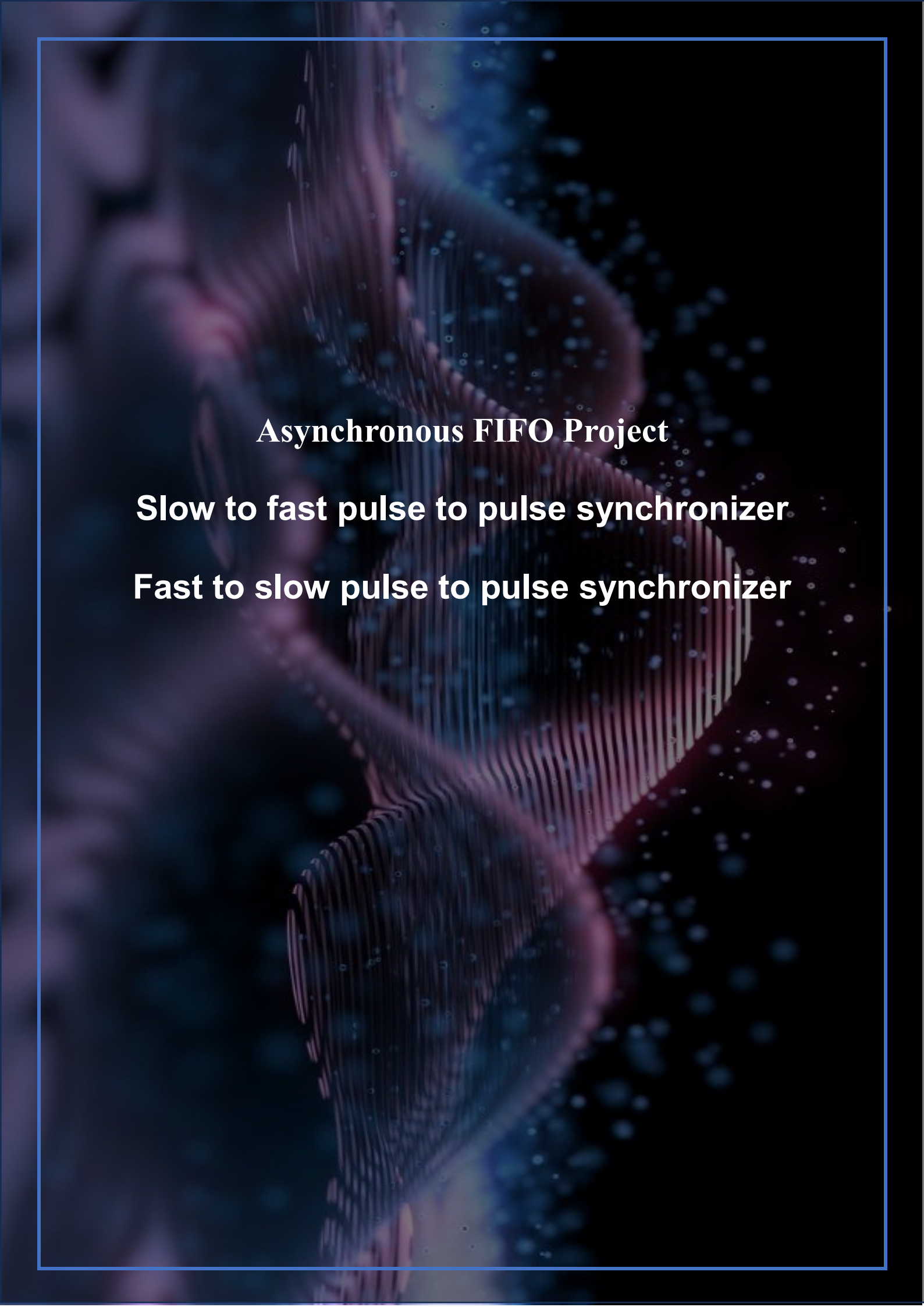
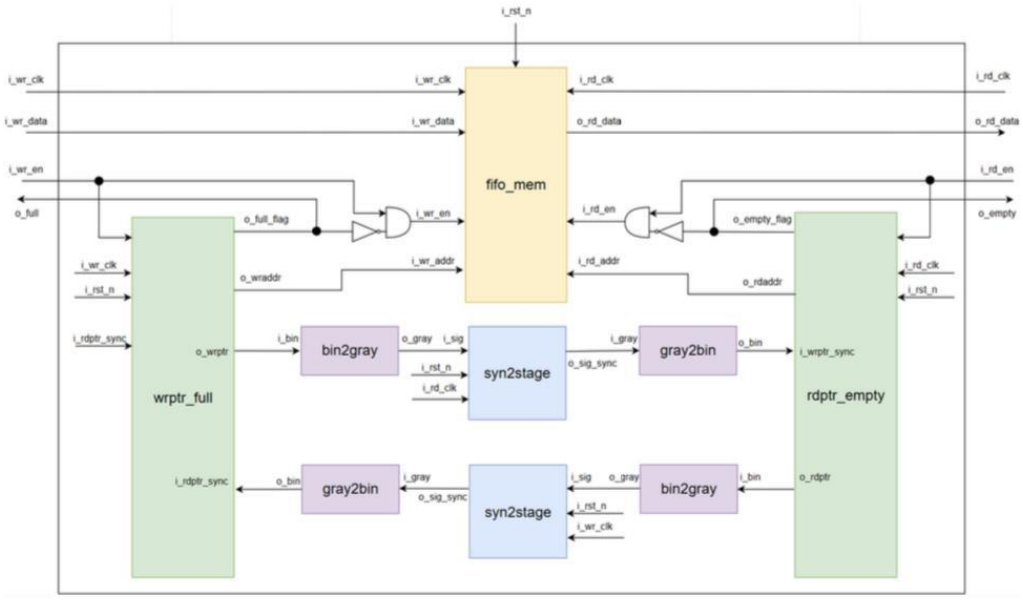

Asynchronous FIFO Project
Slow to fast pulse to pulse synchronizer
Fast to slow pulse to pulse synchronizer

1.0 INTRODUCTION

Asynchronous FIFO

Block Diagram



1.1 TWO STAGE SYNCHRONIZER FLIP-FLOPS

```
1  module syn2stage #(parameter SIG_WIDTH = 4)(
2  input wire i_clk,
3  input wire i_rst_n,
4  input wire [SIG_WIDTH-1:0] i_sig,
5  output reg [SIG_WIDTH-1:0] o_sig_sync
6  );
7  reg [SIG_WIDTH-1:0] sig_r;
8  always@(posedge i_clk or negedge i_rst_n) begin
9  if(!i_rst_n) begin
10 sig_r <= 3'b0;
11 o_sig_sync <= 3'b0;
12 end else begin
13 sig_r <= i_sig;
14 o_sig_sync <= sig_r;
15 end
16 end
17 endmodule
```

1.2 GRAY TO BINARY

```
1  module gray2bin(
2  input wire [3:0] i_gray,
3  output reg [3:0] o_bin
4  );
5  always@(*) begin
6  case(i_gray)
7  4'b0000: o_bin = 4'b0000;
8  4'b0001: o_bin = 4'b0001;
9  4'b0011: o_bin = 4'b0010;
10 4'b0010: o_bin = 4'b0011;
11 4'b0110: o_bin = 4'b0100;
12 4'b0111: o_bin = 4'b0101;
13 4'b0101: o_bin = 4'b0110;
14 4'b0100: o_bin = 4'b0111;
15 4'b1100: o_bin = 4'b1000;
16 4'b1101: o_bin = 4'b1001;
17 4'b1111: o_bin = 4'b1010;
18 4'b1110: o_bin = 4'b1011;
19 4'b1010: o_bin = 4'b1100;
20 4'b1011: o_bin = 4'b1101;
21 4'b1001: o_bin = 4'b1110;
22 4'b1000: o_bin = 4'b1111;
23 default: o_bin = 4'b0000;
24 endcase
25 end
26 endmodule
```

1.3 MEMORY

```
module fifo_mem #(
  parameter MEM_DEPTH = 8,
  parameter MEM_WIDTH = 4
)(
  input  wire i_wr_clk,
  input  wire                          i_rst_n,
  input  wire                          i_wr_en,
  input  wire [MEM_WIDTH-1:0]          i_wr_data,
  input  wire [$clog2(MEM_DEPTH)-1:0] i_wr_addr,
  input  wire                          i_rd_clk,
  input  wire                          i_rd_en,
  input  wire [$clog2(MEM_DEPTH)-1:0] i_rd_addr,
  output reg  [MEM_WIDTH-1:0]          o_rd_data
);

  reg [MEM_WIDTH-1:0] fifo_array [0:MEM_DEPTH-1];
  integer idx;


  always @(posedge i_rd_clk or negedge i_rst_n) begin
    if (!i_rst_n) begin
      o_rd_data <= 'b0;
    end
    else if (i_rd_en) begin
      o_rd_data <= fifo_array[i_rd_addr];
    end
  end


  always @(posedge i_wr_clk or negedge i_rst_n) begin
    if (!i_rst_n) begin
      for (idx = 0; idx < MEM_DEPTH; idx = idx + 1) begin
        fifo_array[idx] <= 'b0;
      end
    end
    else if (i_wr_en) begin
      fifo_array[i_wr_addr] <= i_wr_data;
    end
  end

endmodule
```

1.4 WRITE POINTER

```
module wrptr_full #(
  parameter PTR_WIDTH = 3
)(
  input  wire                  i_wr_clk,
  input  wire                  i_rst_n,
  input  wire                  i_wr_en,
  input  wire [PTR_WIDTH:0]    i_rdptr_sync,
  output reg  [PTR_WIDTH:0]    o_wrptr,
  output wire [PTR_WIDTH-1:0]  o_wraddr,
  output wire                  o_full_flag
);

  assign o_wraddr    = o_wrptr[PTR_WIDTH-1:0];
  assign o_full_flag = ({~o_wrptr[PTR_WIDTH], o_wrptr[PTR_WIDTH-1:0]}
                        == i_rdptr_sync);

  always @(posedge i_wr_clk or negedge i_rst_n) begin
    if (!i_rst_n) begin
      o_wrptr <= 'b0;
    end
    else if (i_wr_en && !o_full_flag) begin
      o_wrptr <= o_wrptr + 1;
    end
  end

endmodule
```

1.5 BINARY TO GRAY

```
module bin2gray(
input wire [3:0] i_bin,
output reg [3:0] o_gray
);
always@(*) begin
case(i_bin)
4'b0000: o_gray = 4'b0000;
4'b0001: o_gray = 4'b0001;
4'b0010: o_gray = 4'b0011;
4'b0011: o_gray = 4'b0010;
4'b0100: o_gray = 4'b0110;
4'b0101: o_gray = 4'b0111;
4'b0110: o_gray = 4'b0101;
4'b0111: o_gray = 4'b0100;
4'b1000: o_gray = 4'b1100;
4'b1001: o_gray = 4'b1101;
4'b1010: o_gray = 4'b1111;
4'b1011: o_gray = 4'b1110;
4'b1100: o_gray = 4'b1010;
4'b1101: o_gray = 4'b1011;
4'b1110: o_gray = 4'b1001;
4'b1111: o_gray = 4'b1000;
default: o_gray = 4'b0000;
endcase
end
endmodule
```

1.6 READ POINTER

```
module rdptr_empty #(
  parameter PTR_WIDTH = 3
)(
  input  wire                  i_rd_clk,
  input  wire                  i_rst_n,
  input  wire                  i_rd_en,
  input  wire [PTR_WIDTH:0]    i_wrptr_sync,
  output reg  [PTR_WIDTH:0]    o_rdptr,
  output wire [PTR_WIDTH-1:0]  o_rdaddr,
  output wire                  o_empty_flag
);

  assign o_rdaddr     = o_rdptr[PTR_WIDTH-1:0];
  assign o_empty_flag = (o_rdptr == i_wrptr_sync);

  always @(posedge i_rd_clk or negedge i_rst_n) begin
    if (!i_rst_n) begin
      o_rdptr <= 'b0;
    end
    else if (i_rd_en && !o_empty_flag) begin
      o_rdptr <= o_rdptr + 1;
    end
  end

endmodule
```

1.0 TOP MODULE

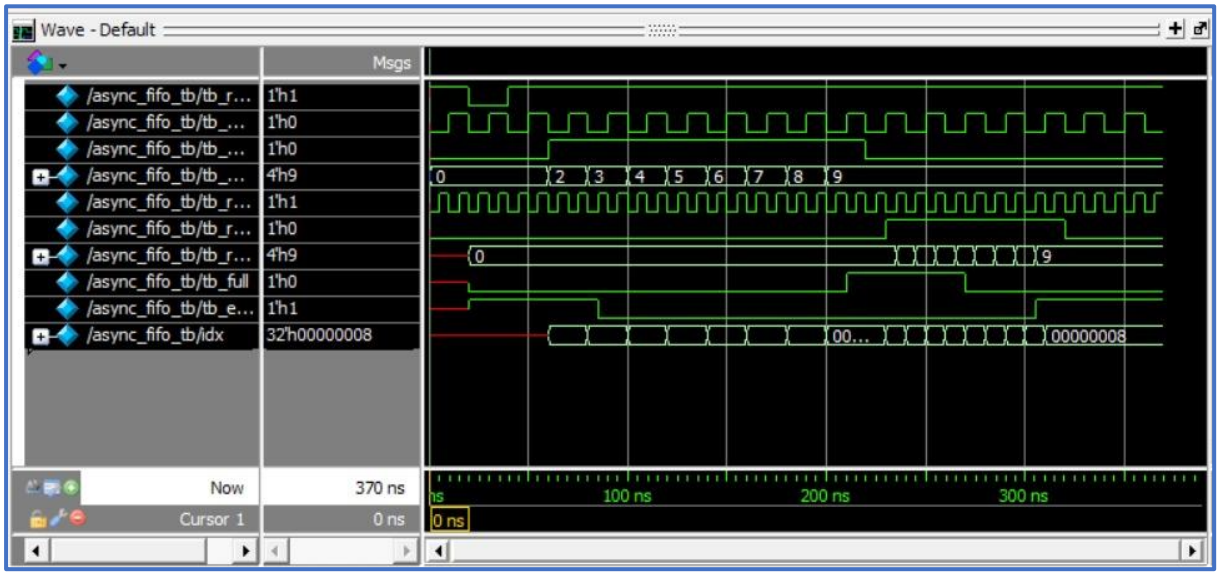
```
module async_fifo #(parameter DATA_WIDTH=4, parameter FIFO_DEPTH=8)(
input wire i_rst_n,
input wire i_wr_clk,
input wire i_wr_en,
input wire [DATA_WIDTH-1:0] i_wr_data,
input wire i_rd_clk,
input wire i_rd_en,
output wire [DATA_WIDTH-1:0] o_rd_data,
output wire o_full,
output wire o_empty
);
parameter PTR_WIDTH = $clog2(FIFO_DEPTH);
wire enable_wr;
wire enable_rd;
wire [PTR_WIDTH:0] rdptr,wrptr;
wire [PTR_WIDTH:0] rdptr_sync;
wire [PTR_WIDTH:0] wrptr_sync;
wire [PTR_WIDTH:0] rdptr_b2g_out;
wire [PTR_WIDTH:0] wrptr_b2g_out;
wire [PTR_WIDTH:0] rdptr_g2b_out;
wire [PTR_WIDTH:0] wrptr_g2b_out;
wire [PTR_WIDTH-1:0] wraddr;
wire [PTR_WIDTH-1:0] rdaddr;
assign enable_wr = (i_wr_en && !o_full);
assign enable_rd = (i_rd_en && !o_empty);
fifo_mem #(.MEM_DEPTH(FIFO_DEPTH), .MEM_WIDTH(DATA_WIDTH)) u_fifo_mem (
.i_rst_n(i_rst_n),
.i_wr_clk(i_wr_clk),
.i_wr_en(enable_wr),
.i_wr_data(i_wr_data),
.i_wr_addr(wraddr),
.i_rd_clk(i_rd_clk),
.i_rd_en(enable_rd),
.i_rd_addr(rdaddr),
.o_rd_data(o_rd_data)
);
wrptr_full #(.PTR_WIDTH(PTR_WIDTH)) u_wrptr_full (
.i_wr_clk(i_wr_clk),
.i_rst_n(i_rst_n),
.i_wr_en(i_wr_en),
.i_rdptr_sync(rdptr_g2b_out),
.o_wrptr(wrptr),
```

```
.o_wraddr(wraddr),
.o_full_flag(o_full)
);

bin2gray u_wrptr_bin2gray(
.i_bin(wrptr),
.o_gray(wrptr_b2g_out)
);
syn2stage #(.SIG_WIDTH(DATA_WIDTH)) u_wrptr_syn(
.i_clk(i_rd_clk),
.i_rst_n(i_rst_n),
.i_sig(wrptr_b2g_out),
.o_sig_sync(wrptr_sync)
);
gray2bin u_wrptr_gray2bin(
.i_gray(wrptr_sync),
.o_bin(wrptr_g2b_out)
);
rdptr_empty #(.PTR_WIDTH(PTR_WIDTH)) u_rdptr_empty (
.i_rd_clk(i_rd_clk),
.i_rst_n(i_rst_n),
.i_rd_en(i_rd_en),
.i_wrptr_sync(wrptr_g2b_out),
.o_rdptr(rdptr),
.o_rdaddr(rdaddr),
.o_empty_flag(o_empty)
);
bin2gray u_rdptr_bin2gray(
.i_bin(rdptr),
.o_gray(rdptr_b2g_out)
);
syn2stage #(.SIG_WIDTH(DATA_WIDTH)) u_rdptr_syn(
.i_clk(i_wr_clk),
.i_rst_n(i_rst_n),
.i_sig(rdptr_b2g_out),
.o_sig_sync(rdptr_sync)
);
gray2bin u_rdptr_gray2bin(
.i_gray(rdptr_sync),
.o_bin(rdptr_g2b_out)
);
endmodule
```

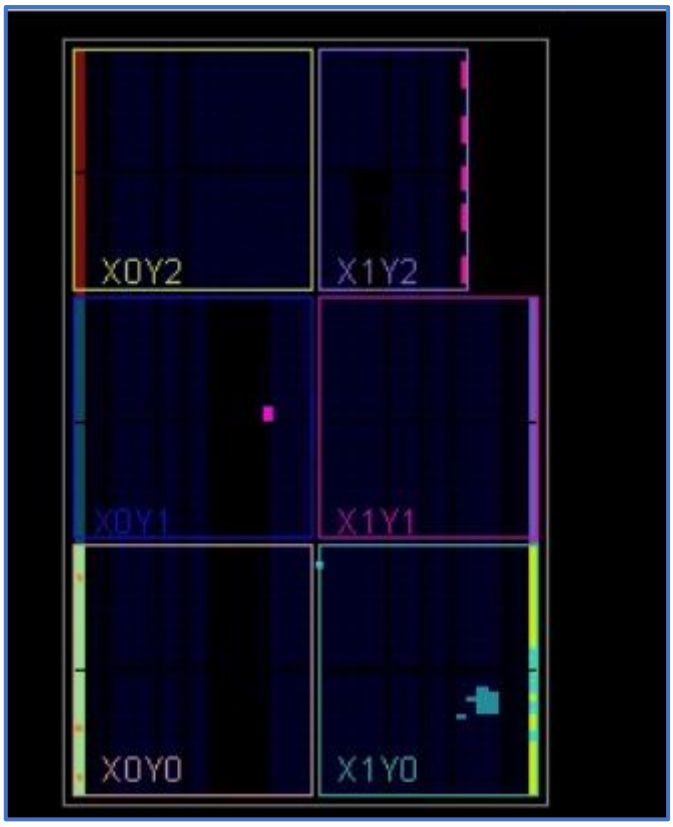
2.0 TESTBENCH

```
module async_fifo_tb;

  parameter DATA_WIDTH = 4;
  parameter FIFO_DEPTH = 8;

  reg                   tb_rst_n;
  reg                   tb_wr_clk;
  reg                   tb_wr_en;
  reg  [DATA_WIDTH-1:0] tb_wr_data;
  reg                   tb_rd_clk;
  reg                   tb_rd_en;
  wire [DATA_WIDTH-1:0] tb_rd_data;
  wire                  tb_full;
  wire                  tb_empty;

  integer idx;

  async_fifo #(.DATA_WIDTH(DATA_WIDTH), .FIFO_DEPTH(FIFO_DEPTH)) u_tb_fifo (
    .i_rst_n   (tb_rst_n),
    .i_wr_clk  (tb_wr_clk),
    .i_wr_en   (tb_wr_en),
    .i_wr_data (tb_wr_data),
    .i_rd_clk  (tb_rd_clk),
    .i_rd_en   (tb_rd_en),
    .o_rd_data (tb_rd_data),
    .o_full    (tb_full),
    .o_empty   (tb_empty)
  );

initial begin
    forever begin
      #5  tb_rd_clk = ~tb_rd_clk;
    end
end

initial begin
    forever begin
      #10 tb_wr_clk = ~tb_wr_clk;
    end
end

  initial begin
    tb_wr_clk  = 0;
    tb_rd_clk  = 0;
    tb_rst_n   = 1;
    tb_wr_en   = 0;
    tb_wr_data = 0;
    tb_rd_en   = 0;

    #20 tb_rst_n = 0;
    #20 tb_rst_n = 1;
    #20;

    for (idx = 0; idx < FIFO_DEPTH; idx = idx + 1) begin
      @(negedge tb_wr_clk);
      tb_wr_en   = 1'b1;
      tb_wr_data = idx + 2;
    end

    @(negedge tb_wr_clk) tb_wr_en = 1'b0;
    @(negedge tb_rd_clk) tb_rd_en = 1'b1;

    for (idx = 0; idx < FIFO_DEPTH; idx = idx + 1) begin
      @(negedge tb_rd_clk);
      $display("Read cycle
 -> Data =
", idx, tb_rd_data);
    @(negedge tb_rd_clk) tb_rd_en = 1'b0;
    #50 $stop;
  end



endmodule


```

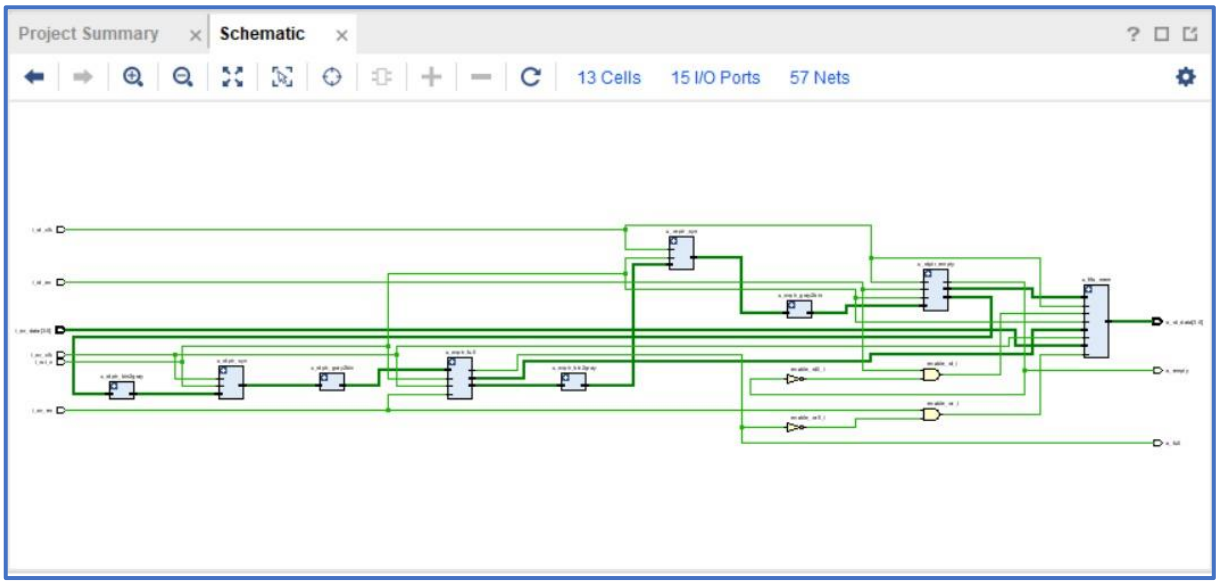
3.0 WAVEFORM



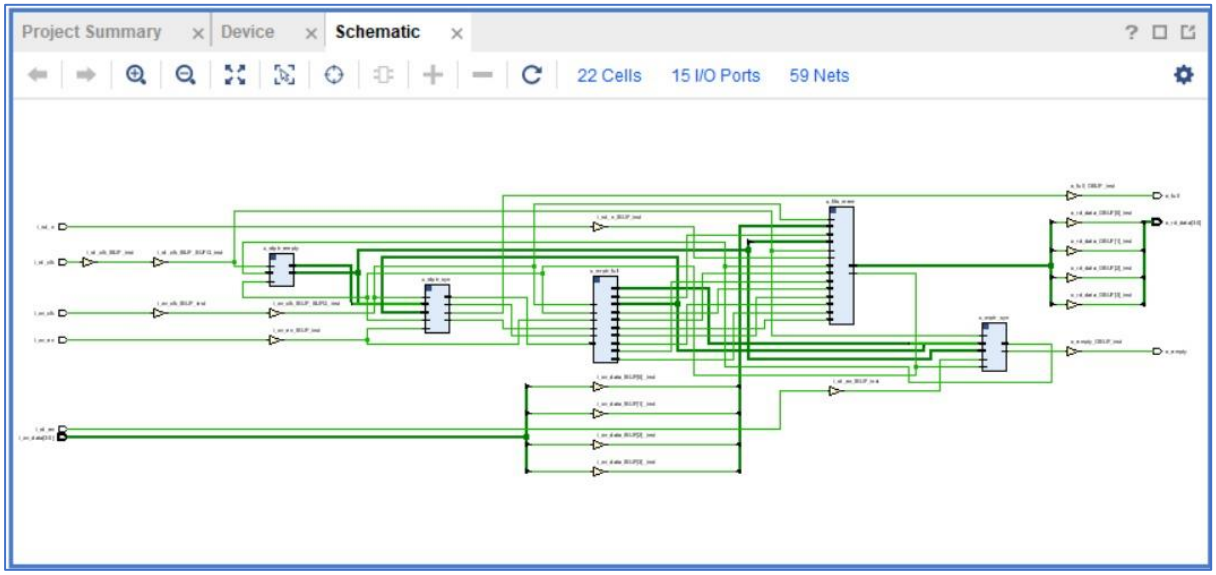
4.0 IMPLEMENTATION



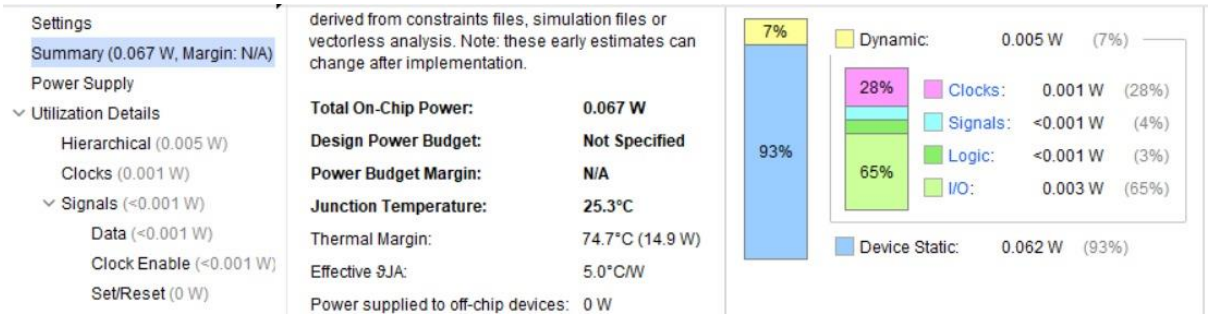
5.0 RTL ANALYSIS



6.0 SYNTHESIS

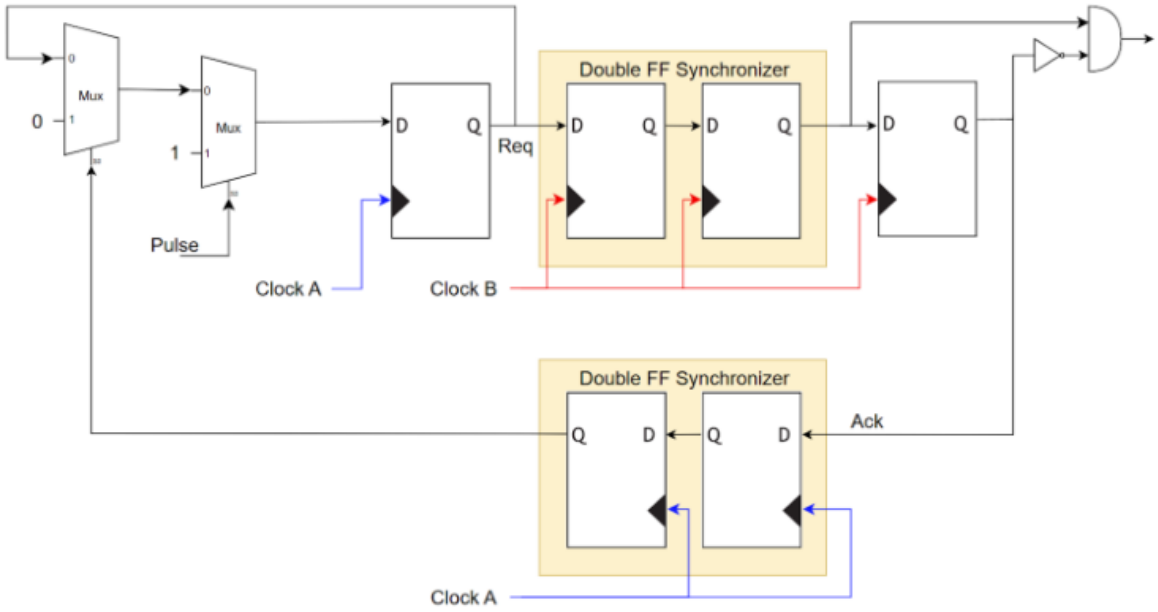


7.0 POWER REPORT



8.0 Timing Report

2.0 Fast to slow pulse to pulse synchronizer



2.1.1 DFF Code

```
1  module DFF (clk,rst,D,Q);
2  input clk ,rst;
3  input D;
4  output reg Q;
5
6  always @(posedge clk or posedge rst) begin
7      if (rst) begin
8          Q <= 0;
9      end else begin
10         Q <= D;
11     end
12 end
13 endmodule
14
```

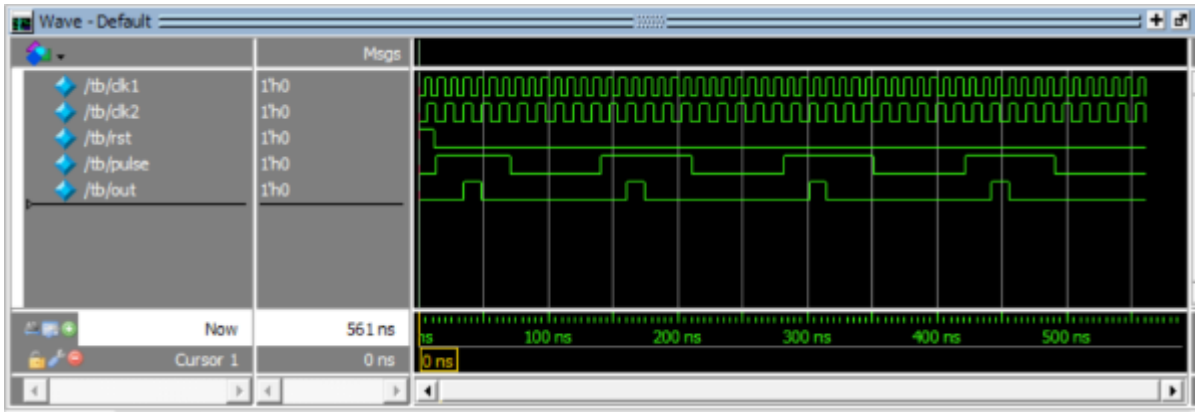
2.1.2 Top Module

```
module top(clk1 ,clk2 ,rst ,pulse ,out);
input clk1 ,clk2 ,rst ,pulse;
output out;
wire out1 ,in2 ,out2 ,in3 ,out3
,in4 ,out4 ,in5 ,out5 ,in6 ,out6
,mux_out1 ,mux_out2;

DFF F1(.clk(clk1),.rst(rst),.D(mux_out2),.Q(out1));
DFF F2(.clk(clk2),.rst(rst),.D(in2),.Q(out2));
DFF F3(.clk(clk2),.rst(rst),.D(in3),.Q(out3));
DFF F4(.clk(clk2),.rst(rst),.D(in4),.Q(out4));
DFF F5(.clk(clk1),.rst(rst),.D(in5),.Q(out5));
DFF F6(.clk(clk1),.rst(rst),.D(in6),.Q(out6));

assign in2 = out1;
assign in3 = out2;
assign in4 = out3;
assign in5 = out4;
assign in6 = out5;
assign mux_out1 = (out6)? 1'b0 : out1;
assign mux_out2 = (pulse)? 1'b1 : mux_out1;
assign out = (out3 & ~out4);

endmodule
```

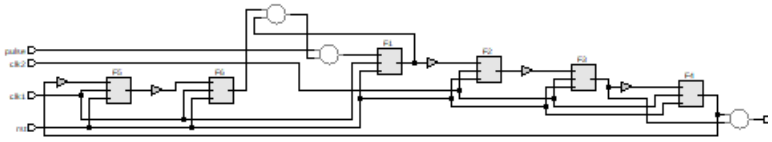
2.1.3 Test Bench

```verilog
module tb();
  reg clk1, clk2, rst, pulse;
  wire out;

  top DUT (
    .clk1(clk1),
    .clk2(clk2),
    .rst(rst),
    .pulse(pulse),
    .out(out)
  );

  initial begin
    clk1 = 0;
    forever #5 clk1 = ~clk1;
  end

  initial begin
    clk2 = 0;
    forever #7 clk2 = ~clk2;
  end

  initial begin
    rst = 1; pulse = 0;
    #12 rst = 0;
    #1 pulse = 1; repeat(3)@(negedge clk1); repeat(3)@(negedge clk2);
    #1 pulse = 0; repeat(3)@(negedge clk1); repeat(3)@(negedge clk2);
    #1 pulse = 1; repeat(3)@(negedge clk1); repeat(3)@(negedge clk2);
    #1 pulse = 0; repeat(3)@(negedge clk1); repeat(3)@(negedge clk2);
    #1 pulse = 1; repeat(3)@(negedge clk1); repeat(3)@(negedge clk2);
    #1 pulse = 0; repeat(3)@(negedge clk1); repeat(3)@(negedge clk2);
    #1 pulse = 1; repeat(3)@(negedge clk1); repeat(3)@(negedge clk2);
    #1 pulse = 0; repeat(3)@(negedge clk1); repeat(3)@(negedge clk2);
    #1 $stop;
  end

  initial begin
    $monitor("T=%0t | rst=%b | pulse=%b | out=%b",
             $time, rst, pulse, out);
  end
endmodule
```

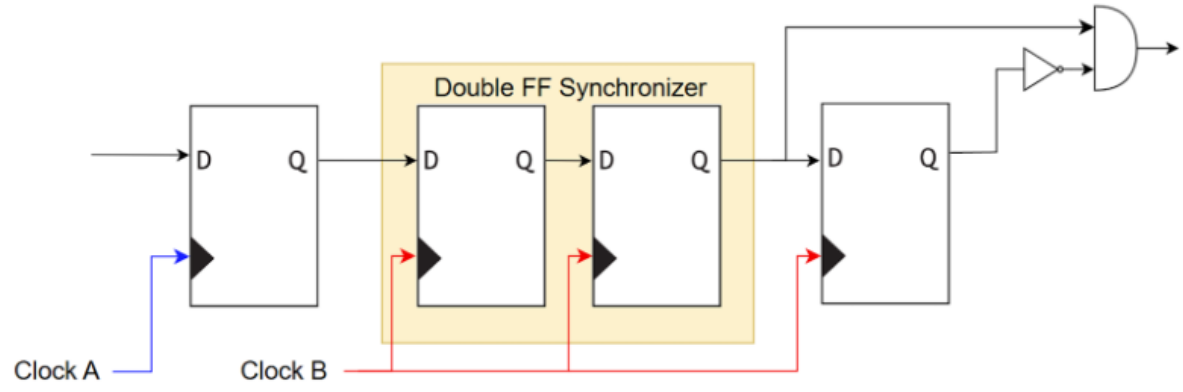
2.2 Waveform



2.3 SYNTHESIS



3.0 Slow to fast pulse to pulse synchronizer



3.1.1 DFF Code

```
1  module DFF (clk,rst,D,Q);
2  input clk ,rst;
3  input D;
4  output reg Q;
5
6  always @(posedge clk or posedge rst) begin
7      if (rst) begin
8          Q <= 0;
9      end else begin
10         Q <= D;
11     end
12 end
13 endmodule
14
```

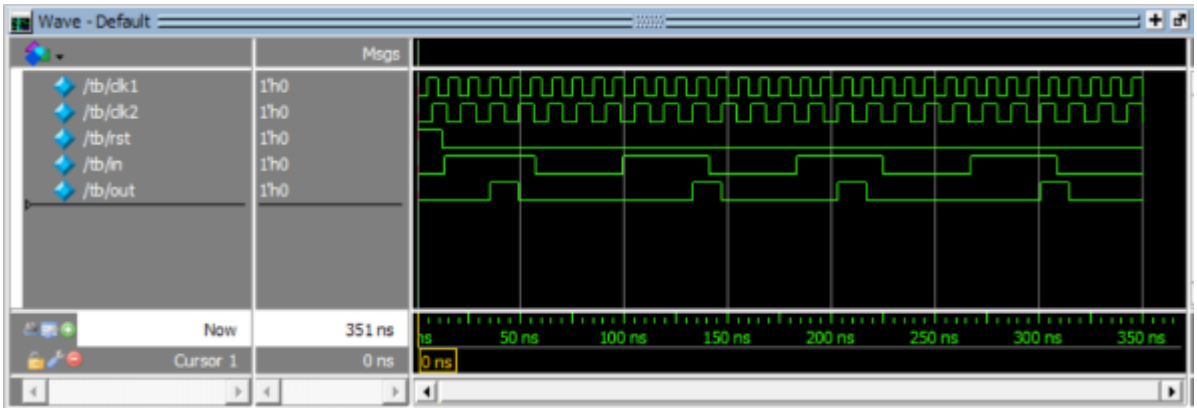
3.1.2 Top Module

```
1  module top(clk1 ,clk2 ,rst ,in ,out);
2  input clk1 ,clk2 ,rst ,in;
3  output out;
4  wire out1 ,in2 ,out2 ,in3 ,out3 ,in4 ,out4;
5
6  DFF F1(.clk(clk1),.rst(rst),.D(in),.Q(out1));
7  DFF F2(.clk(clk2),.rst(rst),.D(in2),.Q(out2));
8  DFF F3(.clk(clk2),.rst(rst),.D(in3),.Q(out3));
9  DFF F4(.clk(clk2),.rst(rst),.D(in4),.Q(out4));
10
11 assign in2 = out1;
12 assign in3 = out2;
13 assign in4 = out3;
14 assign out = (out3 & ~out4);
15
16 endmodule
```

3.1.3 Test Bench

```
module tb;
  reg clk1, clk2, rst, in;
  wire out;

  top DUT (
    .clk1(clk1),
    .clk2(clk2),
    .rst(rst),
    .in(in),
    .out(out)
  );

  initial begin
    clk1 = 0;
    forever #5 clk1 = ~clk1;
  end

  initial begin
    clk2 = 0;
    forever #7 clk2 = ~clk2;
  end

  initial begin
    rst = 1; in = 0;
    #12 rst = 0;
    #1 in = 1; @(negedge clk1); repeat(3)@(negedge clk2);
    #1 in = 0; @(negedge clk1); repeat(3)@(negedge clk2);
    #1 in = 1; @(negedge clk1); repeat(3)@(negedge clk2);
    #1 in = 0; @(negedge clk1); repeat(3)@(negedge clk2);
    #1 in = 1; @(negedge clk1); repeat(3)@(negedge clk2);
    #1 in = 0; @(negedge clk1); repeat(3)@(negedge clk2);
    #1 in = 1; @(negedge clk1); repeat(3)@(negedge clk2);
    #1 in = 0; @(negedge clk1); repeat(3)@(negedge clk2);
    #1 $stop;
  end

  initial begin
    $monitor("T=%0t | rst=%b | in=%b | out=%b",
             $time, rst, in, out);
  end
endmodule
```

3.2 Waveform



3.3 SYNTHESIS