"DESIGN AND IMPLEMENTATION OF A BFSK TRANSCEIVER USING BANDPASS FILTER ARCHITECTURE ON FPGA"

A Report

By

YASH RAJ DASH

(REGD NO.: 122EC0126, NIT ROURKELA)

Submitted after completion of the two-month summer internship under **DRDO (**Defence Research and Development Organisation).

Under the Guidance of

A K SRIVASTAVA

Department of Tele-Command
Integrated Test range

Chandipur-756025, INDIA

01-05-2025 to 30-06-2025

FPGA Architecture and Its Application in BFSK Transceiver Design

3.1 Introduction to FPGA: Types, Applications & Advantages

Field-Programmable Gate Arrays (FPGAs) are integrated circuits that can be configured or programmed by users after manufacturing. They provide a flexible hardware platform that can be customized for various applications without requiring extensive changes to the underlying circuitry.

The need for FPGAs arises from the demand for hardware solutions that can be reconfigured or adapted to specific requirements. Unlike Application-Specific Integrated **Circuits** (ASICs), which are designed for a specific purpose and cannot be changed once manufactured, FPGAs offer the advantage of reprogrammability. This flexibility makes them suitable for a wide range of applications where fast prototyping, rapid development, and customization are essential.

FPGAs are categorized based on their architecture, functionality, and performance characteristics. Some common types include:

1. **SRAM-based FPGAs**: These FPGAs use static random-access memory (SRAM) to store configuration data. They are highly flexible and can be reprogrammed multiple times. However, they require constant power to retain their configuration, making them less suitable for power-constrained applications.
2. **Flash-based FPGAs**: These FPGAs store their configuration data in non-volatile flash memory. They retain configuration even when power is removed, making them suitable for applications where power consumption is a concern. However, they typically offer lower performance compared to SRAM-based FPGAs.
3. **Antifuse-based FPGAs**: These FPGAs use antifuse technology, where initially unconnected circuit elements are permanently fused together during programming. Antifuse-based FPGAs offer high performance, low power consumption, and high reliability, but they are typically more expensive and are one-time programmable.

FPGAs find applications in various fields, including:

1. **Digital Signal Processing (DSP)**: FPGAs can efficiently implement complex signal processing algorithms. They are commonly used in applications such as audio/video processing, image recognition, and wireless communication.
2. **Embedded Systems**: FPGAs provide a flexible platform for developing and prototyping embedded systems. They can be used to implement custom interfaces, integrate multiple peripherals, and accelerate specific functions.
3. **High-Performance Computing (HPC)**: FPGAs are increasingly used in HPC environments to accelerate computationally intensive tasks. They can act as co-

processors to offload specific computations, achieving significant speedups compared to traditional processors.

Advantages of FPGAs:

1. **Flexibility**: FPGAs allow hardware designs to be adapted and reconfigured as requirements evolve. This enables rapid prototyping, iterative development, and easy updates.
2. **High Performance**: Due to their parallelism and dedicated hardware resources, FPGAs offer high-speed processing. Designs can be optimized for specific tasks to ensure efficient and fast execution.
3. **Customization**: Designers can tailor FPGA-based hardware to meet specific application needs, resulting in improved performance and power efficiency compared to general-purpose processors.
4. **Reduced Time-to-Market**: FPGAs help shorten development cycles by allowing immediate implementation and testing of designs, avoiding the delays associated with ASIC manufacturing.
5. **Cost-Effectiveness**: FPGAs eliminate the high non-recurring engineering (NRE) costs associated with ASICs, making them an economical option for low-to-medium volume production.

3.2 Architecture of FPGA

The architecture of an FPGA consists of several key components, including Configurable Logic Blocks (CLBs), interconnects, I/O pads, and a switching matrix. Below is a brief explanation of each component:

1. Configurable Logic Blocks (CLBs):

CLBs are the fundamental building blocks of an FPGA. They consist of a collection of Look-Up Tables (LUTs), registers, multiplexers, and other logic elements. LUTs are programmable tables that allow users to define specific logic functions within the CLB. The outputs of the LUTs can be routed to various destinations within the FPGA fabric.

2. Interconnects:

Interconnects form the network of programmable connections within an FPGA, enabling signal flow between different components. They consist of a series of routing channels that run horizontally and vertically across the FPGA. Interconnects provide the necessary paths to connect the inputs and outputs of CLBs and other functional units such as memory blocks or DSP cores.

3. I/O Pads:

Also known as I/O cells, these pads serve as the interface between the FPGA and the external world. They handle incoming and outgoing signals and support a variety of electrical standards such as LVCMOS, LVDS, and differential signalling. I/O pads also feature voltage level translation, impedance matching, and protection circuitry to ensure signal integrity and compatibility.

4. Switching Matrix:

The switching matrix is responsible for establishing connections between input and output

signals within the FPGA. It is composed of a grid of programmable switches that can be dynamically configured to create the required signal paths. This enables routing from one CLB to another, allowing for complex digital circuit implementation.

Together, these components define the reconfigurable nature of an FPGA. CLBs provide programmable logic and storage elements, interconnects allow for signal routing, I/O pads facilitate communication with external devices, and the switching matrix makes interconnection between functional blocks possible. By configuring these components, users can define the behavior and functionality of an FPGA to meet specific application requirements.

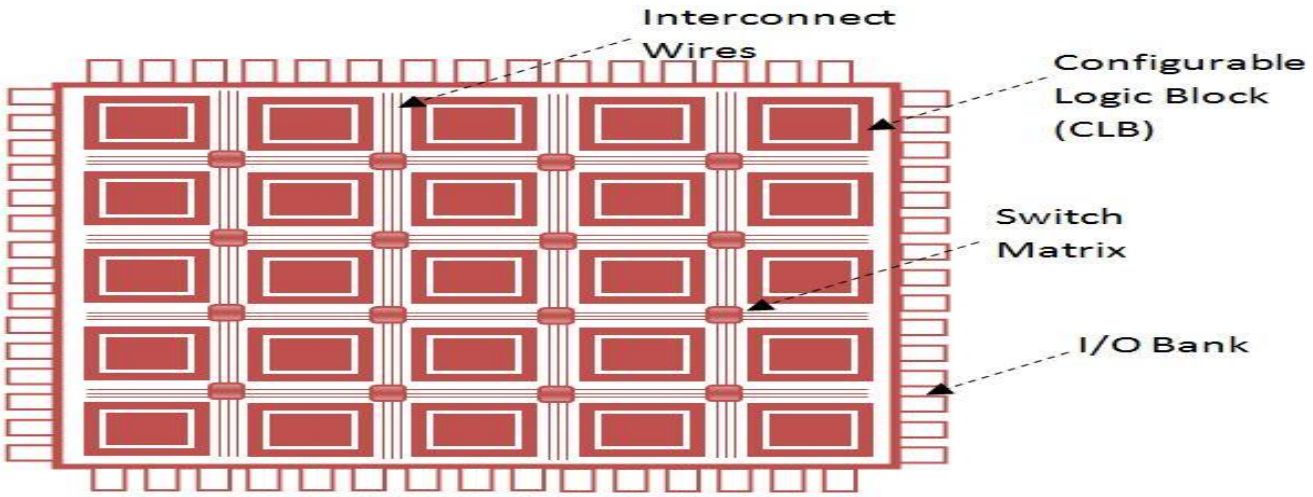

Fig 3.1: FPGA Architecture

3.3 FPGA Design Flow

The FPGA design flow refers to the sequence of steps involved in creating and implementing a design on an FPGA. While the exact steps may vary depending on specific requirements and tools, a general overview of the standard FPGA design flow is as follows:

1. Design Specification:

The first step is to define the design specifications and requirements. This includes identifying the intended functionality, performance goals, input/output interfaces, and any constraints related to power, area, or timing.

2. Design Entry:

In this phase, the design is written using a Hardware Description Language (HDL) such as **VHDL** or **Verilog**. Design entry can be done via a text editor or a graphical design entry tool like Xilinx ISE or Vivado.

3. Simulation and Verification:

The HDL design is simulated using specialized simulation tools. Testbenches are written to verify the functional behavior of the design under various conditions. Simulation helps detect and correct functional errors before moving forward.

4. Synthesis:

The HDL code is synthesized into a gate-level netlist. This step maps the logic to specific FPGA elements such as LUTs, flip-flops, and multiplexers, based on the selected device.

5. Place and Route:

This stage determines the physical placement of logic blocks on the FPGA and establishes signal routing. The placement and routing are optimized for performance, power, and signal integrity while adhering to timing constraints.

6. Timing Analysis:

Timing analysis ensures that the design meets setup and hold timing requirements. Critical paths are identified, and any violations are addressed through optimization techniques such as pipelining, retiming, or logic restructuring.

7. Bitstream Generation:

Once the design is placed, routed, and passes all checks, a **bitstream** file is generated. This file contains the configuration data required to program the FPGA.

8. FPGA Programming:

The generated bitstream is loaded onto the FPGA using a hardware programmer or via JTAG/USB interfaces. The FPGA is now configured to perform the specified functions.

9. Post-Implementation Verification:

After programming, the FPGA is tested on the actual hardware. This phase includes functional verification, debugging, and performance evaluation in a real-world environment.

10. Iteration and Debugging:

Design refinement is often required. If issues are found during post-implementation testing, designers may iterate back to earlier steps, make necessary changes, and repeat the flow as needed.

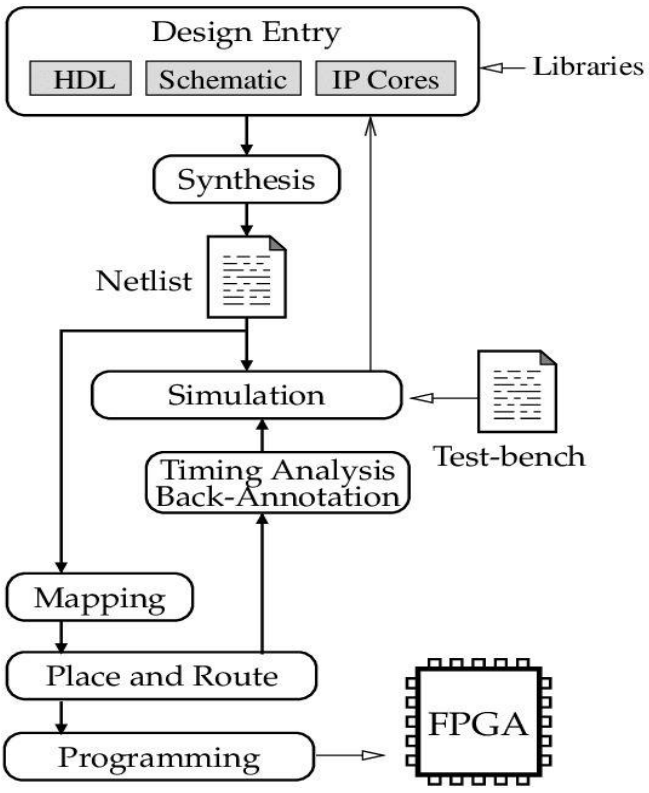

Fig 3.2: FPGA Design Flow

3.4 Virtex-4 FPGA Board: Features and Use

The Xilinx Virtex-4 FPGA (Field-Programmable Gate Array) board is a high-performance and versatile platform ideal for implementing complex digital designs, including

communication systems like the BFSK transceiver. Its flexibility, high-speed logic, and advanced DSP features make it particularly suited for signal processing tasks required in FPGA-based modulator/demodulator architectures.

Key Features of the Virtex-4 FPGA Board:

1. **FPGA Chip**:
   The heart of the board is the Xilinx Virtex-4 FPGA chip, which allows implementation of user-defined digital circuits through reconfiguration.
2. **Logic Blocks**:
   The chip contains an array of Configurable Logic Blocks (CLBs). Each CLB integrates Look-Up Tables (LUTs) for combinational logic, flip-lops for state storage, and multiplexers for routing logic—used extensively in our project for designing shift registers, comparators, and control FSMs.
3. **Block RAM (BRAM)**:
   On-chip Block RAM modules provide fast and efficient storage for temporary data buffering and lookup tables—particularly useful during implementation of the symbol decoding and filter logic in the BFSK receiver.
4. **Digital Signal Processing (DSP) Slices**:
   The Virtex-4 includes specialized DSP slices, which combine multipliers, accumulators, and arithmetic logic units. These were leveraged for real-time processing in the receiver pipeline to support operations like energy calculation and FIR filtering.
5. **Input/Output (I/O) Blocks**:
   These blocks handle data exchange with external components, such as DACs (Digital-to-Analog Converters) and test instruments. The board supports various electrical standards (LVCMOS, LVDS), ensuring compatibility with signal generator outputs and oscilloscope probes.
6. **Clock Management**:
   Advanced clock management tiles (CMTs) enable the generation of different clock frequencies from a single input clock. In our project, this feature allowed us to derive multiple domains like 40 MHz (base), 10 MHz (DDS), and 4 kHz (bit clock) from a common input.
7. **Configuration**:
   The board supports configuration via JTAG or external PROM. We used JTAG programming to repeatedly test and debug intermediate stages of the BFSK transceiver implementation.
8. **Expansion Interfaces**:
   The board includes high-speed connectivity options such as PCI, DDR, and serial interfaces. While not central to this project, they provide extensibility for future integration with real-time telemetry systems or external storage.

Application in the BFSK Transceiver Project

The Virtex-4 board served as the foundation for implementing both the transmitter and receiver sides of the BFSK system. Key modules such as:

- **Direct Digital Synthesizer (DDS)** for sine wave generation
- **Bandpass Filters (via FIR IP Core)**
- **Decision logic using comparators**
- **Timing counters and control FSMs**

were all deployed on the Virtex-4 using Verilog HDL, synthesized and tested using the Xilinx ISE Design Suite.

JTAG Chain in the Virtex-4 Board

The JTAG (Joint Test Action Group) chain is critical for both programming and in-system debugging. In our project, the JTAG chain allowed us to perform:

- Fast bitstream uploads for logic updates
- Device recognition and initialization
- In-system diagnostics during waveform testing

Key Elements:

1. **JTAG Interface**:
   Includes standard pins—TCK, TMS, TDI, and TDO—to communicate with the FPGA and any other serial devices on board.
2. **Boundary-Scan Cells**:
   Provide access to I/O pins at the signal level, useful for checking whether data transmission lines were functioning correctly during early hardware tests.
3. **Instruction Register (IR)** and **Data Register (DR)**:
   Used to specify operations (programming, testing, monitoring) and transport actual configuration data into or out of the FPGA.
4. **Device Chain**:
   Allows connection of multiple devices (e.g., configuration flash or DAC) in series for collective or individual access through the same JTAG header.
5. **TAP Controller**:
   Manages the internal JTAG state transitions and signal flow control.
6. **JTAG Configuration Tools**:
   Xilinx **iMPACT** was used to program the FPGA and validate successful bitstream transfers after each synthesis and implementation cycle.

Boundary Scan Mode

Boundary Scan Mode, supported via JTAG, provides visibility and control over individual pins of the FPGA, allowing verification and testing without the need for probes or logic analyzers.

Uses in This Project:

- Checking if DAC output pins reflected correct waveforms
- Verifying signal routing from DDS to filters and decision logic
- Testing FSM behavior at different clock domains

Advantages:

1. **Testing**: Detects open or short connections during setup.
2. **Debugging**: Enables pin-level inspection and toggling.
3. **In-System Programming**: Eliminates need for direct access to programming pins.
4. **Design Verification**: Confirms correct signal activity before actual flight hardware integration.

Design and Implementation of a BFSK Transceiver Using Bandpass Filter Architecture on FPGA

4.1 Overview of BFSK Modulation

Binary Frequency Shift Keying (BFSK) is a digital modulation technique where binary information is transmitted by shifting the frequency of a carrier signal between two predefined frequencies. Logic '0' is represented by frequency f_0, and logic '1' by frequency f_1, expressed as:

$$S(t) = \begin{cases} A\cos(2\pi f_0 t), & for\ bit\ 0 \\ A\cos(2\pi f_1 t), & for\ bit\ 1 \end{cases}$$

Due to its resilience to noise and simple implementation, BFSK is widely used in telemetry, remote control, and defense communication systems where reliable signal detection is critical. In this project, a complete BFSK transceiver is implemented on a Xilinx FPGA using Verilog HDL. The transmitter employs a Direct Digital Synthesizer (DDS) to generate sine waves corresponding to binary input, while the receiver utilizes FIR filter IP cores for frequency detection. Verilog enables precise modeling of synchronous logic and seamless integration of custom modules with IP cores, ensuring accurate modulation, filtering, and bit recovery in real-time.

4.2 System Architecture and Functional Blocks

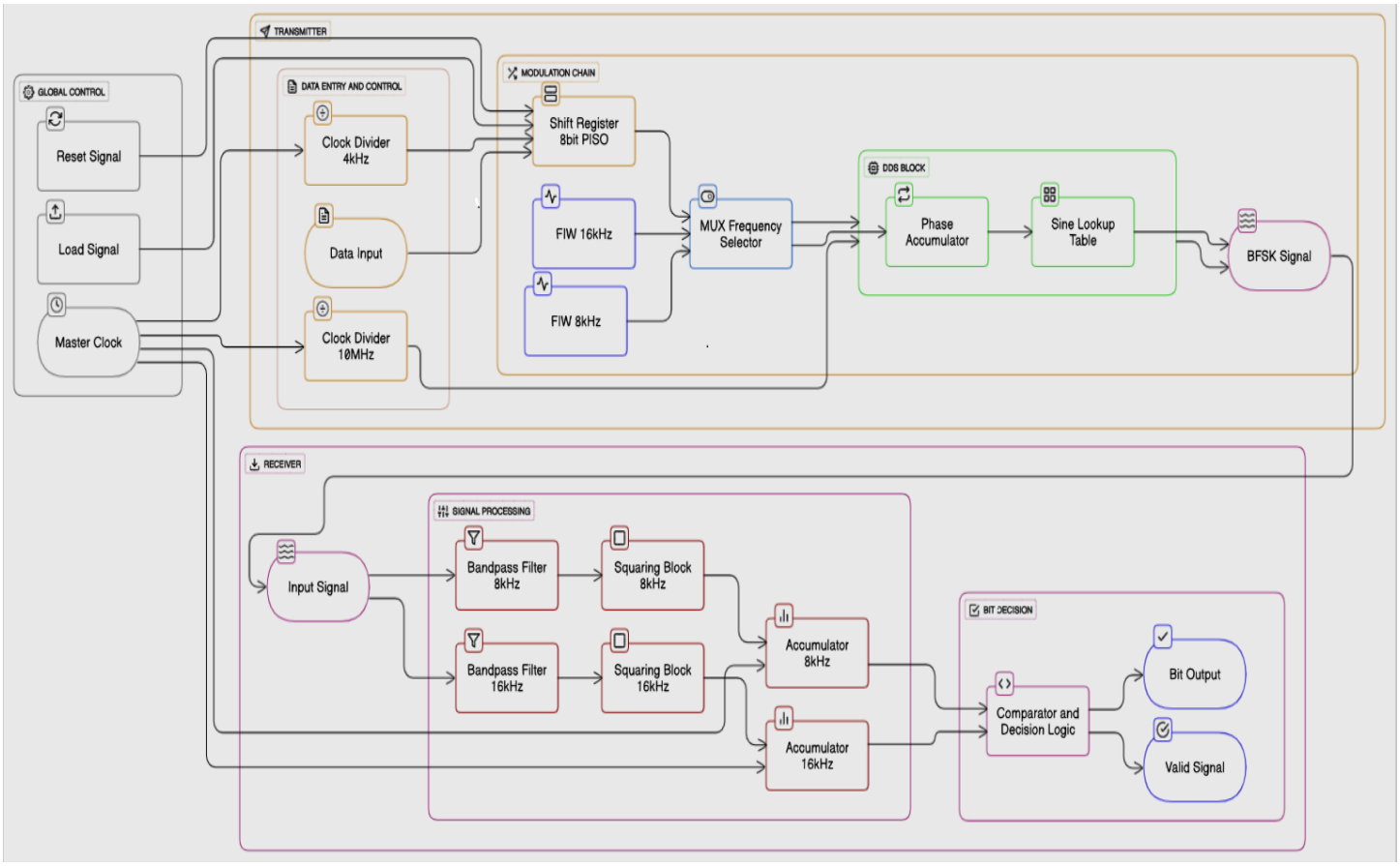

Figure 4.1: Complete BFSK Transceiver System Block Diagram

The complete transceiver architecture is shown in Figure 4.1. The transmitter section serializes an 8-bit parallel input using a shift register clocked at 4 kHz. Based on the serial bit, a

frequency-select MUX chooses the appropriate Frequency Increment Word (FIW) for the DDS core, which runs at 10 MHz. The DDS generates sine waveforms corresponding to either 8 kHz or 16 kHz depending on the bit value.

The receiver section processes the incoming sine wave through two FIR-based Bandpass Filters (BPFs), each tuned to detect one of the BFSK frequencies. The filtered outputs are squared, accumulated over one bit duration (2500 samples at 10 MHz), and compared. The result is used to determine the transmitted bit and assert a valid signal upon successful detection.

The overall system architecture of the BFSK transceiver consists of two major modules — the transmitter and the receiver — both designed using Verilog Hardware Description Language (HDL) and implemented on a Xilinx FPGA. These modules operate synchronously under a common 40 MHz master clock, which is internally divided to generate lower-frequency clocks required by various subcomponents, such as the DDS (10 MHz) and the shift register (4 kHz).

The transmitter is responsible for serializing 8-bit parallel input data and modulating it using two sinusoidal waveforms of distinct frequencies, representing binary logic '0' and '1'. The modulated signal is generated using a Direct Digital Synthesizer (DDS), driven by frequency increment words corresponding to the bit being transmitted.

The receiver demodulates the signal by passing it through two FIR bandpass filters, implemented using Xilinx FIR Compiler IP cores, each centered around one of the two BFSK frequencies. The outputs of these filters are analyzed to determine the dominant frequency in each time window, thereby recovering the original bitstream.

The design is fully modular, with each functional block described and verified independently in Verilog. Key design principles such as clock domain synchronization, parameterized module design, finite state machines (FSMs), and IP core integration are leveraged to ensure efficient hardware utilization and ease of scalability.

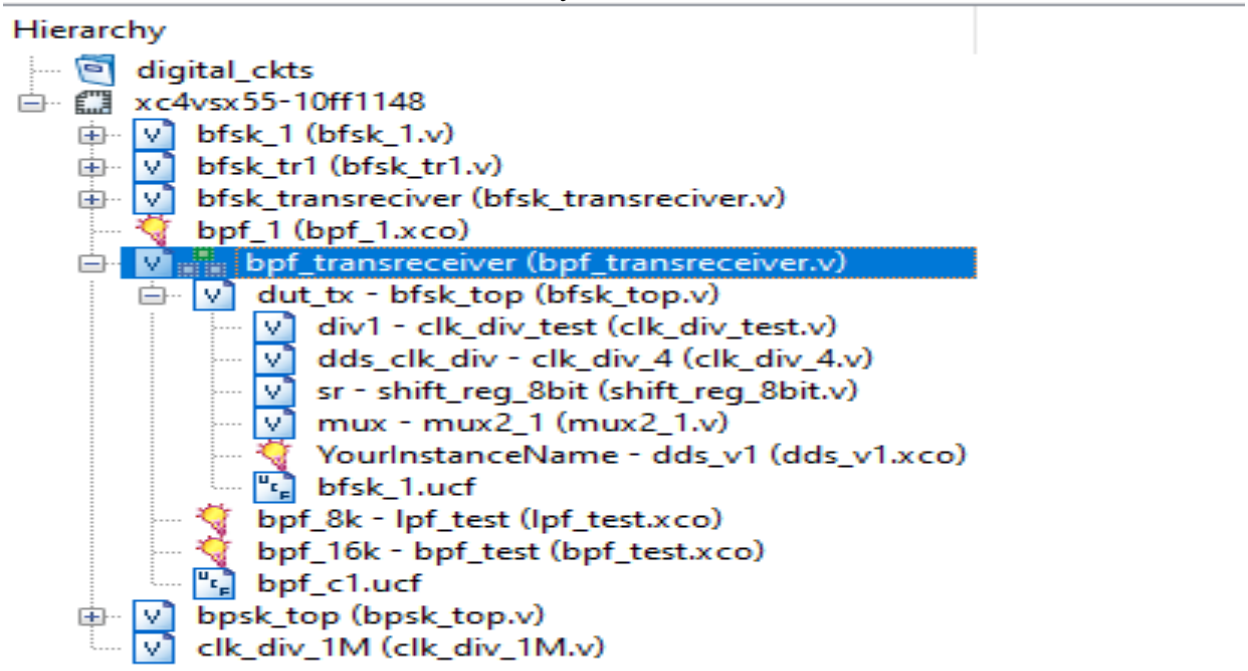

Figure 4.2: Top-Level Verilog Block Diagram of the BFSK Transceiver

Description: The figure illustrates the hierarchical organization of the BFSK transceiver system. It shows the interaction between key modules: bfsk_top(transmitter), bfsk_receiver,

and control logic. Major input signals include the 40 MHz system clock, asynchronous reset, data input, and load signal for the shift register. The output ports include the synthesized analog sine waveform (sine_out), the demodulated bitstream (bit_out), and a valid flag indicating the availability of newly decoded bits.

4.3 Design and Implementation of the Transmitter

The BFSK transmitter is responsible for generating a frequency-modulated waveform based on an 8-bit input word. Each bit is transmitted serially using one of two sine wave frequencies: 8 kHz for binary '0' and 16 kHz for binary '1'. The modulation is realized digitally using a Direct Digital Synthesizer (DDS), controlled via Frequency Increment Words (FIWs).

Key Components

- **Shift Register:** A Parallel-In Serial-Out (PISO) register implemented in Verilog that loads the 8-bit word and shifts out one bit every 250 µs (corresponding to a 4 kHz data rate).

```
21  module shift_reg_8bit(
22  input clk,
23  input reset,
24  input load,
25  //input [7:0]data_in,
26  output  bit_out
27      );
28      parameter data_in=8'b1011_1011;
29  reg [7:0]shift_reg;
30  always@(posedge clk or posedge reset)
31  begin
32  if(reset)
33  shift_reg <= 8'b0;
34  else if(load)
35  shift_reg <= data_in;
36  else
37  shift_reg <= {shift_reg[0] , shift_reg[7:1]};
38  end
39  assign bit_out= shift_reg[0];
40  endmodule
```

Figure4.3: Verilog Code for Shift register

This Verilog module implements an 8-bit right-rotating shift register that loads a fixed value (10111011) when load is high and shifts the bits on each clock cycle. The least significant bit (LSB) is continuously output as bit_out to serialize the data.

- **Clock Dividers:** Two frequency dividers generate:
  - 10 MHz for DDS sampling
  - 4 kHz for shift register operation

  Derived from a common 40 MHz system clock using synchronous counter logic.

```
21  module clk_div_4(
22  input clk_in,
23  input reset,
24  output reg clk_out
25      );
26  reg [1:0] count;
27  always@(posedge clk_in )
28  begin
29  if(reset)
30  begin
31  count <= 2'd0;
32  clk_out <= 1'b0;
33  end
34  else
35  begin
36  if (count==2'd1)
37  begin
38  clk_out <= ~clk_out;
39  count <= 2'd0;
40  end
41  else
42  begin
43  count <= count+1;
44  end
45  end
46  end
47  endmodule
```

Figure 4.4: Verilog Code for 10MHz Clock

```
21  module clk_div_test(
22  input clk_in,
23  input reset,
24  output reg clk_out
25      );
26  reg [13:0] count;
27  always@(posedge clk_in )
28  begin
29  if(reset)
30  begin
31  count <= 14'd0;
32  clk_out <= 1'b0;
33  end
34  else
35  begin
36  if (count==14'd4999)
37  begin
38  clk_out <= ~clk_out;
39  count <= 14'd0;
40  end
41  else
42  begin
43  count <= count+1;
44  end
45  end
46  end
47  endmodule
```

Figure 4.5: Verilog Code for 4KHz Clock

The clock divider modules are used to generate lower-frequency clocks from a high-frequency system clock (e.g., 40 MHz). They work by counting input clock edges and toggling the output clk_out at specific intervals:

- For general division (e.g., to 4 kHz or 10 MHz), a counter_increments on each rising clock edge.
- When the counter reaches a preset value, the output clock toggles and the counter resets.
- This method enables precise timing control for modules like shift registers (e.g., at 4 kHz) or DDS (e.g., at 10 MHz), ensuring proper synchronization in the BFSK transmitter and receiver.

Clock divider modules reduce the system clock to required frequencies using counters, enabling time-aligned operation of all functional blocks within the BFSK transceiver.

- **Frequency Controller:** Based on the current bit Multiplier, selects the corresponding FIW for 8 kHz or 16 kHz frequency synthesis.
- This Verilog module implements a 2-to-1 multiplexer that selects between two 20-bit Frequency Increment Word (FIW) inputs (fiw0 and fiw1) based on the select signal. The selected FIW is output as fiw_out and used in the DDS for BFSK modulation.

```
21  module mux2_1(
22
23  input [19:0] fiw0,
24  input [19:0] fiw1,
25  input select,
26  output reg [19:0] fiw_out
27      );
28  always@(*)
29  begin
30  if(select==1'b0)
31  fiw_out=fiw0;
32  else
33  fiw_out=fiw1;
34  end
35  endmodule
36
```

Figure 4.6: Verilog Code for Multiplier

Mathematical Equation for DDS Frequency
The output frequency of the DDS is governed by the equation:

$$FIW = \frac{fout}{fclk} \times 2^N$$

Where:

- $fout$ = desired output frequency (e.g., 8 kHz or 16 kHz)
- $fclk$ = DDS system clock (10 MHz)
- N= width of the phase accumulator (e.g., 20 bits → 2^{20})

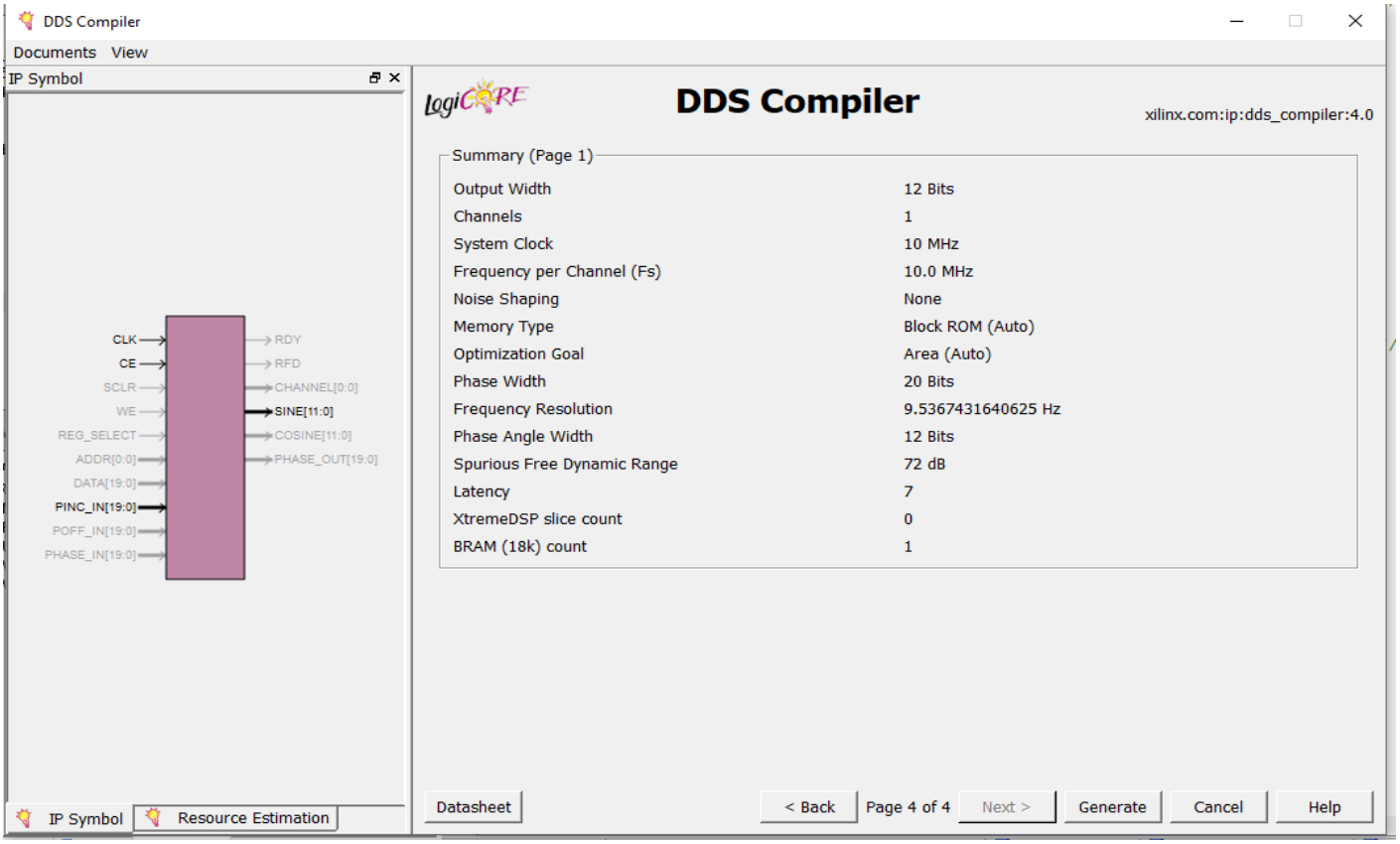

Figure 4.7: DDS Summary and Configuration

- **DDS Module:** Generates a sine wave based on phase accumulator logic and a lookup table (LUT). The sine output is quantized to 12 bits and fed to a DAC or viewed on a DSO.
- Illustrates the internal block-level setup of the DDS, showing phase accumulator, sine LUT, and FIW control lines. Also includes calculated FIW values for 8 kHz and 16 kHz.

```
21  module bfsk_top(
22  input clk_40MHz,
23  input reset,
24  input load,
25  //input [7:0] data_in,
26  output [11:0] sine,
27  output bit_out,
28  output clk_sr_out,clk_dds_out
29
30      );
31     wire [11:0] sine_1;
32  //---clock divider for shift register---
33  /*wire clk_400k,clk_40k,clk_4MHz,clk_sr;
34  clk_div_10 div1(.clk_in(clk_40MHz),.reset(reset),.clk_out(clk_4MHz));
35  clk_div_10 div2(.clk_in(clk_4MHz),.reset(reset),.clk_out(clk_400k));
36  clk_div_10 div3(.clk_in(clk_400k),.reset(reset),.clk_out(clk_40k));
37  clk_div_10 div4(.clk_in(clk_40k),.reset(reset),.clk_out(clk_sr));*/
38
39  wire clk_sr;
40  clk_div_test div1(.clk_in(clk_40MHz),.reset(reset),.clk_out(clk_sr));
41  assign clk_sr_out = clk_sr;
42  //---clock divider for dds---
43  wire clk_dds;
44  clk_div_4 dds_clk_div(.clk_in(clk_40MHz),.reset(reset),.clk_out(clk_dds));
45  assign clk_dds_out = clk_dds;
46
47  //wire clk_dds;
48  //clk_div_1M dds_clk_div(.clk_in(clk_40MHz),.reset(reset),.clk_out(clk_dds));
49  //assign clk_dds_out = clk_dds;
50  //---shift register---
51  //wire bit_out;
52  shift_reg_8bit sr(.clk(clk_sr),.reset(reset),.load(load),.bit_out(bit_out));
53
54  //---mux---
55  wire [19:0] fiw_out_temp,fiw_out;
56  //mux2_1 mux(.clk(clk_sr),.fiw0(20'd839),.fiw1(20'd1678),.select(bit_out),.fiw_out(fiw_out));
57  mux2_1 mux(.fiw0(20'd839),.fiw1(20'd1678),.select(bit_out),.fiw_out(fiw_out_temp));
58
59  //----------- Begin Cut here for INSTANTIATION Template ---// INST_TAG
60  dds_v1 YourInstanceName (
61     .ce(1'b1), // input ce
62     .clk(clk_dds), // input clk
63     .pinc_in(fiw_out), // input [19 : 0] pinc_in
64     .sine(sine_1)
65     //.sine(sine)
66     ); // ouput [11 : 0] sine
67
68  // INST_TAG_END ------ End INSTANTIATION Template ---------
69  assign fiw_out = fiw_out_temp;
70     assign sine = {~sine_1[11], sine_1[11:1]};
71  endmodule
```

Figure 4.8: Verilog Code of BFSK Transmitter

Description:

Clock Generation: Divides 40MHz input to create separate clocks for shift register (‘clk_sr’) and DDS (‘clk_dds’).

Shift Register: Outputs serial bitstream (‘bit_out’) at ‘clk_sr’ rate, controlled by ‘load’ and ‘reset’.

Frequency Selection: Mux selects between two phase increment values (`839` or `1678`) based on ‘bit_out’, setting DDS frequency.

DDS Core: Generates 12-bit sine wave (‘sine_1’) at ‘clk_dds’, using phase increment from the mux.
Output Adjustment: Inverts MSB and halves amplitude of ‘sine_1’ to produce final 12-bit output (‘sine’).

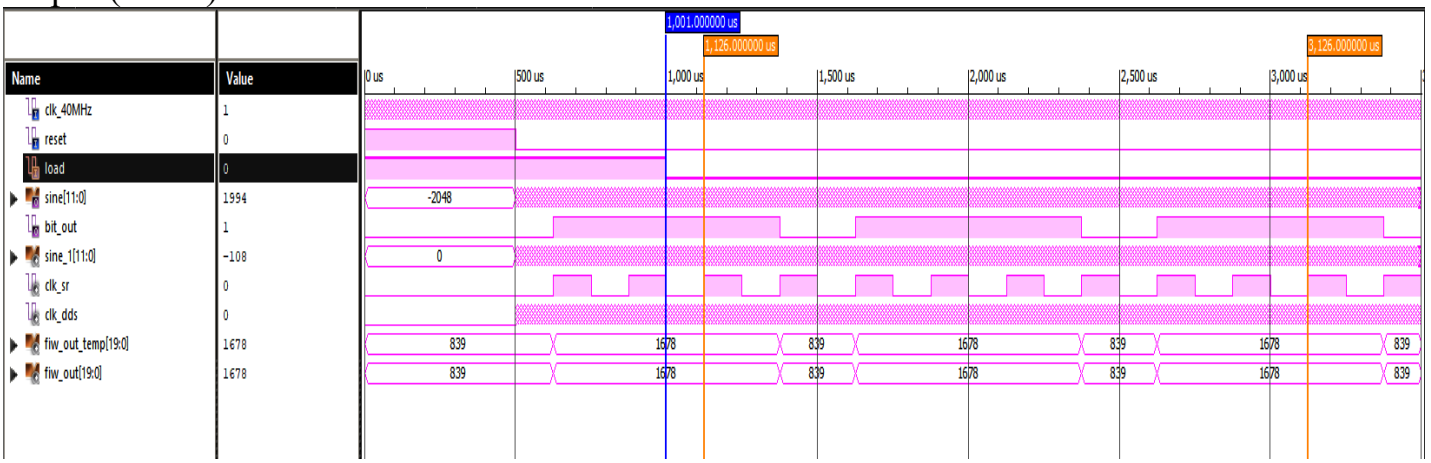

Figure 4.9: Simulation Output of Transmitter

Description: Simulation waveform (e.g., ISim or Vivado) demonstrating frequency shifts corresponding to binary input. It validates the modulated signal for given input pattern of 2ms.

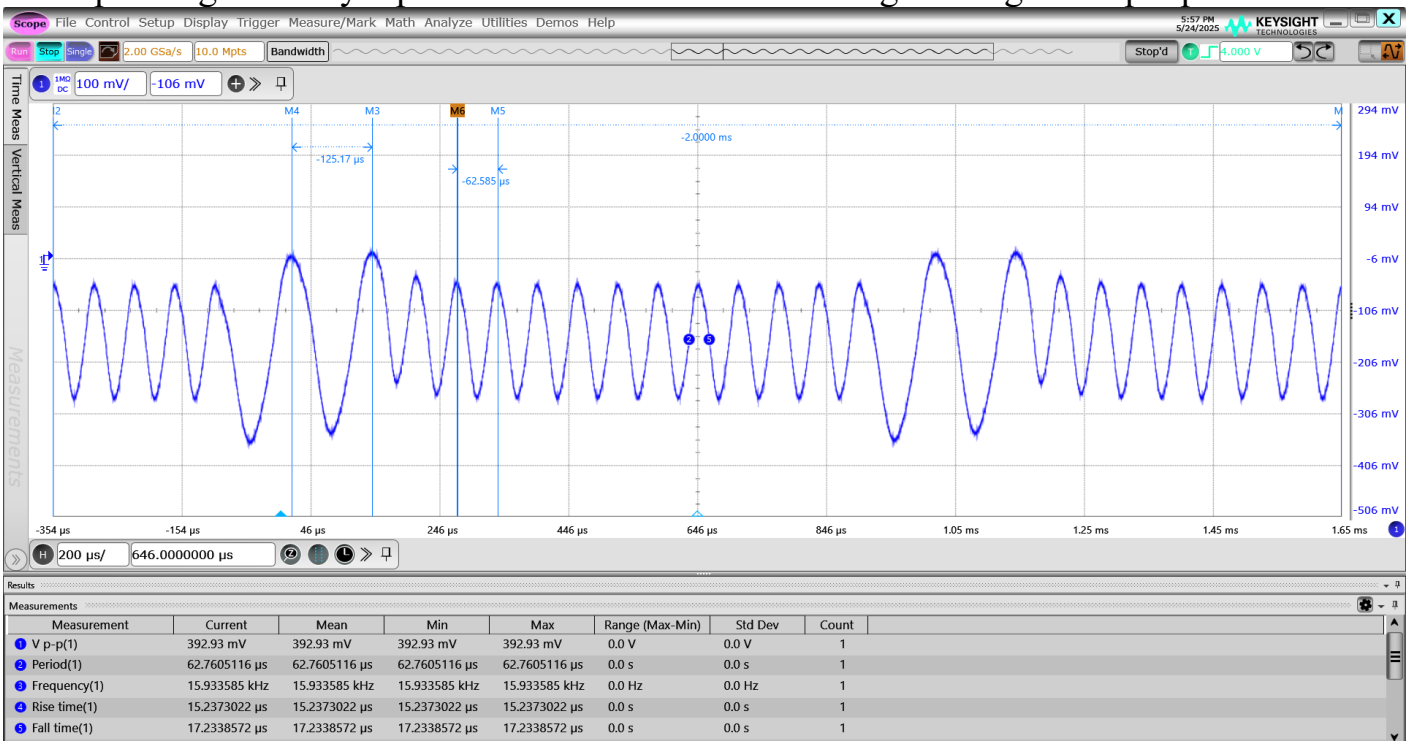

Figure 4.10: DSO Output Captured from Transmitter

Description: Real-time DSO capture of the analog output waveform confirms proper frequency switching as per bitstream. Demonstrates clean sine generation by the DDS for 16 kHz (62.585 µs) and 8 kHz (125 µs).

4.4 Design and Implementation of the Receiver

The receiver reconstructs the transmitted bits by analyzing the frequency content of the received sine waveform. It uses bandpass FIR filters, one each for 8 kHz and 16 kHz, designed

and implemented using Xilinx FIR Compiler IP cores. After filtering, energy levels are computed to detect which frequency is dominant over the bit duration.

Key Components

- **Bandpass Filters (BPFs):** FIR-based digital filters with linear phase, configured to pass narrow bands centered at:
  - 8 kHz (bit '0')
  - 16 kHz (bit '1')
- **Energy Calculator:** Rectifies (squares) the filtered output and integrates it over 2500 samples (250 µs at 10 MHz) for energy estimation.

$$N = T_b * fs$$

Where; N= No of Samples

T_b=Time Period of sine wave input to IP cores

fs= Frequency of IP cores

- **Decision Logic:** Compares the two energy values:

If $E_{16kHz} > E_{8kHz}$, then bit =1; else bit=0

- **Sample Counter:** Keeps track of 2500 samples per bit and resets the detection window at the end of each cycle.

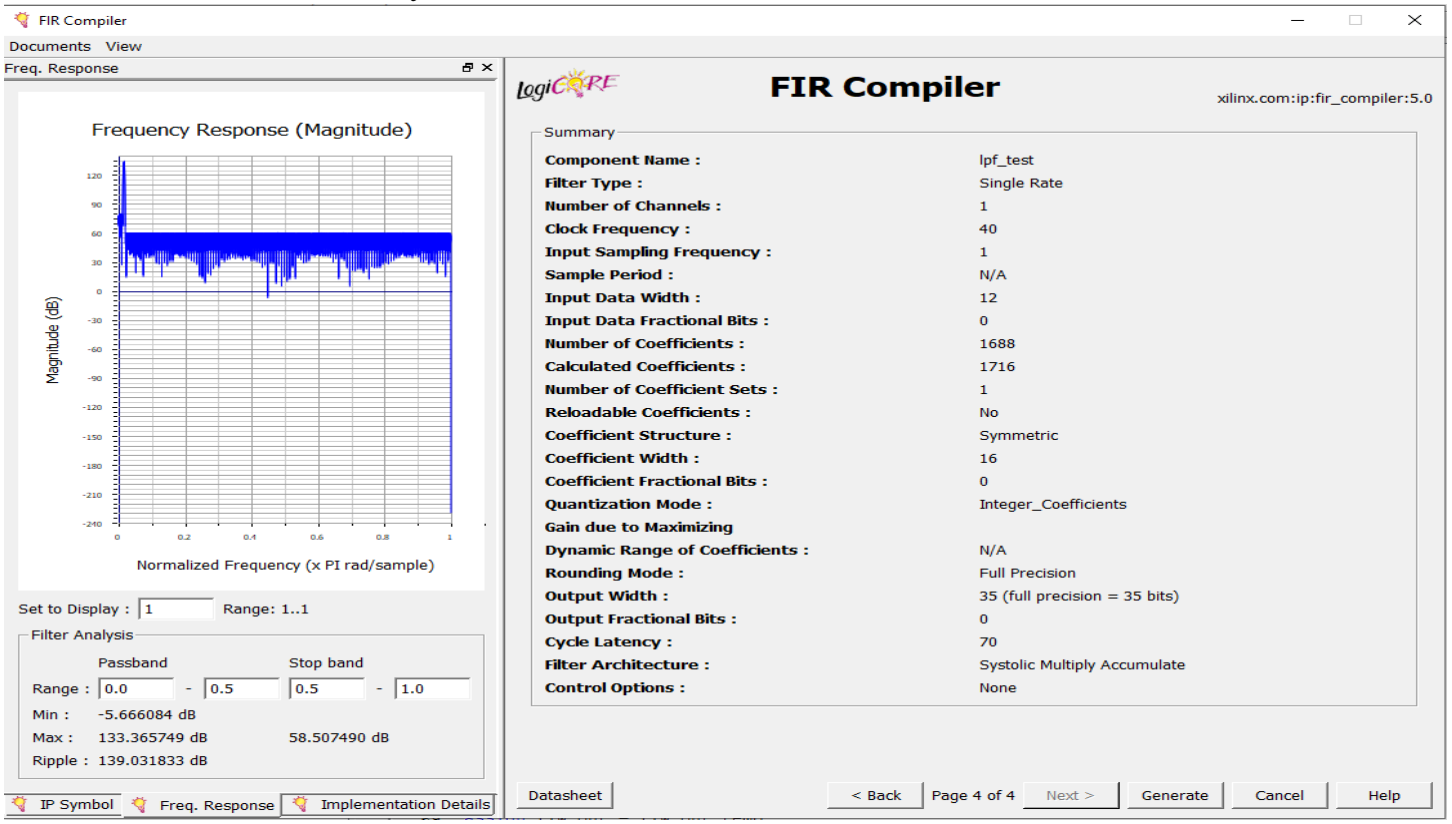

Figure 4.11: FIR IP Core Configuration Summary for 8 kHz and 16 kHz Filters

Description: Filter design parameters such as order, passband/stopband frequencies, and coefficient word lengths are shown. Filters are designed using MATLAB and exported as .coe files.

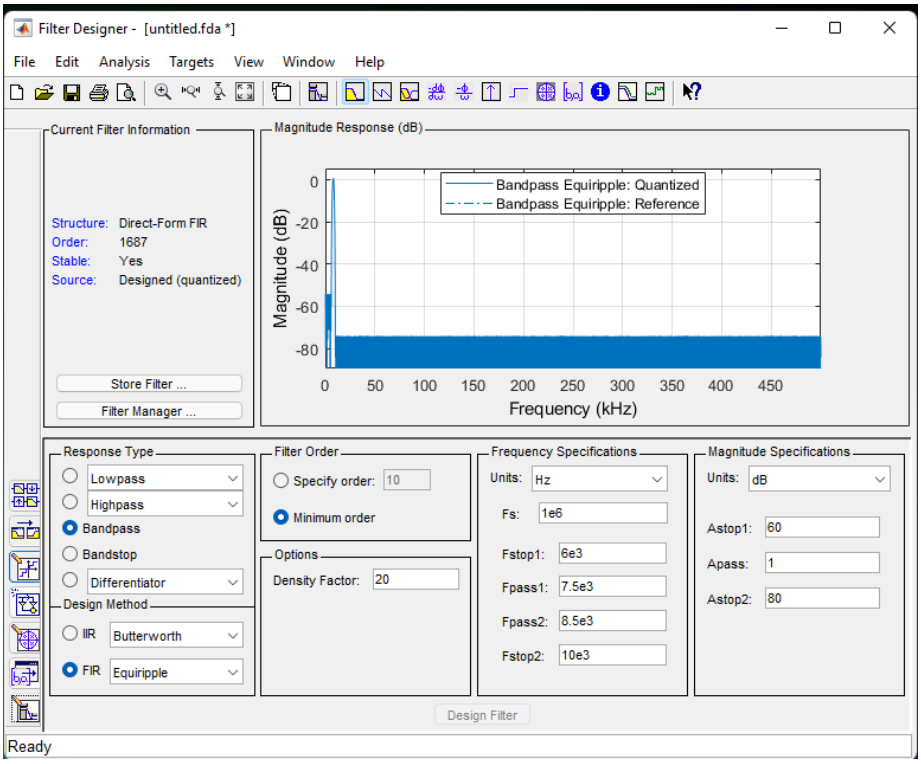

Figure 4.12: Magnitude Response of Bandpass FIR Filter (7.5 kHz – 8.5 kHz)

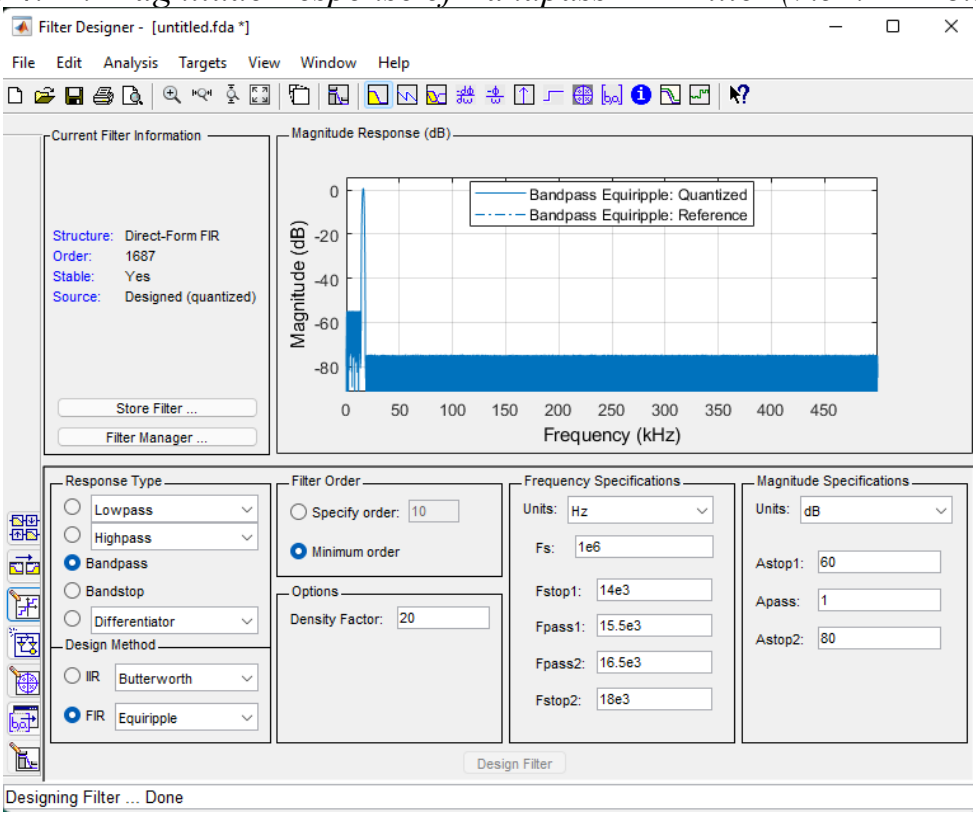

Figure 4.13: Magnitude Response of Bandpass FIR Filter (15.5 kHz – 16.5 kHz)

Description: These two images show MATLAB Filter Designer windows for two different bandpass FIR filters using the Equiripple method. In both cases, the sampling frequency (Fs) is 1 MHz and the filter structure is Direct-Form FIR with an order of 1687. The first filter allows frequencies between 15.5 kHz to 16.5 kHz with stopbands at 14 kHz and 18 kHz, while the second filter allows 7.5 kHz to 8.5 kHz with stopbands at 6 kHz and 10 kHz. Both designs

target a passband ripple of 1 dB and stopband attenuations of 60 dB and 80 dB. The magnitude response plot in each image confirms sharp bandpass characteristics with significant attenuation outside the desired band.

```
module bpf_transreceiver(
input clk_40MHz,
input reset,load,
//input select_2,select_1,
//output [11:0] sine,
output  wire bit_out,clk_sr_out,
output wire valid,bit_out_r
    );
    wire [11:0] sine_1;
    // wire [11:0] sine_2;
    wire [12:1] bpf_8k_ch,bpf_16k_ch;
    wire [34:0] bpf_16k_out,bpf_8k_out;

bfsk_top dut_tx(.clk_40MHz(clk_40MHz),
                  .reset(reset),
                  .load(load),
                  .bit_out(bit_out),
                  .clk_sr_out(clk_sr_out),
                  .sine(sine_1)
                );
//assign sine_2 = {~sine_1[11], sine_1[11:1]};

lpf_test bpf_8k (
   .clk(clk_40MHz), // input clk
   //.rfd(rfd), // ouput rfd
   //.rdy(rdy), // ouput rdy
   .din(sine_1), // input [11 : 0] din
   .dout(bpf_8k_out) // ouput [34 : 0] dout
   );


assign bpf_8k_ch ={~bpf_8k_out[34],bpf_8k_out [34:24]} ;

bpf_test bpf_16k (
   .clk(clk_40MHz),  // input clk
   //.rfd(rfd), // ouput rfd
   //.rdy(rdy), // ouput rdy
   .din(sine_1), // input [11 : 0] din
   .dout(bpf_16k_out) // ouput [34 : 0] dout
   );

assign bpf_16k_ch ={~bpf_16k_out[34],bpf_16k_out [34:24]}
```

```
//assign sine = select_1 ? (sine_2) : (select_2 ? (bpf_8k_ch):(bpf_16k_ch));

reg [31:0] energy_0,energy_1;
reg [15:0] sample_cnt;
reg processing;
reg bit_out_reg,valid_reg;

assign bit_out_r = bit_out_reg;
assign valid = valid_reg;

always @(posedge clk_40MHz)
begin
if(reset)
begin
energy_0 <= 0;
energy_1 <= 0;
processing <= 0;
bit_out_reg <= 0;
sample_cnt <= 0;
valid_reg <= 0;
end
else if(clk_sr_out)begin
energy_0 <= 0;
energy_1 <= 0;
processing <= 1;
sample_cnt <= 0;
valid_reg <= 0;
end
else if(processing)
begin
energy_0 <= energy_0+ (bpf_8k_ch*bpf_8k_ch);
energy_1 <= energy_1+ (bpf_16k_ch*bpf_16k_ch);
sample_cnt <= sample_cnt +1;

if(sample_cnt==2499)
begin
bit_out_reg <= (energy_1 > energy_0)?1'b1:1'b0;
valid_reg <= 1;
processing <= 0;
end
end
end
```

Figure 4.14: RTL Code Structure of BFSK Transceiver with Dual Bandpass Filtering

Description: This Verilog code implements a BFSK transceiver system using two bandpass filters at 8 kHz and 16 kHz. The bfsk_top module generates the modulated sine wave output. Two bpf_test modules act as bandpass filters to separate the BFSK frequencies. Energy values for both channels are computed and compared to recover the transmitted bit. A valid flag indicates when the recovered bit is available.

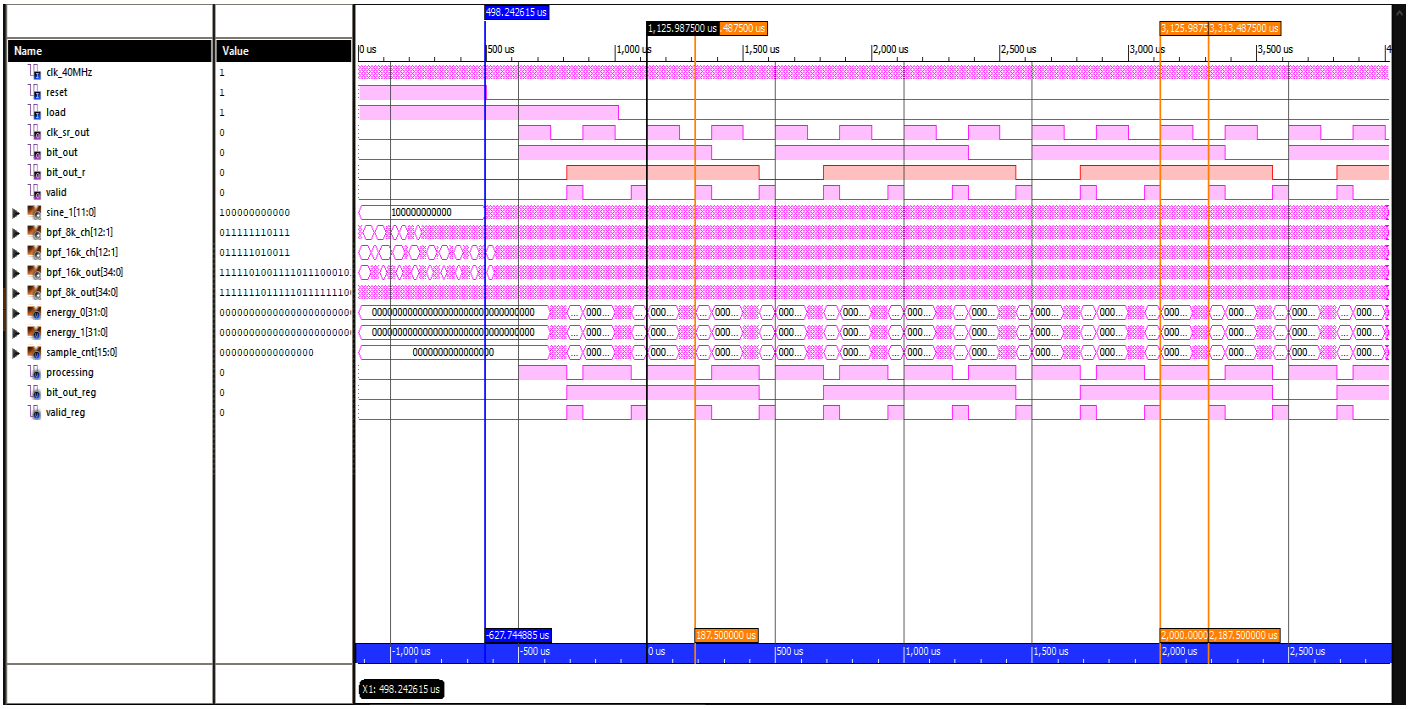

Figure 4.15: Detailed Simulation Output of Receiver Including All Internal Signals and Distortion Effects

Description: Simulation waveform showing how the receiver reconstructs the transmitted bitstream from frequency-differentiated filter outputs but the transmitted (bit_out) and received (bit_out_r) data is having 187 µs of delay.

4.5 RTL Architecture of BFSK Transceiver

The Register Transfer Level (RTL) architecture of the BFSK transceiver system consists of two major subsystems: the transmitter and the receiver. Both subsystems were implemented using Verilog HDL and verified through simulation and synthesis on Xilinx Vivado. The top-level module bfsk_transreceiver integrates these two subsystems to form a complete end-to-end digital BFSK communication system.

4.5.1 BFSK Transmitter

The transmitter generates a frequency-shift keyed waveform based on a given 8-bit input word. It consists of the following RTL components:

- Clock Divider: A 40 MHz master clock is divided down to 4 kHz for the shift register and 10 MHz for the DDS using cascaded frequency dividers.
- 8-bit Shift Register: Operates at 4 kHz and serializes the input data bitstream through right rotation. The output bit is used to modulate the carrier frequency.
- 2:1 Multiplexer: Selects between two predefined Frequency Increment Words (FIWs) corresponding to logic '0' and '1'. For example, FIW_0 (839) generates 8 kHz and FIW_1 (1678) generates 16 kHz.
- Direct Digital Synthesizer (DDS): Receives the selected FIW and 10 MHz clock to produce a 12-bit sine wave. Internally, it contains a phase accumulator and a sine Look-Up Table (LUT).

The transmitter output is a 12-bit sine wave modulated according to the input bit pattern.

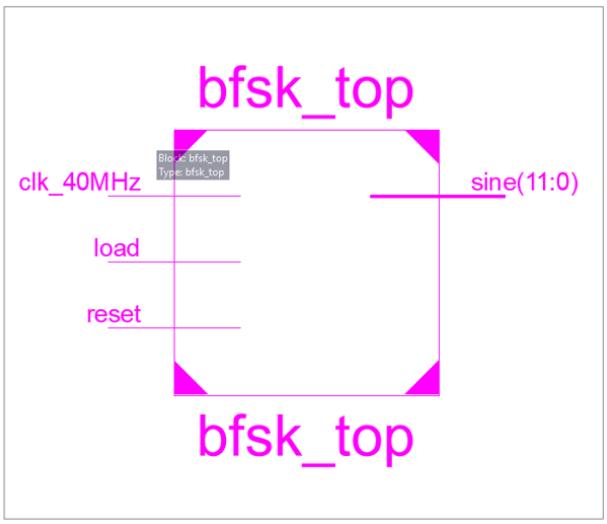

Figure 4.16: RTL Diagram Of Transmitter

4.5.2 BFSK Receiver

The receiver reconstructs the transmitted bitstream from the received modulated waveform. Its architecture includes:

- Sample Counter: Generates a trigger every 2500 samples (at 10 MHz) corresponding to one 4 kHz bit period.
- Bandpass FIR Filters (IP Cores):
  Two FIR bandpass filters are implemented using Xilinx FIR Compiler IP:
  - BPF_1: Centered at 8 kHz
  - BPF_2: Centered at 16 kHz
    These filters are designed in MATLAB and imported via .coe files with a sampling frequency of 1 MHz.
- Energy Estimation: The output of each filter is squared and accumulated over one bit duration to compute signal energy.
- Comparator and Bit Decision Logic: Compares the energy levels from both filters. The output bit is logic '1' if energy at 16 kHz exceeds energy at 8 kHz, else it is logic '0'. A valid signal indicates the availability of each decoded bit.

The BFSK receiver module operates by continuously sampling the incoming sine waveform at a 10 MHz clock rate. Each sample is simultaneously passed through two parallel FIR bandpass filters, each tuned to one of the BFSK frequencies (8 kHz and 16 kHz). The outputs of these filters isolate the respective frequency components present in the received signal. To determine which frequency is dominant during each bit period, the filtered signals are squared to calculate their instantaneous power and accumulated over 2500 samples (equivalent to one 4 kHz bit period). The computed energy values for both frequencies are then compared. If the energy corresponding to the 16 kHz channel exceeds that of the 8 kHz channel, a binary '1' is detected; otherwise, a '0' is declared. A ‘valid’ signal is also generated to indicate when a reliable bit decision has been made, synchronizing with the system’s control logic.

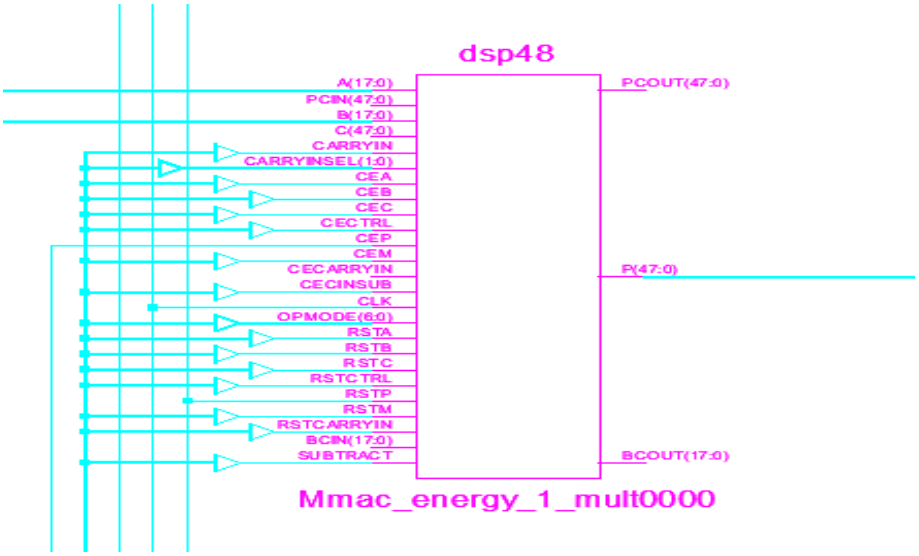

Figure 4.17: DSP48 Block Instance for Energy Computation in BFSK Receiver

This is a DSP48 block primitive from Xilinx FPGAs (like in Vivado-generated schematics). It's a dedicated, high-speed Multiply-Accumulate (MAC) and arithmetic processing unit within the FPGA fabric.

In your schematic:

- Inputs A, B, C carry operand data.
- P output (PCOUT) gives the multiplication or accumulation result.
- Control signals like CEA, CEB, CLK, OPMODE configure how it behaves on each clock cycle.
- The instance name Mmac_energy_1_mult0000 indicates it's performing a multiply operation in your energy computation logic.

In BFSK transceiver system, it's used inside the energy calculation block. Specifically:

- It multiplies the filtered signal samples (like bpf_8k_ch or bpf_16k_ch) with themselves to compute instantaneous power (x^2).
- The result is accumulated over multiple samples to estimate total energy for each band.
- Later, those energy values are compared to detect which frequency was dominant essential for demodulating BFSK signals.

4.5.3 Top-Level Integration

The bfsk_transreceiver module connects the transmitter and receiver. The DDS sine output from the transmitter is directly fed to the receiver. The internal clocks and synchronization signals are appropriately shared between modules to ensure consistent timing.

- Inputs:
  - clk_40MHz: Global input clock
  - reset: System reset
  - load: Data load control for the shift register
  - data_in[7:0]: 8-bit data to transmit
- Outputs:
  - sine_out: MSB of DDS output (for DAC)
  - bit_out: Decoded bit output from receiver
  - valid: High when a decoded bit is available

This RTL-level integration ensures that transmitted data is modulated and demodulated within a single simulation/testbench environment, facilitating verification and analysis.

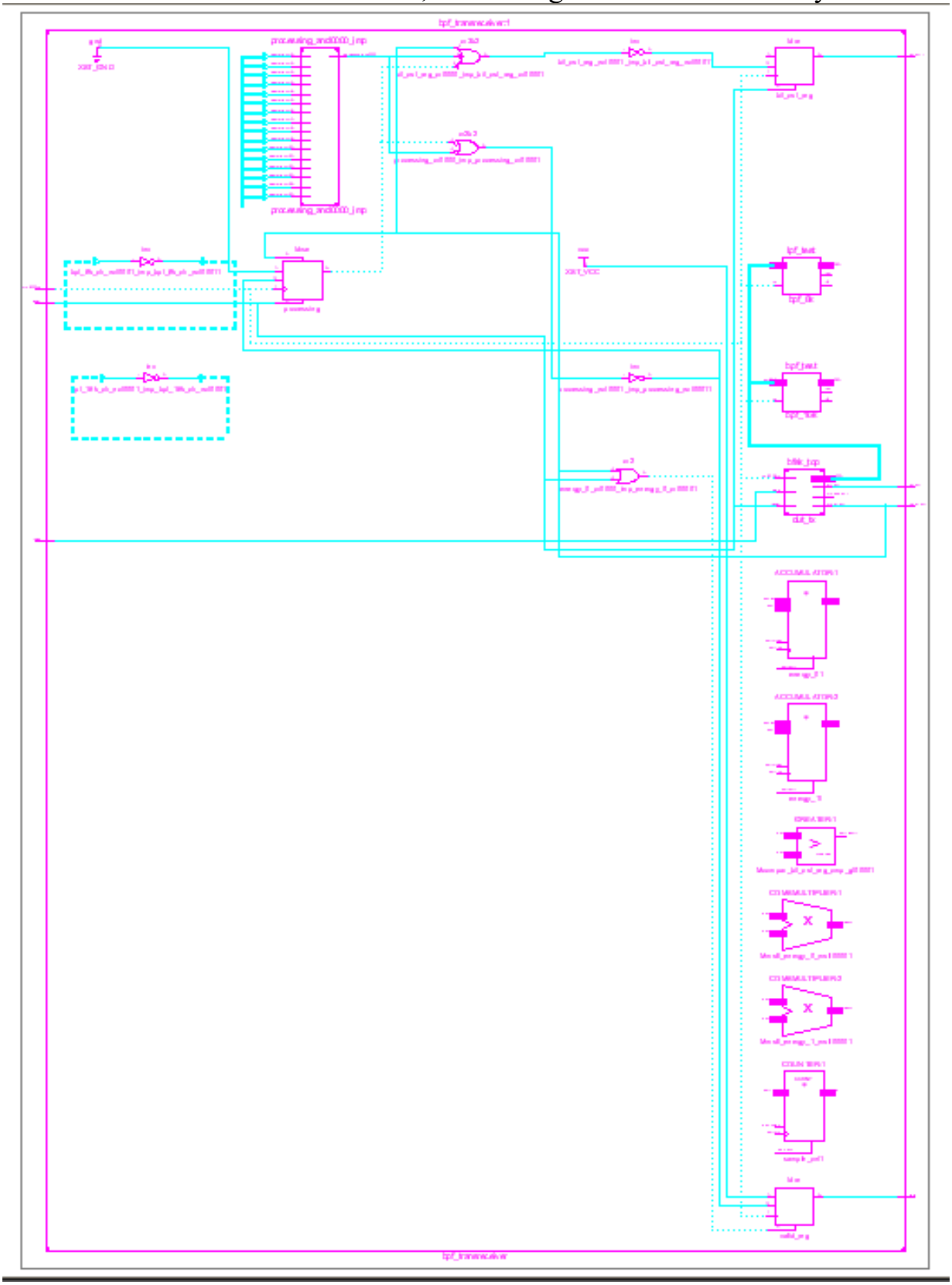

Figure 4.18: RTL Schematic of BFSK Transceiver System

4.6 Simulation, Testing, and Results

The complete BFSK transceiver system was initially simulated using ISim, and later tested on hardware using a Xilinx Virtex-4 FPGA. The simulation verified the correctness of individual modules (transmitter, receiver, and filters) and their timing coordination. Known binary patterns (e.g., 10111011) were used for validation.

- **Transmitter Simulation**: Waveforms from the DDS showed correct switching between 8 kHz and 16 kHz sine waves, synchronized with the 4 kHz bit clock.
- **Receiver Simulation**: FIR filter outputs showed selective amplification corresponding to the transmitted tone. Energy measurement over 2500 samples reliably indicated the correct bit.

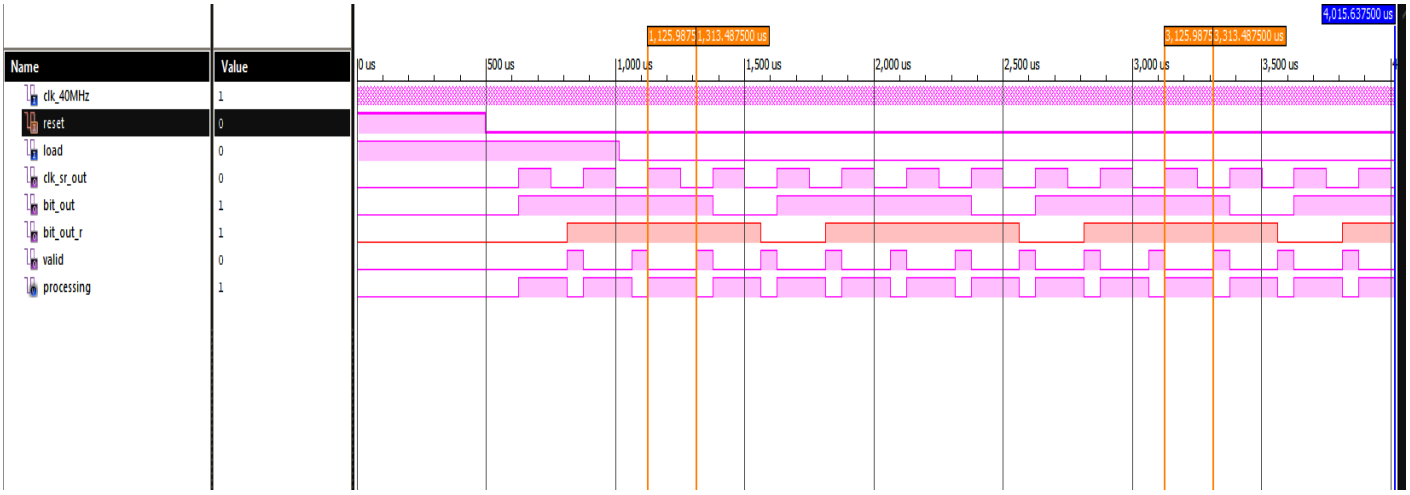

Figure 4.19: Simulation Output of Receiver Core Signals without Distortion Handling

Description: The second image shows the simulation waveform of the receiver's core control signals, including clk, reset, bit_out, valid, and processing without additional data or distortion signals. It verifies the basic timing and output transitions of the receiver logic under normal operating conditions having 187 µs delay in transmitted and received outputs.

- **Hardware Validation**: Real-time signal output was observed on a Digital Storage Oscilloscope (DSO). The analog sine output (from DDS) and the FIR filter responses confirmed the expected spectral characteristics.

```
1  Net "clk_40MHz"  LOC= "H19";
2  Net "reset"  loc  =  "AB23";
3  Net "load"  loc  =  "AB28";
4  net  "sine<0>"  loc  =  "AK24";
5  net  "sine<1>"  loc  =  "AG23";
6  net  "sine<2>"  loc  =  "AG22";
7  net  "sine<3>"  loc  =  "AH19";
8  net  "sine<4>"  loc  =  "AH18";
9  net  "sine<5>"  loc  =  "AH27";
10 net  "sine<6>"  loc  =  "AK27";
11 net  "sine<7>"  loc  =  "AG18";
12 net  "sine<8>"  loc  =  "AG20";
13 net  "sine<9>"  loc  =  "AK22";
14 net  "sine<10>"    loc  =  "AK23";
15 net  "sine<11>"    loc  =  "AJ25";
```

Figure 4.20: FPGA Constraints Summary (UCF File)

Description:This table provides a summary of the User Constraints File (UCF) entries used for pin assignments in the Virtex-4 FPGA implementation. It defines the physical pin mappings between logical signal names and the corresponding hardware pins on the FPGA, as well as any additional timing constraints required for proper operation.

Signal	FPGA Pin	Description
clk_40MHz	H19	System clock input (40 MHz)
reset	AB23	Active-high system reset
load	AB28	Load control signal for shift register
sine[0]	AK24	12-bit DDS output to DAC (bit 0)
sine[1]	AG23	12-bit DDS output to DAC (bit 1)
sine[2]	AG22	12-bit DDS output to DAC (bit 2)
sine[3]	AH19	12-bit DDS output to DAC (bit 3)
sine[4]	AH18	12-bit DDS output to DAC (bit 4)
sine[5]	AK27	12-bit DDS output to DAC (bit 5)
sine[6]	AK26	12-bit DDS output to DAC (bit 6)
sine[7]	AG18	12-bit DDS output to DAC (bit 7)
sine[8]	AG20	12-bit DDS output to DAC (bit 8)
sine[9]	AK22	12-bit DDS output to DAC (bit 9)
sine[10]	AK23	12-bit DDS output to DAC (bit 10)
sine[11]	AJ25	12-bit DDS output to DAC (bit 11)

The UCF file is essential in ensuring that the logical signals in the design are correctly mapped to the physical I/O pins of the FPGA package. All signal assignments are made in accordance with the board schematic and hardware interfacing requirements. Timing constraints, such as clock period specifications, are also defined in the UCF to guide synthesis and implementation tools.

4.7 Challenges Faced and Mitigation Strategies

Challenge	Mitigation Strategy
FIR filter passband leakage	Filter order and transition bandwidth were optimized using MATLAB. Additional stopband attenuation was added to avoid spectral overlap.
DDS phase noise and waveform distortion	A 12-bit fixed-point LUT was used, and output was smoothed via hardware filtering (or DAC filtering).
Timing misalignment between transmitter and receiver	Synchronized using sample counters and FSMs to ensure fixed window size (2500 samples/bit).
IP core FIR integration	Used .coe files generated from MATLAB and verified configuration in Vivado's FIR Compiler GUI. Proper coefficient width and signed representation were ensured.

4.8 Conclusion

This project demonstrated the design and implementation of a Binary Frequency Shift Keying (BFSK) transceiver entirely on a Xilinx FPGA platform using Verilog HDL.

The transmitter module successfully generated modulated signals using a Direct Digital Synthesizer (DDS), where frequency selection was based on serialized binary data. The generated waveform:

$$S(t) = \begin{cases} Acos(2\pi f_0 t), & for\ bit\ 0 \\ Acos(2\pi f_1 t), & for\ bit\ 1 \end{cases}$$

was tested to switch cleanly between 8 kHz and 16 kHz for logic '0' and '1', respectively.

The receiver employed FIR bandpass filters implemented using Xilinx IP cores to isolate and detect the two tones. Bit detection was performed by comparing the energy of each filter output over 2500-sample windows (1 bit = 250 µs at 10 MHz sampling rate).

The system is modular, synchronous, and can be extended in future work to:

- Support M-ary FSK (with more than two frequencies)
- Integrate Goertzel-based detection for resource optimization
- Add error correction and AGC modules for noisy environments

This BFSK transceiver can be readily adapted for real-time telemetry, flight control, or low-data-rate secure communication systems in embedded and aerospace domains.

4.9 Future Work

While the current BFSK transceiver implementation successfully achieves binary frequency modulation and detection using FIR-based bandpass filters, several enhancements can be explored in future iterations:

- **Extension to M-ary FSK**: Expanding the system to support more than two frequencies (e.g., 4-FSK or 8-FSK) to increase data rate and channel efficiency.
- **Goertzel Algorithm-based Detection**: Replacing the dual FIR bandpass filter architecture with a resource-efficient Goertzel algorithm for tone detection, reducing FPGA resource utilization.
- **Integration of Error Detection and Correction Modules**: Incorporating mechanisms like CRC or Hamming codes to improve communication reliability in noisy environments.
- **Automatic Gain Control (AGC)**: Adding AGC modules to maintain consistent signal amplitude before demodulation, improving performance under variable input conditions.
- **Real-time Wireless Communication Interface**: Extending the design to include RF transceivers and antenna interfacing for live over-the-air testing and telemetry applications.

These improvements would further enhance the robustness, scalability, and practical applicability of the BFSK transceiver system in real-world embedded and aerospace communication systems.

References

- Telecommand Group Report Telecommand Unit, ITR, DRDO
- Ashiya R, Gupta J P, Singal Y K, Venkataraman D, Bhaskaranarayana A, Das U N and Seshadri B V, *THE TELECOMMAND SYSTEM*, ISRO Satellite Centre
- All About FPGAs, *https://www.eetimes.com/all-about-fpgas/*
- FPGA Architecture and Applications, *https://www.elprocus.com/fpga-architecture-and-applications/*
- FPGA Manufacturers, *https://www.raypcb.com/fpga-manufacturers/*
- "ITR_V4-SX55-DSP-Board_Final", Virtex-4 Board Manual
- "ISE 6 In-Depth Tutorial", *https://cseweb.ucsd.edu/classes/sp07/cse141L/lab1/ise8tut.pdf*
- MATLAB and Simulink Filter Design Toolbox, MathWorks Inc.
- Xilinx FIR Compiler IP Core Documentation, Xilinx Inc.
- *Verilog HDL Reference Manual*, IEEE Standard 1364-2005
- *Digital Storage Oscilloscope (DSO) User Manual*, Tektronix Instruments
- *Xilinx ISim Simulator Guide*, Xilinx Documentation Archive
- *Xilinx iMPACT Configuration and JTAG Programming Guide*