Final Project

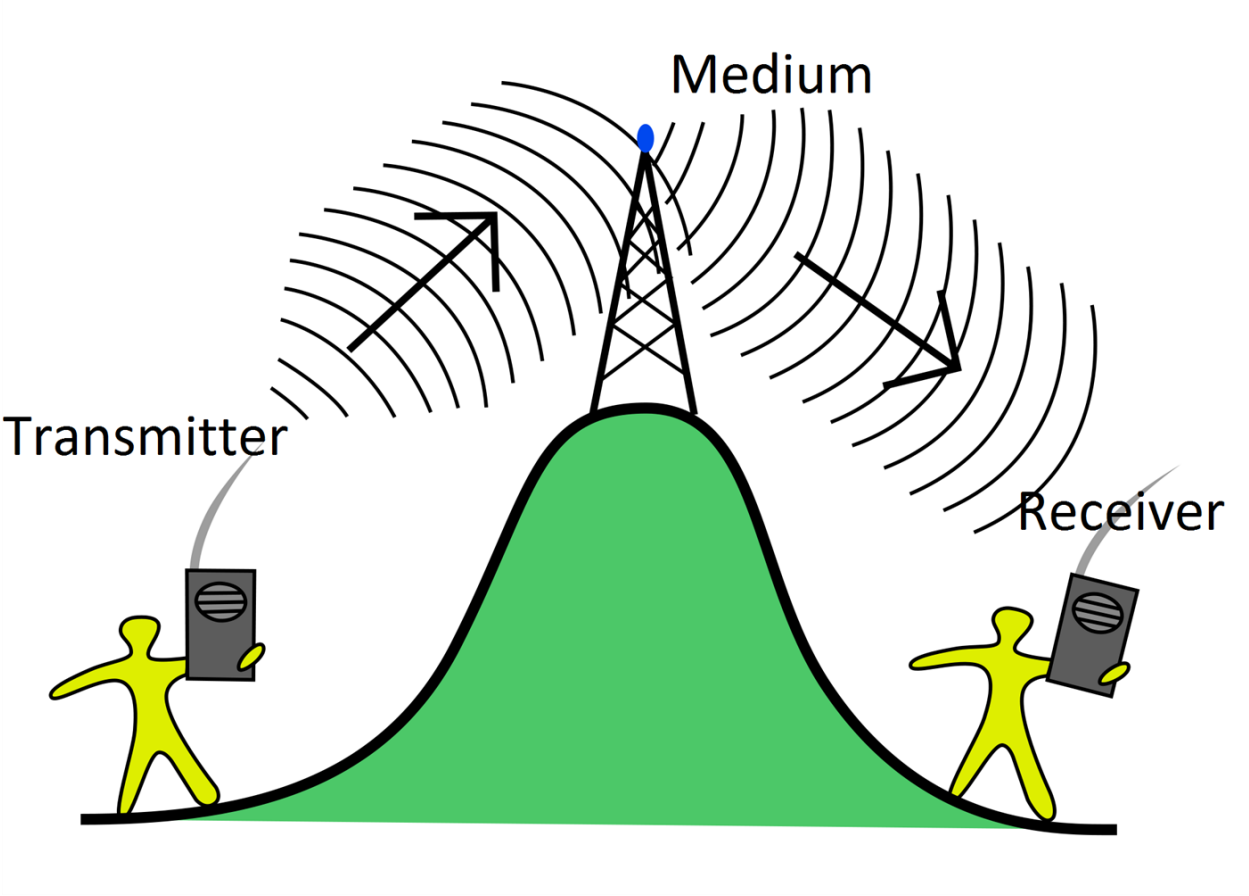
Title: RTL Implementation of Differential Phase Shift Keying (DPSK) Modulation and Demodulation

#Prepared by:

1-Soliman Yehia Soliman Mahmoud

2-Mousa Mohamed El-Sayed Gaballah

communication system:



What is Modulation?:

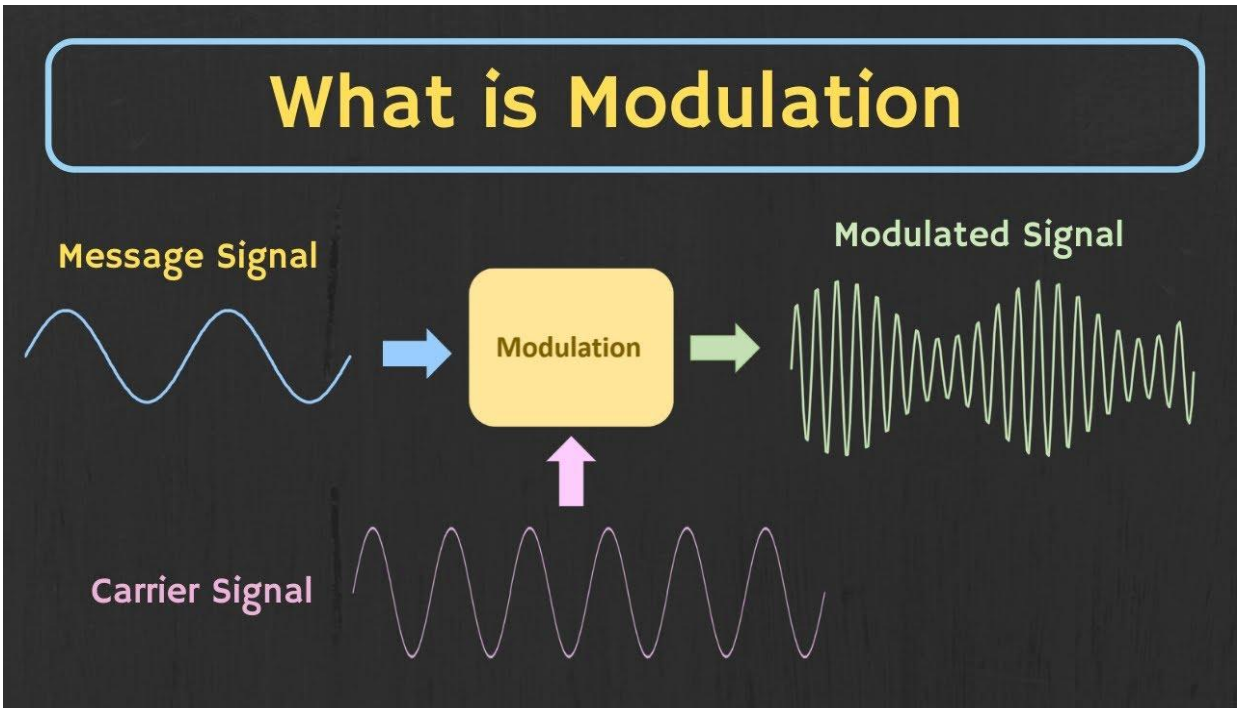
- **Modulation** is the process of varying one or more properties (amplitude, frequency, or phase) of a carrier signal in accordance with the information (data) to be transmitted. The purpose of modulation is to encode the information into a form that can be transmitted over a communication channel efficiently and reliably.

-Why Use Modulation?

1-Transmitting signals over long distances.

2-Reducing interference between channels.

3-Noise Immunity.



What is Demodulation?

Demodulation is the reverse process of modulation. It extracts the original information from the received signal after transmission through the channel.

What is Phase Shift Keying (PSK)?

Phase Shift Keying (PSK) is a digital modulation technique where the phase of the carrier wave is changed according to the binary data (0 and 1), while the amplitude and frequency remain constant.

-Problems with PSK:

1- Complexity of Coherent Detection.

2- Sensitivity to Phase Noise.

3- Requirement for Phase Coherence.

The Solution: Using DPSK (Differential PSK).

What is DPSK?

Differential Phase Shift Keying (DPSK) is an improvement over PSK, where data is encoded based on the difference between the current phase and the previous phase, rather than using an absolute phase reference.

How Does DPSK Work?

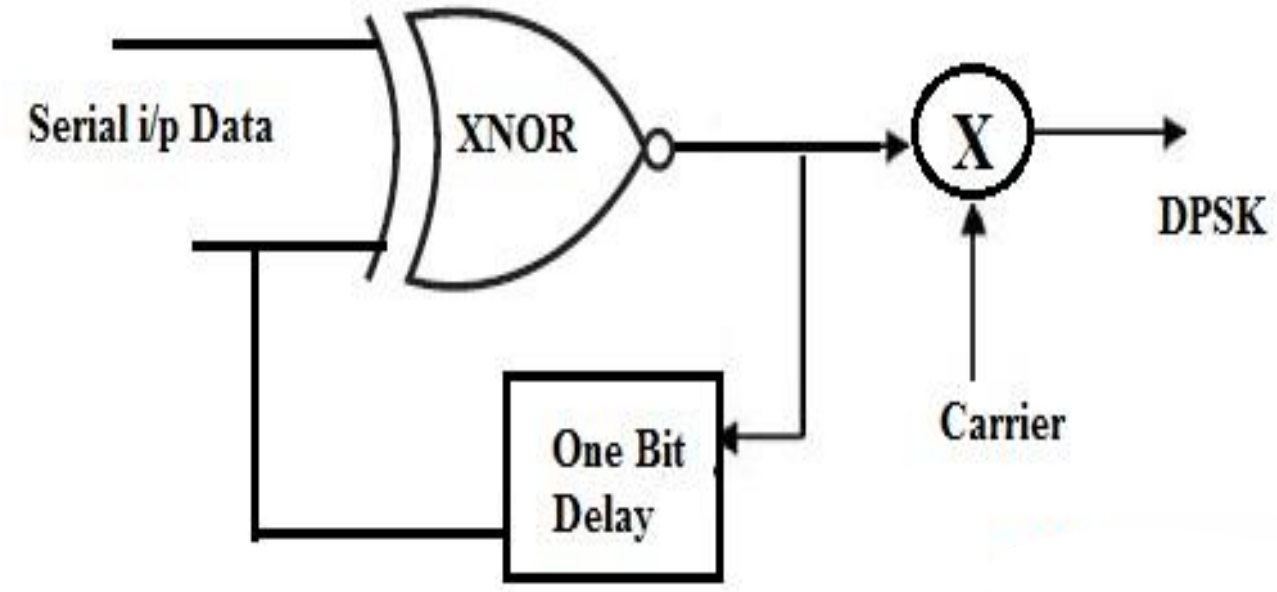
For the same binary sequence **1011**, DPSK works as follows:

- Initial phase: Assume 0°.
- **Encoding rule:**
  If the current bit is the same as the previous bit, there is no phase change (0 degrees).
  If the current bit is different from the previous bit, reverse the phase (180 degrees).

- **Practical Example of DPSK Modulation:**

BIT	PHASE CHANGE	TRANSMITTED PHASE
1	No change	0°
0	Change	180°
1	Change	0°
1	No change	0°

The block diagram of modulation :



The code of modulation.

1- Bit Delay Module:

```
//   (Stores the Previous Bit)
// ----------------------------------
module bit_delay (
   input wire clk,
   input wire rst,
   input wire serial_data_in,  // Current input serial bit
   output reg serial_data_out  // Delayed (previous) serial bit
);
   always @(posedge clk or posedge rst) begin
      if (rst)
         serial_data_out <= 1'b1;  // Initialize to `1` to avoid phase errors
      else
         serial_data_out <= serial_data_in; // Store the current bit for the next cycle
   end
endmodule
```

2- XNOR Logic Module:

```
// XNOR Logic Module (Computes DPSK Bit)
// ----------------------------------
module xnor_logic (
   input wire serial_data,   // Current serial data bit
   input wire delayed_data,  // Previous serial data bit (delayed)
   output wire dpsk_bit      // DPSK bit output
);
   assign dpsk_bit = ~(serial_data ^ delayed_data);  // XNOR operation
endmodule
```

3-Balanced Modulator (Phase Shift Modulation):

```
// Balanced Modulator (Phase Shift Modulation)
// ----------------------------------
module balanced_modulator (
   input wire clk,
   input wire rst,
   input wire dpsk_bit,          // DPSK bit after XNOR encoding
   input signed [15:0] carrier,   // Carrier signal
   output signed [15:0] dpsk_signal // DPSK modulated signal
);

   reg signed [15:0] dpsk_signal_previous; // Store the previous DPSK signal

   always @(posedge clk or posedge rst) begin
      if (rst)
         dpsk_signal_previous <= carrier; // Initialize to the carrier signal
      else
         dpsk_signal_previous <= dpsk_signal; // Store the previous modulated signal
   end

   // If dpsk_bit = 1, keep the same signal; if dpsk_bit = 0, invert the previous signal
   assign dpsk_signal = (dpsk_bit) ? dpsk_signal_previous : -dpsk_signal_previous;

endmodule
```

4-DPSK Modulator (Top Module):

```
/ DPSK Modulator (Top Module)
// ----------------------------------
module dpsk_modulator (
   input wire clk,
   input wire rst,
   input wire serial_data,        // Input serial data
   input signed [15:0] carrier,    // Carrier signal
   output signed [15:0] dpsk_out   // DPSK modulated output signal
);
   wire delayed_bit, dpsk_bit;

   // Bit Delay Unit (Stores previous serial data bit)
   bit_delay delay_unit (
      .clk(clk),
      .rst(rst),
      .serial_data_in(serial_data),
      .serial_data_out(delayed_bit)
   );

   // XNOR Logic to compute DPSK bit
   xnor_logic xnor_mod (
      .serial_data(serial_data),
      .delayed_data(delayed_bit),
      .dpsk_bit(dpsk_bit)
   );

   // Balanced Modulator to generate DPSK output
   balanced_modulator modulator (
      .clk(clk),
      .rst(rst),
      .carrier(carrier),
      .dpsk_bit(dpsk_bit),
      .dpsk_signal(dpsk_out)
   );
endmodule
```

The testbench of modulation:

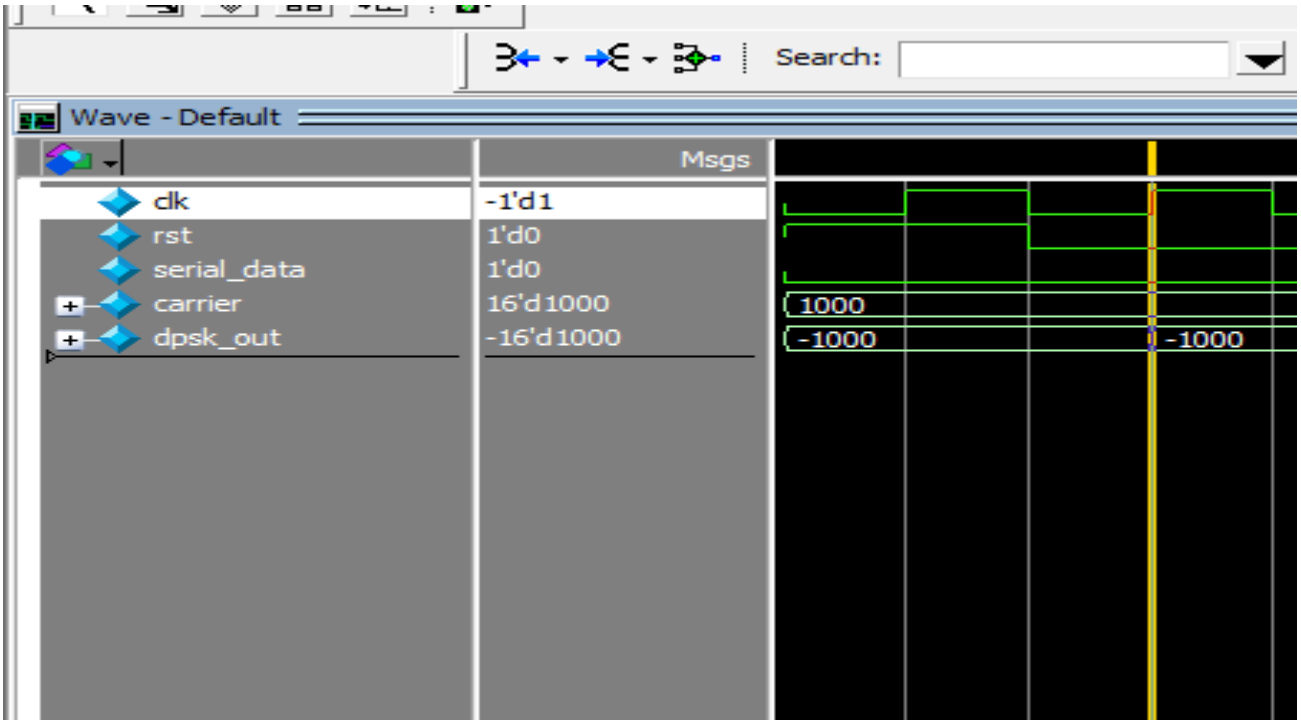
```
// DPSK Modulator Testbench
module dpsk_tb;
   reg clk, rst;
   reg serial_data;
   reg signed [15:0] carrier;
   wire signed [15:0] dpsk_out;
   // Instantiate the DPSK modulator
   dpsk_modulator uut (
      .clk(clk),
      .rst(rst),
      .serial_data(serial_data),
      .carrier(carrier),
      .dpsk_out(dpsk_out)
   );
   // Clock Generation
   initial begin
      clk = 0;
      forever #10 clk = ~clk;
   end
   // Apply Input Test Data
   initial begin
      rst = 1;
      serial_data = 0;
      carrier = 16'sd1000;
      #20 rst = 0; // Release reset
      #20;
      // Test sequence of serial_data inputs
      @(posedge clk); serial_data = 1; // Phase shift
      @(posedge clk); serial_data = 0; // Phase shift
      @(posedge clk); serial_data = 1; // Phase shift
      @(posedge clk); serial_data = 1; // No phase shift
      @(posedge clk); serial_data = 0; // Phase shift
      @(posedge clk); serial_data = 0; // No phase shift
      @(posedge clk); serial_data = 1; // Phase shift
      @(posedge clk); serial_data = 1; // No phase shift
      #500;
      $stop;
   end
   // Monitor Output Signals
   initial begin
      $monitor("Time = %0t | serial_data = %b | delayed_bit = %b | dpsk_bit = %b | dpsk_out = %d",
            $time, serial_data, uut.delay_unit.serial_data_out, uut.xnor_mod.dpsk_bit, dpsk_out);
   end
end module
```

Monitor Output:

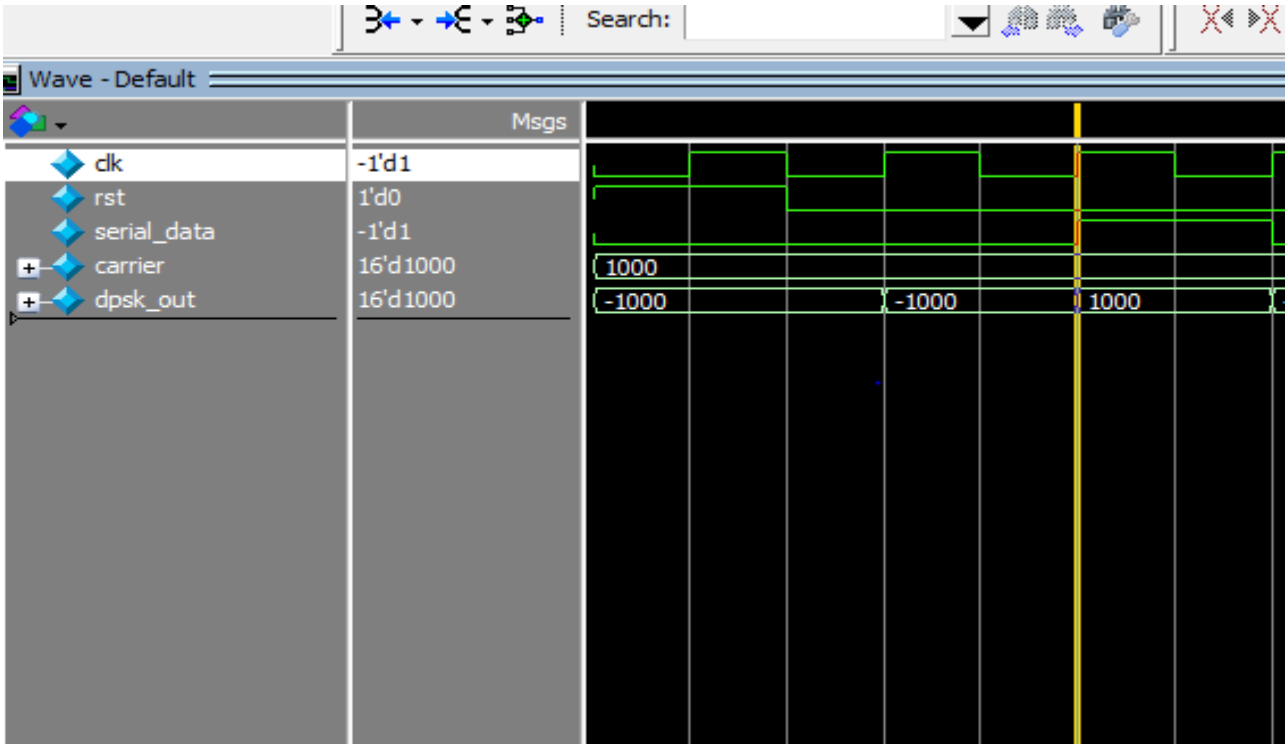
```
Time = 0 | serial_data = 0 | delayed_bit = 1 | dpsk_bit = 0 | dpsk_out =  -1000
Time = 30 | serial_data = 0 | delayed_bit = 0 | dpsk_bit = 1 | dpsk_out =  -1000
Time = 50 | serial_data = 1 | delayed_bit = 0 | dpsk_bit = 0 | dpsk_out =   1000
Time = 70 | serial_data = 0 | delayed_bit = 1 | dpsk_bit = 0 | dpsk_out =  -1000
Time = 90 | serial_data = 1 | delayed_bit = 0 | dpsk_bit = 0 | dpsk_out =   1000
Time = 110 | serial_data = 1 | delayed_bit = 1 | dpsk_bit = 1 | dpsk_out =   1000
Time = 130 | serial_data = 0 | delayed_bit = 1 | dpsk_bit = 0 | dpsk_out =  -1000
Time = 150 | serial_data = 0 | delayed_bit = 0 | dpsk_bit = 1 | dpsk_out =  -1000
Time = 170 | serial_data = 1 | delayed_bit = 0 | dpsk_bit = 0 | dpsk_out =   1000
Time = 190 | serial_data = 1 | delayed_bit = 1 | dpsk_bit = 1 | dpsk_out =   1000
** Note: $stop    : E:/digital ic/ORANGE/pro/mod.v(116)
   Time: 690 ns  Iteration: 0  Instance: /ddfffpsk_tb
Break in Module ddfffpsk_tb at E:/digital ic/ORANGE/pro/mod.v line 116
```

the wave :

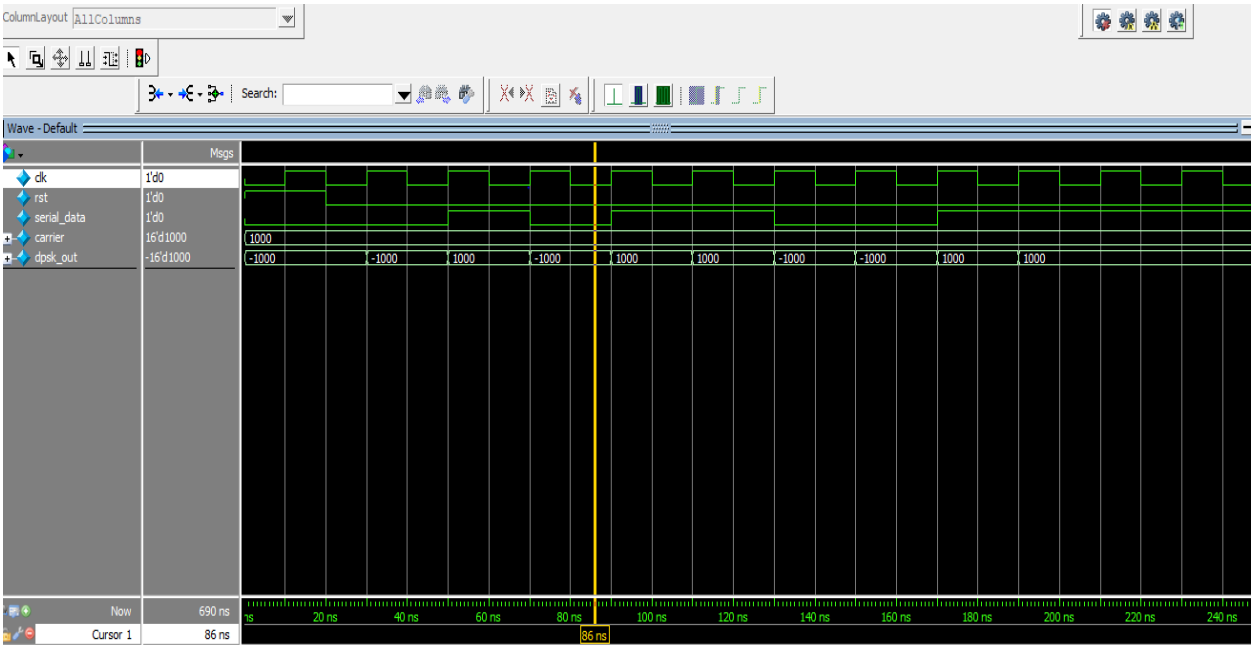
//test 1(no change)



//test 2(change)



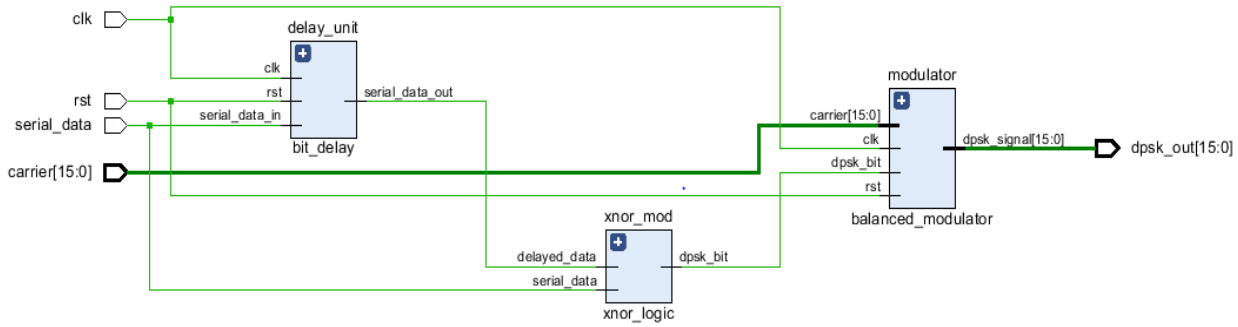
//test 3(all wave)



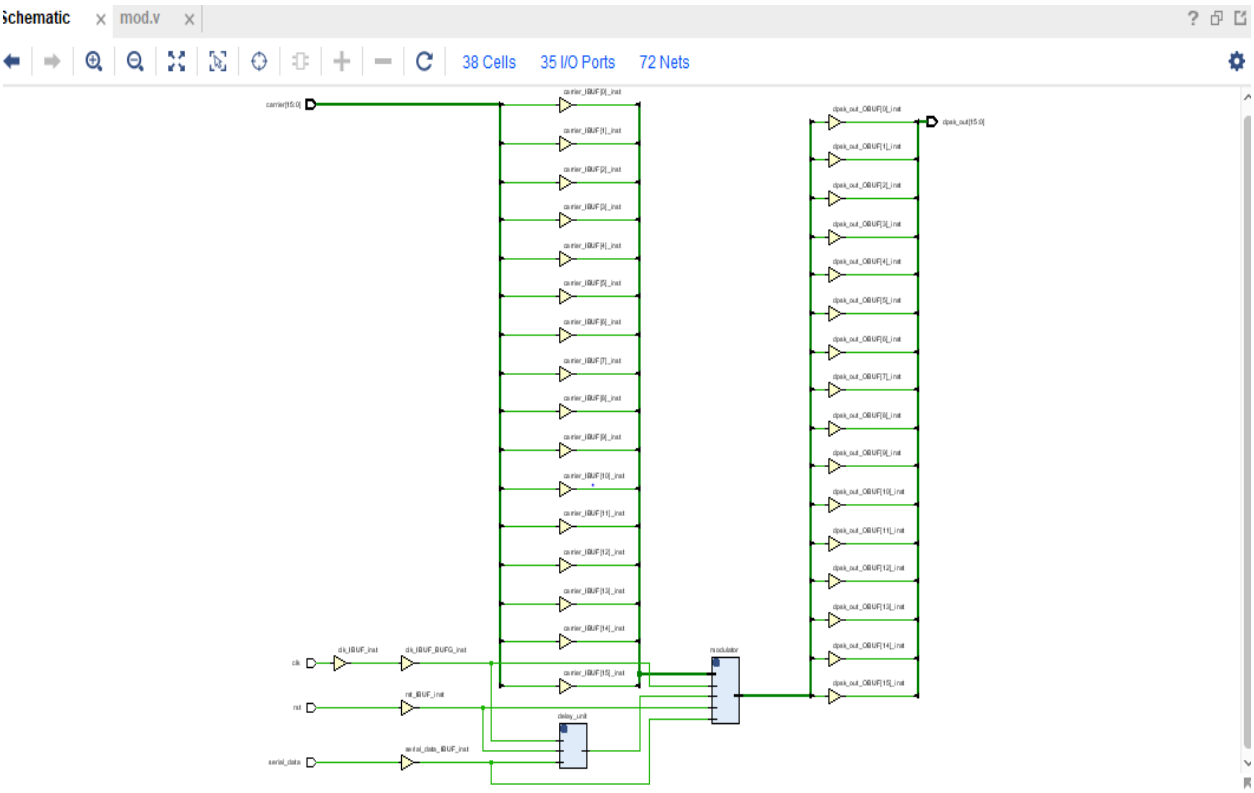
Constraint:

```
## Clock signal
set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports clk]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]

## Switches
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports serial_data]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports {carrier[0]}]
set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS33 } [get_ports {carrier[1]}]
set_property -dict { PACKAGE_PIN W17   IOSTANDARD LVCMOS33 } [get_ports {carrier[2]}]
set_property -dict { PACKAGE_PIN W15   IOSTANDARD LVCMOS33 } [get_ports {carrier[3]}]
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports {carrier[4]}]
set_property -dict { PACKAGE_PIN W14   IOSTANDARD LVCMOS33 } [get_ports {carrier[5]}]
set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 } [get_ports {carrier[6]}]
set_property -dict { PACKAGE_PIN V2    IOSTANDARD LVCMOS33 } [get_ports {carrier[7]}]
set_property -dict { PACKAGE_PIN T3    IOSTANDARD LVCMOS33 } [get_ports {carrier[8]}]
set_property -dict { PACKAGE_PIN T2    IOSTANDARD LVCMOS33 } [get_ports {carrier[9]}]
set_property -dict { PACKAGE_PIN R3    IOSTANDARD LVCMOS33 } [get_ports {carrier[10]}]
set_property -dict { PACKAGE_PIN W2    IOSTANDARD LVCMOS33 } [get_ports {carrier[11]}]
set_property -dict { PACKAGE_PIN U1    IOSTANDARD LVCMOS33 } [get_ports {carrier[12]}]
set_property -dict { PACKAGE_PIN T1    IOSTANDARD LVCMOS33 } [get_ports {carrier[13]}]
set_property -dict { PACKAGE_PIN R2    IOSTANDARD LVCMOS33 } [get_ports {carrier[14]}]

## LEDs
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports {dpsk_out[0]}]
set_property -dict { PACKAGE_PIN E19   IOSTANDARD LVCMOS33 } [get_ports {dpsk_out[1]}]
set_property -dict { PACKAGE_PIN U19   IOSTANDARD LVCMOS33 } [get_ports {dpsk_out[2]}]
set_property -dict { PACKAGE_PIN V19   IOSTANDARD LVCMOS33 } [get_ports {dpsk_out[3]}]
set_property -dict { PACKAGE_PIN W18   IOSTANDARD LVCMOS33 } [get_ports {dpsk_out[4]}]
set_property -dict { PACKAGE_PIN U15   IOSTANDARD LVCMOS33 } [get_ports {dpsk_out[5]}]
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports {dpsk_out[6]}]
set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports {dpsk_out[7]}]
set_property -dict { PACKAGE_PIN V13   IOSTANDARD LVCMOS33 } [get_ports {dpsk_out[8]}]
set_property -dict { PACKAGE_PIN V3    IOSTANDARD LVCMOS33 } [get_ports {dpsk_out[9]}]
set_property -dict { PACKAGE_PIN W3    IOSTANDARD LVCMOS33 } [get_ports {dpsk_out[10]}]
set_property -dict { PACKAGE_PIN U3    IOSTANDARD LVCMOS33 } [get_ports {dpsk_out[11]}]
set_property -dict { PACKAGE_PIN P3    IOSTANDARD LVCMOS33 } [get_ports {dpsk_out[12]}]
set_property -dict { PACKAGE_PIN N3    IOSTANDARD LVCMOS33 } [get_ports {dpsk_out[13]}]
set_property -dict { PACKAGE_PIN P1    IOSTANDARD LVCMOS33 } [get_ports {dpsk_out[14]}]
set_property -dict { PACKAGE_PIN L1    IOSTANDARD LVCMOS33 } [get_ports {dpsk_out[15]}]
##Buttons
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports rst]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports {carrier[15]}]
```

schematic after the elaboration:



schematic after the synthesis:



Snippet from the utilization & timing report & after the synthesis:

Name	Slice LUTs (20800)	Slice Registers (41600)	Bonded IOB (106)	BUFGCTRL (32)
dpsk_modulator	64	49	35	1
delay_unit (bit_delay)	0	1	0	0
modulator (balanced_...	64	48	0	0

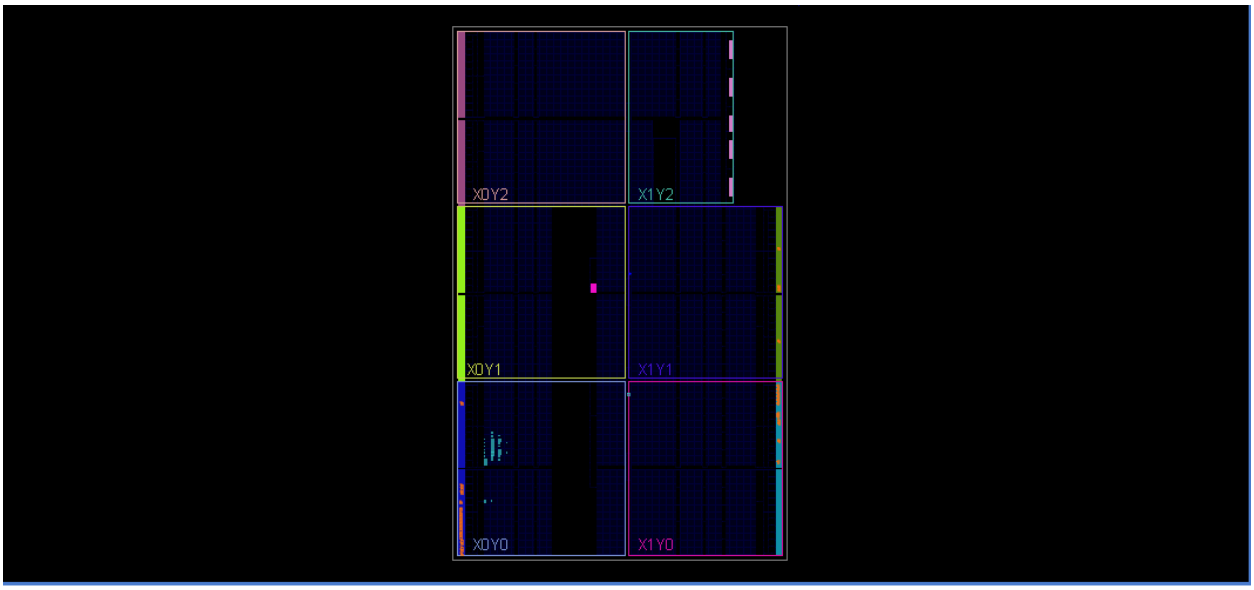
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.056 ns	Worst Hold Slack (WHS):	0.139 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0

Snippet from the device & timing report after the implementation:

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.483 ns	Worst Hold Slack (WHS):	0.179 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	32	Total Number of Endpoints:	32	Total Number of Endpoints:	34



The code of demodulation.

1-Bit Delay Module

```
// Bit Delay Module (Stores the Previous DPSK Bit)
// ----------------------------------
module bit_delay (
   input wire clk,
   input wire rst,
   input wire dpsk_bit_in,  // Current extracted DPSK bit
   output reg dpsk_bit_out  // Delayed (previous) DPSK bit
);
   always @(posedge clk or posedge rst) begin
      if (rst)
         dpsk_bit_out <= 1'b1;  // Initialize to `1` to ensure correct demodulation
      else
         dpsk_bit_out <= dpsk_bit_in; // Store the current DPSK bit for next cycle
   end
endmodule
```

2-DPSK Bit Extraction Module (Converts Signal to Digital Bit):

```
// DPSK Bit Extraction Module (Converts Signal to Digital Bit)
// ----------------------------------
module dpsk_bit_extractor (
   input signed [15:0] dpsk_signal, // Received DPSK modulated signal
   output wire dpsk_bit           // Extracted DPSK bit
);
   assign dpsk_bit = (dpsk_signal >= 0) ? 1'b1 : 1'b0;
endmodule
```

3-XNOR Logic Module (Recovers Original Data):

```
// XNOR Logic Module (Recovers Original Data)
// ----------------------------------
module xnor_logic (
   input wire current_dpsk_bit,   // Current extracted DPSK bit
   input wire previous_dpsk_bit,  // Previous DPSK bit (delayed)
   output wire recovered_data     // Recovered original data
);
   assign recovered_data = ~(current_dpsk_bit ^ previous_dpsk_bit); // XNOR operation
endmodule
```

4-Top Module for DPSK Demodulation

```
// DPSK Demodulator Module (Top Module for DPSK Demodulation)
// ----------------------------------
module dpsk_demodulator (
   input wire clk,
   input wire rst,
   input signed [15:0] dpsk_signal, // Received DPSK modulated signal
   output wire recovered_data       // Output recovered data
);
   wire dpsk_bit, delayed_dpsk_bit;
   // Store previous DPSK bit for comparison
   bit_delay delay_unit (
      .clk(clk),
      .rst(rst),
      .dpsk_bit_in(dpsk_bit),
      .dpsk_bit_out(delayed_dpsk_bit)
   );
    // Extract DPSK digital bit from the modulated signal
   dpsk_bit_extractor extractor (
      .dpsk_signal(dpsk_signal),
      .dpsk_bit(dpsk_bit)
   );

   // Recover original data using XNOR
   xnor_logic xnor_mod (
      .current_dpsk_bit(dpsk_bit),
      .previous_dpsk_bit(delayed_dpsk_bit),
      .recovered_data(recovered_data)
   );

endmodule
```

The testbench of demodulation:

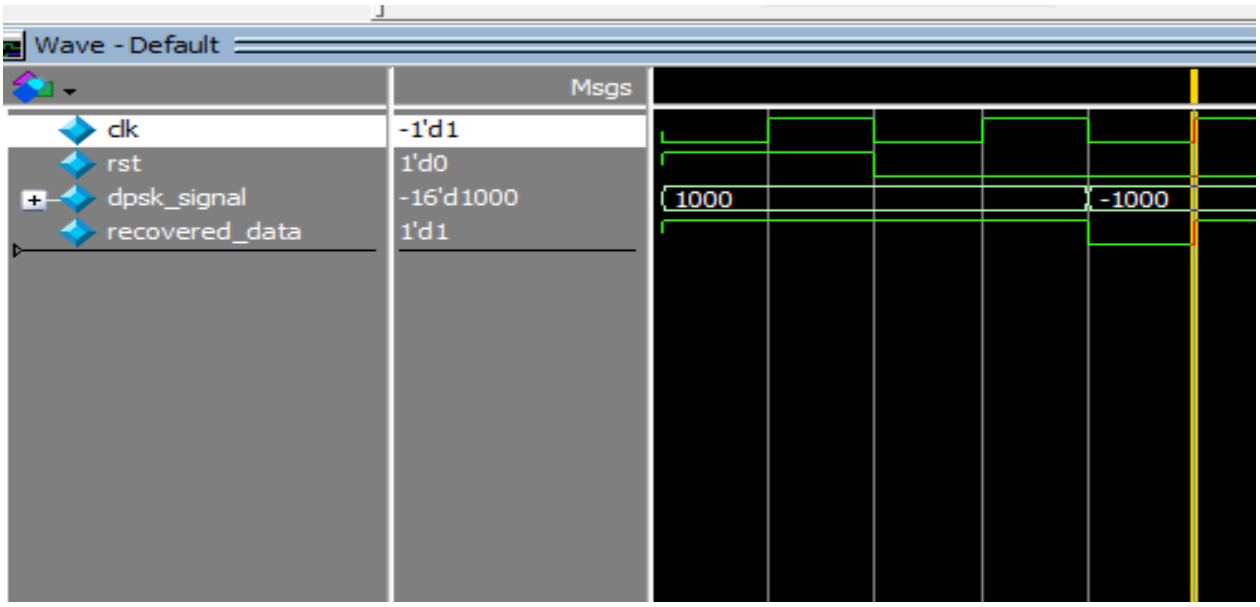
```
// DPSK Demodulation Testbench
// ----------------------------------
module dpsk_demodulator_tb;
   reg clk, rst;
   reg signed [15:0] dpsk_signal;
   wire recovered_data;

   // Instantiate the DPSK Demodulator
   dpsk_demodulator uut (
      .clk(clk),
      .rst(rst),
      .dpsk_signal(dpsk_signal),
      .recovered_data(recovered_data)
   );
   // Clock Generation
   initial begin
      clk = 0;
      forever #10 clk = ~clk; // 50MHz clock (20ns period)
   end

   // Apply Test Data
   initial begin
      rst = 1;
      dpsk_signal = 16'sd1000; // Initial signal
      #20 rst = 0; // Release reset
      // Simulated DPSK signal transitions
      @(negedge clk); dpsk_signal = 16'sd1000;  // No phase change (1)
      @(negedge clk); dpsk_signal = -16'sd1000; // Phase shift (0)
      @(negedge clk); dpsk_signal = 16'sd1000;  // Phase shift (1)
      @(negedge clk); dpsk_signal = 16'sd1000;  // No phase change (1)
      @(negedge clk); dpsk_signal = -16'sd1000; // Phase shift (0)
      @(negedge clk); dpsk_signal = -16'sd1000; // No phase change (0)
      @(negedge clk); dpsk_signal = 16'sd1000;  // Phase shift (1)
      @(negedge clk); dpsk_signal = 16'sd1000;  // No phase change (1)
      #500;
      $stop;
   end
   // Monitor Output Signals
   initial begin
      $monitor("Time = %0t | dpsk_signal = %d | recovered_data = %b",
            $time, dpsk_signal, recovered_data);
   end
endmodule
```

Monitor Output:

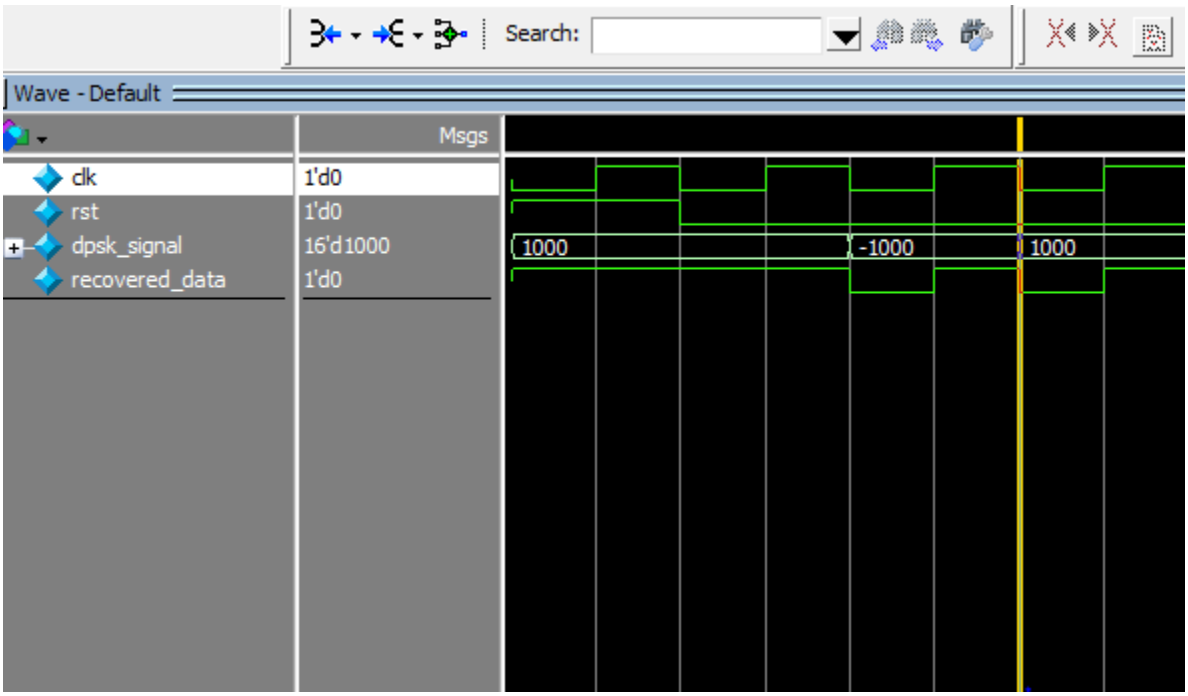
```
dd wave -position insertpoint sim:/dpsk_demodulator_tb/*
SIM 15> run -all
 Time = 0 | dpsk_signal =   1000 | recovered_data = 1
 Time = 40 | dpsk_signal =  -1000 | recovered_data = 0
 Time = 50 | dpsk_signal =  -1000 | recovered_data = 1
 Time = 60 | dpsk_signal =   1000 | recovered_data = 0
 Time = 70 | dpsk_signal =   1000 | recovered_data = 1
 Time = 100 | dpsk_signal =  -1000 | recovered_data = 0
 Time = 110 | dpsk_signal =  -1000 | recovered_data = 1
 Time = 140 | dpsk_signal =   1000 | recovered_data = 0
 Time = 150 | dpsk_signal =   1000 | recovered_data = 1
 ** Note: $stop    : E:/digital ic/ORANGE/pro/demo.v(113)
    Time: 660 ns  Iteration: 0  Instance: /dpsk_demodulator_tb
```

the wave :

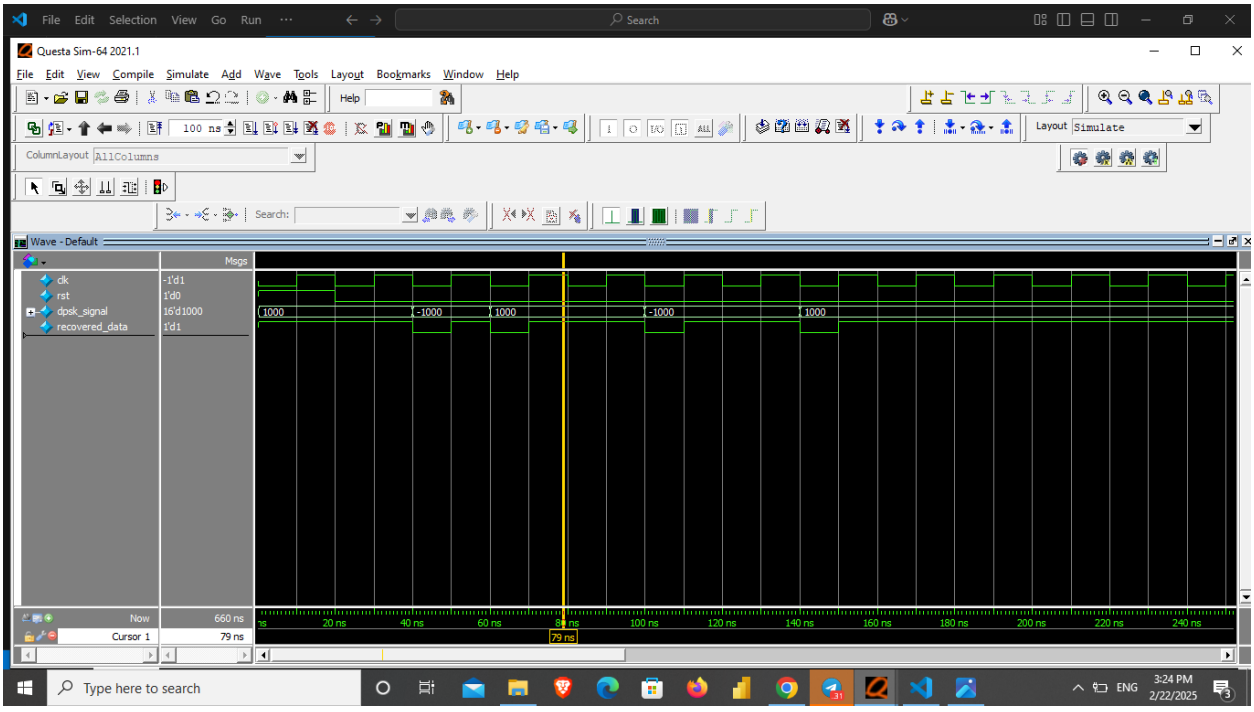
//test 1(no change)



//test 2(change)



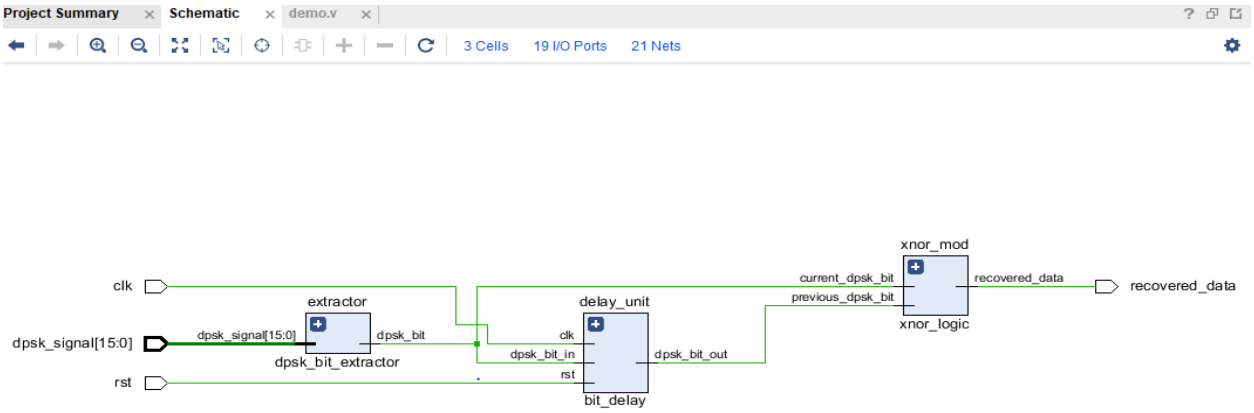
//test 3(all wave)



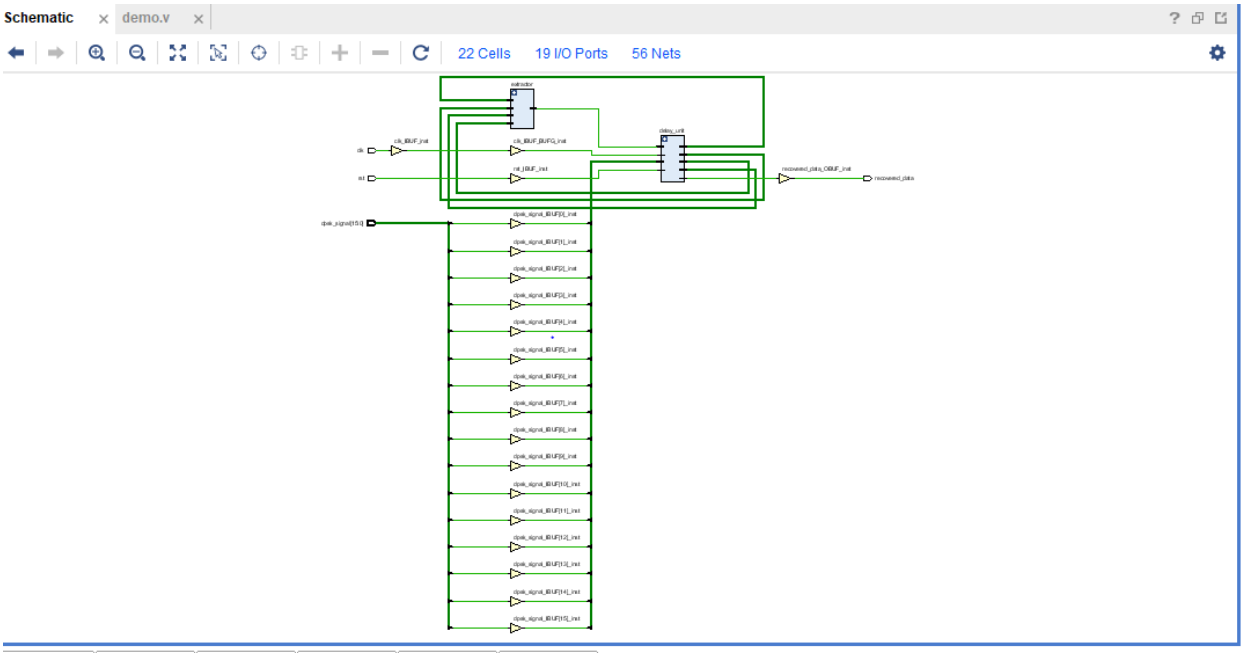
Constraint:

```
## Clock signal
set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports clk]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]
## Switches
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports {dpsk_signal[0]}]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports {dpsk_signal[1]}]
set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS33 } [get_ports {dpsk_signal[2]}]
set_property -dict { PACKAGE_PIN W17   IOSTANDARD LVCMOS33 } [get_ports {dpsk_signal[3]}]
set_property -dict { PACKAGE_PIN W15   IOSTANDARD LVCMOS33 } [get_ports {dpsk_signal[4]}]
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports {dpsk_signal[5]}]
set_property -dict { PACKAGE_PIN W14   IOSTANDARD LVCMOS33 } [get_ports {dpsk_signal[6]}]
set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 } [get_ports {dpsk_signal[7]}]
set_property -dict { PACKAGE_PIN V2    IOSTANDARD LVCMOS33 } [get_ports {dpsk_signal[8]}]
set_property -dict { PACKAGE_PIN T3    IOSTANDARD LVCMOS33 } [get_ports {dpsk_signal[9]}]
set_property -dict { PACKAGE_PIN T2    IOSTANDARD LVCMOS33 } [get_ports {dpsk_signal[10]}]
set_property -dict { PACKAGE_PIN R3    IOSTANDARD LVCMOS33 } [get_ports {dpsk_signal[11]}]
set_property -dict { PACKAGE_PIN W2    IOSTANDARD LVCMOS33 } [get_ports {dpsk_signal[12]}]
set_property -dict { PACKAGE_PIN U1    IOSTANDARD LVCMOS33 } [get_ports {dpsk_signal[13]}]
set_property -dict { PACKAGE_PIN T1    IOSTANDARD LVCMOS33 } [get_ports {dpsk_signal[14]}]
set_property -dict { PACKAGE_PIN R2    IOSTANDARD LVCMOS33 } [get_ports {dpsk_signal[15]}]
## LEDs
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports recovered_data ]
##Buttons
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports rst]
```

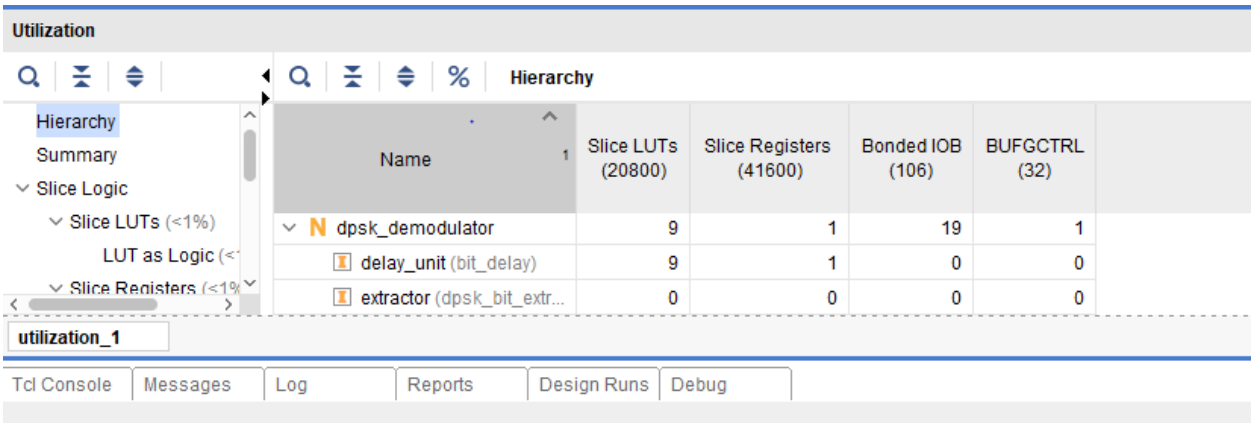
schematic after the elaboration:

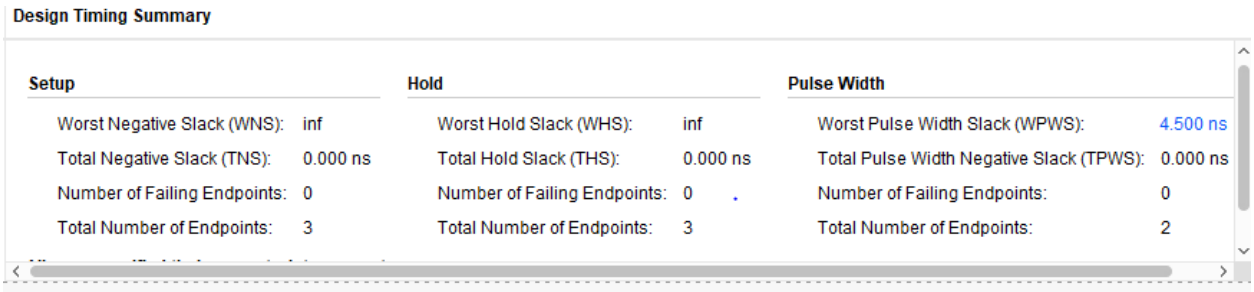


schematic after the synthesis:



Snippet from the utilization & timing report after the synthesis:





Snippet from the device & timing report after the implementation: