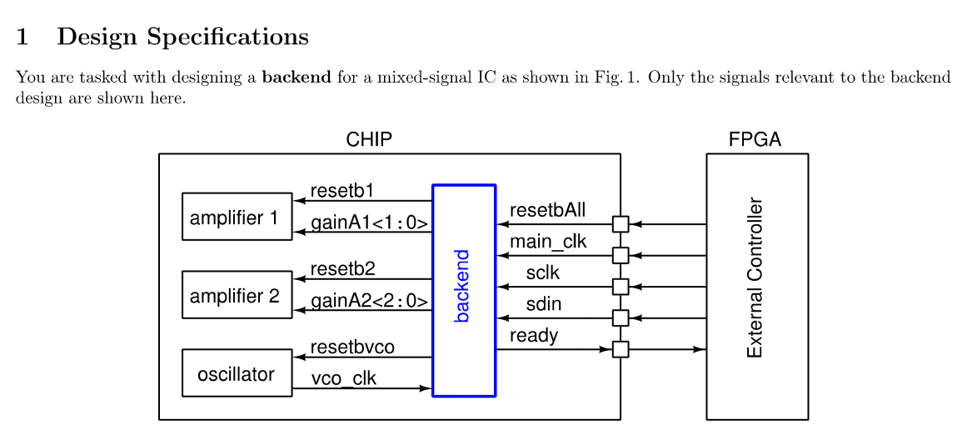
Digital Project:

Rabindra Kharga

Roll: 21204407

Electrical Engg.



A close up of a text

Description automatically generated

A computer program with text

Description automatically generated with medium confidence

Designing backend of a mixed signal IC using Verilog:

Designing in such a way that backend has to execute startup sequence and provide data to the input of an IC

A black background with a black square

Description automatically generated with medium confidence

**Solution:**

**FPGA CODE in Verilog :**

//File Name: FPGA

// Type: Module

// Department : Electrical engineering

// Author: Rabindra Kharga

// Purpose: Module

// Operation : data control

// Date : 10 July 2023

///////////////////////////////////////////////////////////////////////////////

/\*FPGA model

\* Generates reset signal for the chip

\* Sends out progamming bits to the chip

\*/

module FPGA\_model( i\_resetbFPGA,

i\_ready,

i\_mainclk,

o\_resetbAll,

o\_sclk,

o\_sdout);

//============================================================================

//Parameter declarations

parameter opcode\_gainA1 = 1; //range: 0-3. Gain for amplifier 1

parameter opcode\_gainA2 = 5; //range: 0-7. Gain for amplifier 2

//FPGA states

parameter sRESET = 0; //Reset all internal variables. Make o\_resetbAll=0.

parameter sPROGRAM = 1; //Send programming bits to chip.

parameter sIDLE = 2; //Do nothing.

//===========================================================================

//Input output declarations

input i\_resetbFPGA; //resets FPGA

input i\_ready; //Alerts FPGA that chip is programmed and ready

input i\_mainclk; //Main clock

output reg o\_resetbAll; //Reset for the chip

output reg o\_sclk; //Serial clock for communication

output reg o\_sdout; //Serial data out from FPGA (to chip)

//Internal wire, reg declarations

reg [1:0] FPGAstate; //Holds the state of FPGA

reg [3:0] count; //Used for controlling sclk generation and transmission of serial data

reg mainclkby2, mainclkby4, mainclkby8, mainclkby16; //Low frequency clocks derived from mainclk

//=============================================================

//BEHAVIORAL DESCRIPTION

//=============================================================

// Setting the FPGA state

always @ (posedge(i\_mainclk) or negedge(i\_resetbFPGA))

begin

if(i\_resetbFPGA == 0)

FPGAstate <= sRESET;

else

begin

case(FPGAstate)

sRESET: FPGAstate <= sPROGRAM;

sPROGRAM:

begin

if(count== 10)

FPGAstate <= sIDLE;

else

FPGAstate <= sPROGRAM;

end

sIDLE: FPGAstate <=sIDLE;

default: FPGAstate <= sIDLE;

endcase

end

end

//========================================================

/\*resetbALL

\* resetbAll=0, when FPGA is reset.

\* resetbAll=1 at all other times.

\*/

always @(posedge(i\_mainclk) or negedge(i\_resetbFPGA))

begin

if(i\_resetbFPGA == 0)

o\_resetbAll <= 0;

else

begin

if(FPGAstate == sRESET)

o\_resetbAll <= 0;

else

o\_resetbAll <= 1;

end

end

//==========================================================

/\* Generation of low frequency clocks from mainclk using divide-by-2 method

\*/

always @ (posedge(i\_mainclk) or negedge(i\_resetbFPGA))

begin

if(i\_resetbFPGA == 0)

mainclkby2 <= 0;

else

mainclkby2 <= ~mainclkby2;

end

always @ (posedge(mainclkby2) or negedge(i\_resetbFPGA))

begin

if(i\_resetbFPGA == 0)

mainclkby4 <= 0;

else

mainclkby4 <= ~mainclkby4;

end

always @(posedge(mainclkby4) or negedge(i\_resetbFPGA))

begin

if(i\_resetbFPGA == 0)

mainclkby8 <= 0;

else

mainclkby8 <= ~mainclkby8;

end

always @(posedge(mainclkby8) or negedge(i\_resetbFPGA))

begin

if(i\_resetbFPGA == 0)

mainclkby16 <= 0;

else

mainclkby16 <= ~mainclkby16;

end

//==========================================================

/\*count

\* used for the generation of sclk and selecting the data to be sent in through sdin

\*/

always @(posedge(mainclkby16) or negedge(i\_resetbFPGA))

begin

if(i\_resetbFPGA == 0)

count <= 0;

else

begin

if(FPGAstate == sPROGRAM)

begin

if(count == 11)

count <= count;

else

count <= count+1;

end

else

count <= count;

end

end

//=========================================================

/\*sclk

\* sclk is a clock only when the FPGA is programming the chip.

\*/

always @ (posedge(mainclkby16) or negedge(i\_resetbFPGA))

begin

if(i\_resetbFPGA == 0)

o\_sclk <= 1;

else

begin

if(FPGAstate == sPROGRAM)

begin

if (count >= 1 && count <= 10)

o\_sclk <= ~o\_sclk;

else

o\_sclk <= 1;

end

else

o\_sclk <= 1;

end

end

//=========================================================

/\*sdin

\*serial data out from FPGA

\*/

always @ (negedge(o\_sclk) or negedge(i\_resetbFPGA))

begin

if(i\_resetbFPGA == 0)

o\_sdout <= 0;

else

begin

if(FPGAstate == sPROGRAM)

begin

case(count)

4'd2: o\_sdout <= opcode\_gainA1[0];

4'd4: o\_sdout <= opcode\_gainA1[1];

4'd6: o\_sdout <= opcode\_gainA2[0];

4'd8: o\_sdout <= opcode\_gainA2[1];

4'd10: o\_sdout <= opcode\_gainA2[2];

default: o\_sdout <= o\_sdout;

endcase

end

else

o\_sdout <= 0;

end

end

//==========================================================

Endmodule

**BACKEND FILE VERILOG CODE:**

//File Name: BACKNED

// Type: Module

// Department : Electrical engineering

// Author: Sagar

//Author's Email ID: guptasagar19965@gmail.com

// Purpose:TEST

// Operation : TEST

// Date : 10 July 2023

// BACKEND DESIGN //

module backend( i\_resetbAll,

i\_clk,

i\_sclk,

i\_sdin,

i\_vco\_clk,

o\_ready,

o\_resetb1,

o\_gainA1,

o\_resetb2,

o\_gainA2,

o\_resetbvco,);

// INPUT OUTPUT REGISTERS DECLARATION //

input i\_resetbAll, i\_clk, i\_sclk, i\_sdin, i\_vco\_clk;

reg [4:0]data;

integer i;

output reg o\_ready, o\_resetb1, o\_resetb2, o\_resetbvco;

output reg [1:0] o\_gainA1;

output reg [2:0] o\_gainA2;

reg [4:0] shiftreg;

reg [4:0] shift2;

reg [2:0] count;

//take serial data from i\_sdin and read the data using shift register

always @ ( posedge (i\_sclk))

begin

shift2= shiftreg>>1;

shiftreg ={i\_sdin, shift2[3:0]};

count<=count+1;

if (count==4)

begin

o\_gainA1[0] <= shiftreg[0];

o\_gainA1[1] <= shiftreg[1];

o\_gainA2[0] <= shiftreg[2];

o\_gainA2[1] <= shiftreg[3];

o\_gainA2[2] <= shiftreg[4];

end

end

always@(posedge (i\_clk) or negedge(i\_resetbAll))

begin

if(i\_resetbAll) //when i\_resetbAll=1, the starup sequence in initiated

begin

for(i=0;i<5;i=i+1) // store data in reg data

begin

@(posedge i\_sclk);

data <= {data[4:1],i\_sdin};

end

repeat(2) // now we wait for two clock cycles

begin

@(posedge (i\_clk) or negedge(i\_resetbAll)) ;

end

o\_resetbvco <=1; // set o\_resetbvco=1

repeat(10) // wait for 10 clock cycles

begin

@(posedge (i\_clk) or negedge(i\_resetbAll)) ;

end

o\_resetb1 <=1;// set o\_resetb1=1

o\_resetb2<=1;// set o\_resetb2=1

repeat(10) // wait for 10 clock cycles

begin

@(posedge (i\_clk) or negedge(i\_resetbAll)) ;

end

o\_ready<=1; // set o\_ready=1

end

else // when i\_resetbAll=0 all outputs of the backend should be pulled to zero

begin

o\_ready <= 0;

o\_resetb1 <= 0;

o\_resetb2 <= 0;

o\_resetbvco <= 0;

o\_gainA1[0] <= 0;

o\_gainA1[1] <= 0;

o\_gainA2[0] <= 0;

o\_gainA2[1] <= 0;

o\_gainA2[2] <= 0;

shiftreg <=0;

shift2 <= 0;

count <= 0;

data <=0;

end

end

//====================================================================================

Endmodule

**BACKEND TEST BENCH:**

//File Name: Testbanch of FPGA

// Type: Module

// Department : Electrical engineering

// Author: Sagar

//Author's Email ID: guptasagar19965@gmail.com

// Purpose: BACKEND TEST

// Operation : TEST

// Date : 10 July 2023

// Testbanch module for the backend. This is has a module instantiation for

// the FPGA\_model and the backend.

`timescale 1ns / 1ps

//==========================================================================

//Change the Verilog filenames approppriately.

`include "FPGA\_model.v"

`include "backend.v"

//=========================================================================

module backend\_tb();

reg resetbFPGA;

reg main\_clk;

wire [1:0]gainA1 ;

wire [2:0]gainA2 ;

wire resetbAll, resetb1, resetb2, resetbvco;

wire vco\_clk;

wire sclk, sdin;

wire ready;

//==========================================================================

//FPGA model instantiation

FPGA\_model FPGA\_obj( .i\_resetbFPGA (resetbFPGA),

.i\_ready (ready),

.o\_resetbAll (resetbAll),

.i\_mainclk (main\_clk),

.o\_sclk (sclk),

.o\_sdout (sdin));

// Backend instantiation

backend backend\_obj ( .i\_resetbAll (resetbAll),

.i\_clk (main\_clk),

.i\_sclk (sclk),

.i\_sdin (sdin),

.i\_vco\_clk (vco\_clk),

.o\_ready (ready),

.o\_resetb1 (resetb1),

.o\_gainA1 (gainA1),

.o\_resetb2 (resetb2),

.o\_gainA2 (gainA2),

.o\_resetbvco (resetbvco)

);

//============================================================================

//Test signal generation

initial

begin

resetbFPGA <= 0;

main\_clk <= 0;

#1 resetbFPGA <= 1;

end

initial

begin

$dumpfile("Sagar.vcd");

$dumpvars(0,backend\_tb);

end

//Generation of main\_clk

always #2.5 main\_clk <= ~main\_clk;

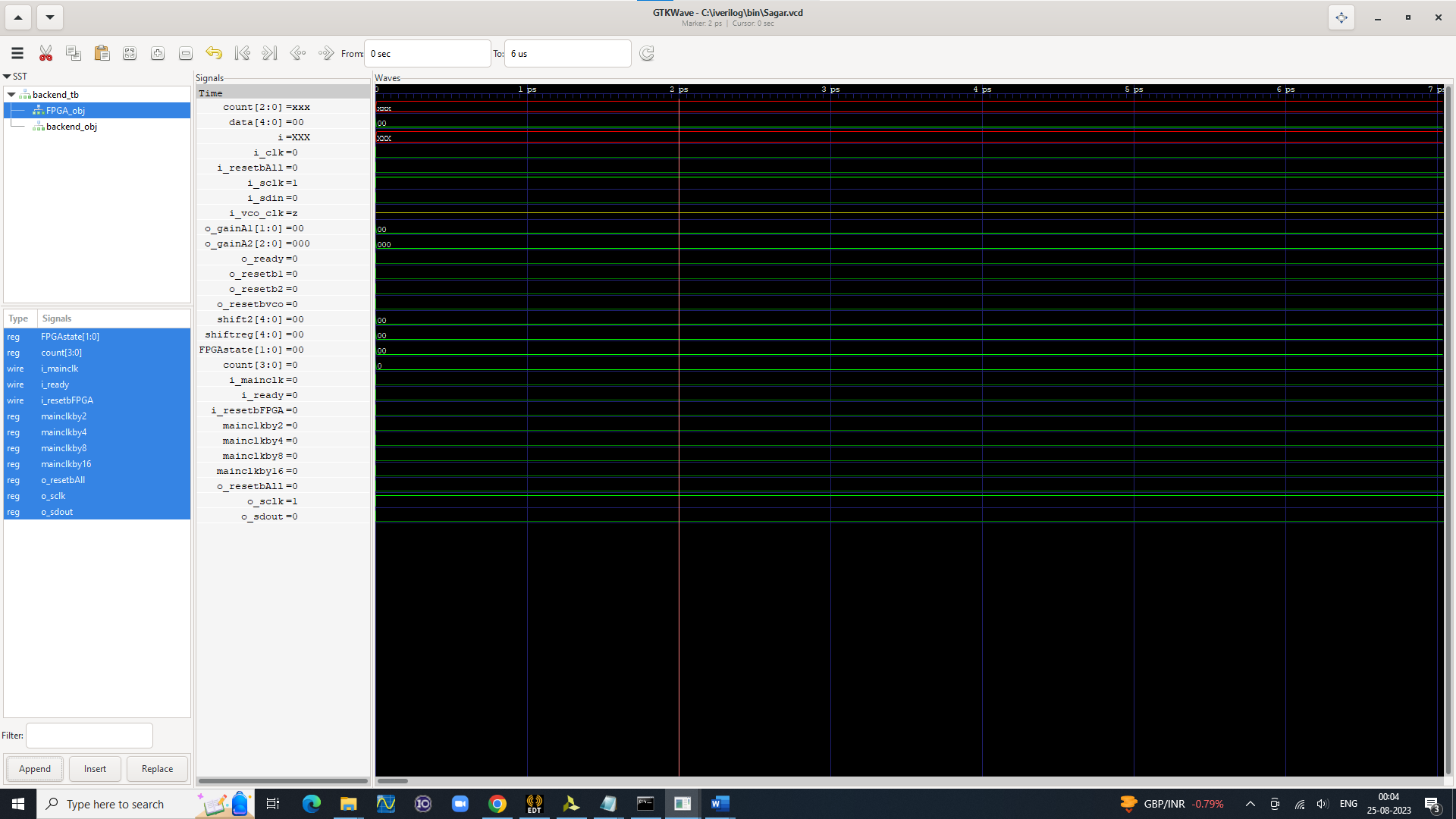
initial

#6000 $finish;

//============================================================================

Endmodule

**SIMULATION OUTPUT:**

****