**Code:**

* **Design Module**
* **parameter Module**

`define Bus\_Length 16

* **ALU Module**

// Code your design here

// Takes two numbers; perform the desired operation(+, -, x, /)

`include"parameter.sv"

module calculator(input reset, //ports

input [(`Bus\_Length) -1:0] first\_num, // first operand

input [(`Bus\_Length) -1:0] second\_num, // second operand

// 2 bits operation 00 = plus; 01 = minus; 10=mult; 11=div;

input [1:0] operation,

output [((`Bus\_Length)\*2) -1:0] result);

//internal regs

reg [((`Bus\_Length)\*2) -1:0] temp\_out;

// logic

always@(\*)

begin

if(reset)

begin

temp\_out <= 0;

end

else

begin

if(operation == 2'b0)

temp\_out = first\_num + second\_num; // Add operation

else if(operation == 2'b01)

temp\_out = second\_num - first\_num; // sub operation

else if(operation == 2'b10)

temp\_out = first\_num \* second\_num; // multiplication operation

else if(operation == 2'b11)

temp\_out = first\_num/second\_num; // divide operation

end

end// end always block

// reg to wire

assign result = temp\_out;

endmodule

* **Register Module**

// Code your design here

`include"parameter.sv"

module reg\_file(input clk, //ports

input regwrite, // operation

input [$clog2(`Bus\_Length) -1 :0] writereg,

input [$clog2(`Bus\_Length) -1 :0] readreg1,

input [$clog2(`Bus\_Length) -1 :0] readreg2,

input [(`Bus\_Length) -1 :0] data,

output [(`Bus\_Length) -1 :0] read1,

output[(`Bus\_Length) -1 : 0] read2);

//internal regs

reg [(`Bus\_Length) -1 :0] out1;

reg [(`Bus\_Length) -1 :0] out2;

reg [(`Bus\_Length) -1 :0] reg\_device [(`Bus\_Length) : 0 ]; // internal memory

//logics

always@(posedge clk)

begin // value @ pose & neg edge both

out1 <= reg\_device[readreg1];

out2 <= reg\_device[readreg2];

end

always@(negedge clk)

begin

out1 <= reg\_device[readreg1];

out2 <= reg\_device[readreg2];

if(regwrite)

begin

reg\_device [writereg ] <= data;

end

end

//reg to wire

assign read1 = out1;

assign read2 = out2;

endmodule

* **Design Module**

// Code your design here

// Takes two numbers; perform the desired operation(+, -, x, /)

`include"parameter.sv"

`include"reg\_file.sv"

`include"ALU.sv"

module Processor( //regfile ports

input clk, //ports

input regwrite, // memory operation

input [$clog2(`Bus\_Length) -1 :0] writereg,

input [$clog2(`Bus\_Length) -1 :0] readreg1,

input [$clog2(`Bus\_Length) -1 :0] readreg2,

input [(`Bus\_Length) -1 :0] data, //input data

//alu ports

input reset,

input [(`Bus\_Length) -1:0] first\_num, // first operand

input [(`Bus\_Length) -1:0] second\_num, // second operand

input [1:0] operation,

output [((`Bus\_Length)\*2) -1:0] result);

//internal regs

reg [(`Bus\_Length) -1 :0] read1;

reg[(`Bus\_Length) -1 : 0] read2;

reg [((`Bus\_Length)\*2) -1:0] temp\_out;

//module instantiations

begin

reg\_file internal\_memory(clk,regwrite,writereg,readreg1,readreg2,data,read1,read2);

calculator alu(reset,read1,read2,operation,temp\_out);

end

//reg to wire

assign result = temp\_out;

endmodule

* **TestBench**

// Code your testbench here

// or browse Examples

// Code your testbench here

// or browse Examples

module tb();

bit clk; //ports

reg regwrite; // memory operation

reg [$clog2(`Bus\_Length) -1 :0] writereg;

reg [$clog2(`Bus\_Length) -1 :0] readreg1;

reg [$clog2(`Bus\_Length) -1 :0] readreg2;

reg [(`Bus\_Length) -1 :0] data; //input data

//alu ports

bit reset;

reg [(`Bus\_Length) -1:0] first\_num; // first operand

reg [(`Bus\_Length) -1:0] second\_num; // second operand

reg [1:0] operation;

wire [((`Bus\_Length)\*2) -1:0] result;

//instantiations

Processor dut(clk,regwrite,writereg,readreg1,readreg2,data,reset,first\_num,second\_num,operation,result);

// clock

always #5 clk = ~clk;

//initial setup ky leay

initial

begin

clk <= 1;

reset <= 0;

operation <= 0; //addition

regwrite <= 1;

writereg <= 32'b1001; //data for at location x

data <= 32'b1111;

#5

regwrite <= 1;

writereg <= 32'b1010; //data for at location y

data <= 32'b1101;

#5

#10

regwrite <= 0; //writing stopped

readreg1 <= 32'b1001; // location x

readreg2 <=32'b1010; // location y

#10;

end

// asal change auroperation testing of the system

initial

begin

#30

regwrite <= 0; //writing stopped

writereg <= 32'b010001;

data <= 32'b100001;

#15

readreg1 <= 32'b1001; // location x

readreg2 <=32'b1010; // location y

#10

operation <= 1;

#10

operation <= 2;

#10

operation <= 3;

#20

reset <= 1;

end

//wave production in eda play ground

// initial

// begin

// $dumpfile("dump.vcd");

// $dumpvars;

// #150

// $finish;

// end

endmodule

**Output:**

A screenshot of a computer

Description automatically generated with medium confidence