```
//Assignment: ECE 474 Homework 4
//Author: Andrew Collins
//File:fifo.sv
//Description: Implementing a 8bit wide, 8byte deep first in first out memory.
module fifo (
       input            wr_clk,   //write clock
       input            rd_clk,   //read clock
       input            reset_n,  //reset async active low
       input            wr,       //write enable
       input            rd,       //read enable
       input      [7:0] data_in,  //data in
       output reg [7:0] data_out, //data out
       output reg   empty,    //empty flag
       output reg full );     //full flag

reg [2:0] wr_ptr;
reg [2:0] rd_ptr;
reg wr_togle;
reg rd_togle;
reg empty_temp;//not used
reg empty_sync;//not used
reg [7:0] mem0;
reg [7:0] mem1;
reg [7:0] mem2;
reg [7:0] mem3;
reg [7:0] mem4;
reg [7:0] mem5;
reg [7:0] mem6;
reg [7:0] mem7;


/*---------------------------------------------*/
always_ff @ (posedge wr_clk, negedge reset_n)begin
        if(~reset_n) begin
                         mem0 <= '0;
                         mem1 <= '0;
                         mem2 <= '0;
                         mem3 <= '0;
                         mem4 <= '0;
                         mem5 <= '0;
                         mem6 <= '0;
                         mem7 <= '0;
        end
        else if(wr) begin
                //if((wr) && (!full)) begin
                        unique case (wr_ptr)
                        3'b000 : mem0 <= data_in;
                        3'b001 : mem1 <= data_in;
                        3'b010 : mem2 <= data_in;
                        3'b011 : mem3 <= data_in;
                        3'b100 : mem4 <= data_in;
                        3'b101 : mem5 <= data_in;
                        3'b110 : mem6 <= data_in;
                        3'b111 : mem7 <= data_in;
                        endcase
                end
end


/*---------------------------------------------*/
always_ff @ (posedge wr_clk, negedge reset_n)begin
        if(~reset_n) begin
                wr_togle <=0;
        end
        else if( wr_ptr == 3'b111)begin
                                wr_togle <=~wr_togle;
                end
        else wr_togle <=wr_togle;
end
```

```

/*---------------------------------------------*/
always_ff @ (posedge wr_clk, negedge reset_n)begin
        if(~reset_n) begin
                wr_ptr <=0;
        end
                else    if( wr) wr_ptr <=wr_ptr+1;
                else wr_ptr <=wr_ptr;
end


/*---------------------------------------------*/
always_ff @(posedge rd_clk, negedge reset_n)begin
        if(~reset_n) rd_togle<=0;
        else if(rd_ptr == 3'b111)       rd_togle <= ~rd_togle;
        else    rd_togle <=rd_togle;
end


/*---------------------------------------------*/
always_ff @(posedge rd_clk, negedge reset_n)begin
if(~reset_n)    rd_ptr <=0;
        else if(rd)     rd_ptr <=rd_ptr+1;
        else    rd_ptr <=rd_ptr;
end


/*---------------------------------------------*/
//setting empty
always_comb begin
if((rd_ptr == wr_ptr) && (rd_togle == wr_togle))
empty = 1;
else
empty = 0;

end


/*---------------------------------------------*/
//setting full
always_comb begin
if((rd_ptr == wr_ptr) && (rd_togle != wr_togle))
full = 1;
else
full = 0;
end


/*---------------------------------------------*/
//data immediately available at the output
always_comb begin
        unique case (rd_ptr)
                3'b000 : data_out=mem0;
                3'b001 : data_out=mem1;
                3'b010 : data_out=mem2;
                3'b011 : data_out=mem3;
                3'b100 : data_out=mem4;
                3'b101 : data_out=mem5;
                3'b110 : data_out=mem6;
                3'b111 : data_out=mem7;
        endcase
end


endmodule
```