VOT79029

SMARTANTENNA DESIGN

(REKA BENTUK ANTENNA PINTAR)

THAREK ABD.RAHMAN
REZA ABDOLEE

RESEARCH VOTE NO:
79029

FAKULTI KEJURUTERAAN ELEKRIK
UNIVERSITI TEKNOLOGI MALAYSIA

2008

The lord has blessed me with a family
to whom this research project report is dedicated.

ACKNOWLEDGEMENTS

This project is sponsored by Ministry of Science, Technology and Innovation of Malaysia under the Escience Fund with VOT number 79029.

ABSTRACT

SMART ANTENNA DESITGN

(Keywords: Smart antennas, digital beamforming, DSP and FPGA implementation)

Smart antenna technologies are emerging as an innovative way to meet the growing demand for more powerful, cost-effective and highly efficient wireless communication systems. In this project, from broad category of smart antenna techniques, the switch beam digital-beamforming technique in the downlink is deployed to improve the fidelity and performance of WiMax application. In this regards, the designed system forms and steer the beam according to the user location which is known to the system. In addition, the system performs sidelobe cancellation base on the chebyshev algorithm to optimize the antenna radiation pattern. The design and implementation steps are as follow: the system is firstly modeled by MATLAB software. After modeling, the algorithm is implemented in DSP by using C and Code Composer Studio. After DSP hardware implementation, the signal management is performed in DSP before transmission to the FPGA board. This management is necessary, in order to make processed signal in DSP suitable for channel separation process in FPGA. FPGA is deployed to split the data stream into sixteen channels corresponding to number of antenna elements. Next, the FPGA and DSP are integrated together to form the baseband switch beam smart antenna system. After integration process, the hardware is tested; the results prove that the system functions properly as we expected from simulation model. In this project, lastly, the initial design of IF, RF-front-end and their necessary circuits are also portrayed to be used in the next smart antenna research project.

Key researchers:

Prof.Tharek Abd Rahman
Dr.Razali Ngah
Reza Abdolee
Vida Vakilian
Email: tharek@fke.utm.my
Tel.No: 07-5536601
Vote No: 79029

Reka Bentuk Antenna Pintar

Teknologi antena pintar telah muncul sebagai satu inovasi untuk memenuhi permintaan sistem yang berkuasa tinggi, kos berpatutan dan berkecekapan tinggi dalam system perhubungan tanpa wayar. Bagi projek ini, dari pelbagai kategori dalam teknik antenna pintar, teknik suis alur berdigital rangkaian bawah telah digunakan untuk memperbaiki kualiti dan mutu dalam penggunaan WiMax. Dengan itu, sistem direkabentuk bagi membentuk alur dan dipandu ke kedudukan pengguna yang telah diketahui oleh sistem. Sebagai tambahan untuk mendapatkan bentuk sinaran antena yang optima, pembatalan cuping sisi dilakukan berdasarkan kepada algoritma chebyshev. Langkah-langkah bagi merekabentuk dan perlaksanaan projek ini adalah seperti berikut: Pada mulanya, perisian MATLAB digunakan untuk mendapatkan model bagi sistem tersebut dan seterusnya algoritma dilakukankan dalam DSP menggunakan bahasa C and 'Code Composer Studio'. Setelah perkakasan DSP dilaksanakan, adalah perlu memastikan pengurusan isyarat dibuat sebelum signal ini dihantar ke papan FPGA. Ini adalah perlu untuk membolehkan data yang sesuai sahaja yang akan di hantar ke papan FPGA tersebut. Setelah itu, saluran perlu dipisahkan kepada enam belas unsur tatasusunan antenna menggunakan papan FPGA. Berikutnya adalah menyatukan FPGA dengan DSP bersama-sama untuk menghasilkan jalur asas alur suis sistem antenna pintar. Perkakasan hasil dari penyatuan diatas telah diuji dan keputusan menunjukkan sistem telah berfungsi dengan baik seperti yang dijangkan dari penyelakuan model. Diakhir projek ini, rekabentuk awal bagi IF, RF 'front-end' dan litar yang bersesuaian telah diberikan bagi tujuan untuk penggunaan penyelidikan antenna pintar di masa hadapan.

Key researchers:

Prof.Tharek Abd Rahman
Dr.Razali Ngah
Reza Abdolee
Vida Vakilian
Email: tharek@fke.utm.my
Tel.No: 07-5536601
Vote No: 79029

Table of Contents

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATION

Symbol	**Definition**
WLAN	wireless local area networks
WiMax	Worldwide Interoperability for Microwave Access
DSP	Digital signal processor
FPGA	Field-programmable gate array
FS	Frame synchronization
SDRAM	synchronous DRAM
EDMA	Enhanced direct memory access
McBSP	Multi channel buffer serial port
PLD	programmable logic device
CPLD	Complex programmable logic device
DIP	dual in-line package
LED	Light-emitting diode

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

I. Introduction

The demand of smart antenna for mobile communications is increased recently and the main purpose for applying smart antennas is feasibility for increasing in capacity and efficiency. The application of smart-antenna arrays has been suggested for mobile-communication systems, to overcome the problem of limited channel bandwidth, satisfying a growing demand for a large number of mobiles on communications channels. Smart antennas, when used appropriately, help in improving the system performance by increasing channel capacity and spectrum efficiency, extending range coverage, steering multiple beams to track many mobiles, and compensating electronically for aperture distortion. They also reduce delay spread, multipath fading, co-channel interference, system complexity, bit error rate (BER).

1.2 Objectives

Base on which we have stated in submitted proposal to Ministry of Science, the main objective of the project is to design and fabricate high gain directive antenna with beamforming capability. The methodology to achieve this objective was to use left-handed material to implement the antenna. Unfortunately, after doing some research, we have found out that this methodology is impractical in the current situation. The reasons are outline here. Firstly, these materials are very expensive to build due to expensive manufacturing devices. Second, these areas of research presently are academic and they have a wide room to find the place for practical implementation, so the output of this kind of research would just be a laboratory scale. Therefore, we had preferred to design this antenna with different methodology. So we have chosen the signal processing technique instead of left-handed material to form and optimize the antenna radiation pattern. To achieve this objective we define three steps. The first step is implementation of the digital beamforming by using the digital signal processor. More precisely, in the first step the aim is to implement the switched beam smart antenna for downlink transmission. According to the algorithm, the beam can steer from 0 to 180 degree in azimuth angle base on user direction with any resolution. Second step of the project is to manage the processed signals in DSP board after digital beamforming and sidelobe cancellation to transmit them to the expansion board. Third and main step of the project is to design and implement of baseband channel separation and synchronization by using FPGA board. By completing these steps, the main objective of the project will be achieved.

1.3 Scope of research project

The project involves both of software modeling and hardware implementation. In the first phase of the project, the TMS320C6713B DSP board is used for beamforming. C and Code Composer Studio software is applied for programming this board. Also MATLAB software is chosen for modeling because of some facilities provided, including a Link for Code Composer Studio Development Tools and signal processing blocksets and toolbox. By using this links, transferring information to and from Code Composer Studio is possible. In the second phase of the project FPGA board is applied for performing the channel separation and synchronization and Quartus II software is used to program this board. In the third phase of the project integration of DSP and FPGA is done by programming the EDMA and McBSP of DSP.

1.4 Report outline

This report is organized as follows. In Chapter 2, background information and basic principle in smart antenna system is explained. In addition, the project structure and block diagram are discusses as well. In Chapter 3, digital beamforming by using DSP board is fundamentally discussed. Moreover, the hardware structure of DSP board shortly reviewed. Also, the model for beamforming is illustrated. In Chapter 4, after FPGA hardware description, channel separation for the project is explained. In this respect, FPGA programming and pin assignment are reviewed as well. In Chapter 5, integration of DSP and FPGA is discussed and also the system model used in this project is given. The IF and RF-front-end design are reviewed in chapter 6. The antenna array design and implementation are presented in chapter 7. Lastly, in Chapter 8, simulation results for digital beamforming and the channel separation are discussed. In this chapter a comparison between hardware and software simulation results is made between DSP and MATLAB software. At the end of this chapter, final conclusion of the work is presented, and some possible future works are suggested.

CHAPTER 2

SMART ANTENNA SYSTEMS

2.1 Introduction

The receiver and transmitter antennas are one of the most critical components in the design of wireless communication systems. A good design of the antenna can relax system requirements, improve overall system performance and greatly reduce the infrastructure costs [1]. It has been demonstrated that using a beamforming antenna instead of an omni-directional antenna in the wireless communication systems can increase the system capacity and improve the overall system performance [2]. This performance enhancement is due to the reduction in the interference by attenuating the interference signals which have different directions of arrivals than the desired signal direction of arrival at the receiver antenna site. This is called spatial processing because the direction of arrival is related to the mobile location.

The system performance can be further improved by exploiting the delay spread of the received signals. The signal of each mobile arrives to the base station antenna in multi-path form. Each path usually has its own delay and direction of arrival. Using the smart antenna alone means that we receive (ideally) only one path and ignore the others.

2.2 Smart Antenna Definition

A smart antenna is defined as an array of antennas with a digital signal processing unit that can change its pattern dynamically to adjust to noise, interference and multipaths. The conceptual block diagram of a smart antenna system is shown in Figure 2.1. The following three main blocks can be identified: (i) array antenna (ii) complex weights and (iii) adaptive signal processor. The array antenna comprises of a Uniform Linear Array (ULA) or Uniform Circular Array (UCA) of antenna elements [3]. The individual antenna elements are assumed to be identical, with omni-directional patterns in the azimuth plane. The signals received at the different antenna elements are multiplied with the complex weights and then summed up. The complex weights are continuously adjusted by the adaptive signal processor which uses all available information such as pilot or training sequences or knowledge of the properties of the signal to calculate the weights. Such a configuration dramatically enhances the capacity of a wireless link through a combination of diversity gain, array gain, and interference suppression. Increased capacity translates to higher data rates for a given number of users or more users for a given data rate per user. This is done so that the main beam tracks the desired user and/or nulls are placed in the direction of interferers and/or side lobes towards other users are minimized. It should be noted that the term “smart” refers to the whole antenna system and not just the array antenna alone.

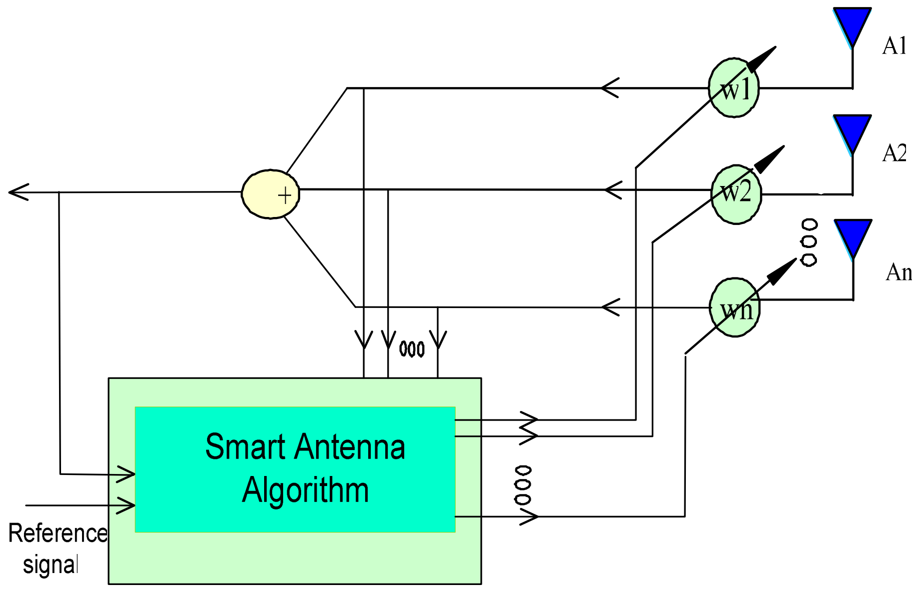


Figure-(2.1) Block Diagram of a Smart Antenna System

2.3 Smart Antenna Operation

The smart antenna works as follows; assume that there is a user sending a signal to the base station. Then each element of smart antenna array in the base station will receive the signal but at different time instance since the distance between the user and each element of array is different from other elements. By using this time delay and the distance between antenna elements the location of the user can be calculated. Therefore, the transmitter can send a signal to the exact location of that user. This strategy can be applied for the system with multiple users as well. A smart antenna receiver can suppress the interference by using this strategy. The smart antenna is able to process the signals received by the array or transmitted by the array using suitable array algorithms to improve wireless system performance. An antenna array consists of a set of distributed antenna elements (dipoles, monopoles or directional antenna elements) arranged in certain geometry (e.g., linear, circular or rectangular grid) where the spacing between the elements can vary. The signals collected by individual elements are coherently combined in a manner that increases the desired signal strength and reduces the interference from other signals. Hence a smart antenna can be viewed as a combination of “regular or conventional” antenna elements whose transmit or received signals are processed using “smart” algorithms.

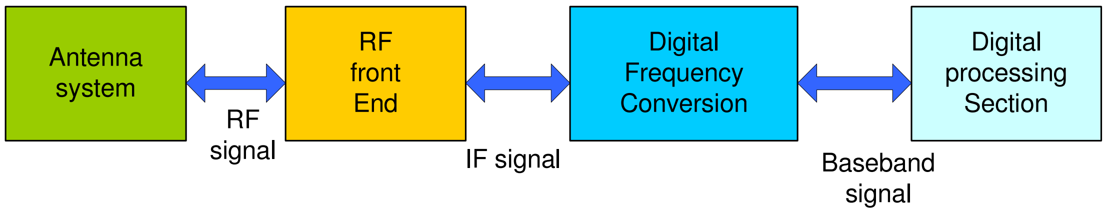


Figure-(2.2) Block diagram of smart antenna implementation

Figure-(2.2) shows a generic implementation of smart antenna system. As shown in this figure, the antenna arrays have input or output as RF signals in the analog domain. These signals are passed to/from the Radio Frequency (RF) analog front end which usually consists of low noise amplifiers, mixers and analog filters. In the receive mode, the RF signals are converted to digital domain by analog to digital converters (ADCs) and in transmit mode, the baseband digital signals are converted to RF using digital to analog converters

(DACs). The down conversion from RF to baseband or up conversion from baseband to RF can involve the use of IF signals. The baseband signals received from each antenna is then combined using the “smart” algorithms in a digital processing section. Each antenna element hence has a RF chain going from the antenna element to RF front end to digital conversion for receiver and vice-versa for transmitter. The digital processing section can be implemented on a microprocessor or a DSP or FPGA. Hence the “smart” algorithm implementation usually is a software code unless implemented in an ASIC or FPGA.

2.4 Classification of Smart Antenna

The fundamental idea behind a smart antenna is not new but dates back to the early sixties when it was first proposed for electronic warfare as a counter measure to jamming [4]. Until recently, cost barriers have prevented the use of smart antennas in commercial systems. Thus in existing wireless communication systems, the base station antennas are either omni-directional which radiate and receive equally well in all azimuth directions, or sector antennas which cover slices of 60 or 90 or 120 degrees [4]. However, the advanced of low cost Digital Signal Processors (DSPs), Application Specific Integrated Circuits (ASICs) and innovative signal processing algorithms have made smart antenna systems practical for commercial use [5]. The smart antenna systems for cellular base stations can be divided into two main categories. These are (i) switched beam system and (ii) adaptive arrays systems. Smart antennas are a solution to capacity and interference problems [4]-[6]. This technology is often described as dynamic sectorization, (i.e. the cells are Sectorized to reduce interference levels but in a way to enhance the capacity of the cell) or as an adaptive antenna. In either case, most smart antennas form narrow beams directed to each particular user in order to enhance the received signal strength (RSS) and/or signal-to-noise ratio (SNR). Smart antennas can be classified into two types i.e. Switched Beam Systems and Adaptive Array Systems.

2.4.1 Switched Beam Systems

A switched beam antenna system consists of several highly directive, fixed, pre-defined beams which can be formed by means of a beamforming network [7] e.g., the Butler 1.2. Smart Antennas for CDMA Cellular System [8, 9] which consists of power splitters and fixed phase shifters. The system detects the signal strength and chooses one beam, from a set of several beams that gives the maximum received power. A switched beam antenna can be thought of as an extension of the conventional sector antenna in that it divides a sector into several micro-sectors [7]. It is the simplest technique and easiest to retro-fit to existing wireless technologies. However switched beam antenna systems are effective only in low to moderate co-channel interfering environments owing to their lack of ability to distinguish a desired user from an interferer, e.g. if a strong interfering signal is at the center of the selected beam and the desired user is away from the center of the selected beam, the interfering signal can be enhanced far more than the desired signal with poor quality of service to the intended user [7].

2.4.2 Adaptive Array System

In an adaptive array, signals received by each antenna are weighted and combined using complex weights (magnitude and phase) in order to maximize a particular performance criterion e.g. the Signal to Interference plus Noise Ratio (SINR) or the Signal to Noise Ratio (SNR). Fully adaptive system use advanced signal processing algorithms to locate and track the desired and interfering signals to dynamically minimize interference and maximize intended signal reception [10]. The main difference between a phased array and an adaptive array system is that the former uses beam steering only, while the latter uses beam steering and nulling. For a given number of antennas, adaptive arrays can provide greater range (received signal gain) or require fewer antennas to achieve a given range [11]. However the receiver complexity and associated hardware increases the implementation costs.

Through beamforming, a smart antenna algorithm can receive predominantly from a desired direction (direction of the desired source) compared to some undesired directions (direction of interfering sources). This implies that the digital processing has the ability to shape the radiation pattern for both reception and transmission [12] and to adaptively steer beams in the direction of the desired signals and put nulls in the direction of the interfering signals. This enables low co-channel interference and large antenna gain to the desired signal.

Beamforming systems can be implemented in two ways; fixed beamforming systems or fully adaptive systems. A fixed beamforming system has a beamforming network (BFN) followed by RF switches which operate in the RF/analog domain. The switches are controlled by a control logic which selects a particular beam. Here the processing required is minimal as the control logic has to choose one of the predetermined set of weights to select a beam. In adaptive beamforming, the antenna gains or weights are chosen adaptively through running array algorithms in the digital domain.

2.5 Advantages of Smart Antennas

Primarily smart antennas were used at base stations in a cellular network to improve user capacity. Capacity here refers to the number of subscribers that can be simultaneously serviced in a system. Usage of omnidirectional antennas causes co-channel interference when two users use the same band of frequency that eventually limits the user capacity in a system. Since smart antennas can focus their beams towards desired user reducing interference to other users using the same frequency band, the user capacity in a system can be improved using spatial division multiple access (SDMA). Figure-(2.3) shows this advantage of SDMA compared to the omnidirectional case, which can reduce co-channel interference using beamforming.

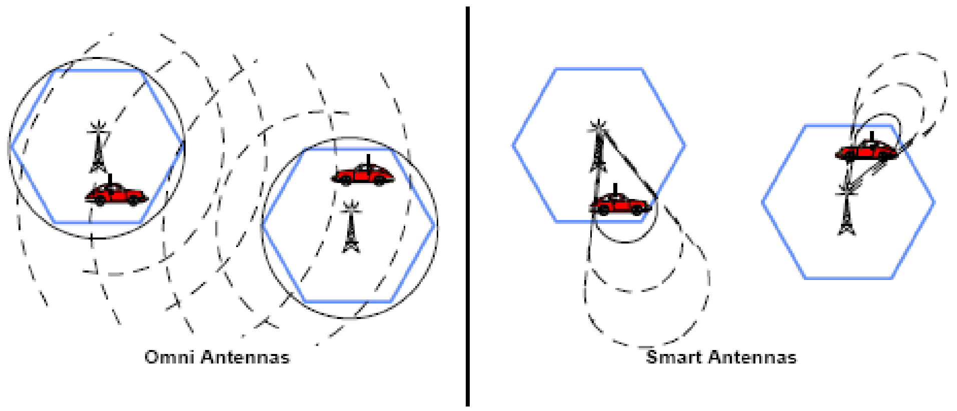


Figure-(2.3) Omnidirectional and smart antennas based cellular system

Other advantages as seen from various types of smart antennas studied include robustness against multipath fading and co-channel interference which improves reliability of received signal; reduced power consumption for handsets; low probability of interception and detection; enhanced location estimates and enhanced range of reception. Because there are obstacles and reflectors in the wireless propagation channel, the transmitted signal arrivals at the receiver from various directions over a multiplicity of paths. Such a phenomenon is called multipath. It is an unpredictable set of reflections and/or direct waves each with its own degree of attenuation and delay. Recent studies on use of smart antennas in mobile terminals have also shown to improve network capacity in ad-hoc networks.

Smart antenna systems can improve link quality by combating the effects of multipath propagation or constructively exploiting the different paths, and increase capacity by mitigating interference and allowing transmission of different data streams from different antennas. More specifically, the benefits of smart antennas can be summarized as follows [16]. Some of the advantages of the smart antenna are as follows:

(i) Increased range/coverage

The *array* or *beamforming gain* is the average increase in signal power at the receiver due to a coherent combination of the signals received at all antenna elements. It is proportional to the number of receive antennas and also allows for lower battery life.

(ii) Lower power requirements and/or cost reduction

Optimizing transmission toward the wanted user (transmit beamforming gain) achieves lower power consumption and amplifier costs.

(iii) Improved link quality/reliability

Diversity gain is obtained by receiving independent replicas of the signal through independently fading signal components. Based on the fact that it is highly probable that at least one or more of these signal components will not be in a deep fade, the availability of multiple independent dimensions reduces the effective fluctuations of the signal.

(iv) Increased spectral efficiency

Precise control of the transmitted and received power and exploitation of the knowledge of training sequence and/or other properties of the received signal (e.g., constant envelope, finite alphabet, cyclostationarity) allows for *interference reduction/ mitigation* and increased numbers of users sharing the same available resources (e.g., time, frequency, codes) and/or reuse of these resources by users served by the same base station/ access point.

2.6 Disadvantages of Smart Antennas

One of the major existing disadvantages of smart antennas is in their design and implementation in hardware. Multiple RF chains can increase the cost and make the transceiver bulkier. Most of the baseband processing requires coherent signals. This means that the entire mixer Local Oscillators and Analogue to Digital Converter clocks (ADCS) need to be derived from same sources. This can present significant design challenges. The phase characteristics of RF components can change over time. These changes are relatively static and hence need calibration procedures to account for phase differences.

Most of the devices such as mixers amplifiers and ADCS used are non-linear devices. Using smart antennas can increase the number of such components used. This can affect the performance of the array if not checked periodically. Further more since antenna arrays use more than one source of signal the data bandwidth required for digital processing increases linearly with number of antenna elements used. This can limit data rates for different applications. Note that the technological challenges in terms of hardware and processing load can be satisfactorily met by resorting to present-day miniaturized RF components and faster and low power processors.

The accommodation of the antenna array, itself within a small factor device however remains a challenge. Base stations can easily host antenna arrays of four or more elements but with existing microstrip or patch antenna technology, up to three elements can be fitted in a handset form-factor. The wrapping of the hand around a handheld device may diminish the performance of a handheld smart antenna system.

2.7 Smart Antenna design in wireless Communication Center (WCC)

Base on the importance of smart antenna system in any wireless communication system, Wireless Communication Centre (WCC), Universiti Teknologi Malaysia started working at this project at the beginning of this year. This centre is targeting to implement two types of smart antenna systems. The first project is “Downlink switch beam smart antenna” for WiMax application and second project would be the same system for uplink transmission.

The smart antenna structure for phase one is similar to Figure-(2.4). The antenna elements consist of 16 microstrip antenna elements with rectangular shape. The algorithm are programmed by computer using C and Code Composer Studio and then loaded in DSP board.

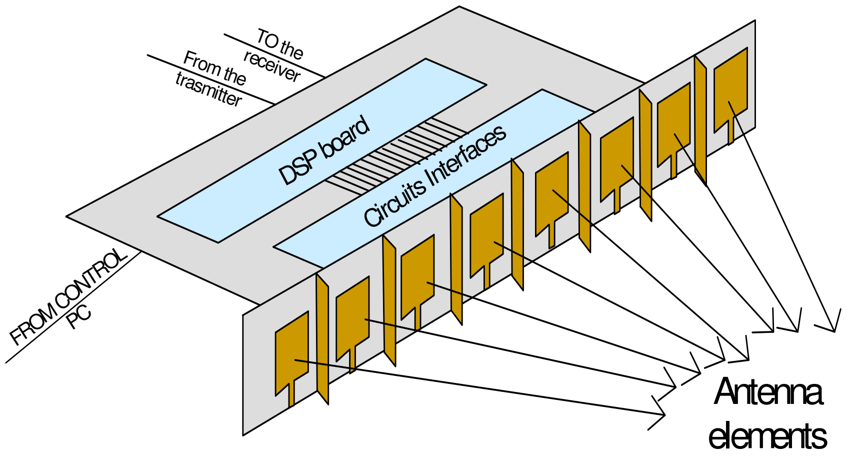


Figure-(2.4) Linear Array Structure

DSP has a main role in smart antenna system. The DSP is used just for forming and steering the beam using the calculated weights in the SDRAM base on the user direction. The location of the user can be identified using one of the directions of arrival algorithm. However, in this research we assume that the location of user is known to the system.

For this project the microstrip antenna array is used. Basically, scanning and shaping the beam is highly depend on array elements pattern. However, any type of antenna such as dipole, monopole, horns, reflectors, loops, aperture depend on application can be used. Compared with other type of antenna, microstrip antenna for wireless mobile communication could be the best option. The reason is due to its low price, ease of design and fair efficiency rather than other types of antenna.

2.8 Design of downlink smart antenna system in WCC

Technically, there are three possible designs which can be considered for this project as shown in Figures-(2.5), (2.6) and (2.7). However, we designed and implement the second structure due to its versatility and performance.

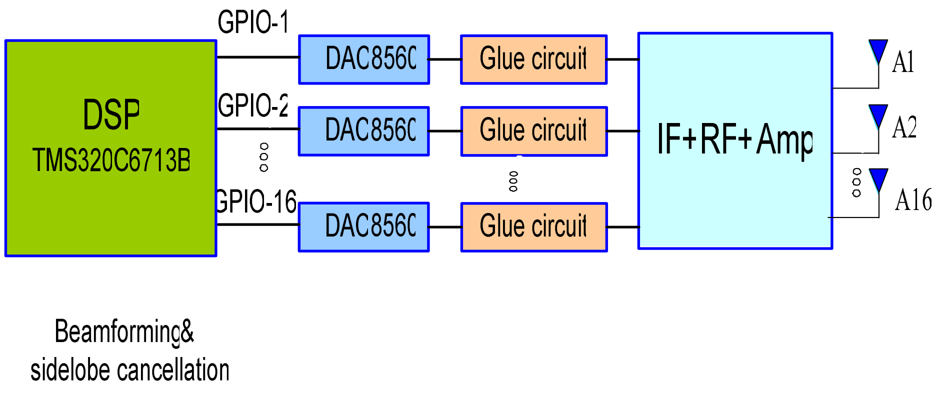


Figure-(2.5) First design

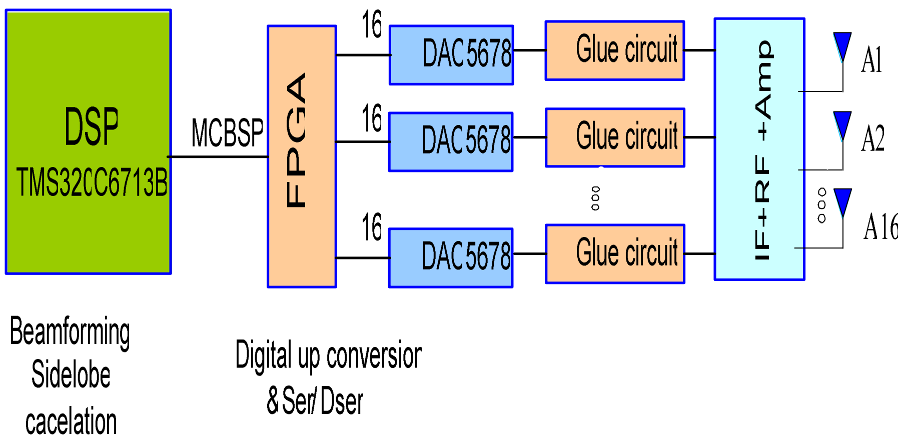


Figure-(2.6) second design

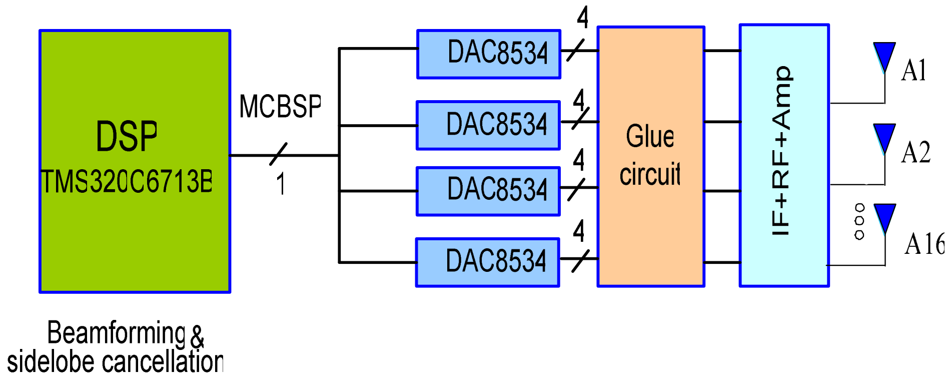


Figure-(2.7) third design

2.9 Expansion board design

The difference between abovementioned designs is on the extension or daughter cards. The expansion board design is affected by customer application. Different application has different requirement and constraints. For more throughput and data rate the high sample rate DAC must be used. Technically, the high speed DAC have parallel data inputs, as an example DAC5687 is parallel input DAC with sampling rate up-to 500MS/s. base on these facts, the required expansion board for such an application is different. Therefore some sort of interface matching is needed. The matching can be achieved using FPGA programming. Other than that, sometimes in order to decrease the price, one level of up-conversion is performed in digital domain instead of analog domain. Of course by doing digital up-conversion, the high speed DAC with high sampling rate must be used. Because, DAC must transform the IF digital signal to analog and therefore base on Nyquist criteria sampling frequency increases to double time of higher IF frequency.

Base on these facts, for this project the second option design for the expansion board is chosen Figure-(2.6). In this design, high sampling data rate DAC such as DAC5678 could be connected to FPGA board. We will explain this configuration in detail in FPGA chapter. The next section gives a picture of the third design.

2.9.1 The third design using programmable DAC

In third design, the daughter card can be designed by using 4 unit of DAC 8534 from TI as shown in Figure-(2.8).

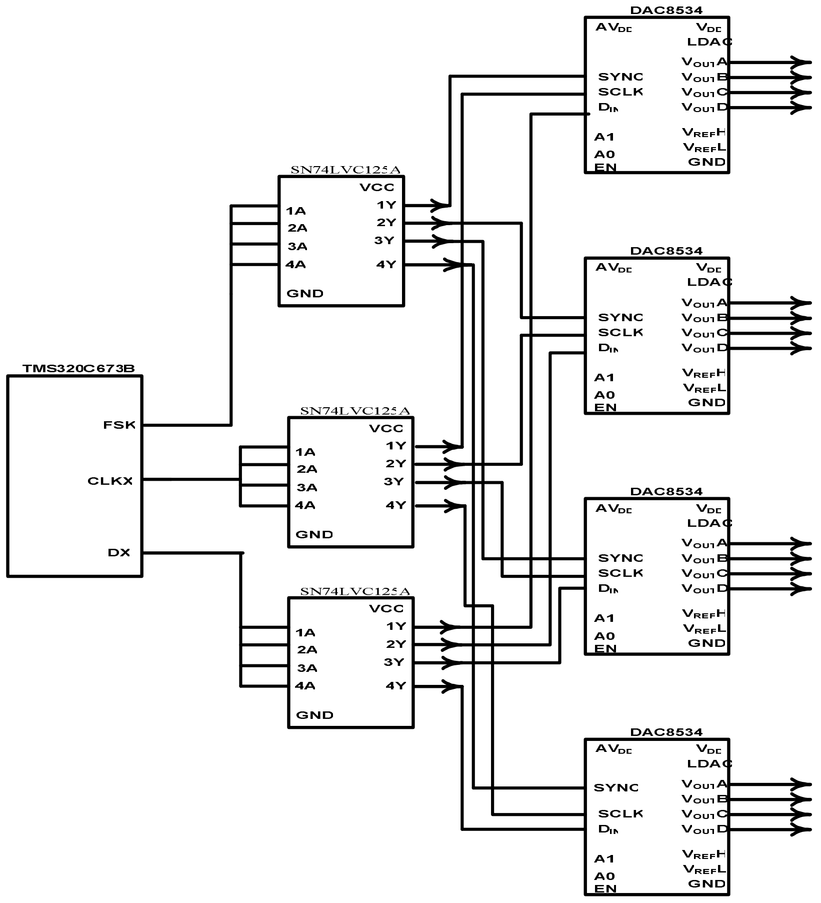


Figure-(2.8) baseband daughter card

These four D/A are 3-wire standard serial D/A which can be connected to MCBSP through quadruple bus buffer gate “SN74LVC125A”. It means they will share the serial channel and they can be synchronized through software programming. The functional block diagram and pin configuration of DAC8534 is shown in Figure-(2.9) and (2.10).

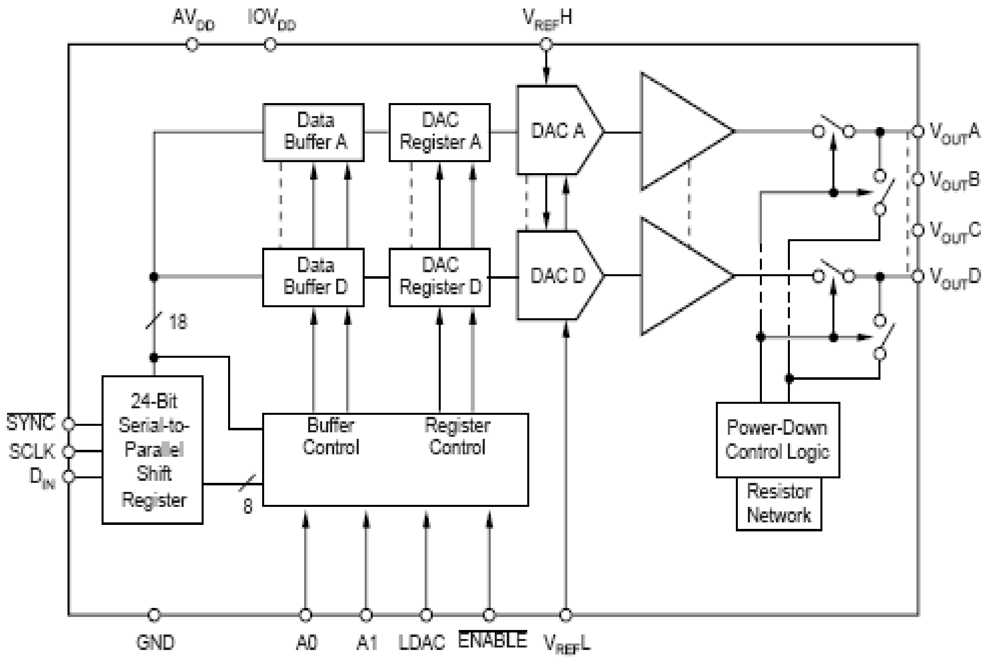


Figure-(2.9) functional block diagram of DAC8534

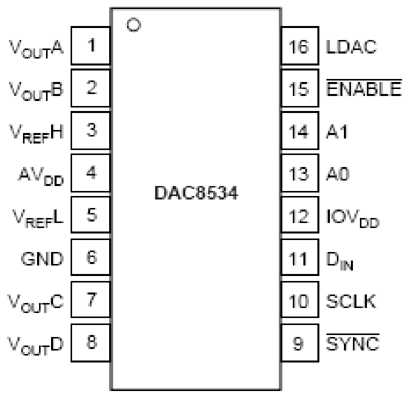


Figure-(2.10) pin configuration of DAC8534

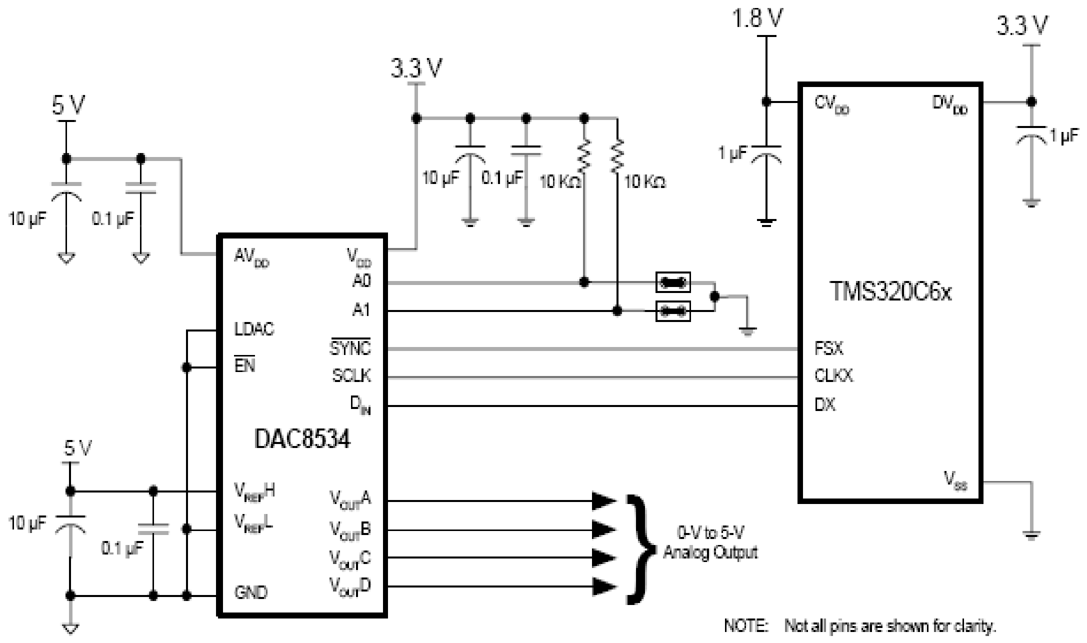


Figure-(2.11) interfacing a DAC8534 with a TMS320C6713B

2.9.1.1 Interfacing a DAC8534 with a TMS320C6713B

As it can be seen each DAC8534 is able to support four channels, therefore we need to interface 4 unit of DAC8534. Of course we need to amplify the output signal of TMS before connecting to DAC. Therefore, quadruple bus buffer gate “SN74LVC125A” is used to drive 4 unit of DAC. So, the baseband daughter card can be designed as shown in Figure-(2.6). VCC for SN74LVC125A can be up-to 3.6v. This buffer can accept the signal level of up-to 5 volt, the output current of this buffer is around 20mA. The pin configuration of this IC is shown in figure-(2.12).

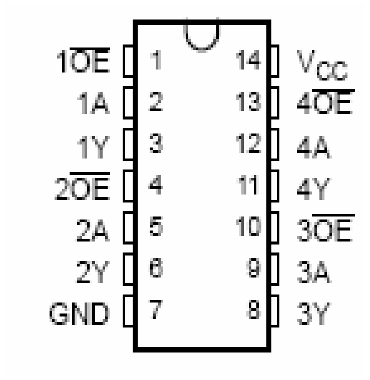


Figure-(2.12) quadruple bus buffer gate "SN74LVC125A"

The first possible option for expansion board is much simpler than the other types. For voice application which they need low sampling rate in compare to data communication the possible design could be Figure-(2.4). In this case we need to have 16 number of serial single channel DAC which can directly connect to McASP. The DAC in this case can be DAC8830 shown in Figure-(2.13) which can support up to 50Mb/s.

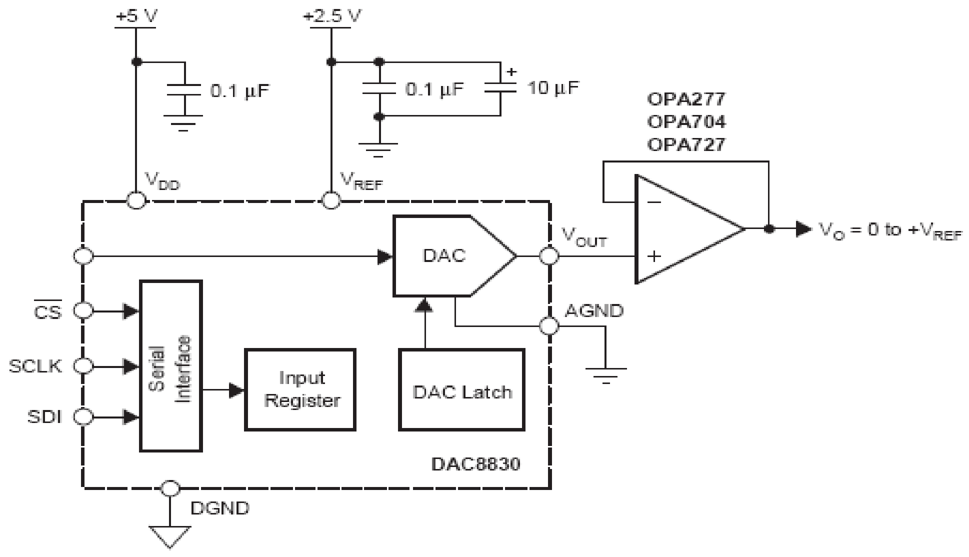


Figure-(2.13) typical connection of DAC8830

Another DAC option for this case could be DAC8560 shown in Figure-(2.14) which can be clocked at 30MHz. The difference between these two is in control part, DAC8560 can be selected using a chip-select pin, however DAC8830 is enabled by SYNCH pin, also there is some sort of control in input serial data.

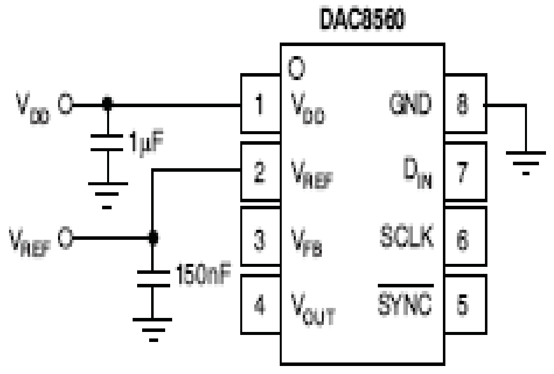


Figure-(2.14) typical connection of DAC8560

CHAPTER 3

DIGITAL BEAMFORMING WITH DSP BOARD

3.1 Introduction

The digital beamforming is integration between antenna technology and digital technology. DBF is based on converting signals into two streams of binary baseband I and Q signals, which represent the amplitudes and phases of signals. The beamforming is carried out by weighting these digital signals, thereby adjusting their amplitudes and phases such that when added together they form the desired beam. In this chapter the digital beamforming by using DSP board is briefly explained fundamentally. First, a general overview on beamforming theory is provided then the DSP board used in this project is introduced. Finally the algorithm of digital beamforming is explained.

3.2 Beamforming

Beamforming is one type of processing used to form beams to simultaneously receive a signal radiating from a specific location and attenuate signals from other locations [17]. Systems designed to receive spatially propagating signals often encounter the presence of interference signals. If the desired signal and interference occupy the same frequency band, unless the signals are uncorrelated, e. g., CDMA signals, the temporal filtering often cannot be used to separate signal from interference. However, the desired and interfering signals usually originate from different spatial locations. This spatial separation can be exploited to separate signal from interference using a spatial filter at the receiver. Implementing a temporal filter requires processing of data collected over a temporal aperture. Similarly, implementing a spatial filter requires processing of data collected over a spatial aperture.

A beamformer is a processor used in conjunction with an array of antennas to provide a versatile form of spatial filtering. The antenna array collects spatial samples of propagating wave fields, which are processed by the beamformer. Typically a beamformer linearly combines the spatially sampled time series from each antenna to obtain a scalar output time series in the same manner that an FIR filter linearly combines temporally sampled data. There are two types of beamformers, narrowband beamformer, and wideband beamformer. A narrowband beamformer is shown in Figure-(3.2).

In Figure-(3.1), the output at time M, y (M), is given by a linear combination of the data at the K sensors at time M:

$$y(M) = \sum_{i=1}^{K} w_i^* x_i(M) \qquad (3.16)$$

Where * denotes complex conjugate [18]. Since we are now using the complex envelope representation of the received signal, both w_i and $x_i(M)$ are complex. The weight w_i is called the complex weight.

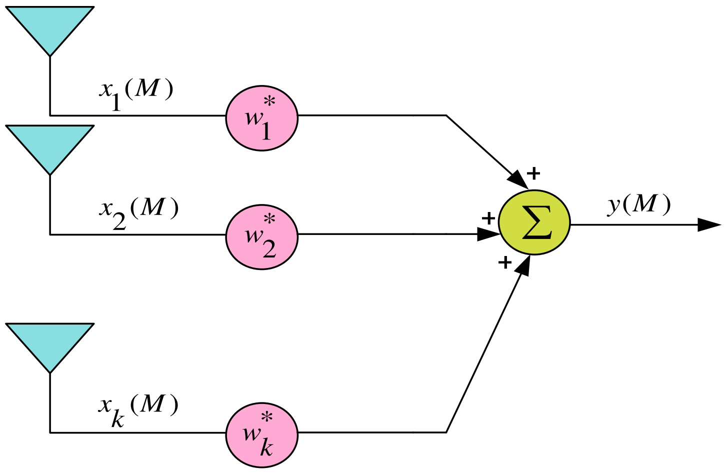


Figure-(3.1) A narrowband beamformer

In this project, for any directions weights are computed for sixteen antenna arrays. In consequence, there are $7 \times 16 = 112$ weights for seven directions. These weights are complex number, so they can change the phase and amplitude of the original signal. The weights can be expressed as:

$$w_k = e^{-j[(k-1)\frac{2\pi}{\lambda} d \sin\theta]} \qquad (3.17)$$

With replacing the $d = \frac{\lambda}{2}$, the equation (3.17) is became:

$$w_k = e^{-j[(k-1)\pi \sin\theta]} \tag{3.18}$$

For example for DOA= 30°, weights can be computed as below:

$$\begin{aligned}
w_1 &= \exp(-j*\pi*(1-1)*\sin(30)) = 1 \\
w_2 &= \exp(-j*\pi*(2-1)*\sin(30)) = -j \\
w_3 &= \exp(-j*\pi*(3-1)*\sin(30)) = -1 \\
&\vdots \\
w_{16} &= \exp(-j*\pi*(16-1)*\sin(30)) = -j
\end{aligned}$$

The matrix of weights for $\theta = 30^\circ$:

$$\begin{bmatrix} W_1 \\ W_2 \\ W_3 \\ W_4 \\ \vdots \\ W_{16} \end{bmatrix} = \begin{bmatrix} 1 \\ -j \\ -1 \\ j \\ \vdots \\ -j \end{bmatrix} \tag{3.19}$$

3.3 DSP Design (Beamforming and sidelobe cancellation)

Digital signal processor can be considered as the brain of smart antenna systems. In the other words, the smartness of smart antenna system is originated from this part. For this project 225 MHz DSP, built on the SDK

TMS320C6713B Figure-(3.2) is considered to use. For academicals and researches purposes usually the SDK module (Starter Development Kit) which is a unified kit consists of CPU, memories modules with some extension connector is technically adequate. The dominant advantages of the SDK version over the EVM (evaluation Module) is that the JTAG emulator is built onboard therefore the overall system cost decreases. Although SDK DSP board is designed to ease the academicals researches, it needs to be extended for the desired specific application.

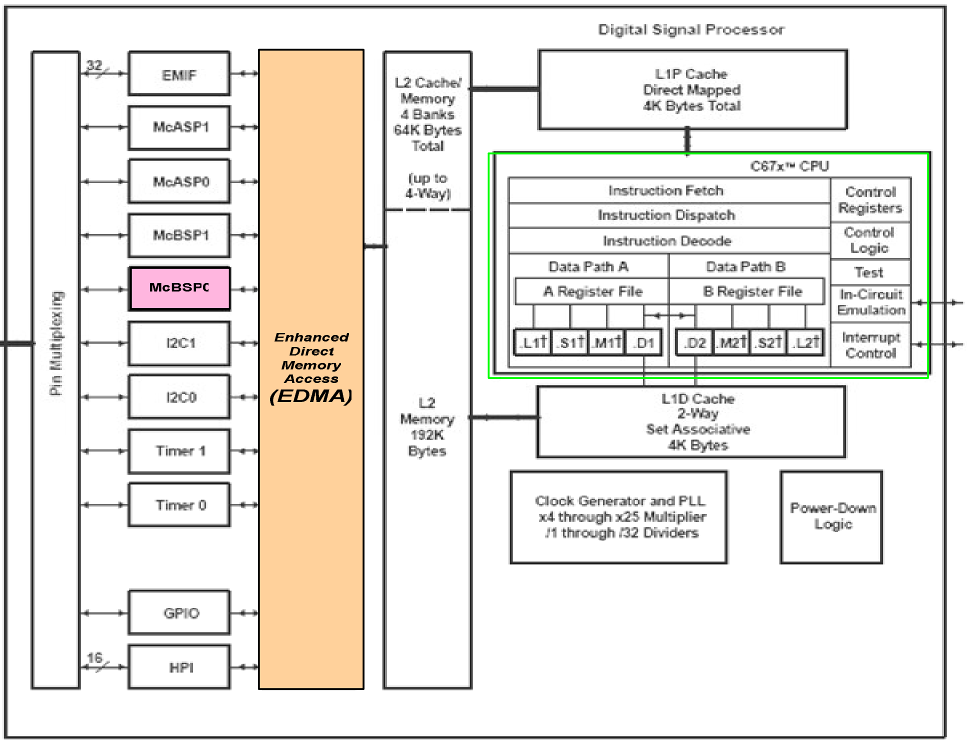


Figure-(3.2) Functional block and CPU (DSP core) diagram

3.4 DSP system design

Real time DSP system design is very challenging engineering task. The different group of specialist must work together in order to design the professional DSP system to apply in the real world application. In wide category, the implementation process is divided into software and hardware sections. These two sections as shown in the Figure-(3.3) must progress in parallel**.**

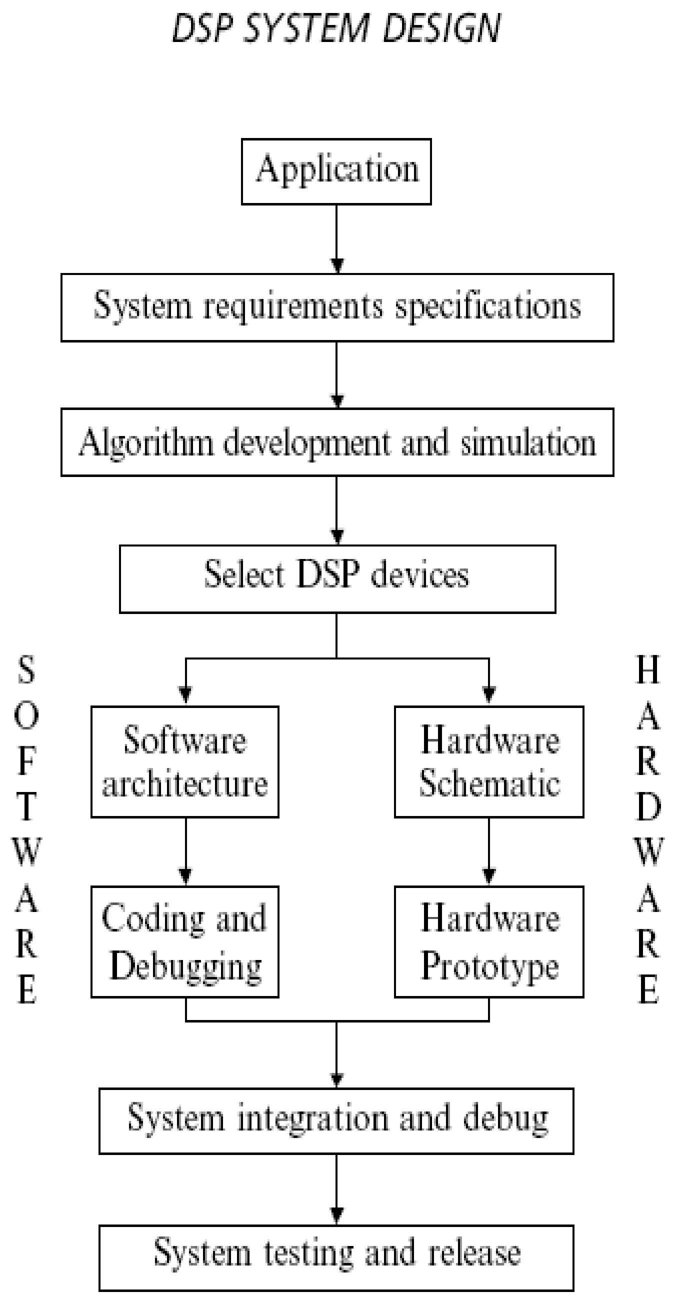


Figure-(3.3) simplified DSP system design

The block diagram of software development process of DSP design is given in Figure-(3.4).

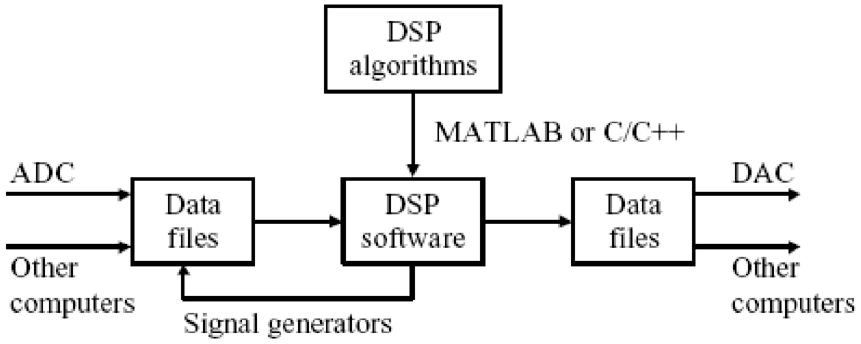


Figure-(3.4) software development process diagram

As it is cleared from the Figure-(3.4), firstly the algorithm is designed according to beamforming and sidelobe cancellation algorithms. Then the algorithms are implemented using MATLAB source code, and then these codes are converted to C language code. In the next step using code composer studio which is software capable of generating assembly code will do the rest of process for DSP programming. Then the digital input signal after ADC come to the system and it save as an input file. The loaded program in DSP manipulates the incoming digital signal. Finally, the results save in the output buffer or memory to be ready to send in the next step which is usually digital to analog transformation.

3.5 DSP speed and real time constraints

A limitation of DSP systems for real time application is that the bandwidth of the system is limited by the sampling rate. The processing speed determines the rate at which the analog signal can be sampled. For example, a real-time DSP system demands that the signal processing time, tp, must be less than the sampling period T, in order to complete the processing task before the new sample comes in. That is,

$$t_p < T$$

This real time constraint limits the highest frequency signal can be processed by a DSP system. This is given

$$f_M < \frac{f_s}{2} < \frac{1}{2t_p}$$

It is clear that the longer the processing the lower the signal bandwidth. Or in the other word to perform the wideband signal we need the high speed DSP processor with small amount of processing time for each sample of incoming data. However, this problem can be solved partially using rate converter or CIC filter as long as the quality of the processing is acceptable for that specific application. The DSP processor used for this project is from 6000 series. The specification is given as below:

1. Floating point processor
2. 2000 million instructions per second (MIPS) at 225 MHz
3. One analogue input/output
4. Memory module expansion
5. Host port interface (HPI)
6. Peripheral expansion
7. Embedded USB JTAG controller with plug and play drivers, USB cable included
8. TI TLV320AIC23 codec
9. 16MB SDRAM
10. 512K bytes of on board Flash ROM
11. On board IEEE 1149.1 JTAG connection

3.6 DSP implementation of smart antenna system

In smart antenna DSP perform just two functions, DOA estimation and beamforming. However, in this project to simplify the implementation process, it is assumed that the direction of the user has been identified before. So, we just need to perform the beamforming process for known direction. By having this assumption in mind, to steer the beam to pre-calculate user direction, the multiplication process of baseband signal to complex number is technically adequate enough. In addition the system is just able to produce one beam toward one single user. It means that to support multiple users simultaneously, the system must produce multiple beams which is out of the project scope. Moreover, to demonstrate the concept of digital beamforming in smart antenna, it is assumed that the application requires having seven beams at the seven desired angle shown in Figure-(3.5) each with 20 degree apart.

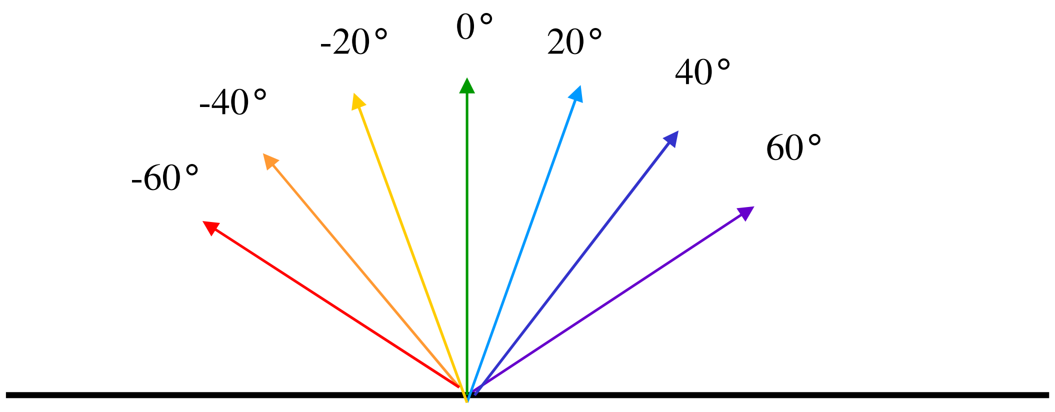


Figure-(3.5) the predefined antenna beam direction

By using computer programming, the antenna can steered the beam to these directions at predefined time interval called T. The radiation pattern of the system is optimized by conventional Chebyshev window weighting. By doing so, the side lobes are minimized and also the radiated power toward the desired user is maximized. It means the total system interference is minimized therefore the SNR increases.

After this introduction, now it is clear that that the main important task in baseband processing is just multiplication process of coming signal from conventional wireless transmitter to complex weights. This process is shown in Figure-(3.6).

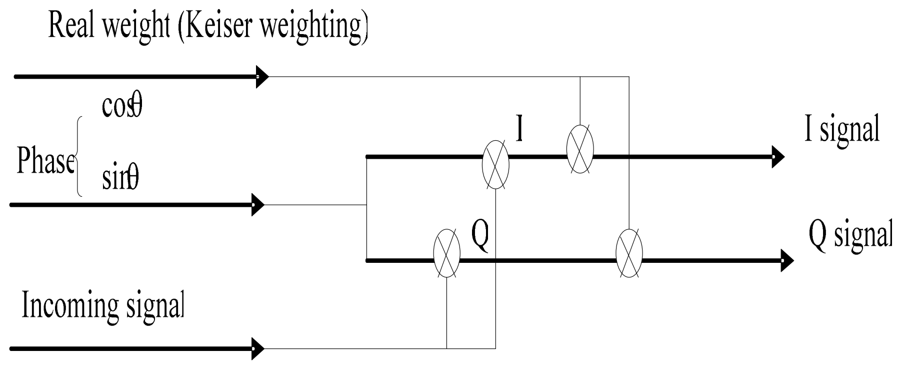


Figure-(3.6) the beamforming flow

3.7 Beamforming matrix

For each direction, there are 2×2×16=64 multiplication operations. This can be simplified by calculating beamforming matrix before applying in the program. It means that the multiplication process for phase and amplitude is manually performed for each direction before programming. Then the results are stored in the memory called lookup table. It means for the direction of θ we would have following operation.

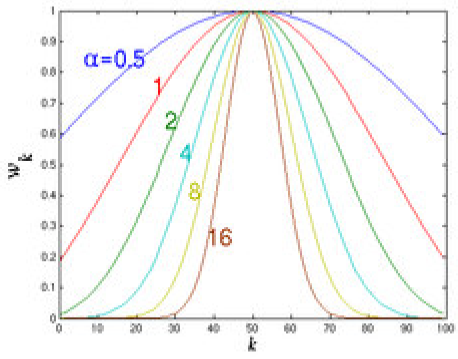
Wkaiser ×COSθ = Wreal ; for 7 directions 7×16 matrix

Wkaiser × SINθ = Wimag ; for 7 directions 7×16 matrix

If A=[Wreal]$_{7\times16}$ and B=[Wimag]$_{7\times16}$ then beamformer matrix=[A ;B]$_{14\times16}$, therefore it can be saved at the onboard SDRAM on the starter kit (DSK6713) , So the RAM expansion is not needed.

3.8 Sidelobe cancellation by using DSP

The Kaiser window is a window function w_k used for digital signal processing, and is defined by the formula:

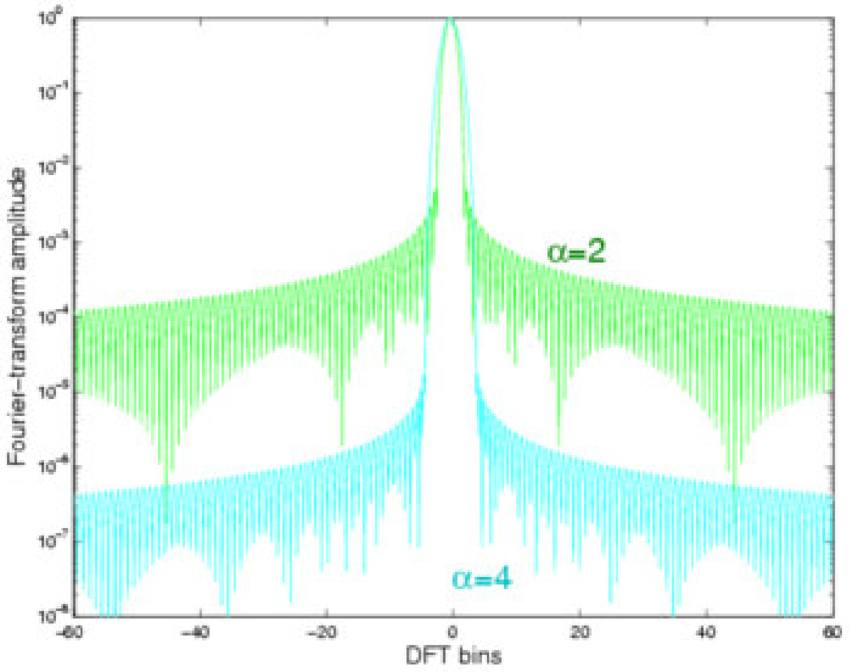


Kaiser Window function for *N*=100 and α= 0.5,1,2,4,8,16

$$w_k = \begin{cases} \dfrac{I_0\left(\pi\alpha\sqrt{1-(\frac{2k}{N}-1)^2}\right)}{I_0(\pi\alpha)} & \text{if } 0 \le k \le N \\ 0 & \text{otherwise} \end{cases}$$

Where I_0 is the zeroth order modified Bessel function of the first kind, α is an arbitrary real number that determines the shape of the window, and the integer N gives the length of the window (N+ 1 point).By construction, this function peaks at unity for $k = N/2$, i.e. at the center of the window, and decays exponentially towards the window edges.

The larger the value of $|\alpha|$, the narrower the window becomes; $\alpha = 0$ corresponds to a rectangular window. Conversely, for larger $|\alpha|$ the width of the main lobe increases in the Fourier transform of w_k, while the side lobes decrease in amplitude. Thus, this parameter controls the tradeoff between main-lobe width and side-lobe area, as is illustrated in the plot of the frequency spectra below. For large α, the shape of the Kaiser window (in both time and frequency domain) tends to a Gaussian curve. The Kaiser window is nearly optimal in the sense of its peak's concentration around ω=0 (Oppenheim *et al.*, 1999).



Frequency spectra of Kaiser Windows for α=2 and α=4.

The sharp minima in the side lobes are places where the amplitude goes all the way to zero, but does not here because of the finite plotting resolution.

CHAPTER 4

CHANNEL SEPARATION WITH FPGA BOARD

4.1 Introduction

In this chapter the channel separation for smart antenna project is briefly explained. First, a general overview on FPGA board is provided then channel separation is shortly discussed.

4.2 Field Programmable Gate Arrays (FPGAs) Overview

A field-programmable gate array (FPGA) is an integrated circuit (IC) that can be programmed in the field after manufacture. It is containing programmable logic components called "logic blocks", and programmable interconnects. Logic blocks can be programmed to perform the function of basic logic gates such as AND, and XOR, or more complex combinational functions such as decoders or simple mathematical functions. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memories. DSP algorithms may also be implemented using field-programmable gate arrays (FPGAs).

Advantages of FPGAs:

1. A shorter time to market
2. Ability to re-program in the field to fix bugs
3. Lower non-recurring engineering costs.

4.2.1Applications of FPGAs

Applications of FPGAs include digital signal processor DSP, software-defined radio, aerospace and defense systems, ASIC prototyping, medical imaging, computer vision, speech recognition, cryptography, bioinformatics, computer hardware emulation and a growing range of other areas.

4.2.2 FPGAs Architecture

There are three key parts of its structure: logic blocks, interconnect, and I/O blocks. Figure-(4.1) illustrates a typical FPGA architecture. The I/O blocks form a ring around the outer edge of the part. Each of these provides individually selectable input, output, or bi-directional access to one of the general-purpose I/O pins on the exterior of the FPGA package.

Inside the ring of I/O blocks lies a rectangular array of logic blocks. And connecting logic blocks to logic blocks and I/O blocks to logic blocks is the programmable interconnect wiring.

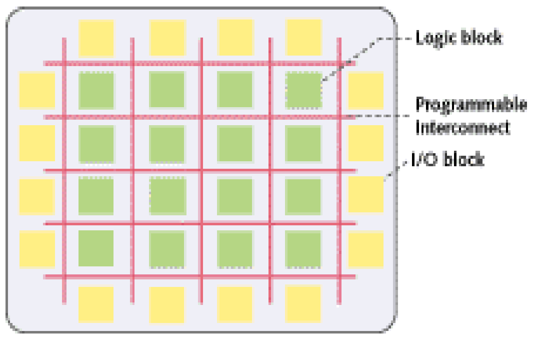


Figure-(4.1) internal structure of an FPGA

The logic blocks within an FPGA can be as small and simple as the macrocells in a Programmable Logic Devices (PLD) or larger and more complex (coarse-grained). However, they are never as large as an entire PLD, as the logic blocks of a Complex Programmable Logic Devices (CPLDs) are. Remember that the logic blocks of a CPLD contain multiple macrocells. But the logic blocks in an FPGA are generally nothing more than a couple of logic gates or a look-up table and a flip-flop.

Because of all the extra flip-flops, the architecture of an FPGA is much more flexible than that of a CPLD. This makes FPGAs better in register-heavy and pipelined applications. They are also often used in place of a processor-plus-software solution, particularly where the processing of input data streams must be performed at a very fast pace. In addition, FPGAs are usually denser (more gates in a given area) and cost less than their CPLD cousins, so they are the de facto choice for larger logic designs.

4.3 Altera Excalibur development board

In this project the Altera Excalibur development board is used. . It contains an APEX EP20K200E FPGA, 8Mbits (512K x 16) of internal Flash RAM, two 1Mbit (64K x 16) internal SRAM devices, an RS-232 communication port, a JTAG port, a parallel port, multiple expansion ports, two LEDs, two 7-Segment displays, and miscellaneous other switches and components [19]. Figure 4.2 shows a diagram of the Excalibur board.

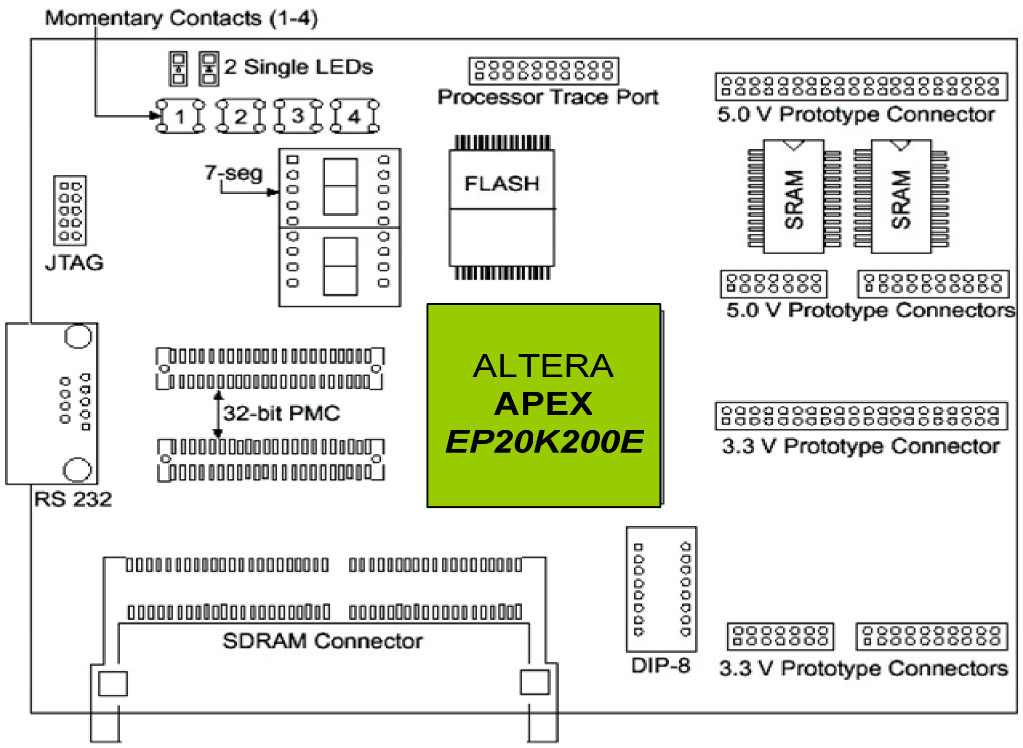


Figure-(4.2) the Excalibur Development Board

Contained within the APEX EP20K200E device is a Nios embedded processor. This soft core processor contains a 16 bit instruction set and is capable of operating with a 16 or 32 bit data bus. It can perform 50 million instructions per second with one instruction per clock cycle. With a Nios processor added to the EP20K200E device (as well as a Fast Fourier Transform block), there are still 150K spare gates of the 200K total gates available for use [19].The structure of the Nios processor resembles Figure 4.3:

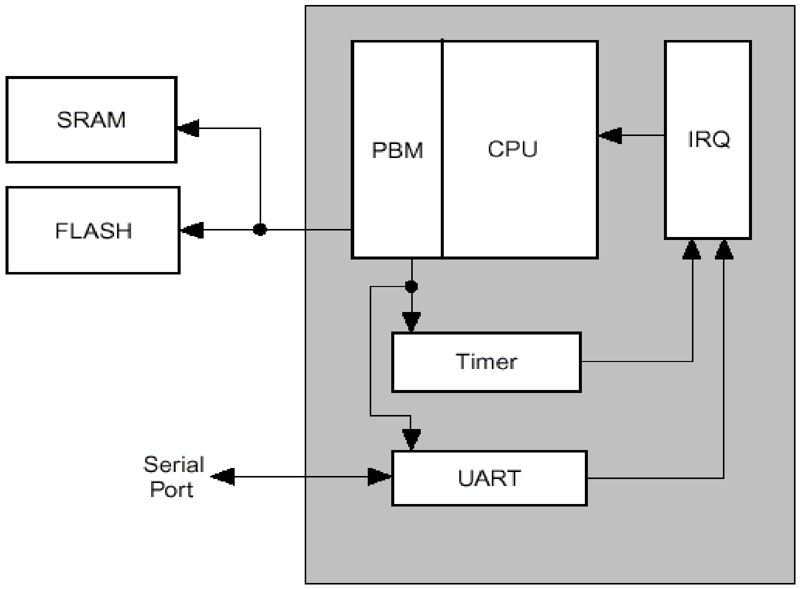


Figure-(4.3) the Nios Embedded Processor

The memory on the Excalibur board is set up as follows. The two internal SRAM devices can be used with 16 or 32 bit applications, but if the Nios processor is operating at 16 bits, only one of the two SRAM devices can be used. A 144 pin SODIMM memory expansion socket is also provided on the board if needed. Both the Nios processor and the APEX device share the flash memory [19].

The flash memory is organized as follows:

Flash Address	Size	Comments
0x1C0000 – 0x1FFFFF	256 Kbytes	Factory-default APEX Configuration
0x180000 – 0x1BFFFF APEX	256 Kbytes	User-defined Configuration data
0x100000 – 0x17FFFF	512 Kbytes	Nios instruction and Nonvolatile data Space

Table 4.1 – Flash Memory Configuration of FPGA

A factory programmed controller chip is contained on the board, a MAX7064 device that loads data from the flash and clocks it into the APEX device. It is possible to use user defined configurations by shorting J2 (normally open), and also to reprogram the device, but it is not recommended as it can result in an unusable board. The beginning address for the factory default is 0x1C0000, and for user defined configurations, the starting address is 0x180000 [19].

For expansion purposes, the Excalibur board provides 5 volt and 3.3 volt daughter cards. For 5 volt cards, a 40 pin connector (JP11), a 20 pin connector (JP13), and a 14 pin connector (JP12) are provided. The same applies to 3.3 volt cards, using JP8, JP10, and JP9, respectively [19].In this project the JP8 is used for transferring data from the DSP to FPGA board. Also, JP10 is used for clock and frame synchronization signals from DSP to FPGA.

There are three devices available for programming on the Excalibur board: the APEX device, the configuration controller, and the PMC (devices for JNC1 and JNC2). The ability to program each is determined by SW8, SW9, and SW10, respectively. If a switch is positioned to the left (marked connect on the board), then the corresponding device is added to the JTAG chain; each switch positioned to bypass will remove the device from the chain [19].

There are seven remaining switches on the board. SW1 is an eight pin user defined DIP switch. SW2 is a special button for resetting the board. Upon a reset, the configuration controller reloads the flash memory into the APEX device. SW3 is the clear function, which is defined by the configuration controller (a CPU reset by factory default). SW4-SW7 is user defined and may perform any function necessary. When pressed, the signals provided are logic zero [19].

There are two clocks for use on the Excalibur board. The first one is a 33.3333MHz signal provided by an onboard oscillator. The second one utilizes the phase locked loop circuitry on the board so the user can create their own clock [19]. In this research, the external clock which is come from DSP board is used.

4.4 FPGA programming

In this project, the Quartus II software is used for programming the FPGA. This program is written by verilog. After programming the pins of FPGA must be assigned for transferring data to the next part. Figure-(4.4) shows the pins assignment of FPGA.

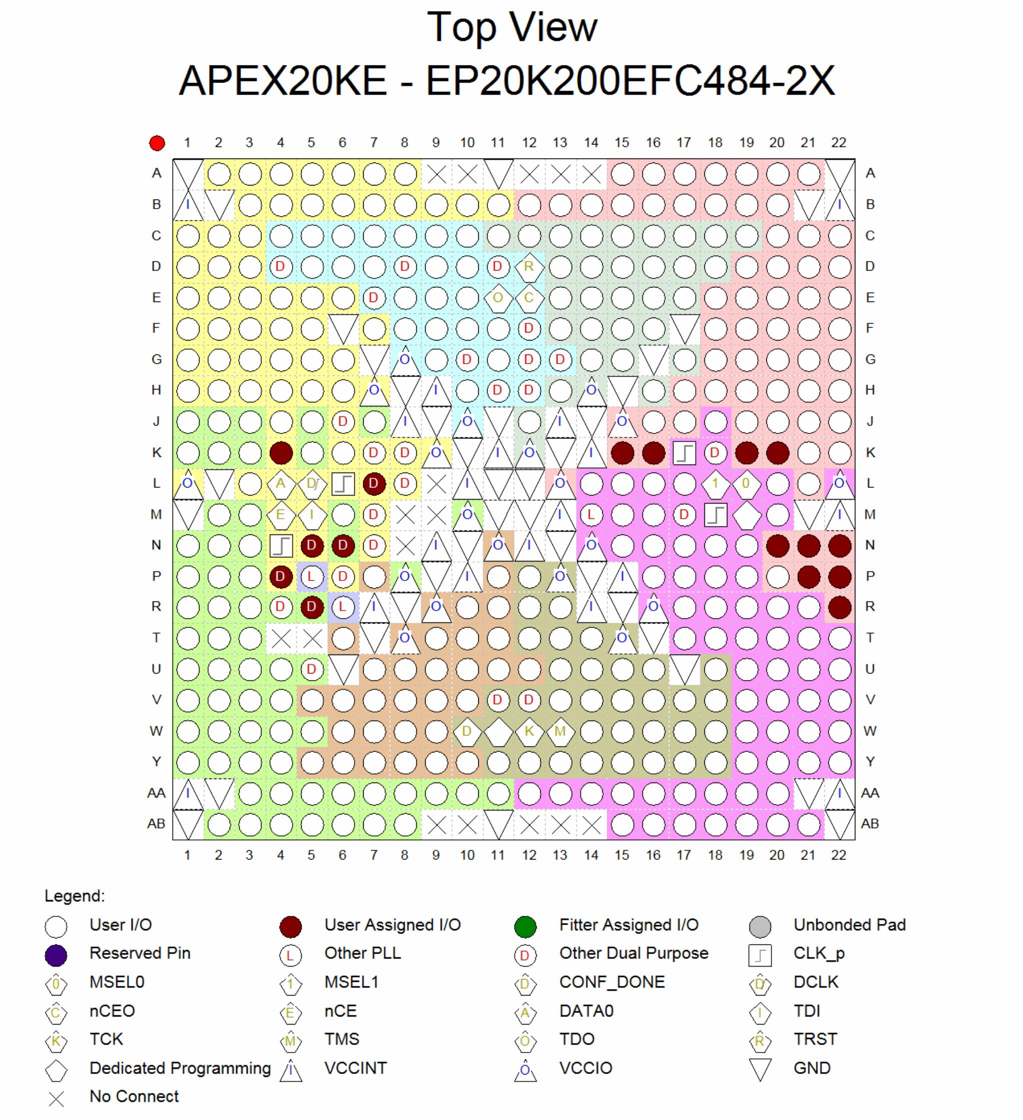


Figure-(4.4) Pin assignment of FPGA

CHAPTER 5

INTERGRATION OF DIGITAL SIGNAL PROCESSOR AND FPGA BOARD

5.1 Introduction

One of the limitations of DSP board is the number of high data rate output port. This board only has two high data rate ports called Multi channel Buffer Serial Port (McBSP). Because of this limitation, the expansion board is needed. In this project, FPGA board is used as an expansion board.

5.2 System design

Figure-(5.1) shows the design for the expansion board is chosen for this project. In this design, high sampling data rate DAC such as DAC5678 can be connected to FPGA for digital to analog conversion.

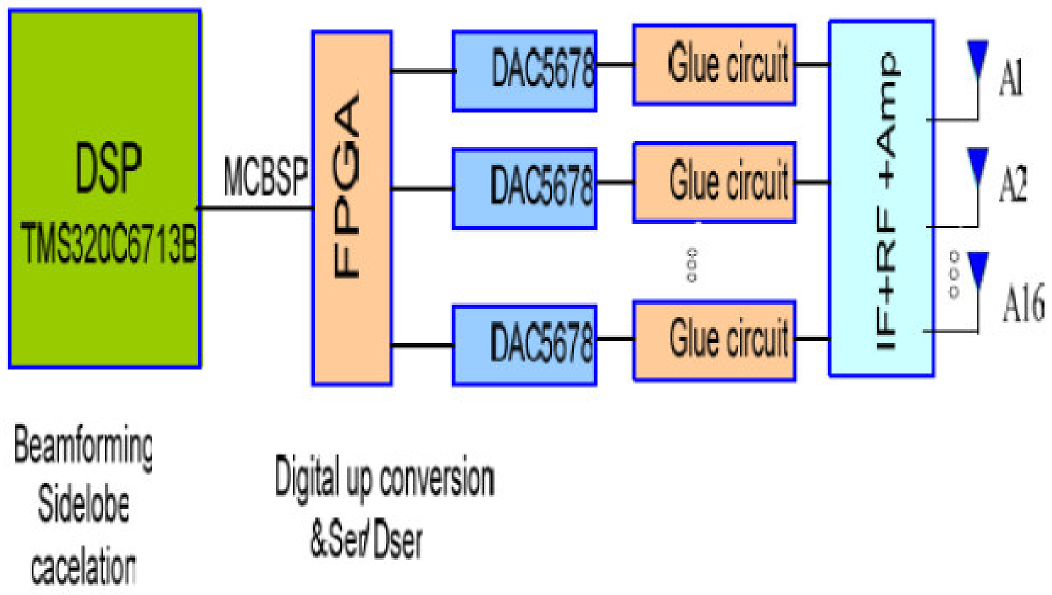


Figure-(5.1) Smart antenna system model

5.3 Integration of DSP and FPGA

For connecting the DSP to FPGA, the TMS320C6713 DSK supports three expansion connectors that follow the Texas Instruments interconnection guidelines. The expansion connector used in this project is called Peripheral Expansion Connector (Figure-5.2). It is an 80 pin 0.050 x 0.050 inches connector. This connector provides both +12V and -12V to the daughter card [20].

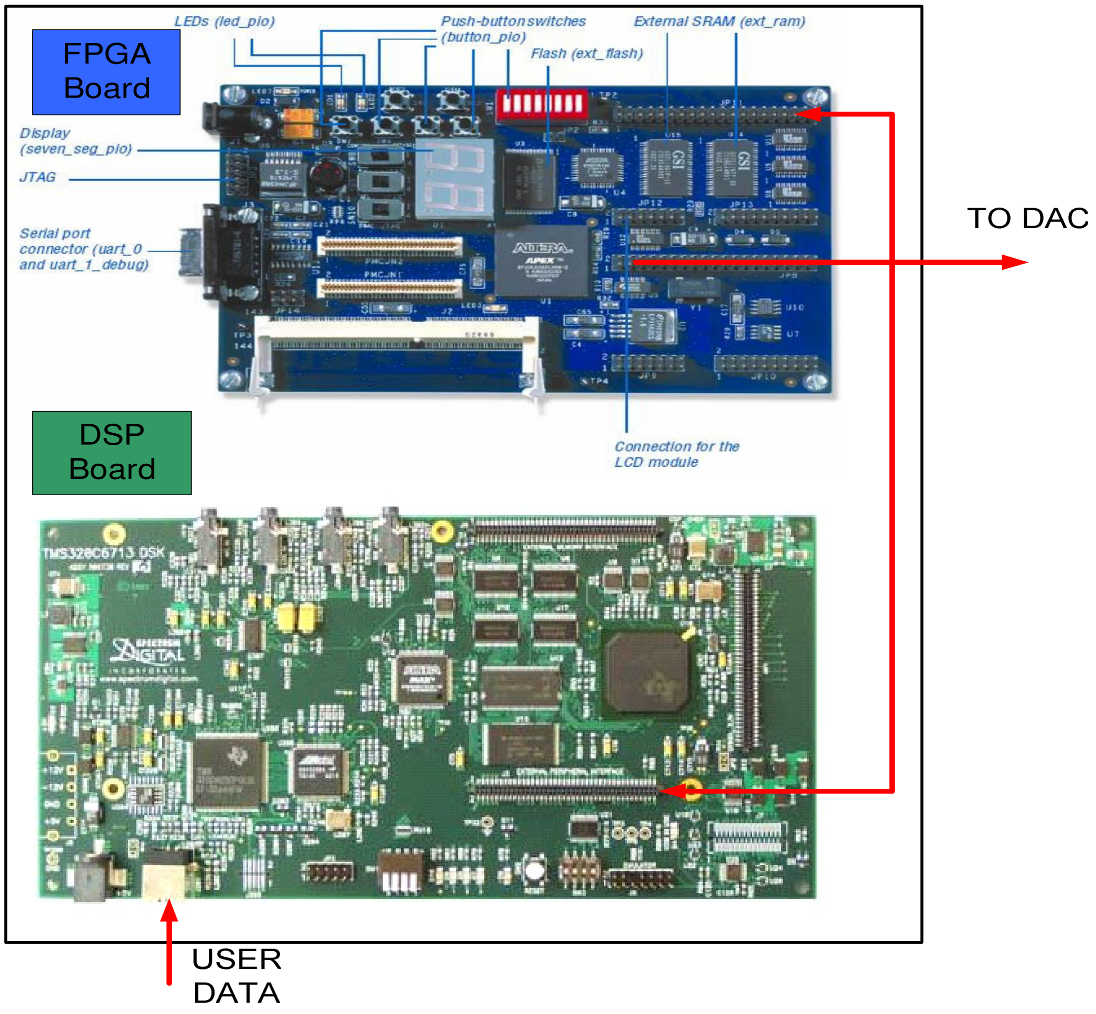


Figure-(5.2) Integration of DSP and FPGA

In this research, firstly for transferring data to device peripherals, EDMA and MCBSP must be programmed. The enhanced DMA (EDMA) controller of the TMS320C6713B device is a highly efficient data transfer engine, capable of maintaining up to 1200 Mbytes per second (MB/s) of data throughput during operation. The EDMA handles all data movement between the level-two memory and the device peripherals (in this project is FPGA), including cache-servicing, non-cacheable memory accesses, user-programmed data transfers, and host accesses (Figure-5.3).

Another part used for transferring data to the FPGA is McBSP ((Figure-5.3). This part provides the following functions:

- Full-duplex communication
- Double-buffered data registers
- Independent framing and clocking for receive and transmit
- External shift clock or an internal

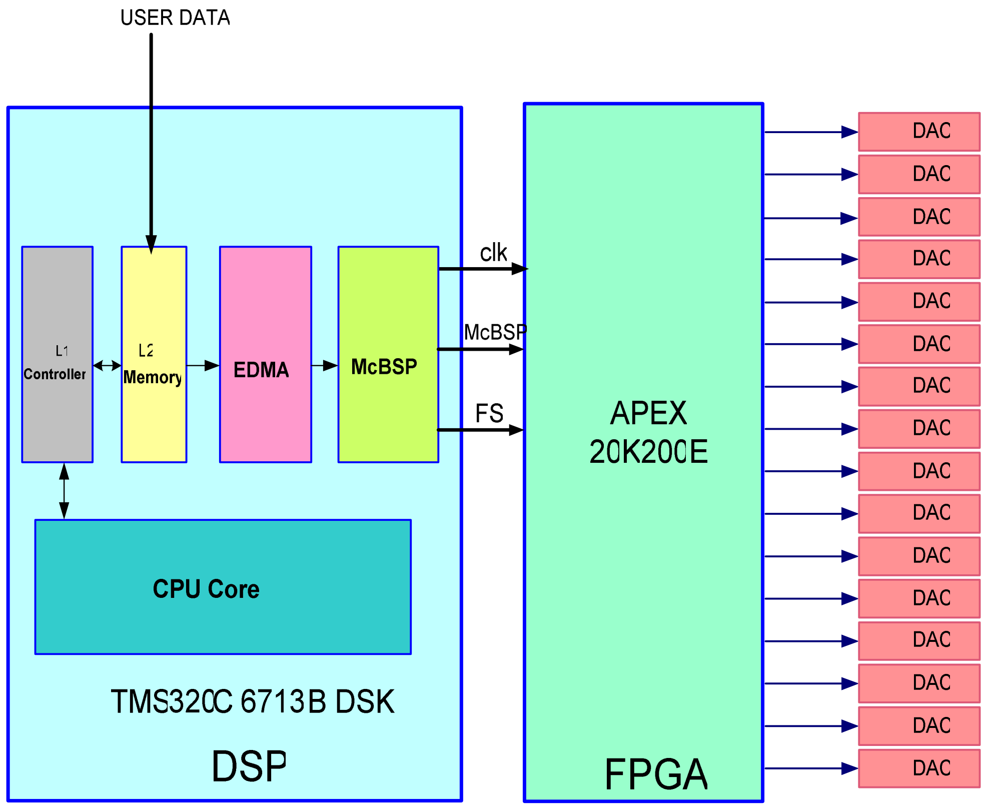


Figure-(5.3) Transferring from DSP and FPGA

5.4 Summary

In this chapter, the integration of DSP and FPGA is explained. For providing this connection, the EDMA and McBSP must be programmed. And these programming is done by using the code composer studio software.

CHAPTER 6

IF AND RF FRONT-END DESIGN FOR SMART ANTENNA

6.1 Introduction

In all above three designs which are discussed in chapter two, we need to have our own design for the IF and RF part because there is no on-the-shelf device RF chain for smart antenna. Although some companies have fabricated IF and RF part, it is very costly and in addition there is no 16 channel IF and RF part! A 16-channel IF and RF chain can be designed as shown in Figure-(6.1).

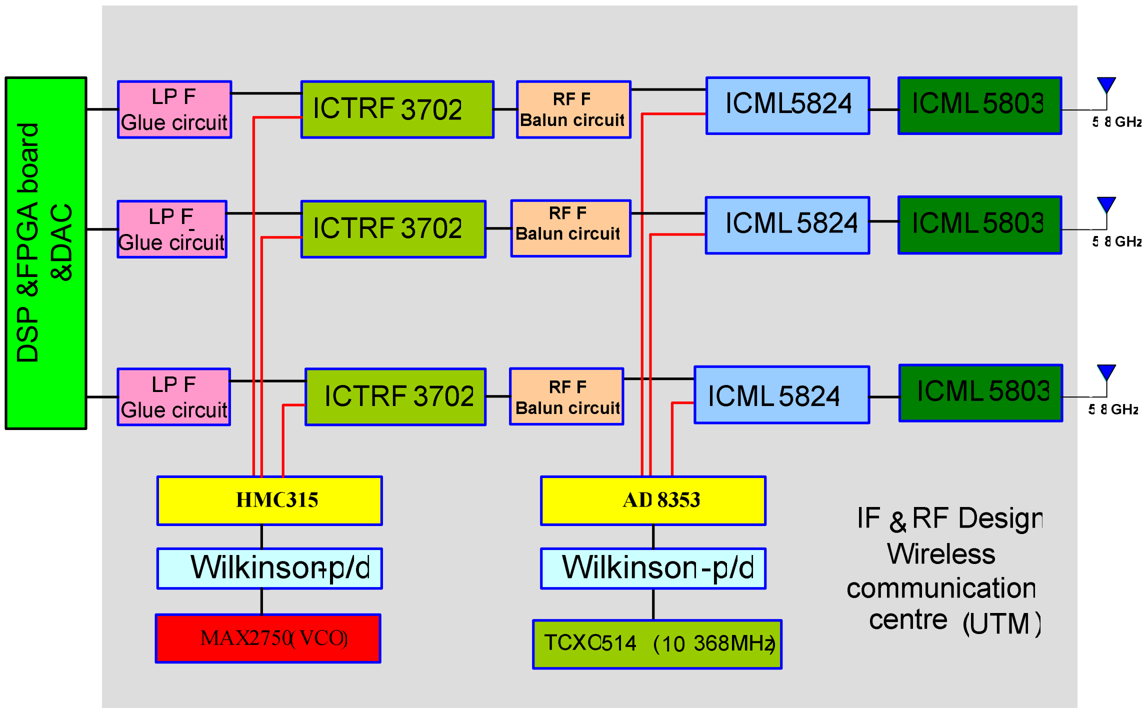


Figure-(6.1) IF and RF front end design

This RF-front-end can transform the baseband data to 5.8GHz RF signal. This design is integration of TI and Micro linear devices companies. The single channel IF part comes from TI and the single channel RF part is designed by Micro linear company.

6.2 Baseband glue circuit

The design is started from multi-channel baseband data which come to the glue circuit in the first step. The data then transform to balanced type using this circuit shown in Figure-(6.2). By using differential signal the noise effect in the IF part can be highly mitigated. This circuit is necessary since the IF transformation is done by TRF3702 is balanced input device.

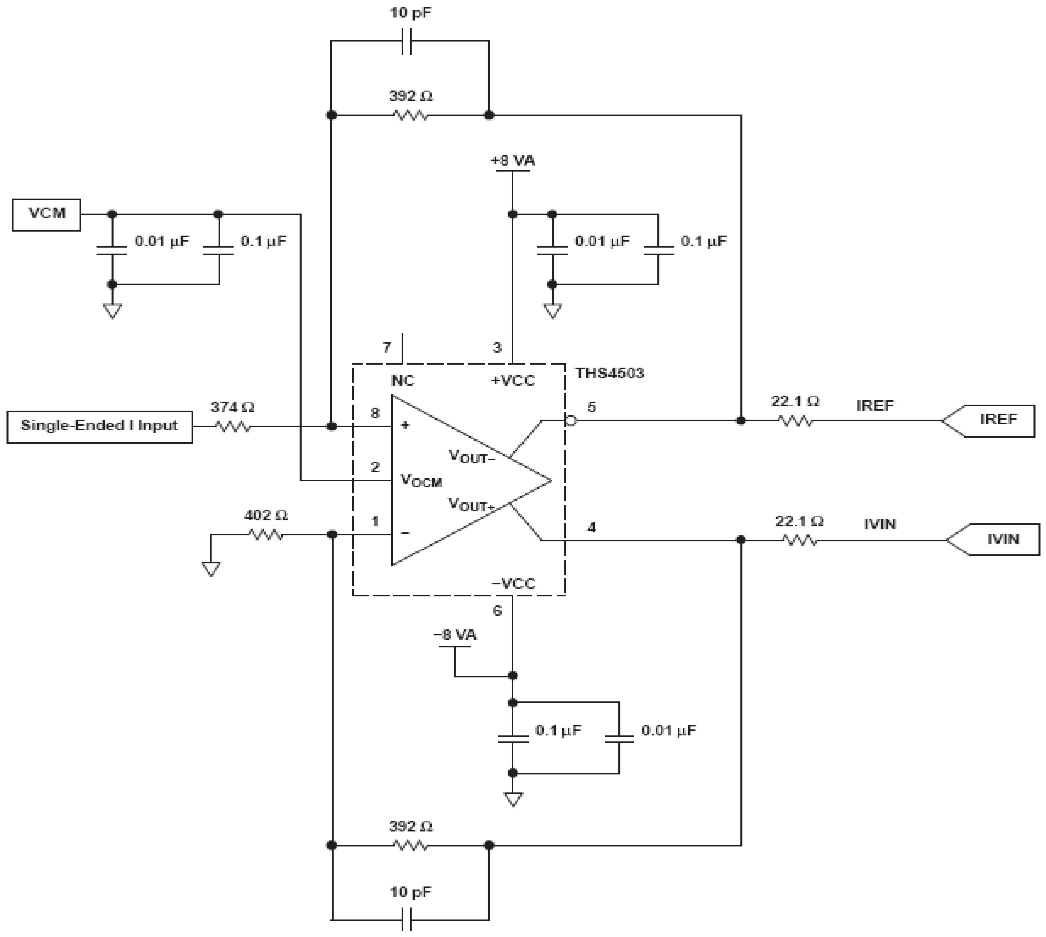


Figure-(6.2) connecting the single ended signal to balanced input of TRF3702

The glue circuit for the second expansion board shown in Figure-(2.10) can be similar to Figure-(6.3) because in this case we do not need to design Balun circuit. Therefore, the Glue circuit as shown in dash-line in Figure-(6.3) could be very simple.

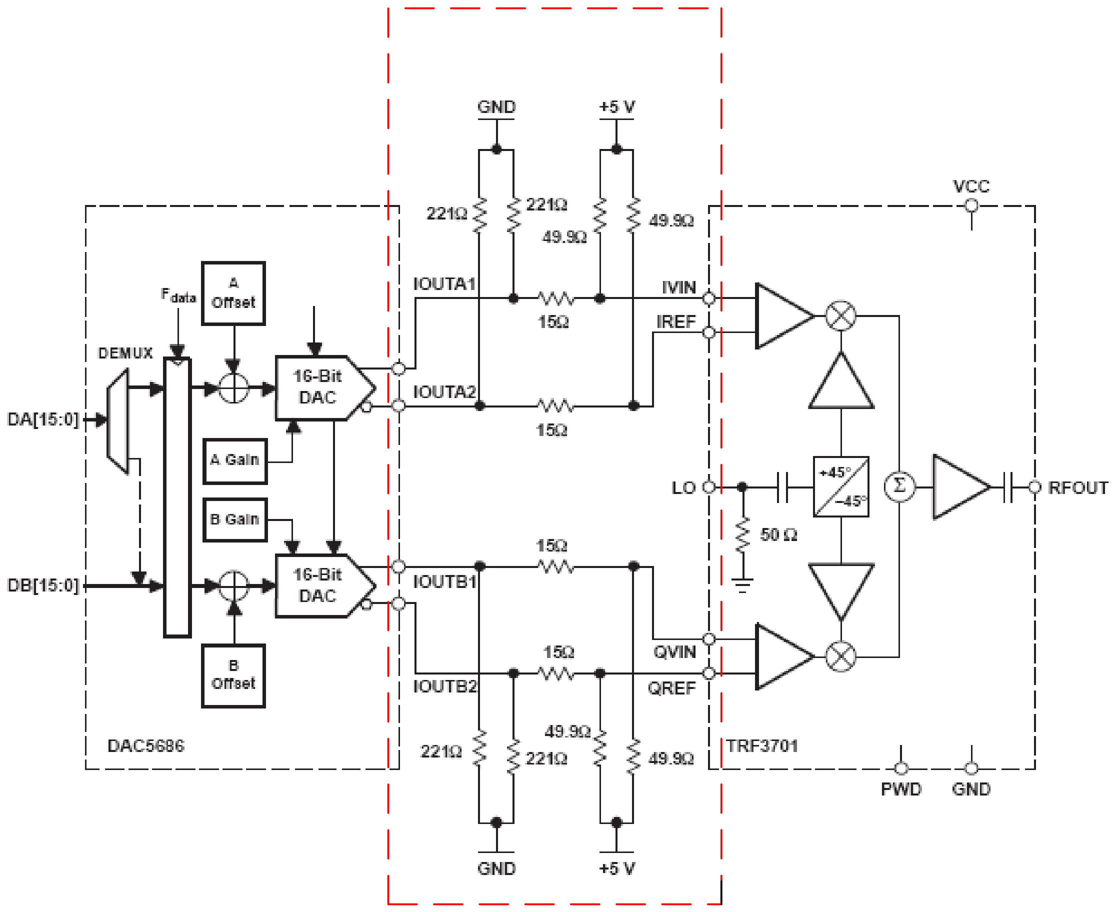


Figure-(6.3) glue circuit connecting the DAC5687 to TRF3702

6.3 Quadrature-modulator

TRF3702 is an IF to RF up-converter, the pin arrangement shown in Figure-(6.4) this IC can up-convert the baseband or IF signal to 2.4GHz signals.

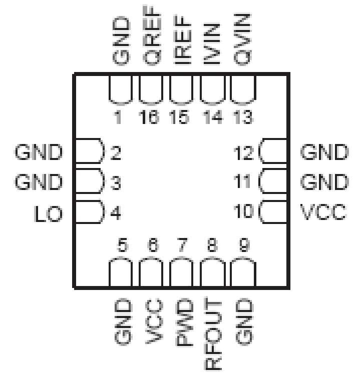


Figure-(6.4) TRF3702 pin arrangement

The functional block diagram of the TRF3702 is shown in Figure-(6.5). This component is able to work with I and Q signals. However, in our project the signal is just real signal therefore the Q signal must be grounded.

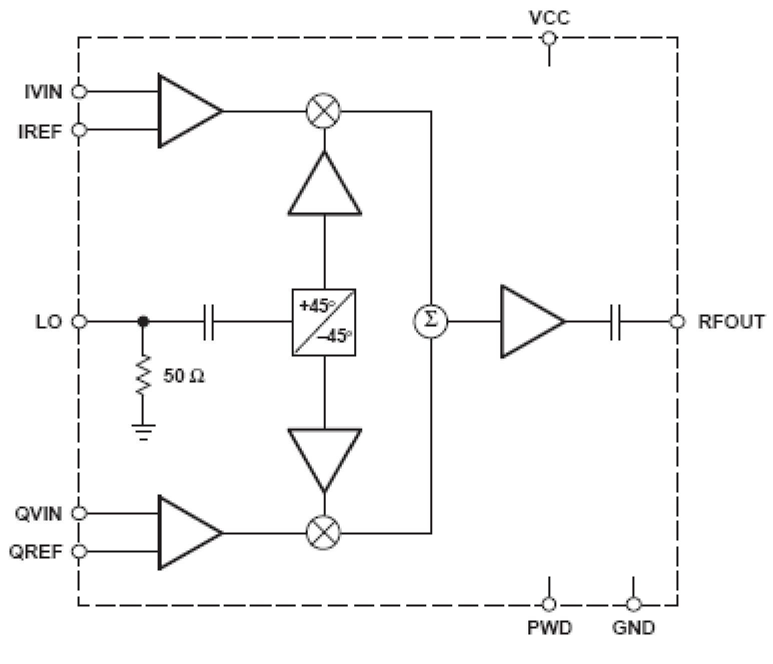


Figure-(6.5) TRF3702 functional block diagram

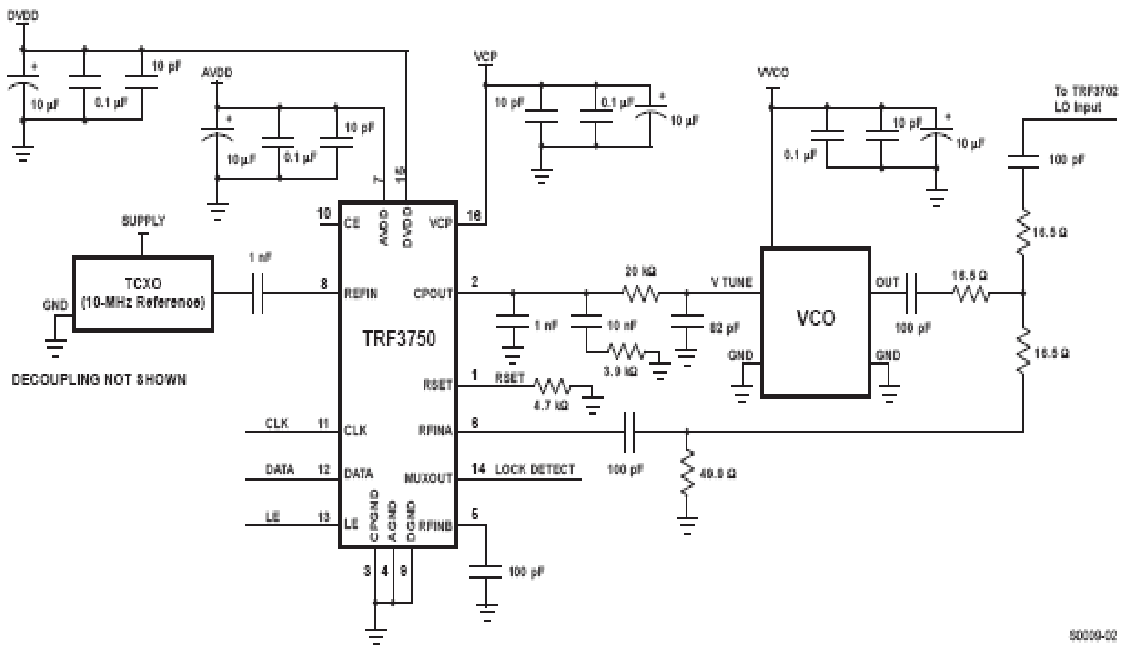


Figure-(6.6) Generating the LO signal for TRF3702 using TRF3750

6.4 local oscillators for 2.4GHz signal

If the modulator need to works with multi carrier system, the TRF3750 shown in Figure 4.6, can be used as a frequency synthesizer. This PLL can generate different frequency components for the system. However, in our system the single RF frequency at is needed. Therefore, we can use a VCO to generate 2.4 RF sin frequencies. The VCO is considered for this project is "**MAX2750**" from MAXIM Company.

Figures 6.7 and 6.8 are shown the typical layout and pin configurations of for MAX2750. The MAX2750/MAX2751/MAX2752 VCOs are implemented as an LC oscillator topology, integrating all of the tank components on-chip. This fully monolithic approach provides an extremely easy-to-use VCO, equivalent to a VCO module. The frequency is controlled by a voltage applied to the TUNE pin, which is internally connected to a varactor.

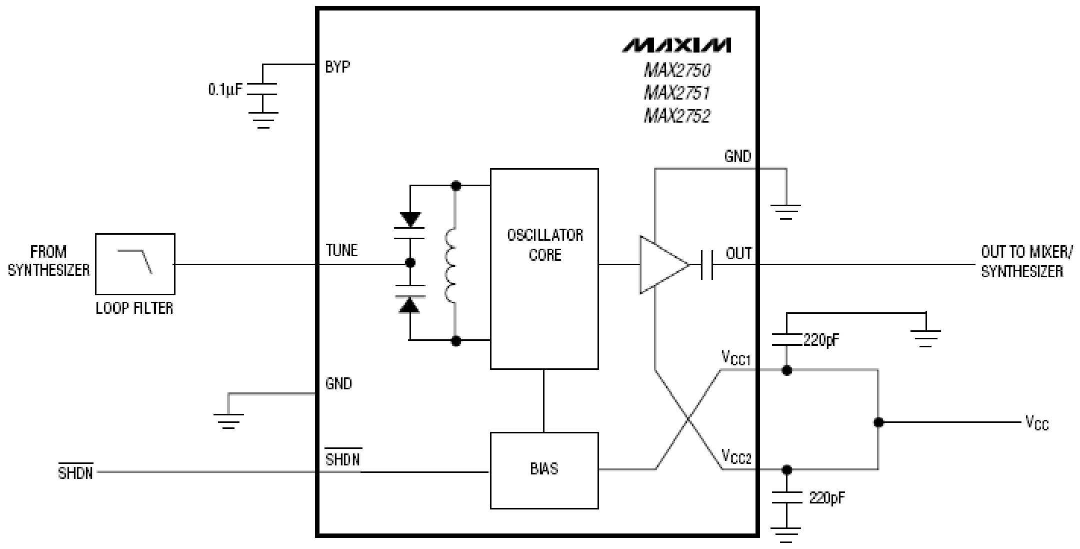


Figure-(6.7) the typical layout for" MAX2750"

The VCO core uses a differential topology to provide a stable frequency versus supply voltage and improve the immunity to load variations. In addition, there is a buffer amplifier following the oscillator core to provide added isolation from load variations and to boost the output power.

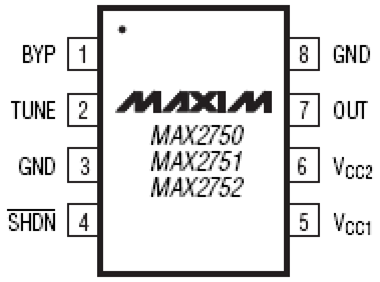


Figure-(6.8) the pin configuration for" MAX2750"

6.5 Wilkinson power divider

A 2.4GHz and 10.368 MHz Wilkinson power divider also need to be designed to equally divide the LO and reference frequency of IF and RF part to their counterparts.

6.6 Balun Circuits

Since, the output of TRF3702 is single ended, the either Balun circuit of in Figure 3.13 by using AD8352 or a Balun chip shown in **F**igure 6.9 circuit can be used**.** The Balun prior to ML5824 circuit is needed since the input of ML5824 is differential input.

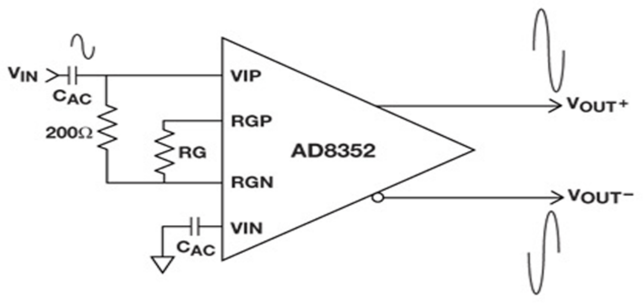


Figure 6.9 Balun circuit before ICML5824, CAC=0.1µf RG=120KΩ

This Balun figure 6.10 circuit functions at 2GHZ, the better choice could be 2450BL15B200 from Ceramic solution which is simple solution for this purpose at the frequency of 2.4. The pin configuration can be seen in figure below.

No.	Function
1	Unbalanced Port
2	NC
3	GND
4	Balanced Port
5	NC
6	Balanced Port

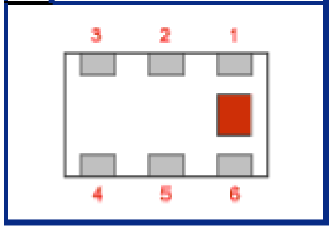


Figure 6.10 Balun circuit configurations at 2.4 GHz

6.7 Temperature compensated crystal oscillator (TCXO)

The Temperature compensated crystal oscillator (TCXO 514) from Oscillant Company can be used in RF part as a reference frequency. The value of reference RF reference frequency value is 10.368MHz.

Figure-(6.11) low profile TCXO-514 frequency range 1.2 to 100MHz

The oscillator for the RF part is a built in oscillator which is built on ML 5824. However, for the IF part the oscillator is voltage controlled oscillator (VCO).

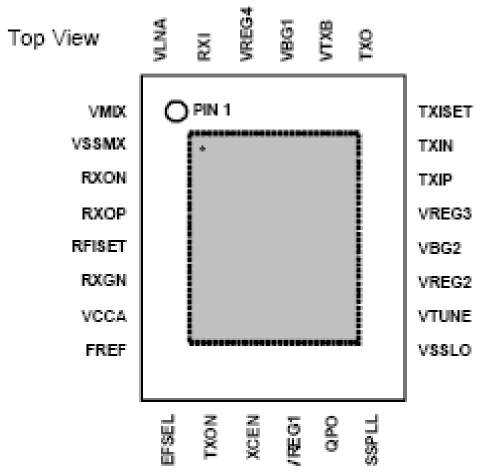


Figure-(6.12) ML5824 top-views

6.8 linear RF and IF amplifier

The Wilkinson power divider divide the power in 16 branch, therefore power amplification is necessary prior to LO in TRF3702. Either of amplifiers in figure 6.13 and 6.14 can be used for this project for the 2.4GHz. However, for the second power Wilkinson AD8353 should be used.

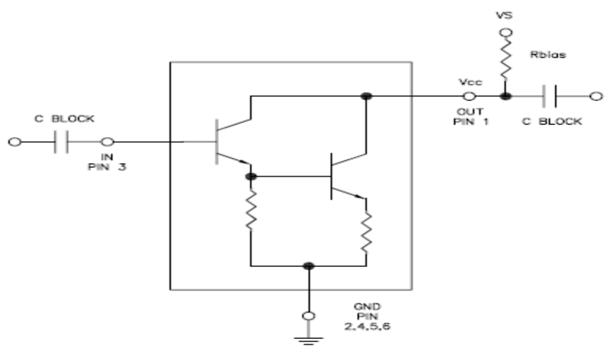


Figure-(6.13) HMC315 linear amplifier frequency range up-to 7 GHZ (C block=0.1micro, Rbias=5k)

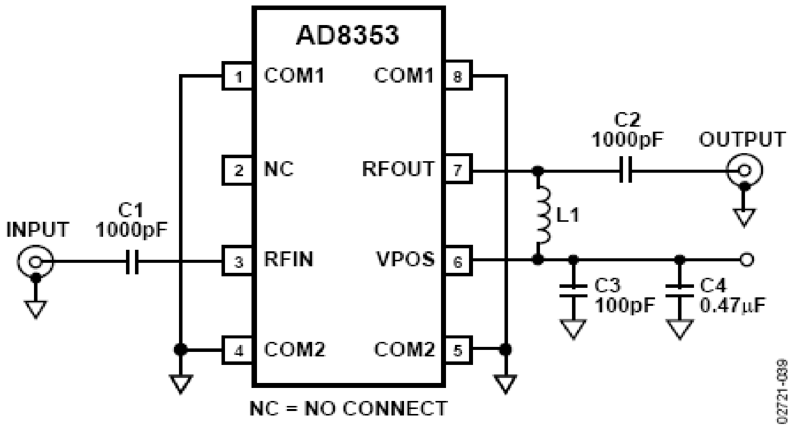


Figure-(6.14) AD8353 amplifier (frequency range 1 MHZ up-to 2700 GHZ)

6.9 Second up-converter

The Ml5824 is a cost effective solution to up-convert 2.4 signals to 5.8 RF signals, in this project to simplify the RF design we used, this IC following by ML5803 a power amplifier**.** The functional block diagram of ML5824 is shown in figure 6.15.

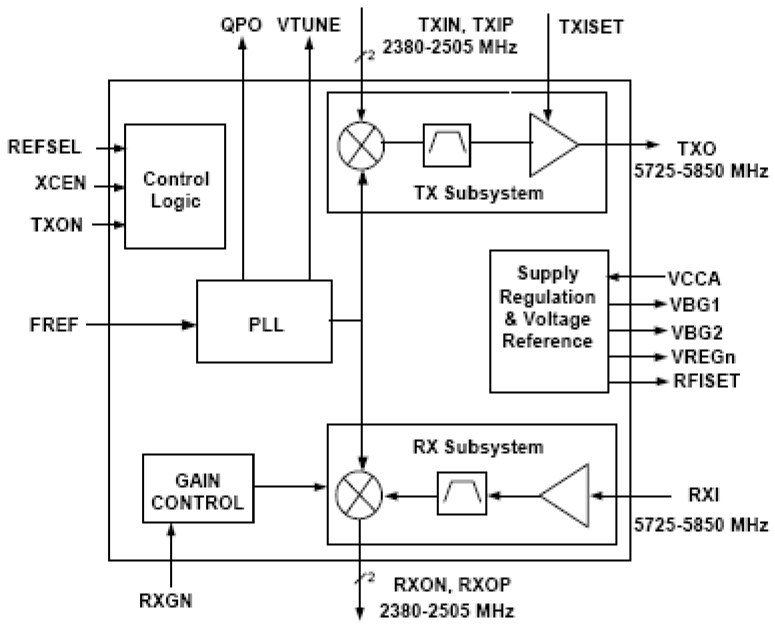


Figure-(6.15) Ml5824 functional block diagram

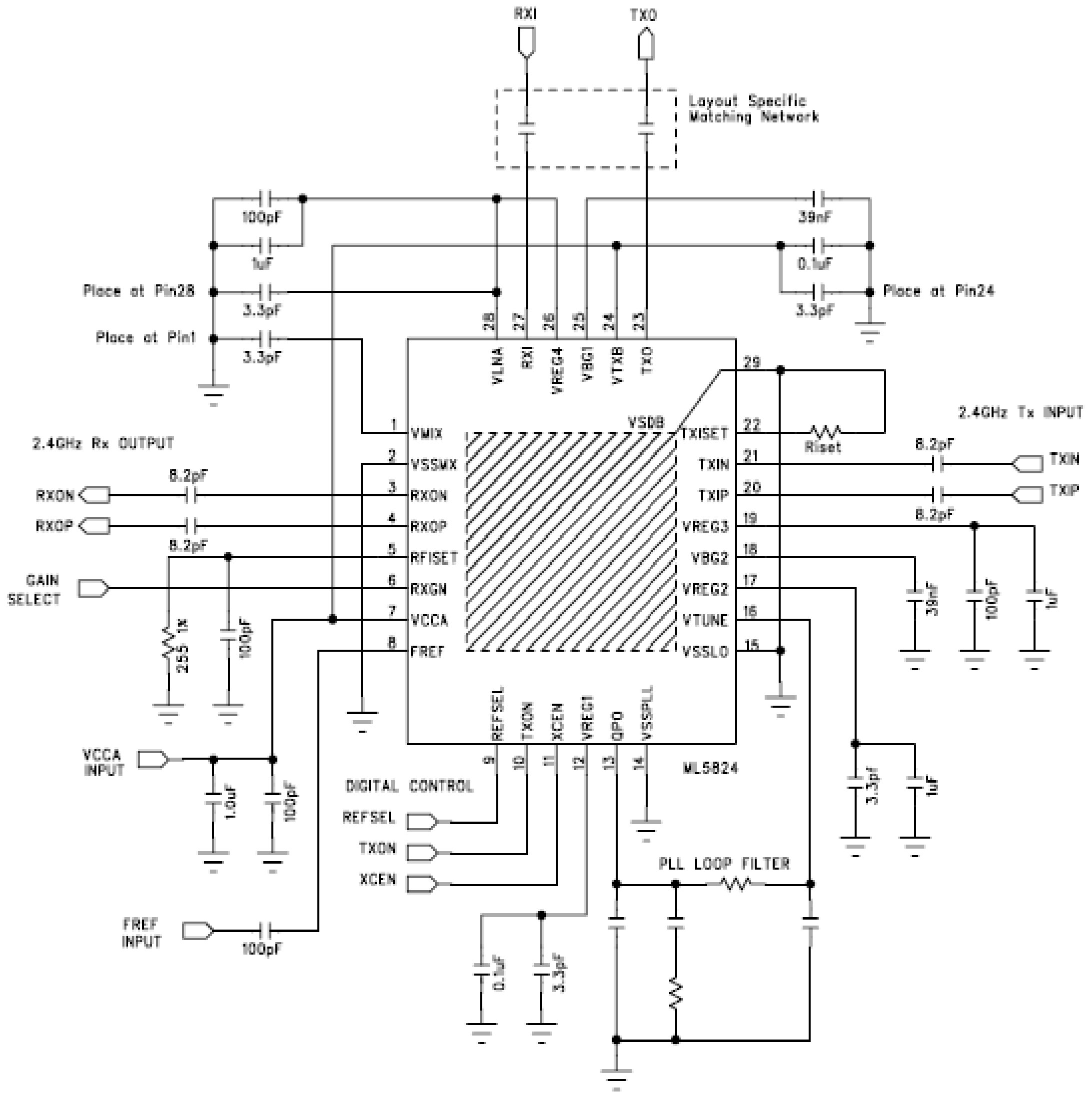


Figure-(6.16) ML5824 glue circuits

CHAPTER 7

ANTENNA ARRAY DESIGN AND FABRICATION

7.1 Introduction

An antenna array consists of a set of antenna elements that are spatially distributed at known locations with reference to a common fixed point [21]. By changing the phase and amplitude of the exciting currents in each of the antenna elements, it is possible to electronically scan the main beam and/or place nulls in any direction.

The antenna elements can be arranged in various geometries, with linear, circular and planar arrays being very common. In the case of a linear array, the centers of the elements of the array are aligned along a straight line. If the spacing between the array elements is equal, it is called a uniformly spaced linear array. A circular array is one in which the centers of the array elements lie on a circle. In the case of a planar array, the centers of the array elements lie on a single plane. Both the linear array and circular array are special cases of the planar array. Arrays whose element locations conform to a given non-planar surface are called conformal arrays.

The radiation pattern of an array is determined by the radiation pattern of the individual elements, their orientation and relative positions in space, and the amplitude and phase of the feeding currents. If each element of the array is an isotropic point source, then the radiation pattern of the array will depend solely on the geometry and feeding current of the array, and the radiation pattern so obtained is called the array factor. If each of the elements of the array is similar but non-isotropic, by the principle of pattern multiplication, the radiation pattern can be computed as a product of the array factor and the individual element pattern [22].

7.2 Uniformly Spaced Linear Array

Consider a K-element uniformly spaced linear array which is illustrated in Figure-(7.1). In Figure-(7.1), the array elements are equally spaced by a distance *d*, and a plane wave arrives at the array from a direction θ of the array broadside. The angle θ is called the *direction-of-arrival* (DOA) or *angle-of-arrival* (AOA) of the received signal, and is measured clockwise from the broadside of the array [23]. The received signal at the first element may be expressed as:

$$y_1(t) = m(t)\cos wt \tag{7.1}$$

$$y_1(t) = m(t)\cos(2\pi f_c t) \tag{7.2}$$

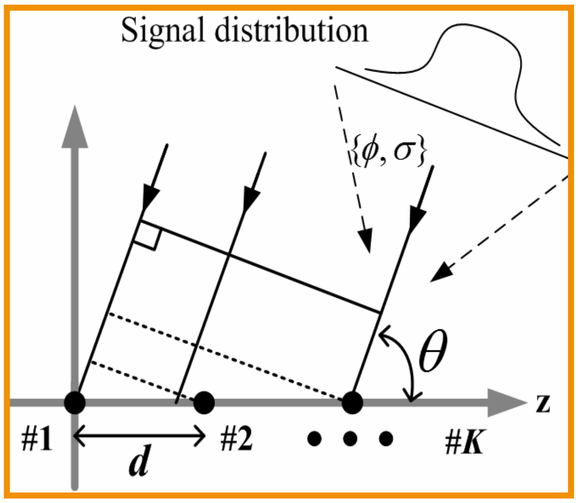


Figure-(7.1) linear array antenna

Where the carrier frequency of the modulated signal is f_c and $m(t)$ is the amplitude of the signal. The complex envelope of $y1(t)$ is given by:

$$y_1(t) = m(t)e^{jwt} \tag{7.3}$$

And the received signal of the second element may be expressed as

$$y_2(t) = m(t-\tau)\cos w(t-\tau) \tag{7.4}$$

If the carrier frequency f_c is large compared to the bandwidth of the signal ($\tau \ll t$), then the modulating signal in above equation reduces to:

$$y_2(t) = m(t)\cos w(t-\tau) \tag{7.5}$$

The complex envelope of $y2(t)$ is therefore given by

$$y_2(t) = m(t)e^{jw(t-\tau)} = m(t)e^{j(wt - w\tau)}$$
$$y_2(t) = m(t)e^{j(2\pi f_c t - 2\pi f_c \tau)}$$
$$y_2(t) = m(t)e^{j2\pi f_c t}e^{-j2\pi f_c \tau} \tag{7.6}$$

Comparing (7.6) with (7.3):

$$y_2(t) = y_1(t)e^{-j2\pi f_c \tau} \tag{7.7}$$

We have used the relation between c and f_c, that is, $f_c = \frac{c}{\lambda}$

$$y_2(t) = y_1(t)e^{-j\frac{2\pi c}{\lambda}\tau} \tag{7.8}$$

The signal arrives at second element after $d\sin\theta$ compare to first element so, the time delay is given by

$$\tau = \frac{d\sin\theta}{c} \tag{7.9}$$

The received signal, at the second element is written as equation (7.10)

$$y_2(t) = y_1(t)e^{(-j\frac{2\pi c}{\lambda}\frac{d\sin\theta}{c})}$$

$$y_2(t) = y_1(t)e^{-j(\frac{2\pi}{\lambda}d\sin\theta)} \qquad (7.10)$$

Similarly, for element i, the complex envelope of the received signal may be expressed as

$$y_i(t) = y_1(t)e^{-j(\frac{2\pi}{\lambda}(i-1)d\sin\theta)} \qquad i = 1,...,k. \qquad (7.11)$$

Adding all the element outputs together gives what is commonly referred to as array factor F :

$$F(\theta) = y_1 + y_2 e^{-j\frac{2\pi}{\lambda}d\sin\theta} + ... = \sum_{k=1}^{k} y_k e^{-j\frac{2\pi}{\lambda}(k-1)d\sin\theta} \qquad (7.12)$$

The equation (7.12) can be expressed in terms of vector inner product:

$$F(\theta) = y(t)v(\theta) \qquad (7.13)$$

Where

$$H_C(z) = 1 - z^{-RM} \qquad (7.14)$$

$$v(\theta) = \begin{pmatrix} 1 \\ e^{-j\frac{2\pi}{\lambda}d\sin\theta} \\ \vdots \\ e^{-j\frac{2\pi}{\lambda}(k-1)d\sin\theta} \end{pmatrix} \tag{7.15}$$

The vector $F(\theta)$ is often referred to as the array input data vector, and $v(\theta)$ is called the steering vector. In the equation (7.15), the signal is assumed to be narrowband.

7.3 Antenna array design and fabrication for smart antenna project

The required linear array antenna firstly designed by Microwave software. However, the software were not able to calculate return loss for 16 linear patches properly although it is able to calculate the radiation pattern correctly. Therefore, the CST software with higher reliability is used for array designs.

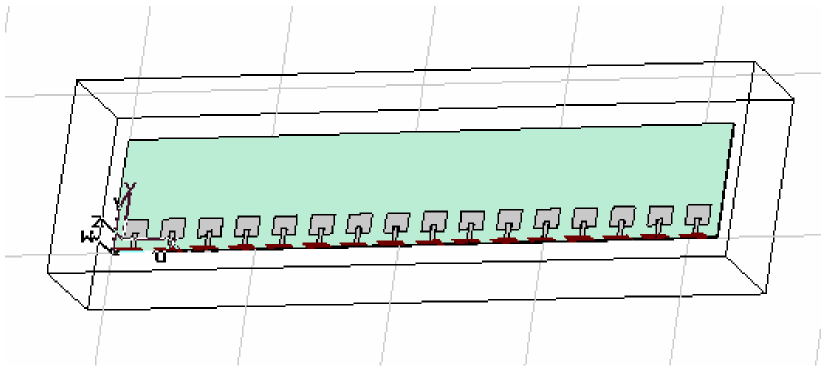


Figure-(7.2) front-view of linear antenna array (CST)

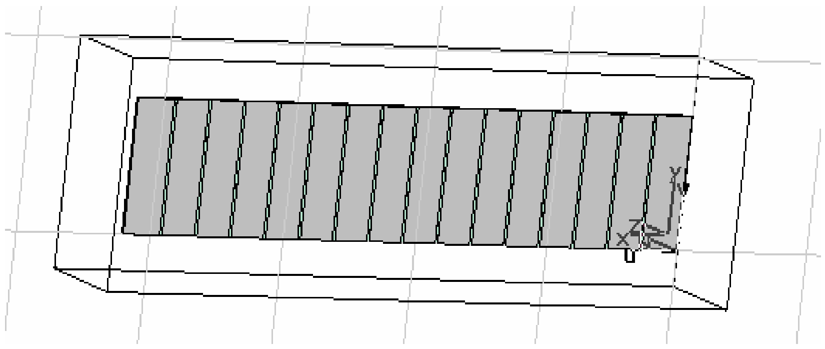
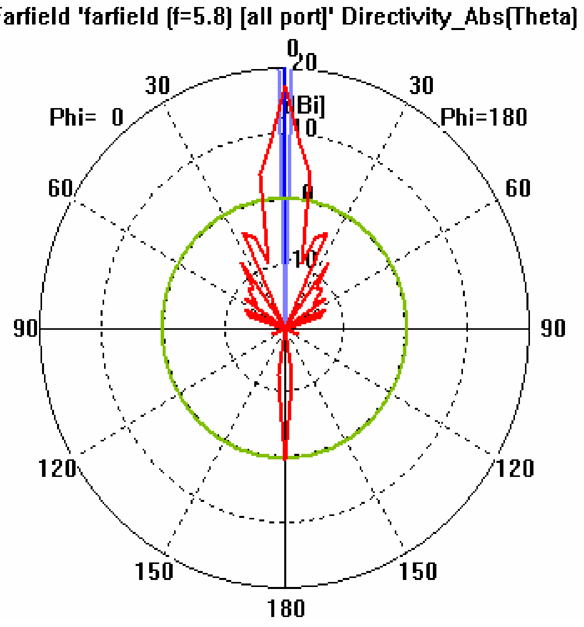
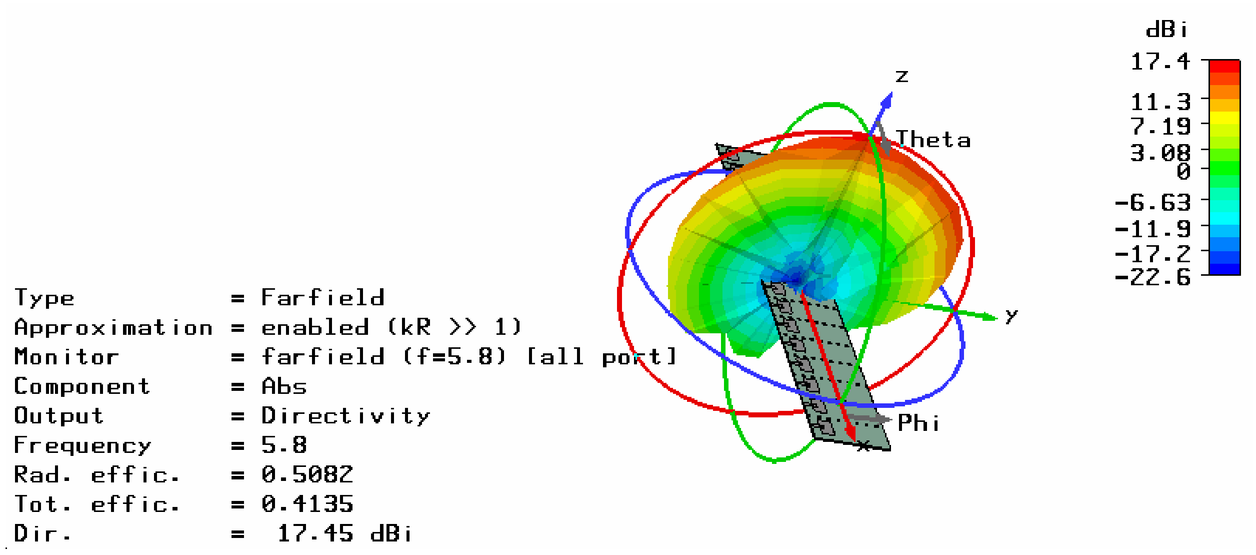


Figure-(7.3) back-view of linear antenna array (CST)



Frequency = 5.8
Main lobe magnitude = 17.4 dBi
Main lobe direction = 0.0 deg.
Angular width (3 dB) = 3.4 deg.
Side lobe level = -17.1 dB

Figure-(7.4) the radiation pattern in direction of o degree



```
Type              = Farfield
Approximation     = enabled (kR >> 1)
Monitor           = farfield (f=5.8) [all port]
Component         = Abs
Output            = Directivity
Frequency         = 5.8
Rad. effic.       = 0.5082
Tot. effic.       = 0.4135
Dir.              =  17.45 dBi
```

Figure-(7.5) the 3D radiation pattern in direction of o degree

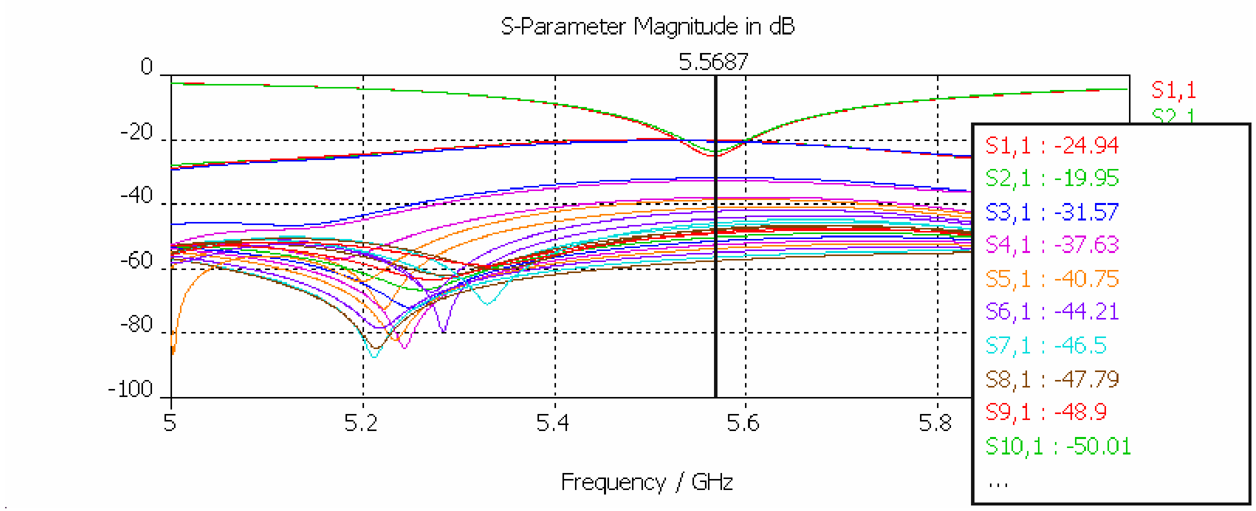


Figure-(7.6) the simulation results for return loss

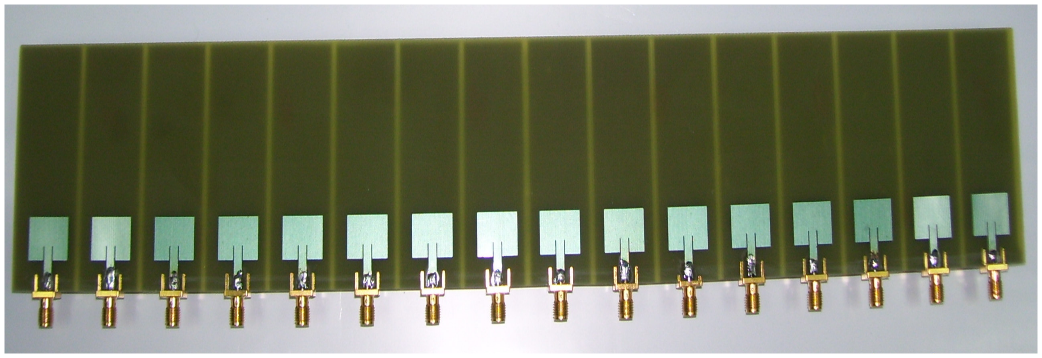


Figure-(7.7) fabricated linear antenna array

The experimental results for return loss for each patch are around 21dB which is not exactly the same as simulation results. However, it can work well in real situation.

CHAPTER 8

SIMULATION RESULTS

8.1 Introduction

In this chapter the simulation results of the project are presented. The results can be classified to the parts as follow: (8.2) result of digital beamforming by using DSP, (8.3) sidelobe cancellation, (8.4) comparisons between the simulation software results and hardware implementation results, (8.5) Channel separation and synchronization by using FPGA, (8.6) the integration of DSP and FPGA board, (8.7) the simulation results of antenna array, (8.8) the conclusion is specifically given for each above part separately. Finally at the end of this chapter discussion and future work are presented.

8.2 Result of digital beamforming by using DSP

The digital beamforming specification can be summarized in table-(8.1). The following result in Figure-(8.1) has been achieved.

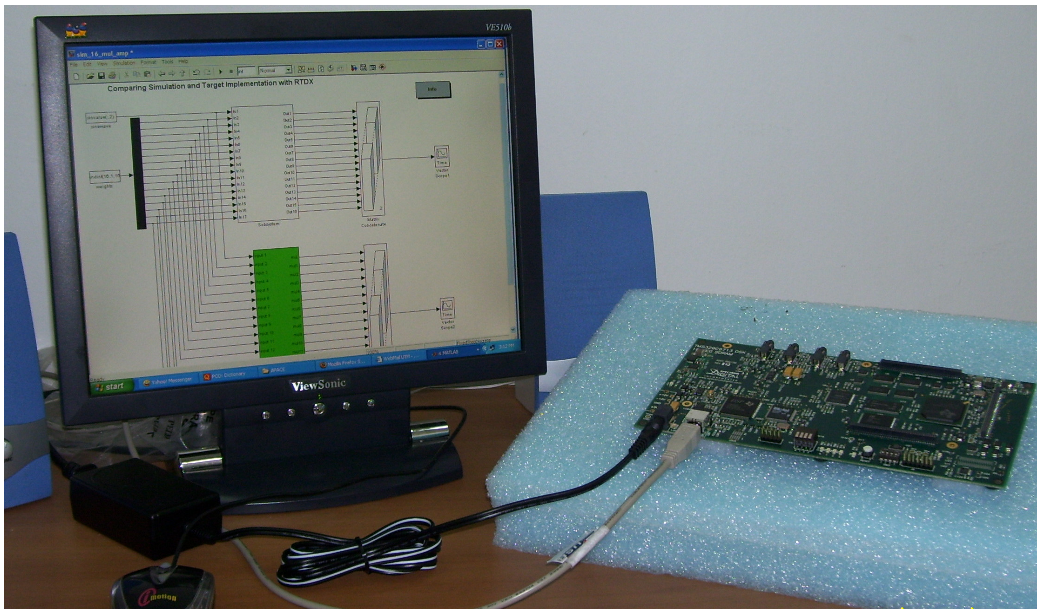


Figure 8.1 the DSP and MATLAB arrangement by PC

Table-(8.1) simulation parameters for beamforming

DSP board	TMS320C6713B
Azimuth angle	30 degree
# channel	16
Sidelobe cancellation technique	Chebyshev algorithms
Antenna array	Uniform linear

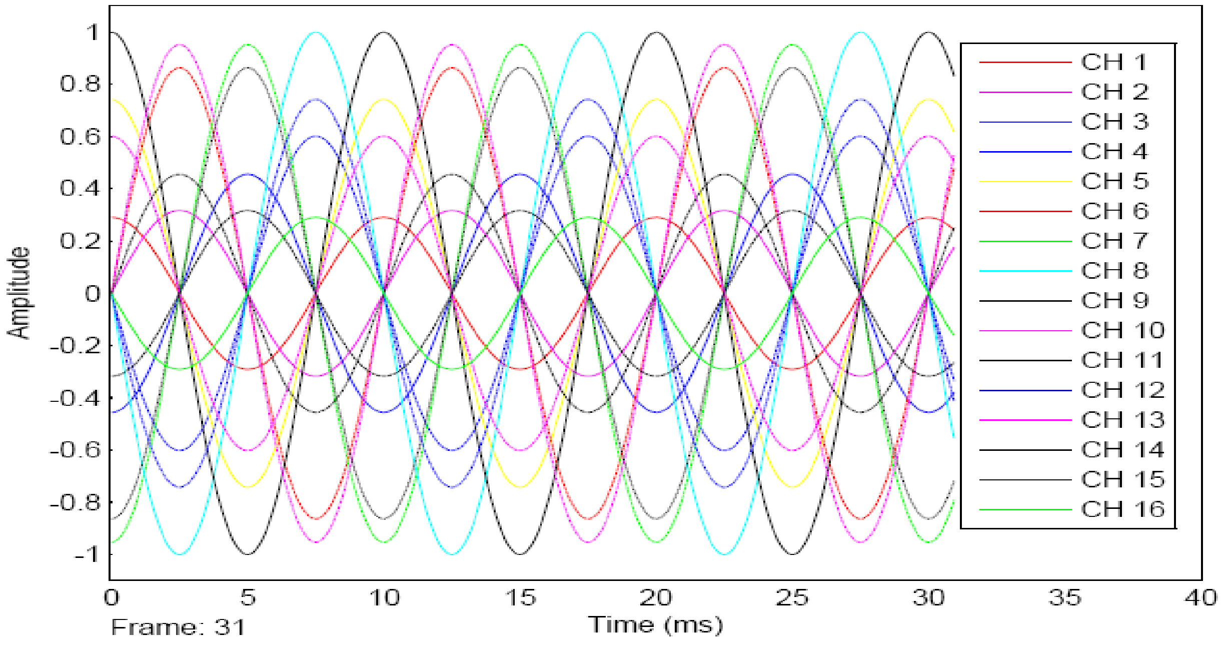
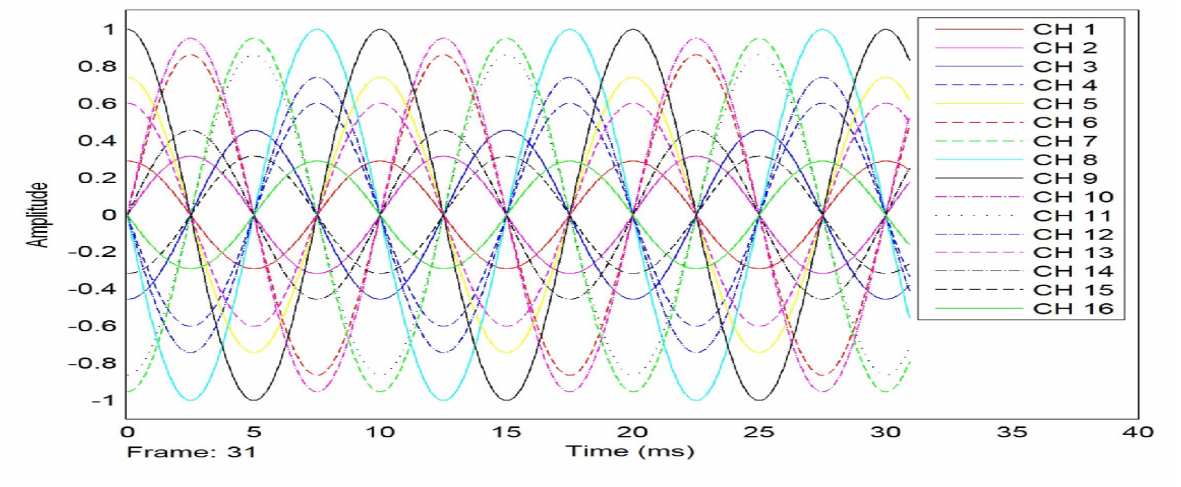


Figure-(8.2) Digital beamforming with DSP board

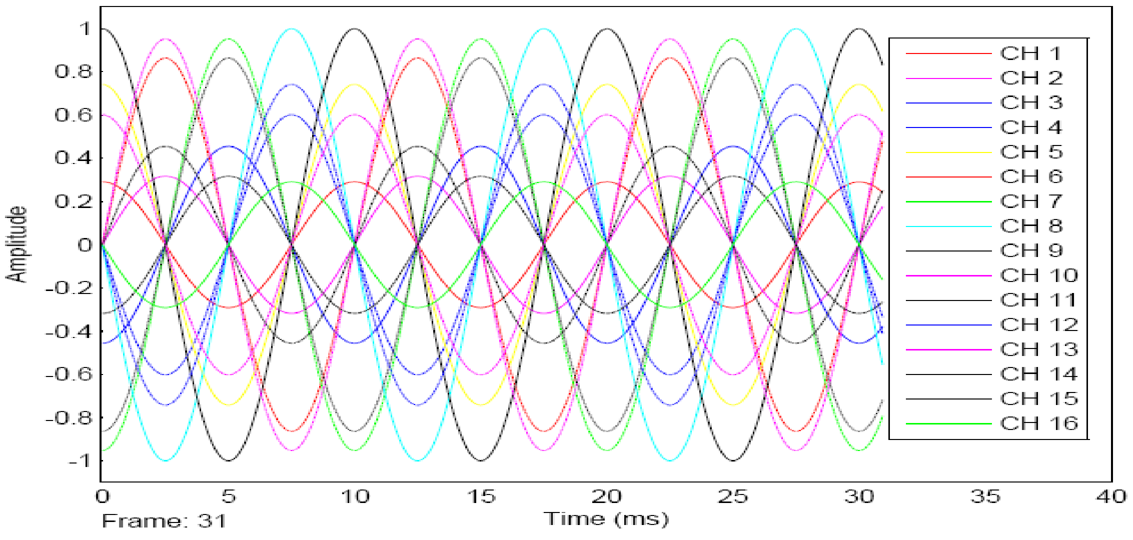
The model for this simulation is available in chapter three. The simulation can be done for each direction. Figure-(8.1) is the output of the beamforming model. The corresponding source code for these is available in Appendix D. As it can be seen from the graph, channels in the middle and in the corner have maximum and minimum amplitude respectively. As an example, the channels number 8 and 9 have maximum amplitudes and channels number 1 and 16 have minimum amplitude. This fact refers to chebyshev algorithm. Also, if you have a look to each wave, you can differentiate the phase difference between them which justify the beamforming process.

8.3 The simulation software results and hardware implementation results

One of the main facilities of DSP board is called Hardware-In-Loop used for comparison between hardware and software results. By using this property of DSP, the correctness of the results can be verified.



(a) MATLAB Result



(b) DSP Result

Figure-(8.3) Comparisons between the software simulation results (a) and hardware implementation results (b)

In this section, the simulation results of digital beamforming algorithm by using MATAB software and the DSP implementation result are compared. Figure-(8.2) indicates that the hardware and software results are the same in terms of amplitude and phases for each channel. So, digital beamforming in DSP board is implemented successfully.

8.4 Antenna array radiation and sidelobe cancellation results

The simulation results below show the antenna array radiation pattern with and without weighting techniques. For beam steering of smart antenna prototype, following simulation results for 16-elements array has been achieved. The beamforming has been performed using signal processing technique. It means that, the required phase delay for each patch for the particular direction has been calculated, and then the signal with different phase feeds to each antenna elements. Fortunately, the results have been accomplished as it has been expected.

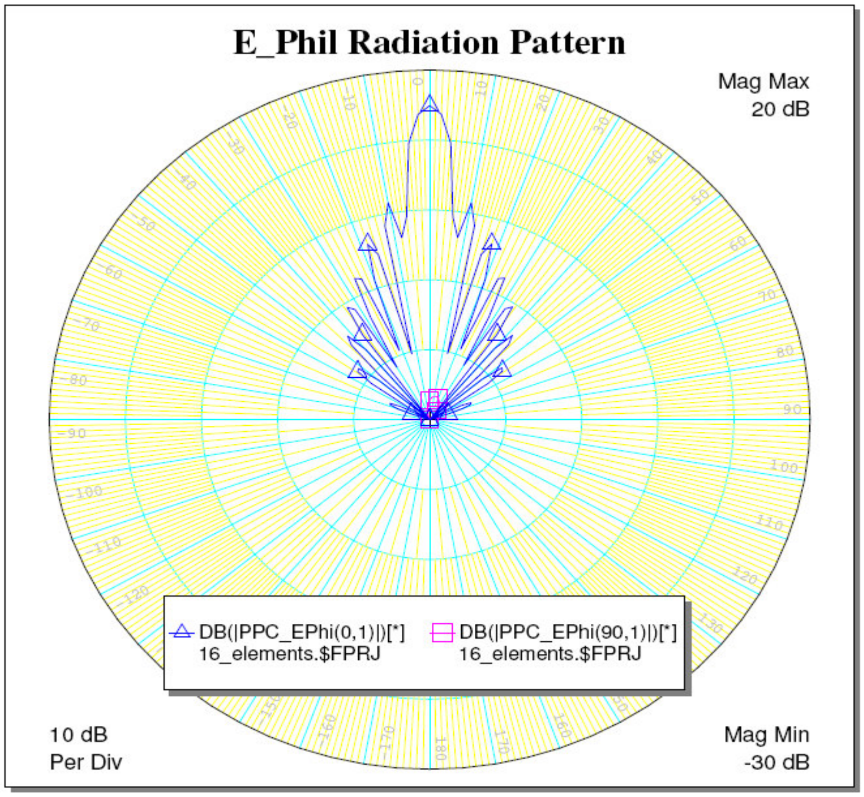


Figure 8.4a 16-elemets zero phase, without weighting

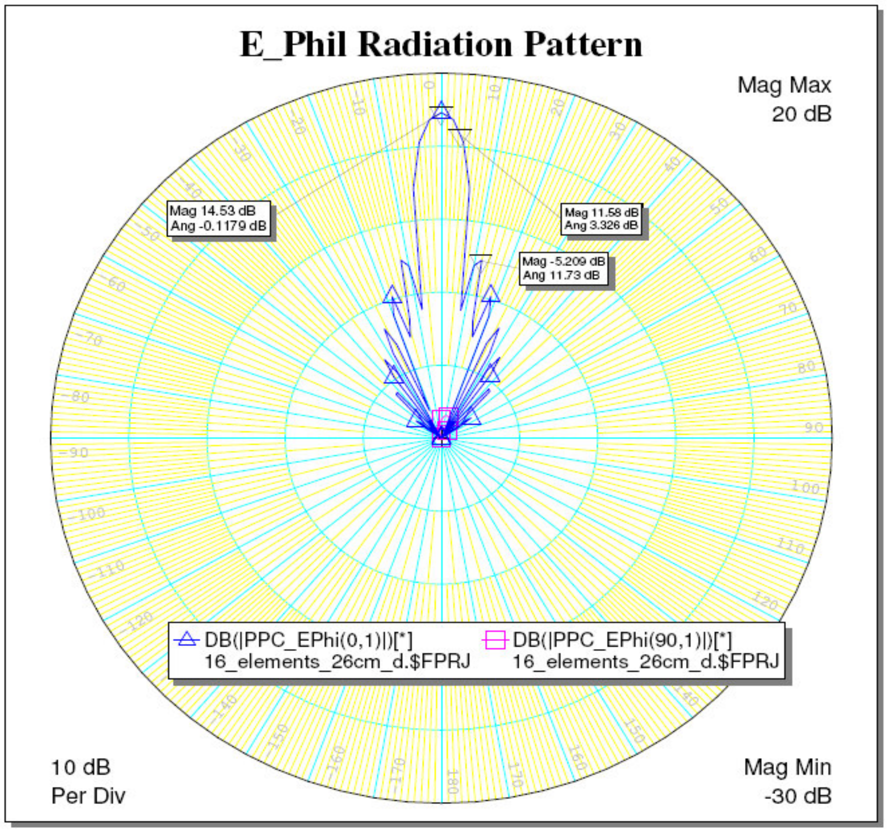


Figure 8.4b 16-elemets zero phase, Kaiser weighting α=3

Figure 8.4b and 8.4a show the antenna pattern with and without sidelobe cancellation respectively. Using Kaiser Technique we can reduce the sidelobe to the desire level. The beamwidth of 16 elements microstrip patch is around 6 degree. Therefore the requirement for 10 degree resolution for the smart antenna project is met. Base on this information we can apply this array to the linear array smart antenna project. Although we need more interfaces and higher DSP processor, there is no alternative unless we go to lower degree of resolution for the system.

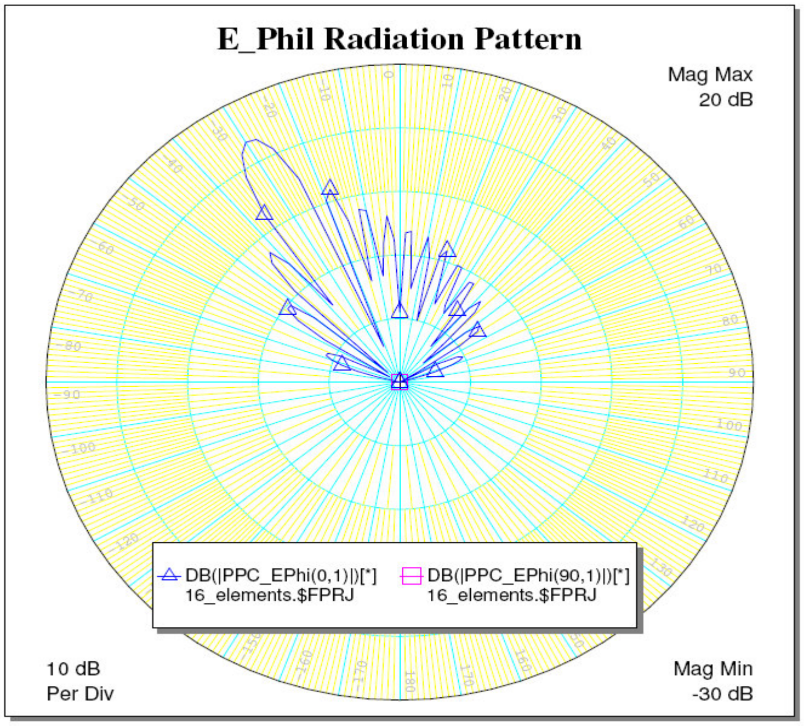


Figure 8.5a 16-elemets -30 phase, without weighting

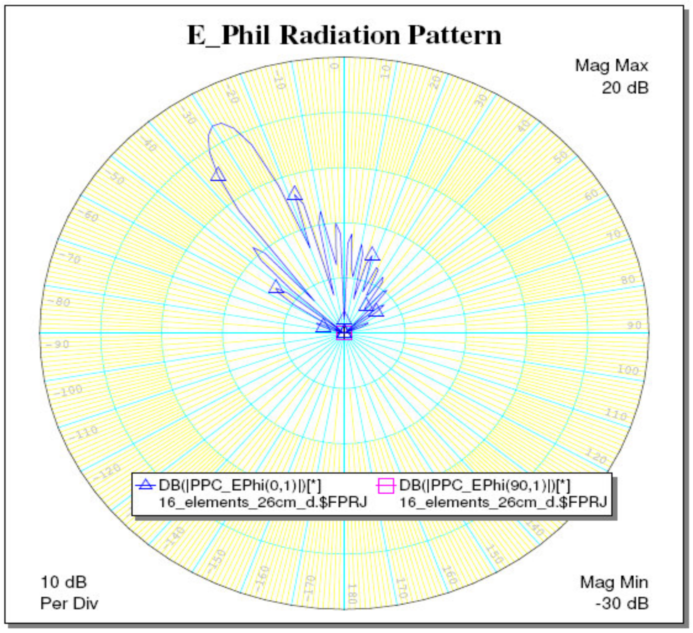


Figure 8.5b 16-elemets -30 phase, Kaiser weighting α=4

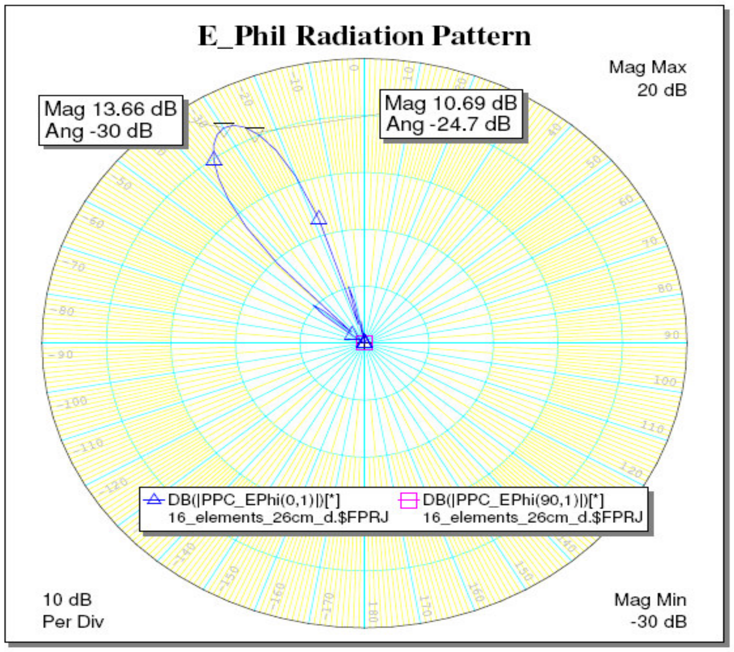


Figure 8.5c 16-elemets, Kaiser weighting for α=8 theta=-30

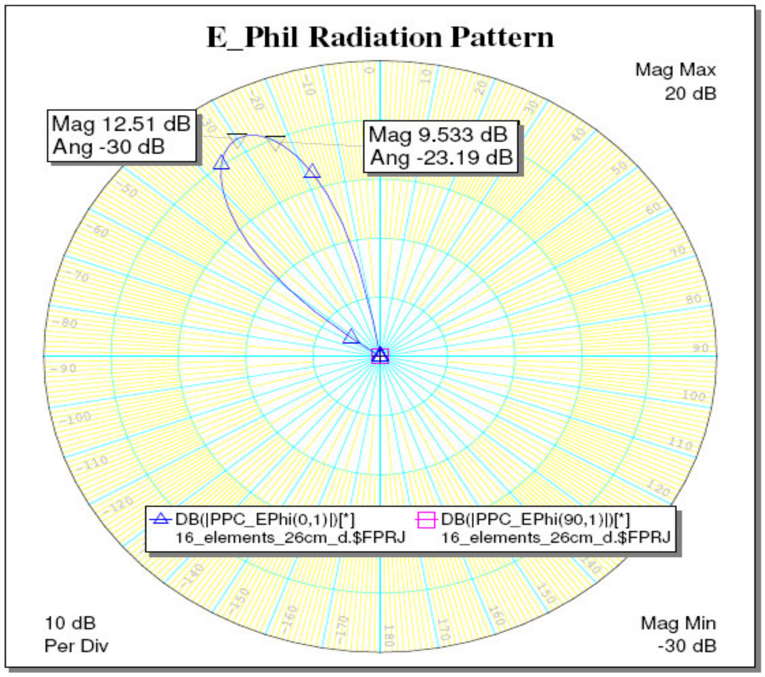


Figure 8.5d 16-elemets, -30 phase, Kaiser weighting α=16

Figure 8.5 shows the results of beamforming for -30 degree azimuth angle. As you can see from this graph the sidelobe can decrease to any desire level at the expense of having wider beamwidth. The weights are calculated using Kaiser formula. As an example base on Kaiser Equation weights for α=8 and α=16 can be calculated as below:

Kaiser weighting for α=8:

W= [14.2604 19.8265 23.6283 26.3342 28.2179 29.4163 30.0000
30.000 29.4163 28.2179 26.3342 23.6283 19.8265 14.2604 3.7623]

Kaiser weighting for α=16:

W= [3.762 14.2604 19.8265 23.6283 26.3342 28.2179 29.4163 30.0000
30.000 29.4163 28.2179 26.3342 23.6283 19.8265 14.2604 3.7623]

8.5 Channel separation and synchronization by using FPGA

In Figure-(8.6), DATA in, clock and FS signals come from DSP to FPGA for splitting data to the channels. After sixteen clocks one sample which has sixteen bits is transferred to FPGA. The VHDL code for this simulation is available in the appendix.

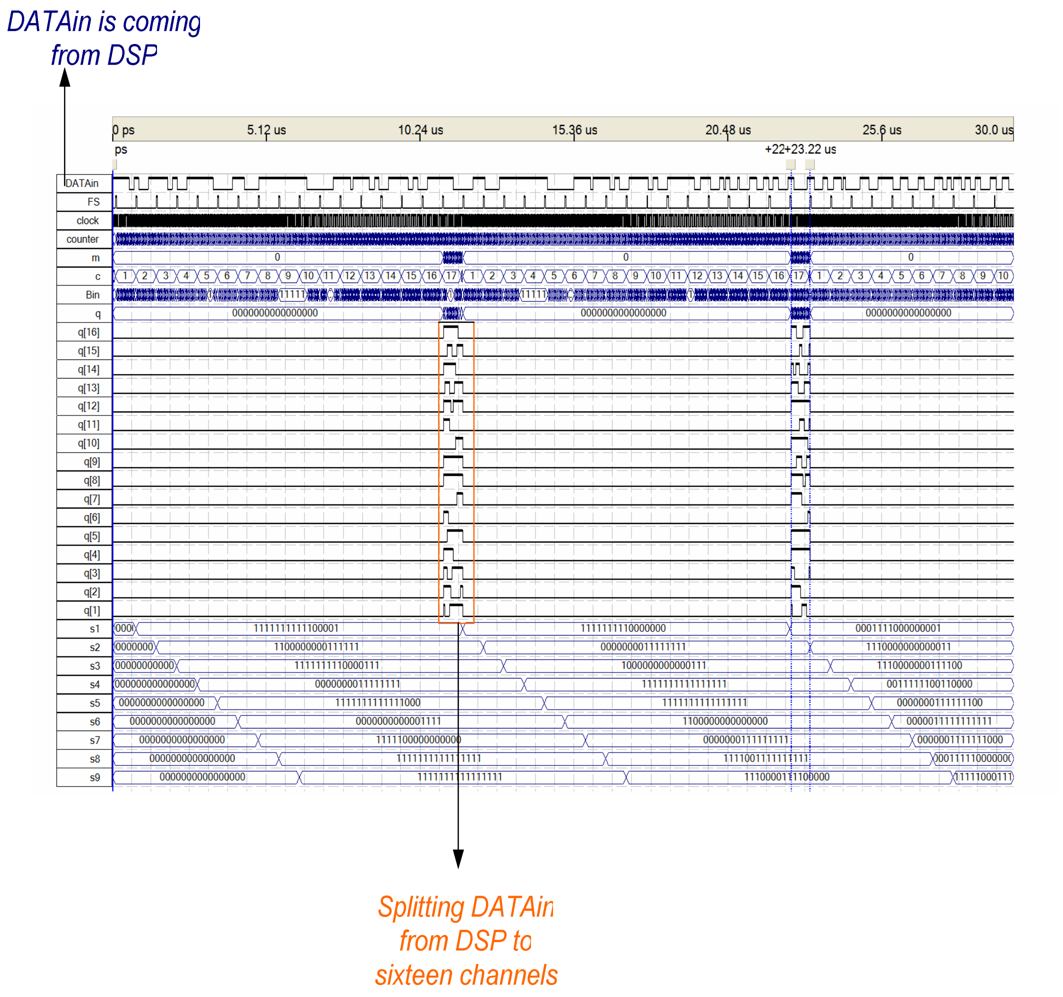


Figure-(8.6) Channel separation and synchronization with FPGA

8.6 Integration of DSP and FPGA board

Figure-(8.7) shows the connection between DSP and FPGA. CPU of DSP processes the data and save them to the SDRAM. Then these data are transferred to the FPGA as it is shown in Figure-(8.4) (blue wire) to split to the sixteen channels. Also clock and FS (Frame Synchronization) go to the FPGA with green and white wires respectively. In addition, four LEDS are used as an indicator. The C source code for the digital beamforming is available in the appendix at end of the report.

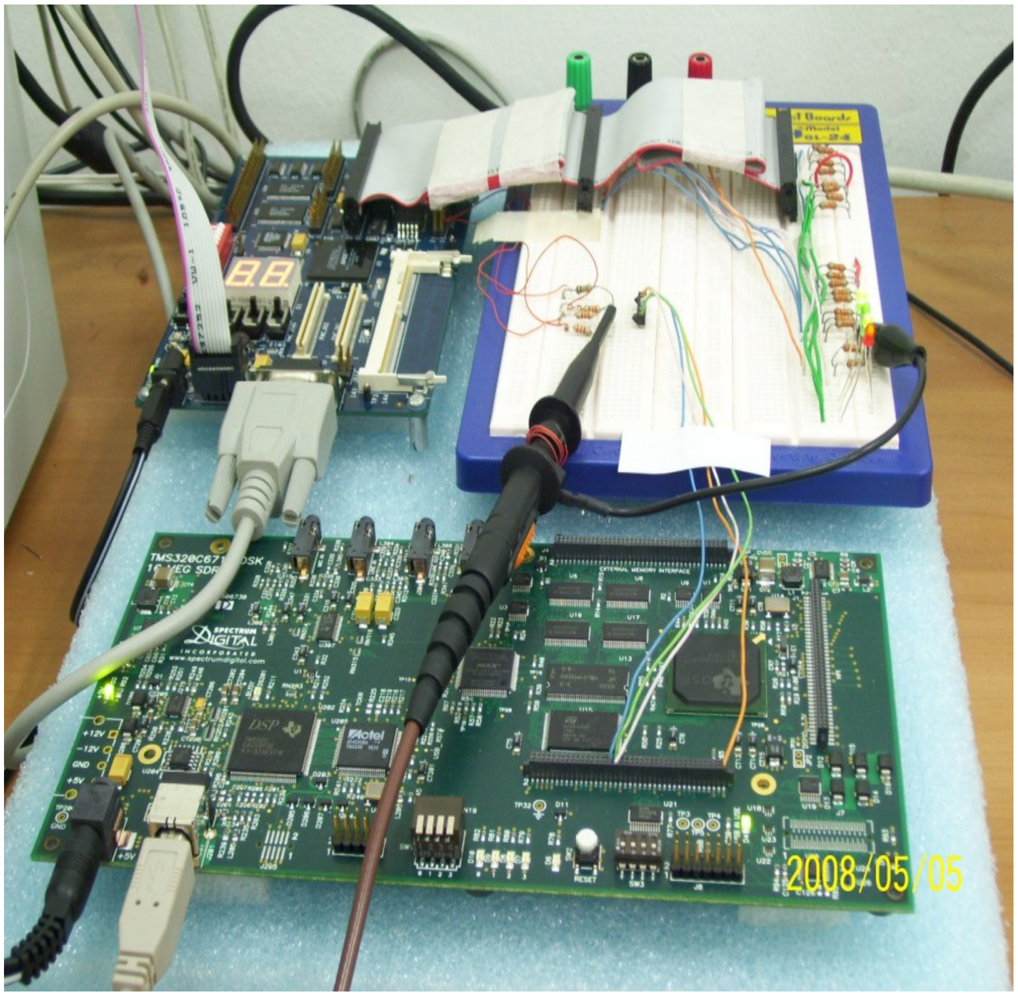


Figure-(8.7) Connection between DSP and FPGA

8.7 Summary and conclusion

In this research, the single-beam switch-beam smart antenna in the baseband frequency is implemented. The popular, up-to-date hardware technology which is integrated DSP and FPGA for this purpose is used. More precisely, the system uses integration of a TMS320C6713B and an Apex-20k200E board for baseband processing. In this respect, baseband processing parts including sidelobe cancellation and digital beamforming is simulated using computer modeling. After modeling, the algorithm is implemented in DSP board. The simulation results and hardware signal measurement prove that digital beamforming and sidelobe cancellation can be done by using DSP board.

Regardless of DSP efficiency, it has only two high data rate serial ports. Because of this, the FPGA is used to split the channels into sixteen antenna array elements. For splitting channels into sixteen antenna array elements, channel separation and synchronization is successfully implemented by programming the FPGA. By connecting this system to a multi-channel RF chain, narrower beams are formed towards the desired user and nulls towards interfering users.

8.8 Future works

One part of this project is about the implementation of digital beamforming and sidelobe cancellation for single user on the downlink transmission. The future work can be defined as the digital beamforming on the uplink transmission and Multiple-beam digital beamforming for smart antenna system. In hardware implementation part for baseband processing, the FPGA board can be used instead of DSP for beamforming to decrease the cost and complexity of the system. In addition, the IF and RF implementation are very challenging and essential to research. Antenna array part is another area in smart antenna system which should be taken into the account. In overall, there are lots of open fields in smart antenna systems which are attractive to research.

References

[1] N. Herscovici and chr.Christodoulou "smart antennas." lEEE Antennas and Propagation Magazine, Vol. 42, No. 3, June 2000.

[2] N. Herscovici and chr.Christodoulou "Smart-Antenna Systems for Mobile Communication Networks Part I: Overview and Antenna Design." IEEE Antenna's and Propagation Magazine, Vol. 44, No. 3, June 2002.

[3] Ramesh Chembil Palat, Dr. Raqibul Mostafa, Dr. Jeffrey H. Reed,"Smart Antennas: A System Level Overview for Software Defined Radios for Creating an API", SDRF-04-I -0057-V0.00 2004.

[4] K. Raith and J. Uddenfeldt, "Capacity of digital cellular TDMA systems," IEEE Trans. Veh. Technolog,,vol. 40, pp. 323-332, 1991.

[5] K. S. Gilhousen, I. M. Jacobs, R. Padovani, A. J. Viterbi, L. A. Weaver, Jr., and C. E. Wheatley 111, "On the capacity of cellular CDMA system," IEEE Trans. Veh. Technol., vol. 40, pp. 303-312, 1991

[6] S. Sivanand, "On adaptive arrays in mobile communication," in Proc. IEEE Nut. Telesystems ConJ, Atlanta, GA, 1993, pp. 55-58.

[7] R. Janaswamy, Radiowave Propagation and Smart Antennas for Wireless Communications. Kluwer Academic Publishers, 2001.

[8] J. L. Butler, "Digital, matrix and intermediate frequency scanning," in Microwave Scanning Antennas, R. C. Hansen, Ed. Academic Press, 1966, vol. 3, ch. 3.

[9] B. Pattan, Robust Modulations Methods and Smart Antennas in Wireless Communications. Prentice Hall PTR, 2000.

[10] I. Stevanovic, A. Skrivervik, and J. R. Mosig, "Smart antenna systems for mobile communications," Ecole Polytechnique Federale De Lausanne, Tech. Rep., Jan. 2003.

[11] J. H.Winters and M. J. Gans, "The range increase of adaptive versus phased arrays in mobile radio systems," IEEE Transactions on Vehicular Technology, vol. 48, no. 2, pp. 353–362, Mar. 1999.

[12] Raqibul Mostafa, "Feasibility of Smart Antennas for Small Wireless Terminals", dissertation submitted to Virginia Tech, April 2003.

[13] Sheikh, K.; Gesbert, D.; Gore, D.; Paulraj, A, "Smart antennas for broadband wireless access networks," IEEE Communications Magazine, Volume: 37 , Issue: 11 , Nov. 1999, Pages:100 – 105.

[14] Kaizhi Huang; Jing Wang; Guoan Chen; Youzheng Wang "Smart antenna and spatial diversity-combining," 55th Vehicular Technology Conference, 2002. Volume: 1 , 6-9 May 2002 Pages:340 – 344.

[15] Shafi, M.; Gesbert, D.; Da-shan Shiu; Smith, P.J.; Tranter, W.H. "Guest editorial MIMO systems and applications. II," Selected Areas in Communications, IEEE Journal on , Volume: 21 , Issue: 5 , June 2003 Pages:681 – 683.

[16] A. Paulraj, R. Nabar, and D. Gore, "Introduction to Space-Time Wireless Communications", Cambridge Univ. Press, 2003.

[17] G. J. Foschini, G. D. Golden, P. W. Wolnianshy and R. A. Valenzuela, " Simplified processing for high spectral efficiency wireless communication employing multi-element arrays," IEEE Journal of Selected Areas in Communications, vol. 17, no. 11, pp. 1841-1852, Nov. 1999.

[18] LAL C. Godara., "Application of Antenna Arrays to Mobile Communications, Part II: Beam-Forming and Direction-of-Arrival Considerations," Selected Areas in Communications, IEEE Trans. vol. 85, No. 8, August 1997.

[19] Altera Corporation. Nios Embedded Processor Development Board Data Sheet v1.1. March 2001.

[20] Spectrum Digital. Inc. "TMS320C6713 DSK Reference Technical" May 2003

[21] S. U. Pillai "Array Signal Processing." Springer-Verlag, New York, 1989.

[22] W. L. Stutzman and G. A. Thiele, "Antenna Theory and Design." John Wiley & Sons, New York, 1981.

[23] Darren S. Goshi, Yuanxun Wang, and Tatsuo Itoh, "Simulation of Adaptive Array Algorithms for CDMA Systems," IEEE Transactions on microwave theory and techniques, vol. 52, No. 12, December 2004.

Appendix A

Digital beamforming C source code:

(i) Main.C

```
/************************************************************/
/* baseband switch beam smart antenna */
/* date: 20.05.08 */
/* written by: Reza Abdolee and Vida Vakilian*/
/* wireless communication centre */
/* University Technology of Malasia */
/************************************************************/
/* FILENAME: main.c */
/* DESCRIPTION: This program performs single-beam digital */
/* beamforming for any user location in azimuth angle, with
*/
/* desired resolution.In addition, the program performs */
/* sidelobe cancellation as much as 30dB to optimize */
/* antenna radiation pattern. The program uses ISR and */
/* McBSP1 to transmit user data countinuesly. */
/* All right are reserved. */
/************************************************************/
/* Header file */
#include <stdio.h>
#include <math.h>
#include "BeamFormcfg.h"
#include "csl.h"
#include "mat.h"
#include "csl_irq.h"
#include "csl_mcbsp.h"
#include "csl_timer.h"
#include "dsk6713.h"
#include "dsk6713_led.h"
#include "dsk6713_dip.h"

/*Declarations*/
#define Num_Antenna ((int)16)
#define Num_Sample ((int)256)
/*Global variables*/
Uint16 CH[Num_Antenna][Num_Sample];
void McBSP1Xmt(void);

/*************************************************************
*****\
* Function: main()
* Description: Enables McBSP1 transmit interrupt
\*************************************************************
*****/
void main()
```

```
{
/* Call BSL init */
DSK6713_init();
DSK6713_rset(DSK6713_MISC, 0x03);
DSK6713_LED_init();
DSK6713_DIP_init();
init();
EDMA_clearChannel(hEdmaCha14);
EDMA_enableChannel(hEdmaCha14);
EDMA_intDisable(14);
EDMA_intClear(14);
EDMA_intEnable(14);
MCBSP_start(hMcbsp1, MCBSP_XMIT_START | MCBSP_SRGR_START|
MCBSP_SRGR_FRAMESYNC,
IRQ_enable(IRQ_EVT_XINT1);
}
/*************************************************************
*******\
* Function: McBSP1Xmt()
* Description: McBSP1 Transmit Interrupt Service Routine.
* Write all channel user data out to the FPGA board.
\*************************************************************
*******/
void McBSP1Xmt(void)
{
/* Wait until a value is received then write it */
while (!MCBSP_xrdy(hMcbsp1));
{
DSK6713_LED_toggle(1);
DSK6713_waitusec(200000);
}
}
/*************************************************************
*******\
* End of main.c
\*************************************************************
*******/
```

(ii) Beamformcfg.C

```c
/* Do *not* directly modify this file. It was */
/* generated by the Configuration Tool; any */
/* changes risk being overwritten. */
/* INPUT BeamForm.cdb */
/* Include Header File */
#include "BeamFormcfg.h"

#ifdef __cplusplus
#pragma CODE_SECTION(".text:CSL_cfgInit")
#else
#pragma CODE_SECTION(CSL_cfgInit,".text:CSL_cfgInit")
#endif
#ifdef __cplusplus
#pragma FUNC_EXT_CALLED()
#else
#pragma FUNC_EXT_CALLED(CSL_cfgInit)
#endif
extern far Uint16 CH[];

/* Config Structures */
EDMA_Config edmaCfg14 = {
0x2B000001, /* Option */
(Uint32) CH, /* Source Address - Extern Decl.Obj */
0x00010010, /* Transfer Counter - Numeric */
0x00000000, /* Destination Address - Numeric */
0x00200002, /* Index register - Numeric */
0x00000000 /* Element Count Reload and Link Address */
};

MCBSP_Config mcbspCfg1 = {
0x00200080, /* Serial Port Control Reg. (SPCR) */
0x00000000, /* Receiver Control Reg. (RCR) */
0x00400F40, /* Transmitter Control Reg. (XCR) */
0x30FF0002, /* Sample-Rate Generator Reg. (SRGR) */
0x00000000, /* Multichannel Control Reg. (MCR) */
0x00000000, /* Receiver Channel Enable(RCER) */
0x00000000, /* Transmitter Channel Enable(XCER) */
0x00000A00 /* Pin Control Reg. (PCR) */
};
/* Handles */
EDMA_Handle hEdmaCha14;
MCBSP_Handle hMcbsp1;

/*
* ======== CSL_cfgInit() ========
*/
void CSL_cfgInit()
{
```

```
hEdmaCha14 = EDMA_open(EDMA_CHA_XEVT1, EDMA_OPEN_RESET);
hMcbsp1 = MCBSP_open(MCBSP_DEV1, MCBSP_OPEN_RESET);
edmaCfg14.dst = EDMA_DST_RMK(hMcbsp1->dxrAddr);
EDMA_config(hEdmaCha14, &edmaCfg14);
E
0x00000000, /* Transmitter Channel Enable(XCER) */
0x00000A00 /* Pin Control Reg. (PCR) */
};
/* Handles */
EDMA_Handle hEdmaCha14;
MCBSP_Handle hMcbsp1;

/*
* ======== CSL_cfgInit() ========
*/
void CSL_cfgInit()
{
hEdmaCha14 = EDMA_open(EDMA_CHA_XEVT1, EDMA_OPEN_RESET);
hMcbsp1 = MCBSP_open(MCBSP_DEV1, MCBSP_OPEN_RESET);
edmaCfg14.dst = EDMA_DST_RMK(hMcbsp1->dxrAddr);
EDMA_config(hEdmaCha14, &edmaCfg14);
EDMA_enableChannel(hEdmaCha14);
MCBSP_config(hMcbsp1, &mcbspCfg1);
}
```

(iii) temp.C

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "mat.h"
#include "dsk6713_led.h"
#include "dsk6713_dip.h"
/*#include <log.h>, it is used when the standard IO for print
and scanf is not used
/*#include "BeamFormcfg.h", this header file is needed since
the log is adjusted in

#define Num_Antenna ((int)16)
#
/*************************************************************
***********/

void init()
{
/* The led indicated that the subroutin executed*/
DSK6713_LED_on(0);
puts("Enter the user location base on azimuth angle in degree
:");
scanf("%d", &user_doa);

/*Verifying the user location*/
printf("The entered user location in angle is : %d\n",
user_doa);
/* base on radian*/
doa_rad=(PI*user_doa)/180;
/*calculating the array factor both real and imaginary part*/
for (c = 0;c < Num_Antenna; c++)
{
w_rl[c]=cos((c-1)*PI*sin(doa_rad));
w_img[c]=-sin((c-1)*PI*sin(doa_rad));
}
/* generating the user data, assume that is a sinosoidal data
streem*/
/*alike data=sin(2*pi*f*n/fs)=cos(2*pi*n/fs-
pi/2)=real(exp(j(2*pi*f*n/fs-pi/

for (b=0;b<Num_Sample;b++)
{
data_rl[b]=sin(2*PI*f*b/fs);
data_img[b]=-cos(2*PI*f*b/fs);
}
/*sidelobe cancellation and digital beamforming */
for (k=0;k<Num_Antenna;k++)
{for (n = 0;n< Num_Sample;n++)
```

```
CH[k][n]=(Uint16)((cheb_coeff[k]*(w_rl[k]*data_rl[n]-
w_img[k]*data_img[n])
/* CH[k][n]= (Uint16) ((sample+1)* data_max);*/
}

}
/*********************End of
temp*************************************/
```

Appendix B

Channel separation using VHDL code:

Date: May 23, 2008 ch_sep.vhd Project: ch_sep
Page 1 of 4 Revision: ch_sep

```
--------------------------------------------------
-- Channel Separtion Module
-- By  Reza Abdolee & Vida Vakilian, 04/2008


---------------------------------------------------

library ieee ;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

---------------------------------------------------

entity ch_sep is
port( datain: in std_logic;
clock: in std_logic;
FS: in std_logic;
CH1,CH2,CH3,CH4,CH5,CH6,CH7,CH8,CH9: out std_logic;
CH10,CH11,CH12,CH13,CH14,CH15,CH16 : out std_logic;
led_CH_TEST,led_clock,led_datain,led_FS : out std_logic;
count_in_value: out std_logic_vector(8 downto 0)

);
end ch_sep;

---------------------------------------------------

architecture behv of ch_sep is

-- initialize the declared signal
signal S: std_logic_vector(255 downto 0);
signal shift_in: std_logic;
signal count_in: std_logic_vector(8 downto 0);
signal count_out: std_logic_vector(7 downto 0);
signal Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8,Q9 : std_logic_vector(15 downto 0);
signal Q10,Q11,Q12,Q13,Q14,Q15,Q16: std_logic_vector(15 downto 0);
signal shift_out,CH_TEST: std_logic;

------------------------------------------------------------
component FS_signal
```

```
port( fs: in std_logic;
clk: in std_logic;
trig: out std_logic

);
end component;

-----------------------------------------------------------------
component puls_show
```

Date: May 23, 2008 ch_sep.vhd Project: ch_sep
Page 2 of 4 Revision: ch_sep

```
port(
P: in std_logic;
led_p: out std_logic


);
end component;

-----------------------------------------------------------------
component shift_reg_16
port(
data: in std_logic_vector(15 downto 0);
clock: in std_logic;
load: in std_logic;
ch_out: out std_logic


);
end component;

-----------------------------------------------------------------
begin



process(datain, clock, shift_in)
begin

-- everything happens upon the clock changing
if clock'event and clock='1' then

if shift_in = '1' then

if (count_in <= x"0FF") then

S <= datain & S(255 downto 1);
count_in <=count_in+1;
count_in_value <=count_in;
shift_out <='0';
else
Q1<=S(15 downto 0);
Q2<=S(31 downto 16);
Q3<=S(47 downto 32);
```

```
Q4<=S(63 downto 48);
Q5<=S(79 downto 64);
Q6<=S(95 downto 80);
Q7<=S(111 downto 96);
Q8<=S(127 downto 112);
Q9<=S(143 downto 128);
Q10<=S(159 downto 144);
Q11<=S(175 downto 160);
Q12<=S(191 downto 176);
Q13<=S(207 downto 192);
```

```
Q14<=S(223 downto 208);
Q15<=S(239 downto 224);
Q16<=S(255 downto 240);

count_in<="000000000";
shift_out<='1';

end if;




end if;
end if;

end process;

U_FS_signal:FS_signal
port map(FS,clock,shift_in);

---------------------------------------------------

U_led_datain: puls_show
port map(datain,led_datain);

U_led_FS: puls_show
port map(FS,led_FS);

U_led_clock: puls_show
port map(clock,led_clock);

CH_TEST<=Q1(0);

U_led_ch_out: puls_show
port map( CH_TEST,led_CH_TEST);
---------------------------------------------------

U_CH1: shift_reg_16
port map(Q1,clock,shift_out,CH1);

U_CH2: shift_reg_16
port map(Q2,clock,shift_out,CH2);
```

```

U_CH3: shift_reg_16
port map(Q3,clock,shift_out,CH3);

U_CH4: shift_reg_16
port map(Q4,clock,shift_out,CH4);

U_CH5: shift_reg_16
port map(Q5,clock,shift_out,CH5);

U_CH6: shift_reg_16
```

```
port map(Q6,clock,shift_out,CH6);

U_CH7: shift_reg_16
port map(Q7,clock,shift_out,CH7);

U_CH8: shift_reg_16
port map(Q8,clock,shift_out,CH8);

U_CH9: shift_reg_16
port map(Q9,clock,shift_out,CH9);

U_CH10: shift_reg_16
port map(Q10,clock,shift_out,CH10);

U_CH11: shift_reg_16
port map(Q11,clock,shift_out,CH11);

U_CH12: shift_reg_16
port map(Q12,clock,shift_out,CH12);

U_CH13: shift_reg_16
port map(Q13,clock,shift_out,CH13);

U_CH14: shift_reg_16
port map(Q14,clock,shift_out,CH14);

U_CH15: shift_reg_16
port map(Q15,clock,shift_out,CH15);

U_CH16: shift_reg_16
port map(Q16,clock,shift_out,CH16);
------------------------------------------------------



end behv;
```

Appendix C:

Channel separation using Verilog code:

```
module channel_separation4  (DATAin,clock,clk_out,q,m,c,Bin,FS,
run_led,LED_clock,LED_FS,LED_DATAin,FS_out,DATAin_out);
input DATAin,clock,FS;
output clk_out,FS_out,DATAin_out,run_led,LED_clock,LED_FS,LED_DATAin;
output [15:0] m,c;
output [15:0] Bin;
output [16:1] q;
integer k,counter,m,c,Ready;
reg [15:0] s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15,s16,
Bin,DATAin1;
reg [16:1] q;
reg run_led;

PULS_show U1_PULS_show(FS,LED_FS);
PULS_show U2_PULS_show(clock,LED_clock);
PULS_show U3_PULS_show(Bin[1],LED_DATAin);


always@(posedge clock)

  begin
   run_led<=1;
    if(FS==1)
       begin
        Ready<=Ready+1;
        c<=c+1;
        q<=0;
        m<=0;
           case(counter)
            16:  s1<=Bin;
            33:  s2<=Bin;
            50:  s3<=Bin;
            67:  s4<=Bin;
            84:  s5<=Bin;
           101:  s6<=Bin;
           118:  s7<=Bin;
           135:  s8<=Bin;
           152:  s9<=Bin;
           169:  s10<=Bin;
           186:  s11<=Bin;
           203:  s12<=Bin;
           220:  s13<=Bin;
           237:  s14<=Bin;
           254:  s15<=Bin;
           271:  s16<=Bin;
          endcase
        end


   if(Ready>=1)
     begin
     counter<=counter+1;
         if(FS==0)
          begin
            DATAin1<=DATAin;
            Bin[0]<=DATAin1; //serial to parallel//
            for (k=1;k<16;k=k+1)
            Bin[k]<=Bin[k-1];
          if(c==17)
           begin
            m<=m+1;

                q[1]<=s1[m];
                q[2]<=s2[m];
                q[3]<=s3[m];
                q[4]<=s4[m];
                q[5]<=s5[m];
                q[6]<=s6[m];
                q[7]<=s7[m];
                q[8]<=s8[m];
                q[9]<=s9[m];
                q[10]<=s10[m];
                q[11]<=s11[m];
                q[12]<=s12[m];
                q[13]<=s13[m];
                q[14]<=s14[m];
                q[15]<=s15[m];
                q[16]<=s16[m];

          end
            if (m==15)
             c<=0;


            if(counter==272)
              counter<=1;

         end
       end

    end

   assign clk_out=5*clock;
   assign FS_out=5*FS;
   assign DATAin_out=DATAin;

 endmodule
```

Appendix D

```c
/*
* File: rtdx_com_16mul_main.c
*
* Real-Time Workshop code generated for Simulink model rtdx_com_16mul.
*
* Model version : 1.184
* Real-Time Workshop file version : 6.6 (R2007a) 01-Feb-2007
* Real-Time Workshop file generated on : Wed May 14 12:14:40 2008
* TLC version : 6.6 (Jan 16 2007)
* C source code generated on : Wed May 14 12:14:41 2008
*/

#include "rtdx_com_16mul.h"
#include "rtdx_com_16mul_private.h"
#include "rtdx_com_16mulcfg.h"
#include "rtwtypes.h"
#include "MW_c6xxx_csl.h"
#include "c6000_main.h"
#include <stdio.h>
#define DSK_CPLD_BASE 0x90080000
#define DSK_USER_REG 0

/* Function: exitprocessing --------------------------------
*
* Abstract:
* Perform various tasks at program exit.
*/
void exitprocessing()
```

```
{
disable_interrupts();
UTL_halt();
}

extern void TSK_prolog(TSK_Handle hTask);
extern void TSK_epilog(TSK_Handle hTask);

//
// TSK prolog/epilog functions.
//
void TSK_prolog(TSK_Handle hTask)
{

#ifdef ENET_SOCKET_CALLS

fdOpenSession( hTask );

#endif

}

void TSK_epilog(TSK_Handle hTask)
{

#ifdef ENET_SOCKET_CALLS

fdCloseSession( hTask );

#endif
```

```

}

//
// This task is run at the highest priority. It is used to
// initialize the model and also to monitor stopping conditions.
// OS executes this task immidiatey after falling out of main().
//
void initTerminateTSK_fcn(void)
{
rtdx_com_16mul_initialize(1);
enable_interrupts();
configureTimers();

/* Wait for a stopping condition. */
SEM_pend(&stopSEM, SYS_FOREVER);

/* We have acquired the STOP semaphore. Perform model
termination. */
/* Suspend syncronous tasks */
{
TSK_epilog( &tBaseRateTSK );
TSK_setpri( &tBaseRateTSK, -1 );
}

LOG_printf(&LOG_MW1, "**stopping the model**");

/* Disable rt_OneStep() here */

/* Terminate model */
```

```
rtdx_com_16mul_terminate();
targetTerminate();
}

void tBaseRateTSK_fcn(void)
{
volatile boolean_T noErr;
TSK_prolog( TSK_self() );
noErr =
rtmGetErrorStatus(rtdx_com_16mul_M) == NULL;
while (noErr ) {
/* Wait for the next timer interrupt */
SEM_pend(&rtClockSEM, SYS_FOREVER);
rtdx_com_16mul_step();
noErr =
rtmGetErrorStatus(rtdx_com_16mul_M) == NULL;
} /* while */

SEM_post(&stopSEM);
}

void main(void)
{
turnOn_L2Cache();
LOG_printf(&LOG_MW1, "**starting the model**");

/* Drop out of main() and enter DSP/BIOS Kernel */
}
```

```
/*
* File: rtdx_com_16mul_data.c
*
* Real-Time Workshop code generated for Simulink model
rtdx_com_16mul.
*
* Model version : 1.184
* Real-Time Workshop file version : 6.6 (R2007a) 01-Feb-2007
* Real-Time Workshop file generated on : Wed May 14 12:14:40 2008
* TLC version : 6.6 (Jan 16 2007)
* C source code generated on : Wed May 14 12:14:41 2008
*/

#include "rtdx_com_16mul.h"
#include "rtdx_com_16mul_private.h"

/* Block parameters (auto storage) */

#pragma DATA_ALIGN(rtdx_com_16mul_P, 8)

Parameters_rtdx_com_16mul rtdx_com_16mul_P = {
0.0F, /* FromRTDX1_IC : '<Root>/From RTDX1'
*/

{ 'a', 'a', 'a', 'a' }, /* FromRTDX1_IC : '<Root>/From RTDX1'
*/
0.0F, /* FromRTDX3_IC : '<Root>/From RTDX3'
*/

{ 'a', 'a', 'a', 'a' }, /* FromRTDX3_IC : '<Root>/From RTDX3'
*/
0.0F, /* FromRTDX2_IC : '<Root>/From RTDX2'
*/

{ 'a', 'a', 'a', 'a' }, /* FromRTDX2_IC : '<Root>/From RTDX2'
*/
0.0F, /* FromRTDX4_IC : '<Root>/From RTDX4'
*/

{ 'a', 'a', 'a', 'a' } /* FromRTDX4_IC : '<Root>/From RTDX4'
*/
};
```

```
/*
* File: rtdx_com_16mul.c
*
* Real-Time Workshop code generated for Simulink model rtdx_com_16mul.
*
* Model version : 1.184
* Real-Time Workshop file version : 6.6 (R2007a) 01-Feb-2007
* Real-Time Workshop file generated on : Wed May 14 12:14:40 2008
* TLC version : 6.6 (Jan 16 2007)
* C source code generated on : Wed May 14 12:14:41 2008
*/

```

```
#include "rtdx_com_16mul.h"
#include "rtdx_com_16mul_private.h"

RTDX_CreateInputChannel(sin_real); /* Channel sin_real for block <Root>/From RTDX1 */
RTDX_CreateInputChannel(real_); /* Channel real_ for block <Root>/From RTDX3 */
RTDX_CreateInputChannel(sin_img); /* Channel sin_img for block <Root>/From RTDX2 */
RTDX_CreateInputChannel(img_); /* Channel img_ for block <Root>/From RTDX4 */
RTDX_CreateOutputChannel(r1); /* Channel r1 for block <S3>/To RTDX1 */
RTDX_CreateOutputChannel(i1); /* Channel i1 for block <S3>/To RTDX2 */
RTDX_CreateOutputChannel(r2); /* Channel r2 for block <S3>/To RTDX3 */
RTDX_CreateOutputChannel(i2); /* Channel i2 for block <S3>/To RTDX4 */
RTDX_CreateOutputChannel(r3); /* Channel r3 for block <S3>/To RTDX5 */
RTDX_CreateOutputChannel(i3); /* Channel i3 for block <S3>/To RTDX6 */
RTDX_CreateOutputChannel(r4); /* Channel r4 for block <S3>/To RTDX7 */
RTDX_CreateOutputChannel(i4); /* Channel i4 for block <S3>/To RTDX8 */
RTDX_CreateOutputChannel(r5); /* Channel r5 for block <S3>/To RTDX9 */
RTDX_CreateOutputChannel(i5); /* Channel i5 for block <S3>/To RTDX10 */
RTDX_CreateOutputChannel(r6); /* Channel r6 for block <S3>/To RTDX11 */
RTDX_CreateOutputChannel(i6); /* Channel i6 for block <S3>/To RTDX12 */
RTDX_CreateOutputChannel(r7); /* Channel r7 for block <S3>/To RTDX13 */
RTDX_CreateOutputChannel(i7); /* Channel i7 for block <S3>/To RTDX14 */
RTDX_CreateOutputChannel(r8); /* Channel r8 for block <S3>/To RTDX15 */
RTDX_CreateOutputChannel(i8); /* Channel i8 for block <S3>/To RTDX16 */
RTDX_CreateOutputChannel(r9); /* Channel r9 for block <S3>/To RTDX17 */
RTDX_CreateOutputChannel(i9); /* Channel i9 for block <S3>/To RTDX18 */
RTDX_CreateOutputChannel(r10); /* Channel r10 for block <S3>/To RTDX19 */
RTDX_CreateOutputChannel(i10); /* Channel i10 for block <S3>/To RTDX20 */
RTDX_CreateOutputChannel(r11); /* Channel r11 for block <S3>/To RTDX21 */
RTDX_CreateOutputChannel(i11); /* Channel i11 for block <S3>/To RTDX22 */
RTDX_CreateOutputChannel(r12); /* Channel r12 for block <S3>/To RTDX23 */
RTDX_CreateOutputChannel(i12); /* Channel i12 for block <S3>/To RTDX24 */
RTDX_CreateOutputChannel(r13); /* Channel r13 for block <S3>/To RTDX25 */
RTDX_CreateOutputChannel(i13); /* Channel i13 for block <S3>/To RTDX26 */
RTDX_CreateOutputChannel(r14); /* Channel r14 for block <S3>/To RTDX27 */
RTDX_CreateOutputChannel(i14); /* Channel i14 for block <S3>/To RTDX28 */
RTDX_CreateOutputChannel(r15); /* Channel r15 for block <S3>/To RTDX29 */
RTDX_CreateOutputChannel(i15); /* Channel i15 for block <S3>/To RTDX30 */
RTDX_CreateOutputChannel(r16); /* Channel r16 for block <S3>/To RTDX31 */
RTDX_CreateOutputChannel(i16); /* Channel i16 for block <S3>/To RTDX32 */

/* Block signals (auto storage) */
#pragma DATA_ALIGN(rtdx_com_16mul_B, 8)

BlockIO_rtdx_com_16mul rtdx_com_16mul_B;

/* Real-time model */
RT_MODEL_rtdx_com_16mul rtdx_com_16mul_M_;
RT_MODEL_rtdx_com_16mul *rtdx_com_16mul_M = &rtdx_com_16mul_M_;

/* Model step function */
void rtdx_com_16mul_step(void)
{
```

```
/* local block i/o variables */
real32_T rtb_multiply[16];
real32_T rtb_Subtract[16];

/* S-Function Block: <Root>/From RTDX1 (rtdx_src) */
RTDX_read( &sin_real, (void*) rtdx_com_16mul_B.FromRTDX1,
16*sizeof(real32_T));

/* S-Function Block: <Root>/From RTDX3 (rtdx_src) */
RTDX_read( &real_, (void*) rtdx_com_16mul_B.FromRTDX3,
16*sizeof(real32_T));

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Product: '<S4>/multiply' */
rtb_multiply[i] = rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX3[0];
}
}

/* S-Function Block: <Root>/From RTDX2 (rtdx_src) */
RTDX_read( &sin_img, (void*) rtdx_com_16mul_B.FromRTDX2,
16*sizeof(real32_T));

/* S-Function Block: <Root>/From RTDX4 (rtdx_src) */
RTDX_read( &img_, (void*) rtdx_com_16mul_B.FromRTDX4,
16*sizeof(real32_T));

{
```

```
int32_T i;
for (i = 0; i < 16; i++) {
/* Sum: '<S4>/Subtract' incorporates:
* Product: '<S4>/multiply1'
*/
rtb_Subtract[i] = rtb_multiply[i] - rtdx_com_16mul_B.FromRTDX2[i]
*
rtdx_com_16mul_B.FromRTDX4[0];
}
}

/* S-Function Block: <S3>/To RTDX1 (rtdx_snk) */
if (RTDX_isOutputEnabled( &r1 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &r1, (void*) rtb_Subtract, 16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Sum: '<S4>/Subtract1' incorporates:
* Product: '<S4>/multiply2'
* Product: '<S4>/multiply3'
*/
rtb_Subtract[i] = rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX3[0] +
rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX4[0];
}
}

/* S-Function Block: <S3>/To RTDX2 (rtdx_snk) */
if (RTDX_isOutputEnabled( &i1 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &i1, (void*) rtb_Subtract, 16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Sum: '<S5>/Subtract' incorporates:
* Product: '<S5>/multiply'
* Product: '<S5>/multiply1'
*/
rtb_Subtract[i] = rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX3[1] -
rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX4[1];
}
}

/* S-Function Block: <S3>/To RTDX3 (rtdx_snk) */
if (RTDX_isOutputEnabled( &r2 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

```

```
RTDX_write( &r2, (void*) rtb_Subtract, 16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Sum: '<S5>/Subtract1' incorporates:
* Product: '<S5>/multiply2'
* Product: '<S5>/multiply3'
*/
rtb_Subtract[i] = rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX3[1] +
rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX4[1];
}
}

/* S-Function Block: <S3>/To RTDX4 (rtdx_snk) */
if (RTDX_isOutputEnabled( &i2 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &i2, (void*) rtb_Subtract, 16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Sum: '<S12>/Subtract' incorporates:
* Product: '<S12>/multiply'
* Product: '<S12>/multiply1'
*/
rtb_Subtract[i] = rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX3[2] -
rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX4[2];
}
}

/* S-Function Block: <S3>/To RTDX5 (rtdx_snk) */
if (RTDX_isOutputEnabled( &r3 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &r3, (void*) rtb_Subtract, 16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Sum: '<S12>/Subtract1' incorporates:
* Product: '<S12>/multiply2'
* Product: '<S12>/multiply3'
*/
rtb_Subtract[i] = rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX3[2] +
rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX4[2];
}
}
```

```

/* S-Function Block: <S3>/To RTDX6 (rtdx_snk) */
if (RTDX_isOutputEnabled( &i3 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &i3, (void*) rtb_Subtract, 16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Sum: '<S13>/Subtract' incorporates:
* Product: '<S13>/multiply'
* Product: '<S13>/multiply1'
*/
rtb_Subtract[i] = rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX3[3] -
rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX4[3];
}
}

/* S-Function Block: <S3>/To RTDX7 (rtdx_snk) */
if (RTDX_isOutputEnabled( &r4 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &r4, (void*) rtb_Subtract, 16*sizeof(real32_T));
}
```

```

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Sum: '<S13>/Subtract1' incorporates:
* Product: '<S13>/multiply2'
* Product: '<S13>/multiply3'
*/
rtb_Subtract[i] = rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX3[3] +
rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX4[3];
}
}

/* S-Function Block: <S3>/To RTDX8 (rtdx_snk) */
if (RTDX_isOutputEnabled( &i4 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &i4, (void*) rtb_Subtract, 16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Sum: '<S14>/Subtract' incorporates:
* Product: '<S14>/multiply'
* Product: '<S14>/multiply1'
*/
rtb_Subtract[i] = rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX3[4] -
rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX4[4];
}
}

/* S-Function Block: <S3>/To RTDX9 (rtdx_snk) */
if (RTDX_isOutputEnabled( &r5 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &r5, (void*) rtb_Subtract, 16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Sum: '<S14>/Subtract1' incorporates:
* Product: '<S14>/multiply2'
* Product: '<S14>/multiply3'
*/
rtb_Subtract[i] = rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX3[4] +
rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX4[4];
}
}

/* S-Function Block: <S3>/To RTDX10 (rtdx_snk) */
```

```
if (RTDX_isOutputEnabled( &i5 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &i5, (void*) rtb_Subtract, 16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Sum: '<S15>/Subtract' incorporates:
* Product: '<S15>/multiply'
* Product: '<S15>/multiply1'
*/
rtb_Subtract[i] = rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX3[5] -
rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX4[5];
}
}

/* S-Function Block: <S3>/To RTDX11 (rtdx_snk) */
if (RTDX_isOutputEnabled( &r6 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &r6, (void*) rtb_Subtract, 16*sizeof(real32_T));
}

{
```

```
int32_T i;
for (i = 0; i < 16; i++) {
/* Sum: '<S15>/Subtract1' incorporates:
* Product: '<S15>/multiply2'
* Product: '<S15>/multiply3'
*/
rtb_Subtract[i] = rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX3[5] +
rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX4[5];
}
}

/* S-Function Block: <S3>/To RTDX12 (rtdx_snk) */
if (RTDX_isOutputEnabled( &i6 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &i6, (void*) rtb_Subtract, 16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Sum: '<S16>/Subtract' incorporates:
* Product: '<S16>/multiply'
* Product: '<S16>/multiply1'
*/
rtb_Subtract[i] = rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX3[6] -
rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX4[6];
}
}

/* S-Function Block: <S3>/To RTDX13 (rtdx_snk) */
if (RTDX_isOutputEnabled( &r7 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &r7, (void*) rtb_Subtract, 16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Sum: '<S16>/Subtract1' incorporates:
* Product: '<S16>/multiply2'
* Product: '<S16>/multiply3'
*/
rtb_Subtract[i] = rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX3[6] +
rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX4[6];
}
}

/* S-Function Block: <S3>/To RTDX14 (rtdx_snk) */
if (RTDX_isOutputEnabled( &i7 )) {
```

```
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &i7, (void*) rtb_Subtract, 16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Sum: '<S17>/Subtract' incorporates:
* Product: '<S17>/multiply'
* Product: '<S17>/multiply1'
*/
rtb_Subtract[i] = rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX3[7] -
rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX4[7];
}
}

/* S-Function Block: <S3>/To RTDX15 (rtdx_snk) */
if (RTDX_isOutputEnabled( &r8 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &r8, (void*) rtb_Subtract, 16*sizeof(real32_T));
}

{
int32_T i;
```

```
for (i = 0; i < 16; i++) {
/* Sum: '<S17>/Subtract1' incorporates:
* Product: '<S17>/multiply2'
* Product: '<S17>/multiply3'
*/
rtb_Subtract[i] = rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX3[7] +
rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX4[7];
}
}

/* S-Function Block: <S3>/To RTDX16 (rtdx_snk) */
if (RTDX_isOutputEnabled( &i8 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &i8, (void*) rtb_Subtract, 16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Sum: '<S18>/Subtract' incorporates:
* Product: '<S18>/multiply'
* Product: '<S18>/multiply1'
*/
rtb_Subtract[i] = rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX3[8] -
rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX4[8];
}
}

/* S-Function Block: <S3>/To RTDX17 (rtdx_snk) */
if (RTDX_isOutputEnabled( &r9 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &r9, (void*) rtb_Subtract, 16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Sum: '<S18>/Subtract1' incorporates:
* Product: '<S18>/multiply2'
* Product: '<S18>/multiply3'
*/
rtb_Subtract[i] = rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX3[8] +
rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX4[8];
}
}

/* S-Function Block: <S3>/To RTDX18 (rtdx_snk) */
if (RTDX_isOutputEnabled( &i9 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */
```

```

RTDX_write( &i9, (void*) rtb_Subtract, 16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Sum: '<S19>/Subtract' incorporates:
* Product: '<S19>/multiply'
* Product: '<S19>/multiply1'
*/
rtb_Subtract[i] = rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX3[9] -
rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX4[9];
}
}

/* S-Function Block: <S3>/To RTDX19 (rtdx_snk) */
if (RTDX_isOutputEnabled( &r10 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &r10, (void*) rtb_Subtract, 16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Sum: '<S19>/Subtract1' incorporates:
```

```
* Product: '<S19>/multiply2'
* Product: '<S19>/multiply3'
*/
rtb_Subtract[i] = rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX3[9] +
rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX4[9];
}
}

/* S-Function Block: <S3>/To RTDX20 (rtdx_snk) */
if (RTDX_isOutputEnabled( &i10 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &i10, (void*) rtb_Subtract, 16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Sum: '<S6>/Subtract' incorporates:
* Product: '<S6>/multiply'
* Product: '<S6>/multiply1'
*/
rtb_Subtract[i] = rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX3[10] -
rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX4[10];
}
```

```
}

/* S-Function Block: <S3>/To RTDX21 (rtdx_snk) */
if (RTDX_isOutputEnabled( &r11 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &r11, (void*) rtb_Subtract, 16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Product: '<S6>/multiply3' */
rtdx_com_16mul_B.Subtract1[i] =
rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX4[10];

/* Sum: '<S6>/Subtract1' incorporates:
* Product: '<S6>/multiply2'
*/
rtdx_com_16mul_B.multiply3[i] =
rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX3[10] +
rtdx_com_16mul_B.Subtract1[i];
}
}

/* S-Function Block: <S3>/To RTDX22 (rtdx_snk) */
if (RTDX_isOutputEnabled( &i11 )) {
```

```
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &i11, (void*) rtdx_com_16mul_B.multiply3,
16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Product: '<S7>/multiply' */
rtdx_com_16mul_B.multiply3[i] =
rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX3[11];

/* Product: '<S7>/multiply1' */
rtdx_com_16mul_B.Subtract1[i] =
rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX4[11];

/* Sum: '<S7>/Subtract' */
rtdx_com_16mul_B.multiply2[i] = rtdx_com_16mul_B.multiply3[i] -
rtdx_com_16mul_B.Subtract1[i];
}
}

/* S-Function Block: <S3>/To RTDX23 (rtdx_snk) */
if (RTDX_isOutputEnabled( &r12 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */
```

```

RTDX_write( &r12, (void*) rtdx_com_16mul_B.multiply2,
16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Product: '<S7>/multiply2' */
rtdx_com_16mul_B.multiply2[i] =
rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX3[11];

/* Product: '<S7>/multiply3' */
rtdx_com_16mul_B.multiply3[i] =
rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX4[11];

/* Sum: '<S7>/Subtract1' */
rtdx_com_16mul_B.Subtract1[i] = rtdx_com_16mul_B.multiply2[i] +
rtdx_com_16mul_B.multiply3[i];
}
}

/* S-Function Block: <S3>/To RTDX24 (rtdx_snk) */
if (RTDX_isOutputEnabled( &i12 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

```

```
RTDX_write( &i12, (void*) rtdx_com_16mul_B.Subtract1,
16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Product: '<S8>/multiply' */
rtdx_com_16mul_B.multiply2[i] =
rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX3[12];

/* Product: '<S8>/multiply1' */
rtdx_com_16mul_B.multiply3[i] =
rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX4[12];

/* Sum: '<S8>/Subtract' */
rtdx_com_16mul_B.Subtract1[i] = rtdx_com_16mul_B.multiply2[i] -
rtdx_com_16mul_B.multiply3[i];
}
}

/* S-Function Block: <S3>/To RTDX25 (rtdx_snk) */
if (RTDX_isOutputEnabled( &r13 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &r13, (void*) rtdx_com_16mul_B.Subtract1,
16*sizeof(real32_T));
```

```
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Product: '<S8>/multiply2' */
rtdx_com_16mul_B.multiply2[i] =
rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX3[12];

/* Product: '<S8>/multiply3' */
rtdx_com_16mul_B.multiply3[i] =
rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX4[12];

/* Sum: '<S8>/Subtract1' */
rtdx_com_16mul_B.Subtract1[i] = rtdx_com_16mul_B.multiply2[i] +
rtdx_com_16mul_B.multiply3[i];
}
}

/* S-Function Block: <S3>/To RTDX26 (rtdx_snk) */
if (RTDX_isOutputEnabled( &i13 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &i13, (void*) rtdx_com_16mul_B.Subtract1,
16*sizeof(real32_T));
}

```

```
{
int32_T i;
for (i = 0; i < 16; i++) {
/* Product: '<S9>/multiply' */
rtdx_com_16mul_B.multiply2[i] =
rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX3[13];

/* Product: '<S9>/multiply1' */
rtdx_com_16mul_B.multiply3[i] =
rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX4[13];

/* Sum: '<S9>/Subtract' */
rtdx_com_16mul_B.Subtract1[i] = rtdx_com_16mul_B.multiply2[i] -
rtdx_com_16mul_B.multiply3[i];
}
}

/* S-Function Block: <S3>/To RTDX27 (rtdx_snk) */
if (RTDX_isOutputEnabled( &r14 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &r14, (void*) rtdx_com_16mul_B.Subtract1,
16*sizeof(real32_T));
}

{
int32_T i;
```

```
for (i = 0; i < 16; i++) {
/* Product: '<S9>/multiply2' */
rtdx_com_16mul_B.multiply2[i] =
rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX3[13];

/* Product: '<S9>/multiply3' */
rtdx_com_16mul_B.multiply3[i] =
rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX4[13];

/* Sum: '<S9>/Subtract1' */
rtdx_com_16mul_B.Subtract1[i] = rtdx_com_16mul_B.multiply2[i] +
rtdx_com_16mul_B.multiply3[i];
}
}

/* S-Function Block: <S3>/To RTDX28 (rtdx_snk) */
if (RTDX_isOutputEnabled( &i14 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &i14, (void*) rtdx_com_16mul_B.Subtract1,
16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Product: '<S10>/multiply' */
```

```
rtdx_com_16mul_B.multiply2[i] =
rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX3[14];

/* Product: '<S10>/multiply1' */
rtdx_com_16mul_B.multiply3[i] =
rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX4[14];

/* Sum: '<S10>/Subtract' */
rtdx_com_16mul_B.Subtract1[i] = rtdx_com_16mul_B.multiply2[i] -
rtdx_com_16mul_B.multiply3[i];
}
}

/* S-Function Block: <S3>/To RTDX29 (rtdx_snk) */
if (RTDX_isOutputEnabled( &r15 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &r15, (void*) rtdx_com_16mul_B.Subtract1,
16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Product: '<S10>/multiply2' */
rtdx_com_16mul_B.multiply2[i] =
rtdx_com_16mul_B.FromRTDX2[i] *
```

```
rtdx_com_16mul_B.FromRTDX3[14];

/* Product: '<S10>/multiply3' */
rtdx_com_16mul_B.multiply3[i] =
rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX4[14];

/* Sum: '<S10>/Subtract1' */
rtdx_com_16mul_B.Subtract1[i] = rtdx_com_16mul_B.multiply2[i] +
rtdx_com_16mul_B.multiply3[i];
}
}

/* S-Function Block: <S3>/To RTDX30 (rtdx_snk) */
if (RTDX_isOutputEnabled( &i15 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &i15, (void*) rtdx_com_16mul_B.Subtract1,
16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Product: '<S11>/multiply' */
rtdx_com_16mul_B.multiply2[i] =
rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX3[15];

```

```
/* Product: '<S11>/multiply1' */
rtdx_com_16mul_B.multiply3[i] =
rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX4[15];

/* Sum: '<S11>/Subtract' */
rtdx_com_16mul_B.Subtract1[i] = rtdx_com_16mul_B.multiply2[i] -
rtdx_com_16mul_B.multiply3[i];
}
}

/* S-Function Block: <S3>/To RTDX31 (rtdx_snk) */
if (RTDX_isOutputEnabled( &r16 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &r16, (void*) rtdx_com_16mul_B.Subtract1,
16*sizeof(real32_T));
}

{
int32_T i;
for (i = 0; i < 16; i++) {
/* Product: '<S11>/multiply2' */
rtdx_com_16mul_B.multiply2[i] =
rtdx_com_16mul_B.FromRTDX2[i] *
rtdx_com_16mul_B.FromRTDX3[15];

/* Product: '<S11>/multiply3' */
```

```
rtdx_com_16mul_B.multiply3[i] =
rtdx_com_16mul_B.FromRTDX1[i] *
rtdx_com_16mul_B.FromRTDX4[15];

/* Sum: '<S11>/Subtract1' */
rtdx_com_16mul_B.Subtract1[i] = rtdx_com_16mul_B.multiply2[i] +
rtdx_com_16mul_B.multiply3[i];
}
}

/* S-Function Block: <S3>/To RTDX32 (rtdx_snk) */
if (RTDX_isOutputEnabled( &i16 )) {
while (RTDX_writing != NULL) {
} /* waiting for rtdx write to complete */

RTDX_write( &i16, (void*) rtdx_com_16mul_B.Subtract1,
16*sizeof(real32_T));
}
}

/* Model initialize function */
void rtdx_com_16mul_initialize(boolean_T firstTime)
{
(void)firstTime;

/* Registration code */

/* initialize error status */
rtmSetErrorStatus(rtdx_com_16mul_M, (const char_T *)0);

```

```
/* block I/O */
{
int_T i;
void *pVoidBlockIORegion;
pVoidBlockIORegion = (void
*)(&rtdx_com_16mul_B.FromRTDX1[0]);
for (i = 0; i < 112; i++) {
((real32_T*)pVoidBlockIORegion)[i] = 0.0F;
}
}

/* S-Function Block: <Root>/From RTDX1 (rtdx_src) */
{
RTDX_enableInput(&sin_real);
}

/* S-Function Block: <Root>/From RTDX3 (rtdx_src) */
{
RTDX_enableInput(&real_);
}

/* S-Function Block: <Root>/From RTDX2 (rtdx_src) */
{
RTDX_enableInput(&sin_img);
}

/* S-Function Block: <Root>/From RTDX4 (rtdx_src) */
{
RTDX_enableInput(&img_);
}
```

```

RTDX_enableOutput(&r1); /* S-Function Block: <S3>/To RTDX1 (rtdx_snk) */
RTDX_enableOutput(&i1); /* S-Function Block: <S3>/To RTDX2 (rtdx_snk) */
RTDX_enableOutput(&r2); /* S-Function Block: <S3>/To RTDX3 (rtdx_snk) */
RTDX_enableOutput(&i2); /* S-Function Block: <S3>/To RTDX4 (rtdx_snk) */
RTDX_enableOutput(&r3); /* S-Function Block: <S3>/To RTDX5 (rtdx_snk) */
RTDX_enableOutput(&i3); /* S-Function Block: <S3>/To RTDX6 (rtdx_snk) */
RTDX_enableOutput(&r4); /* S-Function Block: <S3>/To RTDX7 (rtdx_snk) */
RTDX_enableOutput(&i4); /* S-Function Block: <S3>/To RTDX8 (rtdx_snk) */
RTDX_enableOutput(&r5); /* S-Function Block: <S3>/To RTDX9 (rtdx_snk) */
RTDX_enableOutput(&i5); /* S-Function Block: <S3>/To RTDX10 (rtdx_snk) */
RTDX_enableOutput(&r6); /* S-Function Block: <S3>/To RTDX11 (rtdx_snk) */
RTDX_enableOutput(&i6); /* S-Function Block: <S3>/To RTDX12 (rtdx_snk) */
RTDX_enableOutput(&r7); /* S-Function Block: <S3>/To RTDX13 (rtdx_snk) */
RTDX_enableOutput(&i7); /* S-Function Block: <S3>/To RTDX14 (rtdx_snk) */
RTDX_enableOutput(&r8); /* S-Function Block: <S3>/To RTDX15 (rtdx_snk) */
RTDX_enableOutput(&i8); /* S-Function Block: <S3>/To RTDX16 (rtdx_snk) */
RTDX_enableOutput(&r9); /* S-Function Block: <S3>/To RTDX17 (rtdx_snk) */
RTDX_enableOutput(&i9); /* S-Function Block: <S3>/To RTDX18 (rtdx_snk) */
RTDX_enableOutput(&r10); /* S-Function Block: <S3>/To RTDX19 (rtdx_snk) */
RTDX_enableOutput(&i10); /* S-Function Block: <S3>/To RTDX20 (rtdx_snk) */
RTDX_enableOutput(&r11); /* S-Function Block: <S3>/To RTDX21 (rtdx_snk) */
RTDX_enableOutput(&i11); /* S-Function Block: <S3>/To RTDX22 (rtdx_snk) */
RTDX_enableOutput(&r12); /* S-Function Block: <S3>/To RTDX23 (rtdx_snk) */
RTDX_enableOutput(&i12); /* S-Function Block: <S3>/To RTDX24 (rtdx_snk) */
RTDX_enableOutput(&r13); /* S-Function Block: <S3>/To RTDX25 (rtdx_snk) */
RTDX_enableOutput(&i13); /* S-Function Block: <S3>/To RTDX26 (rtdx_snk) */
RTDX_enableOutput(&r14); /* S-Function Block: <S3>/To RTDX27 (rtdx_snk) */
RTDX_enableOutput(&i14); /* S-Function Block: <S3>/To RTDX28 (rtdx_snk) */
RTDX_enableOutput(&r15); /* S-Function Block: <S3>/To RTDX29 (rtdx_snk) */
RTDX_enableOutput(&i15); /* S-Function Block: <S3>/To RTDX30 (rtdx_snk) */
RTDX_enableOutput(&r16); /* S-Function Block: <S3>/To RTDX31 (rtdx_snk) */
RTDX_enableOutput(&i16); /* S-Function Block: <S3>/To RTDX32 (rtdx_snk) */
}

/* Model terminate function */
void rtdx_com_16mul_terminate(void)
{
RTDX_disableInput(&sin_real); /* S-Function Block: <Root>/From RTDX1 (rtdx_src) */
RTDX_disableInput(&real_); /* S-Function Block: <Root>/From RTDX3 (rtdx_src) */
RTDX_disableInput(&sin_img); /* S-Function Block: <Root>/From RTDX2 (rtdx_src) */
RTDX_disableInput(&img_); /* S-Function Block: <Root>/From RTDX4 (rtdx_src) */
RTDX_disableOutput(&r1); /* S-Function Block: <S3>/To RTDX1 (rtdx_snk) */
RTDX_disableOutput(&i1); /* S-Function Block: <S3>/To RTDX2 (rtdx_snk) */
RTDX_disableOutput(&r2); /* S-Function Block: <S3>/To RTDX3 (rtdx_snk) */
RTDX_disableOutput(&i2); /* S-Function Block: <S3>/To RTDX4 (rtdx_snk) */
RTDX_disableOutput(&r3); /* S-Function Block: <S3>/To RTDX5 (rtdx_snk) */
RTDX_disableOutput(&i3); /* S-Function Block: <S3>/To RTDX6 (rtdx_snk) */
RTDX_disableOutput(&r4); /* S-Function Block: <S3>/To RTDX7 (rtdx_snk) */
RTDX_disableOutput(&i4); /* S-Function Block: <S3>/To RTDX8 (rtdx_snk) */
RTDX_disableOutput(&r5); /* S-Function Block: <S3>/To RTDX9 (rtdx_snk) */
RTDX_disableOutput(&i5); /* S-Function Block: <S3>/To RTDX10 (rtdx_snk) */
RTDX_disableOutput(&r6); /* S-Function Block: <S3>/To RTDX11 (rtdx_snk) */
RTDX_disableOutput(&i6); /* S-Function Block: <S3>/To RTDX12 (rtdx_snk) */
RTDX_disableOutput(&r7); /* S-Function Block: <S3>/To RTDX13 (rtdx_snk) */
RTDX_disableOutput(&i7); /* S-Function Block: <S3>/To RTDX14 (rtdx_snk) */
RTDX_disableOutput(&r8); /* S-Function Block: <S3>/To RTDX15 (rtdx_snk) */
RTDX_disableOutput(&i8); /* S-Function Block: <S3>/To RTDX16 (rtdx_snk) */
RTDX_disableOutput(&r9); /* S-Function Block: <S3>/To RTDX17 (rtdx_snk) */
RTDX_disableOutput(&i9); /* S-Function Block: <S3>/To RTDX18 (rtdx_snk) */
RTDX_disableOutput(&r10); /* S-Function Block: <S3>/To RTDX19 (rtdx_snk) */
RTDX_disableOutput(&i10); /* S-Function Block: <S3>/To RTDX20 (rtdx_snk) */
RTDX_disableOutput(&r11); /* S-Function Block: <S3>/To RTDX21 (rtdx_snk) */
RTDX_disableOutput(&i11); /* S-Function Block: <S3>/To RTDX22 (rtdx_snk) */
RTDX_disableOutput(&r12); /* S-Function Block: <S3>/To RTDX23 (rtdx_snk) */
RTDX_disableOutput(&i12); /* S-Function Block: <S3>/To RTDX24 (rtdx_snk) */
RTDX_disableOutput(&r13); /* S-Function Block: <S3>/To RTDX25 (rtdx_snk) */
RTDX_disableOutput(&i13); /* S-Function Block: <S3>/To RTDX26 (rtdx_snk) */
RTDX_disableOutput(&r14); /* S-Function Block: <S3>/To RTDX27 (rtdx_snk) */
RTDX_disableOutput(&i14); /* S-Function Block: <S3>/To RTDX28 (rtdx_snk) */
RTDX_disableOutput(&r15); /* S-Function Block: <S3>/To RTDX29 (rtdx_snk) */
RTDX_disableOutput(&i15); /* S-Function Block: <S3>/To RTDX30 (rtdx_snk) */
RTDX_disableOutput(&r16); /* S-Function Block: <S3>/To RTDX31 (rtdx_snk) */
RTDX_disableOutput(&i16); /* S-Function Block: <S3>/To RTDX32 (rtdx_snk) */
}
```

```
#include "MW_c6xxx_csl.h"
#include "rtwtypes.h"
```

```
#include "rtdx_com_16mul.h"
#include "rtdx_com_16mul_private.h"
#include "rtdx_com_16mulcfg.h"
#include <hwi.h>
#define _C6XCHIP_SOURCE_FILE_

void cslInitialize (void);
void turnOn_L2Cache(void);
void targetInitialize(void)
{
cslInitialize();
}

void targetTerminate(void)
{
}

TIMER_Handle hTimer1;
void configureTimers(void)
{
Uint32 timerControl = TIMER_CTL_RMK(
TIMER_CTL_INVINP_NO,
TIMER_CTL_CLKSRC_CPUOVR4,
TIMER_CTL_CP_PULSE,
TIMER_CTL_HLD_YES,
TIMER_CTL_GO_NO,
TIMER_CTL_PWID_ONE,
TIMER_CTL_DATOUT_0,
TIMER_CTL_INVOUT_NO,
TIMER_CTL_FUNC_GPIO
);
TIMER_Config timerCfg;
Uint32 timerEventId;

// Initialize control and count fields of
// the timer configuration object
timerCfg.ctl = timerControl;
timerCfg.cnt = 0x0;

// Configure timer for timer interrupt 15
hTimer1 = TIMER_open(TIMER_DEV1, TIMER_OPEN_RESET);
timerCfg.prd = 11250000U;
TIMER_config(hTimer1, &timerCfg);
timerEventId = TIMER_getEventId(hTimer1);
IRQ_map(timerEventId, 15);
IRQ_enable(timerEventId);
TIMER_start(hTimer1);
}

void Timer1_ISR(Uint32 Mailbox)
{
SEM_post( &rtClockSEM );
}

void cslInitialize(void)
{
}

/* Function: enable_interrupts ------------------------------
*
```

```
* Abstract:
* Enable the all DMA and DSP interrupts
*/
void enable_interrupts()
{
}

/* Function: disable_interrupts -----------------------------
*
* Abstract:
* Disable all DSP interrupts
*/
void disable_interrupts()
{
IRQ_globalDisable();
}

//
//EOF -- MW_c6xxx_csl.c
void turnOn_L2Cache()
{
CACHE_setL2Mode (CACHE_64KCACHE);
CACHE_enableCaching (CACHE_CE00);
}
```