IMPLEMENTATION OF TWO-DIMENSIONAL DISCRETE AMBIGUITY DISTRIBUTIONS ON FPGA HARDWARE COMPUTATIONAL STRUCTURES

By

David Andrés Márquez Viloria

A project submitted in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING

in

ELECTRICAL ENGINEERING

UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS

2011

Approved by:

Gladys Ducoudray, Ph.D Member, Graduate Committee	Date
Lizdabel Morales, Ph.D Member, Graduate Committee	Date
Domingo Rodríguez, Ph.D President, Graduate Committee	Date
Keith Wayland, Ph.D Representative of Graduate Studies	Date
Erick Aponte, Ph.D Chairman of the Department	Date

Abstract

David Andrés Márquez Viloria

This project presents an approach for the implementation of digital signal processing algorithms on hardware computational structures. The implementation of Discrete Ambiguity Distributions in two dimensions using FPGA (Field Programmable Gate English Array) is shown as a particular case. This work seeks to understand the different approaches that can be followed in implementing signal processing algorithms which can be classified into software implementations, hardware implementations or a combination of both. First, an implementation of hardware/software of a signal generator that uses a digital signal processor is described. Next, an implementation that combines hardware/software on an FPGA to calculate the ambiguity function is shown. Both hardware/software implementations are using the hardware as a component hardware-in-the-loop into the computational cycle to accelerate the processing. Through the use of operators and Kronecker algebra, the ambiguity function can be expressed with a matrix structure that facilitates the implementation in hardware structures and presents an environment for the analysis, design, implementation, and modification of certain class of signal processing algorithms using an integrated hardware/software approach. This approach consists in five fundamental stages: 1) Signal processing algorithm development using the numeric computation software package Matlab®; 2) Formulation of signal processing algorithms in Simulink®; 3) Algorithms implementation using System Generator for DSP™; 4) Field Programmable Gate Array (FPGA) algorithm simulation and emulation; 5) Signal processing algorithm validation through Matlab.

RESUMEN

David Andrés Márquez Viloria

Este trabajo presenta un enfoque hacia la implementación de algoritmos de procesamiento digital de señales en estructuras computacionales de hardware. Se toma como caso particular la implementación de Distribuciones de Ambiguedad Discreta en dos dimensiones usando FPGA (del inglés Field Programmable Gate Array). Se busca entender los diferentes enfoques que se pueden seguir en la implementación de algoritmos de procesamiento de señales, los cuales se pueden clasificar en implementaciones de software, hardware o la combinación de ambos. Primero se presenta una implementación sobre hardware/software de un generador de señales que usa un procesador digital de señales. Luego se muestra una implementación que combina hardware/software sobre un FPGA que calcula la función de ambiguedad. Ambas implementaciones hardware/software usan el hardware como un componente dentro del ciclo de computación para acelerar el procesamiento (lo que se denomina en inglés como hardware-in-the-loop). A través del uso de operadores y apoyados en algebra de Kronecker podemos expresar la función de ambiguedad con una estructura matricial que facilita la implementación en estructuras de hardware lo que nos presenta un entorno para el análisis, diseño, implementación y modificación de ciertas clases de algoritmos de procesamiento de señales usando un enfoque integrado hardware/software. Este enfoque consiste de cinco pasos fundamentales: 1) Desarrollo del algoritmo de procesamiento de señales usando el paquete de software de computación numérica Matlab®; 2) Formulación usando Simulink® de los algoritmos de procesamiento de señales; 3) Implementación del algoritmo usando System

Generator for DSP^{TM}; 4) Simulación y emulación del algoritmo sobre el FPGA; 5) Validación del algoritmo de procesamiento de señales a través de Matlab.

to Camila...

TABLE OF CONTENTS

page

LIST OF TABLES

Table page

LIST OF FIGURES

Figure page

1. INTRODUCTION

1.1 Justification

In digital signal processing, it is common to find algorithms that are computationally expensive. This is not a problem if there is unlimited time to process information and get the required results. However, real time results are needed in most digital signal processing applications. Current information technologies and communication systems demand fast algorithms for data processing. Thus, signal processing researchers must create new tools and develop novel approaches to improve computational algorithm time. In this respect, Field-Programmable Gate Arrays (FPGA) have gained attention and new occupy an important place alongside the Digital Signal Processors (DSP). The DSPs are microprocessors with a specialized architecture for digital signal processing. DSPs are programmed by the user to process a digital signal, i.e. such as filtering. On the other hand, the FPGAs are integrated circuits whose configuration is determined by the user. On the FPGAs is possible to employ parallel programming to reduce computation times.

The ambiguity function ([1], [2]) is widely used in radar and sonar applications in which immediate answers are required. This function relates the transmitted signal and the reflected signals by the located objects. The ambiguity function represents the distribution of the power spectral density of the cross-correlation between the transmitted signal and the reflected signal. The ambiguity function has been widely studied in time-frequency analysis for several applications. It can be interpreted as a cross-correlation in time and frequency, and it is useful to determine

important parameters in radar applications such as the time delay and shift between two signals. Many efforts have been made to compute the ambiguity function faster, Zhang [3] established a clear relationship between the ambiguity function and the Fourier transform to exploit the utility of specialized algorithms such as FFT (Fast Fourier Transform) whose computational complexity is of order $O(nlog(n))$. This approach has been explored in previous research described by Hermanek et. al. [4]. Due to the importance of the ambiguity function in several applications such as radar systems, where the results are required in the shortest time, this project examines the implementation of an algorithm for the ambiguity function using FPGA.

In addition, this project presents an environment for the analysis, design, implementation, and modification of a certain class of signal processing algorithms using an integrated hardware/software approach. This approach consists of five fundamental stages: algorithms: 1) Signal processing algorithm development using the numeric computation software package Matlab; 2) Simulink formulation of signal processing algorithms; 3) System generator algorithms implementation; 4) Field Programmable Gate Array (FPGA) algorithm simulation and emulation; 5) Signal processing algorithm validation through Matlab.

1.2 Objectives

1.2.1 General Objective

The general objective was to develop an environment for analysis, design, implementation, and modification of several signal processing algorithms using an integrated hardware/software approach through the implementation of the ambiguity function on FPGA. This project used a high level tool to incentivize the digital signal processing community with the use of hardware in implementation and simulation of signal processing algorithm.

1.2.2 Specific Objectives

- Design, modeling and simulation algorithms in MATLAB to calculate the ambiguity function.
- Implement the ambiguity function on FPGA hardware structures.
- Evaluate the performance of the implementation with respect to computing time and resources used.
- Present an integrated hardware/software approach.

1.3 Contribution of this work

This project provides a base for the implementation of digital signal processing algorithms on FPGA. In this work an integrated hardware/software algorithm implementation approach was developed. The implementations of cyclic cross-ambiguity functions on FPGAs for large scale signals using BRAM and Pipeline for better resource manage can serve as a base for other implementation.

Signal generator is another important implementation shown in this work. It can be used as an educational tool in courses related to communication systems and signal theory.

In addition, this work examined the use of operators and Kronecker algebra for the analysis and modification of algorithms facilitating the implementation in hardware structures.

1.4 Outline

Chapter 2 of this project presents the theoretical background, including the fundamentals of ambiguity function. In the theoretical framework, the methods for calculating the ambiguity function are described. **Chapter 3** describes the hardware/software algorithm development approach. An example of software algorithm

development, then a hardware/software algorithm implementation using DSP, and finally, a hardware/software integration for an algorithm that compute the ambiguity function using FPGA. This section describes the hardware/software approach developed for implementing digital signal processing on FPGA and is the core of this research. **Chapter 4** presents the hardware/Software algorithm development results. **Chapter 5** presents conclusions and recommendations for future work.

2. BACKGROUND RESEARCH

In this project, an implementation of the ambiguity function is developed using a mathematical formulation based on matrix representation and Kronecker algebra. The next section presents related work which describes the hardware/software implementation for different digital signal processing and some efforts to improve algorithm performance. Next, the theoretical framework underlying the ambiguity function is presented. Finally, a briefly review of FPGA is presented.

2.1 Related Work

Signal processing algorithms have been widely used in the study of SAR(Synthetic Aperture Radar) systems. In [5] Hilaura Nava described a model for the impulse response for advanced SAR systems. Nava focesed on processing data obtained remotely through sensors or sensory data to extract important information. The impulse response of the SAR system was modeled in a context of time-frequency analysis. In [6] Yu Teng et al. developed a three-dimesional model for radar ambiguity offset. Yu Teng et. al. developed a software tool to evaluate the ambiguity of a radar offset and resolution properties. The resulted simulation showed that the degradation of range and doppler resolution depends largely on the system and the geometry of the target. This article describes the development of the test, calibration, and results of the initial field tests.

Several efforts to improve the digital signal processing algorithms have been made. In [7] Ramirez et al. present a theoretical framework for modeling SAR raw data through the computation of the response function systems impulse and cyclic

convolution operations. Ramirez et al.[7] described hardware/software implementations of imaging algorithms on DSP including the ambiguity function.

R. Tolimieri and L. Auslander [2] have conducted numerous studies on time-frequency algorithms, mainly with the ambiguity function and Wigner distribution. A fast algorithm for computing the ambiguity function was designed using decimated finite. This method was compared with the direct method to calculate the ambiguity function. In their work, the count of arithmetic operations is used as a measure of the efficiency of algorithms to evaluate the performance of the proposed method for calculating the ambiguity function.

In [4] Hermanek et al. presents an attempt to implement a numerically efficient and accurate enough accelerator for the calculation of the cross ambiguity function (CAF) on FPGA. The results demonstrated that this accelerator can be used to calculate the real-time CAF for PCL(Passive Coherent Location) radar systems. The design has been implented using PC accelerator cards based on both Xilinx Virtex and Altera Stratix. The reference presented gives information about the algorithms, architecture design, and performance acceleration achieved. The possibilities for future improvements are discussed.

Tolimieri and Winograd [8], consider the calculation of the discrete ambiguity function. Two simple methods are developed: 1) write the discrete ambiguity function as a filter and 2) write the discrete ambiguity function as a discrete Fourier transform. A modification of discrete Fourier transform method produces an approximation to the discrete ambiguity function, but with increased computational efficiency.

In 1993, Rodriguez et al. [9] formulated a methodology based on algebraic methods for the analysis, design, and modifications of time-frequency signal processing. These algebraic methods are integrated into a computational mathematics environment (CME) developed to facilitate the implementation of time-frequency

algorithms on computational hardware structures. This environment allows the design of algorithms as a composition of several operators that can be described using computer mathematical structures, such as algebra tensor products.

2.2 Ambiguity Function: Theoretical Framework

The ambiguity function can be defined as a tool for calculating the generalized autocorrelation signal for simultaneously estimating the time delay and Doppler frequency between a transmitted signal and its echo.

This function accepts the signal transmitted and received as inputs and generates a surface in two dimensions, one dimension is the time and the other the frequency. These parameters can be used to estimate, through simple algebraic transformations, the range and cross range (azimuth) in the spatial domain of the object.

The range and speed, time and frequency corresponding to the peak in the ambiguity surface can be provided the delay time and frequency offset of the received signal.

The following mathematical description is based on the formulation presented by Ramirez-Rodriguez. [7]. The ambiguity function is given by:

$$A : l^2(Z_N) \times l^2(Z_N) \rightarrow l(Z_N \times Z_N) \tag{2.1}$$

$$(x, y) \rightarrow A(x, y) \tag{2.2}$$

where,

$$A_{(x,y)}[m, k] = \sum_{n \epsilon Z_N} x[n] y^*[\langle n + m \rangle_N] e^{-j2\pi kn/N}; m, k \ \in \ Z_N \tag{2.3}$$

The ambiguity function can be represented using two methods: 1) Filter Method and 2) Transform Method.

2.2.1 Filter Method

Let $x_T, x_R \in l^2(Z_N)$ to be the discrete ambiguity function of x_T and x_R is given by the following expression:

$$A_{x_T,x_R}[m,k] = \sum_{n \in Z_N} x_T[n] x_R^*[\langle n+m \rangle_N] e^{-j2\pi kn/N};\ for\ m,k\ \in\ l^2(Z_N) \tag{2.4}$$

Let,

$$M_k = \psi_k\ \in\ \Psi = \psi_k\ \psi_k[n] = e^{j2\pi kn/N}; k, n \in Z_n. \tag{2.5}$$

The set ψ is called the set of character basis vectors on simply characters. The action of M_k on the signal x_T is defined as follows:

$$M_k\ l^2(Z_N) \rightarrow l^2(Z_N) \tag{2.6}$$

such that,

$$(M_k\{x_T\})[n] = x_T[n]\psi_k[n], n \in Z_N \tag{2.7}$$

Let,

$$x^*_{R\{\langle -m \rangle_N\}} = S_N^{\langle -m \rangle_N}\{x_R^*\}. \tag{2.8}$$

Then,

$$\left(x^*_{R\{\langle -m \rangle_N\}}\right)[n] = \left(S_N^{\langle -m \rangle_N}\{x_R^*\}\right)[n] = x_R^*[\langle n+m \rangle_N]\ allowing\ \langle -m \rangle_N = \langle N-m \rangle_N = l < N \tag{2.9}$$

Thus,

$$(x_R^*\{\langle -m \rangle_N\})[n] = (x_R^*\{l\})[n] \tag{2.10}$$

Gathering expressions results in the following formulation:

$$A_{x_T,x_R}[m,k] = \sum_{n\in Z_N} \left(M_k^*\{x_T\}\right)[n] = \sum_{n\in Z_N} \left(M_k^*\{x_T\}\right)[n] \left(S_N^l\{x_R\}\right)^*[n] \tag{2.11}$$

finally,

$$A_{x_T,x_R}[m,k] = \langle \left(M_k^*\{x_T\}\right)[n] \left(S_N^l\{x_R\}\right)\rangle \tag{2.12}$$

2.2.2 Transform Method

Let x_T, $x_R \in l^2(Z_N)$ be the discrete ambiguity function of x_T and x_R is given by the following expression:

$$A_{x_T,x_R}[m,k] = \sum_{n\in Z_N} x_T[n] x_R^*[\langle n+m\rangle_N] W^{kn} \tag{2.13}$$

for $m,\ k \in l^2(Z_N)$

Let,

$$S_N^{\langle -m\rangle}\{x_R^*\} = x_R^*\{-m\} \tag{2.14}$$

Also, allow

$$x_m = x_T \odot_N x_R^*\{-m\} \tag{2.15}$$

Thus, an evaluation of x_m al $n \in Z_N$ results in:

$$x_m = x_T[n] x^*_{R\{-m\}}[n], n \in Z_N \tag{2.16}$$

Then,

$$A_{x_T,x_R}[m,k] = \sum_{n\in Z_N} x_m[n] W_N^{kn} = ((x_m))[k] = \hat{X}_m[m] \tag{2.17}$$

where, $\hat{X}_m[m] = F_N\{x_m\}$

2.3 FPGA

A Field Programmable Gate Array or FPGA is a semiconductor circuit device that contains configurable logic blocks (CLBs) which can be arranged to perform sequential or combinatorial functions. Around the CLBs there are input/output blocks (IOBs) which connect CLB inputs and outputs to pins in the chip package. The CLBs can be connected to each other using programmable routing channels. The function of the user is to define the logic functions of each CLB and how the IOB should work, and interconnect. This configuration can be programmed using either a Hardware Description Language (HDL) such as VHDL (Very High Speed Integrated Circuit Hardware Description Language), or Verilog, or a schematic.

The FPGA supports parallel programming easily for its architecture. However, the algorithms are developed in serial approaches in most cases. Serial programming are widely used due to the conventional hardware structures that are characterized to limite parallel process. Multicores processor, GPU (Graphical Processor Unit) and FPGA have stimulated the development of new tools for simulation, testing and evaluation of parallel programs such as pMatlab [10].

3. HARDWARE/SOFTWARE ALGORITHM DEVELOPMENT APROACH

The following presents a software implementation of the ambiguity function using Matlab and pMatlab. The implementation of a signal generator using software and hardware is also described. The signal generator was designed using Matlab, Code Composer Studio®and a Texas Instrument DSP board. The implementation is described and the importance in the use of hardware for the digital signal processing algorithms is established. Finally, the hardware/software implementation of the ambiguity function and the approach used to develop that implementation are described. The five stages proposed for the implementation of digital signal processing algorithms using hardware and software followed in the implementation of the ambiguity function using on FPGA are identified.

3.1 Software Algorithm Development

Software development algorithm in this work means primarily implementations that run on computational structures with an operating system. Hence the final decisions on resource management is left to the operating system even when programmer has a good grasp of the architecture and the data transmission process.

In general, digital signal processing algorithm implementations that use this type of development reduce design time. Matlab has powerful graphics tools. Array-based programming is very intuitive for most algorithms. pMatlab (a toolbox design for Matlab) allows multicore simulation and implementation of parallel structures. 3–1 shows the implementation for calculating the ambiguity function of a square

```
%-------Ambiguity Function
    A=zeros(N,N);
    for i=0:1:N-1
        Sr_Shift=circshift(Sr,N-i);
        A(:,i+1)=fftshift(fft(St.*conj(Sr_Shift)));
        subplot(2,1,1);
        stem([0:(length(St)-1)],St,'--*r');
        hold on
        stem([0:(length(St)-1)],Sr_Shift)
        hold off
        title('Square Pulse')
        subplot(2,1,2)
        [X,Y] = meshgrid(-N/2:1:N/2-1);
        mesh(X,Y,abs(A));
        xlabel('Delay');
        ylabel('Doppler');
        title('Ambiguity Function - Square Pulse')
        pause(0.001)
    end
```

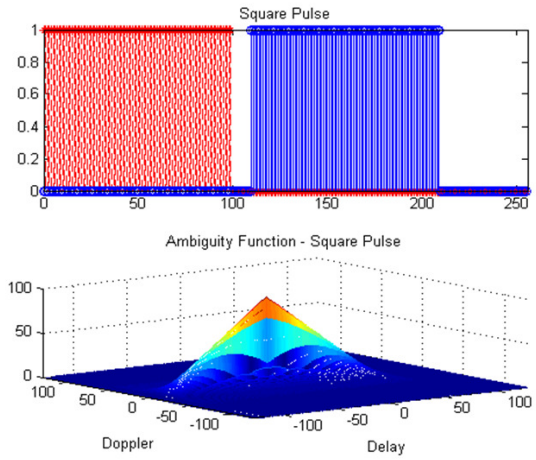


Figure 3–1: Ambiguity Function Square Pulse using Matlab

wave. Figure 3–2 presents the implementation of that ambiguity function in pMatlab.

Step	Pseudocode
1	*for j = 1 to num_global_indices* *Amblocal ← local_correlation_matrix* *next*
2	*Ambloc ← FFT(Amb)*
3	*Amb ← Put_Local_Array_on_Distributed_Array(Amb,Amblocal)*
4	*A ← Convert_to_Regular_Matlab_Array(Amb)*

Step	Description
1	Fill local correlation matrix
2	Calculate the FFT on columns of Correlation Matrix Amblocal (Ambiguity Function Local)
3	Join partial results in distributed Matrix Amb
4	Assign the pMatlab Distributed Array Values to Matlab Regular Array

Step	Matlab/pMatlab Code

1	`for j=1:numel(jglobal)`
	`Ambloc(:,j) = F.*[G(jglobal(j):M) G(1:jglobal(j)-1)];`
	`end`

2	`Ambloc = fft(Ambloc);`
3	`Amb = put_local(Amb,Ambloc);`
4	`A = agg(Amb);`

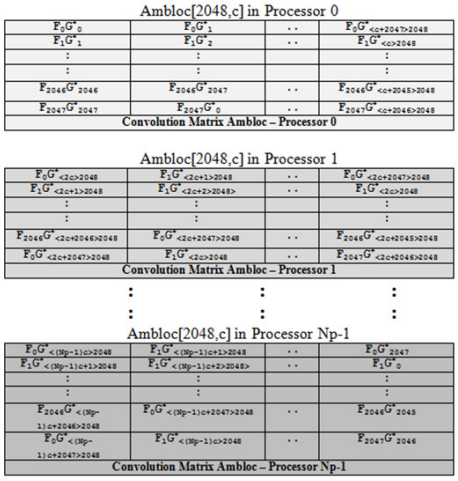


Figure 3–2: Ambiguity Function using pMatlab

Using these tools in the algorithm design can cause the use of multiple resources and them management by the operating systems can result in very high computation times. For algorithms like these of the ambiguity function that can make real time results difficult or impossible. That is the key reason to combine hardware and software. Thereby allowing better computer simulation time well as leading to possible real time implementation. The first stage is to use a DSP microprocessor a defined

architecture that is programmable in C. Implementation with a DSP microprocessor allows a close analogy to the algorithm software development. The next stage to use an FPGA. The FPGA allows greater flexibility it does not have a defined structure and the user builts the required hardware.

3.2 Hardware Algorithm Development

This section shows a hardware algorithm implementation of a signal generator using a DSP. The signal generator produces a linear frequency modulated signal (generally called a Chirp signal) added to several sinusoidal signals. The TMS320C6713 DSP board and the Code Composer Studio (CCS) were used in this signal generator implementation. A graphical interface using Matlab was designed to provide easier interaction with the signal generator.

In this implementation the user can insert the signal parameters such as amplitude, frequency and phase. When the application starts the signal generator produces a signal with the default parameters. When new parameters are entered the application produces without halting the signal generator. The interface shows the spectrum and the time-frequency spectrogram. The Chirp signal was used becouse its time-frequency characteristics are widely known. The signal generator can be developed used in applications that use chirp signals such as filter design, radar systems, and imaging. The implementation not only generates Chirp signals, the signal generator can generate in real time several waveforms that are sums sinusoidal signals. The signal generator can also be used as an educational tool communication systems and signal theory, related courses.

The most important result the implementation of the real time signal generator in hardware. The Texas Instrument DSP board TMS320C6713 of commonly known as 6713 is used. This DSP board comes with a user-friendly for entering the signal generator data.

The next sections show the system design in detail, including the routines used and how to use the system generator. First a description of the Chirp signal is provided. Next a description of the DSP board 6713 is presented. In section 3.2.3, the implementation of the signal generator is described.

3.2.1 Chirp signal

Chirp signals are characterized by the changing their frequency over time. These signals are widely used in several applications such as radar systems. Chirp signals are valuable tools for evaluating filters.

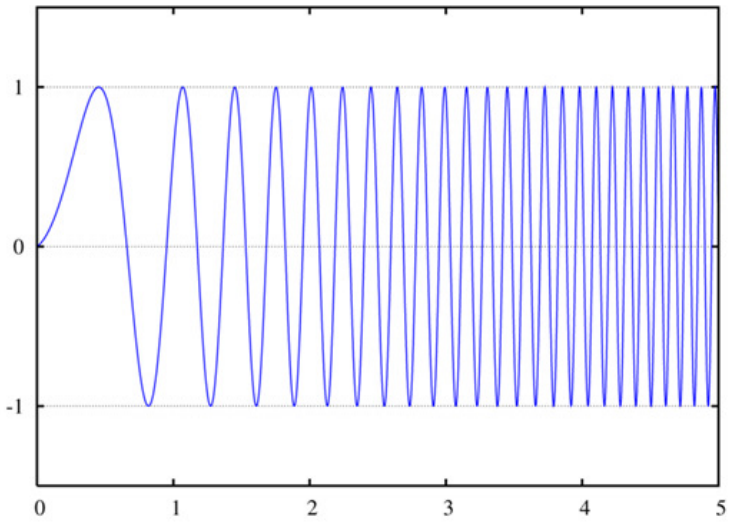


Figure 3–3: Diagram of Chirp Signal

The frequency of a linear chirp signal can be expressed as a function of the time t by:

$$f(t) = f_0 + kt$$

where f_0 is the initial frequency (the frequency at time t=0), and k is the frequency growth rate often called "Chirp Rate" (the instantaneous rate of change in wave frequency).

For example, a sinusoidal wave defined by the form:

$$x(t) = sin((\phi(t))),$$

the instantaneous frequency is given by:

$$f(t) = \frac{1}{2\pi}\frac{d\phi(t)}{dt}$$

and the Chirp Rate of the wave is defined by:

$$c(t) = \frac{1}{2\pi}\frac{d^2\phi(t)}{dt^2}$$

Now the sinusoidal linear Chirp function can be written as:

$$x(t) = sin\left(2\pi(f_0 + \frac{k}{2}t)t\right)$$

A practical way to observe the Chirp signal is as a spectrogram (time-frequency representation based on the short time frequency transform STFT). The spectogram for a sinusoidal linear Chirp signal is shown in the figure 3–4. (See source-code in the appendix A.1).

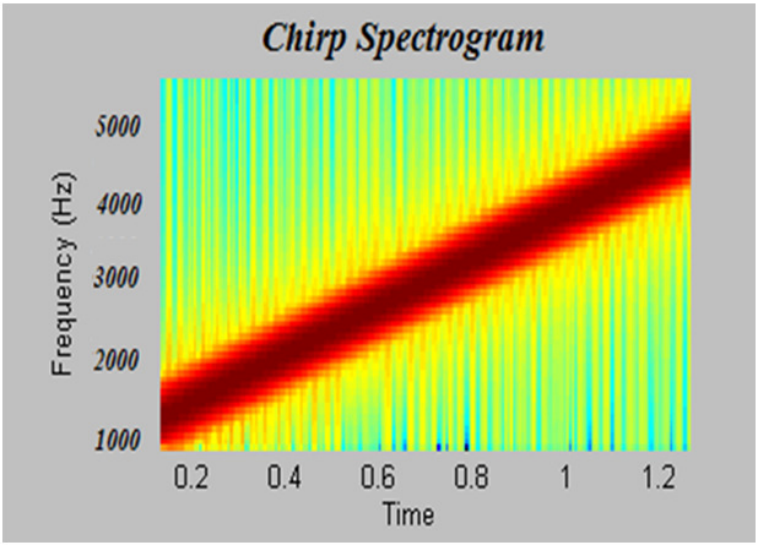


Figure 3–4: Chirp Spectrogram

Figure 3–5 shows the spectrum for the sinusoidal linear Chirp signal.

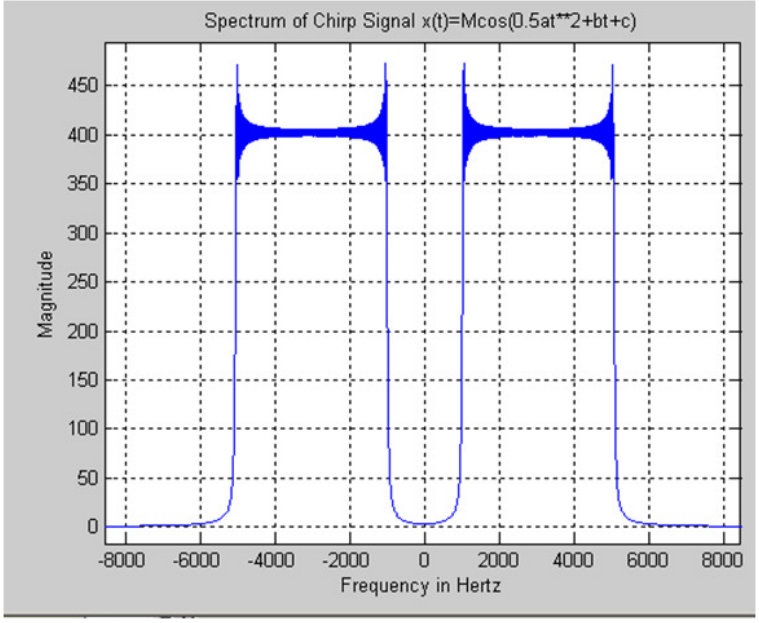


Figure 3–5: Chirp Spectrum

It is not possible to observe a temporal graphic of the full signal because the sampling frequency is very high. Three sections of the signal are shown in Figures: 3–6, 3–7 and 3–8. Figure 3–6 shows the signal generated from 0 to 5 milliseconds. Figure 3–7 shows the signal generated between 345 and 350 milliseconds and the Figure 3–8 shows the signal from 1000 to 1005 milliseconds. Note as the frequency increases with the time.

3.2.2 Characteristics of DSP 6713

Texas Instruments DSP board TMS320C6713 used in this project. Figure 3–9 shows a picture of the Texas Instrument DSP 6713.

The main technical specifications of the DSP 6713 are:

- Operates at 255 MHz.
- AIC23 stereo codec.
- 8 Mbytes of synchronous DRAM.
- 512 Kbytes of nonvolatile Flash memory (256 Kbytes usable in default).

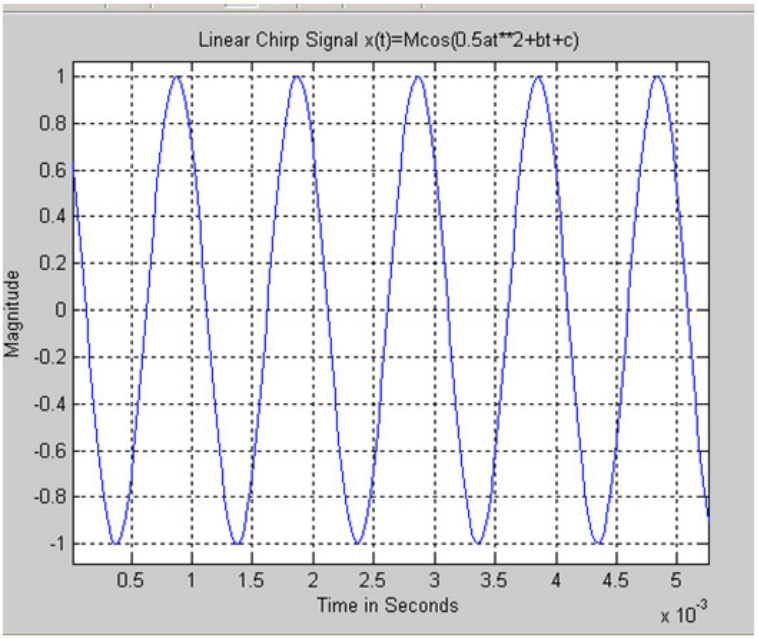


Figure 3–6: Chirp 0 to 5 milliseconds

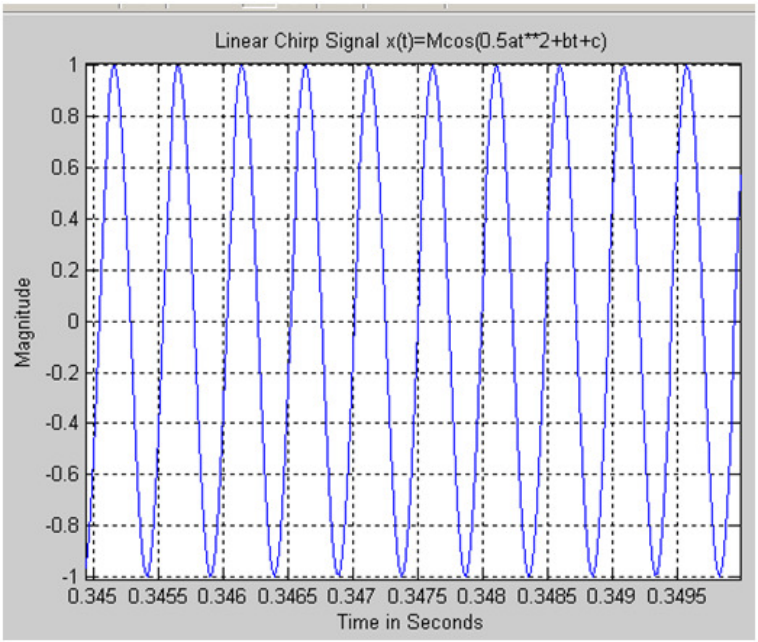


Figure 3–7: Chirp 345 to 350 milliseconds

- 4 LEDs and DIP switches accessible by the user.
- Software board configuration through registers implemented in CPLD.
- Configurable boot options.

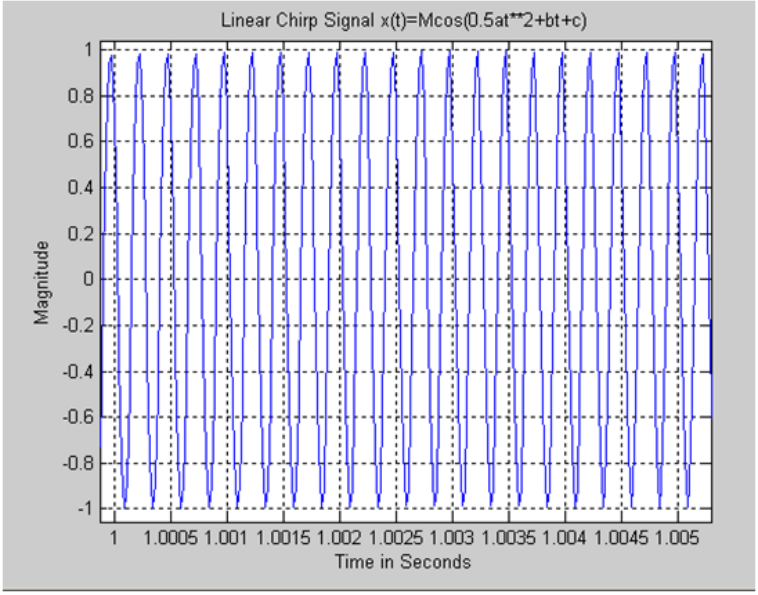


Figure 3–8: Chirp 1000 to 1005 milliseconds

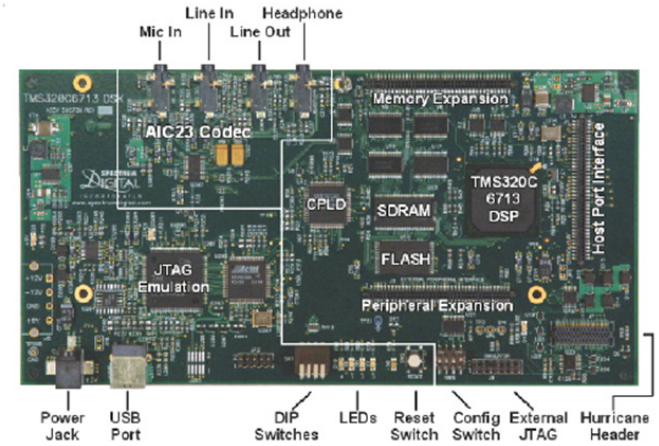


Figure 3–9: Texas Instruments TMS320C6713

- Standard expansion connectors for daughter board.
- JTAG emulation through the emulator on board with USB interface or external emulator.
- Single power supply +5V.

The DSP 6713 has a USB port that can be used to transmit data between the DSP and the PC for real time operation. Texas Instruments has a package for the

Code Composer Studio (CCS) containing Real Time Data Exchange (RTDX) to allow users to exchange real time data between the board and the PC. This toolbox allows the simulation of input and output data, which can be used as feedback for a variety of DSP applications. The input and output signals are stored in buffers allowing the transmission of large data packages.

3.2.3 Implementation of Signal Generator: Frequency Modulated Linear and Harmonic signals

The signal generator developed produces two groups of signals, one is the sum of the harmonics and the other is the Chirp signal. The user can select between generating a pure cosine signal or the sum of several cosine signals and between generating a Chirp signal or a combined harmonic and Chirp signal.

Harmonic signal implemented in the signal generator is given by :

$$x_1(t) = \sum_{k=1}^{N} A_k \cos(2\pi f_k t + \phi_k)$$

$$let N = 5$$

$$x_1(t) = A_1 cos(2\pi f_1 t + \phi_1) + A_2 \cos(2\pi f_2 t + \phi_2)$$

$$+A_3 \cos(2\pi f_3 t + \phi_3) + A_4 \cos(2\pi f_4 t + \phi_4) + A_5 \cos(2\pi f_5 t + \phi_5)$$

In this case, the user can enter the amplitude A, the frequency f and the phase ϕ.

The Chirp signal is given by:

$$x(t) = A \cos(0.5at^2 + bt + c)$$

The user can enter values for the amplitude A, the initial frequency f_a , the final frequency f_b, and the constant c.

A Matlab GUI (Graphical User Interface) was developed as an interface for the implemented signal generator. All CCS functions such as open a project, load a

project in the board, and run the program are controlled from Matlab Figure 3–10. The integration of CCS, Matlab and the DSP 6713 in a block diagram Figure 3–10.

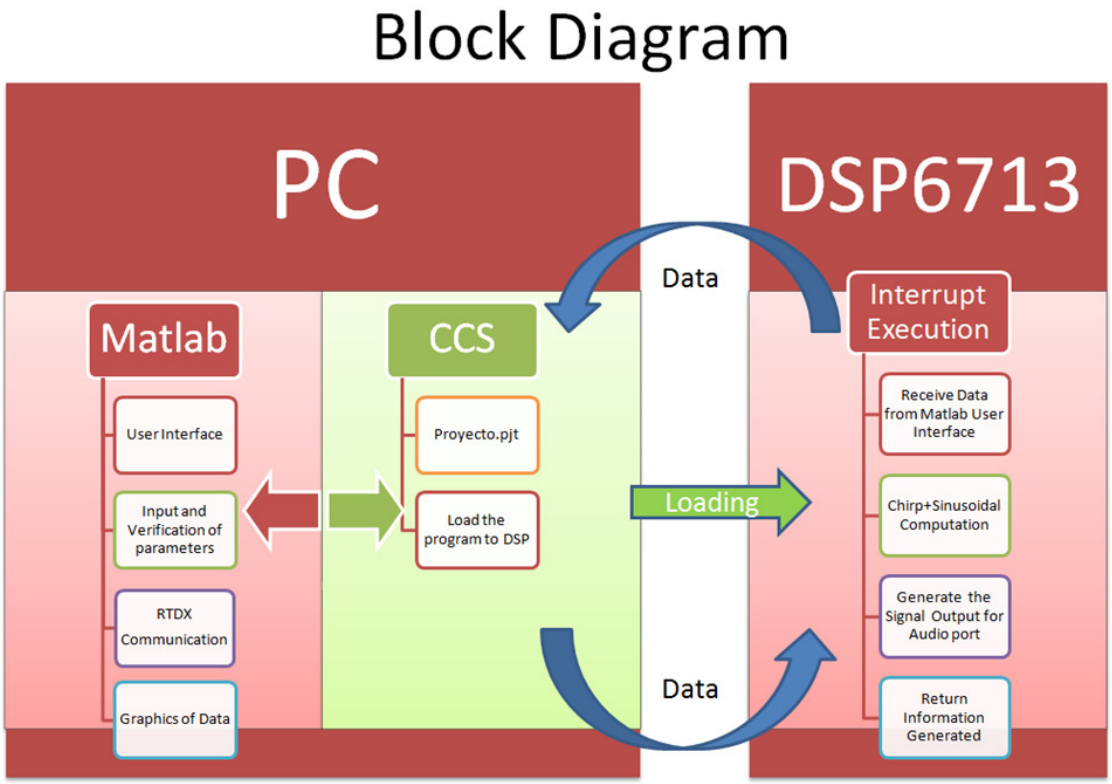


Figure 3–10: Signal Generator Block Diagram

The functional diagram shown in Figure 3–11 illustrates how the routines on Matlab and CCS platforms. The primary purpose of the Matlab platform is to produce a $*.m$ file (called $signal\_generator.m$ in Figure 3–11). The $*.m$ file contains all the information generated by the graphical user interface (GUI). The GUI is stored in a file with the same name as the main function except with extension $*.fig$. The classes for each component in the GUI (e.g. $checkbox$, $pushbutton$, $edit$, $axis$) are included in the $signal\_generator.m$ as well as the loading data, input data verification, results plot and communication routines.

The CCS routines are managed for the project $signal_generator.pjt(SeeFigure)$.

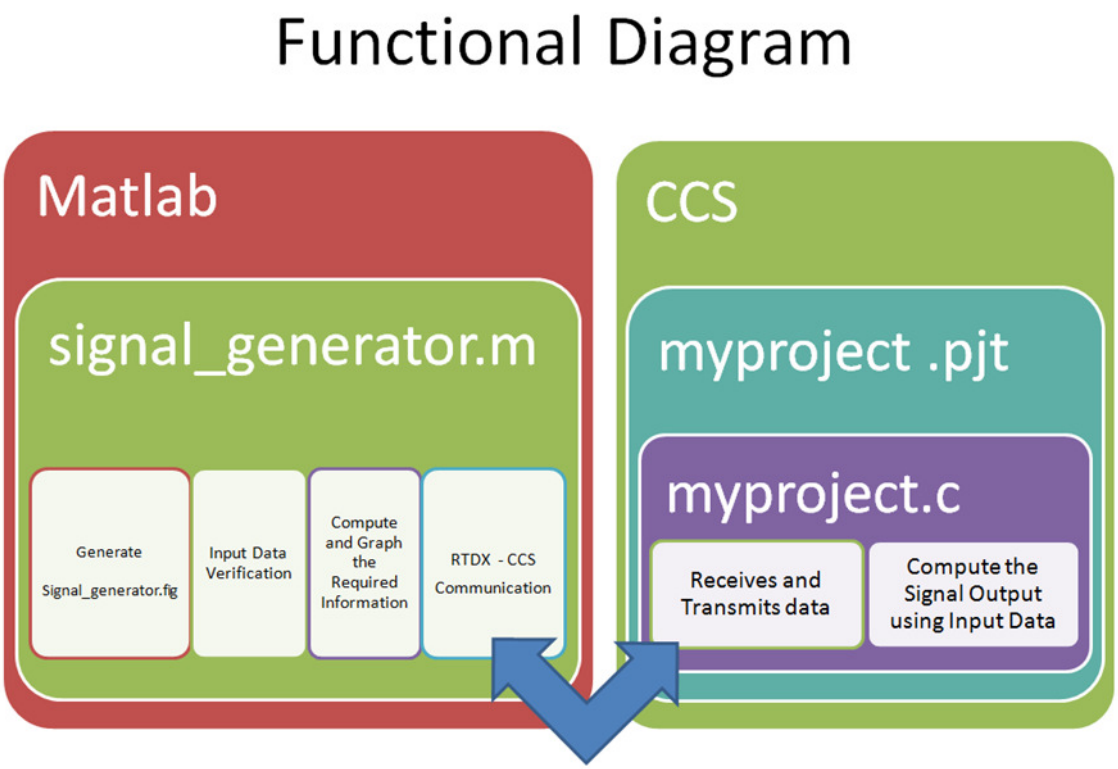


Figure 3–11: Signal Generator Functional Diagram

Figure 3–12 presents the start windows for the Matlab implemented signal generator GUI.

When the *start* button is pressed, the routine verifis the DSP to be sets up the RTDX, runs CCS, opens, download and run the project in the DSP. This button uses the routine described in appendix . The *stop* button ends the program and breaks communication with the CCS. After pressing stop, restart is required. The *stop button* routine is included in appendix A.2.

The user enters the signal parameters with the edit buttons provided in the signal generator GUI. The *duration* parameter is the signal duration. The number of point to be generated and the rate of frequency increase are determined from the duration parameter. The Fs parameter is the sampling frequency to be used in the generated graphics. Two *checkboxes* are provided in the GUI. If the first option is

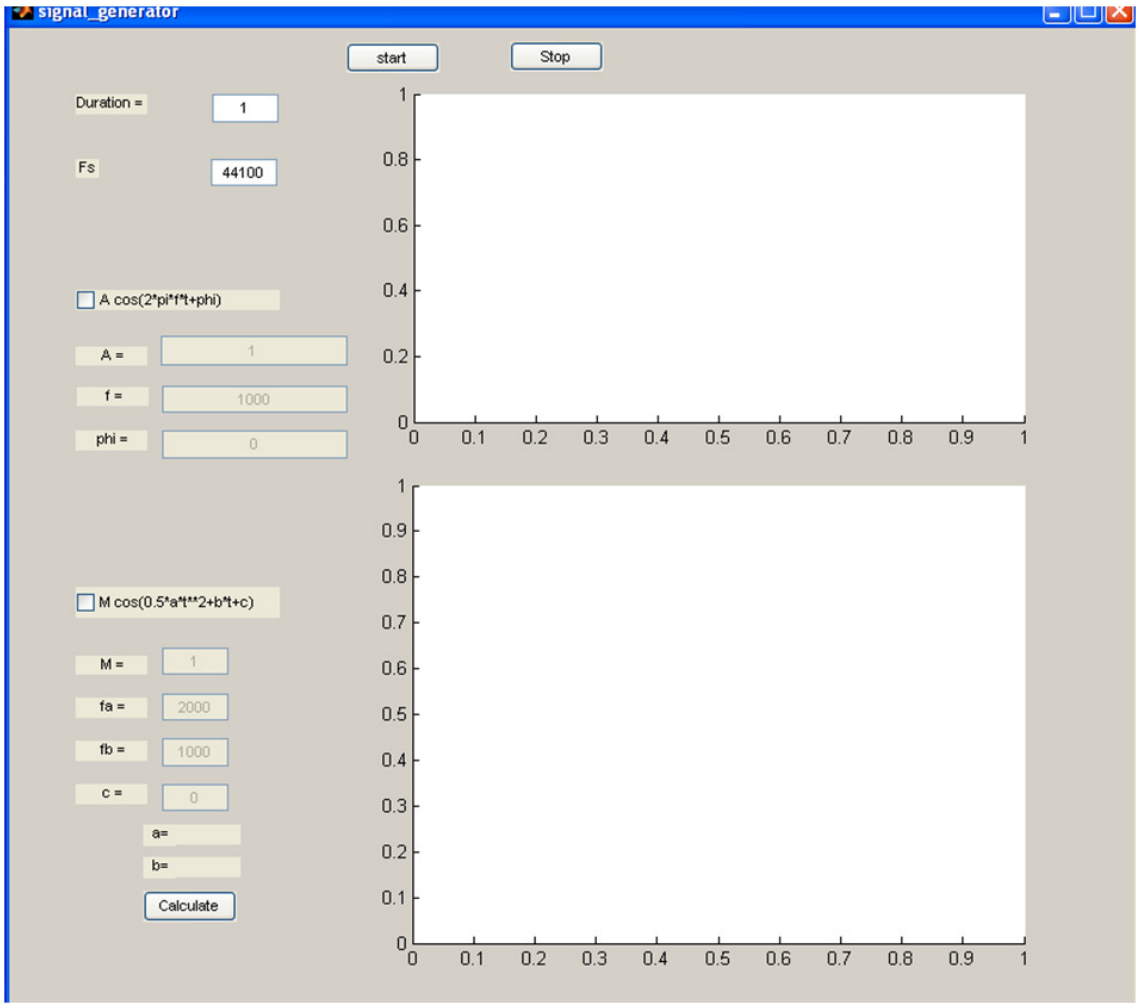


Figure 3–12: GUI Signal Generator

selected, then one cosine or the sum of several cosine signals is generated. If the second *checkbox* is selected, then a Chirp signal is generated. If both *checkboxes* are selected, then the output is a combination of cosine and chirp signals. Cosine signals require amplitude, frequency and phase parameters. Since the sum of several cosine signals requires multiple sets of parameter, the Matlab entries can be vectors for the amplitudes, frequencies and phases. For example, the data can be entered in any of the following forms:

```
1
```

```
0.56
```

```
-1
-2.57
[1 0.56 -1 -2.57]
[1 0.56 -1 -2.57 35 0.002 4 -100 10000]
```

Clearly, the number of amplitudes entered must equal the number of frequencies and phases. The chirp signal requires for parameters: amplitude M, the initial frequency fa, and the final frequency fb, and the phase (c). Note that one only chirp signal can be generated at a time.

The button *calculate* executes the routine to accept the user entered data, pastes the data into an array and sends it to the DSP. Then the DSP generates a signal with the specified parameters which is displayed. It also send a signal to the audio output of PC to produce a sound to associate with the visual output. The routines used in this whole process are presented in appendix. The $*.c$ file implemented for the CCS is included in appendix A.2.

When the Matlab program is initiated, the application is ready to receive data. The CCS displays "Waiting to read". Once the signal parameters are entered Matlab prepares and transmits the data. The CCS indicates a label of "Read Completed" as in Figure 3–13.

Once the data is read, the program generates a signal point by point and sends it to the analog output. Each time a new signal point is calculated the CCS updates all the variables. The program continues until the stop button is pressed.

Figure 3–14 presents the results produced with the signal generator combining a Chirp and a cosine signal. Signal generated by the sum of five cosine signals is shown in Figure 3–15. Figure 3–16 shows a single Chirp signal and Figure 3–17 shows the signal produced by the sum of that Chirp signal with the sum of the five cosine signal.

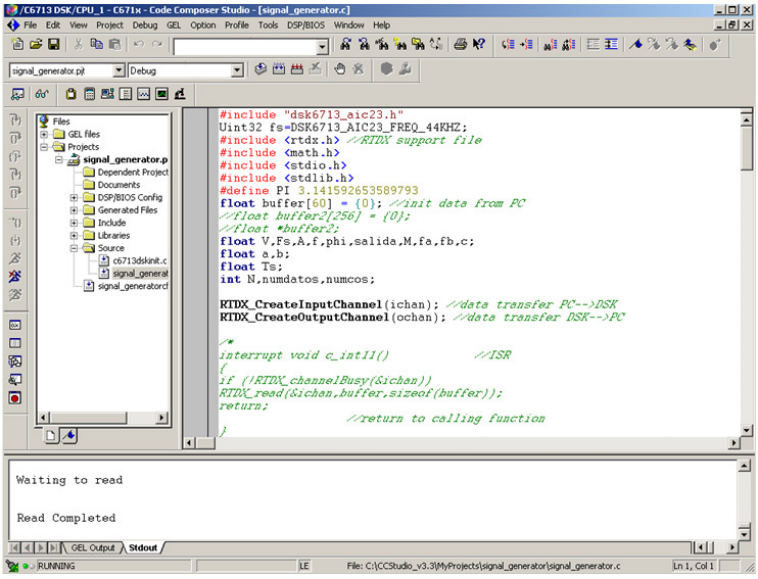


Figure 3–13: Running Code Composer Studio CCS

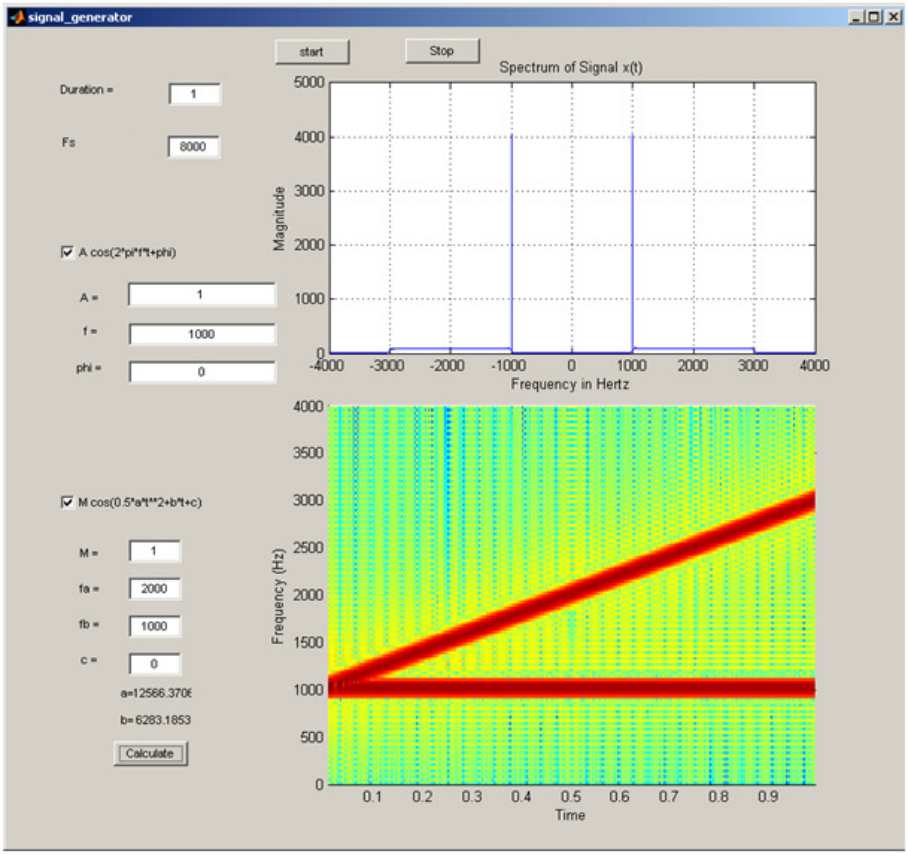


Figure 3–14: Sum of Chirp and Cosine

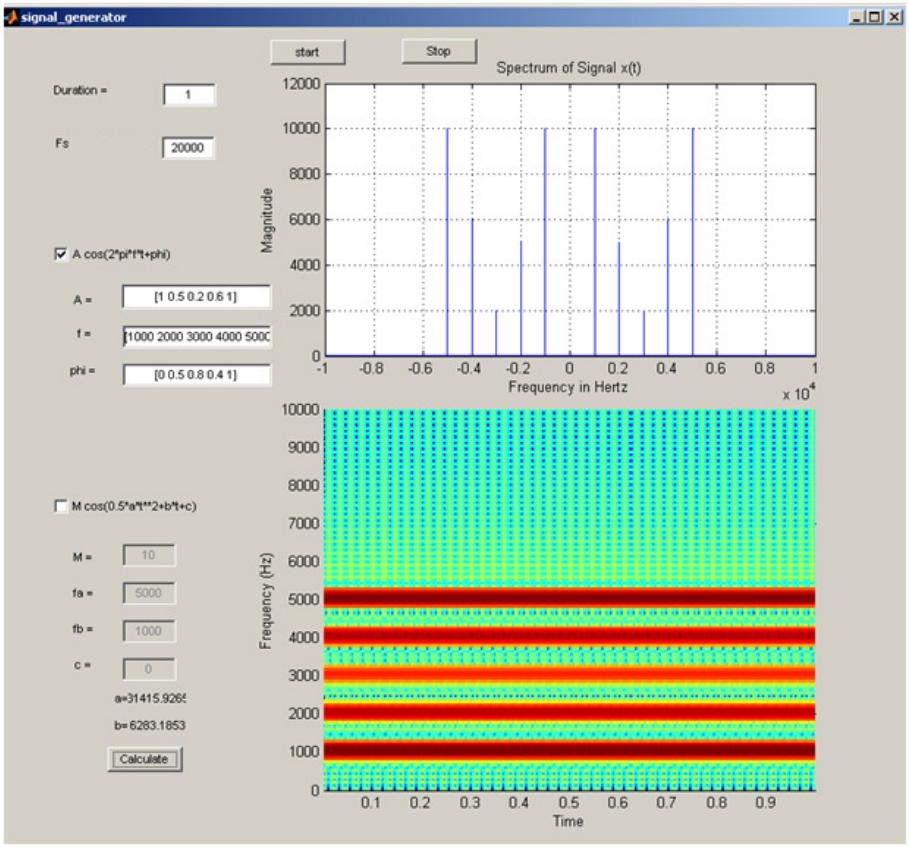


Figure 3–15: Sum of Cosines

The GUI signal generator GUI developed is a user-friendly graphical interface that facilitates the DSP signal generation. The users do not need to work directly with the CCS, they only have to introduce the signal parameters and select one of the three signal generator options: chirp signal, sum of cosine signals, or sum chirp and cosine signals. The number of cosine signals that can be summede was limited to 17 by the buffer size. Note that once the application begins to generate signals, these are generated continuously without interruption, even when the user updates the signal parameters.

3.3 Hardware/Software Integration for Algorithm Development

The ambiguity function is widely used in the analysis of radar and sonar signals where fast computation is required. Researchers in the signal processing field have proposed an algorithm based on the Fast Fourier transform (FFT). In this section

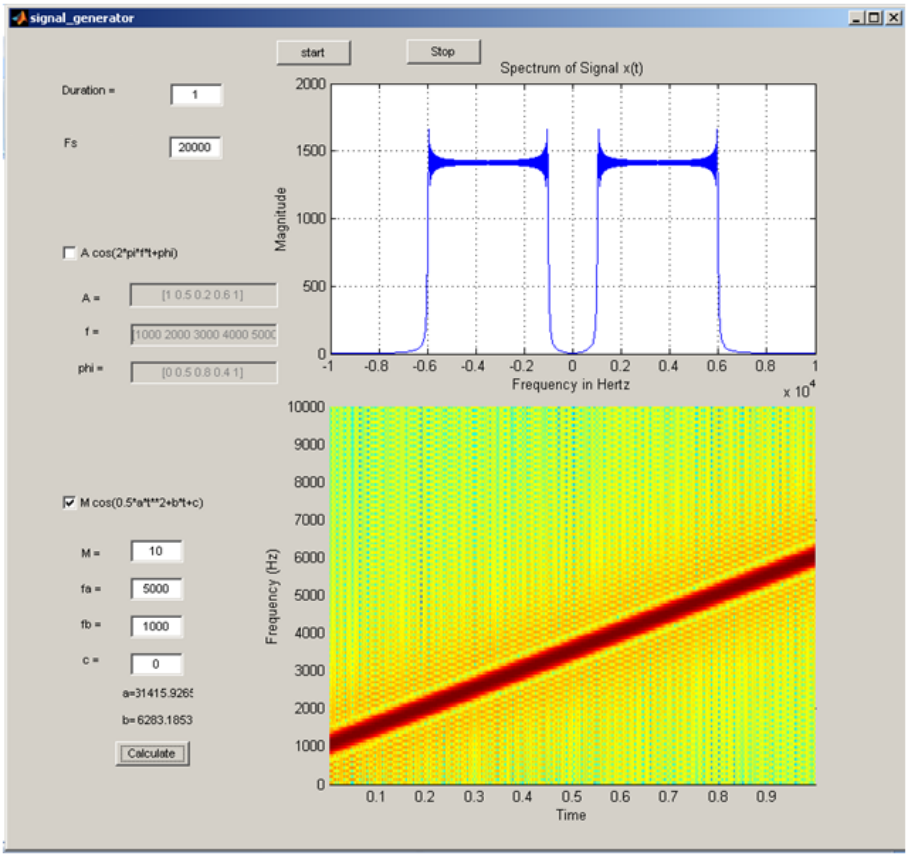


Figure 3–16: Chirp Signal

the implementation of the ambiguity function based on the FFT over a Field Programmable Gate Array (FPGA) is described. The implementation was designed using pMATLAB in order to paralleliza the algorithm. Signals of several legths were used to test the ambiguity function implementation with signals of up to 16384 points.

In addition, this section presents an environment for the analysis, design, implementation, and modification of a certain class of signal processing algorithms using an integrated hardware/software approach. This approach consists in five fundamental stages: 1) Signal processing algorithm development using the numeric computation software package Matlab®; 2) Formulation of signal processing algorithms in Simulink®; 3) Algorithms implementation using System Generator for DSP™; 4) Field Programmable Gate Array (FPGA) algorithm simulation and emulation; 5)

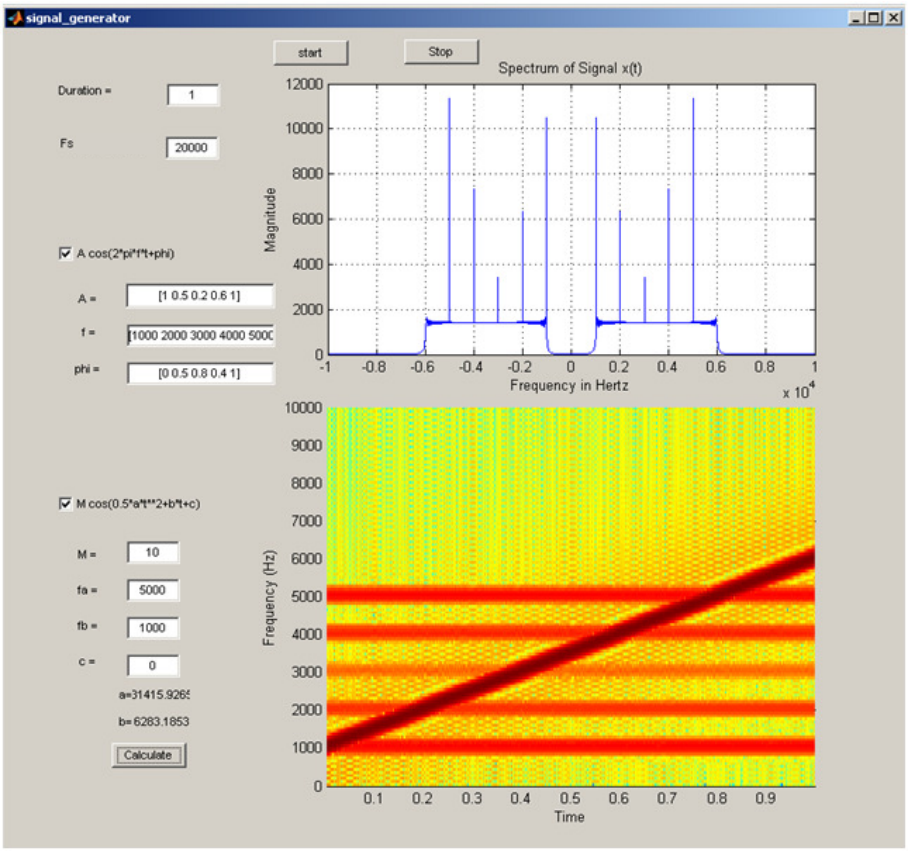


Figure 3–17: Sum of a Chirp Signal and Multiple Cosine Signals

Signal processing algorithm validation through Matlab. The next section provides a concrete example of a successful signal processing algorithm design and implementation using this FPGA hardware/software approach. The example belongs to the class of signal processing algorithms known as time-frequency distribution. In particular, the example deals with the computation of the cross-ambiguity function for a transmitted signal and its return echo ([1], [2]). The radar cross-ambiguity function was successfully implemented on a Virtex 5 field programmable gate array unit using this approach.

3.3.1 Ambiguity Function Formulation

The mathematical formulation of the cross-ambiguity function follows. The ambiguity function is a two-dimensional function that shows the time delay and Doppler frequency $A_{f,g}[m,k]$ based on the reflected pulse distortion caused by the

receiver match filter. Woodward first introduced the ambiguity function in his seminal book *Probability and Information Theory with Applications to Radar* as the means to solve the radar measurement problem. The ambiguity function can be viewed as a linear operator problem as described in [9]. The following formulation is based on this approach.

The narrowband continuous formulation of ambiguity function is given by the expression:

$$A_{f,g}(m,k) = \int_{-\infty}^{\infty} f(n)\, g^*(n-m)\, e^{-jtk} dn$$

where $f(n)$ is the broadcast signal, $g(n)$ is the return signal, $g^*(n)$ is the complex conjugate of the return signal, m is the delay between both signals and k represents the Doppler frequency.

Discrete formulation of the ambiguity functions is given by the expression:

$$\mathscr{A}_{f,g}[m,k] = \sum_{n \in Z_N} f[n]\, g^*[\langle n+m \rangle_N]\, e^{-j\frac{2\pi}{N}kn}$$

$$\mathscr{A}_{f,g}[m,k] = (I_N \otimes F_N)\, v = \begin{bmatrix} F_N & & & \\ & F_N & & \\ & & \cdots & \\ & & & F_N \end{bmatrix}_{N^2 \times N^2} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix}_{N^2 \times 1}$$

$$= \begin{bmatrix} F_N h_0 \\ F_N h_1 \\ \vdots \\ F_N h_{N-1} \end{bmatrix}_{N^2 \times 1} = \begin{bmatrix} H_0 \\ H_1 \\ \vdots \\ H_{N-1} \end{bmatrix}_{N^2 \times 1}$$

Gathering this result, we have:

$$\mathscr{A}_{f,g}[m,k] = (I_N \otimes F_N)\, v \rightarrow \begin{bmatrix} H_0 & H_1 & \dots & H_{N-1} \end{bmatrix}_{N \times N}$$

We can observe N independent processes, making this approach a true *Parallel Operation*. This may be implemented as distributed matrices in pMatlab.

For the Ambiguity Function has the following formulation:

$$\mathscr{A}_{f,g}[m,k] = \sum_{n \in Z_N} f[n]\, g^*[\langle n+m \rangle_N]\, e^{-j\frac{2\pi}{N}kn}$$

let $h_m[n] = f[n]\, g^*[\langle n+m \rangle_N]$

Then, $\mathscr{A}_{f,g}[m,k] = \sum_{n \in Z_N} h_m[n]\, W_N^{kn}$, where $W_N^{kn} = e^{-j\frac{2\pi}{N}kn}$

If the *Shift Operator* S_N acts recursively on g^*, we have:

$$S_N\{g^*\}[n] = d[n] = g^*[\langle n-1\rangle_N] \rightarrow S_N\{S_N\{g^*\}\} = S_N^2\{g^*[\langle n-2\rangle_N]$$

In general, $S_N^m\{g^*\} = g^*[\langle n-m\rangle_N]$

Substituting $-m$ by m, yield $S_N^{-m}\{g^*\} = g^*[\langle n+m\rangle_N]$

Thus, $h_m[n] = f[n]\, S_N^{-m}\{g^*\}[n]$.

In general, $h_m = fS_N^{-m}\{g^*\}$

Let $\delta_{\{m\}}$ be mth orthonormal basis vector in a vector space of N dimensions. So,

$$\delta_{\{0\}} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}_{N\times 1} \qquad \delta_{\{1\}} = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}_{N\times 1} \qquad \delta_{\{N-1\}} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}_{N\times 1}$$

Let $v = \sum_{m\in Z_N} \delta_{\{m\}} \otimes h_m = \sum_{m\in Z_N} \delta_{\{m\}} \otimes \left(fS_N^{-m}\{g^*\}\right)$

Where $\delta_{\{m\}}[n] = \delta_{\{0\}}[\langle n-m\rangle_N] = \begin{cases} 1, & n = m \\ 0, & m \neq n \end{cases}$ $n, m \in Z_N$

$$\text{Then, } v = \begin{bmatrix} h_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}_{N^2\times 1} + \begin{bmatrix} 0 \\ h_1 \\ \vdots \\ 0 \end{bmatrix}_{N^2\times 1} + \ldots + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ h_{N-1} \end{bmatrix}_{N^2\times 1} = \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_{N-1} \end{bmatrix}_{N^2\times 1}$$

3.3.2 MATLAB Algorithm Development

The following section describes the Matlab algorithm development. Matlab is a powerful tool to simulate the ambiguity function and refine the algorithm. Its toolboxes and ease of plotting facilitate peforming a variety of tests as well as making quick adjustments to the algorithm implementation.

Figure 3–18 shows the simulation results for a 1024 sample Chirp pulse with zero padding. The sampling frecuency for Chirp signals was 500 Hz. The instantaneous frequency at time 0 was 0 Hz and the instantaneous frequency at 1 Seconds was 200 Hz. Hence, the chirp rate was 0.4 Hz. The ambiguity function was calculated and plotted assuming that the received signal had a delay of 520 samples in this case. At this point, pMatlab was used to parallelize the algorithm and move a step closer to FPGA algorithm implementation.

3.3.3 Simulink Algorithm Formulation

This section describes the Simulink algorithm formulation stage. After designing the ambiguity function algorithm in Matlab, the next step was to implement the algorithm in the Simulink environment. Simulink has a graphical block diagramming tool interface that allows developing models through multiple predefined blocks.

Figure 3–19 shows the Simulink algorithm implementation. The ambiguity Function subsystem shown contains the blocks nedeed to do the calculation, to send control signals to the scope, and to send the data produced to the to Workspace

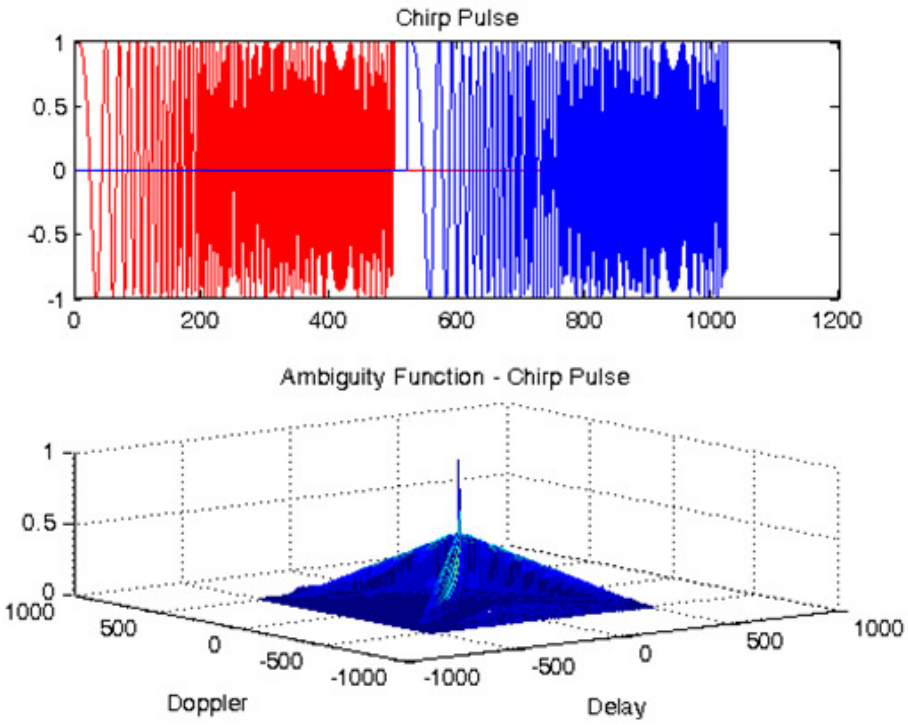


Figure 3–18: Ambiguity Function for a Chirp pulse

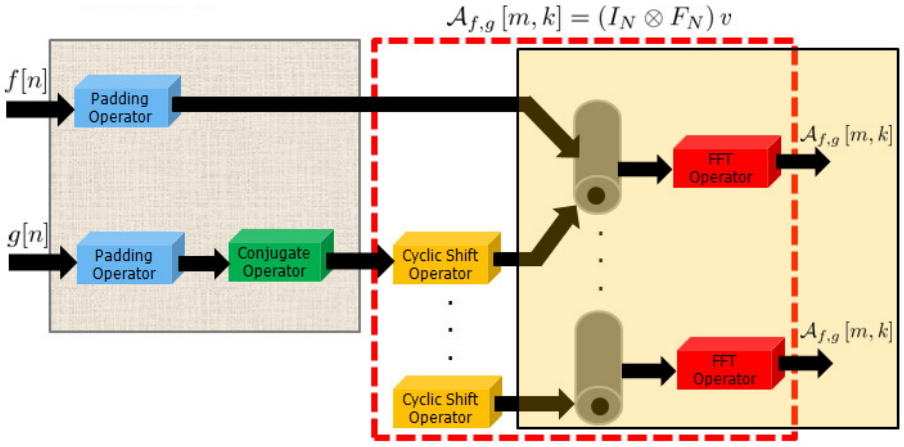


Figure 3–19: Implementation of Ambiguity Function in Simulink

subsystem (see Figure 3–20). This allows visualization of the data in Matlab for comparison with the original algorithm in section 3.3.2.

3.3.4 System Generator Stage

This section describes the System generator stage. The development system used was Xilinx. The Xilinx System Generator for DSP [11] is a powerful tool for

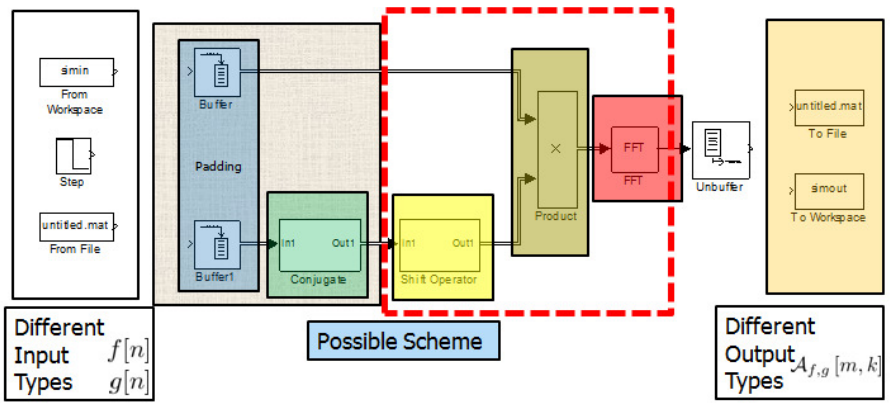


Figure 3–20: Possible Simulink Ambiguity Function Implementation Scheme

design, simulation, and hardware co-simulation of algorithm of FPGA algorithm. It does not replace VHDL programming, but it helps reduce design time. This tool adds new Simulink blocks that can be converted to VHDL code and downloaded to on FPGA, and direct hardware interation using Co-simulation, see Figure 3–20. In/Out Blocks allow communication with the exterior of the FPGA and Block RAM and FIFOs to manage data transmission between the hardware and the Simulink. Various pre-designed blocks are included and using them reduces the implementation effort and accelerates the testing process (Figure 3–21). Figure 3–22 shows what an implementation looks like System Generator development system.

3.3.5 FPGA Simulation/Emulation Stage

This section describes the FPGA simulation and emulation stage. When the design is ready to be tested, it can be simulated or emulated. In the simulation case (see Figure 3–23), Simulink interprets all components of Xilinx System Generator, and calculates latency and computation times for each simulated component. The results can be viewed in Simulink or exported directly to Matlab, or as .mat file, for subsequent analysis. In the case of emulation, or hardware co-simulation, System Generator creates a new component (see Figure 3–24) from the specific FPGA configuration.

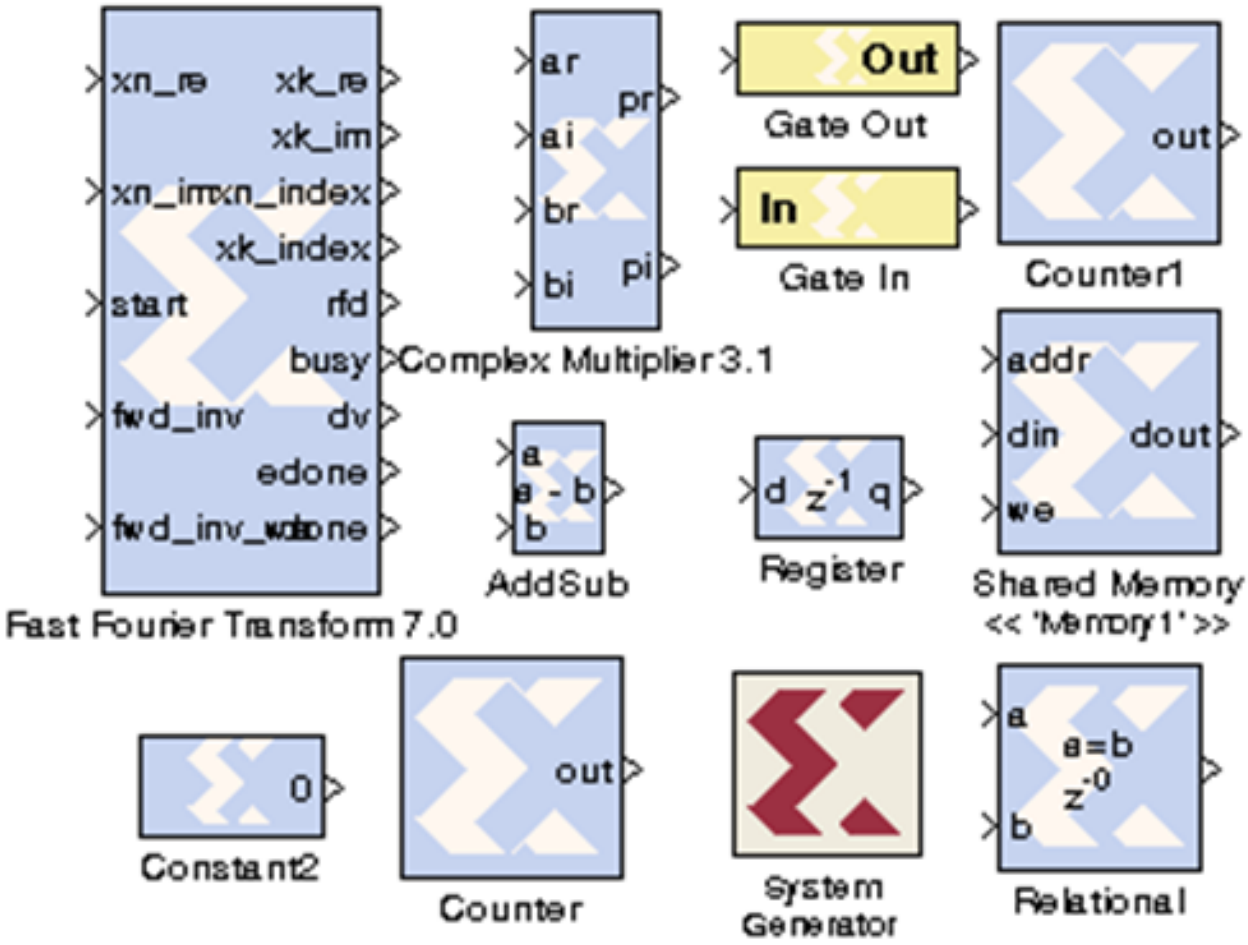


Figure 3–21: Some Blocks used in Ambiguity Function implementation in System Generator for DSP

Parameters include card type, communication type, speed and physical clock location, and any hardware information needed. The FPGA must be on and connected, so the new component can operate the binary program in the FPGA.

The advantage of System Generator is its ability to communicate and share data with Simulink and thereby Matlab. It can also save the results in text files making it more convenient to validate the same.

3.3.6 MATLAB Algorithm Validation

This section describes the Matlab algorithm validation stage. The algorithm can be validated in a timely manner. A matlab program was used to compare the

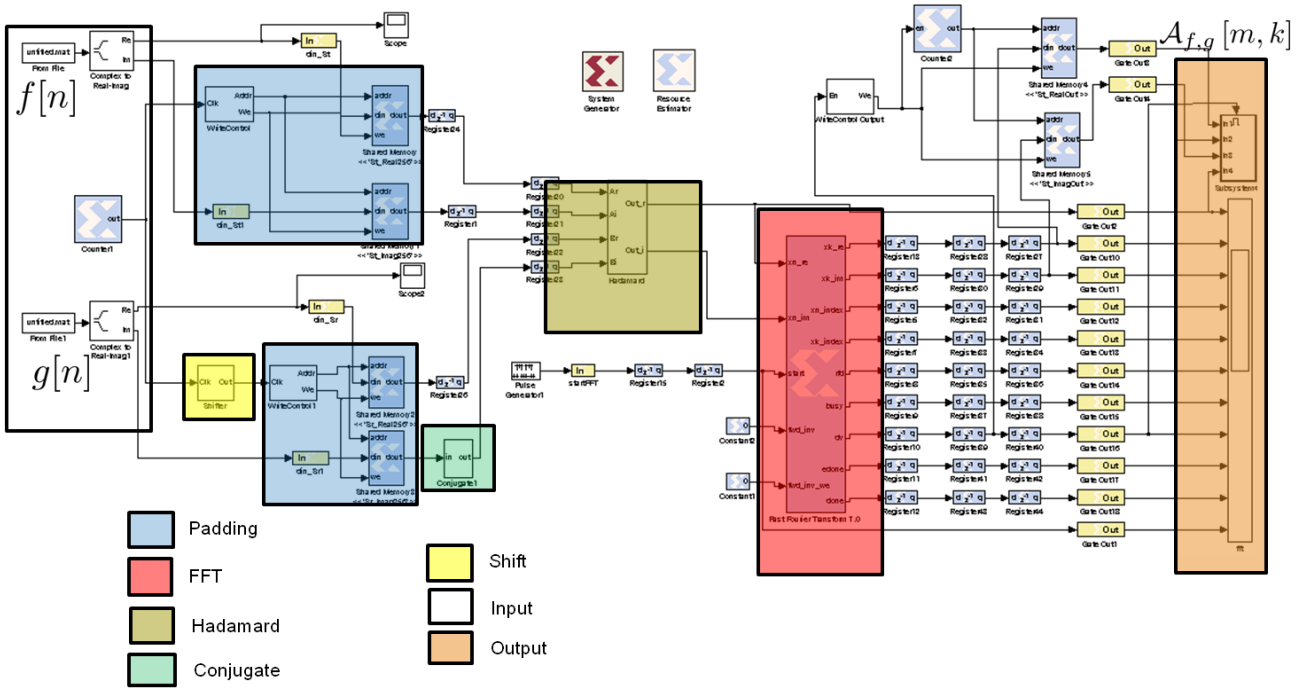


Figure 3–22: Appearance of an Implementation in System Generator

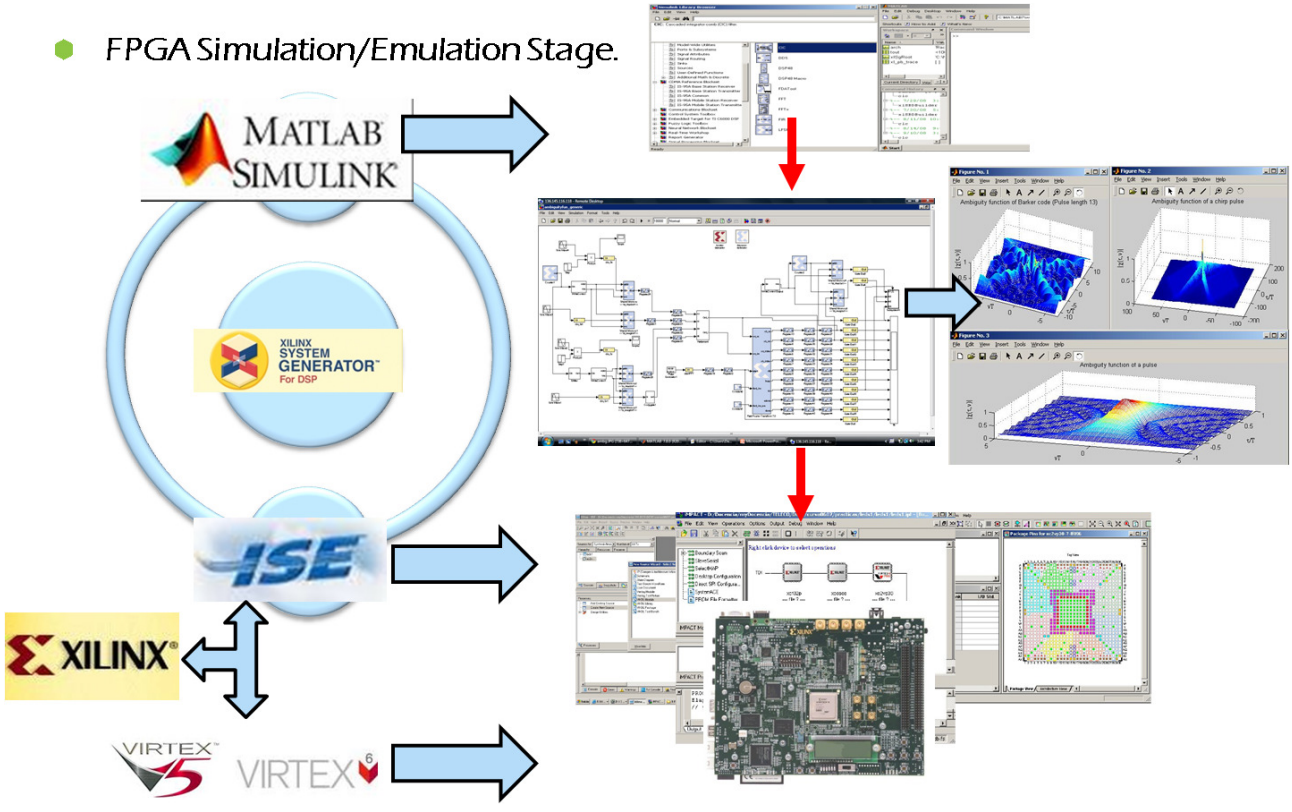


Figure 3–23: FPGA Simulation/Emulation Stage

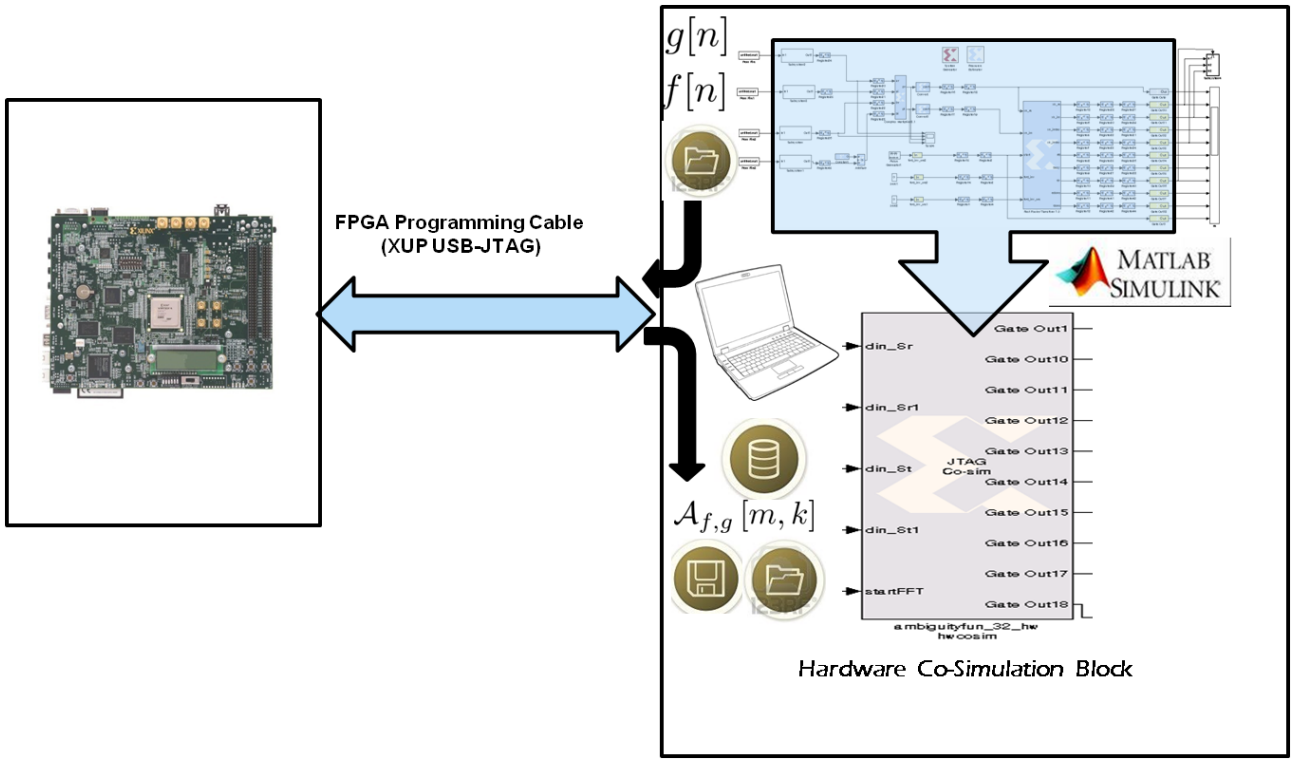


Figure 3–24: Hardware Co-Simulation

ambiguity function results presented in Section 3.3.2 with FPGA implementation. Figure 3–25 shows validations results through reconstruction of large scale signals shown in [12].

3.4 Ambiguity Function implementation in FPGA

The main purpose of this development is processing large scale signals. To this end, the FPGA implementation was done pipeline, trying to save as many slices and logic components as possible to use in future parallelizations. Shared memory was used to store data during processing. This allowed the implementation to use all available memory and compute the ambiguity function for input signals of size up to 16384 samples. Figure 3–26 shows the flow diagram for the ambiguity function implementation.

The FPGA's Ambiguity Function Implementation Architecture has a universal counter synchronizes the computation process. The f and g input signals are stored in shared

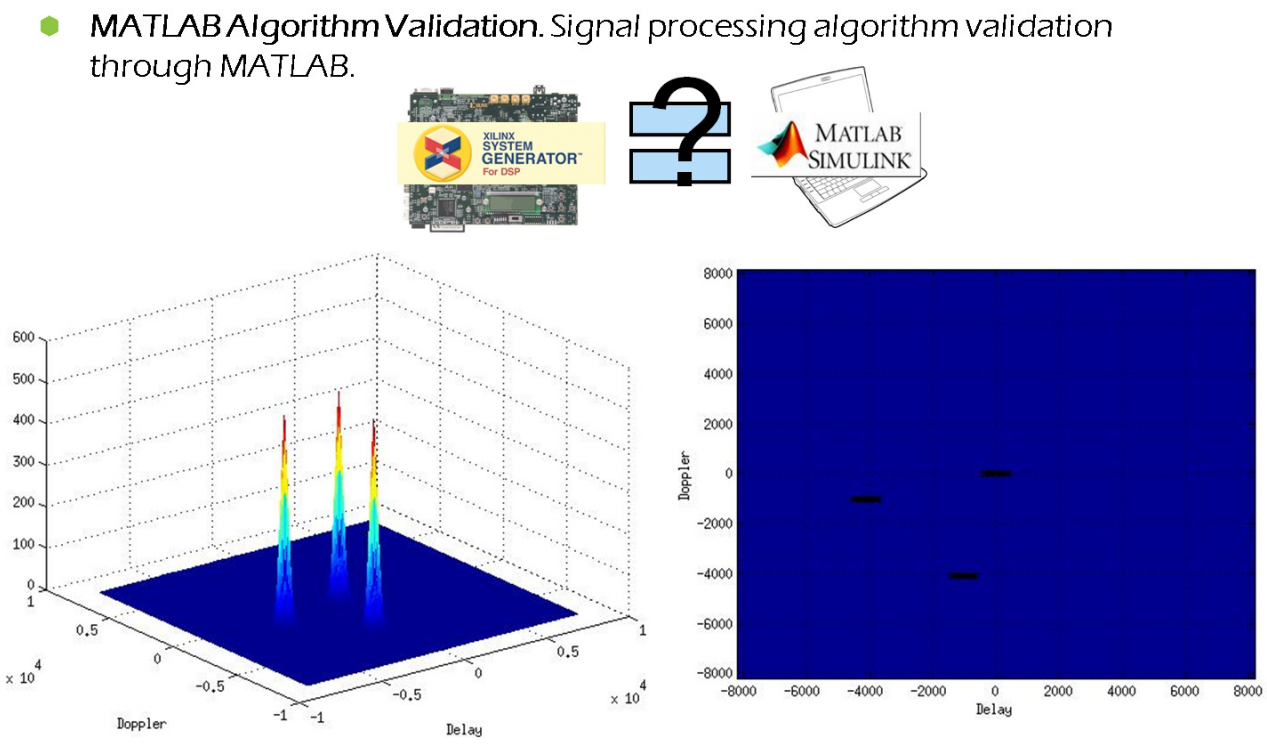


Figure 3–25: MATLAB Algorithm Validation

memories, where the size of each memory is N. The f and g signals are complex, thus four memory blocks are used: two for each signal where the real and imaginary part are stored separately. Since the f and g signal cannot change during the process, write control blocks were used to disable writing to this memory to prevent the loss of data. Two write control blocks are used one for each signal. Each block is responsible to address and write either the real or the imaginary part of the signal. A shifter block is included to do circular shift on the g signal. The shifter block is based in several counters in cascade that determine the memory address to read the signal with a shifted m, it block is connected to the write control. Since the complex conjugate must be computed for the g signal, a block is incorporated in the output share memory for the imaginary part of the signal. Then, Hadamard product is calculated in a block based on a complex multiplier. This result can be loaded into the Fast Fourier Transform block to produce an N-point Fourier transform. This

block is based in the Fast Fourier Transform V7.0 block of Xilinx System Generator [19]. This block has a comprehensive interface that allows full configuration for multiple applications. Finally, a memory of size N is used to store each column of the computed absolute value of the ambiguity function surface. The data is updated each time a new column is computed. This shared memory allows another device to take the data computed for visualization and analysis. The data is overwritten in the computation of each column. A universal counter is used to sinchronize all process as showed in the flow diagram in Figure 3–26.

In order to work with a large scale numeric sequence, its length should be greater than 212 after the zero padding procedure. To achieve this, the pipeline method was used integrating the shared memory to write the data during the process. The maximum length of the numeric sequence produced by this process was 214. The computation of the ambiguity function is an extension of the work presented by Rodriguez et al. in ([13], [9], [14], [15]), and by Rodriguez in ([16], [17], [18]).

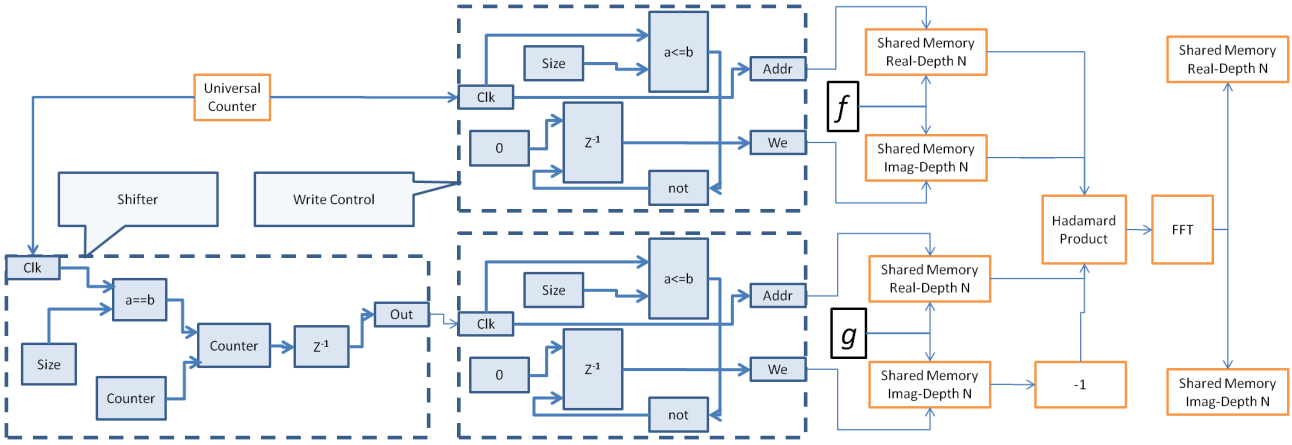


Figure 3–26: Ambiguity Function Diagram

4. HARDWARE/SOFTWARE ALGORITHM DEVELOPMENT RESULTS

In this work, pMATLAB was used to design and to develop an algorithm for computing the ambiguity function (AF) based on the FFT. The algorithm was implemented on FPGA. The high-level design tool Xilinx System Generator [11] was used, to take adventage of Matlab Simulink environment. This environment allows to send the results to the Matlab workspace. Simulink allows to use subsystems, upload VHDL and Matlab codes, and the Simulink block-type can be incorporated in larger deployments. The use of System Generator produced significant time savings in the programming and testing phase. With its building blocks such as adders, multipliers, and registration questions, System Generator can be used in complex designs, since it includes blocks that handle digital signal processing task such as FFT filtering. Taking advantage of the relationship between the ambiguity function and FFT, an algorithm to compute the ambiguity function was designed in pMatlab and implemented on an FPGA.

The ambiguity function algorithm was implemented and executed in both pMATLAB and FPGA to compare large scale signal computation times. The next section describe the experiments conducted and show the results of these test.

4.1 pMATLAB Implementation

The ambiguity function implemented in pMATLAB was done to understand the improvement in the computation using a parallel approach. Parallel implementation was simulated for 1,2,4,6 and 8 processors. Two complex signals with 1024, 2048, and 4096 samples of 32 bits for the real part and 32 bits for the imaginary parts

were used. Table 4–1 shows the average computational time in seconds for each simulation and the sums of the launch and computation times. The implementations were tested on a Intel Core 2 Duo CPU E8400 running at 3.00GHz with 3.25GBytes of RAM. MatLab 7.8.0.347 (R2009a) and pMatlab Parallel Matlab Toolbox v1.0.1 were used. The improvement in the computation times as the number of processors increases can be seen in Table 4–1. For example, the average time using 1 processor and a sample of length 4096 was 1.13 seconds, versus 0.18 seconds for 8 processor.

Table 4–1: pMatlab Ambiguity Function Times

Number of Cores	1024 points Computation Time (Seconds)	2048 points Computation Time (Seconds)	4096 points Computation Time (Seconds)
1	0.049813	0.31766	1.1349
2	0.035858	0.17826	0.5247
4	0.028979	0.10975	0.3046
6	0.028209	0.085887	0.2291
8	0.027989	0.072661	0.1844

4.2 FPGA Implementation

A Virtex 5 XUPV5-LX110T on the Xilinx ML505 evaluation platform board with a clock of 100 MHz was used for computing the ambiguity function. The Xilinx System Generator was used for implementation. Two complex signals with 256, 512, 1024, 2048, 4096, 8192, and 16384 samples of 16 bits for the real part and 16 bits for the imaginary part were used. Table 4–2 presents the computation and latency time for each sample length. Performance was faster than using pMATLAB. For example, the average computation time for the 1024 signal sample was 0.0102 second on FPGA in comparison to the 0.0279 seconds with pMATLAB. Similarly, the FPGA implementation was faster for the 2048 and 4096 signals samples than the 8 processor pMATLAB implementation. FPGA implementation computational times were 0.0420 and 0.168 seconds for 2048 and 4096 signals samples respectively versus 0.0726 and 0.184 seconds with the 8 cores pMATLAB implementation. Figure 4–1 shows the percentages of slices and Block RAM/FIFO of FPGA used in the computation. Note that the slices used for the computation do not change significantly

as the number of points increases. However the Block RAM/FIFO used appears to increase exponentially.

Table 4–2: FPGA Ambiguity Function Implementation Times

Number of Points	Latency(Seconds)	Computation Time(Seconds)
256	6.32E-06	6.62E-04
512	1.16E-05	2.63E-03
1024	2.19E-05	1.05E-02
2048	4.25E-05	4.20E-02
4096	8.35E-05	1.68E-01
8192	1.66E-04	6.71E-01
16384	3.29E-04	2.68E+00

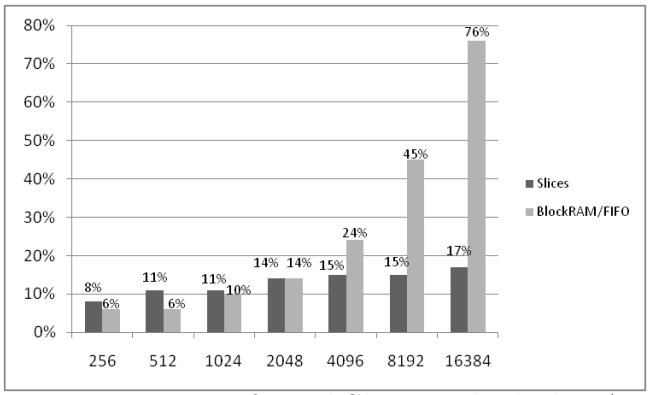


Figure 4–1: Percenteges of Used Slices and Block RAM/FIFO

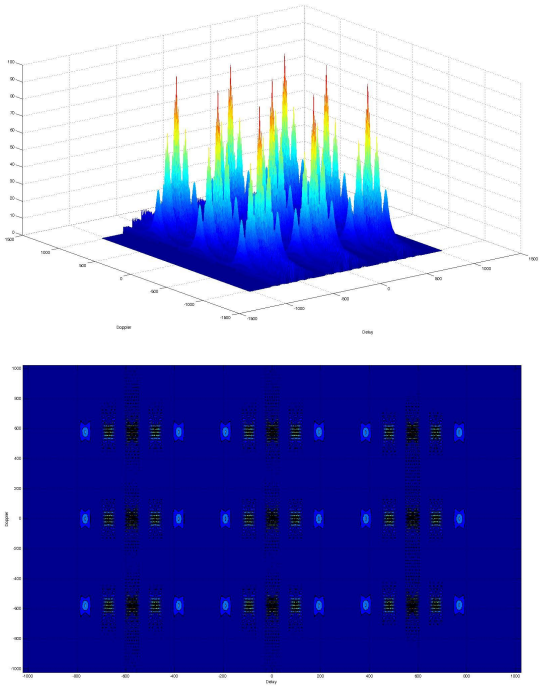


Figure 4–2: Multiple Pulses 1024 points Ambiguity Function

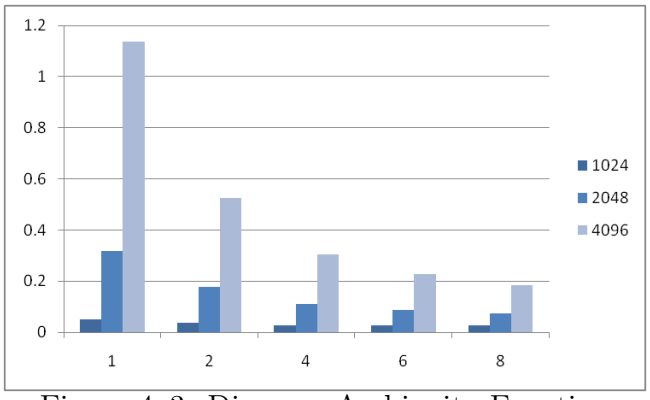


Figure 4–3: Diagram Ambiguity Function

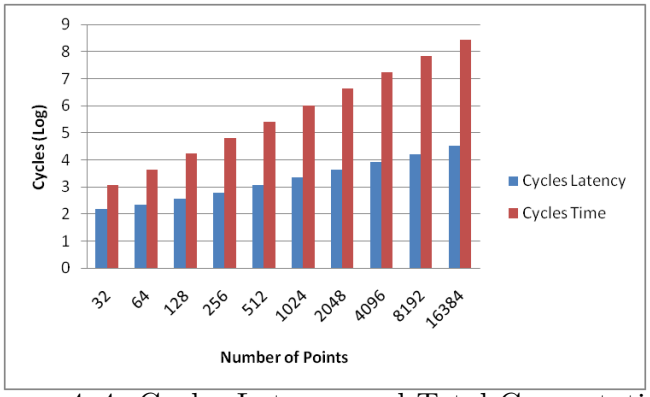


Figure 4–4: Cycles Latency and Total Computation

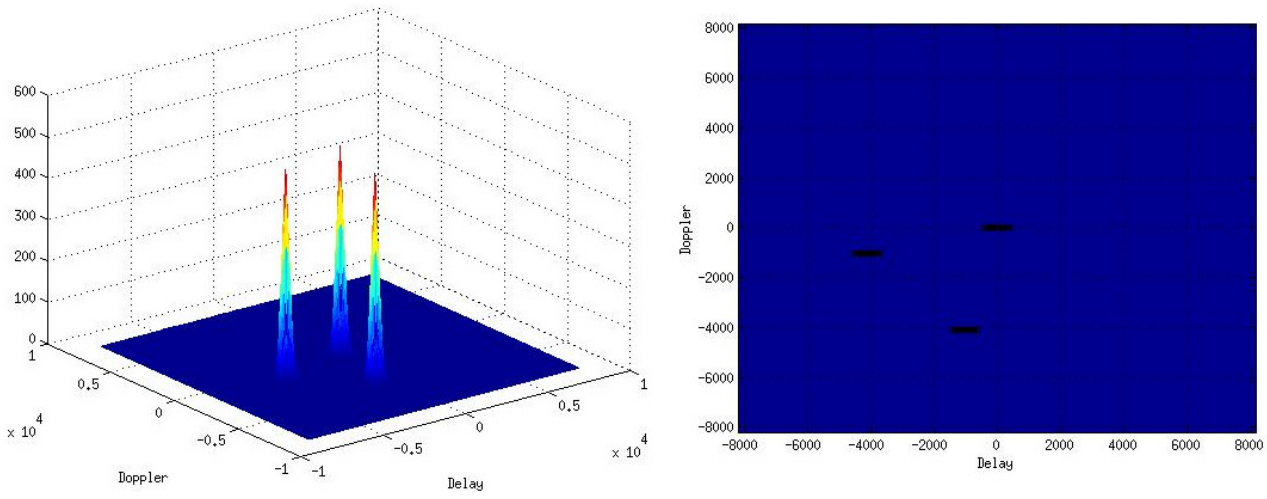


Figure 4–5: Diagram 3 target 16384 points Ambiguity Function

5. CONCLUSION AND FUTURE WORK

5.1 Conclusion

This work presents a process for analyzing, designing, implementing, and modifying certain class of signal processing algorithms using integrated hardware/software. This consists of five stages: algorithms:1) Signal processing algorithm development using the numeric computation software package Matlab; 2) Simulink formulation of signal processing algorithms; 3) System generator algorithms implementation; 4) Field Programmable Gate Array (FPGA) algorithm simulation and emulation; 5) Signal processing algorithm validation through Matlab.

An ambiguity function implementation on FPGA was described. Experiments with several signal lengths were carried out showing improved computation times when compared with pMATLAB implementation. As an algorithm design tool, pMATLAB is useful for examining each implementation stage in a parallel framework. The described implementation can be easily parallelized with either multiples cores or partitioning the data into the same core.

5.2 Future Work

The signal processing algorithm development and implementation approach presented could be used for FPGA implementation of other time-frequency distributions. In particular, the short-time Fourier transform for near real-time analysis of bioacoustic signals for environmental surveillance monitoring applications is of great interest.

APPENDICES

APPENDIXA. SOURCE CODES

A.1 Ambiguity Function Matlab

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Calcula: Ambiguity Function
%By David Marquez
%February 23, 2010  6:21:38.490 PM
%%%%%%%
%La funcion genera una seal transmitida St de "unos",
%y basado en un delay ingresado por el usuario genera
%la seal recibida Sr. Realiza zero padding a la proxima
%potencia de 2, teniendo encuenta la suma de las 2
%seales y el delay de Sr.
%%%%%%%
%Ejemplo:
%A=ambiguityfun(Length_St,Delay)
%A=ambiguityfun(4,3);
%La seal St seria igual a:
%St=[1 1 1 1]
%La seal Sr se crea con el Delay=3:
%Sr=[0 0 0 0 0 0 0 1 1 1 1]
%Realiza zero padding a 16:
%St=[1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0]
%Sr=[0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0]
```

```
%El resultado se almacena en A
function A=ambiguityfun(Length_St,Delay)
close all
%-------Genera las seales
    if nargin == 1
        Delay=3;
    end
    %St=ones(Length_St,1);
    %Si se quiere probar con otras seales especificas deben
    %debe cambiarse la linea anterior St por la seal que se
    %quiera generar, tambien se necesita Length_St y Delay
    %y el algoritmo funcionara de igual forma. Las seales
    %pueden ser complejas.
    %Ejemplo
    %Si Length_St=4 y Delay=3
    %Podemos usar la siguiente seal compleja:
    %
    %%%%   St=complex(ones(Length_St,1),5*ones(Length_St,1));
    %
    %Lo que nos daria como resultado:
    %St=[1+5i 1+5i 1+5i 1+5i]
    %La seal Sr se crea con el Delay=3:
    %Sr=[0 0 0 0 0 0 0 1+5i 1+5i 1+5i 1+5i]
    %Realiza zero padding a 16:
    %St=[1+5i 1+5i 1+5i 1+5i 0 0 0 0 0 0 0 0 0 0 0 0]
    %St=[0 0 0 0 0 0 0 1+5i 1+5i 1+5i 1+5i 0 0 0 0 0]
    %El resultado se almacena en A
```

```
    %
    %sine
    %St=sin(2*pi*2*[0:1/100:1])';
    %chirp
    St=chirp(0:1/500:1,0,1,200)';

    Sr = [zeros(Length_St + Delay - 1,1);St];
    Length_Sr=size(Sr,1);
    N = 2^nextpow2(Length_Sr);
    St = [St;zeros(N-Length_St,1)];
    Sr = [Sr;zeros(N-Length_Sr,1)];
%----------
%-------Calculo de Ambiguity Function
    A=zeros(N,N);
    for i=0:1:N-1
        Sr_Shift=circshift(Sr,N-i);
        A(:,i+1)=fftshift(fft(St.*conj(Sr_Shift)));
        %A(:,i+1)=(fft(St.*conj(Sr_Shift)));
        %pause(0.001)
    end
    subplot(2,1,1);
        %stem([0:(length(St)-1)],St,'--*r');
        plot([0:(length(St)-1)],St,'r');
        hold on
        %stem([0:(length(St)-1)],Sr_Shift)
        plot([0:(length(St)-1)],Sr_Shift)
        hold off
```

```
        title('Chirp Pulse')
        subplot(2,1,2)
        [X,Y] = meshgrid(-N/2:1:N/2-1);
        mesh(X,Y,abs(A)/max(max(A)));
        xlabel('Delay');
        ylabel('Doppler');
%         contourf(X,Y,abs(A));
%         grid on;
%         xlabel('Delay')
%         ylabel('Doppler');

        %mesh(abs((A))); %Para visualizar en 3D
        %imagesc([0:(length(St)-1)],[0:(length(St)-1)],(abs(A)));
        title('Ambiguity Function - Chirp Pulse')
```

A.2 Signal Generator

Spectrogram of a Sinusoidal Linear Chirp Signal

```
Fs=44100;
Ts=1/Fs;
tinc=Ts;
Tm=60*Ts;
m=1000;
V=m*Tm;
N=V*Fs;
t=0:tinc:V-Ts;
finc=Fs/N;
f=-(Fs/2):finc:(Fs/2)-finc;
```

```
M=1;
a=2*pi*4000;
b=1000;
c=0.25*pi;
%****************************************************************
%Signal Computation
%****************************************************************
x=M*cos((0.5*a)*(t.*t)+2*pi*b*t+c);
```

Star bottom routine in the Initial windows of GUI for the Signal Generator

```
global cc;
ccsboardinfo %board info
cc = ccsdsp('boardnum',0); %set up CCS object
reset(cc) %reset board
visible(cc,1); %for CCS window
enable(cc.rtdx); %enable RTDX
if ~isenabled(cc.rtdx)
error('RTDX is not enabled')
end
cc.rtdx.set('timeout', 20); %set 20sec time out for RTDX
open(cc,'signal_generator.pjt'); %open project
load(cc,'./debug/signal_generator.out'); %load executable file
run(cc); %run
configure(cc.rtdx,1024,4); %configure two RTDX channels
```

Stop bottom routine in the Initial windows of GUI for the Signal Generator

```
global cc;
if isrunning(cc), halt(cc); %if DSP running halt processor
end
```

```
disable(cc.rtdx); %disable RTDX
close(cc.rtdx,'ichan'); %close input channel
close(cc.rtdx,'ochan'); %close output channel
clear all
```

Calculate bottom routine in the Initial windows of GUI for the Signal Generator

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global cc;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
val_a=str2num(get(handles.edit5,'string'))*2*pi;
set(handles.text12,'string',num2str(val_a));
val_b=str2num(get(handles.edit6,'string'))*2*pi;
set(handles.text13,'string',num2str(val_b));
%Lectura de todos los valores
V = str2num(get(handles.edit8,'string'));
Fs = str2num(get(handles.edit9,'string'));
Ts=1/Fs;
tinc=Ts;
t=0:tinc:V-Ts;
N=V*Fs;
A = str2num(get(handles.edit1,'string'));
f = str2num(get(handles.edit2,'string'));
phi = str2num(get(handles.edit3,'string'));
% x1=A*cos(2*pi*f*t+phi);
%calcula se?al para uso en Matlab
x1=zeros(1,N);
for i=1:length(A)
x1=A(i)*cos(2*pi*f(i)*t+phi(i))+x1;
```

```
end
%coloca ceros a la amplitud para cancelar la se?al cuando no esta
%seleccionada
if get(handles.checkbox1,'value') == 0
A=0;x1=zeros(1,N);
end
M = str2num(get(handles.edit4,'string'));
fa = str2num(get(handles.edit5,'string'));
fb = str2num(get(handles.edit6,'string'));
c = str2num(get(handles.edit7,'string'));
m=2*Fs;
finc=Fs/N;
% f=-(Fs/2):finc:(Fs/2)-finc;
a=2*pi*fa;
b=2*pi*fb;
c=c*pi;
%calcula se?al para uso en Matlab
x2=M*cos((0.5*a)*(t.*t)+b*t+c);
%coloca ceros a la amplitud para cancelar la se?al cuando no esta
%seleccionada
if get(handles.checkbox2,'value') == 0
M=0;x2=zeros(1,N);
end
x=x1+x2;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%se acomodan los datos a enviar
envio=length([V Fs length(A) A f phi M fa fb c]);
```

```
%se rellena con zeros el vector para lograr su tama?o de 60
indata=[V Fs envio A f phi M fa fb c zeros(1,(60-envio))];
open(cc.rtdx,'ichan','w'); %open input channel
open(cc.rtdx,'ochan','r'); %open output channel
pause(3) %wait for RTDX channel to open
enable(cc.rtdx,'ichan'); %enable channel TO DSK
if isenabled(cc.rtdx,'ichan')
writemsg(cc.rtdx,'ichan', single(indata)) %send 16-bit data to DSK
pause(3)
else
error('Channel ''ichan'' is not enabled')
end
%*****************************************************************
%Plots
%*****************************************************************
if max(x)~=0
fx=fft(x);
sfx=fftshift(fx);
asfx=abs(sfx);
finc=Fs/N;
f1=-(Fs/2):finc:(Fs/2)-finc;
axes(handles.axes1);
plot(f1,asfx)
grid
xlabel('Frequency in Hertz')
ylabel('Magnitude')
title('Spectrum of Signal x(t)')
```

```
axes(handles.axes2);
[Y,F,T,P] = spectrogram(x,128,120,256,Fs);
%[Y,F,T,P] = spectrogram(,256,250,F,1E3,'yaxis');
% The following code produces the same result as calling
% spectrogram with no outputs:
surf(T,F,10*log10(abs(P)),'EdgeColor','none');
axis xy; axis tight; colormap(jet); view(0,90);
xlabel('Time');
ylabel('Frequency (Hz)');
sound(x,Fs);
%wavwrite(x,Fs,'chirp3.wav')
end
```

signal_generator.c:

```
#include "dsk6713_aic23.h"
Uint32 fs=DSK6713_AIC23_FREQ_44KHZ;
#include <rtdx.h> //RTDX support file
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#define PI 3.141592653589793
float buffer[60] = {0}; //init data from PC
float V,Fs,A,f,phi,salida,M,fa,fb,c;
float a,b;
float Ts;
int N,numdatos,numcos;
RTDX_CreateInputChannel(ichan); //data transfer PC-->DSK
RTDX_CreateOutputChannel(ochan); //data transfer DSK-->PC
```

```
/*
interrupt void c_int11() //ISR
{
if (!RTDX_channelBusy(&ichan))
RTDX_read(&ichan,buffer,sizeof(buffer));
return;
//return to calling function
}
*/
void main(void)
{
int i,j;
comm_poll();
IRQ_globalEnable();
IRQ_nmiEnable();
while(!RTDX_isInputEnabled(&ichan)) //for MATLAB to enable RTDX
puts("\n\n Waiting to read "); //while waiting
RTDX_read(&ichan,buffer,sizeof(buffer));//read data by DSK
puts("\n\n Read Completed");
//comm_intr();
while(1) {
if (!RTDX_channelBusy(&ichan)) {
RTDX_readNB(&ichan,buffer,sizeof(buffer));}
V = buffer[0];
Fs= buffer[1];
numdatos = buffer[2];
numcos=(numdatos-7)/3;
```

```
M = buffer[numdatos-4];
fa= buffer[numdatos-3];
fb= buffer[numdatos-2];
c = buffer[numdatos-1];
Fs=44100;
Ts=(1/Fs);
N=V*Fs;
a=2*PI*fa;
b=2*PI*fb;
c=c*PI;
for (i = 0; i < N; i++)
{
salida=M*cos((0.5*a)*((float)i/Fs*(float)i/Fs)+b*(float)i/Fs+c);
for (j = 0; j < numcos; j++)
{
A=buffer[3+j];f=buffer[3+numcos+j];phi=buffer[3+2*numcos+j];
salida=A*cos(2*PI*f*((float)i/Fs)+phi)+salida;
}
output_sample((short)(salida*5000));
};
} // infinite loop
}
```

A.3 Visualizer of the output from FPGA for the Ambiguity function

```
close all
clc
L = 128;
Fs =1;
```

```
NFFT = 2^nextpow2(L);
f = Fs/2*linspace(0,1,NFFT/2);
%%%%%%%%%
syncro = find(simout(:,2) == 0); % take the starting index of valid fft from xilix
out=(simout(:,1).^2+simout(:,3).^2)/L;
K=L;
cols=size(simout,1)/K;
A=zeros(K,cols);
for j=1:cols
    %if(j~=K/2+1)
        for i=1:L
            aux=simout((j-1)*K+i,1).^2+simout((j-1)*K+i,3).^2;
            A(i,j)=aux/K;
        end
    %end
    A(:,j)=fftshift(A(:,j));
end
surf(B)
```

APPENDIXB. HARDWARE COMPUTATIONAL STRUCTURES

B.1 Xilinx FPGAs Architectures

All Xilinx FPGAs contain the same basic resources:

- Slices grouped into Configurable Logic Blocks (CLBs). Contain combinatorial logic and register resources.
- IOBs. Interface between the FPGA and the outside world.
- Programmable interconnect.
- Other resources.

  Memory.

  Multipliers.

  Global clock buffers.

  Boundary scan logic.

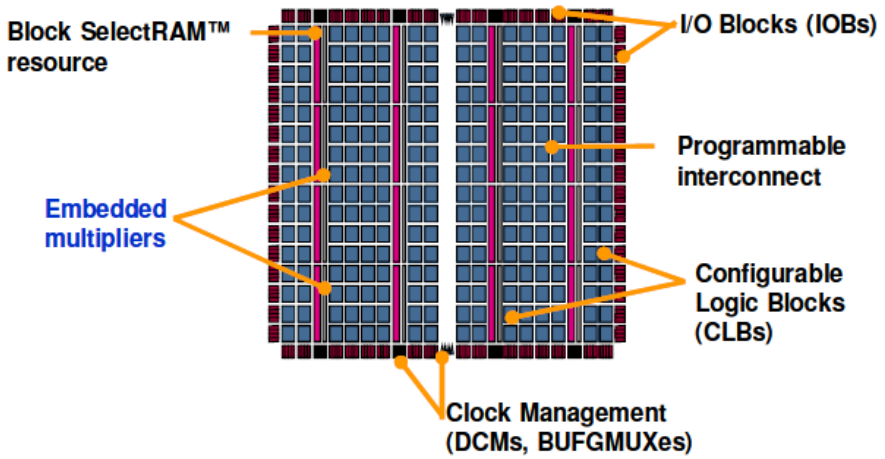


Figure B–1: Virtex II Architecture

B.1.1 CLB and Slices

Combinatorial and sequential logic implemented here. Each Virtex.-II CLB contains four slices.

- Local routing provides feedback between slices in the same CLB, and it provides routing to neighboring CLBs.
- A switch matrix provides access to general routing resources.

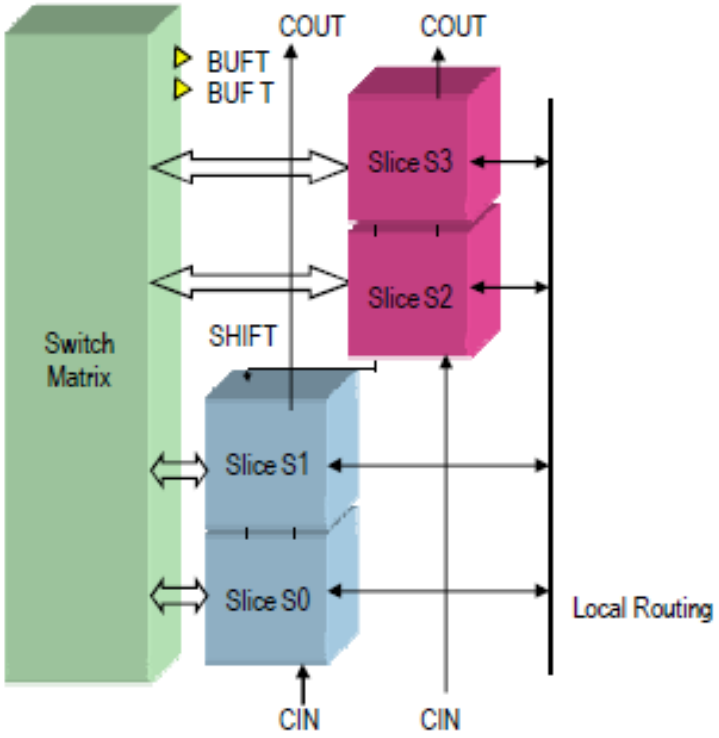


Figure B–2: CLB and Slices

B.1.2 Slice Resources

Each slice contains two:

- Four inputs lookup tables.
- 16-bit distributed SelectRAM.
- 16-bit shift register.

Each register:

- D flip-flop.
- Latch.

Dedicated logic:

- Muxes
- Arithmetic logic
- MULT_AND
- Carry Chain

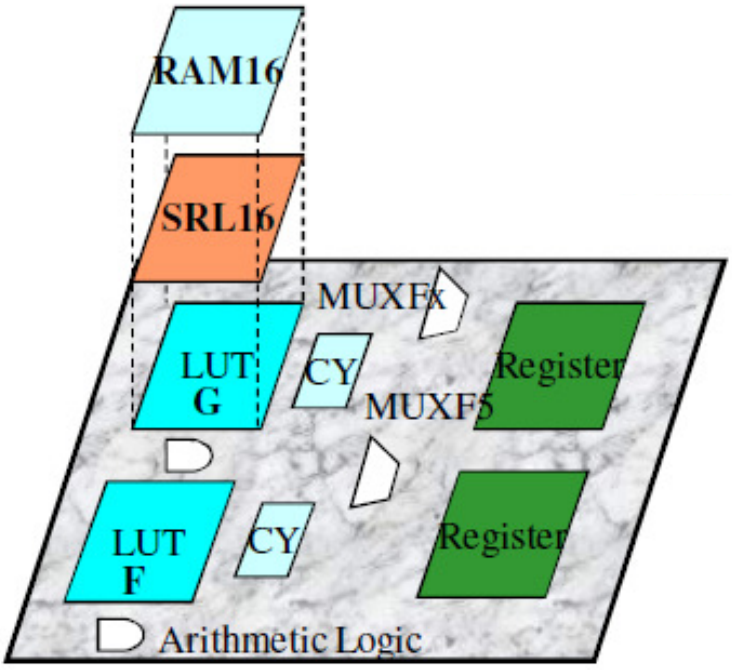
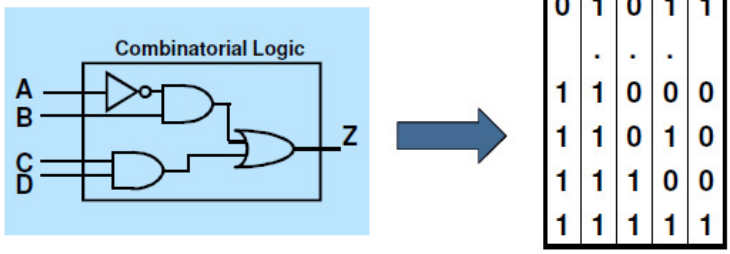


Figure B–3: Slice Resources

B.1.3 Look-Up Tables

Combinatorial logic is stored in Look-Up Tables (LUTs).

- Also called Function Generators (FGs).
- Capacity is limited by the number of inputs, not by the complexity.
- Delay through the LUT is constant.



A	B	C	D	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
	.	.	.	
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Figure B–4: Look-Up Tables

B.1.4 Distributed RAM

- LUTs used as memory inside the fabric.
- Flexible, can be used as RAM, ROM, or shift register.
- Distributed memory with fast access time.
- Cascadable with built-in CLB routing
- Applications

  Linear feedback shift register

  Distributed arithmetic

  Time-shared registers

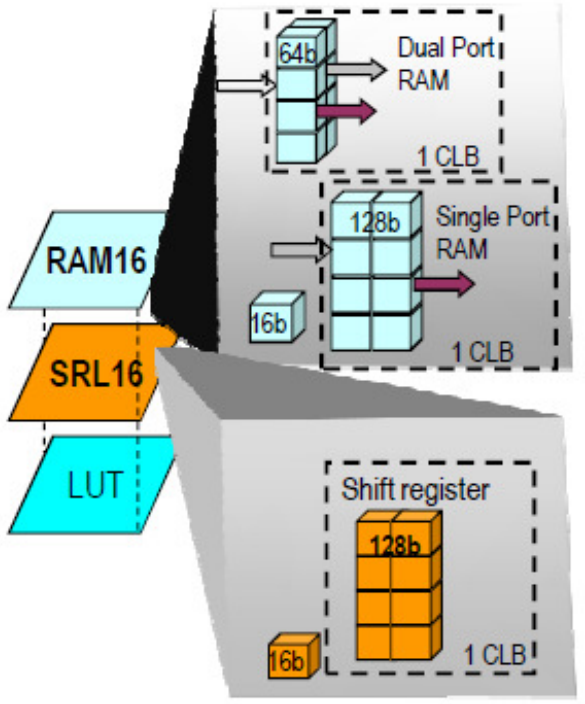


Figure B–5: Distributed RAM

REFERENCE LIST

[1] J.E. Gray. An interpretation of woodward's ambiguity function and its generalization. *Radar Conference, 2010 IEEE*, pages 859 –864, may. 2010.

[2] L. Auslander and R. Tolimieri. Radar ambiguity functions and group theory. *SIAM J. Math. Anal.*, 16:577 –601, May. 1985.

[3] Wei qiang Zhang, Ran Tao, and Yong feng Ma. Fast computation of the ambiguity function. *Signal Processing, 2004. Proceedings. ICSP '04. 2004 7th International Conference on*, 31 Aug.-4 Sept. 2004.

[4] Antonin Hermanek, Michal Kunes, and Michal Kvasnicka. Computation of long time cross ambiguity function using reconfigurable hw. In *Signal Processing and Information Technology, 2006 IEEE International Symposium on*, pages 879 – 883, 2006.

[5] H Nava. Modeling and simulation of point spread functions for advanced sar systems. In *University of Puerto Rico, Mayaguez*, 2004.

[6] Yu Teng, S. Doughty, K. Woodbridge, H. Griffiths, and C. Baker. Netted radar theory and experiments. In *Information, Decision and Control, 2007. IDC '07*, pages 23 –28, feb. 2007.

[7] A.B. Ramirez, I.J. Rivera, and D. Rodriguez. Sar image processing algorithms based on the ambiguity function. In *Circuits and Systems, 2005. 48th Midwest Symposium on*, pages 1430 – 1433 Vol. 2, aug. 2005.

[8] R. Tolimieri and S. Winograd. Computing the ambiguity surface. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-33(5):1239, 1985.

[9] J. Cruz E. Rodriguez, D. Seguel. Algebraic methods for the analysis and design of time-frequency signal processing algorithms. *Circuits and Systems, 1993.,*

ISCAS '93, 1993 IEEE International Symposium on, pages 195 –199, may. 1993.

[10] N.T. Bliss, J. Kepner, H. Kim, and A. Reuther. pmatlab: Parallel matlab library for signal processing applications. *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, 4:IV–1189 –IV–1192, apr. 2007.

[11] Xilinx. System generator for dsp user guide. 2010.

[12] D. Marquez, J. Valera, A. Camelo, C. Aceros, M. Jimenez, and D. Rodriguez. Implementations of cyclic cross-ambiguity functions in fpgas for large scale signals. In *Circuits and Systems (LASCAS), 2011 IEEE Second Latin American Symposium on*, pages 1 –4, feb. 2011.

[13] Xilinx. Fast fourier transform v 7.0. 2009.

[14] Jeremy Johnson, Robert Johnson, Domingo Rodriguez, and Richard Tolimieri. A methodology for designing, modifying, and implementing fourier transform algorithms on various architectures. *In Proc. of Circuits, Systems, and Signal Processing*, 9(4):449–500, 1990.

[15] Domingo Rodriguez, Marlene Vargas Solleiro, and Yvonne Aviles. Dft beamforming algorithms for space-time-frequency applications. *Digital Wireless Communication II*, 4045(1):86–93, 2000.

[16] Ana B. Ramirez and Domingo Rodriguez. Automated hardware-in-the-loop modeling and simulation in active sensor imaging using t16713 DSP units. *2006. MWSCAS '06. 49th IEEE International Midwest Symposium on Circuits and Systems*, 2:300–304, August 2006.

[17] Domingo Rodriguez. SAR point spread signals and earth surface property characteristics. *Part of the SPIE Conference on Subsurface Sensors and Applications, Denver, Colorado*, 3752:292–306, July, 1999.

[18] D. Rodriguez. A computational kronecker-core array algebra SAR raw data-generation modeling system. 1:116–120, 2001.

[19] Domingo Rodriguez. Tensor product algebra as a tool for VLSI implementation of the discrete fourier transform. *Proc. ICASSP, Toronto, ON, Canada*, pages 1025 – 1028, 1991.