Code:

`timescale 1ns / 1ps

module main

(

input clk,rst,ack,rw,scl,newd, //newd = 1:whenever a new data is incoming

inout sda, //in-read , out-write

input [7:0] wdata, //8 bit write data

input [6:0] addr, //8-bit 7-bit : + addr 1-bit : mode

output reg [7:0] rdata, //8 bit read data

output reg done

);

reg sda\_en = 0; //1(write):sda=dat 0(read):sda=z

reg sclt, sdat, donet; //temporary in-program usage

reg [7:0] rdatat; //read data temp

reg [7:0] addrt; //address temp

reg [3:0] state; //13 states

parameter

idle = 0, //initial stage

check\_rw = 1, //check rw signal

wstart = 2, //start operation

wsend\_addr = 3, //send address for write

waddr\_ack = 4, //write address acknowledgment

wsend\_data = 5, //send data for write

wdata\_ack = 6, //write data acknowledgment

wstop = 7, //stop write

rsend\_addr = 8, //send address for read

raddr\_ack = 9, //read address acknowledgment

rsend\_data = 10, //send data for read

rdata\_ack = 11, //read data acknowledgment

rstop = 12 ; //stop read

reg sclk\_wr = 0; //Actual slower clock (except when start-writing,stop-writing,stop-reading)

integer i,count = 0;

////100 M / 400 K = N

////N/2

//Slower clock generation

always@(posedge clk)

begin

if(count <= 9)

begin

count <= count + 1;

end

else

begin

count <= 0;

sclk\_wr <= ~sclk\_wr;

end

end

//FSM

always@(posedge sclk\_wr, posedge rst)

begin

if(rst == 1'b1)

begin

sclt <= 1'b0;

sdat <= 1'b0;

donet <= 1'b0;

end

else begin

case(state)

idle :

begin

sdat <= 1'b0;

done <= 1'b0;

sda\_en <= 1'b1;

sclt <= 1'b1;

sdat <= 1'b1;

if(newd == 1'b1)

state <= wstart;

else

state <= idle;

end

wstart:

begin

sdat <= 1'b0;

sclt <= 1'b1;

state <= check\_rw;

addrt <= {addr,rw};

end

check\_rw: begin //addr remain same for both write and read

if(rw)

begin

state <= rsend\_addr;

sdat <= addrt[0];

i <= 1;

end

else

begin

state <= wsend\_addr;

sdat <= addrt[0];

i <= 1;

end

end

//write state

wsend\_addr : begin

if(i <= 7) begin

sdat <= addrt[i];

i <= i + 1;

end

else

begin

i <= 0;

state <= waddr\_ack;

end

end

waddr\_ack : begin

if(ack) begin

state <= wsend\_data;

sdat <= wdata[0];

i <= i + 1;

end

else

state <= waddr\_ack;

end

wsend\_data : begin

if(i <= 7) begin

i <= i + 1;

sdat <= wdata[i];

end

else begin

i <= 0;

state <= wdata\_ack;

end

end

wdata\_ack : begin

if(ack) begin

state <= wstop;

sdat <= 1'b0;

sclt <= 1'b1;

end

else begin

state <= wdata\_ack;

end

end

wstop: begin

sdat <= 1'b1;

state <= idle;

done <= 1'b1;

end

//read state

rsend\_addr : begin

if(i <= 7) begin

sdat <= addrt[i];

i <= i + 1;

end

else

begin

i <= 0;

state <= raddr\_ack;

end

end

raddr\_ack : begin

if(ack) begin

state <= rsend\_data;

sda\_en <= 1'b0;

end

else

state <= raddr\_ack;

end

rsend\_data : begin

if(i <= 7) begin

i <= i + 1;

state <= rsend\_data;

rdata[i] <= sda;

end

else

begin

i <= 0;

state <= rstop;

sclt <= 1'b1;

sdat <= 1'b0;

end

end

rstop: begin

sdat <= 1'b1;

state <= idle;

done <= 1'b1;

end

default : state <= idle;

endcase

end

end

assign scl = (( state == wstart) || ( state == wstop) || ( state == rstop)) ? sclt : sclk\_wr;

assign sda = (sda\_en == 1'b1) ? sdat : 1'bz;

endmodule