Coimbatore

Department of Electrical and Electronics Engineering

MTech -Embedded Systems

PROJECT RECORD

21ES612 - FPGA BASED SYSTEM DESIGN

II semester, April 2025.

IMPLEMENTATION OF DIGITAL MODULATION TECHNIQUES (ASK, PSK, FSK)

TEAM MEMBERS:

CB.EN.P2EBS24011-KEERTHANA M G

CB.EN.P2EBS24017-NIKHITHA P

Faculty-in-Charge: PRABHU E

INDEX

ABSTRACT

This paper presents the design and FPGA-based implementation of fundamental digital modulation techniques including Amplitude Shift Keying (ASK), Frequency Shift Keying (FSK), and Phase Shift Keying (PSK). Simulation and synthesis are performed using tools such as Xilinx Vivado, and hardware verification is carried out using the Basys 3 FPGA board. The aim of this work is to realize each modulation scheme using hardware description language (HDL) and evaluate their performance on an FPGA platform. The implementation focuses on generating carrier signals, modulating binary input data, and observing the resulting waveforms.

CHAPTER 1
LITERATURE SURVEY

1. Paper: Efficient Design and Implementation of the Real-Time Multi-Type Digital Modulations System Based on FPGA.
   Details: Journal from ResearchGate - 2023

   This paper focuses on the implementation of ASK, FSK, PSK, DPSK, QPSK, DQPSK, 8-PSK, 8-QAM, 16-QAM, 4-Array ASK, and 4-Array FSK in a Spartan 3E FPGA development board using Very High-Speed Hardware Description Language (VHDL) and synthesized using Xilinx ISE 14.7 software.

2. Paper: Review of PSK and QAM - Digital modulation technique on FPGA
   Details: IEEE Conference

   This paper focuses on examining PSK and QAM in the practical implementation on FPGA platforms. This study aims to demonstrate hardware realizations of these modulation techniques. This paper compares both of them in terms of spectral utilization, bit rate error (BRE), and hardware complexity. The paper concludes that FPGA modulation provides high performance and power efficiency.

3. Paper: Implementation of Digital Modulation Techniques in High-Speed FPGA Board
   Details: IEEE Conference

   This paper focuses on ASK, FSK, and PSK. The paper has been implemented on the Xilinx Spartan-3 FPGA board. This paper compares the hardware optimization, resource utilization, and performance evaluation of three modulations. This paper concludes that FPGA is suitable for applications that require high bandwidth and fast data transmission.

4. Paper: Analysis and implementation of Minimum Shift Keying (MSK) modulation on FPGA platform
   Details: IEEE Conference

   This paper focuses on analyzing in terms of spectral efficiency and power efficiency in comparison to other modulation techniques. The main objective is to see how MSK modulation can be effectively implemented in FPGA, the implementation in both transmitter and receiver for MSK. The paper comes with a conclusion that MSK is highly efficient for communication systems in FPGA.

5. Paper: FPGA Implementation of Digital Modulation Techniques
   Details: IEEE Conference

   This paper focuses on design and implementation using Hardware Description Languages (HDL) on FPGA boards. It demonstrates FPGAs potential for real-time processing and versatility in communication systems. It focuses on ASK, FSK, PSK, and QAM. It demonstrates real-time processing and the versatility of communication systems.

CHAPTER-2
METHODOLOGY

1. Understand the modulation technique:

- Choose the modulation technique (amplitude shift keying, binary phase shift keying, or frequency shift keying).
- Implement the modulation technique using Verilog code.
- Generate the carrier signal and see the modulated signal in the digital signal oscilloscope.

2. Setup basys3 FPGA environment:

- Use Vivado for design, simulation, and bitstream generation.
- Use PMOD pins for the output signal.

3. Designing the modulator in Verilog:

- Generate the carrier wave using a toggling signal.
- The modulator accepts bits serially as input.
- Implement the modulation technique logic.

4. Simulation and implementation:

- Use a testbench to verify the modulation technique waveforms.
- Synthesize and implement the modulation technique designs.

5. Connect to the basys3 board:

- Generate the bitstream for ASK, FSK, and BPSK.
- Program the basys3 board using the hardware manager in Vivado.
- Connect the DSO to the output pins.

6. Output in DSO:

- Generate the carrier wave using the Wavegen function in the DSO.

- And connect it to the Pmod pins in the Basys3 board.
- See the modulated output in the DSO.

CHAPTER-3
CODE

3.1 MATLAB CODE:

3.1.1 ASK MATLAB Simulink code:

```
clc; clear; close all;
fs = 1000; % Sampling frequency
fc = 100; % Carrier frequency
Rb = 10; % Bit rate
Tb = 1/Rb; % Bit duration
N = 10; % Number of bits
t = 0:1/fs:Tb-1/fs; % Time vector for one bit
% Generate random binary data
data = randi([0 1], 1, N);
modulated_signal = [];
% Carrier Signal
carrier = cos(2*pi*fc*t);
carrier_full = []; % Store full carrier wave for all bits
% ASK Modulation
for i = 1:N
  carrier_full = [carrier_full, carrier]; % Store continuous carrier wave
  if data(i) == 1
     modulated_signal = [modulated_signal, carrier]; % Send carrier for bit 1
  else
     modulated_signal = [modulated_signal, zeros(1, length(t))] ; % No carrier for bit 0
  end
end
% Time vector for entire signal
```

```
time = 0:1/fs:(N*Tb)-1/fs;
% Plotting
figure;
% Plot Binary Data
subplot(4,1,1);
stairs(0:N-1, data, 'LineWidth', 2);
xlabel('Bit Position'); ylabel('Amplitude');
title('Binary Data');
axis([0 N -0.5 1.5]);
% Plot Carrier Signal
subplot(4,1,2);
plot(time, carrier_full, 'g', 'LineWidth', 1);
xlabel('Time (s)'); ylabel('Amplitude');
title('Carrier Wave');
xlim([0, N*Tb]);
% Plot ASK Modulated Signal
subplot(4,1,3);
plot(time, modulated_signal, 'b', 'LineWidth', 1);
xlabel('Time (s)'); ylabel('Amplitude');
title('ASK Modulated Signal');
xlim([0, N*Tb]);
% ASK Demodulation using Envelope Detector
demodulated_signal = abs(hilbert(modulated_signal));
threshold = max(demodulated_signal) / 2;
received_bits = [];
for i = 1:N
  sample = mean(demodulated_signal((i-1)*length(t) + (1:length(t))));
  received_bits = [received_bits, sample > threshold];
end
% Plot Demodulated Data
subplot(4,1,4);
```

```
stairs(0:N-1, received_bits, 'r', 'LineWidth', 2);
xlabel('Bit Position'); ylabel('Amplitude');
title('Demodulated Data');
axis([0 N -0.5 1.5]);
% Display results
disp('Original Data:');
disp(data);
disp('Demodulated Data:');
disp(received_bits);
```

3.1.2 FSK MATLAB Simulink code:

```
clc; clear; close all;
% Parameters
Fs = 1000;     % Sampling frequency (Hz)
Fc1 = 5;       % Frequency for binary '0' (Hz)
Fc2 = 20;      % Frequency for binary '1' (Hz)
Tb = 1;        % Bit duration (seconds)
N = 8;         % Number of bits
% Generate Random Binary Data
data = randi([0, 1], 1, N); % Random binary sequence
t = 0:1/Fs:N*Tb-1/Fs;       % Time vector for full signal
% Generate FSK Signal
fsk_signal = zeros(1, length(t));
bit_idx = 1;
for i = 1:length(t)
  if t(i) > bit_idx * Tb
     bit_idx = bit_idx + 1; % Move to next bit
  end

  if bit_idx > N
```

```
        break;
    end

    % Select frequency based on bit value
    if data(bit_idx) == 0
        fsk_signal(i) = sin(2 * pi * Fc1 * t(i));
    else
        fsk_signal(i) = sin(2 * pi * Fc2 * t(i));
    end
end
% Plot Results
figure;
subplot(3,1,1);
stem(0:N-1, data, 'filled');
title('Input Binary Data');
ylim([-0.2 1.2]); grid on;
xlabel('Bit Index'); ylabel('Binary Value');
subplot(3,1,2);
plot(t, fsk_signal);
title('FSK Modulated Signal');
xlabel('Time (s)'); ylabel('Amplitude');
grid on;
subplot(3,1,3);
spectrogram(fsk_signal, 128, 120, 128, Fs, 'yaxis');
title('Spectrogram of FSK Signal');
```

3.1.3 PSK MATLAB Simulink Code:

```
clc; clear; close all;
% Parameters
Fs = 1000;      % Sampling frequency (Hz)
```

```
Fc = 10;       % Carrier frequency (Hz)
Tb = 1;        % Bit duration (seconds)
N = 8;         % Number of bits
% Generate Random Binary Data (0s and 1s)
data = randi([0, 1], 1, N); % Random binary sequence
t = 0:1/Fs:N*Tb-1/Fs;       % Time vector for full signal
% Generate BPSK Signal
psk_signal = zeros(1, length(t));
bit_idx = 1;
for i = 1:length(t)
  if t(i) > bit_idx * Tb
     bit_idx = bit_idx + 1; % Move to next bit
  end

  if bit_idx > N
     break;
  end

  % PSK Modulation: Phase Shift 0° for Bit '1', 180° for Bit '0'
  if data(bit_idx) == 1
     psk_signal(i) = sin(2 * pi * Fc * t(i)); % Normal carrier for '1'
  else
     psk_signal(i) = -sin(2 * pi * Fc * t(i)); % Inverted carrier for '0' (180° phase shift)
  end
end
% Plot Results
figure;
subplot(3,1,1);
stem(0:N-1, data, 'filled');
title('Input Binary Data');
ylim([-0.2 1.2]); grid on;
```

```
xlabel('Bit Index'); ylabel('Binary Value');
subplot(3,1,2);
plot(t, psk_signal);
title('PSK Modulated Signal (BPSK)');
xlabel('Time (s)'); ylabel('Amplitude');
grid on;
subplot(3,1,3);
spectrogram(psk_signal, 128, 120, 128, Fs, 'yaxis');
title('Spectrogram of PSK Signal');
```

3.2 VERILOG CODE:

3.2.1 Amplitude Shift Keying (ASK) Modulation Code:

ASK MODULATION CODE:

```
`timescale 1ns / 1ps

module ask_mod (
    input wire clk,         // Clock signal (100 MHz)
    input wire reset,       // Reset signal
    input wire bit_in,      // Input bit stream
    output reg ask_out,     // ASK Modulated Output
    output reg carrier      // Carrier signal output
);

// Generate carrier wave (square wave)
always @(posedge clk or posedge reset) begin
    if (reset) begin
        carrier <= 0;
        ask_out <= 0;
    end else begin
        carrier <= ~carrier;        // Generate 50 MHz carrier wave
    end
end

// ASK Modulation Logic
always @(*) begin
    if (bit_in)
        ask_out = carrier;
    else
        ask_out = 0;
end

endmodule
```

TESTBENCH CODE:

```
`timescale 1ns / 1ps

module ask_mod_tb;

reg clk;
reg reset;
reg bit_in;
wire ask_out;
wire carrier;

// Instantiate the ASK Modulator module
ask_mod uut (
    .clk(clk),
    .reset(reset),
    .bit_in(bit_in),
    .ask_out(ask_out),
    .carrier(carrier)
);

// Clock Generation (100MHz -> 10ns period)
always #5 clk = ~clk;

initial begin
    clk = 0;
    reset = 1;
    bit_in = 0;

    #20 reset = 0; // Release reset

    #50 bit_in = 1; // Enable modulation
    #100 bit_in = 0;
    #150 bit_in = 1;
    #200 bit_in = 0;
    #250 bit_in = 1;

    #500 $finish; // End simulation
end
```

```
initial begin
   $monitor("Time: %0t | bit_in: %b | carrier: %b | ask_out: %b",
            $time, bit_in, carrier, ask_out);
end

endmodule
```

CONSTRAINTS CODE:

```
## Clock Signal (100 MHz)
set_property PACKAGE_PIN W5 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]

## Reset Button (BTNC - Center Button)
set_property PACKAGE_PIN U18 [get_ports reset]
set_property IOSTANDARD LVCMOS33 [get_ports reset]

## Input Bit Stream (SW0 - Switch 0)
set_property PACKAGE_PIN V17 [get_ports bit_in]
set_property IOSTANDARD LVCMOS33 [get_ports bit_in]

## ASK Modulated Output (Pmod JA2)
set_property PACKAGE_PIN J2 [get_ports ask_out]
set_property IOSTANDARD LVCMOS33 [get_ports ask_out]

## Carrier Signal Input (Pmod JA1) - From DSO
set_property PACKAGE_PIN J1 [get_ports carrier]
set_property IOSTANDARD LVCMOS33 [get_ports carrier]
set_property PULLUP TRUE [get_ports carrier]  # Optional pull-up resistor
```

3.2.2 Frequency Shift Keying (FSK) Modulation Code:

FSK MODULATION CODE:

```
module fsk_modulator(
   input clk,          // 100 MHz clock
   input rst,          // Reset signal
   input bit_in,       // Input data bit
```

```
    output reg fsk_out,  // FSK Modulated output
    output reg carrier   // Carrier wave output
);

parameter FREQ_0 = 8;  // Frequency division factor for bit 0
parameter FREQ_1 = 4;  // Frequency division factor for bit 1

reg [7:0] counter_fsk = 0;  // Counter for FSK modulation
reg [7:0] threshold;        // Dynamic threshold selection

// Carrier signal generation (Same as clk)
always @(posedge clk or posedge rst) begin
    if (rst)
        carrier <= 0;
    else
        carrier <= ~carrier;
end

// FSK Modulation Logic
always @(posedge clk or posedge rst) begin
    if (rst) begin
        counter_fsk <= 0;
        fsk_out <= 0;
        threshold <= FREQ_0;  // Default threshold
    end else begin
        // Set threshold based on input bit
        if (bit_in)
            threshold <= FREQ_1;
        else
            threshold <= FREQ_0;

        // Frequency division logic
        if (counter_fsk >= threshold) begin
            fsk_out <= ~fsk_out;
            counter_fsk <= 0;
        end else begin
            counter_fsk <= counter_fsk + 1;
        end
    end
end
```

```
endmodule
```

TESTBENCH CODE:

```
`timescale 1ns / 1ps

module fsk_modulator_tb;

reg clk;
reg rst;        // Use 'rst' instead of 'reset'
reg bit_in;
wire fsk_out;
wire carrier;

// Instantiate the FSK modulator
fsk_modulator uut (
    .clk(clk),
    .rst(rst),    // Corrected from 'reset' to 'rst'
    .bit_in(bit_in),
    .fsk_out(fsk_out),
    .carrier(carrier)
);

// Clock generation (100 MHz -> 10 ns period)
always #5 clk = ~clk;

initial begin
    // Initialize signals
    clk = 0;
    rst = 1;   // Apply reset
    bit_in = 0;
    #20 rst = 0;  // Release reset after 20ns

    // Test case 1: bit_in = 0
    #100 bit_in = 1; // Change bit_in
    #200 bit_in = 0; // Change back

    // Test case 2: bit_in toggles multiple times
    #100 bit_in = 1;
    #100 bit_in = 0;
```

```
    #100 bit_in = 1;

    #200 $stop;  // Stop simulation
end

endmodule
```

CONSTRAINTS CODE:

```
## Clock Signal (100 MHz)
set_property PACKAGE_PIN W5 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]

## Reset Button (BTNC - Center Button)
set_property PACKAGE_PIN U18 [get_ports rst]
set_property IOSTANDARD LVCMOS33 [get_ports rst]

## Input Bit Stream (SW0 - Switch 0)
set_property PACKAGE_PIN V17 [get_ports bit_in]
set_property IOSTANDARD LVCMOS33 [get_ports bit_in]

## FSK Modulated Output (Pmod JA2)
set_property PACKAGE_PIN J2 [get_ports fsk_out]
set_property IOSTANDARD LVCMOS33 [get_ports fsk_out]

## Carrier Signal Output (Pmod JA3)
set_property PACKAGE_PIN J3 [get_ports carrier_out]
set_property IOSTANDARD LVCMOS33 [get_ports carrier_out]
```

3.2.3 Binary Phase Shift Keying (BPSK) Modulation Code:

BPSK MODULATION CODE:

```
module bpsk_modulator (
    input wire clk,         // Clock signal
    input wire rst,         // Reset signal
    input wire data_in,     // Input Data (1 or 0)
    output reg bpsk_out,    // BPSK Modulated Signal
    output reg carrier_out // Carrier Signal
```

```
);

reg carrier;

// Generate carrier wave (square wave)
always @(posedge clk or posedge rst) begin
    if (rst) begin
        carrier <= 0;
        carrier_out <= 0;
    end else begin
        carrier <= ~carrier;  // Toggle carrier wave
        carrier_out <= carrier;
    end
end

// BPSK Modulation
always @(posedge clk or posedge rst) begin
    if (rst)
        bpsk_out <= 0;
    else
        bpsk_out <= (data_in) ? ~carrier : carrier;  // Phase shift when data_in = 1
end

endmodule
```

TESTBENCH CODE:

```
*timescale 1ns / 1ps

module tb_bpsk_modulator;

reg clk;
reg rst;
reg data_in;
wire bpsk_out;
wire carrier_out;

// Instantiate the BPSK Modulator module
bpsk_modulator uut (
```

```
    .clk(clk),
    .rst(rst),
    .data_in(data_in),
    .bpsk_out(bpsk_out),
    .carrier_out(carrier_out)
);

// Clock Generation (50MHz -> 20ns period)
always #10 clk = ~clk;

initial begin
   clk = 0;
    rst = 1;
    data_in = 0;

    #30 rst = 0; // Release reset

    #100 data_in = 1; // Apply Data Input changes
    #200 data_in = 0;
    #200 data_in = 1;
    #300 data_in = 0;
    #400 data_in = 1;

    #1000 $finish; // End simulation
end

initial begin
    $monitor("Time: %0t | data_in: %b | carrier_out: %b | bpsk_out: %b",
             $time, data_in, carrier_out, bpsk_out);
end

endmodule
```

CONSTRAINTS CODE:

```
## Clock Signal (100 MHz)
set_property PACKAGE_PIN W5 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
```

```
## Reset Button (BTNC - Center Button)
set_property PACKAGE_PIN U18 [get_ports rst]
set_property IOSTANDARD LVCMOS33 [get_ports rst]

## Input Bit Stream (SW0 - Switch 0)
set_property PACKAGE_PIN V17 [get_ports data_in]
set_property IOSTANDARD LVCMOS33 [get_ports data_in]

## BPSK Modulated Output (Pmod JA2)
set_property PACKAGE_PIN J2 [get_ports bpsk_out]
set_property IOSTANDARD LVCMOS33 [get_ports bpsk_out]

## Carrier Signal Output (Pmod JA3)
set_property PACKAGE_PIN J3 [get_ports carrier_out]
set_property IOSTANDARD LVCMOS33 [get_ports carrier_out]

## Carrier Signal Input (Pmod JA1) - From DSO (If Needed)
set_property PACKAGE_PIN J1 [get_ports carrier]
set_property IOSTANDARD LVCMOS33 [get_ports carrier]
set_property PULLUP TRUE [get_ports carrier]  # Optional pull-up resistor
```

CHAPTER-4
RESULTS AND DISCUSSION

4.1 ASK Simulink Model:

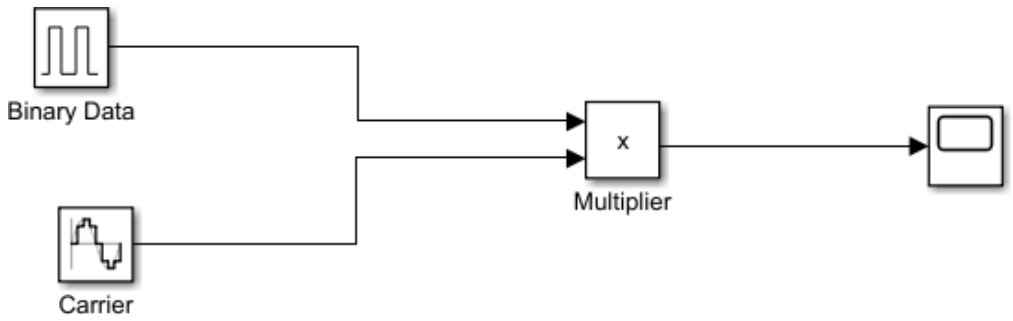

Figure 4.1.1: ASK Simulink model

ASK Simulink Result:

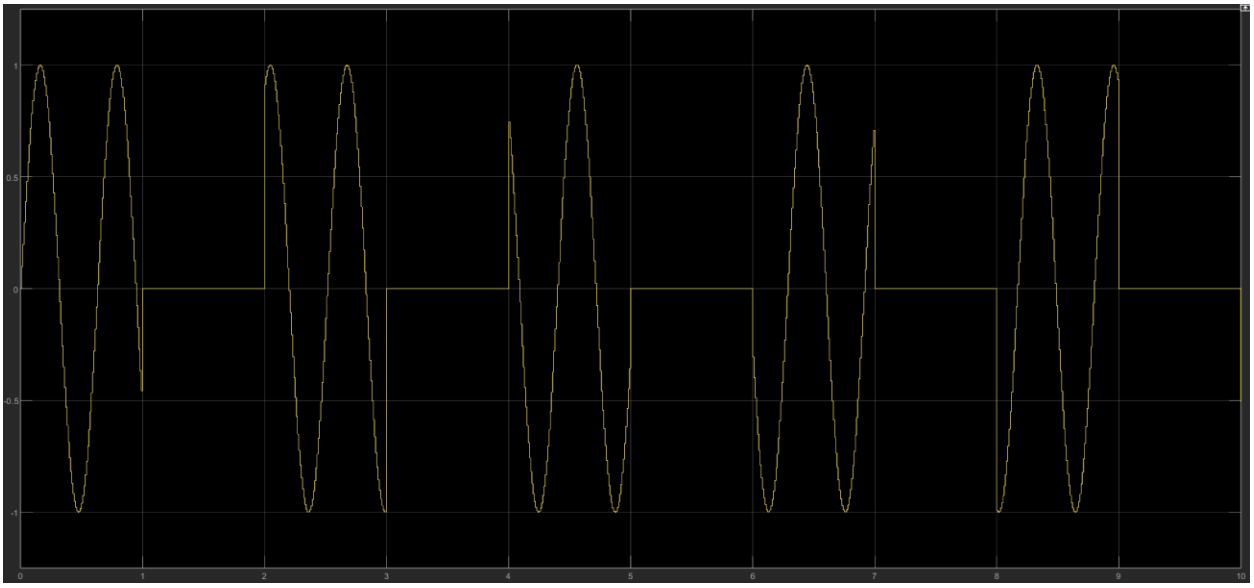

Figure 4.1.2: ASK Simulink output

4.2 FSK Simulink Model:

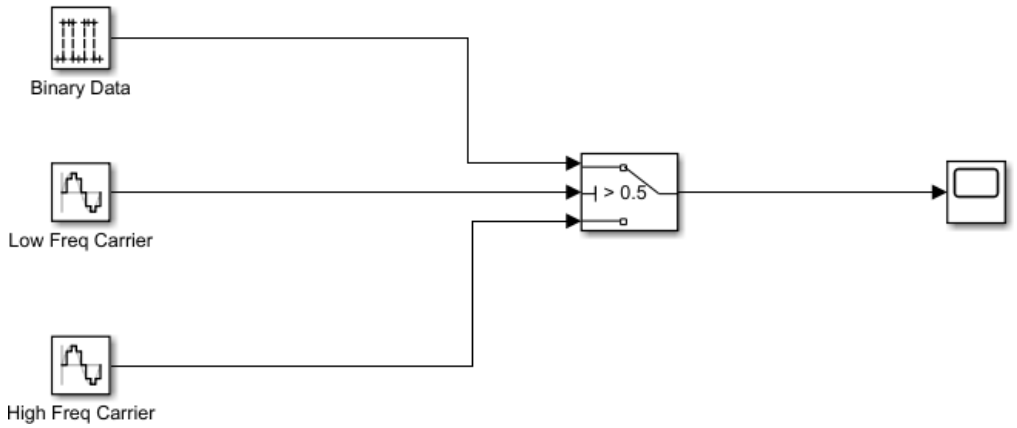

Figure 4.2.1: FSK Simulink model

FSK Simulink Result:

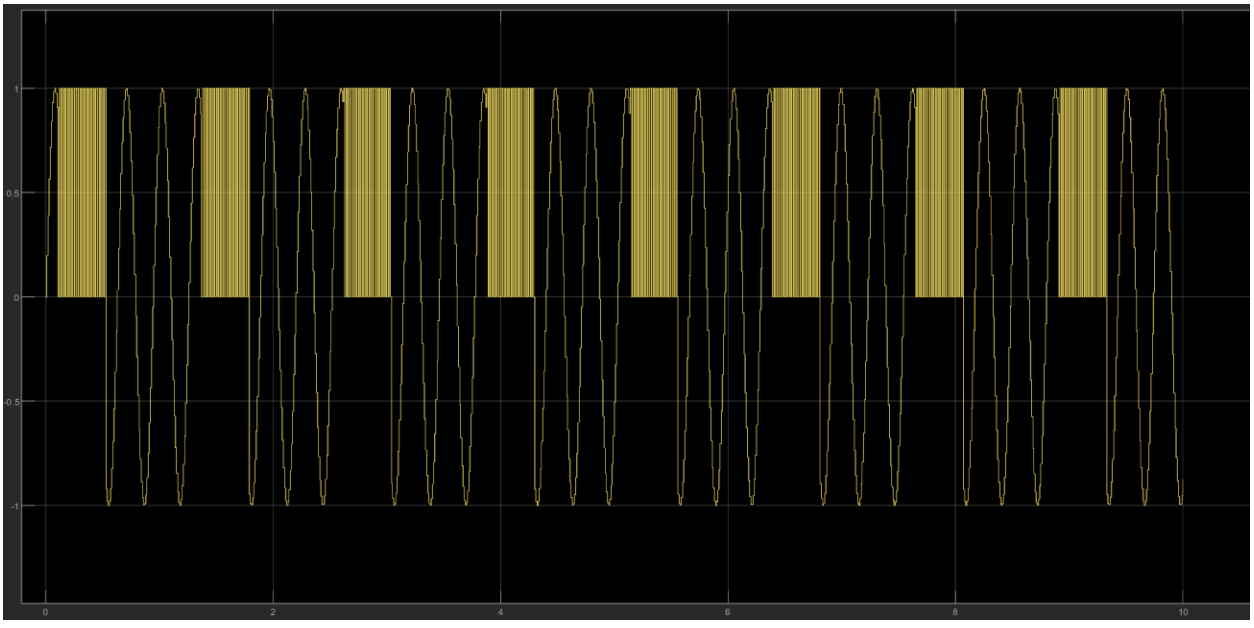

Figure 4.2.2: FSK Simulink output

4.3 PSK Simulink Model:

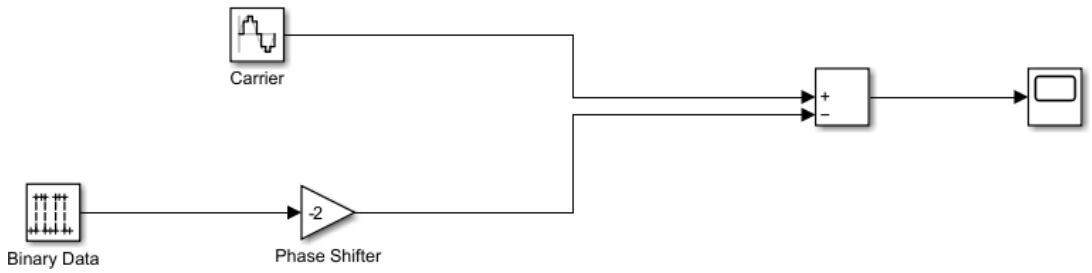

Figure 4.3.1: PSK Simulink model

PSK Simulink Result:

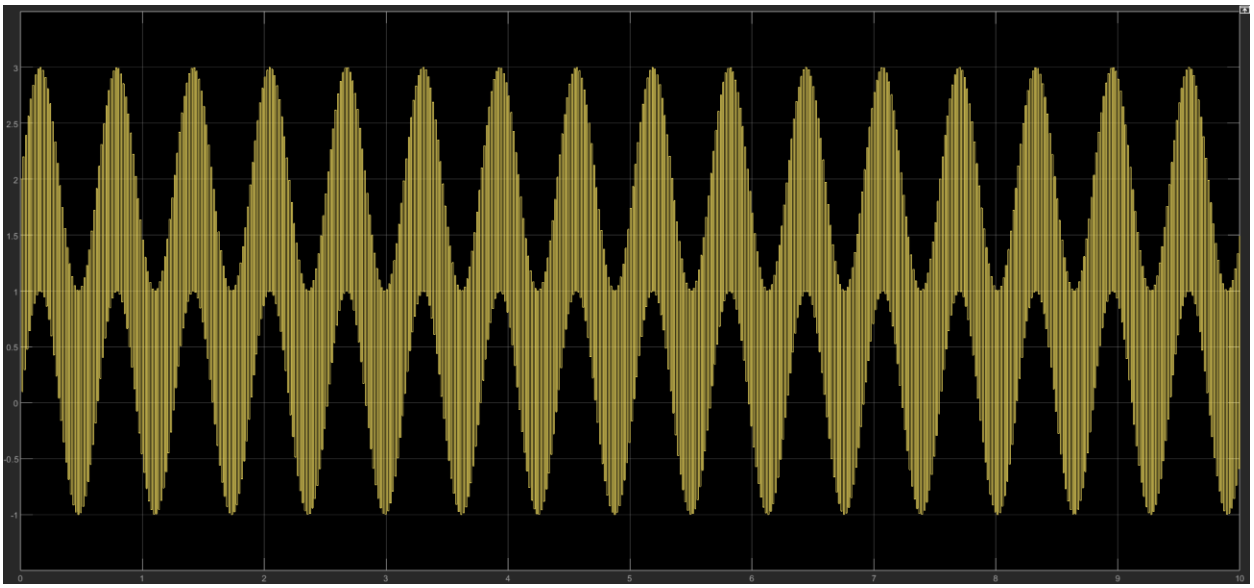

Figure 4.3.2: PSK Simulink output

4.4 ASK

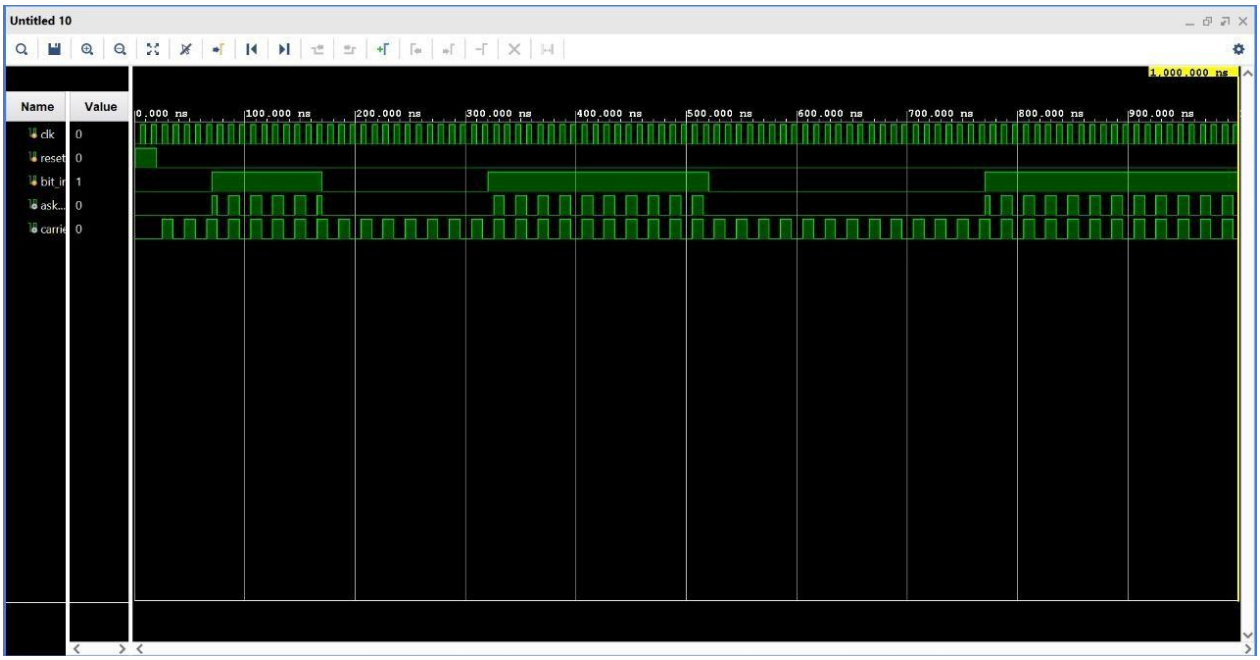

Figure 4.4.1: ASK Simulation output

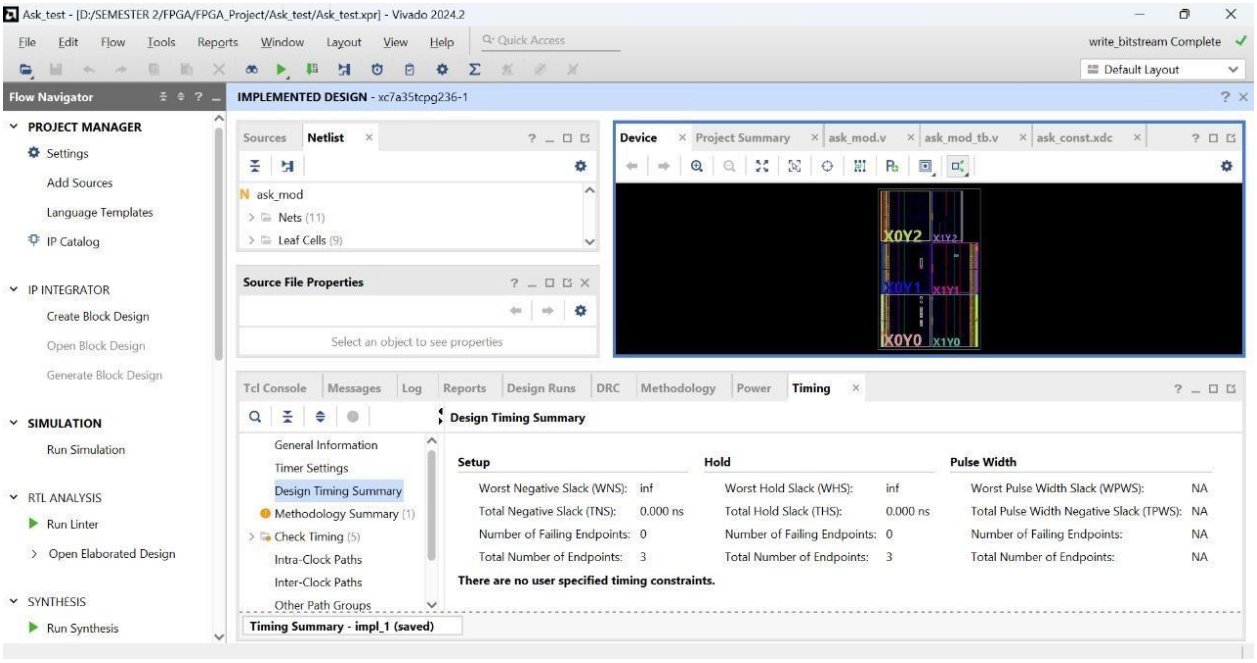

Figure 4.4.2: ASK Bitstream Generation

Simulation:

- The ASK modulator was simulated using behavioral simulation in Vivado, and the testbench provided predefined bit_in bits while observing the carrier and the modulated ASK output (ask_out).
- The carrier signal was generated as a square wave toggling on every positive edge (50 MHz clock).
- The ASK modulated signal was following the carrier signal when the bit_in=1 and showed zero amplitude when bit_in=0.

Hardware:

1. Internal carrier using logic in Verilog code:

- The carrier signal was a continuous toggling signal. Carrier_out and ask_out were mapped to the Pmod pins JA3 and JA2.
- The input was controlled by the switch (SW0), but the output was not obtained in the DSO; rather, it got noise.
- The ASK output should show zero amplitude when the bit_in=0 and follow the carrier signal when bit_in=1.

2. External carrier using WaveGen (DSO):

- Carrier signal was generated using the WaveGen function in the DSO.
- Waveform: Square
- Frequency=1MHz
- Voltage=3.3V
- Carrier was fed into the basys3 board through Pmod pin JA1 and the verilog code was modified to accept the external carrier input
- The ASK modulated output can be observed at Pmod pin JA2 using DSO, but did not get the output, rather got more noise.

4.5 FSK

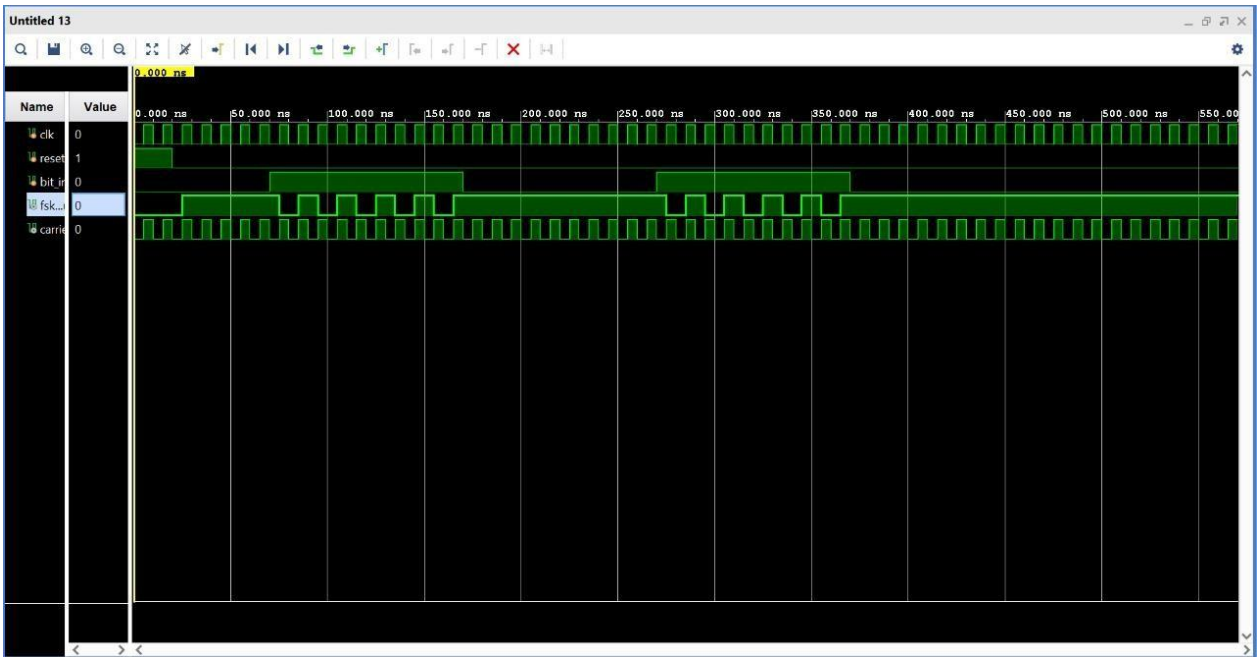

Figure 4.5.1: FSK Simulation output

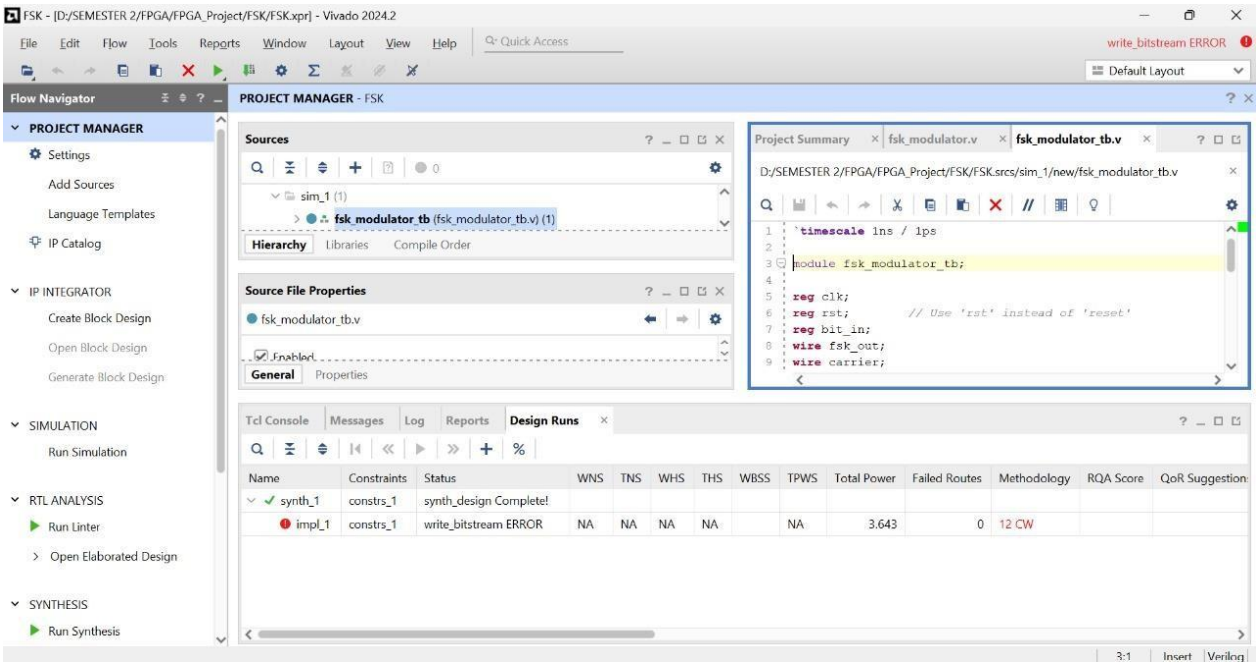

Figure 4.5.2: FSK Bitstream Generation

Simulation:

- The FSK modulator was simulated using behavioral simulation in Vivado, and the testbench provided a predefined bit_in bits while observing the carrier and the modulated FSK output (fsk_out).
- The carrier signal was generated as a square wave toggling on every positive edge (50 MHz clock).
- The FSK-modulated signal had a high frequency (40 MHz) when the bit_in was bit_in=1 and had a low frequency (20 MHz) when bit_in=0.
- But the bitstream generation failed.

4.6 BPSK

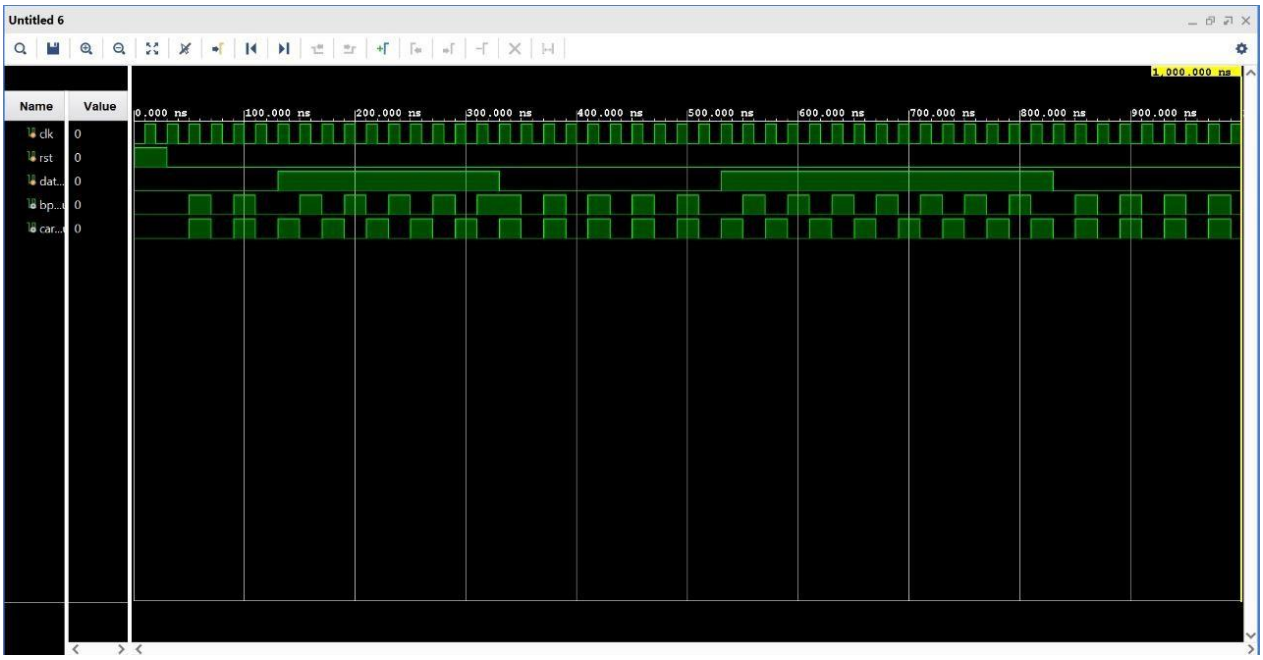

Figure 4.6.1: BPSK Simulation output

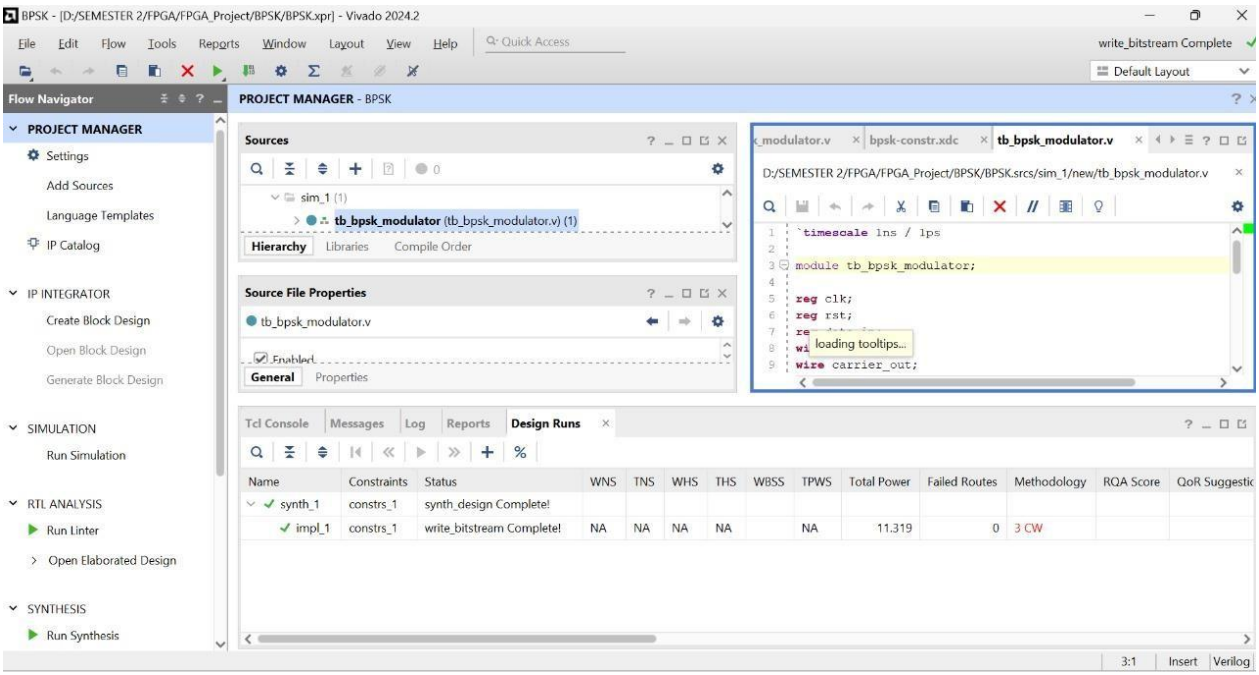

Figure 4.6.2: BPSK Bitstream Generation

Figure 4.6.3: BPSK carrier signal using WaveGen

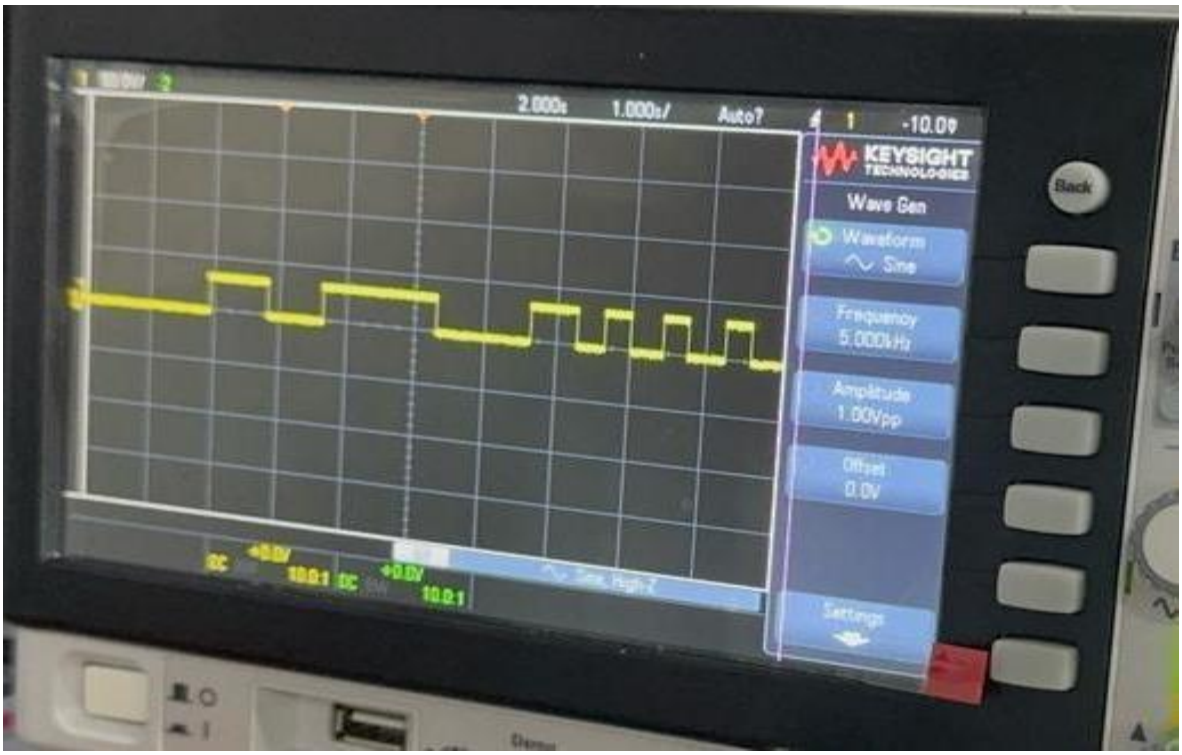

Figure 4.6.4: BPSK modulated output

Simulation:

- The BPSK modulator was simulated using behavioral simulation in Vivado, and the testbench provided predefined data_in bits while observing the carrier and the modulated BPSK output (bpsk_out).
- The carrier signal was generated as a square wave toggling on every positive edge (50 MHz clock).
- The BPSK modulated signal was in phase with the carrier signal when data_in=0 and showed a 180-degree phase shift when data_in=1.

Hardware:

1. Internal carrier using logic in Verilog code:

- The carrier signal was a continuous toggling signal. Carrier_out and bpsk_out were mapped to the Pmod pins JA3 and JA2.
- The input was controlled by the switch (sw0), and the output was obtained in the DSO.
- The BPSK output showed a 180-degree phase shift when the data_in=1 and in phase with the phase when data_in=0.

2. External carrier using WaveGen (DSO):

- The carrier signal was generated using the WaveGen function in the DSO.
- Waveform: Square
- Frequency=1MHz
- Voltage=3.3V
- Carrier was fed into the basys3 board through Pmod pin JA1 and the verilog code was modified to accept the external carrier input
- The BPSK modulated output was observed at the Pmod pin JA2 using DSO.

CHAPTER-5
CONCLUSION

BPSK, ASK and FSK modulation techniques were successfully implemented and tested BPSK and ASK in Basys 3 FPGA board using Verilog. Simulations in Vivado confirmed the correct functionality and output waveforms observed on the DSO which matched the expected results. Each modulation technique clearly showed its unique characteristics i.e. phase shift in BPSK, amplitude change in ASK and frequency variation in FSK which demonstrated a good understanding of the digital modulation techniques and their practical realization on the FPGA board.

REFERENCES

1. Al-Safi, A.; and Bazuin, B. (2017). Toward digital transmitters with amplitude shift keying and quadrature amplitude modulators implementation examples. Proceedings of the 7th IEEE Annual Computing and Communication Workshop and Conference, Las Vegas, USA.

2. Shahana, K.; and Parikh, K.S. (2016). FPGA design and implementation of selectable M-PSK modulators. International Journal of Engineering Research and Reviews.

3. Jammu, B.R.; Botcha, H.K.; Sowjanya, A.V.; and Bodasingi, N. (2017). FPGA implementation of BASK-BFSK-BPSK-DPSK digital modulators using system generators. Proceedings of the International Conference on circuits Power and Computing Technologies [ICCPCT], Kollam, India.

4. Muthalagu, R.; Sudheer, R.; and Ibrahim, S. (2018). FPGA implementation of optimized QPSK and OQPSK Using VHDL. Journal of Communications**.**