REAL TIME IMPLEMENTATION OF AUDIO SPECTROGRAM ON FIELD PROGRAMMABLE GATE ARRAY (FPGA)

A Thesis

Presented to the

Faculty of

San Diego State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Electrical Engineering

by

Akshay KrishneGowda Hebbal

Spring 2014

SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

Thesis of Akshay KrishneGowda Hebbal:

Real Time Implementation of Audio Spectrogram on Field

Programmable Gate Array (FPGA)

[illegible]

Ashkan Ashrafi, Chair
Department of Electrical and Computer Engineering

[illegible]

Frederic J. Harris
Department of Electrical and Computer Engineering

[illegible]

Christopher Paolini
Department of Computer Science

12-12-2013

Approval Date

DEDICATION

To My Family, Friends and the Almighty God.

The busiest person is the one who has time for everything.

--Anonymous

ABSTRACT OF THE THESIS

Real Time Implementation of Audio Spectrogram on Field
Programmable Gate Array (FPGA)
by
Akshay KrishneGowda Hebbal
Master of Science in Electrical Engineering
San Diego State University, 2014

An audio spectrogram is a visual representation of sound. Spectrogram in general is defined as the time varying spectral representation which shows the variation of spectral density of a signal with respect to time. Spectrograms (also known as voicegrams, sonograms or spectral waterfalls) are typically used to identify phonetic sounds, to analyze the cries of animals and also in other fields like speech processing, sonar, seismology, etc. Spectrograms can be created using two methods, using a series of band pass filters to form the approximated filter bank and short-time Fourier transform (STFT) calculated from the time signal. This thesis concentrates on using the short-time Fourier transform (STFT) to obtain the audio spectrogram.

The objective of this research is the real time implementation of real time spectrogram of an audio signal on a video monitor using Xilinx Virtex-5 ML506 Evaluation Board. The Xilinx ML506 Virtex-5 Evaluation Board has powerful audio and video capabilities, which are utilized in this research. The input audio signal is processed in the FPGA to calculate STFT of the signal. Once the STFT is calculated, it must be converted into a form suitable for displaying by the video monitor. The video signal sent to the monitor shows the real time audio spectrogram. This research has several applications in scientific and commercial devices. The design is written is Verilog & VHDL, simulated using Xilinx ISE Web pack and programmed on the Xilinx Virtex-5 ML506 Evaluation Board.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

GLOSSARY

FPGA	Field Programmable Gate Array
STFT	Short Time Fourier Transform
FFT	Fast Fourier Transform
VGA	Video Graphics Array
DVI	Digital Visual Interface
DAC	Digital to Analog Converter
ADC	Analog to Digital Converter
FIFO	First In First Out
BRAM	Block Random Access Memory
LSB	Least Significant Bit
MSB	Most Significant Bit
SD	Spectrographic Display
XUP	Xilinx University program
DVI	Digital Visual Interface
VGA	Video Graphics Array
SRAM	Synchronous Random Access Memory
ROM	Read Only Memory

ACKNOWLEDGMENTS

It is a pleasure to thank all who have made this thesis possible.

Foremost, it is difficult to put in words my gratitude towards my thesis advisor, Dr. Ashkan Ashrafi. Without his enthusiasm, sound advice, great explanation skills, efforts to put me on the right path, and continued patience, this would not have been possible.

I would also take this opportunity to thank Dr. Frederick J Harris and Dr. Christopher Paolini for their co-operation and time. I thank my friends, Mahesh Kumar, Kishore Reddy and Kiran Noojibail, for the informal support and encouragement I received in the form of insightful comments, challenging questions and guidance when I was lost.

Lastly and importantly, I would like to thank my family for providing me moral, emotional and financial support all along. Thanks Mom, Dad and Ashwin.

CHAPTER 1

INTRODUCTION

1.1 Spectrogram Introduction

Ever since the first implementation of spectrogram in the 1950's, there have been significant changes till date. Spectrographic display (SD) as it was called, developed by the Bell Telephone Laboratories was first used at the Detroit Day School for computerized speech synthesis and as a feedback tool for teaching speech to deaf students [1, 2]. Spectrograms are extensively used in medical applications. The real time spectrogram display (SD) is a very useful tool in biomedical feedback and has been used to obtain information about the frequency components of the speech or acoustic signal [3, 4]. The display shows the variations in frequency over a specific period of time. Frequency is measured along the vertical axis, time along the horizontal axis and amplitude or loudness is examined by the gray scale or color intensity.

Spectrogram in general is defined as the time varying spectral representation which shows the variation of spectral density of a signal with respect to time. Spectrograms are also known as voice grams, sonograms, and spectral waterfalls. Typically they are used to identify the phonetic sounds, to analyze the cries of animals, but are also used in other fields like speech processing, sonar, seismology, etc. Spectrograms can be created using two methods. The first method is using a series of band pass filters to form the approximated filter bank. The second is the short time Fourier transform (STFT) calculated from the time signal using Fast Fourier transform (FFT).

The approximated filter bank method was the only method available before the advent of modern digital signal processing. This method uses analog signal processing to divide the input signal into smaller frequency domain slices or bands. A transducer is then used to record the spectrogram as an image depending on the output magnitude of each filter [5]. Since this is an old method employing analog signal processing which is prone to errors, it is rarely used nowadays.

The short time Fourier transform is the most commonly used method for finding the spectrogram. The process involved using STFT is digital. The digitally sampled data in time domain is divided into intervals and Fourier transform is calculated to find the magnitude of frequency spectrum of each interval. Each interval corresponds to a vertical line in the image which is the measurement of magnitude versus frequency over a period of time. Mathematically speaking, the spectrogram can be calculated by squaring the magnitude of STFT of the signal. The horizontal axis corresponds to time and the vertical axis corresponds to frequency, a third dimension indicating the amplitude of a particular frequency at a particular time is represented by the intensity or color of each point in the image (spectrogram) [6]. This will be explained in detail later.

1.2 Overview of FPGA Technology

Prior to the invention of Programmable Logic Devices (PLD's), the electronic system designers had to use specialized integrated circuits (IC) called discrete logic containing only a few gates. To create a moderately complex device, tens of such chips had to be mounted on a single board which leads to more complex board layout and reduced performance. The first type of Programmable Logic Devices (PLD's) namely the Programmable Logic Arrays (PLAs) was introduced in the early 1970's. Texas Instruments (TI) developed a mask-programmable IC based on the IBM read-only associative memory or ROAM. The term Programmable Logic Array for this device was coined by TI [7]. PLAs were one time programmable chips containing AND and OR gates having the ability to implement simple logic functions. Complex Programmable Logic Devices (CPLD) were later invented which is seen as a continuation of PLAs [8]. CPLD chip includes logic blocks (*macrocells*) at the borders of the chip, and a connection matrix located at the central part. Each macrocell has a structure similar to that of PLA. Hence CPLDs are nothing but a set of PLAs on one chip with programmable interconnects. It was not until Xilinx brought us the Field Programmable Gate Array (FPGA) in the late 1980's that are the PLDs crossed paths with the ASIC world. The main difference between CPLD and FPGA is not in configuration memory, but in the underlying architecture [9].

The first Field-programmable Gate Array (FPGA) – the XC2064 was developed in Xilinx in 1985 by its co-founders namely Ross Freeman and Bernard Vonderschmitt [10].

The XC2064 had programmable gates and programmable interconnects between gates indicating the beginnings of a new technology and market [11]. The XC2064 had just 64 configurable logic blocks (CLBs), with two 3-input lookup tables (LUTs) [12]. The number of gates on a chip was in low thousands and was not so effective at that time. This technology was patented in 1992 after which the FPGA technology took off [13]. This was initially used in telecommunication and networking equipment and later in automotive and consumer applications. By the turn of the century, the chips were starting to have millions of gates on them.

The FPGA is an integrated circuit designed with the ability to reconfigure its circuitry for a desired application of function at any time after manufacturing hence it is called "field-programmable". Similar to the Application-Specific Integrated Circuit (ASICs), the FPGA configuration is generally specified using a Hardware Description Language (HDL). Also the FPGAs can be used to implement any logical function that an ASIC could perform. It offers advantages for many applications with its ability to update the functionality after shipping, partial re-configuration of a portion of the design and low non-recurring engineering costs relative to an ASIC design (notwithstanding the generally higher unit cost).

The FPGAs contain **programmable logic** components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together" similar to the changeable logic gates that can be inter-wired in (many) different configurations. These Logic blocks can be configured to perform complex **combinational functions**, or simple **logic gates** like **AND** and **XOR**. The logic blocks of most FPGAs also include memory elements, which may be simple **flip-flops** or more complete blocks of memory. Files can be downloaded onto FPGAs as many times as we want with different functionalities for different applications as and when required. If there is some mistake in the functionality, we can just fix the logic, recompile and reprogram the FPGA. No printed circuit board, solder or change of component is necessary. The design runs faster on FPGAs than if designed with discrete components, since everything runs within the FPGA on a single silicon die. FPGAs lose their functionality when there is no power to it, as a result it has to be reprogrammed when powered on to restore their functionality.

In addition to digital functions, some FPGAs have analog features. The most common analog feature is programmable slew rate and drive strength on each output pin, allowing the engineer to set slow rates on lightly loaded pins that would otherwise ring unacceptably, and to set stronger, faster rates on heavily loaded pins on high-speed channels that would otherwise run too slow [14]. Another common analog feature is differential comparators on input pins designed to be connected to differential signaling channels [15]. Some of the "mixed signal FPGAs” has integrated peripheral Analog-to-Digital Converters (ADCs) and Digital-to-Analog Converters (DACs) with analog signal conditioning blocks allowing them to operate as a system on-a-chip. Such devices eliminate the line between an FPGA, which carries digital ones and zeros on its internal programmable interconnect fabric, and field-programmable analog array (FPAA), which carries analog values on its internal programmable interconnect fabric. There are many FPGA vendors in the market namely Xilinx, Altera, Actel, Atmel, Lucent, Cypress, QuickLogic, IBM, Motorola. Figures 1.1[16] & 1.2 [17] show the most recent FPGAs of the top two companies namely Xilinx and Altera.

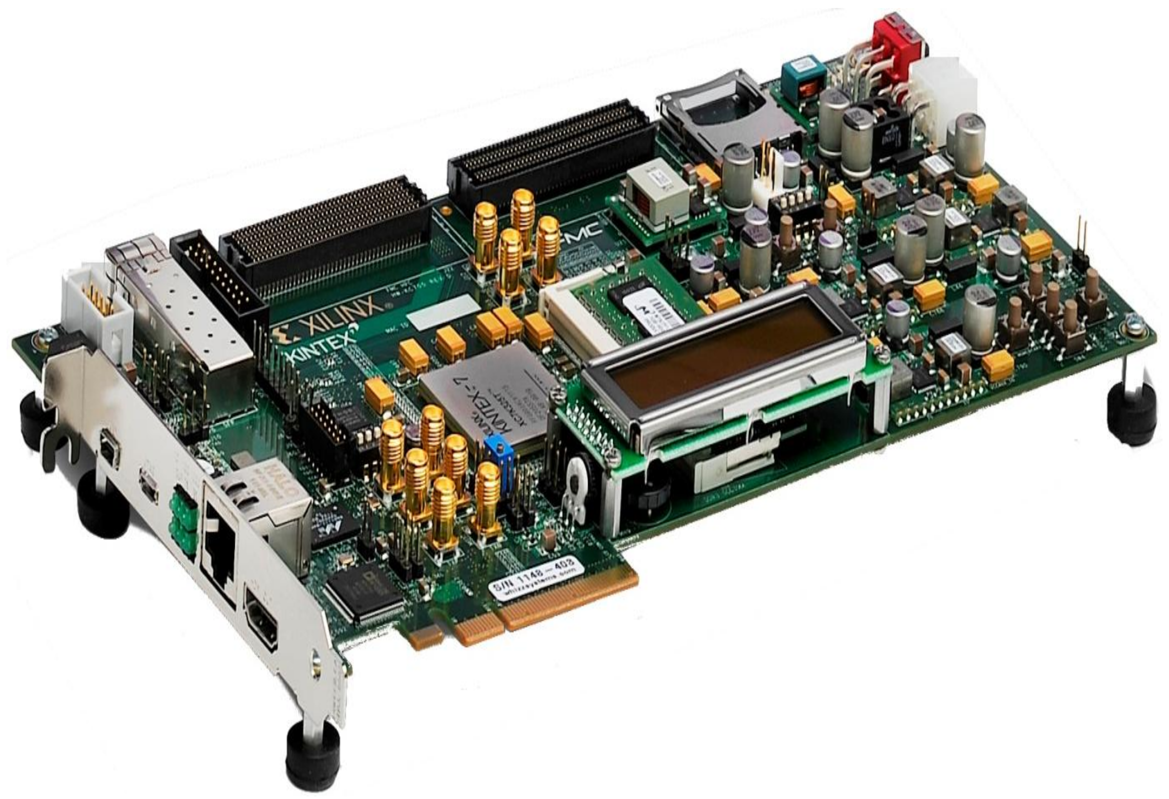


Figure 1.1. Xilinx Kintex-7 FPGA KC 705 board with XC7K325T FPGA. Source: Wong, William, “FPGA Kits Support FMC Mezzanine Card Development,” Electronic Design, Last modified February. 2, 2012, http://electronicdesign.com/author/william-wong.

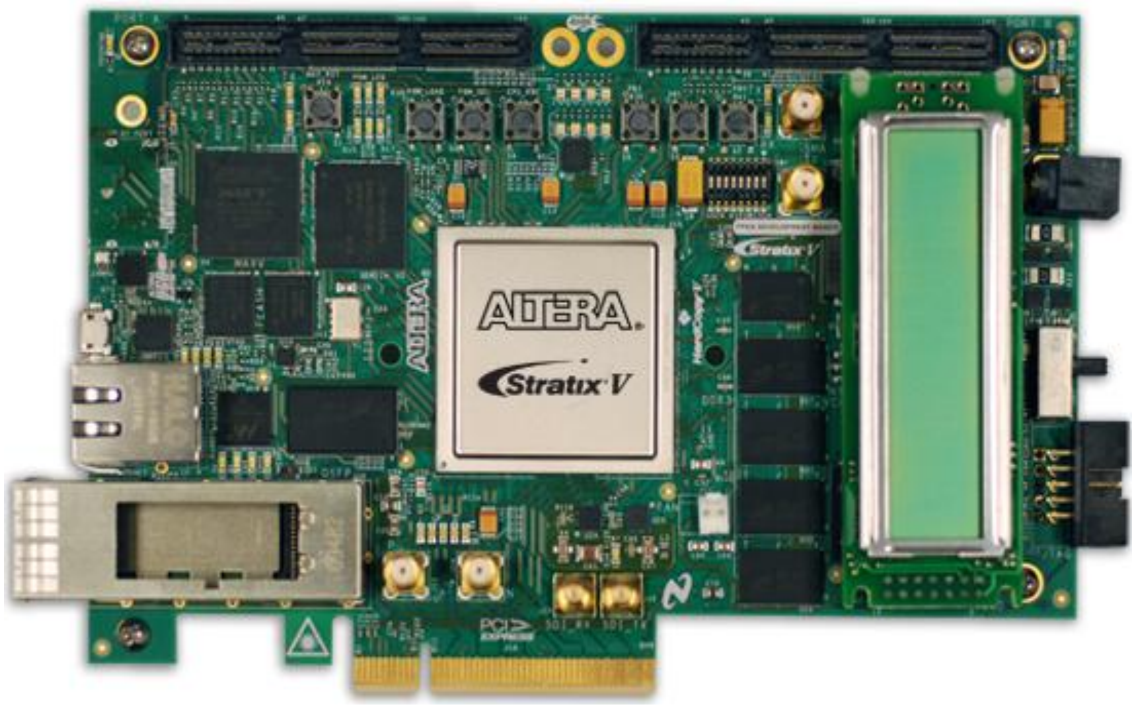


Figure 1.2. Altera Stratix V GX FPGA development board. Source: Altera, "Stratix V GX FPGA Development Kit: Figure 1," Last modified 2011, http://www.altera.com/products/devkits/altera/kit-sv-gx-host.html#RelatedLinks.

1.2.1 Advantages of the FPGA

As mentioned in the overview of FPGA, today there are many FPGA Prototype boards available in the market supplied by various FPGA vendors. The most important task is the selection of the FPGA suitable to the application that we are trying to implement.

The advantages of FPGAs in general are discussed below

- ✓ The main advantage of the FPGA is its computation speed. With the implementation of hardware parallelism, the computation power of FPGAs is much more than compared to the digital signal processors (DSP) by breaking the paradigm of sequential execution and accomplishing more per clock cycle. The control of inputs and outputs at the hardware level increases the response time and dedicated functionality to meet the application needs.
- ✓ The second advantage of the FPGAs is the time to market. Due to the increased flexibility and the rapid prototyping capabilities of the FPGAs, any idea or application can be tested and verified on the hardware without having to go through the process of fabrication which is in case with ASIC design. Any changes to be implemented can be programmed on the FPGA within a short period of time. Commercial off-the-shelf (COTS) hardware is also available with different types of I/O already connected to a user-programmable FPGA chip.

- ✓ The cost usually referred to as the nonrecurring engineering (NRE) expense of the FPGA based hardware solution is far less than that of custom ASIC design. The initial investment of ASICs is very large compared to the FPGAs since it involves chip fabrication process but is justified by the fact that thousands of chips are shipped per year. The end user may require custom hardware functionality for many systems, in which case the programmable silicon with no fabrication process is used. The system requirements may also change over time which requires respinning in case of ASICs resulting in large expenses whereas the FPGAs can be programmed just by making changes in the programs and uploading the bitstream.
- ✓ The FPGAs are field-upgradable and do not require the time and expense involved with ASIC redesign. Re-Configurability of the FPGAs help in making future modifications and functional enhancements without hardware redesign or board layout modification. There may be time consuming applications and expensive designs which can be avoided through application specific integration of IP cores in the FPGA, especially for the specialized applications. As the older FPGA's are replaced with the newer ones, the IP cores can be integrated with new FPGA's without any redesign.

1.2.2 Applications of the FPGA

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

The FPGA particularly finds applications in any area which requires high level of parallelism that is offered by their architecture. The FPGAs are used increasingly instead of microprocessors in conventional high performance computing applications where computational kernels such as the Fast Fourier Transform or convolution are performed on the FPGA. The intrinsic parallelism of the logic resources enable considerable amount of computational output even at a lower clock rates. Also the flexibility of the FPGA allows higher performance by trading off precision and range in the number format for an increased number of parallel arithmetic units. This leads to a type of processing called reconfigurable computing, in which the time intensive tasks are unloaded from software to FPGAs. The FPGAs have a structural design which is more complex than the conventional software; as a result it is limited only to high performance computing.

Traditionally, the FPGAs have been reserved for specific applications where the volume of production is small. The premium that companies pay in hardware costs per unit for a programmable chip is more affordable than the development resources spent on creating an ASIC for these low-volume applications. Nowadays, new cost and performance dynamics have broadened the range of viable applications.

Considering the application that has to be implemented, choosing the right FPGA board goes a long way in minimizing the task of designers. Since we had to implement the spectrogram (STFT) of audio signals, the Xilinx Virtex-5 ML506 Xtreme DSP board serves as the perfect choice. The detailed discussion is provided in the next section.

1.3 XILINX VIRTEX-5 ML506 FPGA EVALUATION BOARD

The spectrogram was implemented using a suitable FPGA board namely the Xilinx Virtex-5 ML506 FPGA board. The features can be found in the user guide here [18]. Some of the most important features of the FPGA board used in this research are briefly listed below

- Stereo AC97 audio codec with line-in, line-out, 50-mW headphone, microphone-in jacks, SPDIF digital audio jacks, and piezo audio transducer
- Video input/output
  - Video input (VGA)
  - Video output DVI connector (VGA supported with included adapter)
- 64-bit wide, 256-MB DDR2 small outline DIMM (SODIMM), compatible with EDK supported IP and software drivers
- ZBT synchronous SRAM, 9 Mb on 32-bit data bus with four parity bits
- JTAG configuration port for use with Parallel Cable III, Parallel Cable IV, or Platform USB download cable
- General purpose DIP switches (8), LEDs (8), pushbuttons, and rotary encoder
- RS-232 serial port, DB9 and header for second serial port 16-character x 2-line LCD display
- Xilinx System ACE™ CompactFlash configuration controller with Type I CompactFlash connector
- Two Xilinx XCF32P Platform Flash PROMs (32 Mb each) for storing large device
- Configurations
- Clocking
  - Programmable system clock generator chip
  - One open 3.3V clock oscillator socket

  - External clocking via SMAs (two differential pairs)
- PS/2 mouse and keyboard connectors
- Intel P30 StrataFlash linear flash chip (32 MB)
- Serial Peripheral Interface (SPI) flash (2 MB)
- 10/100/1000 tri-speed Ethernet PHY transceiver and RJ-45 with support for MII, GMII, RGMII, and SGMII Ethernet PHY interfaces

1.4 Objective

The main objective of this research is to obtain the real time spectrogram of an audio signal on the monitor using Verilog and VHDL implemented on the FPGA evaluation board. This research has been implemented by integrating the three difference phases namely

- ✓ Audio recording and playback
- ✓ Real time display on the Digital Visual Interface (DVI) monitor
- ✓ The spectrogram representation using STFT

In order to implement these phases, the research requires the knowledge of Verilog and/or VHDL programming languages. Selecting a suitable FPGA board for the application, understanding the necessary features related to the particular FPGA is also an important aspect. Implementing the necessary functions to obtain the final result involves understanding the FPGA development software used for onboard testing.

This research is organized into three phase.

1.4.1 Audio Recording and Playback

A real time audio signal is captured using the microphone jack present on the FPGA board. This phase is subdivided based on the three functions implemented namely loopback, recording and playback.

- Loopback the real time audio input signal through the microphone to the speakers connected to the headphone jack (onboard).
- Recording the input audio and storing it in the onboard memory which is later retrieved by the playback module.
- Playback the recorded audio saved in the memory when the corresponding pushbutton is pressed.

All these functions including the increase and decrease of audio volume is implemented with the help of pushbuttons.

1.4.2 Real Time Audio Signal Display on DVI Monitor

This phase implements the digital audio to video conversion, implemented by the DVI core written in VHDL. When the on board loopback push button is pressed, the real time audio is output through the speakers and the corresponding video is displayed in real time on the DVI monitor. The frequency and amplitude are varied in the signal generator and the corresponding variations are heard on the speakers and also shown on the DVI monitor.

1.4.3 The Spectrogram Representation of an Audio Signal Using Short Time Fourier Transform (STFT)

The last phase of the research is the display of real time audio spectrogram on the DVI monitor. The Short time Fourier Transform (STFT) of the real time audio signal is calculated by integrating it with the previous two phases. Hann window is used for STFT and overlapping window technique is applied to the signal to eliminate the loss at the edges of the window. Similar to the previous phase, pushbutton controls are used for Loopback, recording and playback. The LED indicates the operation currently being executed when pushbuttons are pressed and also gives the memory status.

All the three phases use the Verilog & VHDL programming language and the Xilinx ISE web pack for simulation, debugging and implementation on board.

Block diagram representation of the Audio Spectrogram implementation is as shown in Figure 1.3.

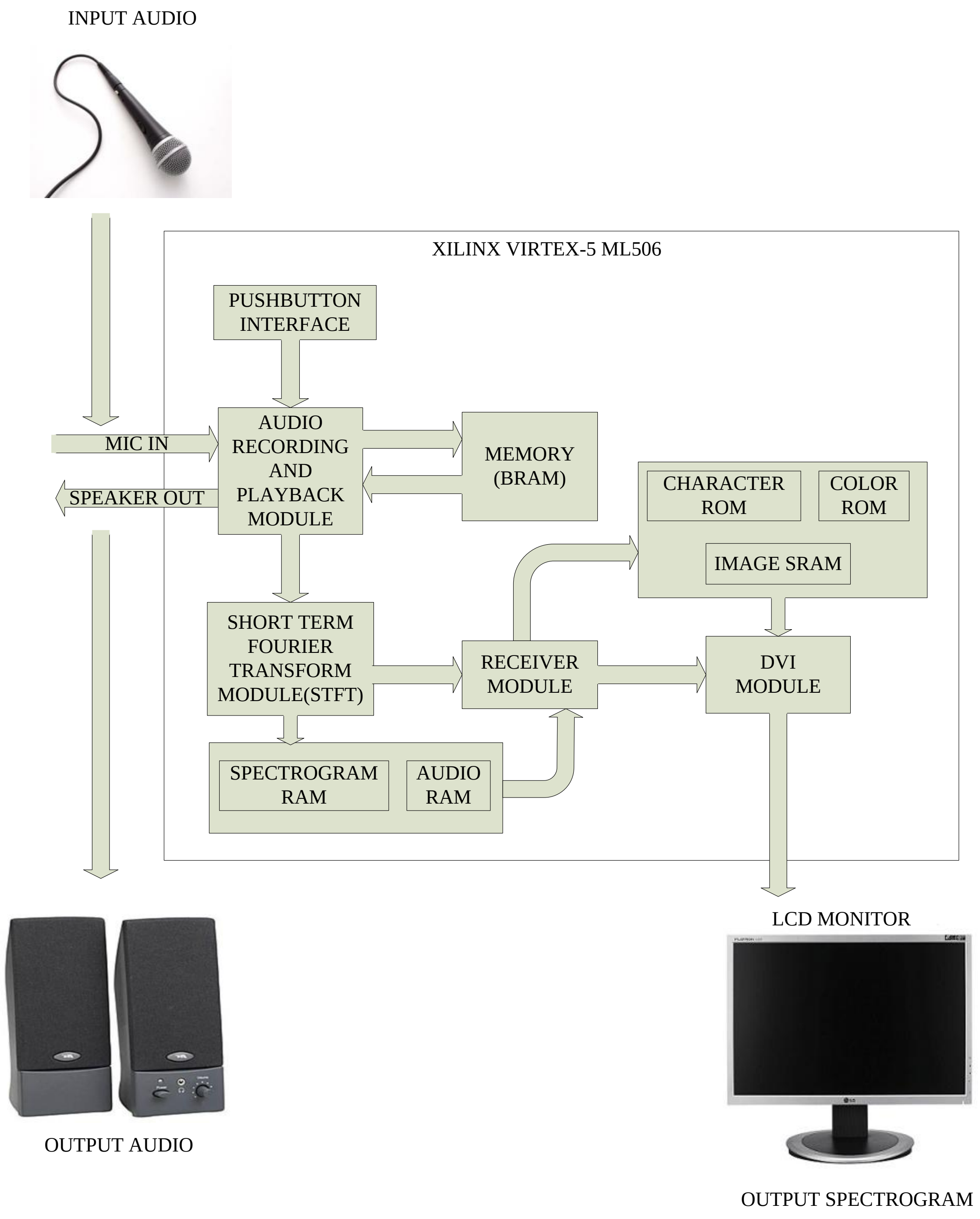


Figure 1.3. Block diagram representation of audio spectrogram implementation.

CHAPTER 2

AUDIO RECORDING AND PLAYBACK

Audio Recording and Playback is the first phase of this research. This phase involves understanding the basic functionality of the onboard Audio codec AD1981B used for loopback, recording and playback. The pushbuttons activate these functions and LED's indicate when these functions are active. The RTL schematic representation of this phase is as shown in Figure 2.1.

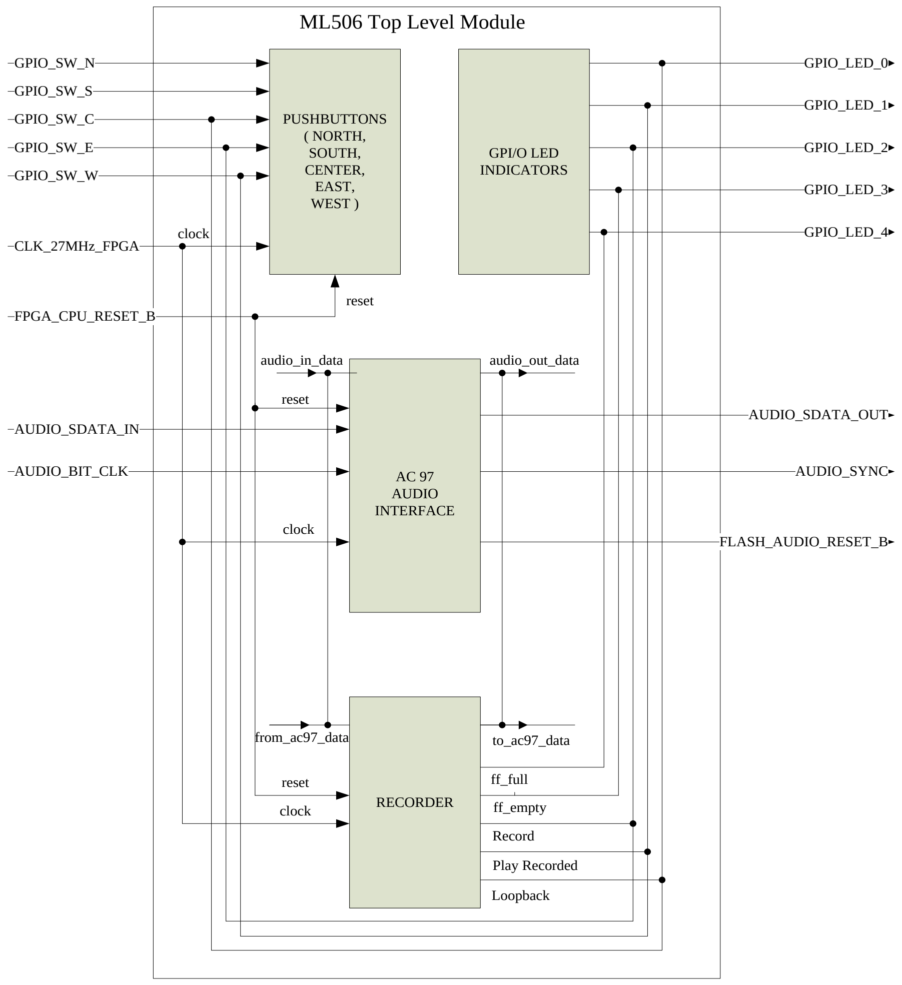


Figure 2.1. RTL schematic of audio recording and playback.

The audio recording and playback has three important interface modules namely

1. AC97 audio codec interface
2. Recorder interface
3. Pushbutton interface

2.1 AC97 Audio Codec Interface

This module forms the core and consists of the necessary interface for configuring the onboard AD1981B AC97 audio codec [19]. In order for the proper operation of the codec, we need to configure the codec registers. The data sheet of the codec has all the information of how the data can be received and register configuration [19]. The codec has two inputs line in and microphone in and two outputs headphone and line output. This research uses microphone as an input and headphone as output. The AD1981B AC97 audio codec [19] can record and playback high quality audio. The ADC's and DAC's of the codec supports sampling rates of 48 kHz and 44.1 kHz and 20-bit data format.

The codec also has the following features

- ✓ Integrated stereo headphone amplifier
- ✓ Variable sample rate audio
- ✓ External audio power-down control
- ✓ Stereo full-duplex codec
- ✓ 20-bit PCM DAC
- ✓ 3 analog line-level stereo inputs for line-in, AUX, and CD
- ✓ Mono line-level phone input
- ✓ Dual MIC input with built-in programmable preamplifier
- ✓ High quality CD input with ground sense
- ✓ Mono output for speakerphone or internal speaker

The AD1981B codec is 48-Lead LQFP pin package [19]. The main pins that are used and their functional descriptions are listed below [19]

a. **SDATA_OUT:** This is the input to codec, AC-Link Serial Data Output and AD1981B Data Input Stream. This is actually the input for the AC Link Output frames from an AC '97 Digital Audio Controller to the AD1981B codec. The frames will contain both the control data and DAC PCM audio data. The input is sampled by the AD1981B on the falling edge of BIT_CLK.

b. **BIT_CLK:** This is the AC Link clock. It is an OUTPUT in primary Codec mode and provides a 12.288 MHz clock for AC Link. This pin is an INPUT when the codec is configured in Secondary mode.

c. **SDATA_IN:** This is the output from codec. AC-Link Serial Data Input and AD1981B Data Output Stream. This is actually the output for the AC Link Input frames from AD1981B codec to an AC’97 Digital Audio Controller. The frames will contain both the control data and PCM audio data from the ADC’s. The AD1981B clocks this on the rising edge of BIT_CLK.

d. **AUDIO_SYNC:** It is the AC Link Frame Sync. This input defines the boundaries of AC link frames.

e. **RESET:** It is the AC-Link Reset, the AD1981B Master Hardware Reset. The reset for the AC’97 codec is shared with the reset signal for the flash memory chips and is designed to be asserted at power-on or at system reset. It is an active low signal.

The AC’97 is a serial interface in which the data is transmitted to and from the codec one bit at a time. On every cycle of AC’97 BIT_CLK, one bit of data is transferred from the AC’97 controller to the codec over the SDATA_OUT wire, and one bit of data is transferred from the codec to the FPGA over the SDATA_IN wire.

A constant stream of data that is passing between the FPGA and the codec is divided into frames. Each frame has 256 bits. They are divided into twelve slots of 20 bits each and a 16-bit tag field serving as the frame header. The rising edge of the SYNC signal represents the start of the frame. The SYNC signal goes high one clock cycle before the first bit of a frame, and goes low at the same time as the last bit of the tag field is sent. The codec does not implement all of the features defined in AC’97 specification. And hence all the slots are not used.

From the RTL schematic representation shown in the Figure 2.2, the AC’97 module has two sub modules namely the *ac97* and *ac97commands*. The ac97 module is used for assembling and disassembling the AC’97 serial frames. This module is also used for generating the AUDIO_SYNC & ready signal and to decide which data must be sent to the left and the right channels respectively. The *ac97commands* module is used for issuing initialization commands to AC’97. The commands for configuring most of the registers are provided in this module. The headphone volume, PCM volume, Record source select, Record gain, microphone gain, general purpose register configuration for the PCM output are all

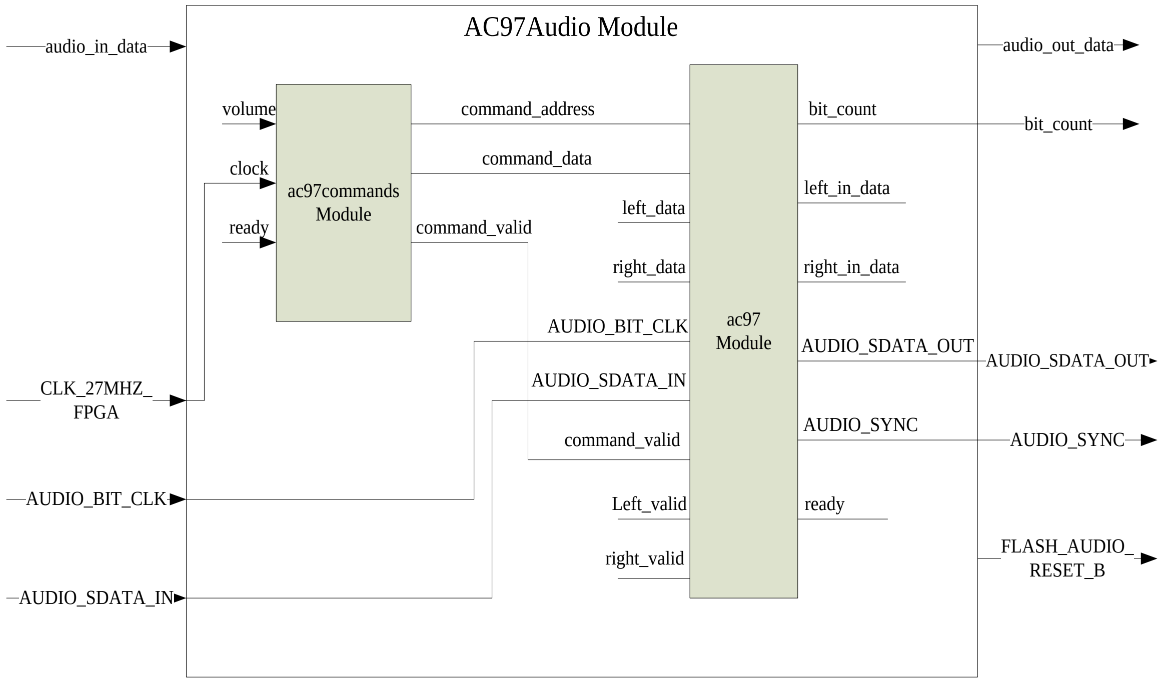


Figure 2.2. RTL schematic of AC97 audio interface.

defined in this module. A brief description of the registers and their configurations are as below

- ✓ The Headphone Volume register has right and left headphone volume control bits which are configured so that the user can control the volume of the output by using the pushbuttons present on board.
- ✓ The Record Select control register has 3bits each for selecting Left and Right Record source. The source can be Microphone, CD, Mute, AUX, Line in, Stereo Mix, Mono mix and Phone in. In this research microphone is selected as the source.
- ✓ The record gain register controls the gain which ranges from 0 to 22.5dB. We use the maximum gain of 22.5dB.
- ✓ A Microphone gain of 20dB is set by the Microphone volume register.
- ✓ The General purpose register is set to the default values of ADC/DAC Digital Loopback Mode, MIC1 select and Mixer Mono Output.

A more detailed description of the registers and their configuration can be found in the AD1981B data sheet [19].

The module also includes an interface to help the user to determine the function performed when the pushbuttons are in use with the help of general purpose LEDs present on board. The five general purpose LEDs that are used and their functions are described below

a. **GPIO_LED_0:** This LED switches ON to indicate the FIFO memory is full.

b. **GPIO_LED_1:** Initially this LED is ON indicating the FIFO memory is empty.

c. **GPIO_LED_2:** This LED switches ON when the GPIO_SW_E (record) push button is pressed, indicating the audio is recording and stored onto FIFO memory.

d. **GPIO_LED_3:** This LED switches ON when the GPIO_SW_W (play_recorded) push button is pressed, indicating the recorded audio is played back through the output speakers.

e. **GPIO_LED_4:** This LED switches ON when the GPIO_SW_C (Loopback) push button is pressed, indicating the input audio is loop backed through the output speakers.

2.2 Pushbutton Interface

Five active-High user pushbuttons are available for general purpose usage and are arranged in a North-East-South-West-Center orientation. The pushbutton interface consists of the necessary interface for detecting the push of a button which is present on the Xilinx Virtex-5 ML506 Evaluation Board. The Switch Debounce module is instantiated in the top level module for the five push button functions that are responsible for the audio recording and playback of this phase. The five pushbutton functions that are performed and their detailed description are as below

a. **GPIO_SW_N:** The general purpose North push button is implemented to allow the user to increase the volume of the output audio connected to headphone or speaker. The volume is increased by 1 decimal value on each press of the button. A 5 bit volume register is used and the maximum volume is set to decimal value 31. The default volume when the reset button is pressed is set to decimal value 8.

b. **GPIO_SW_S:** The general purpose South push button is implemented to allow the user to decrease the volume of the output audio connected to headphone or speaker. The volume is decreased by 1 decimal on each press of the button. A 5 bit volume register is used and the minimum volume is set to decimal value 0.

c. **GPIO_SW_C:** The general purpose Center push button is used to implement the Loopback functionality. When this button is pressed and held, the output speakers will loopback the audio received as input from microphone and when released the loopback stops.

d. **GPIO_SW_E:** The general purpose East push button is used to implement the Record functionality. When this button is pressed and held, the real time audio input is stored in the First in First out (FIFO) Block Random Access memory (BRAM).

e. **GPIO_SW_W:** The general purpose West push button is used to implement the Playback functionality. When this button is pressed and held, the audio data saved in the Block Random Access Memory (RAM) is sent to the output headphone jack and is played back using the speakers.

2.3 RECORDER INTERFACE

The Recorder module is one of the most important module and is responsible for performing the three basic functionalities of the Audio Recording and Playback Phase namely loopback, recording and playback. It performs several functions to make sure that the recording and playback of the Audio signal goes through without any problems. The recorded data has to be stored in the memory when record button is pressed and retrieved later for playback when playback button is pressed. The First in First out (FIFO) Block Random Access Memory (BRAM) is used for storing the data temporarily before sending for playback. In the Top Level Recorder module, at the positive edge of the Audio BIT_CLK, we check if the AC97 data is available using the ready signal. If the data is available, we check to see what button is pressed by using the Case statement. The 3 possible cases will be

a. If **GPIO_SW_C** button is pressed, it represents Loopback, so the data received through the microphone input is sent back to the speaker output.
b. If **GPIO_SW_E** button is pressed, it represents Record, so the data received through the microphone input is stored in the BRAM.
c. If **GPIO_SW_W** button is pressed, it represents Playback, so the data stored in the BRAM is retrieved and played back through the Speaker output.

The RTL schematic representation of Recorder Module is as shown in Figure 2.3.

The Recorder Interface has two instantiated sub modules within the Recorder Interface. Their introduction and functional description are explained in detail below.

a. **Recording_fifo:** The Recording _fifo in recorder interface instantiates another module named ram_4_recording, which is the FIFO BRAM generated using the Xilinx Core generator. The ram _4_ recording module is explained in detail later. The Recoding_fifo defines the BRAM memory that is required for storing approximately 10 seconds of the audio signal. The memory required is calculated using the following equation

Two flags are used to check whether the FIFO memory is full or empty. The ff_full_o flag checks to see when the memory reaches maximum value, sets it to 1 and switches the GPIO_LED_0 ON when memory is full. The ff_empty_o checks to see when the memory is greater than 1, sets it to 1 and switches the GPIO_LED_1 ON indicating the memory is empty.

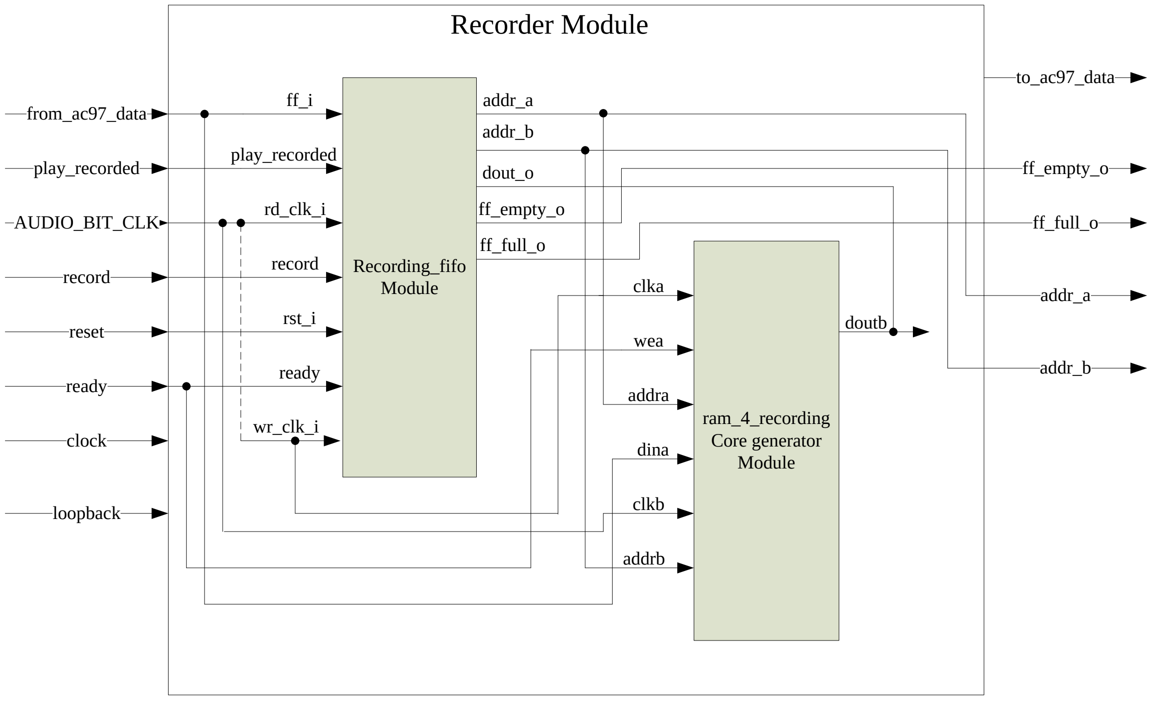


Figure 2.3. RTL schematic of recorder interface.

At the positive edge of the wr_clk_i, the write clock, if the reset flag is 1, the write address of the FIFO is set to zero. Else if the ready signal is 1 indicating AC97 data is available at input, record button is pressed and FIFO is not full then the write address is increased so as to store the incoming audio in the BRAM.

At the positive edge of rd_clk_i, the read clock, if the reset flag is 1, the read address of the FIFO is set to zero. Else if the ready signal is 1, play_recorded button is pressed and FIFO is not empty, then the read address is increased and the stored data in memory is retrieved and sent to the speaker output.

This design uses 48 kHz sampling rate and 8 bit depth. This means that we receive the data from ADC present in audio codec at 48000 times per second. The data is written into the BRAM at the same rate. The playback process is similar and the data is read from BRAM and transferred to the DAC present in audio codec at 48000 times per second.

Bit depth= 8 bit (1 byte).

Amount of byte per second = 48000 * 8 bit (1 byte) = 48000 byte.

Memory required to record 8 seconds of audio = 48000 * 8 = 384000 = 384KB

b. **ram_4_recording:** This is a Xilinx Core generated BRAM IP core. The Block Memory Generator uses the following specifications

- ✓ Simple Dual port RAM with single input port and single output port.
- ✓ Two address ports addra for writing and addrb for reading.
- ✓ Write and read width is 8 and depth 384000 which is the 384KB memory. It uses the write first operating mode.
- ✓ This phases use 64 36Kb BRAMs.
- ✓ The address width is 19 bits corresponding to the memory of 384KB.

Implementing the Audio recording and playback phase on the FPGA requires the following steps

1. Loading the bit file generated from the Xilinx ISE Project Navigator onto the FPGA using the ISE design tool “iMPACT”.
2. Once the bit file is loaded, the input is given from the signal generator to the onboard “MICROPHONE” jack and speakers are connected to the onboard “HEADPHONE” jack.
3. The frequency is varied in the signal generator from 700Hz to 20 kHz and the speakers simultaneously output the corresponding change in the audio.

CHAPTER 3

REAL TIME AUDIO SIGNAL DISPLAY ON DVI MONITOR

3.1 INTRODUCTION TO DIGITAL VISUAL INTERFACE (DVI)

Before explaining the implementation of Real time audio signal display on DVI monitor, a detailed description of DVI is given.

The PC monitor landscape has changed from the bulky CRT monitors to the sleek LCD monitor. Unlike CRT, the LCD monitors are inherently digital devices. DVI is the acronym for Digital Visual Interface and was developed by the Digital Display Working Group (DDWG). The digital interface is used to connect a video source to a display device, such as a computer monitor. The interface is designed to transmit uncompressed digital video data between PC host and a digital display (for example a LCD monitor). Since the DVI enables communication with the digital displays in the native form, it improves the image quality [20].

DVI can be configured to support multiple modes such as DVI-D (digital only), DVI-A (analog only), or DVI-I (digital and analog) and two classes namely single and dual link. It is the only standard which uses digital and analog transmission option in single connector. The reason for these classes and modes was that DVI-I format was originally designed to be a backward-compatible which could support older analog equipment such as CRT monitors. While the DVI-D and DVI-A formats have only one (digital or analog) signal present, the DVI-I format has both analog and digital signal available concurrently. Since the DVI has this flexibility along with improved performance, it is widely accepted as the new connectivity standard. Single link DVI supports a bandwidth of up to 165 MHz and the dual link can support a bandwidth of up to 330 MHz which is twice that of single link. For most consumer applications today, single link DVI is more than sufficient, but that could change in the future. As the bandwidth capability increases, cable length limitation becomes critical. The standard specification for DVI is upto 5 meters (16.4 ft.). Higher quality copper cables and fiber optic cables are available which can support DVI signal to much greater distances [21].

The data format used in DVI is based on the Panel Serial format that uses TMDS (Transition Minimized Differential Signaling). Each DVI link consists of four twisted pair of wires to transmit 24 bits per pixel. One each with a color code of red, blue and green, and one for a clock signal. The clock signal is virtually the same as that of the analog video signal, while the picture is sent electronically line by line with blanking intervals separating each line and frame, without using any packetization method. Also DVI does not use compression and if a modified part of an image is to be transmitted, then the only choice DVI has is to retransmit the entire frame again. It also supports analog connection with optional compatibility to VGA interface. Most commonly found in computer devices, it is also present in electronic devices like television set even though most of the newer sets come with HDMI all-digital connector [22].

3.2 Comparison of DVI with Other Older Technologies

One of the most commonly used standard before the development of DVI was Video Graphics Array (VGA). VGA was designed especially for CRT-based devices and hence discrete time was not taken into consideration. In VGA while the horizontal line of the image is transmitted, it varies the output voltage to represent the desired brightness level. The CRT responds to these voltage level changes by varying the intensity of the electron beam as it scans from one end of the screen to the other [23, 24].

In digital systems, the brightness value for each pixel needs to be selected to display the image properly. The decoder achieves this by sampling the input signal voltage at regular intervals. Since these are purely digital signals there are some inherent problems associated like distortion in the signal if the sample is not taken exactly from the center of the pixel and possibility of crosstalk interference.

DVI as a digital system has an entirely different approach in which the brightness level of each pixel is transmitted in binary code. With this technique, every pixel in the output buffer of the source device will correspond directly to one pixel in the display device. DVI is also free from noise and distortion which is inherent in analog signals.

During the in depth analysis of the differences between analog and digital, the key difference is the PC hosts and LCD monitors are digital. The CRT monitors have analog inputs. As a result the PC has to convert digital data into analog voltages before transmitting it to a CRT monitor. LCD monitors with analog inputs will require two such conversions since they are digital devices. The first conversion happens at the PC, which must convert digital data into analog voltages similar to CRT monitors and the second conversion at the LCD monitor to convert the analog voltages to digital values before it can be displayed. These conversions result in the image quality being degraded. Additionally, analog signals are prone to errors and noise. Lastly, the Analog-to-digital converter (ADC) at the display side increases the cost of monitors. Figure 3.1 shows the pictorial representation of the above discussion.

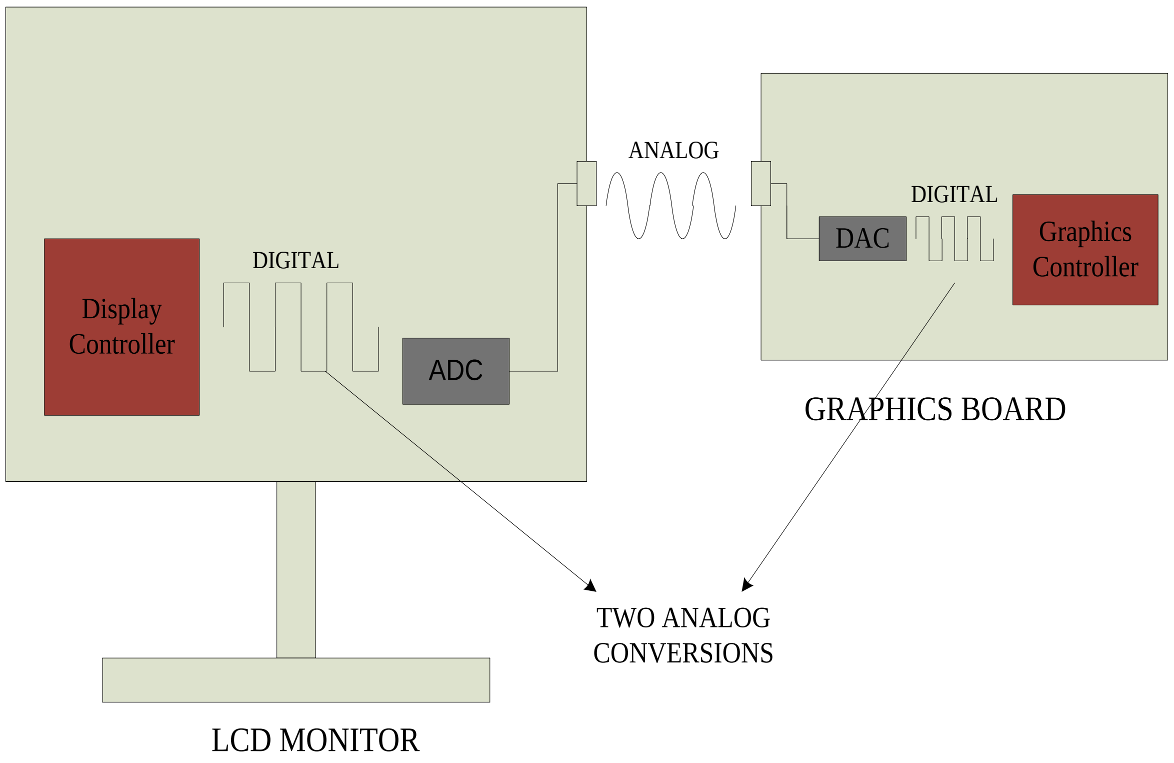


Figure 3.1. Analog conversions when using VGA connectors for LCD monitors.

In case of digital-only connector, pure digital signal is sent from the graphics controller directly to the LCD monitor using TMDS transmitter and receiver with no data conversions. This also ensures that there are no errors which results from conversion back and forth. Figure 3.2 shows the pictorial representation.

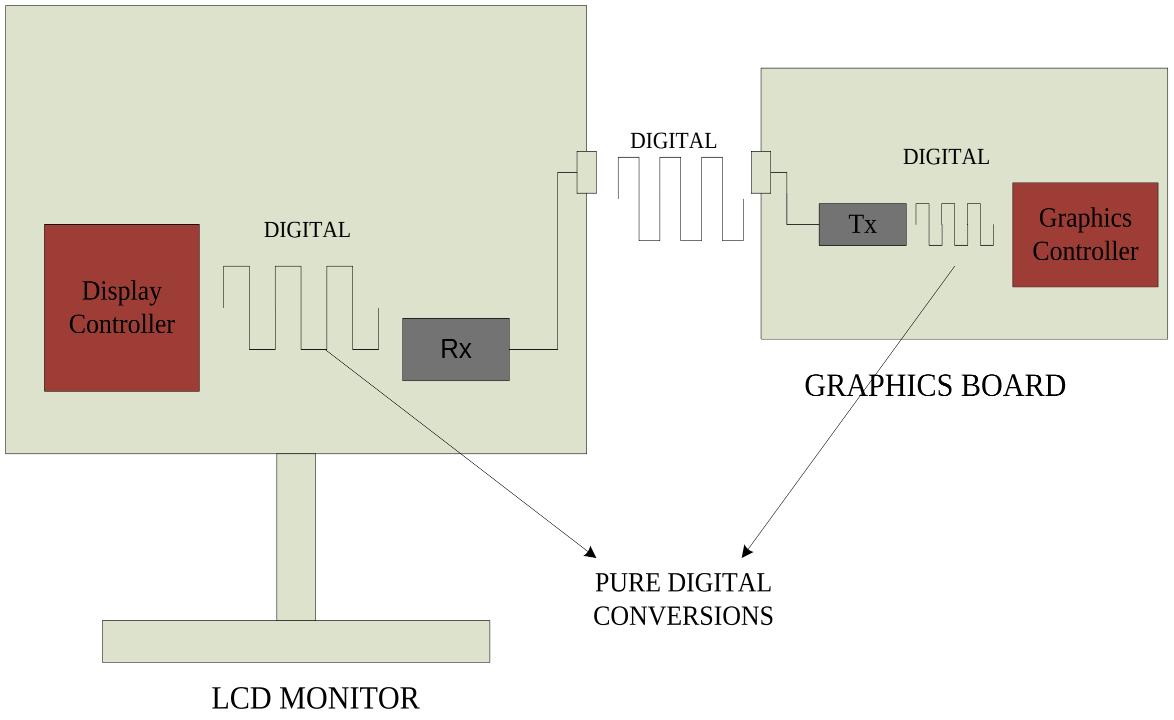


Figure 3.2. Digital-only conversion when using the DVI connector.

3.3 DVI Technology Basics

The DVI technology basics can be subdivided into DVI basics, DVI signaling and display basics. They are explained in detail below.

3.3.1 DVI Basics

The most popular use of DVI is in the digital interface between the LCD monitor and the graphics board. The DVI standard consists of four main components namely

1. TMDS transmitter
2. TMDS receiver
3. DVI connector
4. DVI cable

The TMDS transmitter or receiver can be either integrated within the graphics/display controller or implemented as a discrete component [25].

- *Graphics Controller*: In order to display on the LCD monitor, three components of color namely Red, Green and Blue (RGB) are required. Each of the color components uses 8 bits and that gives 256 shades to be selected from. The 24 bits of parallel output from the graphics controller is input to the TMDS transmitter.

- *TMDS transmitter*: Once the transmitter receives 24 bits of parallel data, it encodes and serializes it for transmission. Each RGB color component and the clock are transmitted on separate channels in a process known as differential signaling. There are a total of four channels (differential pairs): three for RGB and fourth for the clock.
- *TMDS receiver*: The TMDS receiver receives the serial data and the clock via the four channels, decodes it and outputs the data in parallel to the display controller. A simple block diagram of the TMDS Logical link is as shown in Figure 3.3 [26].

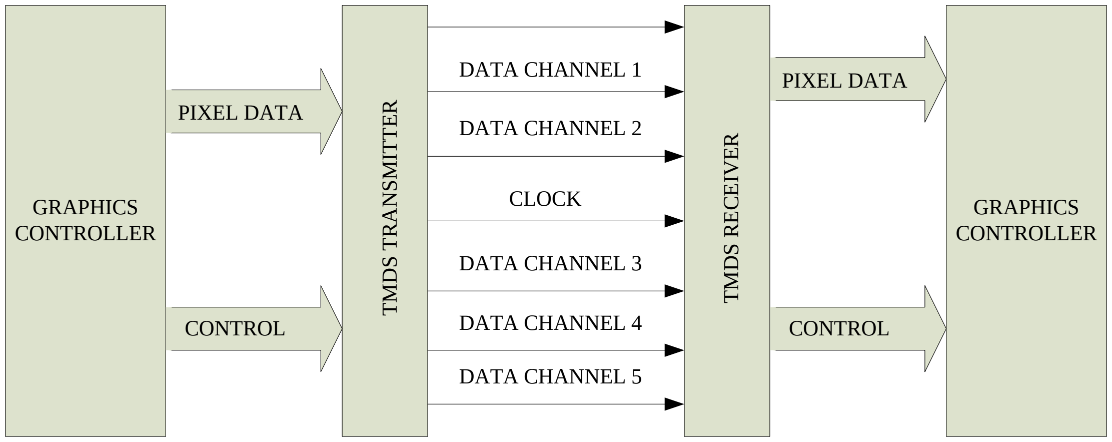


Figure 3.3. TMDS logical link. Source: Digital Display Working Group (DDWG), "Digital Visual Interface," Revision 1.0, Last modified April 2, 1999, http://www.cs.unc.edu/Research/stc/FAQs/Video/dvi_spec-V1_0.pdf.

The pin description will be explained in the next section. The discussion of encoding, serialization, TMDS are not within the scope of this research. More details about it can be found in [25, 26].

3.3.2 DVI Signaling

There are three different DVI connectors available namely DVI-I: supports both digital and analog, DVI-D: supports digital only, DVI-A: supports analog only. DVI-I is the most commonly used one since it gives the flexibility to drive either a digital or an analog display interface. To drive an analog display interface, a VGA cable with a special connector is used at the graphics board end.

DVI-I connector has a total of 29 pins available, out of which 18 pins are used for TMDS, including the pins for dual-link configuration, 6 pins for analog and 5 pins for Plug and Play. DVI connector pin out is as shown in Figure 3.4 [27]. DVI-I analog data, digital

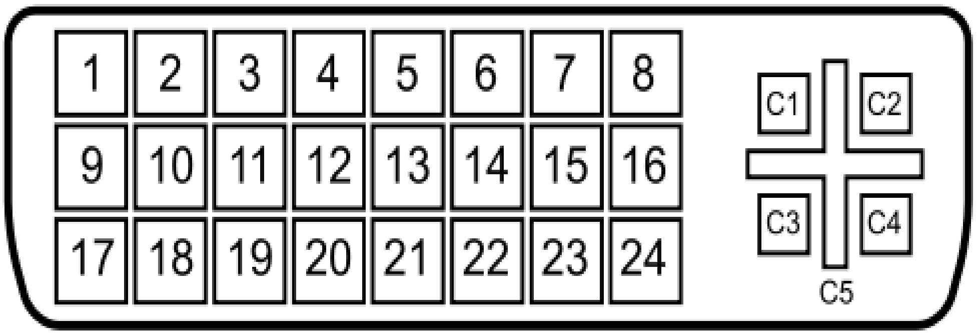


Figure 3.4. DVI connector pin out. Source: Wikipedia, "Digital Visual Interface," Last modified December 26, 2013, http://en.wikipedia.org/wiki/Digital_Visual_Interface.

data, shielding, plug & play, digital clock pin descriptions are depicted in Table 3.1 & 3.2 [27].

Table 3.1. DVI-I Analog Data Pins and Their Description

PIN	SIGNAL
C1	Analog red
C2	Analog green
C3	Analog blue
C4	Analog horizontal sync
C5	Analog Ground Return for R, G and B signals

Source: Wikipedia, "Digital Visual Interface," Last modified December 26, 2013, http://en.wikipedia.org/wiki/Digital_Visual_Interface.

3.3.3 Display Basics

One of the major differences between a CRT and LCD monitor is that the LCD uses a fixed array of pixels. LCD has the best image quality when viewed at its native resolution and implements scaling to support multiple resolutions below its native resolution, whereas the CRT has an advantage of supporting multiple resolutions.

A LCD monitor consists of a number of pixels which form the native resolution depending on the LCD monitor we use. Each such pixel is comprised of three sub pixels namely Red, Green and Blue. Figure 3.5 shows this representation.

Table 3.2. DVI-I Digital Data, Shielding, Plug & Play, Digital Clock Pins and Their Description

PIN	SIGNAL	PIN	SIGNAL	PIN	SIGNAL
1	TMDS Data 2- Digital Red- (link1)	**9**	TMDS Data 1- Digital Green- (link1)	**17**	TMDS Data 0- Digital Blue- (link1) and digital sync
2	TMDS Data 2+ Digital Red+ (link1)	**10**	TMDS Data 1+ Digital Green+ (link1)	**18**	TMDS Data 0+ Digital Blue+ (link1) and digital sync
3	TMDS data Shield 2/4	**11**	TMDS data Shield 1/3	**19**	TMDS data Shield 0/5
4	TMDS Data 4- Digital Green- (link2)	**12**	TMDS Data 3- Digital Blue- (link2)	**20**	TMDS Data 5- Digital Red- (link2)
5	Data 4+ Digital Green+ (link2)	**13**	TMDS Data 3+ Digital Blue+ (link2)	**21**	TMDS Data 5+ Digital Red+ (link2)
6	DDC Clock	**14**	+5 V Power for monitor when in stand by	**22**	TMDS clock shield
7	DDC Data	**15**	Ground Return for pin 14 and analog sync	**23**	TMDS clock+ Digital clock+ (Link1 and Link 2)
8	Analog vertical sync	**16**	Hot plug detect	**24**	TMDS clock- Digital clock- (Link1 and Link 2)

Source: Wikipedia, “Digital Visual Interfac,” Last modified December 26, 2013, http://en.wikipedia.org/wiki/Digital_Visual_Interface.

A graphics board has to constantly send out a stream of data to tell the display controller depending on the pixel needs to be turned on and is coordinated via the pixel clock. The pixel data (which is 24 bits) determines the color of the pixel by the 256 levels set for each RGB sub pixel.

The pixel clock provides the display controller with all the information required in lighting up a pixel, row-by-row and frame-by-frame. The pixel clock is determined by calculating the total number of pixels multiplied by the refresh rate. There is also the blanking time, etc. which adds more overhead to the pixel clock time.

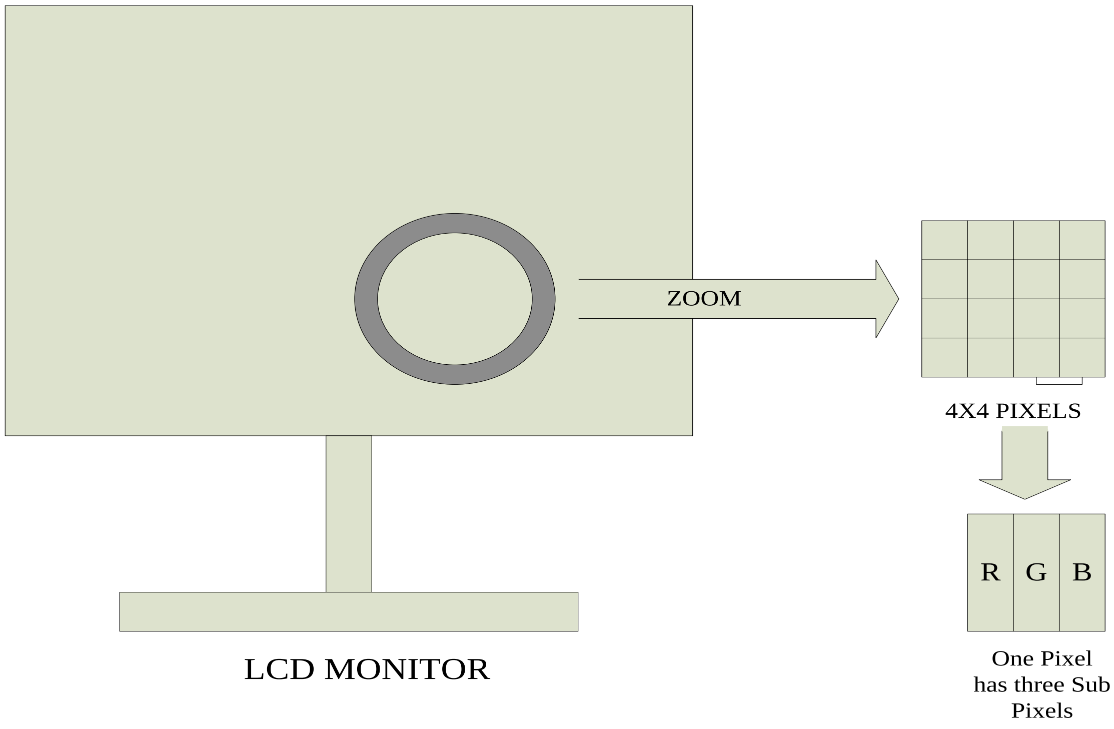


Figure 3.5. Pixels and sub pixel.

3.4 IMPLEMENTING REAL TIME AUDIO DISPLAY USING FPGA BOARD

The Real time Audio display using the DVI monitor involves understanding the basics of DVI (discussed in the previous section) and interfacing the current phase with the Audio Codec interface of the first phase. The RTL schematic representation of this phase is as shown in Figure 3.6

The Real time Audio display using the DVI monitor has three important interface modules namely

1. Top Level AC97 Interface
2. Receiver Interface
3. DVI Video Interface

3.4.1 Top Level AC97 Interface

The Top level AC97 interface has the modules described in the first phase. Except in this phase, there will be no recording and playback but just the loopback. The audio loopback output from this module will serve as input to the next module namely the receiver module.

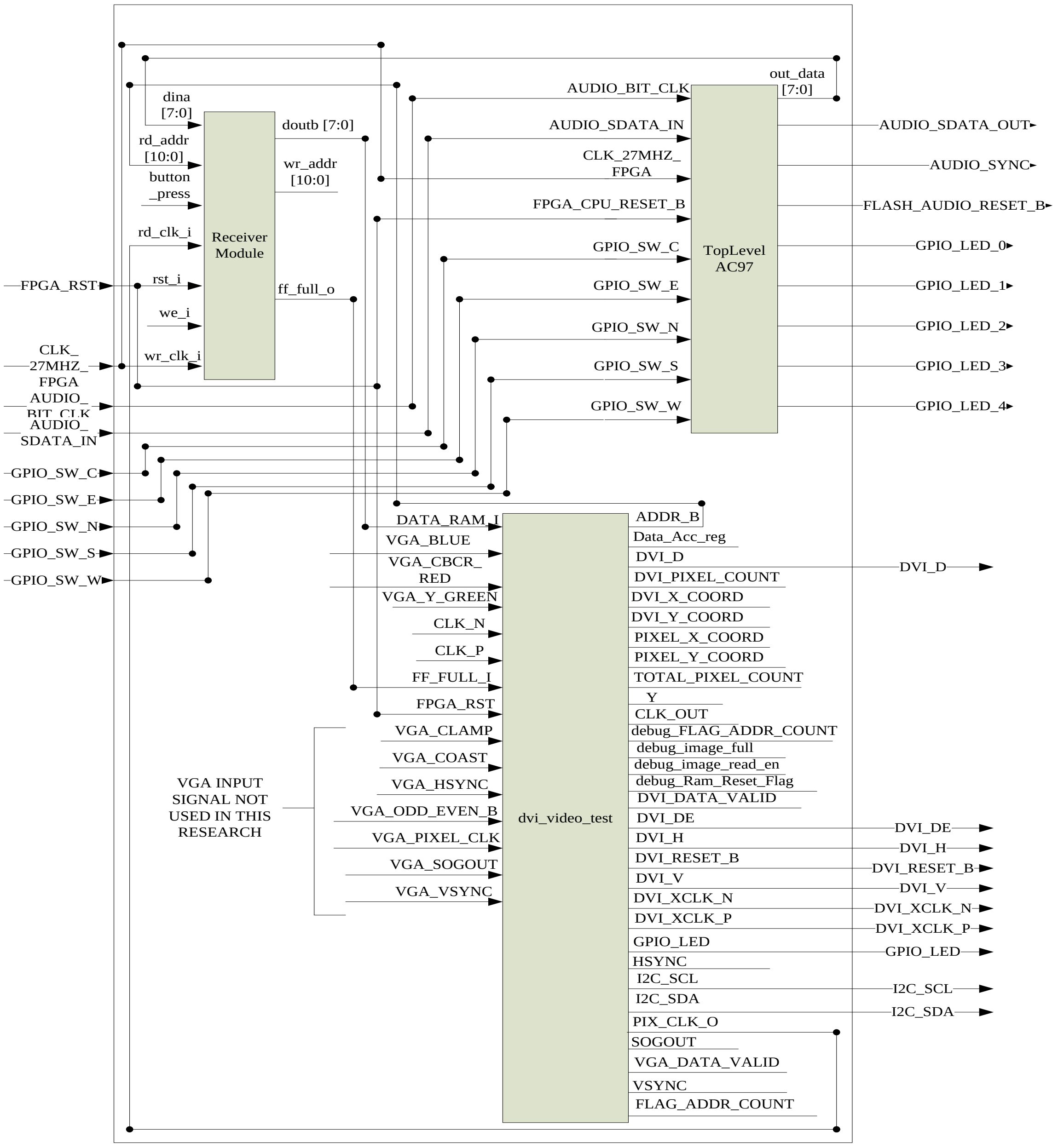


Figure 3.6. RTL schematic of real time audio signal display.

This module will output the audio whenever the GPIO_SW_C (loopback) pushbutton switch is pressed. There are some signals added to this module to support the receiver interface.

3.4.2 Receiver Interface

This module receives the input from the Top level AC97 interface, stores it and forwards to the DVI video interface for display. The data has to be sent in synchronization with the pixel clock of the DVI interface, since it has to display the input audio at the DVI monitor. This module also consists of synchronizing the button press with the Display. This is necessary for simultaneous audio output from speakers and displaying it on the monitor. The RTL schematic representation of this interface is as shown in Figure 3.7.

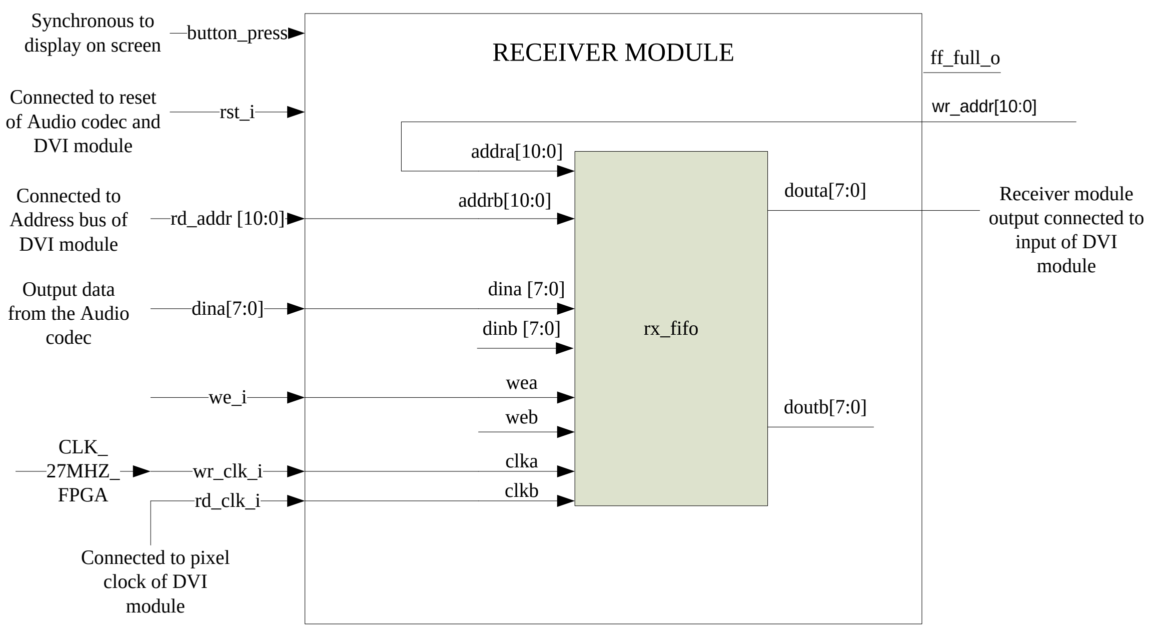

Synchronous to display on screen
button_press
RECEIVER MODULE
ff_full_o
Connected to reset of Audio codec and DVI module
rst_i
wr_addr[10:0]
addra[10:0]
Connected to Address bus of DVI module
rd_addr [10:0]
addrb[10:0]
douta[7:0]
Receiver module output connected to input of DVI module
Output data from the Audio codec
dina[7:0]
dina [7:0]
dinb [7:0]
rx_fifo
we_i
wea
web
doutb[7:0]
CLK_ 27MHZ_ FPGA
wr_clk_i
clka
rd_clk_i
clkb
Connected to pixel clock of DVI module

Figure 3.7. RTL schematic of receiver interface.

As shown in the Figure 3.7, we can see that this is the intermediate interface between the Top level AC97 and the DVI video. It includes several signals: their description & functions are as described below.

- **button_press:** This is used to synchronize the GPIO_SW_C (the loopback button) with the display_2screen signal of the DVI video interface to ensure that button press is recognized by the video module for display on monitor.
- **rst_i:** this signal is used for synchronizing FPGA_CPU_RESET_B of the Top level AC97 module and the FPGA_RST of the DVI video module with the onboard reset button.
- **rd_addr:** It is synchronized with the address bus on the DVI module to read the data from specific addresses to be displayed on screen.
- **dina:** 8 bit data output from the Top level AC97 module is received and stored in the rx_fifo and given as input to the DVI module on the pixel clock so that the data to be displayed is available.
- **rd_clk_i:** It is connected to the pixel clock of the DVI module
- **douta:** 8 bit data output from the receiver module given as input to the DVI video module.

All the signals explained so far except the first two are the signals related to the rx_fifo module which is instantiated in the Receiver module. The rx_fifo is a Xilinx Core generated BRAM IP core. It uses the following specifications

- ✓ True Dual port RAM with single input port and single output port.
- ✓ Two address ports - "addra" for writing and "addrb" for reading.
- ✓ Write and read width is 8 bits and depth 2048 bits. It uses the write first operating mode.
- ✓ This phases use one 18Kb BRAM.
- ✓ The address width is 11 bits depending on the memory used to store and retrieve data for the display.

3.4.3 DVI Video Interface

This is the most important interface of the Real time Audio display on DVI monitor. The DVI code written in VHDL is used and has many module instantiations. These modules help in configuring the onboard VGA chip: Analog devices AD9980 [28], the DVI chip Chrontel CH7301C [29] and the IIC interface. The Chrontel CH7301C [29] is controlled by video IIC bus. The DVI connector supports the IIC protocol to allow the board to read the monitor's configuration parameters. The DVI circuitry utilizes the Chrontel CH7301C capable of 1600 X 1200 resolution with 24 bit color. The video interface chip drives both the digital analog signals to the DVI connector. The Virtex-5 ML506 FPGA board supports only DVI-D [18], so a DVI monitor can be connected to the board directly. A VGA monitor can also be connected to the board using an active DVI-to-VGA adaptor. For the purpose of this research we are interested in configuring the DVI chip Chrontel CH7301C.

Before understanding the working of this interface let us look at some of the features, pin descriptions and register configurations of the Chrontel CH7301C [30] display controller device. It has the following features [29]

- ✓ It is offered in a 64 pin LQFP package which accepts a digital graphic input signal, encodes and transmits data through a DVI.
- ✓ It accepts data over one 12-bit wide variable voltage data port. The port supports different formats including the RGB and YCrCb.
- ✓ It supports pixel rate of up to 165MHz, supporting the UXGA resolution display. No scaling of input data is performed on the data output to the DVI device.

- ✓ The DVI processor includes a low jitter PLL for generation of high frequency serialized clock and all circuitry required for encoding, serializing and transmitting data.
- ✓ Color space conversion from YCrCb to RGB is supported in both DVI and VGA bypass mode.

The main pins that are used and their functional descriptions are listed below

a. **DVI_DE:** It is the input data enable pin. It accepts active high data enable signal when the video data is input to the device and is low rest of the time. The levels are 0 and DVDDV, and VREF signal is used as the threshold level.

b. **DVI_RESET_B:** When this pin is low, device is held in the power-on reset condition. When this pin is high, reset is controlled through the serial port register.

c. **DVI_VSYNC:** A buffered version of the VGA vertical sync can be acquired from this pin.

d. **DVI_HSYNC:** A buffered version of the VGA horizontal sync can be acquired from this pin.

e. **DVI_D [11:0]:** These pins accept 12 data inputs from a digital video port of a graphics controller.

f. **DVI_XCLK_N, DVI_XCLK_P:** These inputs form the differential clock signal input to the CH7301C for use with DVI_DE, DVI_D [11:0] data. If differential clocks are not available, DVI_XCLK_N should be connected to VREF.

These signals are nothing but the DVI controller connections. The CH7301C is capable of being operated in two different modes. Firstly in RGB bypass mode, data sync and clock signals are input to CH7301C from a graphics device and bypassed directly to D/A converters to implement a second CRT DAC function. External sync signals must be supplied from the graphics device. Secondly in DVI Output mode, multiplexed input data, sync and clock signals are input to the CH7301C from the graphics controller's digital output port. The 12 data inputs support 5 different multiplexed data formats, each of which can be used with different clocking options. The details about these modes of operation, the different data formats, clocking, register control and registers for read and write can be found in Chrontel CH7301C data sheet and application notes. The RTL schematic representation of DVI interface is as shown in Figure 3.8.

Figure 3.8 shows most of the DVI signals that were described before. We can now look at how the data will be sent to the display controller to be mapped on to the DVI

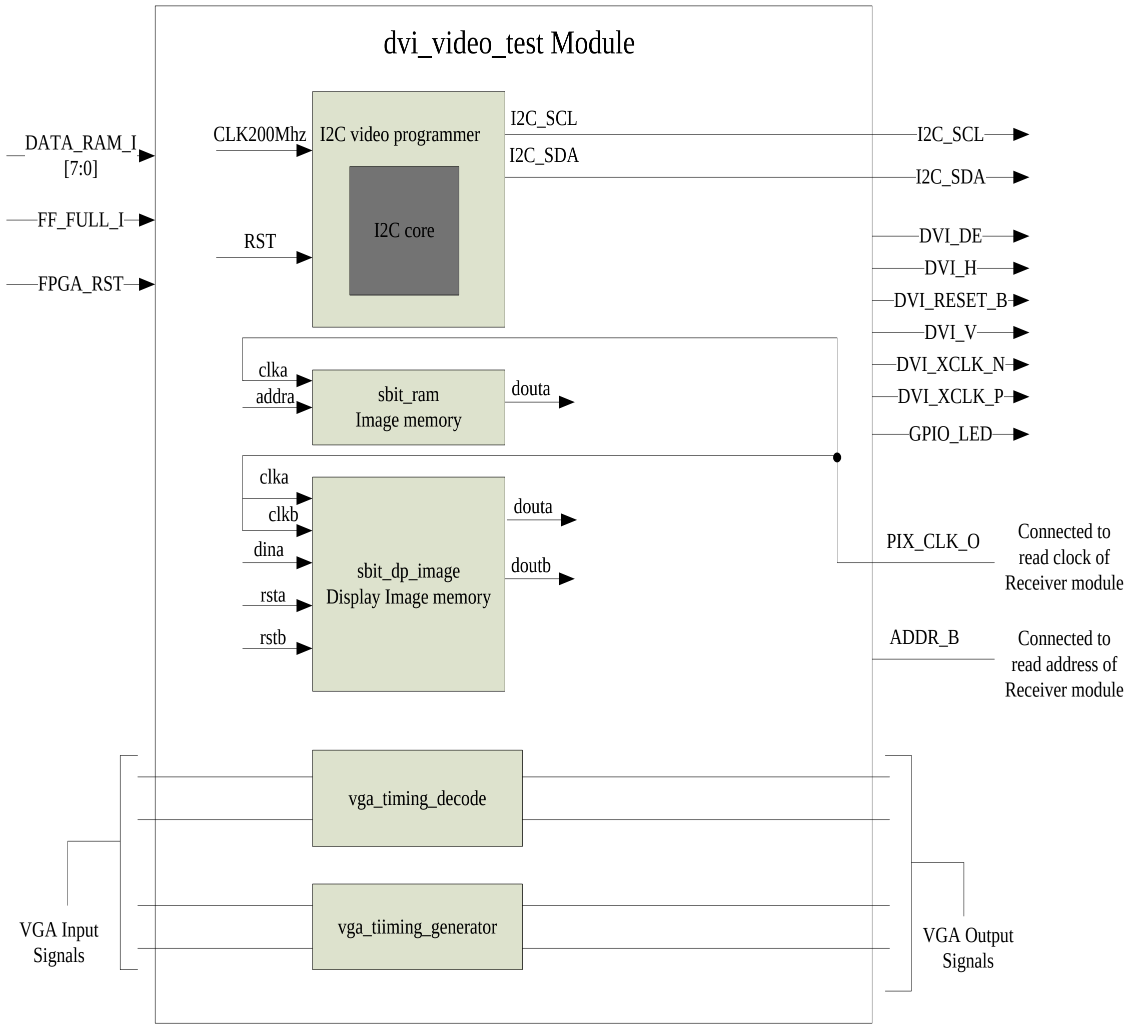


Figure 3.8. RTL schematic of DVI video interface.

monitor for display. There are certain modules necessary to display the audio data on the monitor. The interfaces and their functions are described briefly in next section.

3.4.3.1 I2C VIDEO PROGRAMMER

The I2C programmer is responsible for transfer of data between the graphics controller and the DVI monitor. It interfaces the I2C core module and helps in configuring the VGA IN I2C codec and the DVI OUT I2C codec.

3.4.3.2 SBIT_RAM IMAGE MEMORY

The sbit_ram image memory is Xilinx core generated BRAM memory used for address count of the 256 values of the audio signal that is to be displayed on the monitor. It uses the following specifications

- ✓ Single port RAM with single input port and single output port.
- ✓ Single address port for the address count.
- ✓ Write and read width is 8 bits and depth 256 bits.
- ✓ This phases use one 18Kb BRAM.
- ✓ The address width is 8 bits

3.4.3.3 SBIT_DP_IMAGE DISPLAY IMAGE RAM

The sbit_dp_image display image RAM is the Xilinx core generated BRAM memory used for address count of the 256 values of the audio signal that is to be displayed on the monitor. It uses the following specifications

- ✓ True dual port RAM with two input and output ports.
- ✓ Two address ports, one for the image count and the other for the pixel.
- ✓ Write and read width is 1bit to display at each pixel on the monitor and read and write depth of 163840 bits corresponding to the display resolution of 640 X 480
- ✓ This phase uses five 36Kb BRAMs.
- ✓ The address width is 18 bits which corresponds to 640X 480 resolution.

The code snippet (Code Snippet 1) in the DVI video interface that corresponds to displaying of the audio data at each corresponding pixel on the DVI monitor is as given below.

```
process (DATA_RAM_I)
begin
IF (FPGA_RST = '0' or col = "1010000000") THEN
Data_Acc <= (OTHERS => '0');
ELSIF (DATA_RAM_I(7)='1') then
Data_Acc  <=  256 - (("00" & unsigned(DATA_RAM_I(6 DOWNTO 0))));--(127  + ("00" &
unsigned(data(6 DOWNTO 0))));
ELSIF (DATA_RAM_I(7)='0') then
Data_Acc <= (127 - ("00" & unsigned(DATA_RAM_I(6 DOWNTO 0))));
ELSE
Data_Acc <= Data_Acc;
END IF;
END PROCESS;
```

3.4.4 CODE SNIPPET 1: DVI DISPLAY

The audio signal consists of positive and negative values at each point when the signal is quantized. We use this method to read the value at each instant of time and is displayed simultaneously. The first bit of the 8 bit register DATA_RAM_I tells whether the value is positive or negative. If positive, it is displayed above the zero axis and if negative, displayed below the zero axis. When displaying on screen, the y-axis contains 256 points corresponding to the value of the audio data and 640 points on the x-axis corresponding to the resolution of the display.

CHAPTER 4

SPECTROGRAM REPRESENTATION OF THE AUDIO SIGNAL USING SHORT TERM FOURIER TRANSFORM (STFT)

Spectrogram in general is defined as the time varying spectrum of a signal. Typically they are used to identify the phonetic sounds, to analyze the cries of animals, but are also used in other fields like speech processing, sonar, seismology, etc. Spectrograms can be created using two methods. First is using a series of band pass filters to form the approximated filter bank. The second is short time Fourier transform (STFT) calculated from the time signal. The STFT is used in this research and the detailed discussion is continued below.

4.1 INTRODUCTION TO SHORT TIME FOURIER TRANSFORM (STFT)

The STFT is used to find the spectrum of a signal as it changes over time. The Fourier transform of a signal x(t) and its inverse are defined by [31]

$$X(\Omega) = \int_{-\infty}^{\infty} x(t)\, e^{-j\Omega t}\, \mathrm{d}t \text{ and } x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\Omega)\, e^{j\Omega t}\, \mathrm{d}\Omega \qquad (4.1)$$

The phase of the Fourier transform contains information about the time distribution of the spectral content.

The idea behind the short-time Fourier transform is to apply the Fourier transform to a small portion of the original signal. This can be done by introducing a sliding window function w(t) which localizes and truncates (and weight) the analyzed signal x(t). The Fourier transform is calculated for the localized part of the signal gives the spectral content of that portion within the time interval defined by the width of the window function. The STFT, a time-frequency representation of the signal is then obtained by sliding the window along the signal. The STFT calculation is illustrated as in Figure 4.1 [31].

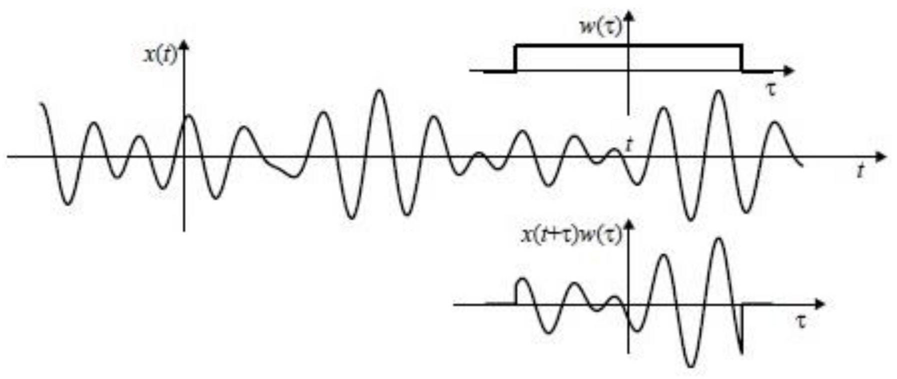


Figure 4.1. Illustration of the signal localization in the STFT calculation. Source: Ljubisa, Stankovic, Dakovic, Milos, and Thayaparan Thayananthan, Time-Frequency Signal Analysis with Applications, Norwood: Artech House, 2013.

The STFT calculation represented in analytic form is [31]

$$X(t,\Omega) = \int_{-\infty}^{\infty} x(t+\tau)\, w(\tau)\, e^{-j\Omega\tau}\, d\tau \tag{4.2}$$

Where $w(\tau)$ is the window function $x(t)$ is the signal to be transformed $X(\tau,\Omega)$ is the STFT.

The Spectrogram, is defined by [31]

$$S(t,\Omega) = |X(t,\Omega)|^2 \tag{4.3}$$

Where $S(\tau,\Omega)$ is the Spectrogram $X(\tau,\Omega)$ is the STFT.

When the spectrogram is represented, the horizontal axis corresponds to time and the vertical axis corresponds to frequency, a third dimension indicating the amplitude of a particular frequency at a particular time is represented by the intensity or color of each point in the image (spectrogram). The higher is the volume of the audio signal, the higher will be the corresponding amplitude on the display. Frequency is measured in Hertz (Hz) or Kilo Hertz (kHz).

4.2 WINDOW FUNCTIONS

Window functions are used typically for spectral analysis, filter design and beamforming. In signal processing, a window function is a mathematical function that is zero-valued outside a chosen interval. A function that is constant inside the interval and zero elsewhere is called a rectangular window, which describes the shape of its graphical representation. A signal or a function multiplied by the window function results in zero value outside the window interval and what remains is the part where they overlap.

There are a number of window functions which are used depending on the application. Most commonly used window functions are listed and briefly explained below.

- **Rectangular Window:** The simplest window is the rectangular window, defined by [31]

$$w\ \tau\ = \begin{matrix} 1 & \tau\ < T \\ 0 & elsewhere \end{matrix} \tag{4.4}$$

  And its Fourier transform is [31]

$$W_R(\Omega) = {}_{-T}^{\ T} e^{-j\Omega\tau}\ d\tau = \frac{2sin(\Omega T)}{\Omega} \tag{4.5}$$

  The rectangular window function has very strong and oscillating side lobes in the frequency domain, since the function sin(ΩT)/Ω converges very slowly, toward zero, in Ω as Ω→±∞. Due significant discontinuity in time domain, at t =± T, there is slow convergence in the Fourier domain. The main lobe width of Fourier transform is d_Ω=2π/T. Other window functions have been introduced to enhance signal localization in the frequency domain.

- **Triangular Window:** It is defined by [31]

$$w\ \tau\ = \begin{matrix} 1 - \ \tau/T & for\ \ \ \tau\ < T \\ 0 & elsewhere. \end{matrix} \tag{4.6}$$

  The triangular window can be considered as a convolution of the rectangular window of duration T with itself. Its Fourier transform is a product of two Fourier transforms of the triangular window of the width T [31],

$$W_T(\Omega) = \frac{4\sin^2(\Omega T/2)}{\Omega^2} \tag{4.7}$$

  Convergence of this function toward zero is of the $1/\Omega^2$ order as Ω→±∞. It is a continuous function of time, with discontinuities in the first derivative at t=0 and t =± T. The mainlobe of this window function is twice wider in the frequency domain than that of the rectangular window. As a result d_Ω=4π/T.

- **Hann Window:** The Hann window is of the form [31]

$$w\ \tau\ = \begin{matrix} 0.5(1+\cos(\pi\tau/T)) & for\ \ \ \tau\ < T \\ 0 & elsewhere. \end{matrix} \tag{4.8}$$

  Since $\cos(\pi\tau/T) = [\exp(j\pi\tau/T) + \ \exp(-j\pi\tau/T)]/2$, the Fourier transform of this window is related to the Fourier transform of the rectangular window of the same width as shown below [31]

$$W_H(\Omega) = \frac{1}{2}W_R(\Omega) + \frac{1}{4}W_R(\Omega - \pi/T) + \frac{1}{4}W_R(\Omega + \pi/) = \frac{\pi^2\sin(\Omega T/2)}{\Omega(\pi^2 - \Omega^2T^2)} \tag{4.9}$$

  $W_H(\Omega)$ decays in frequency asΩ^3, much faster than$W_R(\Omega)$. Les us consider for example a Hann window w(τ) of the width 2T. Approximately we may estimate that its Fourier transform$W_H(\Omega)$ is nonzero within the main lattice $\ \Omega\ \ < 2\pi/T$ only, since the sidelobes decay very fast. Then d_Ω=4π/T. A more detailed explanation can be found here [31].

- **Hamming Window**: The Hamming window is defined as [31]

$$w\ \tau\ = \begin{matrix} 0.54 + 0.46\cos(\pi\tau/T)) & for\quad \tau\ < T \\ 0 & elsewhere. \end{matrix} \tag{4.10}$$

  The relation between Hamming and the rectangular window transform is same as in the case of Hann window. The Hamming window was derived starting from [31]

$$w\ \tau\ = a + (1-a)\ \cos(\pi\tau/T) \tag{4.11}$$

  This window has several sidelobes, next to the mainlobe, lower than the previous two windows. However, since it it not continuous at t =± T, its decay frequency, as Ω→±∞, is not fast. A more detailed explanation of these windows and some more can be found here [31].

4.3 SPECTROGRAM IMPLEMENTATION USING FPGA BOARD

The Spectrogram display on the DVI monitor involves understanding the basics of STFT, windowing techniques, overlapping windows to eliminate loss at the edges of window, ZBT SRAM of ML506, color display and character display for the spectrogram. The STFT and window functions were briefly explained above. They will be explained in detail along with the implementation of the Verilog modules necessary to perform these functions in the following section. A simplified RTL schematic representation of this phase is as shown in Figure 4.2.

4.3.1 Phase Locked Loop (PLL) Interface

Phase Locked loop is generated using the Xilinx IP core generator. It generates a 54MHz clock signal form the ML506 board clock which is used for FFT transformation synchronous clock. It also has the standard 27MHz board clock which is used for the pushbutton interface.

4.3.2 Key Interface

Key interface brings together all the modules necessary for the functionality of pushbuttons used for loopback, record & playback of audio & video and the LED indicators representing these functions. The functions were explained in Chapter 1 but since the functionality is also integrated with video display, it's briefly explained again below.

a. **GPIO_SW_N:** The general purpose North push button is implemented to allow the user to increase the volume of the output audio connected to headphone or speaker.

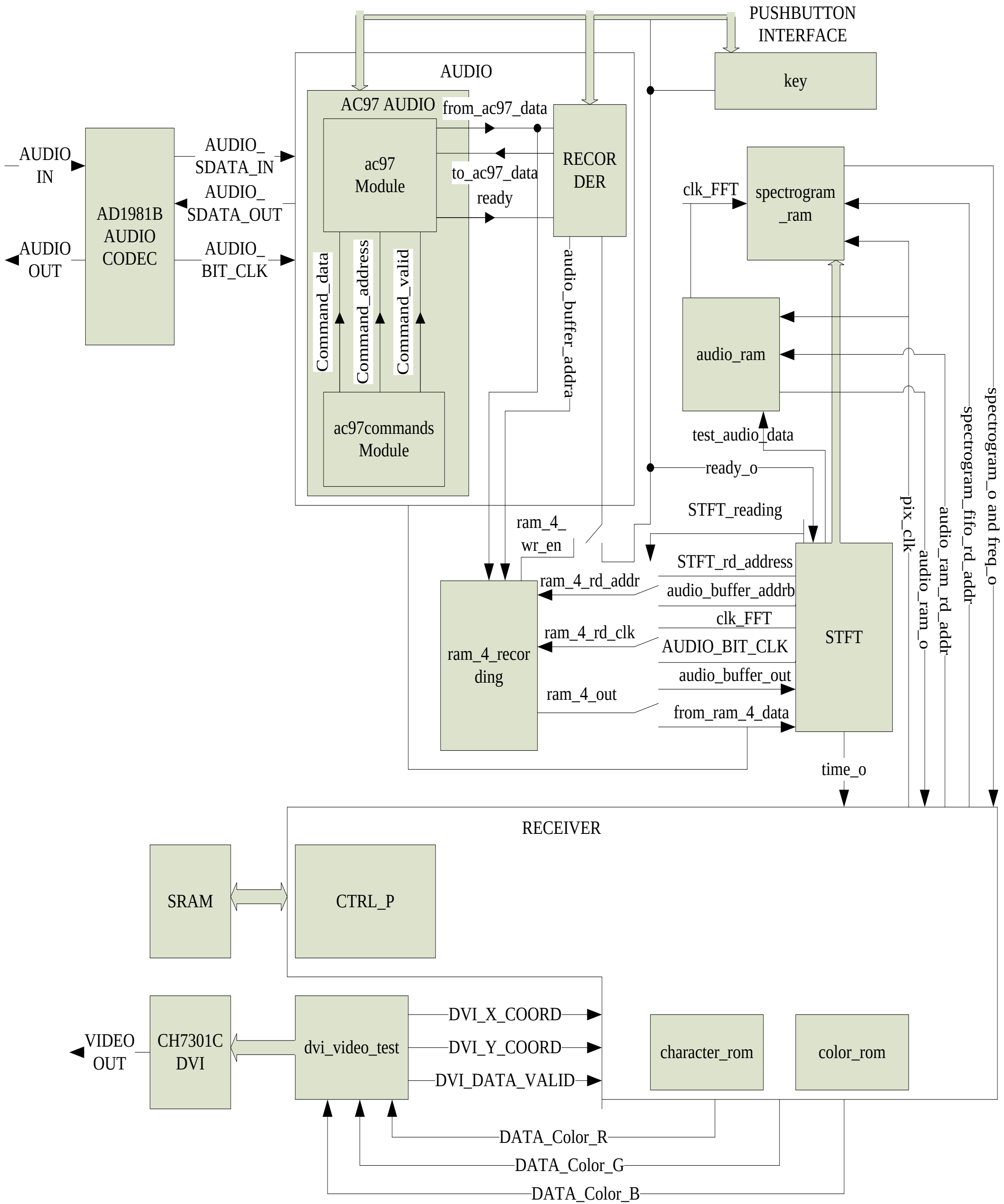


Figure 4.2. RTL schematic of the spectrogram calculation unit.

b. **GPIO_SW_S:** The general purpose South push button is implemented to allow the user to decrease the volume of the output audio connected to headphone or speaker.

c. **GPIO_SW_C:** The general purpose Center push button is used to implement the Loopback functionality. When this button is pressed and held, the output speakers will loopback the audio that is input at the microphone. The LCD monitor shows the input audio in time domain and the real time Audio Spectrogram.

d. **GPIO_LED_4** turns ON when this button is pressed indicating loopback functionality. GPIO_LED_1 is ON indicating the memory is empty.

e. **GPIO_SW_E:** The general purpose East push button is used to implement the Record functionality. When this button is pressed and held, the real time audio input is received and stored in the First in First out (FIFO) Block Random Access memory (BRAM). Also the Spectrogram of the recorded audio is calculated and stored in the memory. It records about 8 seconds of audio data.
GPIO_LED_2 turns ON as soon as GPIO_SW_E is pressed indicating record functionality. As soon as 8 seconds is passed, GPIO_LED_0 turns ON indicating the memory is full and recording is complete.

f. **GPIO_SW_W:** The general purpose West push button is used to implement the Playback functionality. When this button is pressed and held, the audio data saved in the Block Random Access Memory (RAM) is sent to the output headphone jack and is played back using the speakers. The LCD monitor displays the saved audio and the corresponding Spectrogram.
GPIO_LED_3 turns ON when this button is pressed indicating playback functionality.

4.3.3 Audio Interface

The Audio interface is explained in detail in Chapter 1. Some extra signals are added to the sub modules which help in interfacing with the STFT, and the display interface.

4.3.4 STFT Interface

The STFT Interface is responsible for calculating the STFT. This module reads the audio data from ram_4_recording while calculating the continuous FFT transform. It instantiates several sub modules which control the data being read from BRAM ram_4_recording and in calculating the transform. The audio data read from BRAM ram_4_recording is written into another BRAM Audio_RAM for displaying the audio data. The STFT result is written into the BRAM Spectrogram_RAM for displaying the Spectrogram.

STFT Interface instantiates 4 sub modules namely *Ctrl_STFT*, *Ctrl_FFT*, *FFT* and *Hanning_4096* and a top level module called STFT. The RTL schematic of the STFT interface is as shown in Figure 4.3.

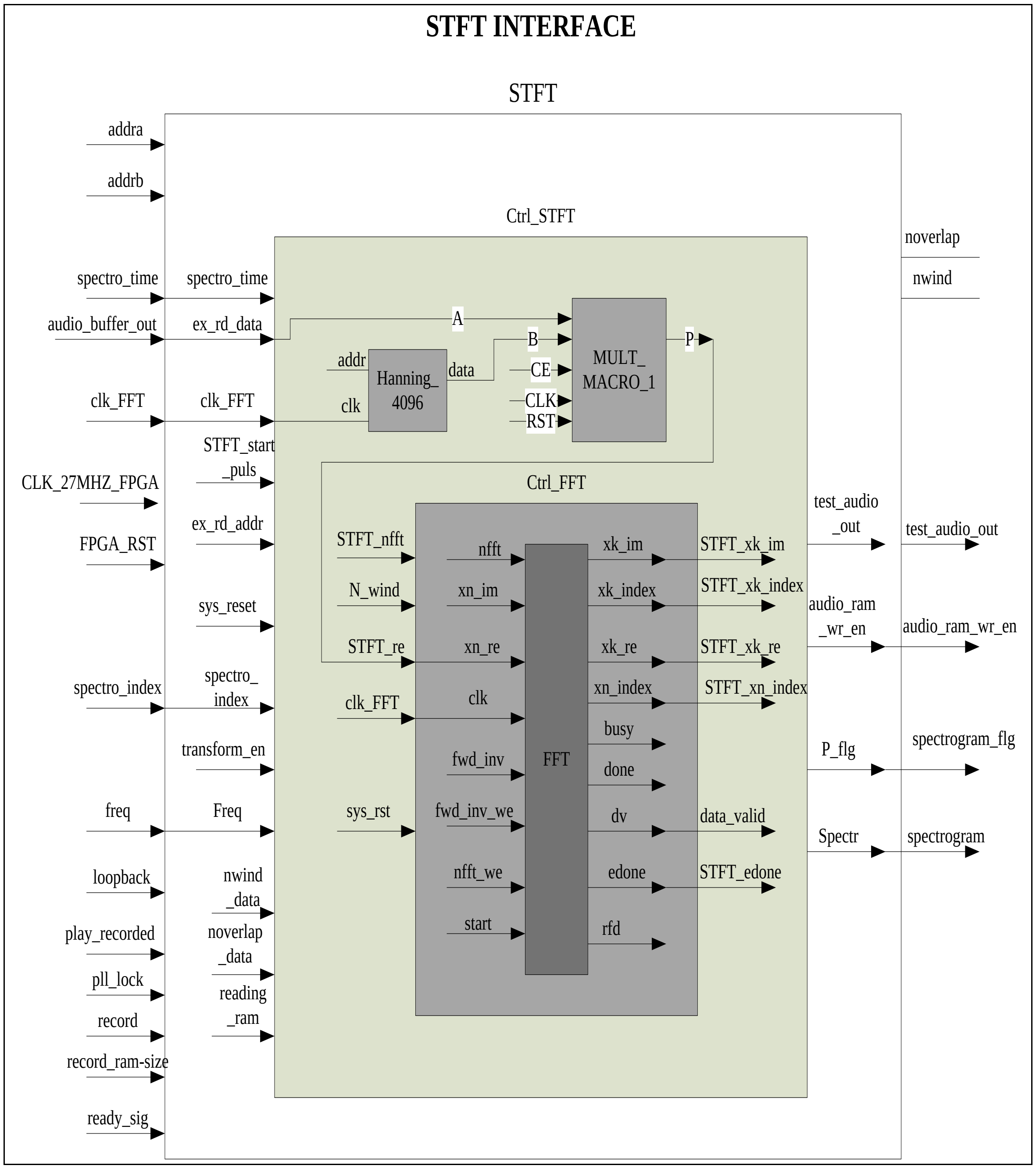


Figure 4.3. STFT interface.

- *STFT Module***:** In loopack or record mode, *addra* increases with respect to the audio data written into the BRAM and in playback mode, *addrb* increases with respect to the audio data read from the BRAM. STFT process requires *nwind* (window size) number of data before it can begin the transform. Hence whenever *addra* or *addrb* is greater than *nwind*, *frame_en*, which is the enable signal for the STFT is set high.

The pin *ram_4_addr* is the address to read audio data from the BRAM (*ram_4recording*) which is necessary to find the *ex_rd_addr* and the *frame_addr_tmp*. The pin *ex_rd_addr* is the address necessary for computing the FFT and it varies from 0 to *nwind-1* and *frame_addr_tmp* a register for holding the temporary read address or write address corresponding to the function (loopback, record or playback) selected.

- *Ctrl_STFT Module:* The user assigned parameters for calculation of and their description are briefly explained below
  - ✓ parameter para_St=48000: It is the Sampling frequency.
  - ✓ parameter [15:0] para_nwind=16'h1000: Represents the window size varying from 8to 4096 expressed as 2^n where n is natural number.
  - ✓ parameter [15:0] para_noverlap =16'hED4: It is the overlapping window size
  - ✓ parameter [15:0] F_factor=12.
  - ✓ parameter [15:0] Pa_factor=16'h40.
  - ✓ parameter [15:0] Pd_factor=16'h80.
  - ✓ parameter [4:0] para_nfft=5'b01100: It depends on the window size and is set in reference with "nfft setting" [32] of FFT core generator as in Table 4.1 [32].

We know that the Xilinx ISE core generated FFT module is designed for receiving complex numbers as input and outputs complex numbers. Whereas in our case, FFT module accepts audio input data with real numbers, but the output is spectrogram data with complex numbers. Therefore the spectrogram amplitude of output data is calculated as sum of the square of real and imaginary part assigned to the variable *Sxx* in our design. The square of real and imaginary part is calculated using the *MULT_MACRO* unimacro.

$$Sxx = re^2 + im^2 \tag{4.12}$$

Also the audio signal is amplified when multiplied with the window data; hence we define two constants namely *Pa_factor* and *Pd_factor* to scale the output. When the frequency is zero, the FFT output is double than when not zero and as a result scaling factor represented by *Pd_factor* must be double that of *Pa_factor*.

Table 4.1. Valid NFFT Settings

NFFT[4:0]	**Transform size (*N*)**
00011	8
00100	16
00101	32
00110	64
00111	128
01000	256
01001	512
01010	1024
01011	2048
01100	4096
01101	8192
01110	16384
01111	32768
10000	65536

Source: Xilinx, Inc., "LogiCORE IP Fast Fourier Transform v7.1: DS260," Last modified March 1, 2011, http://www.xilinx.com/support/documentation/ip_documentation/xfft_ds260.pdf.

$$Pa\_factor = \frac{sum\ of\ hanning\ window\ value}{Window\ size} \tag{4.13}$$

$$Pd_{factor} = Pa\_factor * 2 \tag{4.14}$$

The virtex-5 FPGA Library guide supports use of several Unimacros for arithmetic functions and a macro instantiated in our design is *MULT_MACRO*. One such instantiation in the design is as shown in Code Snippet 2

```
MULT_MACRO #(
    .DEVICE("VIRTEX5"),      // Target Device: "VIRTEX5", "VIRTEX6", "SPARTAN6"
    .LATENCY(3),             // Desired clock cycle latency, 0-4
    .WIDTH_A(8),             // Multiplier A-input bus width, 1-25
    .WIDTH_B(8)              // Multiplier B-input bus width, 1-18)
MULT_MACRO_inst (
    .P(ex_rd_data_tmp),          // Multiplier output bus, width determined by WIDTH_P
parameter
    .A(ex_rd_data),              // Multiplier input A bus, width determined by WIDTH_A
parameter
```

```
    .B(window_data),              // Multiplier input B bus, width determined by WIDTH_B
parameter
    .CE(1'b1),             // 1-bit active high input clock enable
    .CLK(clk_FFT),          // 1-bit positive edge clock input
    .RST(1'b0)             // 1-bit input active high reset );
```

4.4 Code Snippet 2: Virtex-5 Library Unimacro Instantiation

- *Ctrl_FFT Module*: This module is responsible for enabling the signals in synchronous with the Xilinx ISE Core generator generated *FFT* core. Table 4.2 shows the signals used with respect to FFT core. A more detailed explanation of the FFT core implementation is provided in [32].
- *FFT Core*: This module is generated using the Xilinx ISE core generator. The input data of FFT is represented as follows [33]

$$x = -x_{17} + {}_{t=0}^{16} x_t 2^{-17+t} \qquad (4.15)$$

Where, x_t is the t-th bit of x and x is input data.

Table 4.2. FFT Core Signal Description

	From FFT core [32]	Ctrl_FFT.v
1	Clk	clk_FFT
2	nfft_we	nfft_we
3	Nfft	STFT_nfft
4	fwd_inv	fwd_inv
5	fwd_inv_we	fwd_inv_we
6	Start(figure 10)	Start
7	rfd(figure 10)	Rfd
8	busy(figure 10)	Busy
9	Edone	Edone
10	dv(figure 10)	Done
11	xn_re(figure 10)	xn_re
12	xn_index(figure 10)	xn_index
13	xk_index(figure 9)	xk_index
14	xk_re(figure 9)	xk_re
15	xk_im(figure 9)	xk_im

Source: Xilinx, Inc., "LogiCORE IP Fast Fourier Transform v7.1: DS260," Last modified March 1, 2011, http://www.xilinx.com/support/documentation/ip_documentation/xfft_ds260.pdf.

Since we use the FFT of 16-bit system instead of 18-bit system,

$$x = -x_{15} + {}_{t=0}^{14} x_t 2^{-15+t} \qquad (4.16)$$

In other words, the input of FFT must be in (-1, 1) and audio input must be matched with this. Bits 0~15 of output data represent decimal part and bits 16~30 represent integer part. Bit 31 represents the sign bit.

- *Hanning_4096 Module*: The window function used for the spectrogram is the Hann window. The hann window coefficients are calculated in MATLAB and added to the hanning_4096 module. The matlab command used is shown below

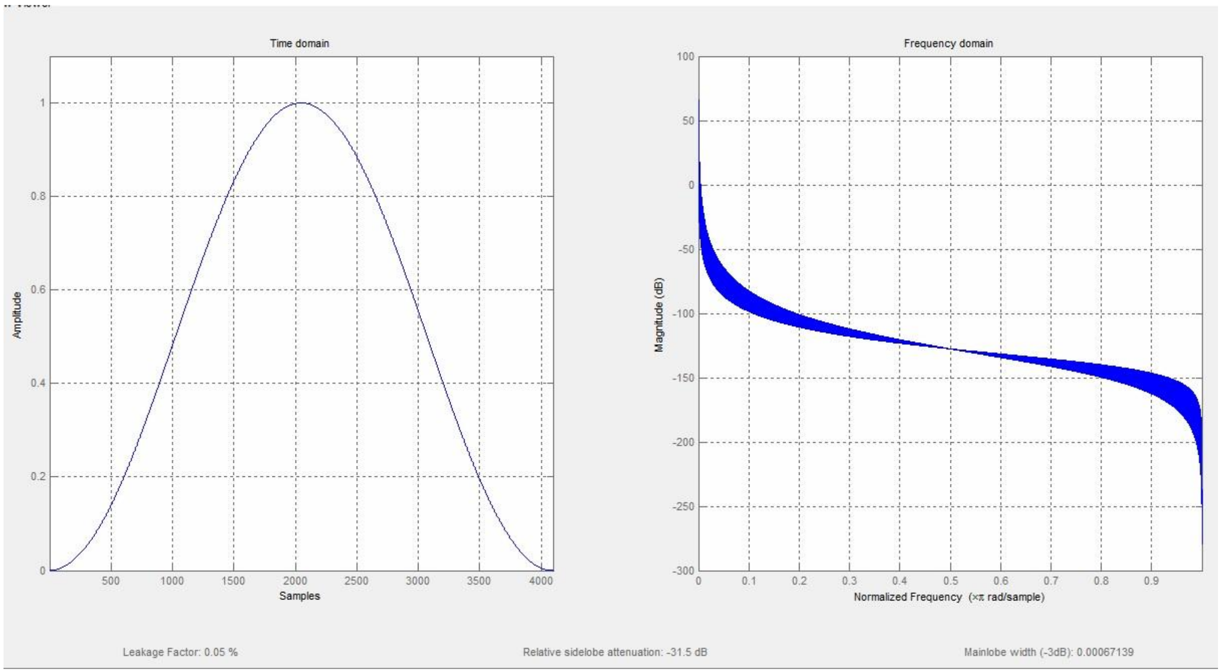
  L= 4096;

  X= hann(L);

  wvtool(X);Where, L is size of window.

  The time domain and frequency domain representation of hann window for a window size L of 4096 is as shown in Figure 4.4. The frequency domain plot by default is the magnitude square of the Fourier transform of the window vector in decibels (dB).

- *Spectrogram_RAM Core*: This is the Xilinx core generated BRAM modules for intermediate storage of spectrogram data before sending to receiver module for synchronization and later to display.



Leakage Factor: 0.05 %   Relative sidelobe attenuation: -31.5 dB   Mainlobe width (-3dB): 0.00067139

Figure 4.4. Time domain and frequency domain representation of Hann window.

4.5 RECEIVER INTERFACE

The Receiver Interface is the module responsible for reading the audio data from Audio_RAM, spectrogram data from Spectrogram_RAM before synchronizing them with the display clock of display interface. It also assigns the coordinates of X and Y axis for displaying time, frequency, amplitude, spectrogram and audio signal. The display X and Y coordinates that are of importance are shown in Figure 4.5.

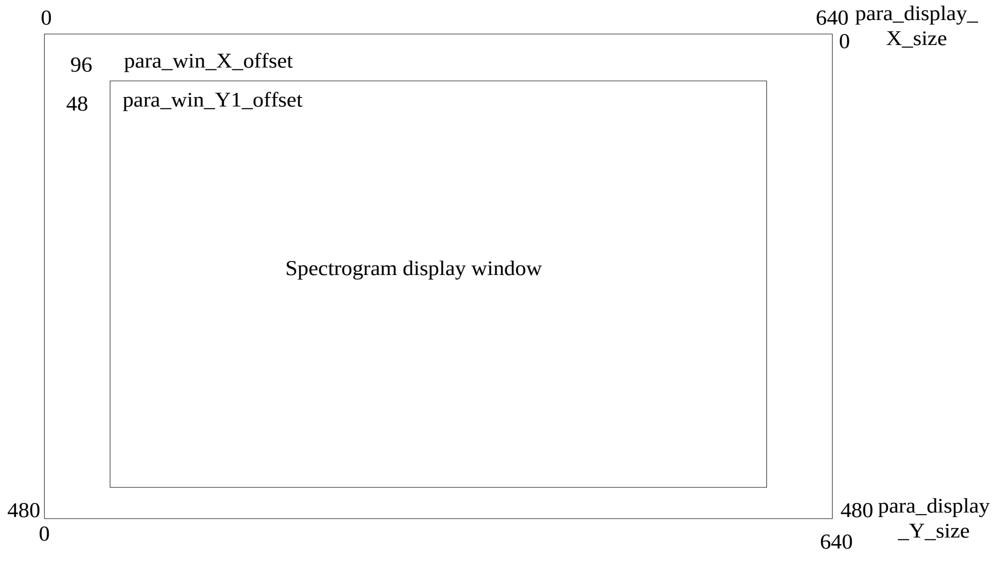


Figure 4.5. Display co-ordinate.

- *Writing Spectrogram Data and Characters to SRAM*: Writing the spectrogram data to SRAM is the penultimate step in displaying the spectrogram. As shown in Code Snippet 3, the temporary registers *ram_X_COORD* is assigned the time axis display and *ram_Y_COORD* the frequency axis display. The data is written to register *image_ram_wr_data*. In a similar way, the characters are displayed by reading the corresponding data from *character_rom* module as shown in Code Snippet 4.

```
if (dispaly_spectrogram_en) begin    //SRAM_spectrogram write
ram_X_COORD_tmp1<={1'b0,disp_time_tmp};
if (spectrogram_fifo_rd_addr<500) begin //nwind/2
spectrogram_fifo_rd_addr<=spectrogram_fifo_rd_addr+1;
if (freq<=para_win_freq_amp)
ram_Y_COORD_tmp1<=(para_win_freq_amp-freq)/disp_freq_Y_step;
else ram_Y_COORD_tmp1<=para_win_Y2_size+1;
ram_Y_COORD_tmp2<=ram_Y_COORD_tmp1+para_win_Y1_size;
image_ram_wr_address<=ram_Y_COORD_tmp2*para_win_X_size+ram_X_COORD_tmp1
;
if (!receiver_clr) image_ram_wr_data<={24'hFFFFFF,spectrogram};
else image_ram_wr_data<={24'hFFFFFF,8'h00};
```

```
end else begin
spectrogram_fifo_rd_addr<=0;
image_ram_wr_address<=(para_win_Y1_size+para_win_Y2_size+1)*para_win_X_size+1;
image_ram_wr_data<=32'hFFFFFF00;
dispaly_spectrogram_en<=1'b0;
end
```

4.6 Code Snippet 3: Write Spectrogram Data to SRAM

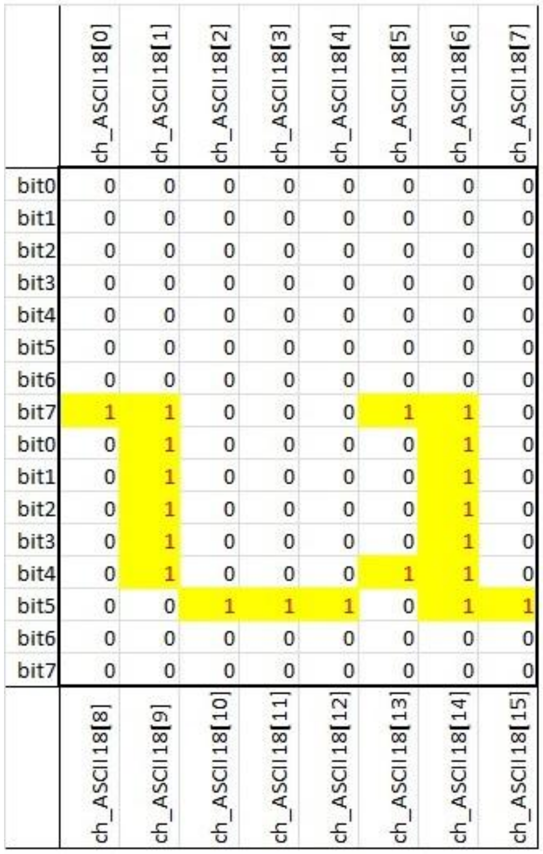
```
//display charactor section
if (dot) image_data_tmp<=255;
 else image_data_tmp<=back_ground;
if ((DVI_Y_COORD>=16) & (DVI_Y_COORD<32)) begin
pos_Y<=16;
if (DVI_X_COORD==56) begin pos_X<=56;addr_char<=7'h48;//H
end else if (DVI_X_COORD==64) begin pos_X<=64;addr_char<=7'h7A;//z
end else if (DVI_X_COORD==72) begin pos_X<=72;addr_char<=7'h20;
end else if (DVI_X_COORD==314) begin pos_X<=314;addr_char<=7'h53;//S
end else if (DVI_X_COORD==322) begin pos_X<=322;addr_char<=7'h70;//p
end else if (DVI_X_COORD==330) begin pos_X<=330;addr_char<=7'h65;//e
end else if (DVI_X_COORD==338) begin pos_X<=338;addr_char<=7'h63;//c

end else if (DVI_X_COORD==346) begin pos_X<=346;addr_char<=7'h74;//t
end else if (DVI_X_COORD==354) begin pos_X<=354;addr_char<=7'h72;//r
end else if (DVI_X_COORD==362) begin pos_X<=362;addr_char<=7'h6F;//o
end else if (DVI_X_COORD==370) begin pos_X<=370;addr_char<=7'h67;//g

end else if (DVI_X_COORD==378) begin pos_X<=378;addr_char<=7'h72;//r
end else if (DVI_X_COORD==386) begin pos_X<=386;addr_char<=7'h61;//a
end else if (DVI_X_COORD==394) begin pos_X<=394;addr_char<=7'h6D;//m
end else if (DVI_X_COORD==402) begin pos_X<=402;addr_char<=7'h20;
end else if (DVI_X_COORD==410) begin pos_X<=410;addr_char
```

4.7 Code Snippet 4: Character Display

- *Color_ROM Module*: This module defines 256 colors for RGB display. The RGB colors are generated in the display interface depending on the 8 bit spectrogram amplitude. The highest amplitude is represented by red color, lowest by blue and it varies as amplitude increases from lowest to highest.
- *Character_ROM Module*: This module defines character patters for displaying the letters, numbers and symbols. Figure 4.6 shows an example of how character pattern is defined.



	ch_ASCII18[0]	ch_ASCII18[1]	ch_ASCII18[2]	ch_ASCII18[3]	ch_ASCII18[4]	ch_ASCII18[5]	ch_ASCII18[6]	ch_ASCII18[7]
bit0	0	0	0	0	0	0	0	0
bit1	0	0	0	0	0	0	0	0
bit2	0	0	0	0	0	0	0	0
bit3	0	0	0	0	0	0	0	0
bit4	0	0	0	0	0	0	0	0
bit5	0	0	0	0	0	0	0	0
bit6	0	0	0	0	0	0	0	0
bit7	1	1	0	0	0	1	1	0
bit0	0	1	0	0	0	0	1	0
bit1	0	1	0	0	0	0	1	0
bit2	0	1	0	0	0	0	1	0
bit3	0	1	0	0	0	0	1	0
bit4	0	1	0	0	0	1	1	0
bit5	0	0	1	1	1	0	1	1
bit6	0	0	0	0	0	0	0	0
bit7	0	0	0	0	0	0	0	0
	ch_ASCII18[8]	ch_ASCII18[9]	ch_ASCII18[10]	ch_ASCII18[11]	ch_ASCII18[12]	ch_ASCII18[13]	ch_ASCII18[14]	ch_ASCII18[15]

Figure 4.6. Character pattern.

- *Image_RAM (SRAM) Module*: This module interacts with the onboard SRAM for storing and retrieving the image data to be displayed on the LCD monitor.

4.8 DVI Interface

The DVI interface receives data from SRAM and sends it to the DVI port for LCD display. Its operation was explained in Chapter 3.

CHAPTER 5

RESULT AND DEMONSTRATION

5.1 DEMONSTRATION

This chapter demonstrates the real time audio spectrogram display through the steps and process. First of all, Xilinx ISE Design Suite is installed and source codes made available on the host laptop. Secondly, the Xilinx Platform USB software installed and ready on the host laptop which is necessary for communication between the laptop and the Xilinx Virtex-5 ML506 FPGA Development board.

5.2 HARDWARE SETUP

1. Connect the Platform USB cable from the host laptop to the FPGA development board and 7.5V DC adaptor to the source.
2. Connect the DVI-D cable from the board to the LCD monitor. Connect the microphone to MIC IN and speaker to HEADPHONE.
3. Turn on the board power by toggling the ON/OFF switch on the development board. Figure 5.1 shows the hardware setup

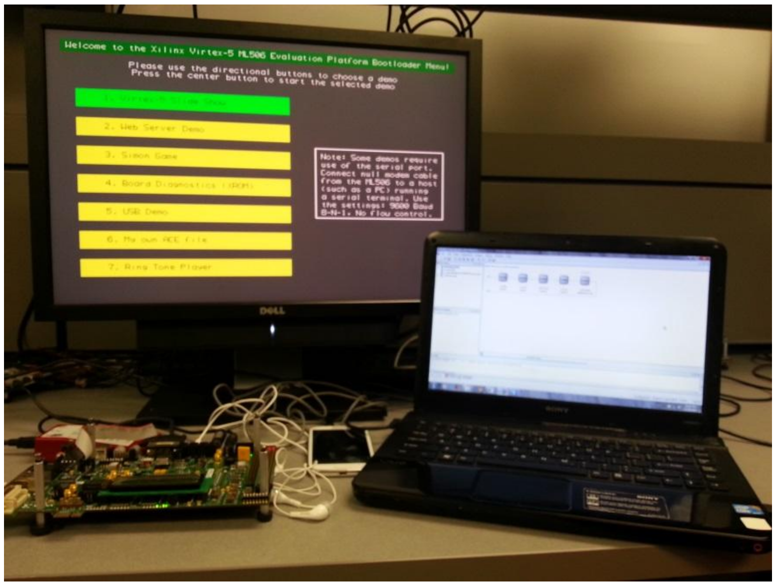

Welcome to the Xilinx Virtex-5 ML506 Evaluation Platform Bootloader Menu!
Please use the directional buttons to choose a demo
Press the center button to start the selected demo
2. Web Server Demo
3. Simon Game
4. Board Diagnostics (ROM)
5. USB Demo
6. My own ACE File
7. Ring Tone Player
Note: Some demos require use of the serial port. Connect null modem cable from the ML506 to a host (such as a PC) running a serial terminal. Use the settings: 9600 Baud 8-N-1, No flow control.

Figure 5.1. Hardware set up.

5.3 Programming the Development Board for Real Time Audio Spectrogram Display

Step 1: Open the Xilinx ISE Project Navigator and open the project by selecting the Audio Spectrogram project file. This will load all the source files linked to the project (Figure 5.2).

Step 2: Synthesize, Implement design and Generate programming file by double clicking on each of them in that order. These options are present in the Processes window as shown in Figure 5.3.

Step 3: The console window shows when each of these processes are complete. After the console window displays "Process 'Generate Programming File' completed successfully", a Design Summary window is displayed. Design Summary window shows the Project Status, Design Utilization Summary, Performance Summary and detailed reports (Figure 5.4).

Step 4: The programming file generated called the "bit file" has to be loaded onto the FPGA to program it. This can be done using the Xilinx ISE iMPACT tool. Turn on the FPGA board, open the Impact tool on the host laptop and double click on "Boundary Scan" (Figure 5.5). On the window that appears, right click and select "Initialize Chain" (Figure 5.6). If it is successful, it shows as in Figure 5.7.

Step 5: Right click on the FPGA (xc5vsx50t), click "Assign New Configuration File" and select the .bit file from the project folder generated previously (Figure 5.8). Right click the FPGA (xc5vsx50t) again and click "Program" (Figure 5.9). If it succeeds, it displays "Program Succeeded" (Figure 5.10).the FPGA (xc5vsx50t) again and click "Program" (Figure 5.9). If it succeeds, it displays "Program Succeeded" (Figure 5.10).

5.4 Results

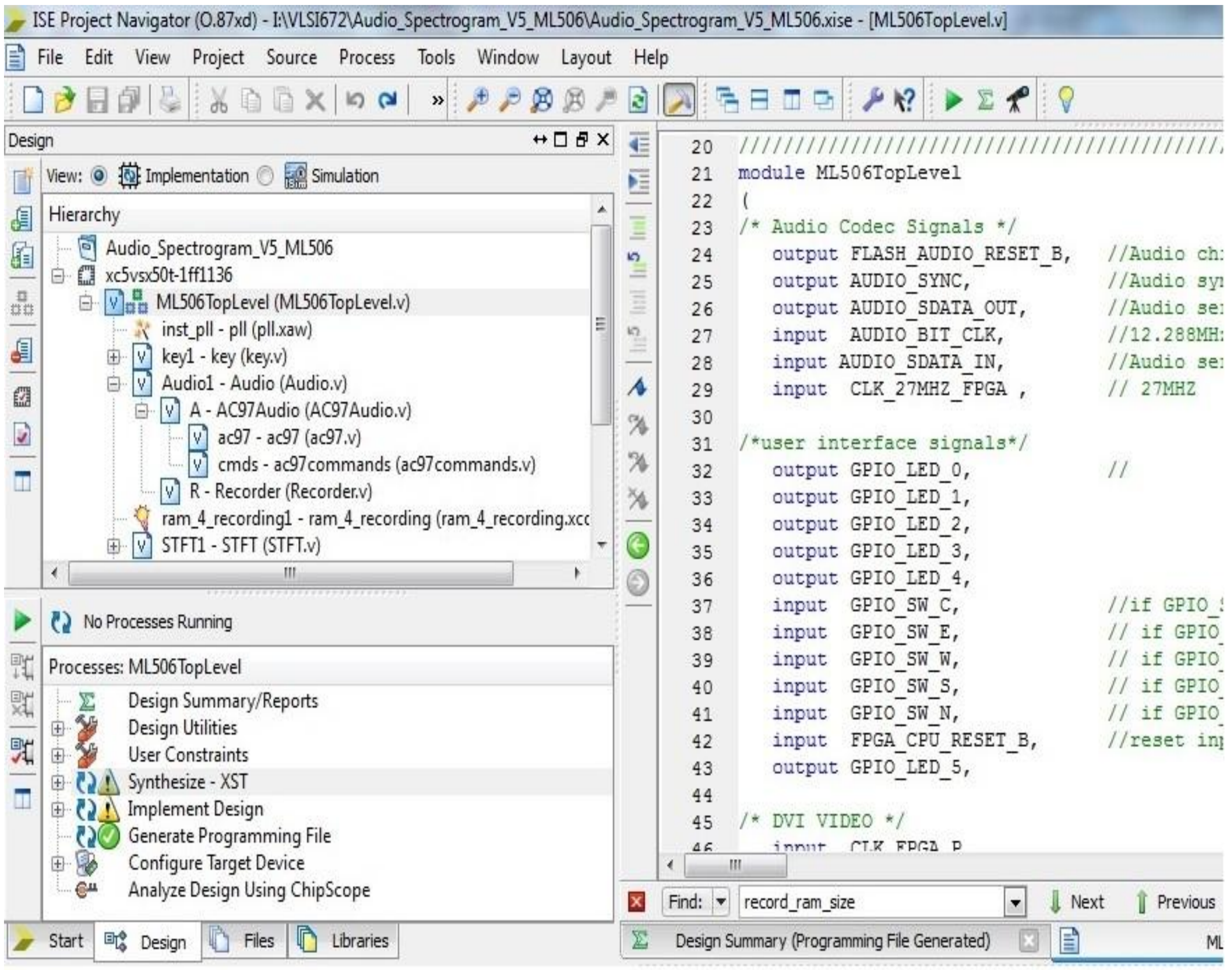
After the program is loaded onto the FPGA development board, the set up and initial LED status is as shown in Figures 5.11 & 5.12.

5.4.1 Real Time Audio Spectrogram

When the **GPIO_SW_C** pushbutton is pressed and held, the audio input through the microphone is loopback through the output speakers and the LCD monitor displays the Real time Audio Spectrogram (Figure 5.13). **GPIO_LED_1 = ON**, BRAM memory Empty; **GPIO_LED_4 = ON**, Audio Loopback and Real time Audio Spectrogram (Figure 5.14).

5.4.2 Audio Recording

When the **GPIO_SW_E** pushbutton is pressed and held, the audio input through the microphone starts recording and is stored on the BRAM. **GPIO_LED_2 = ON**, Audio



ISE Project Navigator (O.87xd) - I:\VLSI672\Audio_Spectrogram_V5_ML506\Audio_Spectrogram_V5_ML506.xise - [ML506TopLevel.v]

File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy

- Audio_Spectrogram_V5_ML506
- xc5vsx50t-1ff1136
  - ML506TopLevel (ML506TopLevel.v)
    - inst_pll - pll (pll.xaw)
    - key1 - key (key.v)
    - Audio1 - Audio (Audio.v)
      - A - AC97Audio (AC97Audio.v)
        - ac97 - ac97 (ac97.v)
        - cmds - ac97commands (ac97commands.v)
      - R - Recorder (Recorder.v)
    - ram_4_recording1 - ram_4_recording (ram_4_recording.xcc
    - STFT1 - STFT (STFT.v)

No Processes Running

Processes: ML506TopLevel

- Design Summary/Reports
- Design Utilities
- User Constraints
- Synthesize - XST
- Implement Design
- Generate Programming File
- Configure Target Device
- Analyze Design Using ChipScope

```
//////////////////////////////////////////
module ML506TopLevel
(
/* Audio Codec Signals */
   output FLASH_AUDIO_RESET_B,   //Audio ch
   output AUDIO_SYNC,            //Audio sy
   output AUDIO_SDATA_OUT,       //Audio se
   input  AUDIO_BIT_CLK,         //12.288MH
   input AUDIO_SDATA_IN,         //Audio se
   input  CLK_27MHZ_FPGA ,       // 27MHZ

/*user interface signals*/
   output GPIO_LED_0,            //
   output GPIO_LED_1,
   output GPIO_LED_2,
   output GPIO_LED_3,
   output GPIO_LED_4,
   input  GPIO_SW_C,             //if GPIO_
   input  GPIO_SW_E,             // if GPIO
   input  GPIO_SW_W,             // if GPIO
   input  GPIO_SW_S,             // if GPIO
   input  GPIO_SW_N,             // if GPIO
   input  FPGA_CPU_RESET_B,      //reset in
   output GPIO_LED_5,

/* DVI VIDEO */
   input  CLK_FPGA_P
```

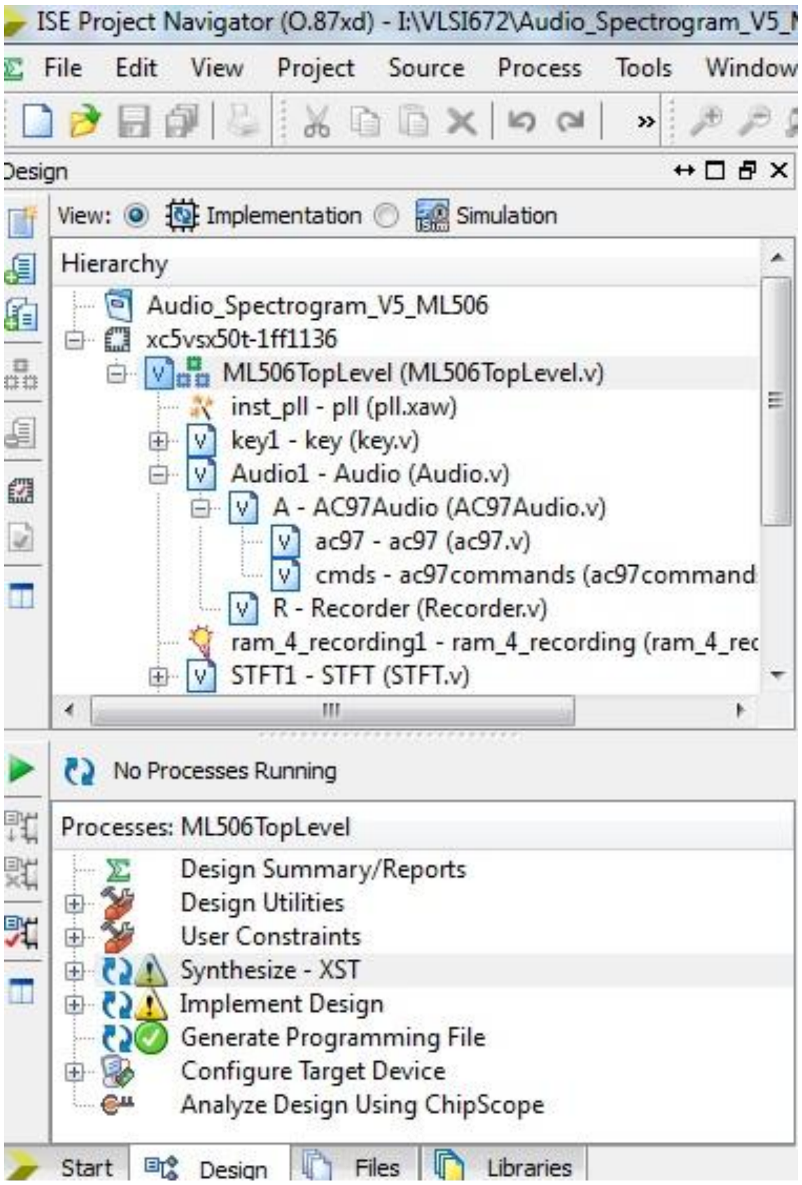
Find: record_ram_size Next Previous
Start Design Files Libraries
Design Summary (Programming File Generated)

Figure 5.2. Xilinx ISE project file.

Recording; **GPIO_LED_0 = ON** (after 8 seconds of recording), BRAM memory Full (Figures 5.15-5.17).

5.4.3 Playback Recorded Spectrogram

When the **GPIO_SW_W** pushbutton is pressed and held, the audio recorded is played back through the speakers and its corresponding Spectrogram is displayed. **GPIO_LED_3 = ON**, Playback (Figures 5.18 & 5.19).



ISE Project Navigator (O.87xd) - I:\VLSI672\Audio_Spectrogram_V5_M

File Edit View Project Source Process Tools Window

Design

View: Implementation Simulation

Hierarchy

```
Audio_Spectrogram_V5_ML506
xc5vsx50t-1ff1136
  ML506TopLevel (ML506TopLevel.v)
    inst_pll - pll (pll.xaw)
    key1 - key (key.v)
    Audio1 - Audio (Audio.v)
      A - AC97Audio (AC97Audio.v)
        ac97 - ac97 (ac97.v)
        cmds - ac97commands (ac97command
      R - Recorder (Recorder.v)
    ram_4_recording1 - ram_4_recording (ram_4_rec
    STFT1 - STFT (STFT.v)
```

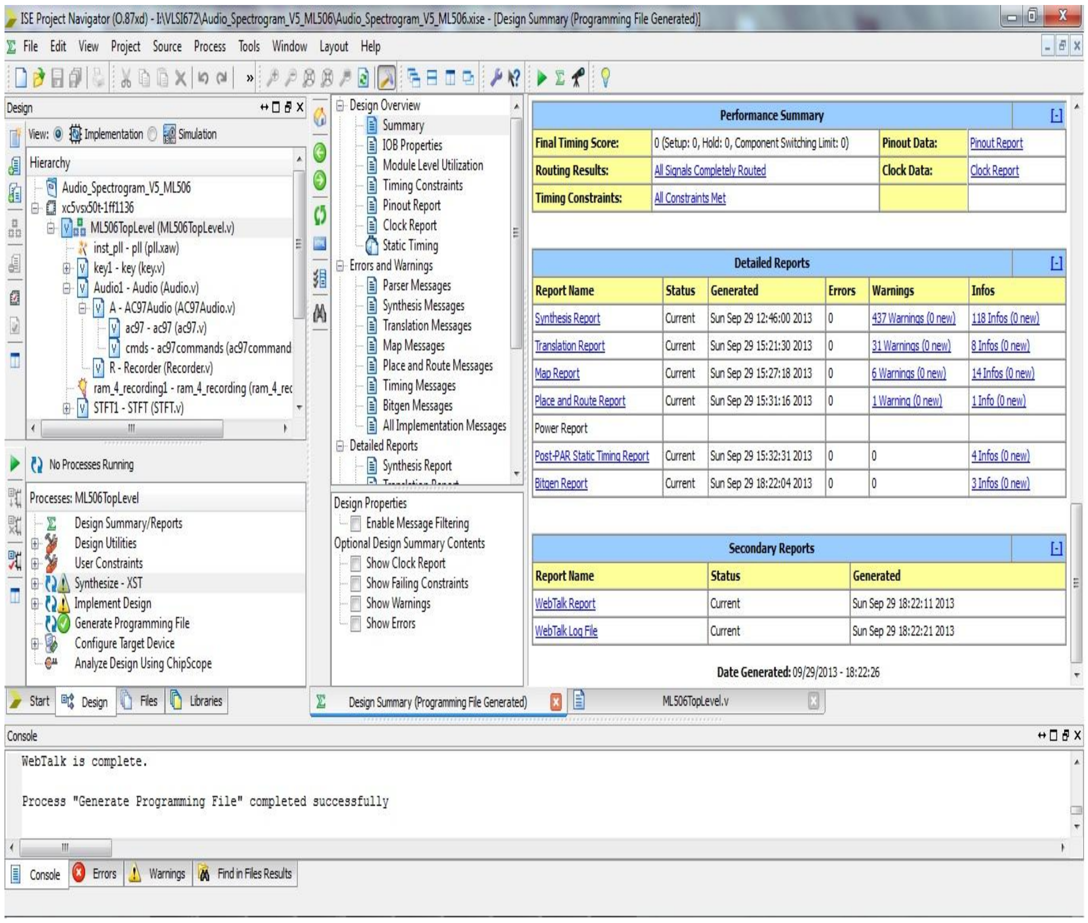
No Processes Running

Processes: ML506TopLevel

```
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Generate Programming File
Configure Target Device
Analyze Design Using ChipScope
```

Start Design Files Libraries

Figure 5.3. Xilinx ISE processes window.



ISE Project Navigator (O.87xd) - I:\VLSI672\Audio_Spectrogram_V5_ML506\Audio_Spectrogram_V5_ML506.xise - [Design Summary (Programming File Generated)]

Performance Summary			
Final Timing Score:	0 (Setup: 0, Hold: 0, Component Switching Limit: 0)	Pinout Data:	Pinout Report
Routing Results:	All Signals Completely Routed	Clock Data:	Clock Report
Timing Constraints:	All Constraints Met		

Report Name	Status	Generated	Errors	Warnings	Infos
Synthesis Report	Current	Sun Sep 29 12:46:00 2013	0	437 Warnings (0 new)	118 Infos (0 new)
Translation Report	Current	Sun Sep 29 15:21:30 2013	0	31 Warnings (0 new)	8 Infos (0 new)
Map Report	Current	Sun Sep 29 15:27:18 2013	0	6 Warnings (0 new)	14 Infos (0 new)
Place and Route Report	Current	Sun Sep 29 15:31:16 2013	0	1 Warning (0 new)	1 Info (0 new)
Power Report					
Post-PAR Static Timing Report	Current	Sun Sep 29 15:32:31 2013	0	0	4 Infos (0 new)
Bitgen Report	Current	Sun Sep 29 18:22:04 2013	0	0	3 Infos (0 new)

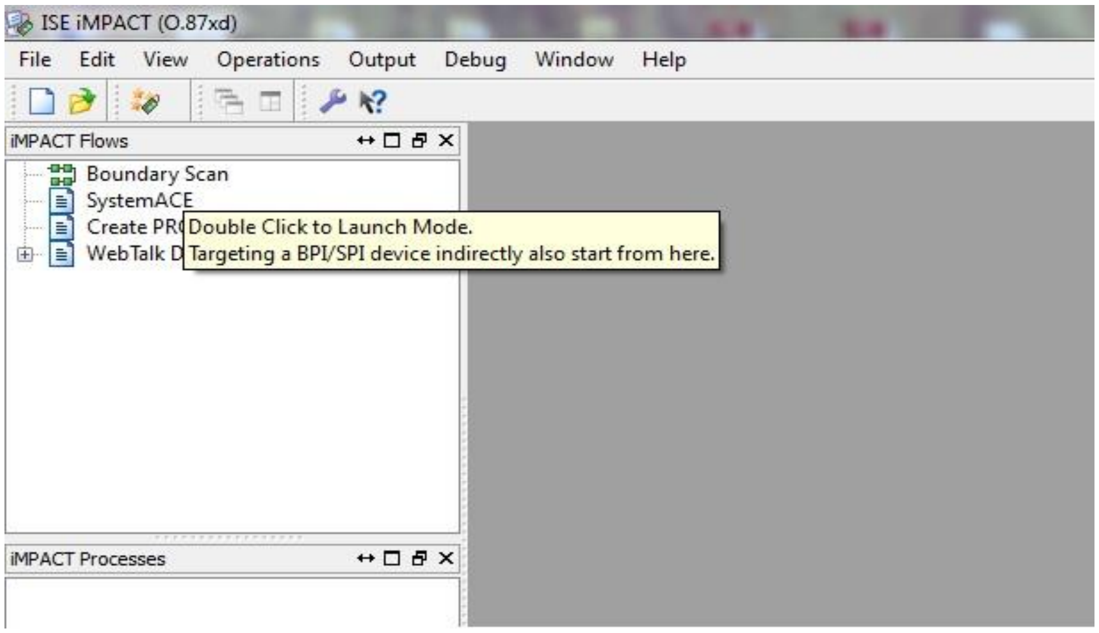
Report Name	Status	Generated
WebTalk Report	Current	Sun Sep 29 18:22:11 2013
WebTalk Log File	Current	Sun Sep 29 18:22:21 2013

Date Generated: 09/29/2013 - 18:22:26

```
WebTalk is complete.

Process "Generate Programming File" completed successfully
```

Figure 5.4. Synthesis, implement and generate programming file processes.



Double Click to Launch Mode.
Targeting a BPI/SPI device indirectly also start from here.

Figure 5.5. Xilinx ISE iMPACT boundary scan.

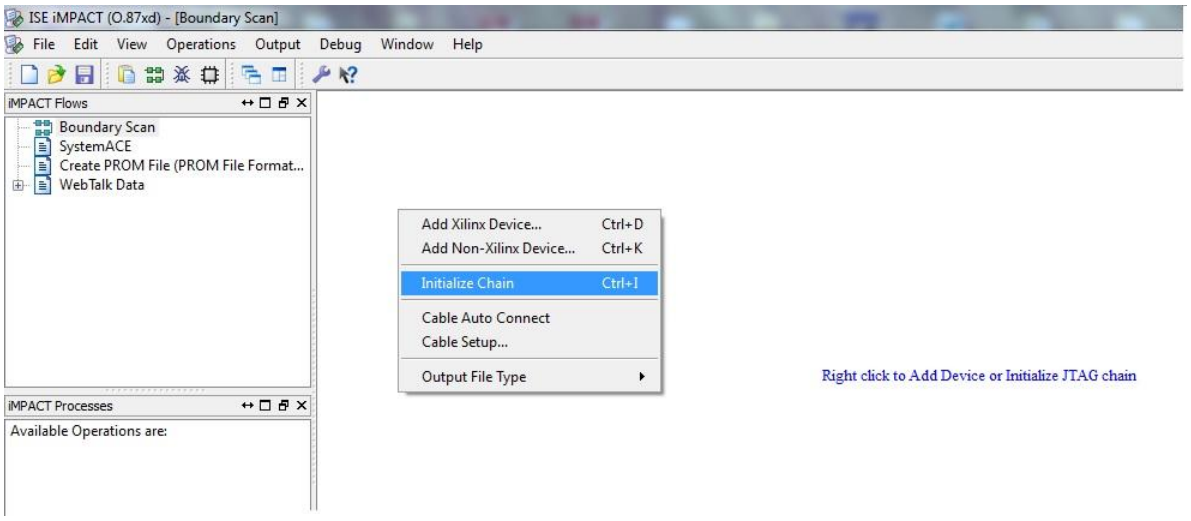
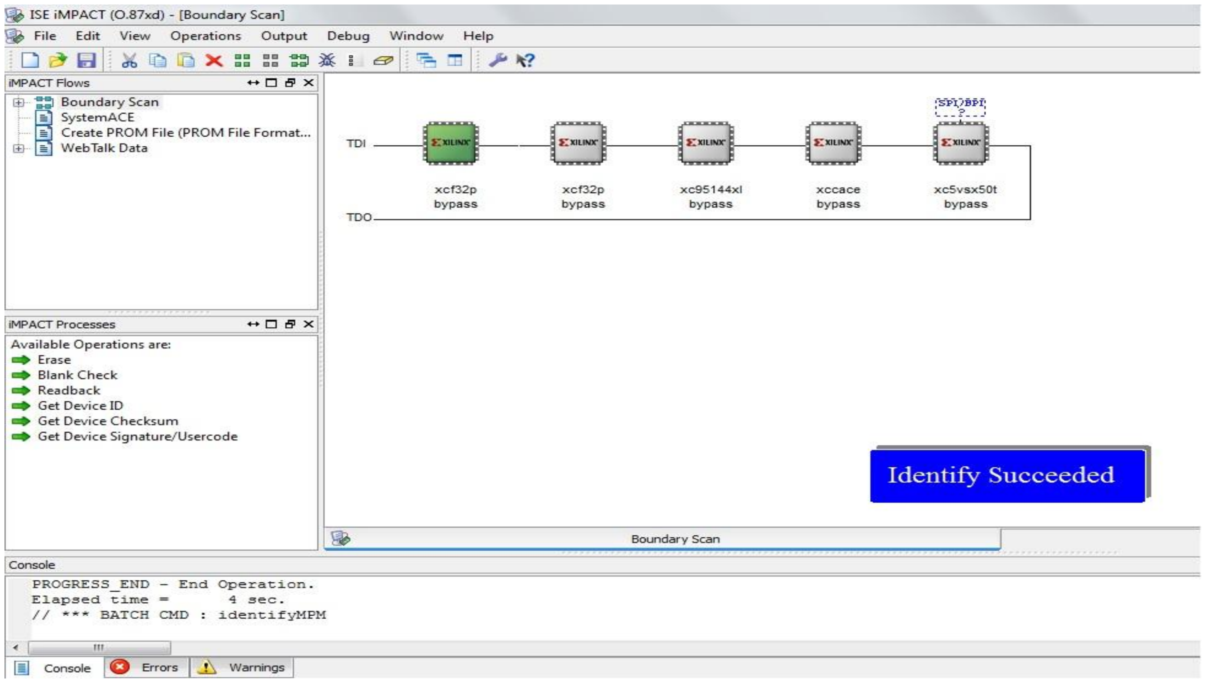
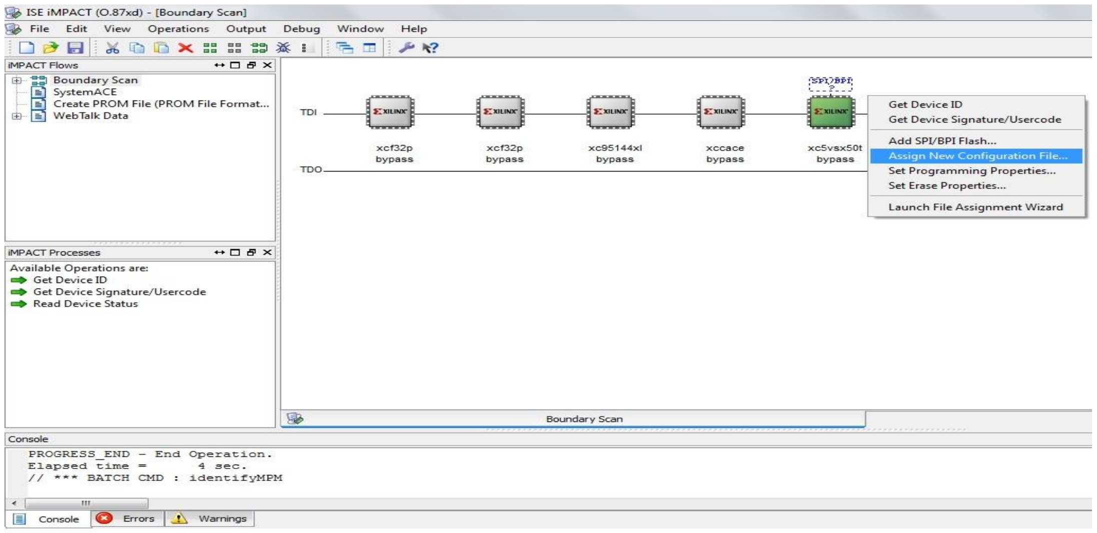

ISE iMPACT (O.87xd) - [Boundary Scan]
File Edit View Operations Output Debug Window Help
iMPACT Flows
Boundary Scan
SystemACE
Create PROM File (PROM File Format...
WebTalk Data
iMPACT Processes
Available Operations are:
Add Xilinx Device... Ctrl+D
Add Non-Xilinx Device... Ctrl+K
Initialize Chain Ctrl+I
Cable Auto Connect
Cable Setup...
Output File Type
Right click to Add Device or Initialize JTAG chain

Figure 5.6. Xilinx ISE iMPACT initialize chain.



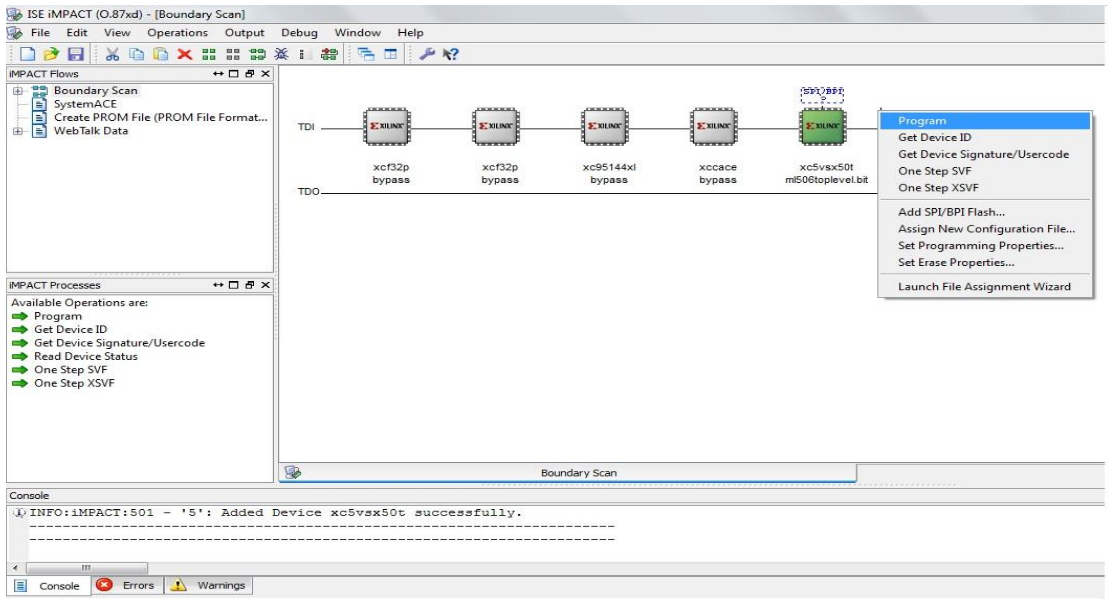
```
PROGRESS_END - End Operation.
Elapsed time =      4 sec.
// *** BATCH CMD : identifyMPM
```

Figure 5.7. Xilinx ISE iMPACT identify succeeded.


ISE iMPACT (O.87xd) - [Boundary Scan]
File Edit View Operations Output Debug Window Help
iMPACT Flows
Boundary Scan
SystemACE
Create PROM File (PROM File Format...
WebTalk Data
iMPACT Processes
Available Operations are:
Get Device ID
Get Device Signature/Usercode
Read Device Status
TDI
TDO
SPI/BPI
xcf32p bypass
xcf32p bypass
xc95144xl bypass
xccace bypass
xc5vsx50t bypass
Get Device ID
Get Device Signature/Usercode
Add SPI/BPI Flash...
Assign New Configuration File...
Set Programming Properties...
Set Erase Properties...
Launch File Assignment Wizard
Boundary Scan
Console
Console
Errors
Warnings

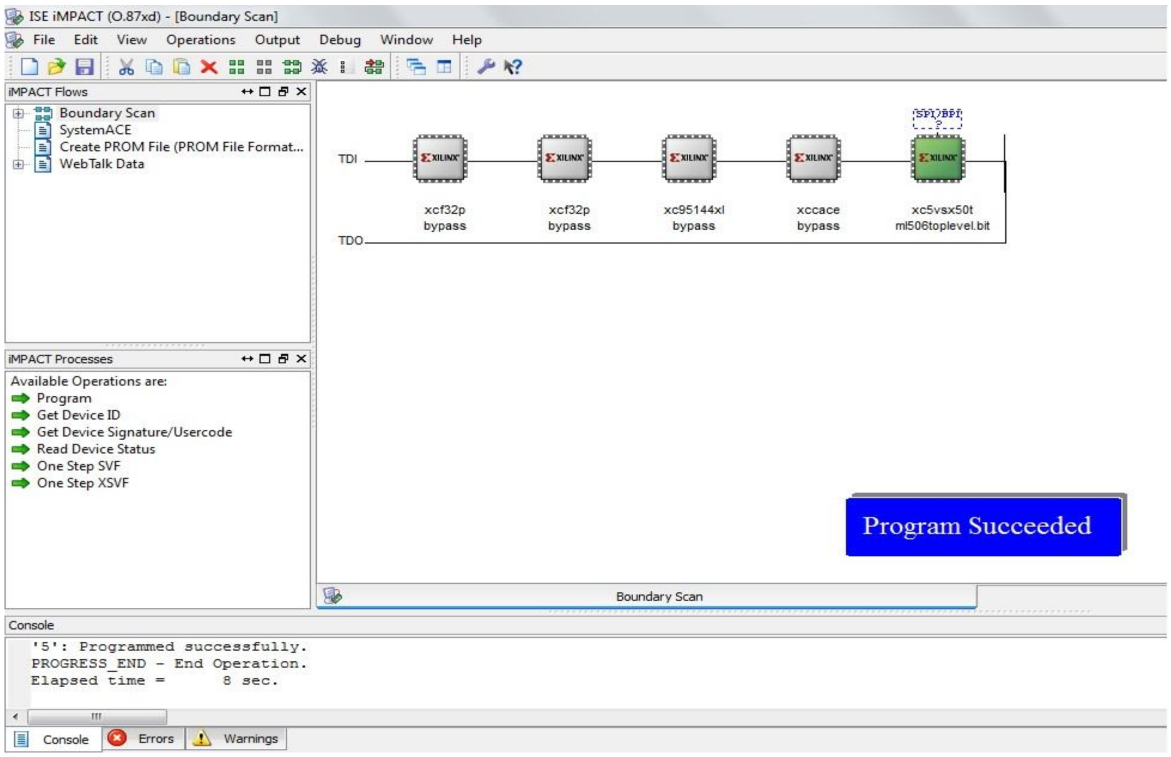
```
PROGRESS_END - End Operation.
Elapsed time =      4 sec.
// *** BATCH CMD : identifyMPM
```

Figure 5.8. Xilinx ISE iMPACT assign new configuration file.


ISE iMPACT (O.87xd) - [Boundary Scan]
File Edit View Operations Output Debug Window Help
iMPACT Flows
Boundary Scan
SystemACE
Create PROM File (PROM File Format...
WebTalk Data
iMPACT Processes
Available Operations are:
Program
Get Device ID
Get Device Signature/Usercode
Read Device Status
One Step SVF
One Step XSVF
TDI
TDO
SPI/BPI
xcf32p bypass
xcf32p bypass
xc95144xl bypass
xccace bypass
xc5vsx50t ml506toplevel.bit
Program
Get Device ID
Get Device Signature/Usercode
One Step SVF
One Step XSVF
Add SPI/BPI Flash...
Assign New Configuration File...
Set Programming Properties...
Set Erase Properties...
Launch File Assignment Wizard
Boundary Scan
Console
Console
Errors
Warnings

```
INFO:iMPACT:501 - '5': Added Device xc5vsx50t successfully.
----------------------------------------------------------------------
----------------------------------------------------------------------
```

Figure 5.9. Xilinx ISE iMPACT program the FPGA.



Available Operations are:
- Program
- Get Device ID
- Get Device Signature/Usercode
- Read Device Status
- One Step SVF
- One Step XSVF

```
'5': Programmed successfully.
PROGRESS_END - End Operation.
Elapsed time =       8 sec.
```

Figure 5.10. Xilinx ISE iMPACT program succeeded.

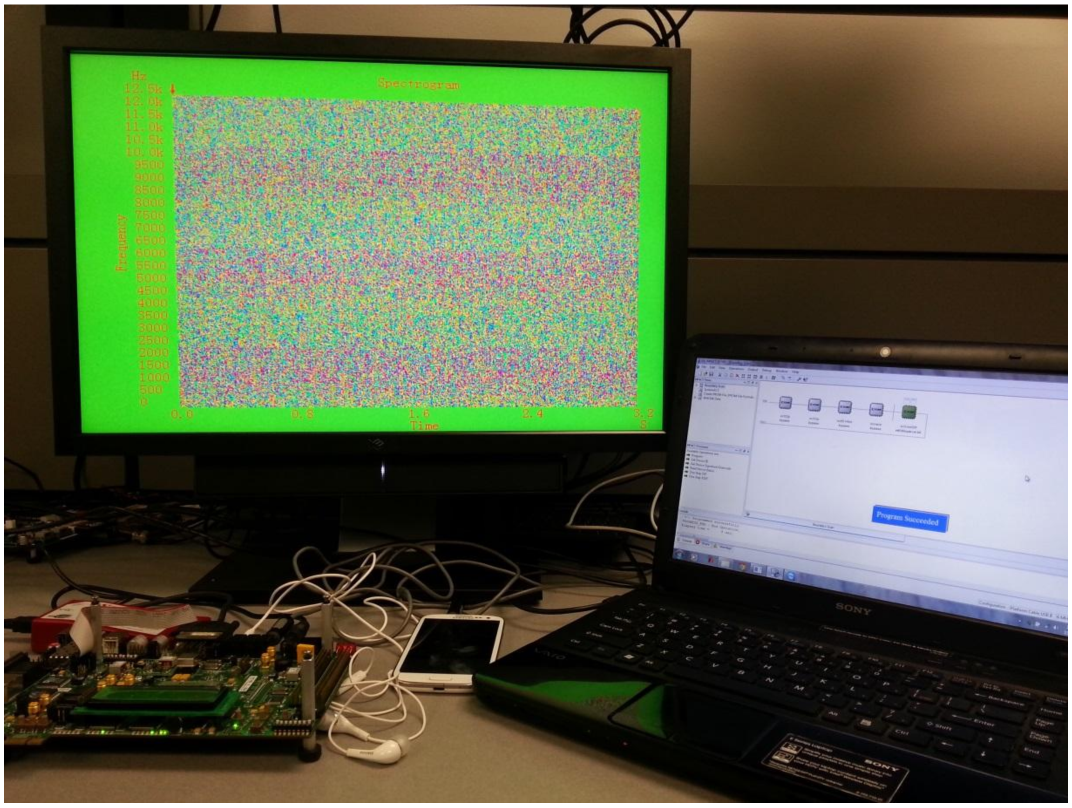


Figure 5.11. Programming the FPGA.

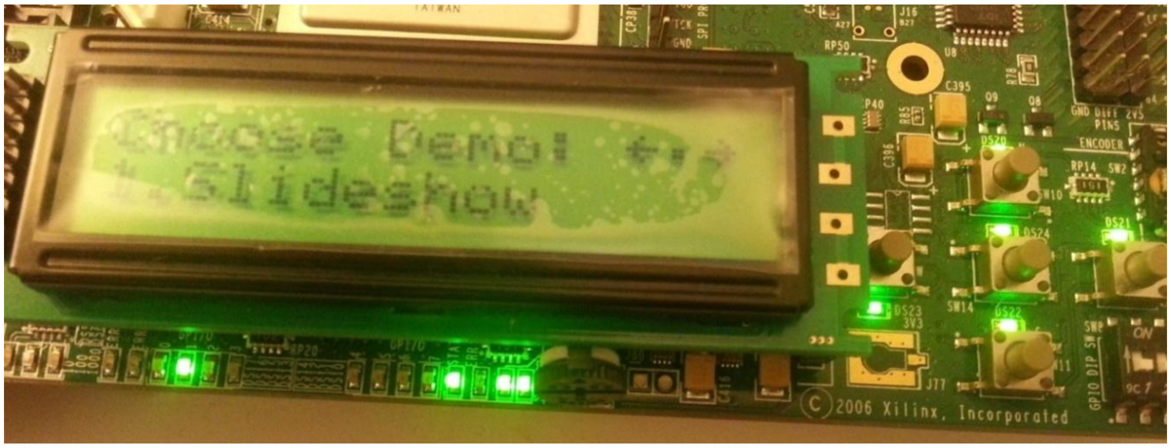


Figure 5.12. Initial LED status.

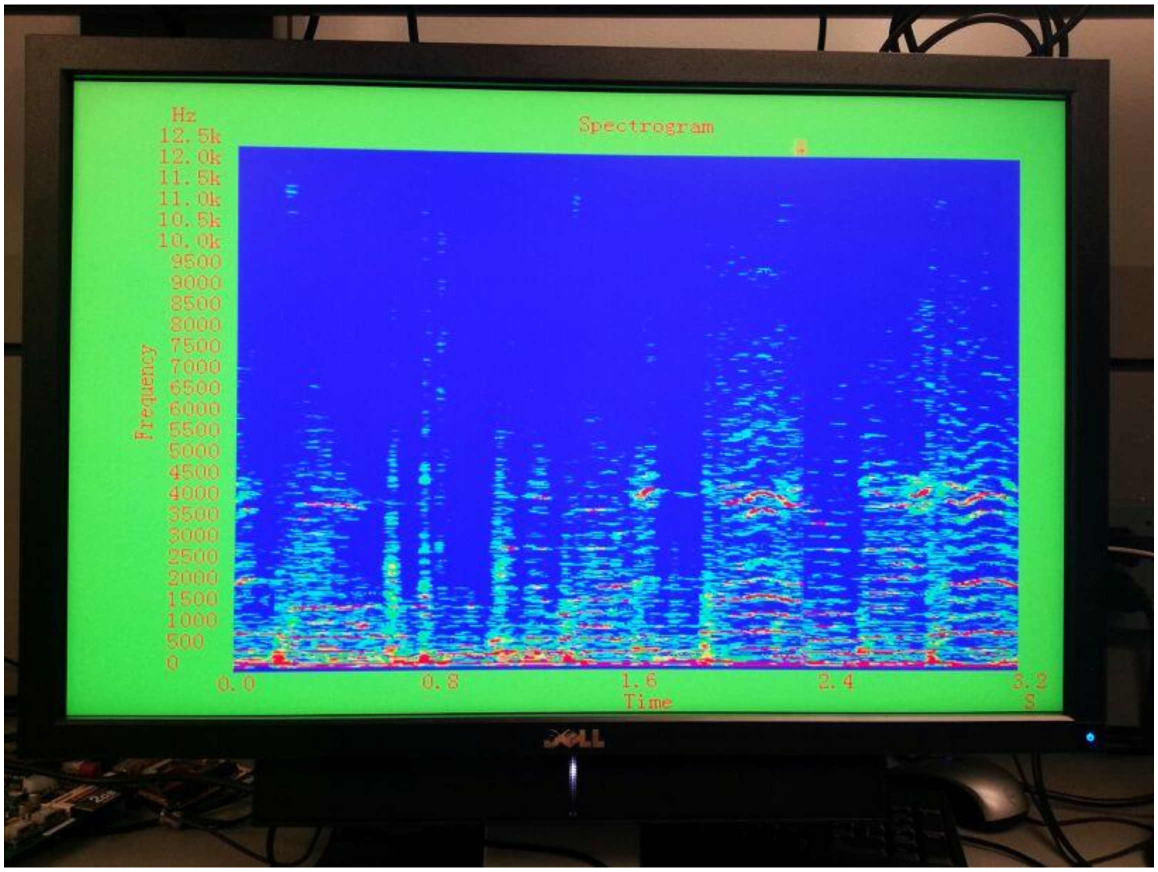


Figure 5.13. LCD Display showing real time audio spectrogram.

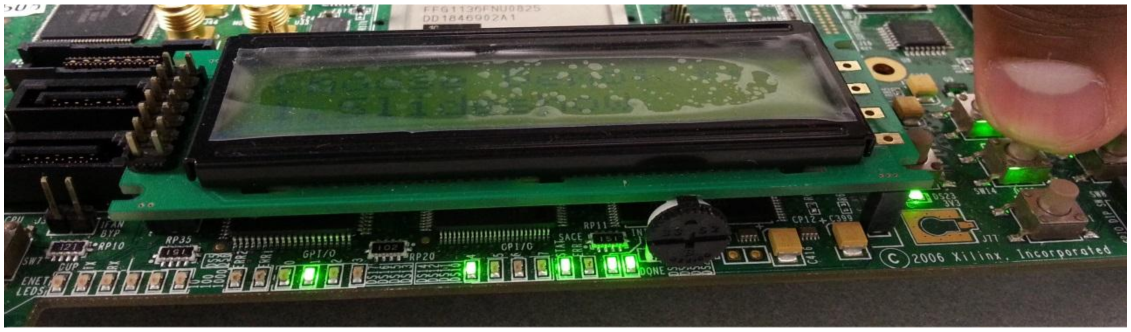


Figure 5.14. Loopback LED status.

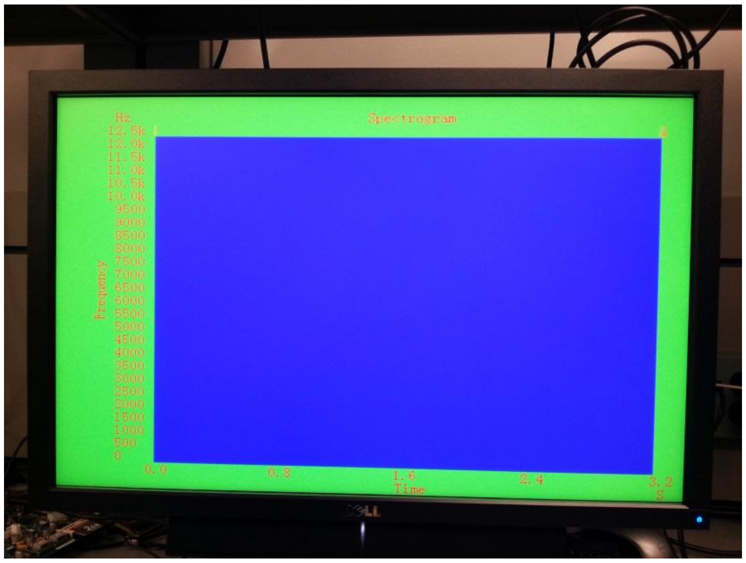


Figure 5.15. LCD display while recording.

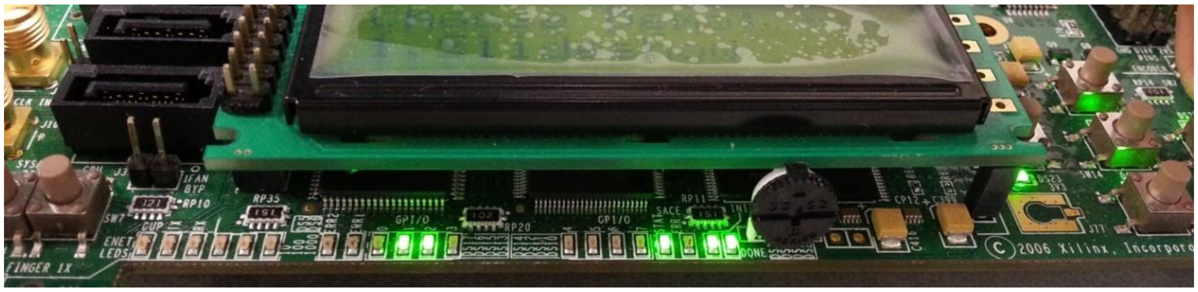


Figure 5.16. Recording start LED status.

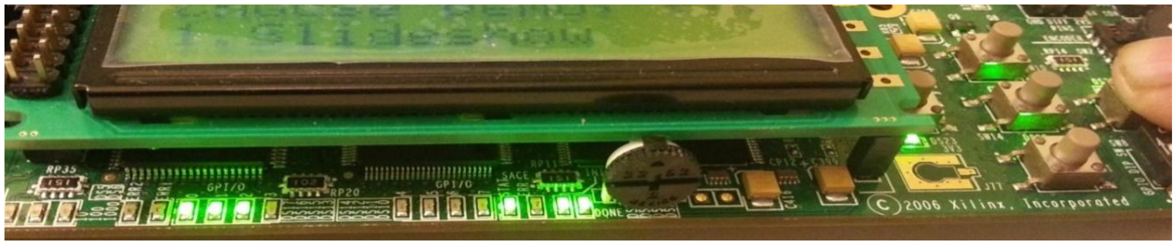


Figure 5.17. Recording end LED status.

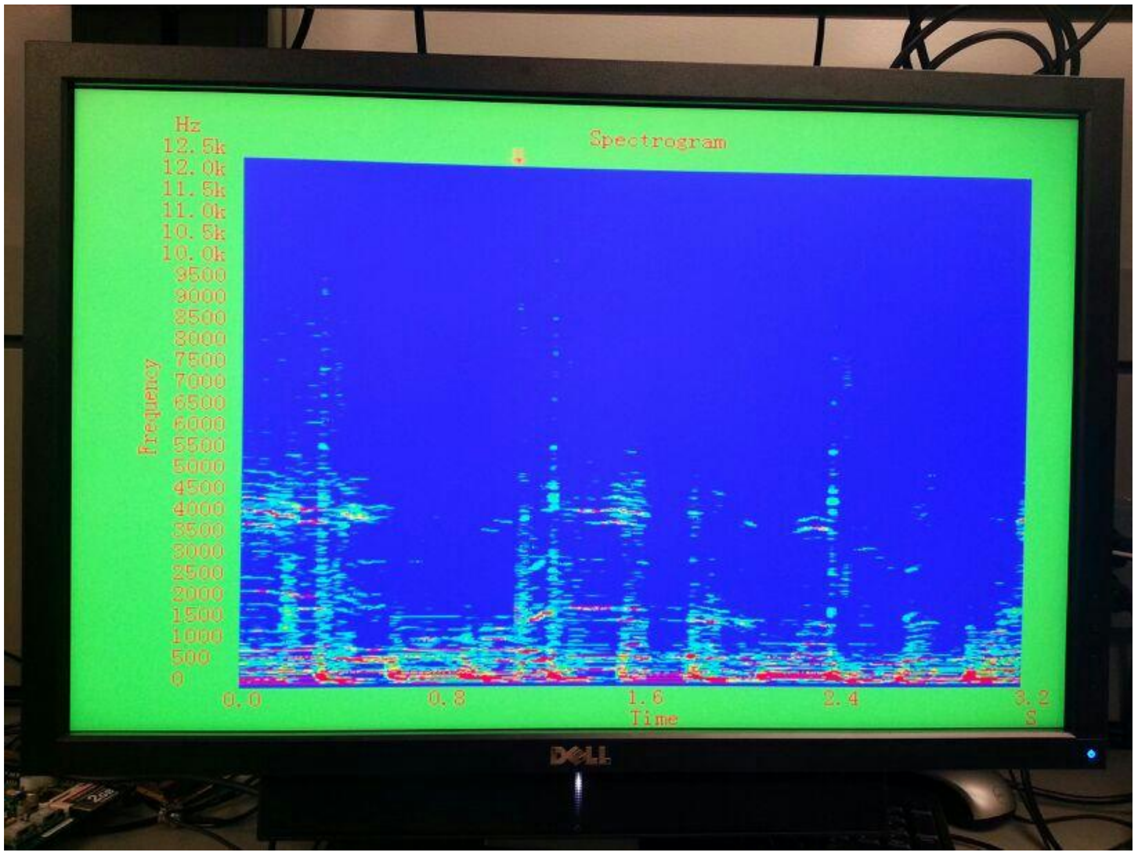


Figure 5.18. LCD display during playback.

Figure 5.19. Playback LED status.

CHAPTER 6

CONCLUSION

A FPGA based Real Time Audio Spectrogram was designed successfully which can display real time audio spectrogram, record audio and playback the audio as well as the spectrogram of the recorded audio. The recording time is 8 seconds. Four modules have been designed separately for the audio recording and playback, short time Fourier transform (STFT), Receiver interface and DVI display interface. However they are combined in the Top level module using the state machine structures at each interface. The interfaces use different clock signal. They are synchronized with the help of first in first out (FIFO) memory and SRAM is used for storing and displaying the spectrogram. Audio spectrograms can be applied to a wide range of areas such as phonetic and speech analysis, assists in overcoming speech defects and also employed is seismic stations for real time analysis. Future work based on the current platform includes designing an audio spectrogram using wavelet transform, compare and analyze the differences between them.

REFERENCES

[1] Boothroyd, A. “Impact of Technology on the Management of Deafness.” *Volta Review* 92, no. 4 (1990): p 74-82.

[2] Elssmann, S. F., & Maki, J. E. “Speech Spectrographic Display: Use of Visual Feedback by Hearing-Impaired Adults During Independent Articulation Practice.” *American Annals of the Deaf* 132, no. 4, (1987): 276-279.

[3] Ertmer, D. J., & Maki, J. E. “A Comparison of Speech Training Methods with Deaf Aolescents: Spectrographic Versus Noninstrumental Instruction.” *Journal of Speech, Language, and Hearing Research 43*, no. 6 (2000): 1509-1523.

[4] Ertmer, D. J., & Stark, R. E. “Eliciting Prespeech Vocalizations in a Young Child with Profound Hearing Loss: Usefulness of Real-Time Spectrographic Speech Displays.” *American Journal of Speech-Language Pathology* 4, no. 1 (1995): 33-38.

[5] Truax, Barry. *Handbook of Acoustic Ecology*. Originally published by the World Soundscape Project, Simon Fraser University, and ARC Publications, 1978/1999.

[6] Smith, Julius O. “Mathematics of the Discrete Fourier Transform (DFT), with Audio Applications-Second Edition.” Last modified April 13, 2007. http://ccrma.stanford.edu/~jos/mdft/.

[7] Andres, Kent. *A Texas Instruments Application Report: MOS Programmable Logic Arrays*. Bulletin: Texas Instruments, 1970.

[8] CXilinx. "CPLD." Xilinx Inc. Accessed November 17, 2013. http://www.xilinx.com/cpld/.

[9] Barr, Michael. Programmable Logic: What's It To Ya*? Embedded Systems Programming*, June 1999.

[10] Clarke, Peter. Xilinx ASIC Vendors Talk Licensing. *EE Times*, June 2001. http://www.eetimes.com/document.asp?doc_id=1180867

[11] Funding Universe. “Xilinx, Inc. History.” Accessed January 15, 2009. http://www.fundinguniverse.com/company-histories/xilinx-inc-history/.

[12] Maxfield, Clive. Xilinx Unveil Revolutionary 65nm FPGA Architecture: The Virtex-5 Family. *EE Times*, May 2006. http://www.eetimes.com/document.asp?doc_id=1300189.

[13] Nat Technology Inc. “Integrated circuit computing device comprising a dynamically configurable gate array having a microprocessor and reconfigurable instruction execution means and method therefor: US Patent: US 5361373 A.” Last modified 2012. https://www.google.com/patents/WO1994014123A1?cl=en&dq=Integrated+circuit+computing+device+comprising+a+dynamically+configurable+gate+array+having+a+microprocessor+and+reconfigurable+instruction+execution+means+and+method+therefor.

&hl=en&sa=X&ei=4GzMUr-tB4XpoASxuoLwDQ&ved=0CDcQ6AEwAA#forward-citations.

[14] Altium and Jason Howie. “FPGA Signal Integrity Tutorial-Simulating the Reflection Characteristics” Last modified December 1, 2008. http://wiki.altium.com/display/ADOH/FPGA+SI+Tutorial+-+Simulating+the+Reflection+Characteristics.

[15] Thompson, Mike. Mixed-Signal FPGAs Provide Green Power. *EE Times*, July 2007. http://www.eetimes.com/document.asp?doc_id=1271543.

[16] Wong, William. “FPGA Kits Support FMC Mezzanine Card Development.” Electronic Design. Last modified February. 2, 2012. http://electronicdesign.com/author/william-wong.

[17] Altera. “Stratix V GX FPGA Development Kit: Figure 1.” Last modified 2011. http://www.altera.com/products/devkits/altera/kit-sv-gx-host.html#RelatedLinks.

[18] Xilinx. “ML505/ML506/ML507 Evaluation Platform: User Guide: UG347.” Revision v3.1.2. Last modified May 16, 2011. http://www.xilinx.com/support/documentation/boards_and_kits/ug347.pdf.

[19] Analog Devices. “AD1981B: AC ’97 SoundMAX Codec.” Last modified 2005. http://www.analog.com/static/imported-files/data_sheets/AD1981B.pdf.

[20] Xiaoping, Bai and Zhang Hongwei. "Study of Digital Video Interface (DVI) Hardware Design in Multimedia System." Paper presented at IEEE Workshop Microelectronics and Electron Devices (WMED), April 20, 2007.

[21] Li, Shuang and Ruiguang Wang. "DVI interface, Optical Fiber Video Transceiver Used in LED Display System." Paper presented at IEEE 3rd International Conference Communication Software and Networks (ICCSN), Xi‘an, China, May 27-29, 2011.

[22] Luo, Yan-Bin, Ping Chen, Qui-Ting Chen, Chih-Yong Wang, Chan-Hao Chang, Szu-Jui Fu, Chien-Ming Chen *et al.* "A 250Mb/s-to-3.4Gb/s HDMI Receiver with Adaptive Loop Updating Frequencies and an Adaptive Equalizer." Paper presented at IEEE International Solid-State Circuits Conference - Digest of Technical Papers, San Francisco, CA, February 8-12, 2009.

[23] Thompson, S. "VGA-Sign Choices for A New Video Subsystem." *IBM Systems Journal* 27, no. 2 (1988): 185,197.

[24] Infante, C. "Advances in CRT Displays." Paper presented at Conference Record of the International Display Research Conference, San Diego, CA, October 4-6, 1988.

[25] Silicon Image. “Digital Visual Interface & TMDS Extensions.” Last modified October 2004. http://www.siliconimage.com/docs/SiI-WP-007-A.pdf.

[26] Digital Display Working Group (DDWG). “Digital Visual Interface.” Revision 1.0. Last modified April 2, 1999. http://www.cs.unc.edu/Research/stc/FAQs/Video/dvi_spec-V1_0.pdf.

[27] Wikipedia. “Digital Visual Interface.” Last modified December 26, 2013. http://en.wikipedia.org/wiki/Digital_Visual_Interface.

[28] Analog Devices. "AD9980: High Performance 8-bit Display Interface." Last modified 2005. http://www.analog.com/static/imported-files/data_sheets/AD9980.pdf.

[29] Chrontel."CH7301C DVI Transmitter Device." Revison 1.5. Accessed March 17, 2010.http://www.chrontel.com/media/Datasheets/7301ds.pdf.

[30] Chrontel. "AN41 (Application Notes): CH7009A/B, CH7010 A/B, CH7011A, CH7012A, CH7301 A/B Encoder registers Read/Write operation." Revision 1.3. Last modified May 15, 2003. http://www.chrontel.com/media/Application%20Notes/an41.pdf.

[31] Ljubisa, Stankovic, Dakovic, Milos, and Thayaparan Thayananthan. *Time-Frequency Signal Analysis with Applications*. Norwood: Artech House, 2013.

[32] Xilinx, Inc. "*LogiCORE IP Fast Fourier Transform v7.1: DS260*." Last modified March 1, 2011. http://www.xilinx.com/support/documentation/ip_documentation/xfft_ds260.pdf.

[33] Xilinx, Inc. "IP LogiCORE Discrete Fourier Transform v3.1: DS615." Last modified March 1, 2011. http://www.xilinx.com/support/documentation/ip_documentation/dft_ds615.pdf.