UNIVERSITÀ DI PISA

Digital System Design

Real-Time Audio Signal Processing on FPGA

Implementation of a Dual-Mode Lock-in Amplifier and FFT Spectrum Analyzer on Altera DE2

October-December 2025

Professor:
Prof. Roberto Salvetti
Class:
Digital Systems Design

Students:
Pietro Faraggiana
Leonardo Ricciardi
Marcel Kenfack

Contents

Chapter 1

Introduction

This project presents the design and implementation of a real-time audio signal processing system on the Altera DE2 board. The architecture integrates two distinct laboratory instruments into a single FPGA core: a Fast Fourier Transform Spectrum Analyzer and a Dual-Mode Lock-in Amplifier.

The complete source code and technical references are available in the GitHub Repository.

1.1 Project Overview and Objectives

The primary motivation for this work is to explore the capabilities of the Altera Cyclone II FPGA for high-speed digital signal processing (DSP). Unlike software-based solutions running on general-purpose CPUs, FPGAs offer deterministic latency and massive parallelism, making them ideal for spectral analysis and weak signal detection.

The main objective is to create a standalone system capable of:

- **Audio Acquisition:** Sampling analog signals at 48 kHz (24-bit) via the on-board CODEC.
- **Processing:** Computing a 512-point FFT for spectrum analysis and implementing a digital Lock-in Amplifier for phase-sensitive detection.
- **Visualization:** Driving a VGA monitor to display the frequency spectrum or signal magnitude/phase in real-time without a host PC.

1.2 The Altera DE2 Board

The development platform selected for this project is the Altera DE2 Development and Education Board. As detailed in the DE2 User Manual [1], this board provides a robust environment for advanced digital design.

The core of the system is the Altera Cyclone II 2C35 FPGA, which features 33,216 Logic Elements and 35 embedded multipliers. These resources are critical for performing the complex arithmetic operations required by the FFT and mixing stages.

To achieve the project goals, the following on-board peripherals are utilized:

- **Wolfson WM8731 Audio CODEC:** Used for high-quality A/D and D/A conversion, communicating with the FPGA via I2C and I2S protocols.
- **Analog Devices ADV7123 VGA DAC:** Converts digital pixel data into analog RGB signals for the monitor (640×480 resolution).
- **Switches and Push-buttons:** Employed for runtime mode selection (switching between FFT and Lock-in) and frequency tuning.

Chapter 2

Theoretical Background

2.1 Digital Audio Interface Protocols

Digital audio processing systems require the precise coordination of control signals and data streams. The interface between the Altera Cyclone II FPGA and the Wolfson WM8731 audio codec relies on two distinct protocols, each serving a specific purpose in the system architecture:

- **Control Interface (I2C):** Used for the initial configuration of the codec parameters, such as sampling rate, data format, and active channels. This is a low-speed, intermittent communication where the FPGA acts as the Master.
- **Streaming Interface (I2S):** Used for the continuous, real-time transfer of audio samples between the ADC/DAC and the FPGA. This is a high-speed, synchronous communication where the Codec acts as the Master.

2.1.1 The I2C Protocol (Codec Configuration)

The Inter-Integrated Circuit (I2C) is a two-wire serial protocol. In this implementation, the FPGA acts as the I2C Master, while the WM8731 codec acts as the Slave. The physical layer consists of two open-drain lines requiring external pull-up resistors:

- **SCL (Serial Clock):** Generated by the Master to synchronize data transfer. In our design, this is derived from the 50 MHz system clock down to approximately 100 kHz.
- **SDA (Serial Data):** A bidirectional line for data transfer.

Protocol Transaction Structure

A standard I2C write transaction, managed by the Finite State Machine (FSM) in the `i2c_config_codec_standard` module, follows a specific sequence:

1. **START Condition:** The Master pulls SDA low while SCL remains high.
2. **Address Frame:** The Master transmits the 7-bit slave address of the WM8731 (`0011010`) followed by a Write bit (0).
3. **Acknowledge (ACK):** The Slave pulls SDA low during the 9th clock pulse to confirm the reception.
4. **Data Frames:** The Master sends the register address (7 bits) concatenated with the configuration data (9 bits). In the physical transmission, this is split into two 8-bit bytes, each followed by an ACK from the slave.
5. **STOP Condition:** The Master releases SDA from low to high while SCL is high, terminating the transaction.

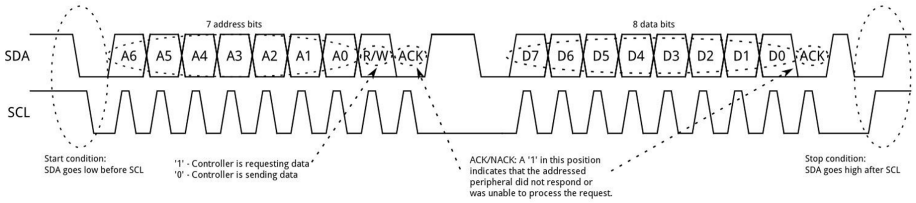

Figure 2.1: I2C Write Sequence: Start, Address, Data Bytes, and Stop condition[2].

WM8731 Register Configuration

The configuration module implements a "one-shot" initialization sequence. Upon reset, the FSM sequentially writes 10 configuration words to the codec. Table 2.1 summarizes the register settings used to establish a 48 kHz sampling rate and 24-bit I2S data format.

Sequence	Register	Value (Hex)	Description
0	Reset	1E00	Reset device to default state
1	Power Down	0C00	Power On ADC, DAC, and Output
2	Analog Path	0815	Enable Mic Boost, Select DAC/Mic
3	Digital Path	0A00	Disable soft mute and de-emphasis
4	Digital I/F	0E4A	Master Mode, 24-bit, I2S Format
5	Sampling	1002	Normal Mode, 48 kHz (MCLK 18.432 MHz)
6	Active	1201	Activate Digital Interface
7–9	Volume	Various	Set Line-In and Headphone gains

Table 2.1: WM8731 Configuration Sequence[1].

2.1.2 The I2S Protocol (Audio Streaming)

The Inter-IC Sound (I2S) protocol is designed for high-fidelity digital audio transport. Unlike the control interface, I2S runs continuously to stream audio samples.

Master/Slave Hierarchy and Signals

A critical design choice is the clock generation role. The Codec is configured in Master Mode (via Register 4), meaning it generates the synchronization clocks for the FPGA. The interface comprises three signals:

- **BCLK (Bit Clock):** The serial clock for shifting data bits. For 24-bit stereo audio at 48 kHz, the frequency is approximately 3.072 MHz.
- **ADCLRCK (Left/Right Clock):** Also known as Word Select. A low level indicates the Left channel, while a high level indicates the Right channel.
- **ADCDAT (ADC Data):** The serial data stream containing the audio samples (MSB first).

In the I2S standard, the data is valid on the rising edge of BCLK, and the Most Significant Bit (MSB) is transmitted one clock cycle after the transition of ADCLRCK. The `i2s_controller` module captures this serial stream and converts it into a 24-bit parallel word.

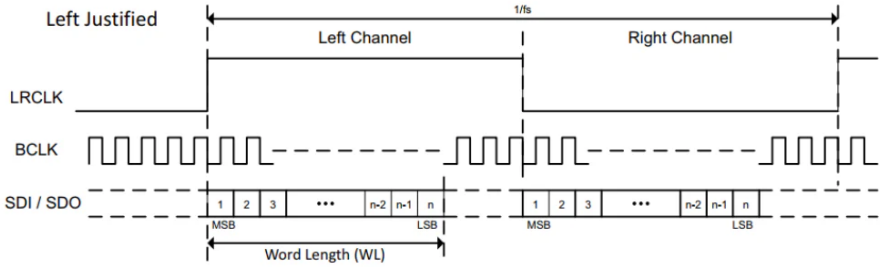

Figure 2.2: I2S Protocol Timing: Relationship between BCLK, LRCLK, and Data [3].

Clock Domain Crossing (CDC)

The audio subsystem introduces a challenge regarding clock domains. The I2S signals (BCLK, ADCLRCK) are generated by the codec's internal oscillator or PLL and are asynchronous with respect to the FPGA's system clock (`CLOCK_50`).

Directly sampling asynchronous signals can lead to metastability, where a register enters an undefined state. To mitigate this, the `top_audio` module implements a synchronization strategy:

1. **Data Capture:** The audio data is first captured in the `i2s_controller` using the codec's BCLK.
2. **Flag Synchronization:** A data-valid flag is passed through an optimized synchronizer to safely cross domains.
3. **Double Buffering:** Once the valid flag is synchronized, the data is written into a Double Buffer. This decouples the continuous streaming rate of the audio codec from the block-based processing rate of the FFT and Lock-in modules.

2.2 The FFT Algorithm

The Discrete Fourier Transform (DFT) is a fundamental algorithm in DSP used to convert a discrete signal from the time domain into the frequency domain. Given a finite sequence of N complex samples $x[n]$, the DFT is defined as:

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \quad 0 \leq k \leq N-1 \tag{2.1}$$

where W_N is the twiddle factor, defined as the complex root of unity:

$$W_N = e^{-j\frac{2\pi}{N}} \tag{2.2}$$

Direct evaluation of (2.1) requires N^2 complex multiplications and $N(N-1)$ complex additions, resulting in a computational complexity of $O(N^2)$. For real-time applications on FPGA platforms, this latency is often unacceptable for large N.

The Fast Fourier Transform (FFT) refers to a family of algorithms designed to compute the DFT efficiently with a complexity of $O(N \log_2 N)$. The most common implementation for hardware architectures is the Cooley-Tukey Radix-2 algorithm [4].

Since the analyzed input signal is real-valued, the spectrum exhibits conjugate symmetry, limiting the useful frequency range to the Nyquist frequency $f_s/2$. With $N = 512$ points and a sample frequency $f_s = 48$ kHz, the resulting frequency resolution (bin width) is $\Delta f = f_s/N = 93.75$ Hz.

2.2.1 Radix-2 Decimation-in-Time (DIT)

The Radix-2 DIT algorithm divides a DFT of size N into two smaller DFTs of size $N/2$, assuming N is a power of 2. The input sequence $x[n]$ is decimated into its even-indexed and odd-indexed components:

$$X[k] = \sum_{r=0}^{N/2-1} x[2r] W_{N/2}^{rk} + W_N^k \sum_{r=0}^{N/2-1} x[2r+1] W_{N/2}^{rk} \tag{2.3}$$

This recursive decomposition allows the computation to be performed in $\log_2 N$ stages. In the context of FPGA implementation, this regularity allows for efficient pipelining and memory management.

2.2.2 The Butterfly Operation

The fundamental atomic operation in the Radix-2 FFT is the Butterfly. It takes two complex inputs (A and B) and produces two complex outputs (A' and B'). Figure 2.3 illustrates the signal flow graph of a DIT butterfly.

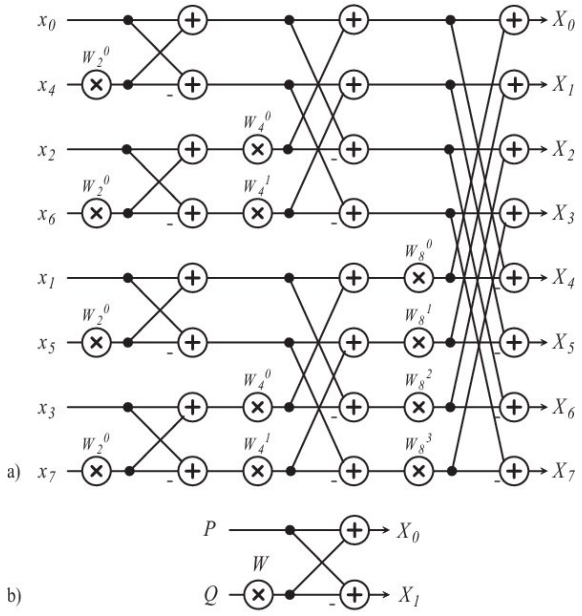

Figure 2.3: Data flow graph of an 8-point Radix-2 DIT FFT algorithm. (a) The complete three-stage computation structure illustrating the bit-reversed input ordering ($x_0, x_4, x_2, \dots$) and natural output ordering ($X_0, X_1, \dots$). (b) The schematic of the fundamental Butterfly unit used in the decomposition, showing inputs P, Q and twiddle factor W[5].

The mathematical operations performed are:

$$A' = A + W_N^k \cdot B \tag{2.4}$$

$$B' = A - W_N^k \cdot B \tag{2.5}$$

Since A, B, and W_N^k are complex numbers, a single butterfly requires four real multiplications and six real additions (two for the complex product, four for the sums/subtractions). In hardware, specific DSP blocks (embedded multipliers) are usually employed to execute these operations in parallel [6].

2.2.3 Bit-Reversal Permutation

The Decimation-in-Time algorithm requires the input data to be rearranged in a non-natural order known as bit-reversal order, while the output $X[k]$ appears in natural order (or vice versa, depending on the implementation). For an index n represented by binary bits $(b_{M-1} \dots b_1 b_0)$, the bit-reversed index is obtained by mirroring the bits to $(b_0 b_1 \dots b_{M-1})$. This permutation is typically handled by the memory addressing logic before the first stage of the FFT computation. As shown in Figure 2.3(a), the input sequence $x[n]$ is reordered ($0 \rightarrow 000_2$, $4 \rightarrow 100_2$, etc.) while the output $X[k]$ is produced in natural order.

2.2.4 Magnitude Approximation Techniques

The output of the FFT consists of complex numbers $X[k] = I[k] + jQ[k]$. To visualize the frequency spectrum on a VGA display, the magnitude $|X[k]|$ is required:

$$|X[k]| = \sqrt{I[k]^2 + Q[k]^2} \tag{2.6}$$

Implementing a square root function on an FPGA is computationally expensive and resource-intensive, often requiring iterative algorithms (like CORDIC in vectoring mode) or large look-up tables (LUT). However, for visualization purposes such as spectral analysis, a small approximation error is acceptable in exchange for hardware simplicity.

Alpha Max plus Beta Min Algorithm

A widely used linear approximation technique is the Alpha Max plus Beta Min algorithm [7]. It approximates the Euclidean distance by a linear combination of the absolute values of the real and imaginary parts. Let:

$$Max = \max(|I|, |Q|)$$
$$Min = \min(|I|, |Q|)$$

The magnitude is approximated as:

$$|X[k]| \approx \alpha \cdot Max + \beta \cdot Min \tag{2.7}$$

By selecting coefficients α and β carefully, the error can be minimized. To avoid the use of hardware multipliers, we selected coefficients that can be implemented using simple logical bit-shifts. We utilize the following coefficients:

$$\alpha = 1, \quad \beta = 0.375 = \frac{3}{8} \tag{2.8}$$

This specific selection allows the calculation to be performed as follows:

$$|X[k]| \approx Max + \left(\frac{1}{4}Min + \frac{1}{8}Min\right) \tag{2.9}$$

In terms of digital logic, the operation in Equation (2.9) corresponds to:

$$\text{Magnitude} = Max + (Min \gg 2) + (Min \gg 3) \tag{2.10}$$

To validate this design choice, we analyzed the approximation accuracy numerically. As illustrated in Figure 2.4, the resulting magnitude shape deviates slightly from the ideal unit circle. Python script was developed to quantify this deviation, revealing a calculated maximum error of 6.80% and an average error of 4.25%. These error margins are well within the acceptable tolerance for a VGA spectral visualization, where the resolution limits the visual perceptibility of such small deviations.

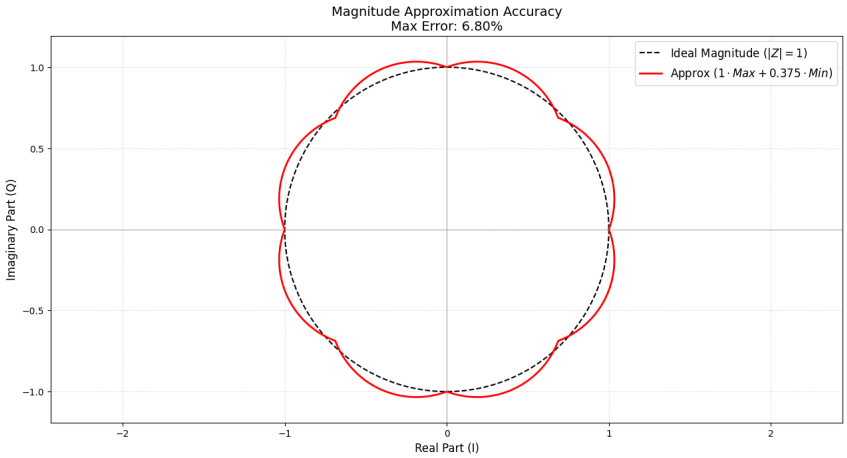

Figure 2.4: Comparison between the ideal Euclidean magnitude (unit circle, dashed line) and the linear approximation implemented on the FPGA (solid red line). The approximation utilizes coefficients $\alpha = 1$ and $\beta = 0.375$. The deviation represents the calculation error, which fluctuates between -2.8% and +6.8% depending on the phase angle of the complex vector.

2.3 Phase-Sensitive Detection (Lock-in Theory)

The Lock-in Amplifier is a measurement technique used to extract a signal with a known carrier wave from a noisy environment. The core of this technique is the Phase-Sensitive Detection [8], which acts as a narrow-band filter with a high quality factor, centered exactly at the reference frequency ω.

Mathematically, let us consider an input signal $V_{in}(t)$ containing the component of interest with amplitude A, frequency ω, and phase Φ, superimposed with broadband noise $n(t)$:

$$V_{in}(t) = A\sin(\omega t + \Phi) + n(t) \tag{2.11}$$

The lock-in detection process involves mixing this input signal with two orthogonal reference signals generated locally at the same frequency ω: a sine wave (In-phase) and a cosine wave (Quadrature):

$$r_I(t) = \sin(\omega t) \tag{2.12}$$
$$r_Q(t) = \cos(\omega t) \tag{2.13}$$

The multiplication of the input signal $V_{in}(t)$ by these references effectively performs a frequency shifting. Considering only the signal component (as noise is uncorrelated and will be averaged out), the mixing operations yield the intermediate signals $X(t)$ and $Y(t)$:

$$X(t) = V_{in}(t) \cdot \sin(\omega t) = A\sin(\omega t + \Phi)\sin(\omega t) \tag{2.14}$$
$$Y(t) = V_{in}(t) \cdot \cos(\omega t) = A\sin(\omega t + \Phi)\cos(\omega t) \tag{2.15}$$

By applying trigonometric identities, these expressions can be expanded into a DC component and an AC component at twice the frequency (2ω):

$$X(t) = \frac{A}{2}[\cos(\Phi) - \cos(2\omega t + \Phi)] \tag{2.16}$$
$$Y(t) = \frac{A}{2}[\sin(\Phi) + \sin(2\omega t + \Phi)] \tag{2.17}$$

To extract the information of interest, these signals are passed through a Low-Pass Filter (LPF). In an ideal scenario, the LPF completely removes the AC components at frequency 2ω as well as the broadband

noise $n(t)$ (which does not have a coherent DC component after mixing). The resulting filtered signals, I and Q, represent the projection of the signal vector onto the reference axes:

$$I = \langle X(t) \rangle_{LPF} = \frac{A}{2} \cos(\Phi) \tag{2.18}$$

$$Q = \langle Y(t) \rangle_{LPF} = \frac{A}{2} \sin(\Phi) \tag{2.19}$$

These DC values contain all the necessary information to reconstruct the magnitude and phase of the original signal relative to the reference. The magnitude R (proportional to amplitude A) and the phase Φ are obtained through rectangular-to-polar conversion:

$$\frac{A}{2} = \sqrt{I^2 + Q^2} \implies A = 2\sqrt{I^2 + Q^2} \tag{2.20}$$

$$\Phi = \arctan\left(\frac{Q}{I}\right) \tag{2.21}$$

In the context of this FPGA implementation, the mixing is performed digitally using a Direct Digital Synthesizer (DDS) for reference generation, the Low-Pass Filter is implemented as a Cascade Integrator-Comb (CIC) filter for efficiency, and the final coordinate conversion is executed using the CORDIC algorithm in vectoring mode.

2.3.1 CORDIC Algorithm

The CORDIC (COordinate Rotation DIgital Computer) algorithm is an iterative technique used to compute hyperbolic and trigonometric functions using only bit-shifts and additions, avoiding computationally expensive multipliers [9]. In this implementation, it is used in Vectoring Mode to compute the magnitude (A) and phase (Φ) of a complex vector given its Cartesian coordinates (x, y).

The relationship between Cartesian and Polar coordinates is defined as:

$$x = A \cos(\Phi) \tag{2.22}$$

$$y = A \sin(\Phi) \tag{2.23}$$

To extract the magnitude A, the input vector (x, y) is rotated towards the x-axis until the y-component becomes zero. The rotation of a vector by an angle $-\Phi$ can be expressed in matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R(-\Phi) \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos(-\Phi) & -\sin(-\Phi) \\ \sin(-\Phi) & \cos(-\Phi) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} A \\ 0 \end{bmatrix} \tag{2.24}$$

By factoring out the cosine term, the rotation matrix can be rewritten to isolate the tangent component:

$$R(-\Phi) = \cos(\Phi) \cdot \begin{bmatrix} 1 & \tan(\Phi) \\ -\tan(\Phi) & 1 \end{bmatrix} \tag{2.25}$$

The angles α_i are chosen such that their tangent is a power of 2:

$$\tan(\alpha_i) = 2^{-i} \quad \Rightarrow \quad \alpha_i = \arctan(2^{-i}) \tag{2.26}$$

This simplification allows the multiplication by the tangent term to be performed via simple binary bit-shifts. The general rotation is thus approximated by N iterations:

$$R(\Phi) \approx \prod_{i=0}^{N-1} K_i \cdot \begin{bmatrix} 1 & \sigma_i 2^{-i} \\ -\sigma_i 2^{-i} & 1 \end{bmatrix} \tag{2.27}$$

where $\sigma_i \in \{+1, -1\}$ indicates the direction of rotation for the i-th iteration, chosen to drive the y-component towards zero. The scaling factor $K_i = \cos(\alpha_i) = (1 + 2^{-2i})^{-1/2}$ represents the gain introduced at each step.

The iterative equations for the CORDIC vectoring mode become:

$$x_{i+1} = x_i - \sigma_i \cdot y_i \cdot 2^{-i} \tag{2.28}$$

$$y_{i+1} = y_i + \sigma_i \cdot x_i \cdot 2^{-i} \tag{2.29}$$

$$z_{i+1} = z_i - \sigma_i \cdot \alpha_i \tag{2.30}$$

Here, z acts as an accumulator that tracks the total angle rotated. The direction σ_i is determined by the sign of y_i: if $y_i \geq 0$, we rotate clockwise ($\sigma_i = -1$ in standard derivation, but implementation specific); effectively, the goal is to reduce $|y|$.

After N iterations, the cumulative system gain A_n is:

$$A_n = \prod_{i=0}^{N-1} \sqrt{1 + 2^{-2i}} \approx 1.647 \tag{2.31}$$

Consequently, the final outputs are:

$$x_N \approx A_n \cdot \sqrt{x_{in}^2 + y_{in}^2} \tag{2.32}$$

$$y_N \approx 0 \tag{2.33}$$

$$z_N \approx \arctan(y_{in}/x_{in}) \tag{2.34}$$

To obtain the exact magnitude, the result x_N must be scaled by the inverse of the CORDIC gain ($1/A_n \approx 0.607$). The standard CORDIC algorithm has a limited convergence range defined by the sum of the elementary rotation angles, $\sum_{i=0}^{\infty} \arctan(2^{-i}) \approx 1.74$ rad (99.7°). Consequently, the algorithm is natively restricted to input vectors located in the first and fourth quadrants (the right half-plane, where $-\pi/2 < \Phi < \pi/2$). To overcome this limitation, in this FPGA implementation, specific logic handles vectors in the second and third quadrants (where $x_{in} < 0$) by initially rotating the input by π (pre-rotation), effectively mapping them to the right half-plane and extending the valid operation range to the full circle.

2.3.2 CIC Filter Theory

To extract the DC component from the mixed signals $X(t)$ and $Y(t)$, a LPF is required. In FPGA-based DSP, the CIC filter [10], is a widely adopted architecture for this purpose. It is particularly efficient for applications requiring both filtering and decimation (sample rate reduction), as it eliminates the need for hardware multipliers, utilizing only adders, subtractors, and delay elements.

The CIC filter consists of N cascaded integrator stages operating at the high input sampling rate (f_s), followed by a decimator by a factor R, and finally N cascaded comb stages operating at the low output sampling rate (f_s/R). The system transfer function in the z-domain is given by:

$$H(z) = H_I(z) \cdot H_C(z) = \left(\frac{1}{1 - z^{-1}}\right)^N \cdot (1 - z^{-RM})^N \tag{2.35}$$

where:

- N is the order of the filter (number of stages).
- R is the decimation factor.
- M is the differential delay in the comb stage (typically $M = 1$ or 2).

Combining these stages and moving the decimator results in an equivalent transfer function relative to the high sampling rate:

$$H(z) = \left(\frac{1 - z^{-RM}}{1 - z^{-1}}\right)^N \tag{2.36}$$

Frequency Response and Zeros

The frequency response of a CIC filter typically exhibits a sinc-like shape. The magnitude response is defined as:

$$|H(f)| = \left| \frac{\sin(\pi M f)}{\sin(\frac{\pi f}{R})} \right|^N \tag{2.37}$$

The filter introduces spectral zeros at multiples of the decimated sampling frequency $f_{out} = f_s/R$. This structure effectively acts as a moving average filter. For a Lock-in Amplifier, this is ideal because it extracts the stable DC component (the signal of interest) while filtering out background noise and the unwanted 2ω frequency component generated by the mixer.

Register Width and Bit Growth

One critical design constraint of CIC filters is the internal bit growth. Since the integrator stages have unity feedback with no loss, they can potentially overflow. However, CIC's math has a C2 wrap around that ensures correct results provided that the total register width is sufficient to hold the maximum possible output value.

The maximum gain G_{max} of a CIC filter is given by:

$$G_{max} = (R \cdot M)^N \tag{2.38}$$

Consequently, the number of bits B_{out} required at the output of the filter stages to avoid data loss relative to the input bit width B_{in} is:

$$B_{out} = B_{in} + \lceil N \log_2(R \cdot M) \rceil \tag{2.39}$$

In our specific implementation for the Lock-in module, we selected a 2nd-order filter ($N = 2$) with a differential delay $M = 1$ and a decimation factor equal to the buffer depth ($R = 512$). Using the audio sample width of $B_{in} = 42$ bits (accumulated from previous stages), the bit growth is calculated as:

$$\text{Bit Growth} = 2 \cdot \log_2(512) = 2 \cdot 9 = 18 \text{ bits} \tag{2.40}$$

Therefore, the internal integrators and comb registers are sized to $42 + 18 = 60$ bits. This guarantees that no overflow error propagates to the final output, preserving the high dynamic range required for precise phase-sensitive detection. The final output is then truncated back to the system data width after the comb stages.

2.3.3 Direct Digital Frequency Synthesis (DDFS)

The DDFS is a method for generating arbitrary periodic waveforms from a fixed-frequency reference clock using DSP blocks. In the context of the lock-in amplifier, the DDFS is critical for generating the reference signals needed for the demodulation process.

The architecture implemented is based on a phase accumulation approach then converted to amplitude oscillations via a LUT. The system is designed to generateSine and Cosine simultaneously, which are essential for I/Q demodulation.

Phase Accumulation

The core of the DDFS is the Phase Accumulator. It acts as a digital integrator that increments a phase variable $\phi[n]$ by a constant value, known as the Frequency Tuning Word (FTW), at every update cycle.

Given a phase accumulator with a bit-width of N, the operation can be described by the following difference equation:

$$\phi[n] = (\phi[n-1] + FTW) \mod 2^N \tag{2.41}$$

where:

- N is the accumulator width (e.g., $N = 16$).

- FTW is the input control word that determines the output frequency.
- The modulo 2^N operation represents the natural overflow behavior of binary adders, corresponding to the periodicity of the phase (wrapping around 2π).

The rate at which the accumulator updates is determined by the sampling frequency f_s (in this system, governed by the audio strobe at 48 kHz). The output frequency f_{out} is linearly related to the tuning word by the fundamental DDFS equation:

$$f_{out} = \frac{FTW \cdot f_s}{2^N} \tag{2.42}$$

From (2.42), the frequency resolution Δf, which is the smallest possible frequency step change, is defined as:

$$\Delta f = \frac{f_s}{2^N} \tag{2.43}$$

For a system with $f_s = 48\,\text{kHz}$ and $N = 16$, this yields a resolution of approximately 0.73 Hz, allowing for precise tuning of the reference signal.

Amplitude Conversion

The output of the accumulator represents a linear phase ramp ranging from 0 to $2^N - 1$. To generate a sinusoidal wave, this phase value must be mapped to amplitude values. This is achieved using a LUT that stores samples of a sine wave.

Since the size of the memory grows exponentially with the address width, it is impractical to use the full N bits of the accumulator to address the LUT. Instead, a technique known as Phase Truncation is employed. Only the M MSBs of the accumulator are used as the address pointer for the LUT, where $M < N$.

The truncated address $A[n]$ is given by:

$$A[n] = \lfloor \frac{\phi[n]}{2^{N-M}} \rfloor \tag{2.44}$$

In the proposed implementation, $M = 10$, balancing the memory usage and the phase quantization noise.

Quadrature Signal Generation

Instead of using two separate DDFS blocks or two distinct LUT contents, the property of trigonometric symmetry is exploited:

$$\cos(\theta) = \sin\left(\theta + \frac{\pi}{2}\right) \tag{2.45}$$

In the digital domain, a phase shift of $\pi/2$ corresponds to one quarter of the full scale of the address space. Given the LUT depth of 2^M words, the address offset is calculated as:

$$\text{Offset}_{90^\circ} = \frac{2^M}{4} = 2^{M-2} \tag{2.46}$$

Therefore, the system generates the two orthogonal carriers using a single Sine LUT by reading from two different addresses in the same clock cycle:

$$Q_{out}[n] = \text{LUT}(A[n]) \tag{2.47}$$

$$I_{out}[n] = \text{LUT}\left((A[n] + 2^{M-2}) \mod 2^M\right) \tag{2.48}$$

This efficient architecture ensures perfect phase synchronization between the I and Q channels, as they share the same underlying phase accumulator.

2.4 VGA Signaling and Timing Standard

The Video Graphics Array (VGA) is a widely adopted analog interface standard for video transmission. Although modern displays utilize digital technologies (LCD, OLED), the VGA protocol remains relevant for legacy compatibility and embedded systems due to its simplicity. On the Altera DE2 board, the FPGA generates digital video signals, which are converted into analog voltages by the Analog Devices ADV7123, a triple 10-bit high-speed video DAC [1], [2].

2.4.1 Analog Interface and Synchronization

The VGA standard typically utilizes five active signals to drive a monitor: three analog color channels (Red, Green, Blue) and two digital synchronization signals (Horizontal Sync and Vertical Sync).

- **RGB Channels:** These carry the color information as analog voltage levels ranging from 0 V (black) to 0.7 V (full intensity). The DAC converts the digital pixel data from the FPGA into these continuous signals.
- **Synchronization Signals (HSync, VSync):** These are TTL-compatible digital pulses that coordinate the scanning process of the display. They are typically active low.
- **Blanking Interval:** Historically derived from Cathode Ray Tube (CRT) technology, the blanking interval represents the time required for the electron beam to return to the start of a line (horizontal retrace) or the start of a frame (vertical retrace). During this time, the color signals must be driven to a black level (ground) to prevent visual artifacts. On the DE2 board, this is managed via the $\overline{\text{BLANK}}$ signal fed to the ADV7123.

2.4.2 Timing Protocol

The video transmission is a serial process where pixels are transmitted left-to-right and top-to-bottom. The timing is strictly governed by the Pixel Clock. A complete line scan is divided into four distinct intervals:

1. **Active Video:** The interval where valid RGB data is driven to the display.
2. **Front Porch (FP):** A guard interval immediately following the active video, where signals are blanked before the synchronization pulse occurs.
3. **Sync Pulse:** The synchronization trigger (HSync or VSync) that signals the monitor to reset its internal counters (move to the next line or next frame).
4. **Back Porch (BP):** A stabilization interval after the sync pulse, allowing the beam (or logic) to settle before the next active video phase begins.

This structure applies to both horizontal (line) and vertical (frame) timing. The total duration of a horizontal line (T_{line}) in terms of clock cycles is the sum of these four components:

$$T_{line} = T_{active} + T_{fp} + T_{sync} + T_{bp} \tag{2.49}$$

Similarly, the total frame time (T_{frame}) is the sum of visible lines and vertical non-display lines.

2.4.3 The 640x480 @ 60 Hz Standard

For this project, we target the standard Industry Standard Architecture (ISA) resolution of 640×480 pixels at a refresh rate of approximately 60 Hz. The timing constraints necessitate a specific Pixel Clock frequency (f_{clk}).

The total number of horizontal clocks per line is 800, and the total number of lines per frame is 525. Therefore, the required pixel clock is calculated as:

$$f_{clk} = \text{Total Horizontal Pixels} \times \text{Total Vertical Lines} \times \text{Refresh Rate} \tag{2.50}$$

$$f_{clk} = 800 \times 525 \times 60 \approx 25.2\,\text{MHz} \tag{2.51}$$

In our implementation, a 25 MHz clock (derived from the 50 MHz system clock) is sufficient to drive the monitor within acceptable tolerance margins. Table 2.2 summarizes the timing parameters used for the controller logic.

Table 2.2: VGA Timing Parameters for 640x480 @ 60 Hz (25 MHz Clock).

Parameter	**Horizontal (Clocks)**	**Vertical (Lines)**
Active Display Area	640	480
Front Porch (FP)	16	10
Sync Pulse	96	2
Back Porch (BP)	48	33
Total	**800**	**525**

The synchronization signals HSync and VSync are generated as active-low pulses within the Sync Pulse window, while the Video On signal (active high) is asserted only when the pixel counters are within the Active Display Area range ($0 \leq x < 640$ and $0 \leq y < 480$).

Chapter 3

Implementation of Processing Modules

3.1 Global Architecture and Clock Distribution

As illustrated in Figure 3.1, the top-level architecture integrates three primary subsystems: the Audio Capture Unit, the DSP Core (comprising both the FFT and Lock-in Amplifier modules), and the VGA Visualization Interface.

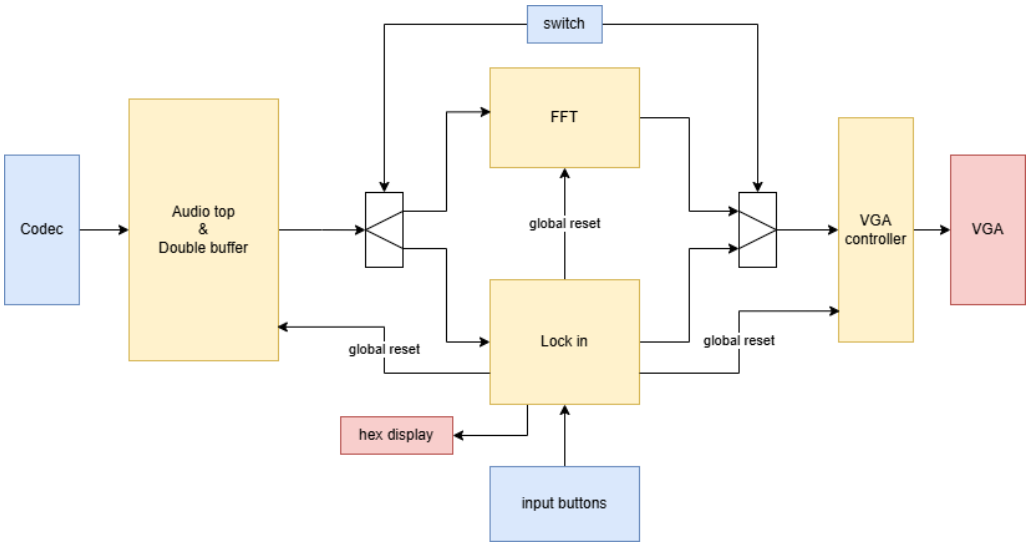

Figure 3.1: Block diagram of the overall structure of the module. Blue blocks represent external inputs, Red blocks represent physical outputs, and Yellow blocks indicate the internal processing sub-modules.

3.1.1 Clock Distribution and Reset Strategy

The design operates across three distinct clock domains to satisfy the specific timing requirements of the peripheral interfaces:

- **System Clock (50 MHz):** Derived directly from the board's oscillator, this clock drives the core logic, including the FFT processing, the Lock-in math operations, and the control FSMs.
- **Audio Clock (18.432 MHz):** Generated via an internal Phase-Locked Loop (PLL) within the `top_audio` module. This frequency is mandatory for the Wolfson WM8731 codec to achieve a precise 48 kHz sampling rate.
- **Pixel Clock (25 MHz):** Used by the VGA controller to drive the monitor at a 640×480 resolution with a 60 Hz refresh rate.

A synchronous global reset strategy is implemented to ensure the system starts in a deterministic state. Triggered by a push-button (Key 0), the reset signal propagates to all sub-modules, clearing the internal buffers, resetting the FSMs to their `IDLE` states, and re-initializing the I2C configuration sequence for the audio codec.

3.1.2 Mode Selection Multiplexer

To optimize resource usage while providing dual functionality, the system features a runtime mode selection mechanism controlled by a hardware switch (`i_switch_mode`). As seen in the `vga_top_system` module, this switch drives a combinatorial multiplexer that routes the video signals to the Video DAC. Depending on the switch state, the multiplexer selects either the spectral data from the FFT visualizer or the magnitude/phase plots from the Lock-in visualizer. This allows the user to seamlessly toggle between the Spectrum Analyzer and Lock-in Amplifier views in real-time without requiring FPGA reconfiguration.

3.2 Audio Capture Unit

The Audio Capture Unit is the hardware front-end responsible for interfacing the Altera Cyclone II FPGA with the Wolfson WM8731 codec. The top-level module, `top_audio`, integrates three main sub-blocks: a PLL (generated by Quarts Megawizard tool) for clock generation, an I2C controller for codec configuration, and an I2S receiver for real-time data streaming.

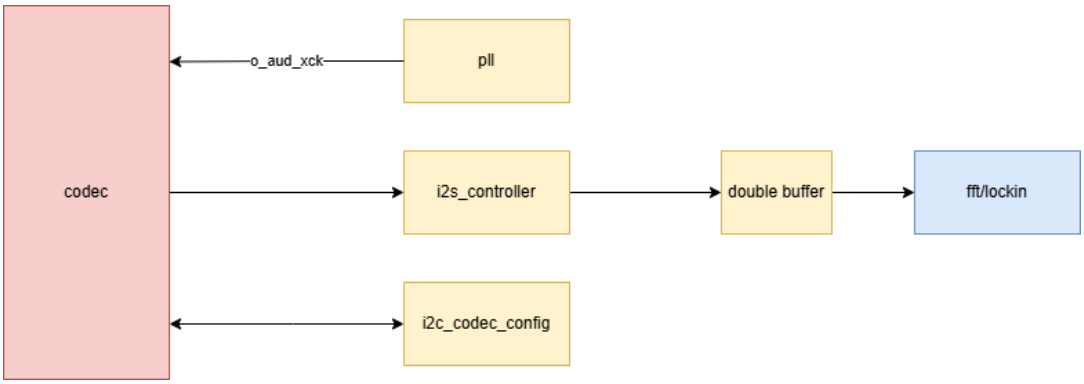

Figure 3.2: RTL Block Diagram of the Audio Capture Unit.

The module also instantiates an `audio_pll` to synthesize the precise 18.432 MHz MCLK required by the codec to operate at a 48 kHz sampling rate, deriving it from the 50 MHz system clock.

3.2.1 I2C Codec Configuration Controller

Upon system reset, the WM8731 must be initialized via the I2C control interface. The implementation is based on the `i2c_config_codec_standard` module, which operates as a bit-bang master. Since the DE2 board pulls up the I2C lines externally, the FPGA drives the SDA and SCL lines in open-drain mode: logic '0' is driven strongly, while logic '1' is achieved by setting the line to high impedance (Z).

The configuration sequence is managed by a FSM synchronized to a 100 kHz tick generated by an internal clock divider. The FSM iterates through a LUT containing 10 configuration words. Listing 3.1 shows the register mapping used to enable the digital audio path, select the microphone input with boost, and set the device as a Master for the I2S interface.

```
always @(*) begin
    case (reg_index)
            4'd0: current_reg = 16'h1E00; // Reset
            4'd1: current_reg = 16'h0C00; // Power Down control: every 1
                enables a chip function, evry 0 disables it.
            4'd2: current_reg = 16'h0815; // Analog Audio Path: 1's are DACSEL
                (Digital audio sent to output), INSEL (Audio from mic),
                MICBOOST (+20dB)
```

```
            4'd3: current_reg = 16'h0A00; // Digital Audio Path: all 0's (De
                emphasis, Soft mute, DC filtering)
            4'd4: current_reg = 16'h0E4A; // Digital audio interface format:
                master, 24 bit, I2S
            4'd5: current_reg = 16'h1002; // Sampling control: Normal mode
                (18.432 MHz), BOSR (384fs), 48kHz
            4'd6: current_reg = 16'h1201; // Active control: Digital interface
                active
            4'd7: current_reg = 16'h0097; // Left line in: 0dB
            4'd8: current_reg = 16'h0297; // Right line in: 0dB
            4'd9: current_reg = 16'h0479; // Headphone L gain (for debugging,
                playback)
            default: current_reg = 16'h0679; // Headphone R gain
        endcase
end
```

Listing 3.1: WM8731 Register Configuration LUT

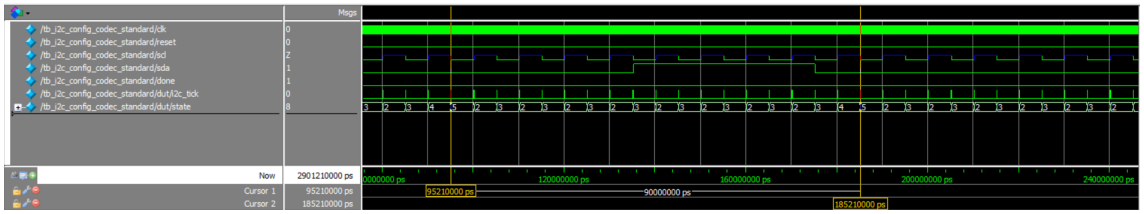

Figure 3.3: ModelSim simulation of an I2C write transaction: zoom on a single byte transmission and the corresponding ACK bit.

3.2.2 I2S Streaming and Clock Domain Crossing

Once configured, the codec acts as the bus master, generating the (BCLK) and the ADCLRCK. The `i2s_controller` module captures the serial data stream (`i_aud_adcdat`) on the rising edge of BCLK.

The deserialization logic employs a shift register that accumulates bits whenever theADCLRCK is stable. An edge detection mechanism on `lrclk` identifies the completion of a channel transmission. When a rising edge on `lrclk` is detected (transition from Left to Right channel), the parallelized 24-bit word is latched, and a valid flag is asserted.

```
always @(posedge bclk) begin
    lrclk_d1 <= lrclk;
    if (lrclk_d1 != lrclk) begin
        bit_cnt <= 5'd0;
        if (lrclk == 1'b1) begin
            o_audio_data <= shift_reg_rx;
            o_audio_valid <= 1'b1;
        end
    end
end
```

Listing 3.2: I2S Edge Detection and Data Latching

Since the I2S logic runs on the codec's clock domain (asynchronous to the FPGA's 50 MHz clock), a CDC strategy is implemented in `top_audio`. We utilize an optimized synchronizer approach: the `i2s_valid_bclk` flag is passed through a 3-stage shift register clocked by the system clock. The data bus itself is not passed through flip-flops to save resources; instead, it is sampled only when the synchronized valid pulse is stable, ensuring data integrity without setup/hold violations.

```
wire valid_pulse_sys = (valid_sync[1] && !valid_sync[2]);
always @(posedge clk or posedge reset) begin
```

```
    valid_sync <= {valid_sync[1:0], i2s_valid_bclk};
    if (valid_sync[1] && !valid_sync[2]) begin
        audio_data_sys <= i2s_data_bclk;
    end
end
```

Listing 3.3: CDC Pulse Synchronizer in top_audio

3.2.3 Double Buffering Strategy

To decouple the continuous audio stream from the block-based processing of the FFT and Lock-in modules, the `i2s_double_buffer` module implements a double buffer memory scheme. The memory is physically instantiated as a single array of $2 \times N$ words, where $N = 512$ is the buffer depth.

The addressing logic uses the MSB of the address as the bank selector. A `write_buffer_sel` bit toggles automatically when the write pointer wraps around (reaches 511), instantaneously swapping the read and write banks. This ensures that while the DSP core reads a static frame from one bank, the I2S controller fills the other bank without race conditions.

```
always @(posedge clk) begin
    if (i_audio_valid) begin
        mem_buffer[{write_buffer_sel, write_addr}] <= i_audio_data;
        if (write_addr == BUFFER_DEPTH[ADDR_WIDTH-1:0] - 1'b1) begin
            write_addr <= {ADDR_WIDTH{1'b0}};
            write_buffer_sel <= ~write_buffer_sel;
            read_buffer_sel <= write_buffer_sel;
            o_data_ready_reg <= 1'b1;
        end else begin
            write_addr <= write_addr + 1'b1;
        end
    end
end
```

Listing 3.4: Ping-Pong Buffer Swapping Logic

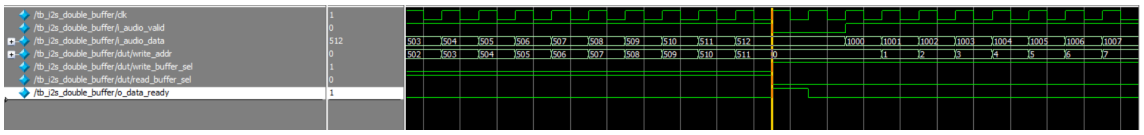

Figure 3.4: Simulation of the Double Buffer swap event. The `o_data_ready` flag asserts when `write_addr` wraps, signaling a full frame is available.

3.3 The Fast Fourier Transform Module

The FFT module is the core computational engine of the system, designed to compute a 512-point DFT using the Radix-2 DIT algorithm. The top-level module, `fft_top`, acts as a coprocessor that instantiates and interconnects the control logic, memory blocks, and arithmetic cores.

The architecture is built around a shared memory scheme where a single Dual-Port RAM stores the intermediate results of the computation. This design minimizes resource usage compared to fully pipelined (streaming) architectures, making it suitable for the limited logic elements of the Cyclone II FPGA. The module operates on 24-bit signed fixed-point data to maintain a high dynamic range for audio signals.

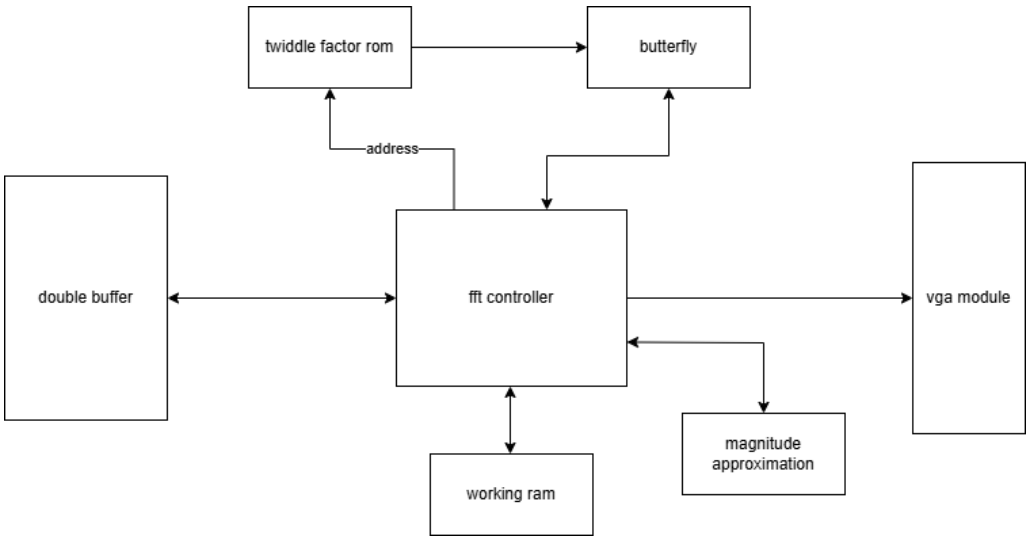

Figure 3.5: RTL Block Diagram of the FFT Subsystem, highlighting the data path between the shared memory and the arithmetic butterfly core.

3.3.1 Controller

The control logic is implemented in the `fft_controller` module, which governs the data flow and the execution timing of the Cooley-Tukey algorithm. The controller is designed as a FSM comprising 13 states, grouped into three main operational phases: Data Loading, Computation, and Magnitude Extraction.

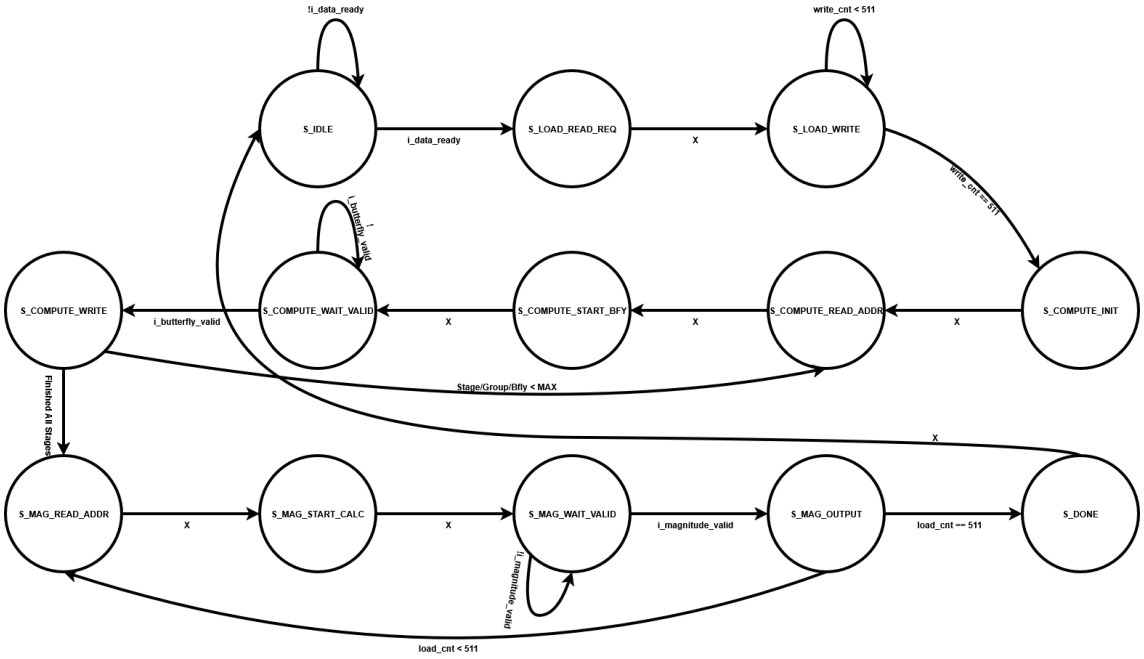

Figure 3.6: FSM diagram of the FFT Controller. The transition from `S_IDLE` is triggered by the `i_data_ready` signal from the Audio Capture Unit. This is a Moore's FSM, as the output is dependent only on the status.

Data Loading and Bit-Reversal

The process begins when the Audio Capture Unit signals that a full buffer of 512 samples is ready via the `i_data_ready` signal (generated by the `i2s_double_buffer` module). The controller transitions

to S_LOAD_READ_REQ and sequentially reads data from the external buffer.

The DIT algorithm requires the input sequence to be permuted in bit-reversed order. To optimize latency, we perform this permutation during the loading phase. The controller generates a linear read address for the input buffer but writes to the internal RAM using a bit-reversed address. This is implemented efficiently in hardware using a `generate` block:

```
genvar i;
generate
    for (i = 0; i < LOG2_FFT_POINTS; i = i + 1) begin : bit_rev_gen
        assign bit_reversed_write_addr[i] = write_counter_reg[LOG2_FFT_POINTS
            -1-i];
    end
endgenerate
```

Listing 3.5: Bit-reversal logic implementation using a generate block.

Simulation of this phase confirms that while the `load_counter_reg` increments linearly (0, 1, 2...), the `o_ram_addr_a` assumes the bit-reversed values (0, 256, 128...), correctly ordering the data for the first stage of the FFT.

Computation Loop

The core processing is handled by states `S_COMPUTE_INIT` through `S_COMPUTE_WRITE`. The controller manages the three nested loops characteristic of the Radix-2 algorithm:

1. **Stages:** Iterates $\log_2(512) = 9$ times.
2. **Groups:** Iterates through the butterfly groups within a stage.
3. **Butterflies:** Iterates through the butterfly operations within a group.

The iteration logic is embedded directly into the state transitions of the FSM. As shown in Listing 3.6, the controller checks the boundary conditions of the butterfly, group, and stage indices in the write state to determine the next step.

```
if (stage_reg == LOG2_FFT_POINTS - 1 && group_idx_reg == num_groups - 1 &&
    bfly_idx_reg == bfly_per_group - 1) begin
    state_next = S_MAG_READ_ADDR;
end else if (group_idx_reg == num_groups - 1 && bfly_idx_reg == bfly_per_group
    - 1) begin
    stage_next = stage_reg + 1'b1;
    group_idx_next = {LOG2_FFT_POINTS{1'b0}};
    bfly_idx_next = {LOG2_FFT_POINTS{1'b0}};
end else if (bfly_idx_reg == bfly_per_group - 1) begin
    group_idx_next = group_idx_reg + 1'b1;
    bfly_idx_next = {LOG2_FFT_POINTS{1'b0}};
end else begin
    bfly_idx_next = bfly_idx_reg + 1'b1;
end
```

Listing 3.6: Hardware implementation of the nested loops via conditional state transitions.

In the S_COMPUTE_READ_ADDR state, the controller calculates the effective RAM addresses (A and B) and the corresponding Twiddle Factor address (W_N^k). The address generation logic depends on the current stage and group index:

```
wire [LOG2_FFT_POINTS-1:0] m_half = 1'b1 << stage_reg;
wire [LOG2_FFT_POINTS-1:0] m = 1'b1 << (stage_reg + 1);
wire [LOG2_FFT_POINTS-1:0] addr_a = (group_idx_reg * m) + bfly_idx_reg;
wire [LOG2_FFT_POINTS-1:0] addr_b = addr_a + m_half;
wire [LOG2_FFT_POINTS-1:0] twiddle_addr = bfly_idx_reg * (FFT_POINTS >> (
    stage_reg + 1));
```

Listing 3.7: Address calculation logic for Radix-2 DIT.

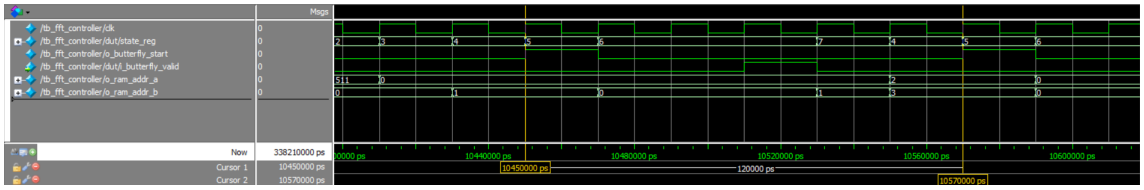

Figure 3.7: Controller Simulation. The FSM asserts start in state 5, then waits in `S_COMPUTE_WAIT_VALID` (state 6) for exactly 3 clock cycles, perfectly matching the latency of the pipelined butterfly unit before writing results (state 7).

3.3.2 Computation Core

The arithmetic heavy-lifting is performed by the `fft_butterfly` module, assisted by the `twiddle_factor_rom` and the `fft_working_ram`.

The Pipelined Butterfly Unit

To maximize the operating frequency, the Butterfly unit is designed with a 3-stage pipeline structure:

- **Stage 1 (Input Registration):** Registers the complex inputs A, B, and the coefficient W.
- **Stage 2 (Complex Multiplication):** Computes the product $B \cdot W$. Since the inputs are 24-bit, the multiplication results in 48-bit values. The result is then scaled back to 24-bit using a specific rounding logic to preserve spectral purity.
- **Stage 3 (Sum/Difference and Scaling):** Computes $A' = A + (B \cdot W)$ and $B' = A - (B \cdot W)$. To prevent numeric overflow (bit growth) during the 9 stages, the output is divided by 2 at every stage.

Rounding vs. Truncation Strategy A critical design choice in Stage 2 is the handling of bit-width reduction. When scaling the 48-bit product back to 24 bits, a simple arithmetic right shift (`>>>`) would act as a floor function (truncation). In two's complement representation, truncation introduces a systematic negative error (mean error of -0.5 LSB). Over the multiple stages of the FFT, this "DC bias" accumulates, leading to significant spectral noise and reduced dynamic range.

To mitigate this, we implemented a Round-to-Nearest logic. As shown in Listing 3.8, a rounding constant (`round_const`) equivalent to half the weight of the Least Significant Bit is added to the product before shifting. This ensures that the quantization error is zero-mean, preventing bias accumulation and maximizing the Signal-to-Noise Ratio (SNR).

```
1 localparam SHIFT_VAL = TWIDDLE_WIDTH - 1;
2 wire signed [PRODUCT_WIDTH-1:0] round_const = 1'b1 <<< (SHIFT_VAL - 1);
3 wire signed [DATA_WIDTH-1:0] prod_re_scaled = (prod_re_full + round_const) >>>
      SHIFT_VAL;
```

Listing 3.8: Rounding logic in the butterfly multiplier stage.

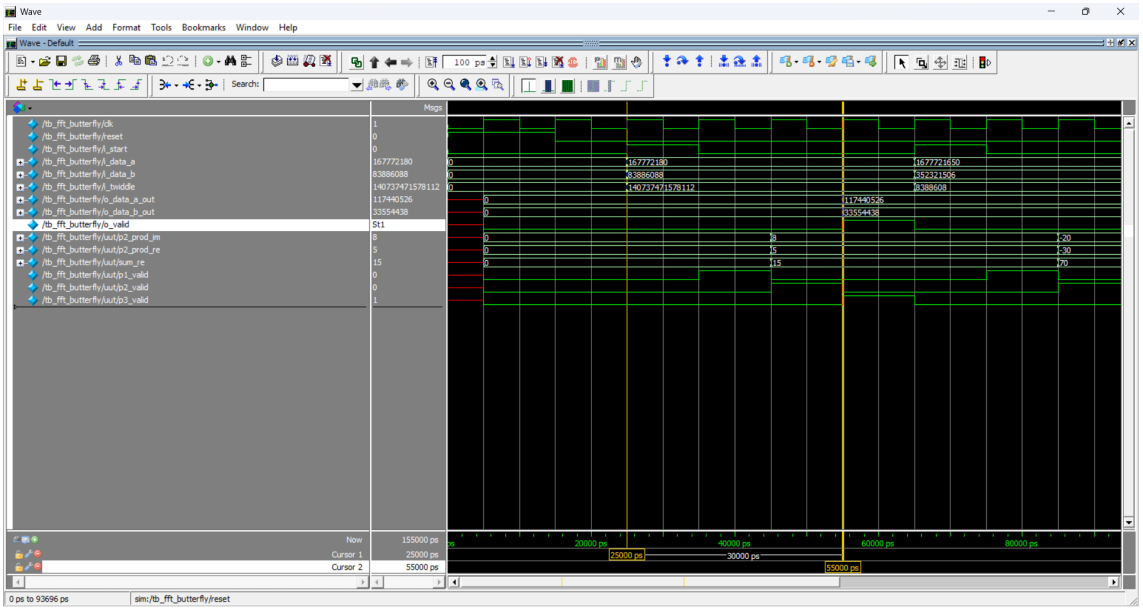

Figure 3.8: ModelSim simulation of the Butterfly unit. The cursors highlight the 3-clock-cycle latency between the input trigger (`i_start`) and the valid output (`o_valid`). The rounding logic is verified by comparing the internal 48-bit product with the scaled 24-bit output.

Twiddle Factor Optimization and Symmetry

The `twiddle_factor_rom` module stores the complex roots of unity W_N^k required for the butterfly operations. A direct implementation for a 512-point FFT would require storing 512 entries of 48 bits. To optimize the hardware resource utilization, we exploited the rotational symmetry property of the twiddle factors:

$$W_N^{k+N/2} = -W_N^k \tag{3.1}$$

This property allows the ROM to store only the first $N/2 = 256$ coefficients, effectively halving the memory footprint. This optimization is crucial, as it significantly reduces the consumption of M4K Block RAMs.

The addressing logic interprets the MSB of the requested index k as an inversion flag.

- If $0 \leq k < 256$ (MSB = 0), the value is read directly from memory.
- If $256 \leq k < 512$ (MSB = 1), the module outputs the arithmetic negation of the stored value corresponding to index $k - 256$.

Since the data is stored in C2's format, the negation $-W_N^k$ is implemented using bitwise inversion followed by an increment (+1). Crucially, this operation must be performed independently on the Real and Imaginary parts of the 48-bit word to maintain mathematical correctness.

```
wire [DATA_WIDTH-1:0] negated_data = {
    ~ram_data_out[DATA_WIDTH-1:PART_WIDTH] + 1'b1, // Negate Real
    ~ram_data_out[PART_WIDTH-1:0] + 1'b1           // Negate Imaginary
};

assign twiddle_factor_q = (invert_flag_out) ? negated_data : ram_data_out;
```

Listing 3.9: Twiddle factor symmetry and negation logic.

Resource vs. Timing Trade-off: The introduction of the negation logic adds a small combinational delay to the critical path. However, static timing analysis confirms that this penalty is negligible compared to the substantial 50% saving in memory resources. The output MUX ensures that the latency remains consistent (1 clock cycle) regardless of the address accessed.

Memory Management

The `fft_working_ram` is a Dual-Port RAM. We utilized the synthesis attribute `(* ramstyle = "no_rw_check" *)` to force the Quartus synthesizer to infer M4K embedded memory blocks instead of logic registers. Without this attribute, the synthesizer's conflict detection logic (Read-During-Write behavior) would prevent the usage of Block RAMs, exhausting the FPGA's logic resources. Simulations verified that the RAM has a read latency of 1 clock cycle and that the two ports (A and B) operate independently, allowing simultaneous read/write operations required by the butterfly core.

3.3.3 Magnitude Approximation Implementation

For visualization, the complex result $R + jI$ is converted to magnitude $|Z|$ using the "Alpha Max plus Beta Min" algorithm. The approximation $|Z| \approx \max(|R|, |I|) + 0.375 \cdot \min(|R|, |I|)$ avoids expensive square roots and multipliers. The coefficient 0.375 is implemented via bit-shifts:

```
wire [DATA_WIDTH-1:0] min_div_4 = p2_min >> 2;
wire [DATA_WIDTH-1:0] min_div_8 = p2_min >> 3;
wire [DATA_WIDTH-1:0] min_scaled = min_div_4 + min_div_8;

wire [DATA_WIDTH:0] magnitude_full = {1'b0, p2_max} + {1'b0, min_scaled};
```

Listing 3.10: Magnitude approximation calculation.

A saturation check is included: if the resulting sum exceeds the 24-bit range, the output is clamped to the maximum value to prevent overflow wrapping, which would cause visual artifacts on the display.

3.4 The Lock-in Amplifier Module

The Lock-in Amplifier module implements the secondary processing function of the system. It accepts an input buffer of 512 samples, consisting of 24-bit signed integers acquired by the Audio Capture Unit. The module processes this data to extract the amplitude and phase of a specific frequency component ω. The output consists of two 42-bit integers representing the magnitude and phase, respectively.

Structurally, the design employs a pipelined architecture divided into three main stages, corresponding to the mathematical operations described in Section 2.3. The first stage is a Mixer, which multiplies incoming samples by orthogonal reference waves generated by a DDFS. This is followed by a Low-Pass Filter, implemented as a CIC filter. Finally, a CORDIC vectoring module converts the filtered I/Q components into magnitude and phase.

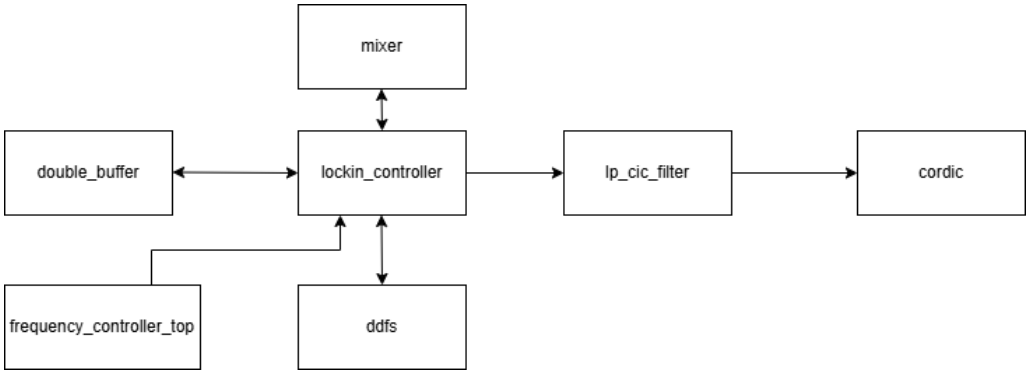

Figure 3.9: Block structure of the Lock-in module.

3.4.1 DDFS

The DDFS module acts as the local oscillator for the mixing stage, generating the orthogonal reference signals $\sin(\omega t)$ and $\cos(\omega t)$. The architecture implements the phase accumulation and truncation method

described in Section 2.3.3. To optimize hardware resources, the module utilizes a single sine LUT to generate both components simultaneously.

The logic operates at the audio sampling rate (48 kHz). At each strobe, the 16-bit phase accumulator is incremented by the tuning word. The Q signal is obtained by reading the ROM with a static address offset corresponding to a 90° phase shift, exploiting the symmetry of trigonometric functions.

```
always @(posedge clk) begin
    if (sample_en) begin
        phase_acc <= phase_acc + tuning_word;
    end
end

wire [LUT_DEPTH-1:0] addr_sin = phase_acc[ACC_WIDTH-1 : ACC_WIDTH-LUT_DEPTH];

localparam OFFSET_90_DEG = 1 << (LUT_DEPTH - 2);

wire [LUT_DEPTH-1:0] addr_cos = addr_sin + OFFSET_90_DEG;

always @(posedge clk) begin
    sine_out   <= sine_rom[addr_sin];
    cosine_out <= sine_rom[addr_cos];
end
```

Listing 3.11: DDFS Core Logic: Phase Accumulator and Quadrature Offset

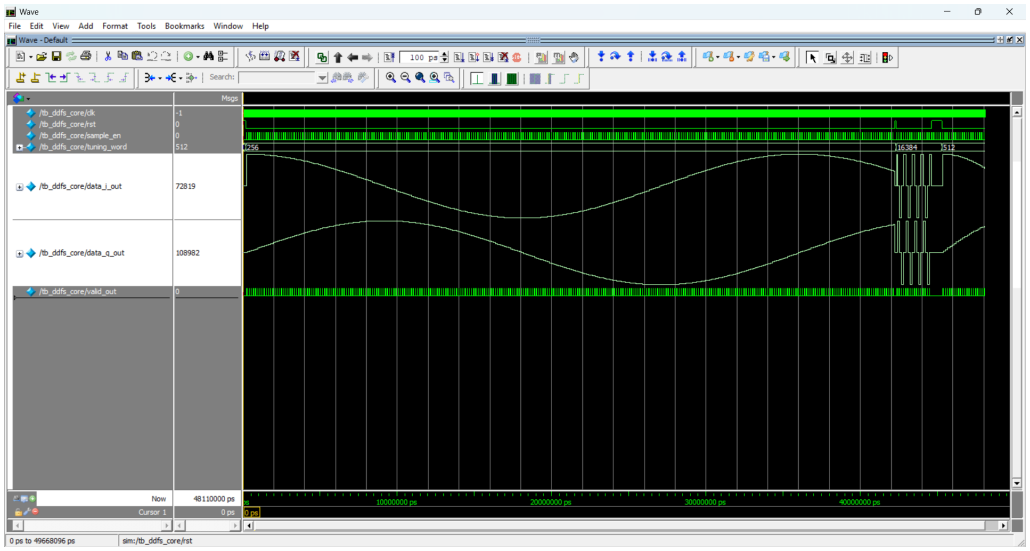

Figure 3.10: Functional simulation of the DDFS module showing the generated Sine and Cosine signals. The phase relationship is strictly maintained by the hardware address offset, ensuring orthogonality for the mixer stage across all frequencies.

3.4.2 User Interface and Frequency Control

The system utilizes the four push-buttons on the DE2 board to control the reference frequency ω. Key 3 selects the scale factor (powers of 10: 1, 10, 100, 1000 Hz), while Key 2 and Key 1 increment and decrement the frequency value by the selected scale, respectively. Key 0 triggers a global reset. The current frequency and scale factor are visualized in real-time on the 7-segment displays.

Input Debouncing and Synchronization

The raw input from the mechanical buttons is prone to noise and bouncing. Furthermore, the buttons are asynchronous with respect to the system clock. To address this, a `button_edge_detector` module

is employed for each key. It synchronizes the input signal and generates a clean, single-cycle pulse on the falling edge (button press).

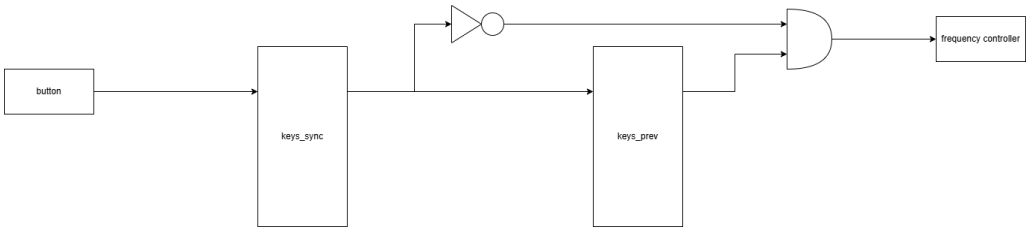

Figure 3.11: Structure of the `button_edge_detector` module. The shift register `keys_sync` synchronizes the asynchronous input. The logic condition `(!keys_sync && keys_prev)` detects the falling edge transition, generating a single positive pulse.

```
    always @(posedge clk) begin
        keys_prev <= keys_sync;
        keys_sync <= keys_in;
        if (!keys_sync[0] && keys_prev[0]) pulse_out[0] <= 1'b1;
        else pulse_out[0] <= 1'b0;
    end
```

Listing 3.12: Edge Detection Logic

Frequency Memory and Display

The `frequency_memory` module maintains the state of the current frequency and scale. It applies saturation logic to prevent the frequency from exceeding the operational range (0-8192 Hz) or underflowing.

For visualization, the `display_control` module converts the binary frequency value into Binary-Coded Decimal (BCD) format to drive the 7-segment displays. This is achieved using the Double Dabble algorithm. A state machine coordinates the iterative shifting and "add-3" operations required by the algorithm.

```
    always @(posedge clk) begin
        if (reset) begin
            state <= S_IDLE;
            shift_counter <= 0;
            bcd_output <= 0;
        end else begin
            case (state)
                S_IDLE: begin
                    shift_reg <= {16'b0, frequency_in};
                    shift_counter <= W;
                    state <= S_CONVERT;
                end

                S_CONVERT: begin
                    shift_reg <= {bcd_dig3_c, bcd_dig2_c, bcd_dig1_c,
                        bcd_dig0_c, shift_reg[W-1:0]} << 1;

                    if (shift_counter == 1) begin
                        state <= S_UPDATE;
                    end else begin
                        shift_counter <= shift_counter - 1;
                    end
                end

                S_UPDATE: begin
                    bcd_output <= shift_reg[W+15:W];
```

```
                    state <= S_IDLE;
                end
            endcase
        end
    end
```

Listing 3.13: Double Dabble State Machine for BCD Conversion

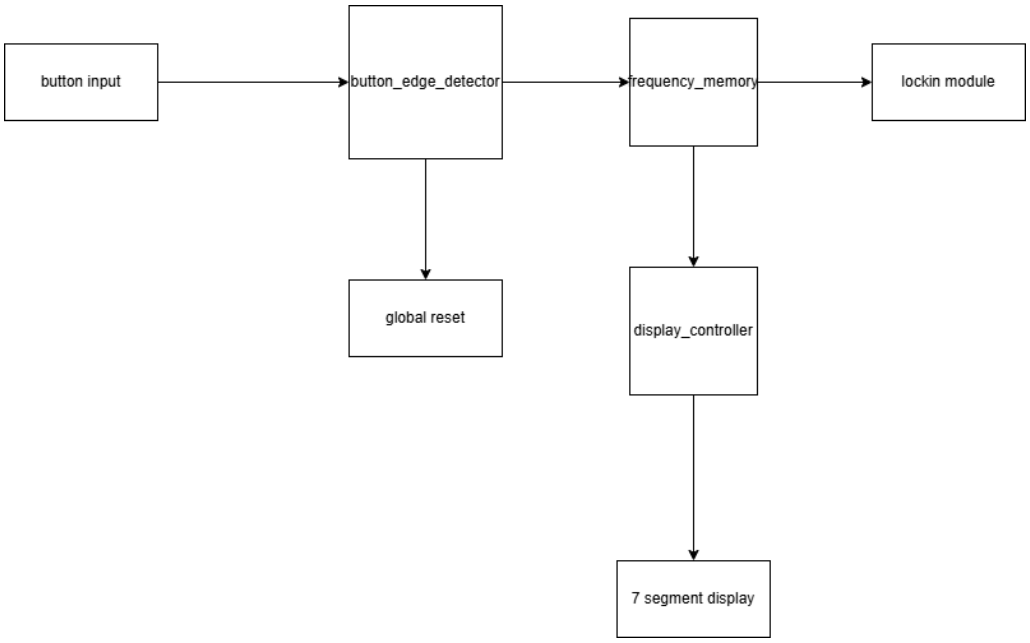

Figure 3.12: Block structure of the frequency controller subsystem, highlighting the data flow from button inputs to the 7-segment display output.

3.4.3 Demodulation and filtering

The demodulation process is orchestrated by the `lockin_controller` module, which acts as the central control unit synchronizing the data acquisition, signal generation, and mixing stages. The architecture is designed to process the 512-sample buffer sequentially.

Control Logic and Mixing

The controller implements a FSM that iterates through the input buffer using a `sample_counter`. The process follows a fetch-execute cycle:

1. **Synchronization (`S_FETCH` to `S_WAIT_DATA`):** For each sample index, the controller requests data from the Double Buffer and simultaneously triggers the DDFS module to generate the corresponding sine and cosine values for the current timestamp.

2. **Mixing (`S_MIX_START`):** Once both the audio sample and the reference signals are valid, the controller enables the `mixer` module.

3. **Accumulation (`S_OUTPUT`):** The results are forwarded to the CIC filter. The cycle repeats until the buffer counter reaches 511, at which point the controller returns to `S_IDLE` awaiting the next audio frame.

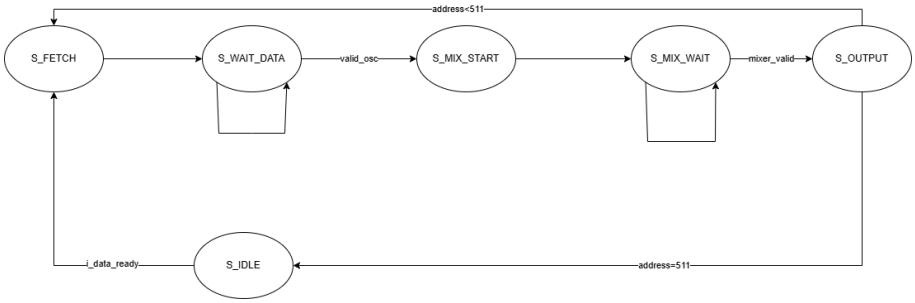

Figure 3.13: FSM of the `lockin_controller`. The FSM synchronizes the latency between the memory fetch, the DDFS generation, and the Mixer pipeline.

The `mixer` module performs the digital heterodyning. It is implemented as a 2-stage pipelined signed multiplier. A critical design decision was to avoid truncation at this stage to preserve maximum precision for the filtering process. The module multiplies the 24-bit audio input with the 18-bit reference signals, producing two 42-bit results (Phase and Quadrature) without any data loss.

CIC Filter Implementation

The Low-Pass Filter is implemented using a 2nd-order CIC architecture. As detailed in Section 2.3.2, this topology is highly efficient for FPGA implementation as it requires no hardware multipliers. The filter operates with a decimation factor $R = 512$, matching the buffer depth. This means the integrator stages function at the input sampling rate (f_s), while the comb stages and the final output update only once per buffer processing cycle ($f_s/512$).

A major implementation challenge for CIC filters is managing bit growth to prevent register overflow in the integrator stages. The required internal bit width was calculated as:

$$W_{internal} = W_{in} + N \cdot \log_2(R \cdot M) \tag{3.2}$$

With an input width $W_{in} = 42$ (from the mixer), order $N = 2$, decimation $R = 512$, and differential delay $M = 1$, the logic requires 18 bits of growth, resulting in a 60-bit internal datapath.

The following Verilog snippet highlights the parametrization of the bit growth and the conditional execution of the integrator (always active) versus the comb/decimator (active only at the end of the buffer):

```
localparam GROWTH_BITS = 2 * $clog2(BUFFER_DEPTH);
localparam INTERNAL_WIDTH = DATA_WIDTH + GROWTH_BITS;

always @(posedge clk) begin
    if (valid_in) begin
        int1_phase <= int1_phase + phase_in;
        int2_phase <= int2_phase + int1_phase;
    end

    if (valid_in && (addr_in == BUFFER_DEPTH - 1)) begin
        diff_phase_stg1 = int2_phase - comb1_phase_d;
        comb1_phase_d   <= int2_phase;

        phase_out <= diff_phase_stg2 >>> GROWTH_BITS;
        valid_out <= 1'b1;
    end
end
```

Listing 3.14: CIC Filter Bit Growth and Decimation Logic

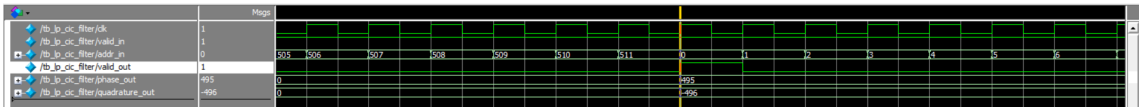

Figure 3.14: Functional simulation of the CIC Filter decimation logic. The waveform highlights the control handshake: as the input address counter (`addr_in`) completes the buffer scan (reaching index 511), the filter updates the output registers and asserts the `valid_out` signal for a single clock cycle. This mechanism effectively decimates the input rate by a factor of $R = 512$.

The output of the module is a pair of 42-bit integers, which are then passed to the CORDIC module for Cartesian-to-Polar conversion.

3.4.4 Coordinate Conversion

The final stage of the Lock-in Amplifier chain is the `cordic` module. This block accepts the 42-bit filtered I/Q components from the CIC filter and computes the magnitude and phase. The implementation relies on a sequential architecture governed by a two-state FSM, minimizing logic utilization by reusing a single adder/subtractor core over multiple clock cycles.

Input Conditioning and Quadrant Mapping

Standard CORDIC vectoring is mathematically restricted to the right half-plane (where $x \geq 0$). To support the full 360° phase range required for audio signal analysis, the implementation includes a pre-rotation stage.

In the `S_IDLE` state, the module monitors the `valid_in` signal. Upon data arrival, the logic checks the sign of the input Real component (`I_in`). If the vector lies in the second or third quadrant ($I < 0$), the coordinates are inverted to map them into the first or fourth quadrant, and a flag `is_flipped` is asserted. This transformation allows the core algorithm to always operate in a valid convergence region.

```
S_IDLE:
if (valid_in) begin
    if (I_in < 'sd0) begin
        x <= -{{GUARD_BITS{I_in[WIDTH-1]}}, I_in};
        y <= -{{GUARD_BITS{Q_in[WIDTH-1]}}, Q_in};
        is_flipped <= 1'b1;
    end else begin
        x <= {{GUARD_BITS{I_in[WIDTH-1]}}, I_in};
        y <= {{GUARD_BITS{Q_in[WIDTH-1]}}, Q_in};
        is_flipped <= 1'b0;
    end
    state <= S_RUN;
end
```

Listing 3.15: Input conditioning and quadrant mapping logic.

Iterative Vectoring Core

The core computation occurs in the `S_RUN` state. The module performs 42 iterations (`ITER`), matching the bit-width of the data to ensure maximum precision. Instead of instantiating a large ROM, the arctangent LUT is implemented as a Verilog `function` (`get_atan_val`), which the synthesizer optimizes into logic gates.

The iteration logic follows the standard shift-add structure. A dedicated counter `iter_cnt` controls the shift amount and the lookup index. To prevent arithmetic overflow during the intermediate additions, the internal registers x, y, and z are extended with 4 guard bits.

Output Reconstruction

Once the iterations complete, the result requires post-processing to restore the correct phase and manage the CORDIC gain. If the `is_flipped` flag was set during initialization, the calculated phase is shifted by

π to return the vector to its original quadrant. The code utilizes a high-precision fixed-point representation of π (PI_CONST) to ensure the transition is seamless.

Regarding the magnitude, the CORDIC algorithm introduces a systematic gain of approximately 1.647. To save FPGA resources, we opted not to implement a hardware multiplier for the inverse gain correction ($K \approx 0.607$). Instead, the output mag_out retains this scaling. Since the Lock-in Amplifier results are primarily used for relative visualization on the VGA display, this constant gain factor does not affect the qualitative analysis of the signal spectrum. Saturation logic is included to handle potential overflows before the final assignment.

```
// Fix phase:
if (is_flipped) begin
    if (z > 'sd0) begin
        phase_out <= z[WIDTH-1:0] - PI_CONST[WIDTH-1:0];
    end else begin
        phase_out <= z[WIDTH-1:0] + PI_CONST[WIDTH-1:0];
    end
end else begin
    phase_out <= z[WIDTH-1:0];
end
// Fix mag:
if (x[EXT_WIDTH-1:WIDTH-1] != 'sd0 && x[EXT_WIDTH-1:WIDTH-1] != -1)
     mag_out <= {1'b0, {(WIDTH-1){1'b1}}};
else
     mag_out <= x[WIDTH-1:0];
```

Listing 3.16: Phase reconstruction and output saturation.

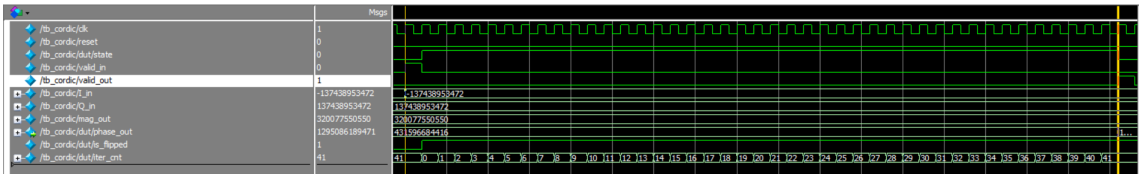

Figure 3.15: ModelSim functional simulation of the CORDIC core. The waveform captures the processing of an input vector situated in the second quadrant ($I < 0, Q > 0$). The vertical cursors highlight the computation latency: triggered by valid_in, the FSM transitions to S_RUN and asserts the is_flipped flag to map the vector into the convergence region. After 42 iterations (visible via iter_cnt), the valid_out signal is asserted, presenting the final calculated Magnitude and Phase.

3.5 VGA Visualization

The visualization subsystem is the final stage of the processing chain, responsible for converting the digital results from the FFT and Lock-in modules into a readable, while basic, analog video signal. The system drives a standard VGA monitor at a resolution of 640×480 pixels with a refresh rate of 60 Hz.

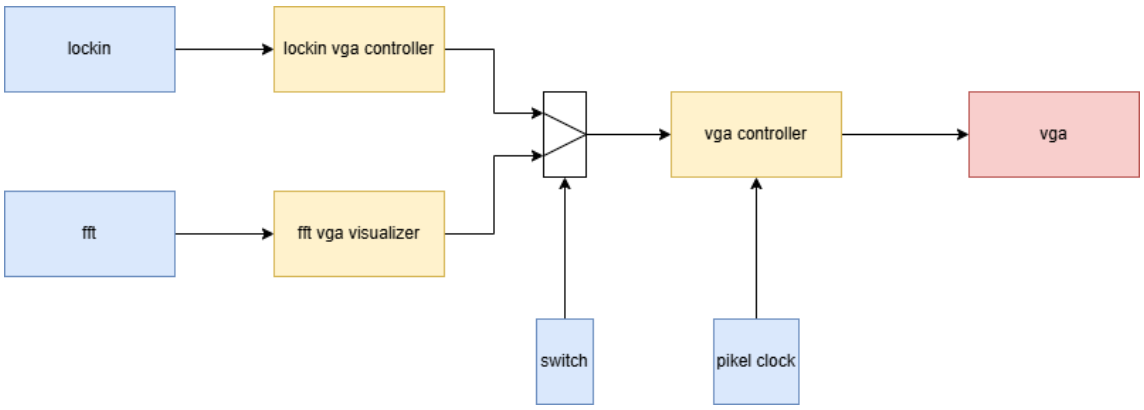

Figure 3.16: The architecture is hierarchical: a top-level module (`vga_top_system`) multiplexes the RGB data from the two processing cores based on a user switch, while a common timing controller generates the synchronization signals required by the ADV7123 Video DAC.

3.5.1 VGA Timing Controller

The `vga_controller` module acts as the timebase for the video subsystem. As discussed in Section 2.4, the VGA standard requires strict adherence to timing intervals (Active Video, Front Porch, Sync Pulse, Back Porch).

Given the standard industry requirements for 640×480 @ 60 Hz, the module is driven by a 25 MHz clock signal (`pixel_clk`), which is derived from the system's 50 MHz oscillator via a frequency divider (clock division by 2). The controller employs two counters:

- **Horizontal Counter (`h_count`):** Counts from 0 to 799 (800 pixel clocks per line).
- **Vertical Counter (`v_count`):** Counts from 0 to 524 (525 lines per frame).

The synchronization signals `h_sync` and `v_sync` are generated as active-low pulses when the counters enter the specific synchronization windows.

```
always @(posedge pixel_clk) begin
    if (reset) begin
        h_count <= 0;
        v_count <= 0;
    end else begin
        if (h_count < H_TOTAL - 1)
            h_count <= h_count + 1;
        else begin
            h_count <= 0;
            if (v_count < V_TOTAL - 1)
                v_count <= v_count + 1;
            else
                v_count <= 0;
        end
    end
end
```

Listing 3.17: VGA Counter Logic

Crucially, the module exports a `frame_over` signal, asserted at the start of the vertical blanking interval. This signal serves as a "tick" for the visualizers to swap memory buffers, ensuring that data updates do not occur during the active display time, which would cause visual problems known as "screen tearing."

3.5.2 FFT and Lock-in Visualizers

Both visualizers face a common challenge: the Clock Domain Crossing. The processing cores operate at the system frequency (50 MHz), while the VGA display consumes data at the pixel rate (25 MHz).

To solve this, we implemented a Double Buffer scheme. However, the signal that triggers the buffer swap (`frame_over`) originates in the pixel clock domain. To safely transmit this event to the writing logic (system clock domain) without metastability, we implemented an optimized 2-stage synchronizer. The `active_read_bank` toggles on V-Sync and is passed through two flip-flops clocked by the system clock. The inverted output of the second stage selects the write bank.

```
always @(posedge pixel_clk) begin
    if (i_frame_over) begin
        active_read_bank <= ~active_read_bank;
    end
end

always @(posedge clk) begin
    active_read_bank_sync1 <= active_read_bank;
    active_read_bank_sync2 <= active_read_bank_sync1;
end

assign write_bank_select = ~active_read_bank_sync2;
```

Listing 3.18: Optimized 2-Stage Synchronizer for CDC

On every `frame_over` pulse (V-Sync), the banks are swapped. This ensures the VGA controller always reads a stable, complete frame of data.

FFT Visualization

The `fft_vga_visualizer` maps the 512 frequency bins computed by the FFT module to the screen. The magnitude values are scaled to fit the vertical resolution (480 pixels). To optimize visibility, we apply a bit-shift scaling factor (>> 10) and saturate the result to prevent integer wrapping, which would render high-amplitude signals as low bars.

```
wire [23:0] shifted_mag = i_fft_mag >> MAG_SCALE_SHIFT;
wire [8:0] ram_data_in = (shifted_mag > SCREEN_HEIGHT) ? 9'd480 : shifted_mag
    [8:0];
```

Listing 3.19: Magnitude scaling and saturation logic

The memory read operation introduces a latency of 1 clock cycle. To align the pixel coordinates with the data arriving from RAM, the control signals (`pixel_y`, `video_on`) are passed through a pipeline delay register.

```
always @(posedge pixel_clk) begin
    pixel_y_d1 <= pixel_y;
    video_on_d1 <= video_on;
    pixel_y_d2 <= pixel_y_d1;
    video_on_d2 <= video_on_d1;
end
```

Listing 3.20: Pipeline alignment in FFT Visualizer

The output is rendered as a bar graph: if the current vertical pixel index `pixel_y_d2` is greater than `SCREEN_HEIGHT - bar_height`, the pixel is colored blue; otherwise, it is white.

Lock-in Visualization

The `lockin_vga_visualizer` employs a split-screen layout to simultaneously display the Magnitude (A) and Phase (ϕ) of the demodulated signal.

- **Left Half ($0 \leq x < 320$):** Displays the Magnitude.
- **Right Half ($320 \leq x < 640$):** Displays the Phase.

Unlike the FFT, which displays a snapshot of the spectrum, the Lock-in visualizer functions as an oscilloscope, showing the history of the signal over time. This is implemented using a circular buffer pointer (`write_ptr`) that increments with every new valid data sample.

```

always @(posedge clk) begin
    if (i_valid) begin
        if (write_ptr == 319)
            write_ptr <= 0;
        else
            write_ptr <= write_ptr + 1;
    end
end
```

Listing 3.21: Circular write pointer update logic

Phase data requires special handling for visualization. The signed 42-bit phase output is shifted and offset to center the zero-crossing at the middle of the right panel ($y = 240$).

```
wire signed [CORDIC_WIDTH-1:0] phs_shifted = i_phase >>> PHS_SHIFT;
wire signed [10:0] phs_calc = 11'sd240 - phs_shifted[10:0];
```

Listing 3.22: Phase data shifting and screen centering

Character Rendering: Character reading can be implemented in different ways: the first being calling a ROM for a full font set and the second being a lightweight character generation logic function. We implemented the second as it is lighter on the resource utilization and as didactical exercise. This function `is_char_pixel` calculates the bounding boxes for the characters 'A' and 'Φ' using coordinate comparators, saving memory resources.

```
    // Symbol Generation (A and Phi)
    function is_char_pixel;
        input [9:0] px, py;   // current pixel coordinates
        input [9:0] ox, oy;   // origin of the character
        input [1:0] char_sel; // 0 = 'A', 1 = 'Phi'
        reg [9:0] dx, dy;
        begin
            // Calculate relative coordinates
            dx = px - ox;
            dy = py - oy;
            is_char_pixel = 0; // Default: pixel off

            // Check bounding box
            if (dx < 8 && dy < 10) begin
                case(char_sel)
                    0: begin // Character 'A'
                       // Logic for horizontal bar, top, and sides
                       if (dy==4 || (dx==0 && dy>1) || (dx==7 && dy>1)
                           || (dy==0 && dx>0 && dx<7))
                           is_char_pixel = 1;
                    end
                    1: begin // Character 'Phi'
                       // Logic for vertical line and circle approximation
                       if (dx == 3 || dx == 4)
                           is_char_pixel = 1;
                       else if ((dy == 1 || dy == 8) && (dx > 1 && dx < 6))
                           is_char_pixel = 1;
                       else if ((dx == 0 || dx == 7) && (dy > 2 && dy < 7))
                           is_char_pixel = 1;
                    end
                endcase
            end
        end
```

```
34      endfunction
```

Listing 3.23: Algorithmic character generation using a Verilog function.

3.5.3 Top-Level Integration and Hardware Interface

The `vga_top_system` module serves as the final integration stage. It instantiates the timing controller and both visualizers, functioning as a hardware multiplexer that routes the RGB data from the active core (FFT or Lock-in) to the physical output pins based on the user's selection (`i_switch_mode`).

The signals are then routed to the Analog Devices ADV7123 DAC. Two hardware-specific constraints were addressed in this final assignment stage:

1. **Clock Inversion:** The `o_pixel_clk` sent to the DAC is inverted relative to the internal logic. This ensures that the setup and hold times of the DAC are met, as the data is stable on the rising edge of the DAC's clock (falling edge of the FPGA's clock).

2. **Sync-on-Green:** The DE2 board requires `o_vga_sync_n` to be driven high to disable the legacy Sync-on-Green feature, preventing green tint overlays on the monitor.

```
1  assign o_vga_r = (i_switch_mode) ? fft_r : lockin_r;
2  assign o_vga_g = (i_switch_mode) ? fft_g : lockin_g;
3  assign o_vga_b = (i_switch_mode) ? fft_b : lockin_b;
4
5  assign o_vga_blank_n = video_on;
6
7  assign o_vga_sync_n  = 1'b1;
8
9  assign o_pixel_clk   = ~pixel_clk;
```

Listing 3.24: Top-level Multiplexer and DAC Signal Assignments

Chapter 4

Experimental Results and Analysis

4.1 Hardware Resource Utilization

The complete system was synthesized and implemented on the Altera Cyclone II EP2C35F672C6 FPGA using Quartus II. The resource utilization analysis is critical to verify that the design fits within the device constraints and to evaluate the efficiency of the architectural choices, particularly regarding the trade-off between logic elements (LEs) and dedicated hardware blocks (Multipliers and Memory).

4.1.1 Global Resource Usage

Table 4.1 summarizes the total resource consumption of the top-level design. The system utilizes approximately 13% of the available Logic Elements and 54% of the embedded 9-bit multiplier elements.

Resource Type	Used	Total Available	Utilization (%)
Total Logic Elements	4,473	33,216	13.5
Total Registers	1,803	33,216	5.4
Total Memory Bits	107,520	483,840	22.0
M4K RAM Blocks	28	105	27.0
Embedded Multiplier 9-bit elements	38	70	54.3
PLLs	1	4	25.0

Table 4.1: Global Resource Utilization Summary

The design is comfortably within the logic and memory limits of the FPGA, leaving significant room for future logic expansions. However, the embedded multiplier usage is the dominant constraint, occupying over half of the available DSP resources. This is consistent with the nature of DSP applications, where arithmetic intensity is high.

4.1.2 Module-Level Breakdown

To better understand the distribution of resources, Table 4.2 details the consumption for the primary processing modules. The data is extracted from the synthesis hierarchy report.

Module Entity	Logic Cells	Registers	Memory Bits	DSP Elements (9-bit)
FFT Subsystem (`fft_top`)	**1,366**	**431**	**36,864**	**30**
Butterfly Unit	743	243	0	28
Controller	388	76	0	2
RAM	0	0	24,576	0
Lock-in Amplifier (`lockin_math_top`)	**2,630**	**1,147**	**18,432**	**8**
CORDIC Core	1,294	170	0	0
CIC Filter	726	565	0	0
Mixer	162	86	0	8
VGA System (`vga_top_system`)	**243**	**90**	**27,648**	**0**
Audio Interface (`top_audio`)	**190**	**134**	**24,576**	**0**

Table 4.2: Resource Utilization by Processing Module

Analysis of Resource Distribution

FFT Module vs. DSP Usage The FFT module is the primary consumer of embedded multipliers (30 out of 38 used). This is driven by the `fft_butterfly` unit, which performs complex multiplications. Since the Cyclone II multipliers are 18×18 bits (configurable as two 9×9), and the architecture uses 24-bit fixed-point precision, the synthesis tool must cascade multiple DSP blocks to achieve the required bit width without overflow. This explains why a single butterfly unit consumes a substantial portion of the DSP budget.

Lock-in Amplifier vs. Logic Usage Conversely, the Lock-in Amplifier exhibits the highest LEs consumption (2,630 LEs), nearly double that of the FFT. This is primarily due to the CORDIC algorithm (1,294 LEs) and the CIC Filter (726 LEs).

- The CORDIC algorithm is iterative and relies purely on shift-and-add operations to compute magnitude and phase. It does not use hardware multipliers, shifting the burden entirely to the FPGA's LUTs.
- The Mixer is the only component in the Lock-in chain that requires multiplication, utilizing 8 DSP elements for mixing the input signal with the reference sine/cosine waves.

Memory Distribution The memory (M4K blocks) is evenly distributed across the system to support the data buffering strategy described in Chapter 3:

- **Audio Interface:** Uses 24,576 bits for the input Double Buffer, ensuring no sample loss during processing.
- **FFT:** Uses 36,864 bits, split between the internal working RAM and the Twiddle Factor ROM.
- **VGA:** Uses 27,648 bits for the display buffers, allowing the visualization of both FFT and Lock-in data without screen tearing.

In conclusion, the resource analysis confirms the architectural decisions: the FFT exploits the dedicated DSP blocks for speed, while the Lock-in utilizes the general-purpose logic for the iterative CORDIC calculation, achieving a balanced usage of the FPGA resources.

4.2 Frequency Response and Accuracy Analysis

4.2.1 VGA Dynamic Range and Noise Floor

The visualization subsystem was tested using a basic cell phone earphone. Given the naturally lower signal amplitude of microphone inputs compared to line-level sources, the system relies on the internal gain of the WM8731 codec and the digital scaling within the FPGA.

The scaling factors in the visualizers were optimized to display a a completely flat response (zero magnitude) in a silent environment and a hard to reach saturation at 480 pixels. This indicates that the DC bias removal and the rounding logic in the Butterfly unit successfully eliminate background noise and quantization artifacts, preventing false positives on the display.

4.2.2 Lock-in Amplifier Accuracy and DDFS Scaling

To verify the frequency accuracy of the Lock-in Amplifier, we performed a sweep of measurements comparing the internal set frequency with the actual generated frequency. We observed a systematic deviation where the real frequency was consistently lower than the set frequency. Table 4.3 presents the experimental data.

Set Frequency (Hz)	Real Frequency (Hz)	Ratio (Real/Set)
500	366	0.732
1000	732	0.732
2000	1464	0.732
3000	2196	0.732
4000	2928	0.732
5000	3660	0.732
6000	4392	0.732
7000	5124	0.732
8000	5856	0.732

Table 4.3: Comparison between Set Frequency (Lock-in) and Real Frequency

The measured ratio is consistently ≈ 0.732. The values are rounded to the nearest integer because the Lock-in module's sensitivity is limited to 1 Hz steps over a range of 8192, making higher decimal precision impossible to display.

Analysis of the Scaling Factor

The factor 0.732 is identified as a normalization mismatch in the DDFS logic. The DDFS utilizes a 16-bit phase accumulator ($2^{16} = 65536$ steps) to scan the Sine LUT. However, the system operates at a nominal sampling rate of $F_s = 48$ kHz.

The frequency control logic mapped the target frequency directly to the phase increment value, implicitly assuming that the accumulator range matched the sampling rate. The resulting scaling factor is derived mathematically as:

$$\text{Ratio} = \frac{F_s}{2^{16}} = \frac{48000}{65536} \approx 0.73242 \tag{4.1}$$

This exact match confirms that the discrepancy is purely algorithmic and not due to hardware clock failure.

To further validate that the hardware clock was operating correctly (and thus that the error was isolated to the DDFS logic), we can analyze the PLL configuration used to generate the MCLK. The WM8731 requires 18.432 MHz to achieve 48 kHz sampling. The FPGA synthesizes this from the 50 MHz system clock using a fractional approximation ($M = 7, D = 19$ ratio):

$$F_{MCLK} = 50\,\text{MHz} \times \frac{7}{19} \approx 18.421\,\text{MHz} \tag{4.2}$$

The deviation from the ideal 18.432 MHz is only 0.06%. This negligible hardware error confirms that the codec was indeed sampling at approximately 48 kHz, validating the conclusion that the 0.732 factor is strictly a result of the 48000/65536 normalization coefficient. Another, weak but nonetheless relevant, proof is a frequency sweep in the FFT Mode where at 24kHz the Nyquist frequency was reach as predicted by FFT theory.

4.3 VGA Output Visualization

The VGA output screen serves for a qualitative analysis. The following subsections present the visual results captured directly from the monitor.

4.3.1 FFT Spectrum Analysis

In the FFT mode, the system displays the magnitude of the 512 frequency bins computed by the hardware. The horizontal axis represents the frequency index (0 to 511), while the vertical axis represents the magnitude.

Figure 4.1 shows the response to a 1 kHz sinusoidal input. Two distinct peaks are visible. Since the input signal is real-valued, the FFT output exhibits conjugate symmetry around the Nyquist frequency ($N/2$). The left peak represents the positive frequency component, while the right peak represents the negative frequency component (aliased to $N/2 \ldots N-1$). This visualization confirms the correctness of the butterfly address generation and the bit-reversal permutation logic.

Figure 4.1: FFT visualization of a 1 kHz sine wave. The display correctly shows the conjugate symmetry characteristic of the FFT for real-valued inputs.

Figure 4.2 shows the spectrum visualized in a noisy environment. Again, the output is symmetrical to the Nyquist frequency.

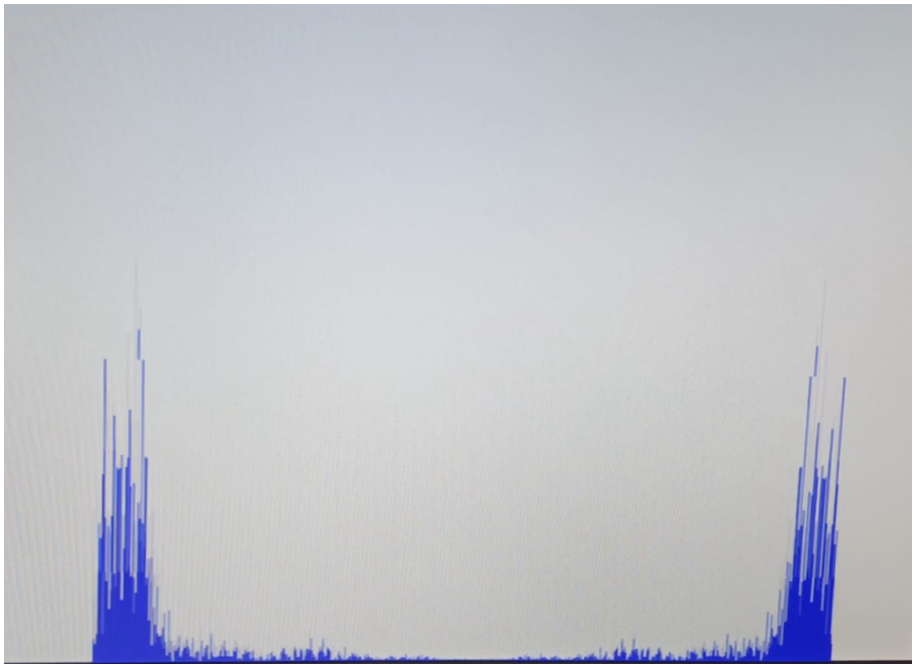

Figure 4.2: FFT visualization in a noisy enviornment.

4.3.2 Lock-in Amplifier Visualization

The Lock-in visualizer functions as a digital oscilloscope, plotting the history of the demodulated signal over time. The screen is split into two sections: the left panel displays the Magnitude (A), and the right panel displays the Phase (Φ).

Figure 4.3 presents the output when the system is locked onto a test signal.

- **Magnitude (Left):** The magnitude plot appears as a flat, stable horizontal line. This indicates that the system successfully extracted the constant DC component from the mixer output, validating the stability of the CIC Low-Pass Filter.
- **Phase (Right):** Ideally, if the reference frequency matches the signal frequency perfectly, the phase difference should be a constant DC value (a straight horizontal line). However, the captured image shows a drifting sawtooth pattern.

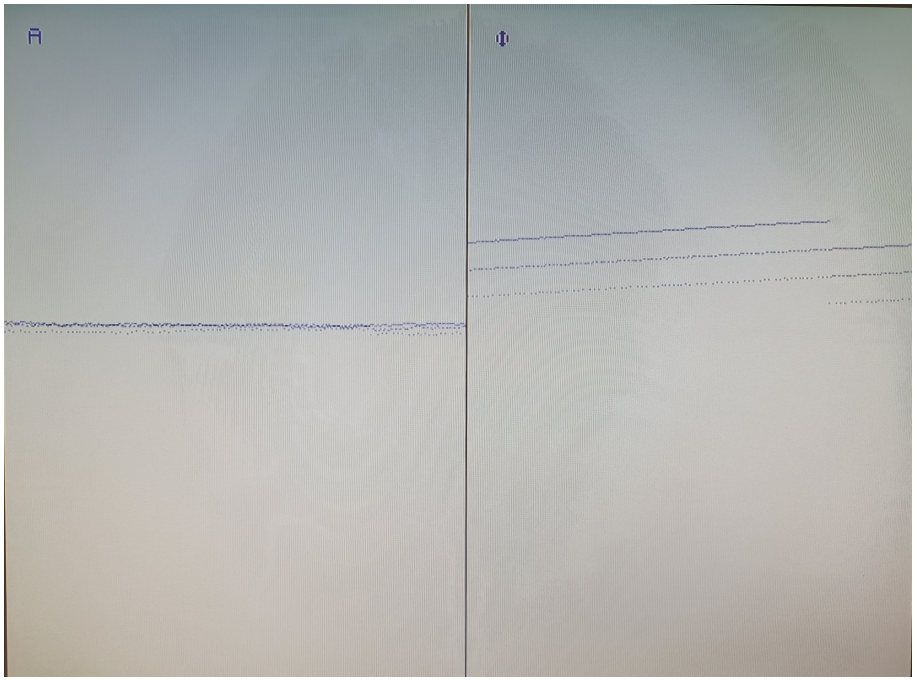

Figure 4.3: Lock-in Amplifier output. Left: Stable Magnitude. Right: Phase drift caused by the DDFS frequency scaling mismatch.

This behavior is a direct visual confirmation of thefrequency scaling error analyzed in Section 4.2.2. Although the user input was set to 1000 Hz, the internal DDFS generated a reference of approximately 732 Hz due to the normalization mismatch. Even when manually adjusting the source to approach the internal frequency, a residual frequency difference ($\Delta\omega = \omega_{sig} - \omega_{ref}$) remains. Mathematically, a constant frequency difference results in a linear phase accumulation over time:

$$\phi(t) = (\omega_{sig} - \omega_{ref})t + \phi_0 \tag{4.3}$$

The visualizer plots this linearly increasing value. When the phase exceeds π (180°), it wraps around to $-\pi$, creating the distinct sawtooth "lines" visible on the right side of the screen.

4.4 Timing Closure and Critical Paths

The Static Timing Analysis was performed to verify that the design meets the required operating frequency of 50 MHz. The timing constraints were defined in an SDC file, specifying the main system clock period:

```
create_clock -name {clk_50} -period 20.000 [get_ports {CLOCK_50}]
derive_pll_clocks
derive_clock_uncertainty
```

Listing 4.1: SDC Timing Constraints

4.4.1 Setup Time and Maximum Frequency

The analysis confirms that the design achieves Timing Closure with a positive slack. The reported Maximum Frequency (F_{max}) is 71.24 MHz, which provides a safety margin of ≈42% over the required 50 MHz.

Table 4.4 summarizes the worst-case timing metrics from the Slow Model (setup) and Fast Model (hold).

Metric	Value
Target Clock Frequency	50.00 MHz
Achieved F_{max}	71.24 MHz
Worst-Case Setup Slack	5.963 ns
Worst-Case Hold Slack	0.215 ns

Table 4.4: TimeQuest Timing Analysis Summary

4.4.2 Critical Path Analysis (Setup)

The critical path determines the maximum speed of the system. The Setup analysis reveals that the longest path (worst slack of 5.963 ns) occurs within the FFT subsystem.

Specifically, the path originates in the `fft_controller` (register `stage_reg[1]`) and terminates at the `fft_working_ram` address port (`portb_address_reg`). This path corresponds to the address generation logic for the Butterfly operations. Since the FFT calculates memory addresses dynamically based on the current stage and group index (using shifters and adders described in Chapter 3), the signal must propagate through the combinatorial logic of the controller before reaching the synchronous RAM block inputs.

Despite being the critical path, the substantial positive slack indicates that the pipeline depth chosen for the FFT controller is sufficient.

4.4.3 Hold Time Analysis

The analysis shows a worst-case Hold Slack of 0.215 ns, located in the `display_control` module.

The critical hold paths are found in the `shift_counter` registers. As seen in Listing 3.13 for the Double Dabble algorithm:

```
65 if (shift_counter == 1) begin
66     state <= S_UPDATE;
67 end else begin
68     shift_counter <= shift_counter - 1;
69 end
```

This is a local feedback loop where the register feeds directly back into itself through a subtractor. Because the source and destination are physically close (low clock skew) and the delay of a decrementer is minimal, the data arrives very fast causing the smallest Slack possible.

Chapter 5

Conclusion

Designed primarily for educational purposes, this project demonstrates the feasibility of hardware-implemented frequency analysis on the Altera DE2 board. The system successfully implements the intended functions, producing a clear qualitative estimation of the spectral composition of audio signals captured via the microphone.

However, some limitations remain due to design constraints and the available testing time. Specifically, the visualization interface does not currently display physical units of measurement. Regarding the Lock-in Amplifier, as discussed in the experimental analysis, there is a minor discrepancy between the requested and generated frequencies, as well as a settling latency—a trade-off accepted to maximize noise rejection, which could be minimized with further tuning. Additionally, the FFT visualizer currently displays the full conjugate symmetry (mirroring) of the spectrum; while this was valuable for verifying the correct operation of the internal sub-modules, it limits the resolution of the useful frequency band.

In conclusion, while these factors currently prevent the use of the module for strict quantitative analysis, the project successfully validates the architectural choices and the processing power of the FPGA for real-time audio applications.

Bibliography

[1] Terasic Technologies, "Altera DE2 Development and Education Board User Manual".

[2] P. Nannipieri, "Introduction to VGA & I2C".

[3] ebyteiot.com "I2S Timing Diagram Picture"

[4] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series".

[5] Jarmo Takala and Konsta Punkka, "Butterfly unit supporting radix-4 and radix-2 FFT".

[6] A. V. Oppenheim and R. W. Schafer, "Discrete-Time Signal Processing".

[7] R. G. Lyons, "Understanding Digital Signal Processing", section 13.2..

[8] Robert H. Dicke, The Measurement of Thermal Radiation at Microwave Frequencies

[9] J. E. Volder, "The CORDIC Trigonometric Computing Technique"

[10] M.P. Donadio "CIC Filter Introduction"

[11] T.Abiseha Aruna and V.Bhanumathi, "Double Dabble"