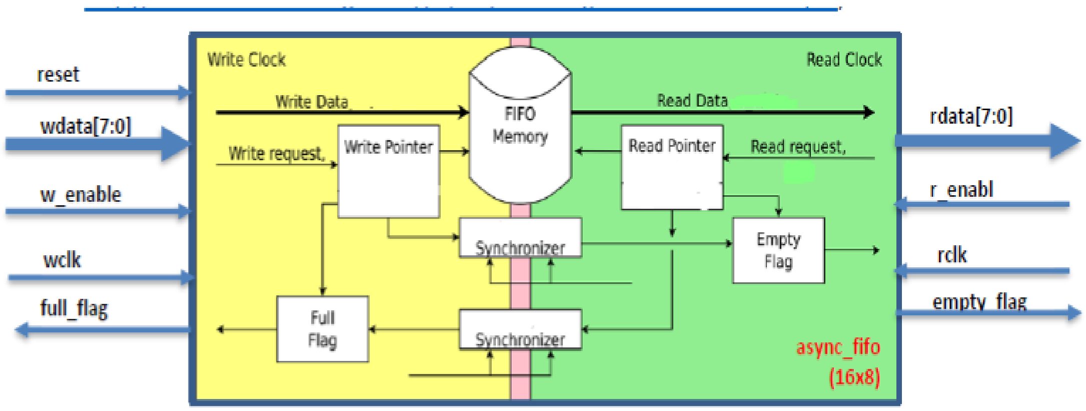
Design and Verification of 16x8 Asynchronous FIFO



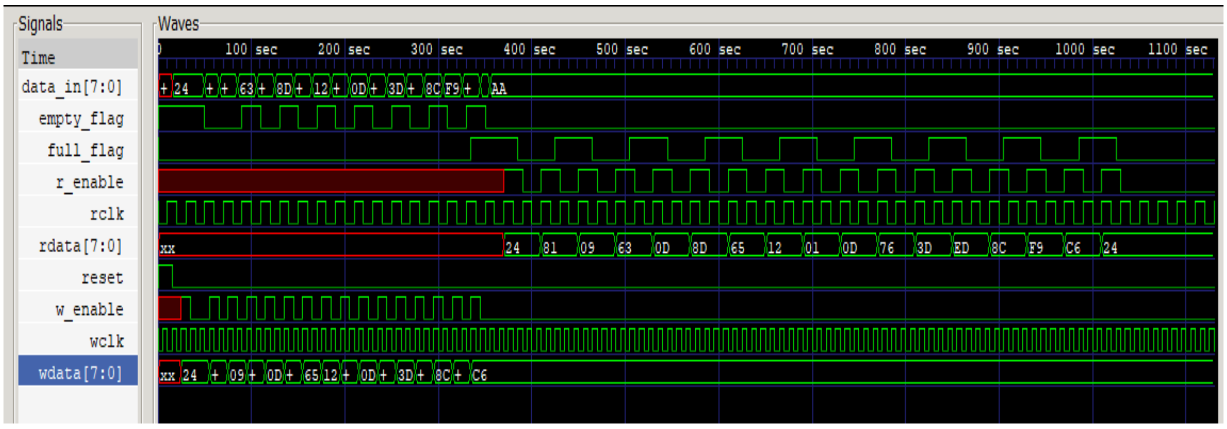
Code:

```
Module async_fifo(rdata,full_flag,empty_flag,wdata,wclk,rclk,reset,
w_enable,r_enable);
parameter depth=16,ptr_size=4;
input wclk,rclk,reset,w_enable,r_enable;
input [7:0] wdata;
output reg [7:0] rdata;
output full_flag,empty_flag;
reg [7:0] mem [depth-1:0];
reg [ptr_size:0] wptr,rptr;
wire [ptr_size:0] wp_sync,rp_sync;
//Synchronizing pointers using double synchronizer
double_sync read (rp_sync,wclk,reset,rptr);
double_sync write (wp_sync,rclk,reset,wptr);
assign full_flag = ({~wptr[ptr_size],wptr[ptr_size-1:0]} ==
rp_sync[ptr_size:0]);//write pointer wrapped around once making MSB
1
assign empty_flag=(wp_sync == rptr); //read pointer catching write
pointer
always@(posedge wclk,posedge reset) begin
if(reset) wptr<=0;
else begin
    if(w_enable && ~full_flag) begin
        mem[wptr[ptr_size-1:0]]<=wdata;wptr<=wptr+1;
        end
    end
     end
always@(posedge rclk,posedge reset) begin
if(reset) rptr<=0;
else begin
    if(r_enable && ~empty_flag) begin
        rdata<=mem[rptr[ptr_size-1:0]];rptr<=rptr+1;
        end
    end
     end
endmodule
module double_sync(output reg [4:0] q2,input clk,reset,input [4:0]
d1);
```

```
reg q1;
always@(posedge clk,posedge reset)
begin
if(reset==1'b1) begin
    q1<=0;q2<=0;
        end
else
    begin
    q1<=d1;q2<=q1;
    end
end
endmodule
//Test bench
module test;
parameter depth=16,pointer=4;
reg wclk,rclk,reset,w_enable,r_enable;
reg [7:0] wdata,data_in;
wire [7:0] rdata;
wire full_flag,empty_flag;
integer count;
async_fifo f
(rdata,full_flag,empty_flag,wdata,wclk,rclk,reset,w_enable,r_enable)
;
task write;
input [7:0] data_in;
begin
@(posedge wclk)
    if(!full_flag) begin
        w_enable=1;wdata=data_in;
@(posedge wclk)
    w_enable=0;
        end
    end
endtask
task read;
begin
@(posedge rclk)
    if(!empty_flag) begin
        r_enable=1;
@(posedge rclk)
    r_enable=0;
        end
    end
endtask
task rw;
begin
data_in=$random;write(data_in);read();
end
endtask
task full_empty;
 begin
     for(count=0;count<depth+1;count=count+1) begin
        data_in=$random;write(data_in);
        end
     for(count=0;count<depth+1;count=count+1) begin
      read();
```

```
              end
          end
          endtask
always #5 wclk = ~wclk;
always #10 rclk = ~rclk;
initial
 begin
     $dumpfile("fifo.vcd");$dumpvars;
     wclk=0;rclk=0;
     reset=1;
     #15;reset=0;
     @(posedge wclk)
         rw;full_empty;
         #100
         $finish;
     end
 endmodule
```

Timing Diagram:


Signals
Waves
Time
data_in[7:0]
empty_flag
full_flag
r_enable
rclk
rdata[7:0]
reset
w_enable
wclk
wdata[7:0]
100 sec
200 sec
300 sec
400 sec
500 sec
600 sec
700 sec
800 sec
900 sec
1000 sec
1100 sec
+ 24 + + 63 + 8D + 12 + 0D + 3D + 8C F9 + AA
xx 24 81 09 63 0D 8D 65 12 01 0D 76 3D ED 8C F9 C6 24
xx 24 + 09 + 0D + 65 12 + 0D + 3D + 8C + C6