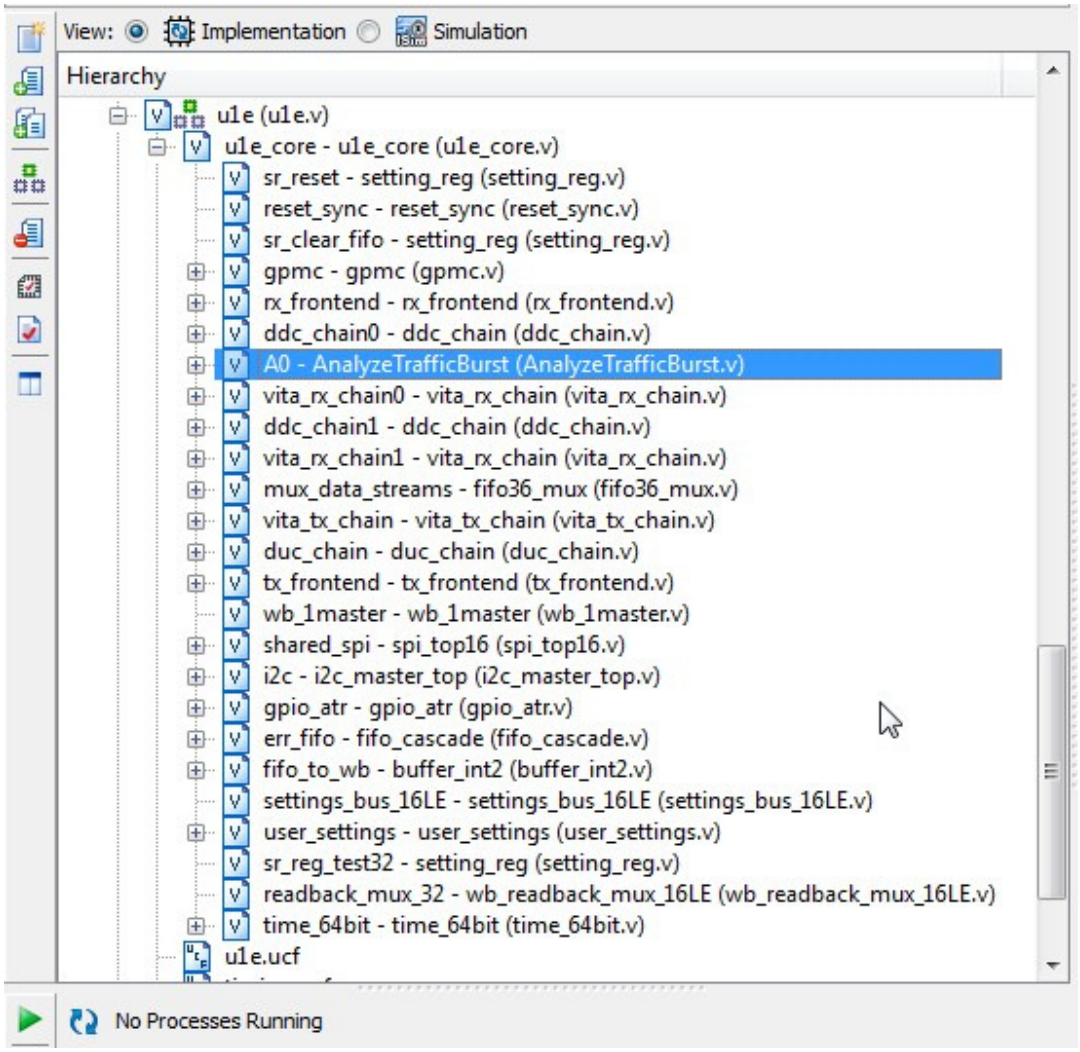


Figure 4-5: The FPGA code hierachy from Xilinx tool

4.2.2 Analyze Traffic Burst FPGA Resource Report

The results of implementation are generated by Xilinx's design tools v14.1; namely ISE, and ISIM. The USRP is equipped with Xilinx's FPGA named Spartan 3A-DSP 3400. The resource utilization due to the mapped design is reported in Table 4-3.

When the design is implemented using floating point arithmetic, the synthesis operation fails to map the design into the FPGA. This is expected because a floating point multiplier can consume one, or more FPGA units [13][31]. Therefore, we apply the two guidelines that were mentioned in Section 4.1.1, and Section 4.1.2. The first guideline makes use of the internal DSP units in the FPGA as shown in Figure 4-2, to implement the correlation. This results in saving the general logic cells for the rest of the design. The second guideline eliminates the need to implement a wide operand multiplier, in the order of O(35) bits. Alternatively, wide operand multipliers were built based on internal FPGA 18x18-bit multipliers.

After applying the design process, the design can fit the function "Analyze Traffic Burst" into the FPGA with resource usage less than 50%. The DSP48 blocks have relatively high utilization of 45%, because they are required to implement multiplication operation without using the logic slices of FPGA [32]. Note that, the multipliers have 36 bits operands, and hence they can be used for both cases of (iwl = 62, fwl = 4) and (iwl = 52, fwl = 2), with the same resource utilization. Finally, the experimental results verify the validity of the proposed design process.

Resources	$i_{wl} = 52$			$i_{wl} = 62$		
	Used	Available	%	Used	Available	%
Slices	7538	23872	31%	9548	23872	40%
Flip Flops	9273	47744	19%	15278	47744	32%
4-LUTs	14074	47744	29%	18620	47744	39%
BRAMs	7	126	5%	7	126	5%
GCLKs	2	24	8%	2	24	8%
DSP48s	57	126	45%	57	126	45%

Table 4-3: Logic utilization for the function "Analyze Traffic Burst"

4.3 HDL implementation of RLS Equalizer

4.3.1 Experiment Setup

The test vectors for the channel equalization are extracted from the MATLAB system model that was created in Chapter 2. Two input vectors are needed; namely received signal vector, and training signal vector. One control input is needed to choose whether the equalizer works in training mode or equalization mode. The simulation is run and the output vectors are stored in text files. Finally, these vectors are compared to the output vectors from the MATLAB simulation.

4.3.2 RLS FPGA Resource Report

In order to verify the usage of the proposed division method experimentally, the resource utilization for the Xilinx FPGA "Spartan3A-DSP1800" [27] is shown in Table III. Results are obtained using Xilinx development suite ISE12:1.

Using the automatic synthesis procedure, the synthesizer fails to design the division process and ends with error messages. This result was expected because the implementation of division operation into FPGA is problematic [28].

To solve this synthesis failure, divisions are implemented as proposed using memory elements, which is called Block RAM (BRAM). The synthesis process succeeded with chip utilization less than 5%. In addition the guideline of Section 4.1.3 is applied, to make use of the internal DSP units in the FPGA as shown in Figure 4-2. This will save the general logic cells for the rest of the design, and will protect general logic cell from being digested by the multiplications and shift registers in the tapped delay line.

Resources	N = 8	N = 12	N = 16
Slice	127(1%)	177(1%)	206(1%)
D-FF	136(1%)	186(1%)	226(1%)
LUT-4	206(1%)	313(1%)	403(1%)
BRAM	1(1%)	2(2%)	4(4%)
DSP48	2(2%)	3(3%)	3(3%)

Table 4-4: FPGA resources utilization for channel equalizer

Chapter 5 Conclusions

SDR has a desirable nature of adding new features by reconfiguration. However, this will increase the resource usage and may affect system performance. One solution was to implement algorithms in fixed point number representation. In this work, a new design process was proposed to link between system performance and computational accuracy using simulation. To validate the proposed process a case study of the OpenBTS project was considered. It was not possible to implement the case study into FPGA without applying the proposed process, while maintaining system performance. Moreover, the system performance was relaxed to obtain more savings in resource usage. Finally, the results were verified experimentally using FPGA implementation. It was shown that the utilization of FPGA did not exceed 50 % of the available resources.

During our research, it was discovered that some system modules should be paid more attention such as channel equalization. Choosing a channel equalization algorithm affects the total system resources considerably. Therefore, a novel metric was developed to compare fairly between channel equalization algorithms. In addition, a new method was presented to implement the channel equalization algorithms that contain division operation. It was usually advised to include division operation into FPGA designs. By using the proposed method, it was permitted to implement division on FPGA.

The results here can be used by both academia and industry. For industry, the proposed model can be deployed for different generations of mobile networks form 2G, 3G, and 4G. For academia, a broad range of research areas can benefit from this work, such as design automation for SDR, and open source networking.

This work was based on creating a simulation model for the system under improvement. Therefore, we can extend this work by adding a library for common system modules such as source coders and burst formatters. Another trend is to apply the same approach for recent mobile networks such as the fourth generation mobile networking, and wireless fidelity. In addition, instead of applying our work into base station only for OpenBTS, the concept of network in a box can be achieved. Network in a box aims at collecting all mobile network elements into one device, that can be programmed and operated with minimal installation effort.

References

[1] J. Bard, V. Kovarik "Software Defined Radio The Software Communications Architecture", First Edition, Wiley 2007, ISBN 978-0-470-86518-7

[2] G. Feng, C. J. Chiang, Y. M. Gottlieb, and R. Chadha "GNU Radio-based digital communications: Computational analysis of a GMSK transceiver." In Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE, pp. 1-6. IEEE, 2011.

[3] H. Nguyen, D. Menard, and O. Sentieys, "Design of Optimized Fixed point WCDMA Receiver", European Signal Processing Conference, pp.993-997, 2009

[4] D. Novo, M. Li, B. Bougard, L. Perre, and F. Catthoor, "Finite Precision Processing in Wireless Applications", Design, Automation and Test in Europe (DATE), 978-3-9810801-5-5, 2009

[5] D. Novo, B. Bougard, A. Lambrechts, L. Van der Perre, and F. Catthoor, "Scenario Based Fixed-point Data Format Refinement to Enable Energy scalable Software Defined Radios", Design, Automation and Test in Europe (DATE), 978-3-9810801-3-1, 2008

[6] J. Cioffi and T. Kailath, "Fast, Recursive Least-Squares Transversal Filters for Adaptive Filtering", IEEE Transactions on Acoustics, Speech, and Signal processing, vol. 32, no. 2, Apr. 1984, pp. 304-337

[7] N. Al-Dhahir, J. Cioffi, "Fast computation for Channel-Estimate based Equalizers in Packet Data Transmission", IEEE Transactions on Signal Processing, vol. 43, no. 11, Nov. 1995, pp. 2462-2473

[8] Y. Zakharov, G. White, and J. Liu, "Low-Complexity RLS Algorithms Using Dichotomous Coordinate Descent Iterations", IEEE Transactions on Signal Processing, vol. 56, no. 7, July. 2008, pp. 3150-3161

[9] F. Jondral, "Software Defined Radio Basics and Evolution to Cognitive Radio", EURASIP Journal on Wireless Communications and Networking, vol. 3, pp.275-283, 2005

[10] D. Menard, R. Serizel, R. Rocher, and O. Sentieys, "Accuracy Constraint Determination in Fixed-Point System Design", EURASIP Journal on Embedded Systems Volume 2008, Article ID 242584

[11] H. Sayed, Fundamentals of Adaptive Filtering, 2nd ed., New Jersey: Wiley-IEEE Press, 2003, pp. 600-620

[12] E. B. Hogenauer, "An Economical Class of Digital Filters for Decimation and Interpolation", IEEE Transactions on Acoustics, Speech and Signal Processing(ASSP), vol. 29, no. 2, April 1981, pp. 155-162

[13] S. Yassin, and H. Tawfik,"Reduced Complexity Decision Feedback Channel Equalizer using Series Expansion Division", Advanced International Conference on Telecommunications, June 2013, pp. 219-223.

[14] D. Burgess and S. Harvind, "The OpenBTS Project.", http://openbts.org,Jan. 2014.

[15] L. Kashka , "Embedded software radio design with the E100 software radio peripheral", B.S. Thesis, Kansas State University, 2002

[16] P. Balister, “High Performance Interface between the OMAP3 and an FPGA”, Open SDR April, 2011 <www.opensdr.com>

[17] M. Ettus, “Ettus Research Products and Roadmap”, ETTUS research, September 2011, <www.ettus.com>

[18] J. Hennesy, and D. Patterson, ”Computer Architecture: A Quantitative Approach”, 2nd ed., pp. 18-29, 1996

[19] T. Rappaport, Wireless Communication: Priciples and Practices, 2nd ed., New Jersey: Prentice Hall, 2001, pp. 308-318

[20] J. Proakis, “Adaptive Equalization for TDMA Mobile Radio”, IEEE Transactions on Vehicular Technology, vol. 40, no. 2, May. 1991, pp. 333-341

[21] B. Bjerke, J. Proakis, K. Martin Lee, and Z. Zvonar, “A Comparison of GSM Receivers for Fading Multipath Channels with Adjacent- and Co-Channel Interference”, IEEE Journal on selected areas in communications, vol. 18, Nov. 2000, pp. 2211-2219

[22] Y. Yang, X. Gao, Z. Gao, and X. Wang, “An Classification-based Adaptive Decision Feedback Equalizer for Rayleigh Multipath Channel”, Journal of Computational Information Systems, vol. 8, no. 2, Jan. 2012, pp. 869-876

[23] M.J. Flynn, and S.F. Oberman, Advanced Computer Arithmetic Design”, 2nd ed., New York: John Wiley and Sons, 2001, pp. 113-125

[24] O. Dabeer, and U. Madhow “Channel Estimation with Low-Precision Analog-to-Digital Conversion”, IEEE International Conference on Communications (ICC), vol. 2, May. 2010, pp. 23-27

[25] R. Singh, P. Kumar, and B. Singh, “Performance Analysis of 32-Bit Array Multiplier with a Carry Save Adder and with a Carry Look-Ahead Adder”, International Journal of Recent Trends in Engineering, vol. 2, no. 6, Nov. 2009, pp. 83-86

[26] IEEE Standard for Floating-Point Arithmetic, “IEEE Std 754-2008”, pp.2-7, Aug. 2008, doi: 10.1109/IEEESTD.2008.4610935.

[27] Xilinx Inc., “XtremeDSP DSP48A for Spartan-3A DSP FPGAs User Guide”, UG431 (v1.3), July. 2008, pp.32-34

[28] N. Sorokin, “Implementation of high-speed fixed-point dividers on FPGA”, Journal of Computer Science and Technology, vol. 6 no. 1, May. 2006, pp. 8-11

[29] J. G. Proakis, M. Salehi, and G. Bauch, “Contemporary communication systems using MATLAB”, 3rd edition, CengageBrain. com, 2012

[30] European Telecommunications Standards Institute, Digital cellular telecommunications system (Phase 2+); Radio transmission and reception (GSM 05.05)

[31] A. Ramesh, A. Tilak, and A. Prasad, “An FPGA based high speed IEEE-754 double precision floating point multiplier using Verilog”, International Conference on Emerging Trends in VLSI, Embedded System, NanoElectronics, and Telecommunication System, Jan. 2013, pp.1-5

[32] S. Yassin, I. R. Kamel, and H. Tawfik “A New Design Process to Reduce Resource Usage in SDR Systems”, The Ninth International Conference on Systems, Feb. 2014, pp.1-5.

Appendix A: Software Profiling

What is profiling

The profiling process can determine the time spent by each function by a processor within any software. In addition, it records the order of function calls during execution of the main program. This information is vital to discover software functions that are slower than expected. Accordingly the program can be modified to improve execution time. The profiling can also report the calling frequency of all functions. The calling frequency is the number of function calls over a predefined period of time. This may help spot bugs that had otherwise become unnoticed.

Profiling tools used

A software profiling tool can be classified according to its output as either flat, or call graph profilers. A flat profiler computes the total execution time spent in each function and its percentage of the total running time only. A call graph profiler shows the call times, and frequencies of the functions, and also the call chains involved based on the caller function. As explained earlier in Chapter 2, both execution time and calling frequency are needed to indicate the processor utilization. In the following paragraph, famous call graph profilers are presented.

i. Gprof

The most commonly used profiler on Linux systems is the program gprof. Gprof comes bundled with the open source GNU Compiler Collection (GCC). Profiling a program with Gprof involves three steps:

1. Prepare for profiling by adding profiling options while compiling.
2. Execute the program under evaluation to collect data.
3. Run Gprof to analyze the results.

ii. Valgrind

Valgrind is a suite of tools for both debugging, and profiling. Valgrind is famous for its Memory check tool which can be used to detect memory leaks and errors. However, Valgrind also includes the Cachegrind, and Callgrind tools which can be used to construct a profile of a program. Valgrind is basically a virtual machine or processor emulator. The program should be executed, meaning that it is not a static profiler. Valgrind records information about the instructions the program executes, and the memory accessed.

Since Valgrind is a processor emulator it does not need to augment a program to profile it. This means that we don't need to prepare a special program for profiling. Therefore, special compilation options such as the `-pg' can be removed. This also means that we can run Valgrind on programs that we do not have the source code for. However, running a program through Valgrind will cause the program to run around 50 times slower.

Differences between Gprof and Valgrind

Both techniques can be used to output flat profile, and call graph of the program under evaluation. There are few differences between Gprof and Valgrind as shown in Table 5-1. Although Valgrind output is much slower than Gprof, it will be used in our work because we don't need to compile the source code. This is required to cope with the flexibility of a SDR system. The increased execution time will not be a bottleneck, because the profiling is performed once at the start of the design process.

	Gprof	Valgrind
Speed relative to original code	3x slower	50x slower
Result elaboration	Less readable because results are output to text files	More user friendly due to the automatic creation of call graphs.
Static profiling	Yes, the source code must be compiled	No, it can be run without the source code

Table 5-1: Comparison between two famous software profiling tools

استخدام العتاد الأمثل في أنظمة الراديو المعرفة برمجيًا

إعداد

سامح يس رشاد

رسالة مقدمة إلي

كلية الهندسة جامعة القاهرة

كجزء من المتطلبات للحصول على درجة

ماجستير في

الالكترونيات و الاتصالات الكهربية

كلية الهندسة جامعة القاهرة

الجيزة، جمهورية مصر العربية

٢٠١٤

استخدام العتاد الأمثل في أنظمة الراديو المعرفة برمجيًا

إعداد

سامح يس رشاد

رسالة مقدمة إلي

كلية الهندسة جامعة القاهرة

كجزء من المتطلبات للحصول على درجة

ماجستير في

الالكترونيات و الاتصالات الكهربية

تحت اشراف

الأستاذ الدكتور
محمد حازم توفيق
الأستاذ بقسم الالكترونيات والاتصالات
كلية الهندسة جامعة القاهرة

الدكتور
حسام علي حسن فهمي
الأستاذ المساعد بقسم الالكترونيات
والاتصالات كلية الهندسة جامعة القاهرة

كلية الهندسة جامعة القاهرة

الجيزة، جمهورية مصر العربية

٢٠١٤

استخدام العتاد الأمثل في أنظمة الراديو المعرفة برمجيًا

إعداد

سامح يس رشاد

رسالة مقدمة إلي

كلية الهندسة جامعة القاهرة

كجزء من المتطلبات للحصول على درجة

ماجستير في

الالكترونيات و الاتصالات الكهربية

يعتمد من لجنة الممتحنين:

الأستاذ الدكتور:	أحمد أبو عوف	الممتحن الخارجي
الأستاذ الدكتور:	مجدي فكري رجائي	الممتحن الداخلي
الأستاذ الدكتور:	محمد حازم توفيق	المشرف الرئيسي
الدكتور:	حسام علي حسن فهمي	المشرف

كلية الهندسة جامعة القاهرة

الجيزة، جمهورية مصر العربية

٢٠١٤

مهندس:	سامح يس رشاد
تاريخ الميلاد:	١٩٨٥/٩/٨
الجنسية:	مصري
تاريخ التسجيل:	٢٠٠٨/١٠/١
القسم:	اتصالات و الكترونيات
الدرجة:	ماجستير
المشرفون:	أ.د. محمد حازم توفيق
	د. حسام علي حسن فهمي
الممتحنون:	أ.د. أحمد أبوعوف
	أ.د. مجدي فكري رجائي
	أ.د. محمد حازم توفيق
	د. حسام علي حسن فهمي

عنوان الرسالة:

استخدام العتاد الأمثل في أنظمة الراديو المعرفة برمجيا

الكلمات الدالة:

أجهزة الراديو المعرفة برمجيا، انتفاع مصادر العتاد، مصفوفة البوابات المبرمجة بالحقل، تكافوء قناة الاتصال

ملخص الرسالة:

أنظمة الراديو المعرفة برمجيًا هو اطار عمل يوفر المرونة لأنظمة الاتصالات. يقصد بالمرونة القدرة على استخدام العديد من أنظمة الراديو المختلفة بنفس العتاد عن طريق تغيير البرمجيات فقط. تلك المرونة تتيح اضافة باقة من الخدمات أو أكثر مثل باقة التعديل الموجي، تقدير تأثير قناة الاتصال الخ.

كلما زاد الطلب على زيادة سرعات الاتصال و زيادة حجم المعلومات، تزداد الحاجة الي تطوير أجهزة الراديو المعرفة برمجيا. علاوة على ذلك يمكن استخدامها لتوفير خدمات بأسعار منخفضة للأماكن النائية مقارنة بأسعار تلك الخدمات في المدن الكبرى. لذلك تهتم الحكومات و العديد من الجهات لتطوير خدمات القرى و الأماكن النائية بتطبيق أجهزة الراديو المعرفة برمجيًا، لتحقيق مبدأ "العدالة تكنولوجية".

ثلاث تحديات تواجه أنظمة الراديو المعرفة برمجيا: استهلاك الطاقة الكهربية، استهلاك مصادر العتاد، و وقت تنفيذ العمليات. في هذا العمل ندرس طرق مختلفة لمواجهة تحدي استهلاك مصادر العتاد مع مقايضة التحديان الاخران للحصول على شبكة اتصال مثلى من حيث عدد و جودة الخدمات المتاحة في النظام تحت التطوير. نتائج هذه الدراسة تم التحقق منها عن طريق المحاكاة، و استخدام نظام شهير و هو "برمجيات عالمية للتحكم بالراديو".

الملخص

أنظمة الراديو المعرفة برمجيًا هو اطار عمل يوفر المرونة لأنظمة الاتصالات. يقصد بالمرونة القدرة على استخدام العديد من أنظمة الراديو المختلفة بنفس العتاد عن طريق تغيير البرمجيات فقط. تلك المرونة تتيح اضافة باقة من الخدمات أو أكثر مثل باقة التعديل الموجي، باقة تقير تأثير قناة الاتصال الخ. تأتي أهمية أنظمة الراديو المعرفة برمجيًا من الحاجة اليها في قطاعات هامة مثل الصحة و التطبيقات العسكرية.

كلما زاد الطلب على زيادة سرعات الاتصال و زيادة حجم المعلومات، تزداد الحاجة الي تطوير أجهزة الراديو المعرفة برمجيا. علاوة على ذلك يمكن استخدامها لتوفير خدمات بأسعار منخفضة للأماكن النائية مقارنة بأسعار تلك الخدمات في المدن الكبرى. لذلك تهتم الحكومات و العديد من الجهات لتطوير خدمات القرى و الأماكن النائية بتطبيق أجهزة الراديو المعرفة برمجيا، لتحقيق مبدأ "العدالة تكنولوجية".

ثلاث تحديات التي تواجه أنظمة الراديو المعرفة برمجيا: استهلاك الطاقة الكهربية، استهلاك مصادر العتاد، و وقت تنفيذ العمليات. يمكن الحصول على أداء أفضل في شبكات الاتصال بالتغلب على هذه التحديات.

في هذا العمل ندرس طرق مختلفة لمواجهة تحدي استهلاك مصادر العتاد. بديهيا عند تقليل استهلاك المصادر قد يتأثر التحديان الاخران. لذلك نقايض التديان الاخران للحصول على شبكة اتصال مثلى من حيث عدد و جودة الخدمات المتاحة في النظام تحت التطوير. نتائج هذه الدراسة و المقايضة تم التحقق منها عن طريق المحاكاة بالكمبيوتر، و استخدام عتاد شهير من أنظمة الراديو المعرفة برمجيا و هو "USRP" أو "نظام عالمي للتحكم بالراديو بالبرمجيات".