5193 FPGA and Verilog HDL

Summer 2019

Assignment 4 – GCD Design

|  |  |  |
| --- | --- | --- |
| Group Member | ID | Contribution |
| Yeum, Daniel | xzd304 |  |
| Manzano, Oscar | uyz655 |  |

|  |  |
| --- | --- |
| Date Submitted |  |
| Design |  |
| Synthesis |  |
| Implementation & Demo |  |
| Report: Approach |  |
| Report: Discussions |  |

**Table of Contents**

Problem Statement ……………………………………………………………………………………………... Pg. 3

Explanation of Approach ………………………………………………………………………………………. Pg. 3

Problems Encountered at Simulation and Implementation ……………………………………………………. Pg. 3

Block Diagram …………………………………………………………………………………………………. Pg. 4

Algorithmic State Machine ……………………………………………………………………………………. Pg. 5

Verilog Codes Used ………………………………………………………………………………………… Pg. 5-24

Simulation Waveforms ………………………………………………………………………………………. Pg. 24

**Problem statement:**

In this assignment, we are to design a processor to calculate the Greatest Common Divisor of two integers implemented using Euclidian’s Algorithm.

**Explanation of Approach:**

Based on the steps for Euclideans’ Algorithm, we knew the processor would need to have a subtractor, comparators, and a few multiplexers. Our first approach was to create the data path to understand the logic of the processor, this consisted of creating two inputs going through a select line and into registers, because the inputs going into the starting registers would be different depending on the state the single bit select line would be necessary. The outputs would feed into either the comparator, subtractor, or the register to output the final value. The output of the subtractor would feed back as an input to the starting input registers and would continue to do so until some point that the two values are equal. Our approach for the data path was to instantiate each component inside as a module and use wires to make the necessary connections to the control unit. Therefore, we created a module for the two input registers, output register, multiplexers as the select line for the inputs, a subtractor, and a comparator. Most of the modules used in this assignment were used in the previous assignment, such as the subtractor and multiplexers.

Furthermore, we designed the control unit with a four state, state machine, with the first state (S0) being the start and to load the input values into the registers. At the first state the decision if the inputs are equal to, greater than or not greater than, and all else would be determined for the next states. The second state (S1) would output the final value should the two starting values already be equal. The third state (S2) would go to the subtractor if the first input (In0) be greater than the second input (In1), and the fourth state (S3) would do the same if the second input were greater than the first. After, the data path module and control unit module, the top module would instantiate the two with wires and output the value by encoding the value onto the board, similar to the past assignment.

**Problems Encountered at Simulation and Implementation:**

The output to display onto the board the way we wanted was tedious and challenging. There were things such as getting the counter the right speed to show every state or calculation of the inputs and the resulting output. Initially, the simulation would not display the final output, this was because of wiring issues and some faults in the code for the submodules.

**Block Diagram:**

A close up of a map

Description automatically generated

**Algorithmic State Machine:**

A close up of a map

Description automatically generated

**Verilog Codes Used:**

module GCD(

input CLK100MHZ, rst, //start

input SW1,

output reg [7:0] a\_to\_g,

output reg [7:0] an

);

parameter start = 1'b1;

parameter [15:0] In0 = 16'h0321;

parameter [15:0] In1 = 16'h0123;

reg [3:0] LED\_BCD;

wire [3:0] LED\_activating\_counter;

reg [19:0] refresh\_counter;

wire valid, Wr\_In0, Wr\_In1, Wr\_C, Sel\_In0, Sel\_In1, Sel\_a, Sel\_b; //

wire eq, gth;

wire [31:0] C, I0\_out, I1\_out;

wire [1:0] State\_Y;

GCD\_datapath DP(CLK100MHZ, rst, Wr\_In0, Wr\_In1, Wr\_C, Sel\_In0, Sel\_In1, Sel\_a, Sel\_b, In0, In1, eq, gth, C, I0\_out, I1\_out);

GCD\_controlunit CU(CLK100MHZ, rst, start, eq, gth, valid, Wr\_In0, Wr\_In1, Wr\_C, Sel\_In0, Sel\_In1, Sel\_a, Sel\_b, State\_Y);

always @(posedge CLK100MHZ) begin

if(rst)

refresh\_counter <= 0;

else

refresh\_counter <= refresh\_counter + 1;

end

assign LED\_activating\_counter = refresh\_counter[19:17];

always @(\*) begin

case(SW1)

1'b1: begin

case(LED\_activating\_counter)

3'b000: begin

an = 8'b01111111;

LED\_BCD = I1\_out[31:28];

end

3'b001: begin

an = 8'b10111111;

LED\_BCD = I1\_out[27:24];

end

3'b010: begin

an = 8'b11011111;

LED\_BCD = I1\_out[23:20];

end

3'b011: begin

an = 8'b11101111;

LED\_BCD = I1\_out[19:16];

end

3'b100: begin

an = 8'b11110111;

LED\_BCD = I1\_out[15:12];

end

3'b101: begin

an = 8'b11111011;

LED\_BCD = I1\_out[11:8];

end

3'b110: begin

an = 8'b11111101;

LED\_BCD = I1\_out[7:4];

end

3'b111: begin

an = 8'b11111110;

LED\_BCD = I1\_out[3:0];

end

endcase

end

1'b0: begin

case(LED\_activating\_counter)

3'b000: begin

an = 8'b01111111;

LED\_BCD = In0[15:12];

end

3'b001: begin

an = 8'b10111111;

LED\_BCD = In0[11:8];

end

3'b010: begin

an = 8'b11011111;

LED\_BCD = In0[7:4];

end

3'b011: begin

an = 8'b11101111;

LED\_BCD = In0[3:0];

end

3'b100: begin

an = 8'b11110111;

LED\_BCD = In1[15:12];

end

3'b101: begin

an = 8'b11111011;

LED\_BCD = In1[11:8];

end

3'b110: begin

an = 8'b11111101;

LED\_BCD = In1[7:4];

end

3'b111: begin

an = 8'b11111110;

LED\_BCD = In1[3:0];

end

endcase

end

endcase

end

always @(LED\_BCD) begin

casex(LED\_BCD)

4'b0000: a\_to\_g = 8'b1\_0000001;

4'b0001: a\_to\_g = 8'b1\_1001111;

4'b0010: a\_to\_g = 8'b1\_0010010;

4'b0011: a\_to\_g = 8'b1\_0000110;

4'b0100: a\_to\_g = 8'b1\_1001100;

4'b0101: a\_to\_g = 8'b1\_0100100;

4'b0110: a\_to\_g = 8'b1\_0100000;

4'b0111: a\_to\_g = 8'b1\_0001111;

4'b1000: a\_to\_g = 8'b1\_0000000;

4'b1001: a\_to\_g = 8'b1\_0001100;

4'b1010: a\_to\_g = 8'b1\_0001000;

4'b1011: a\_to\_g = 8'b0\_1100000;

4'b1100: a\_to\_g = 8'b1\_0110001;

4'b1101: a\_to\_g = 8'b1\_1000010;

4'b1110: a\_to\_g = 8'b1\_0110000;

4'b1111: a\_to\_g = 8'b1\_0111000;

default: a\_to\_g = 8'bX\_0000000;

endcase

end

endmodule

//--------------------------------------------------------------------------------

module display(

input [15:0] a,b,

input CLK100MHZ, rst,

output reg[7:0] a\_to\_g,

output reg[7:0] an

);

reg [3:0] LED\_BCD;

wire [3:0] LED\_activating\_counter;

reg [19:0] refresh\_counter;

always @(posedge CLK100MHZ) begin

if(rst)

refresh\_counter <= 0;

else

refresh\_counter <= refresh\_counter + 1;

end

assign LED\_activating\_counter = refresh\_counter[19:17];

always @(\*) begin

case(LED\_activating\_counter)

3'b000: begin

an = 8'b01111111;

LED\_BCD = a[15:12];

end

3'b001: begin

an = 8'b10111111;

LED\_BCD = a[11:8];

end

3'b010: begin

an = 8'b11011111;

LED\_BCD = a[7:4];

end

3'b011: begin

an = 8'b11101111;

LED\_BCD = a[3:0];

end

3'b100: begin

an = 8'b11110111;

LED\_BCD = b[15:12];

end

3'b101: begin

an = 8'b11111011;

LED\_BCD = b[11:8];

end

3'b110: begin

an = 8'b11111101;

LED\_BCD = b[7:4];

end

3'b111: begin

an = 8'b11111110;

LED\_BCD = b[3:0];

end

endcase

end

always@(LED\_BCD)begin

casex(LED\_BCD)

4'b0000: a\_to\_g = 8'b1\_0000001;

4'b0001: a\_to\_g = 8'b1\_1001111;

4'b0010: a\_to\_g = 8'b1\_0010010;

4'b0011: a\_to\_g = 8'b1\_0000110;

4'b0100: a\_to\_g = 8'b1\_1001100;

4'b0101: a\_to\_g = 8'b1\_0100100;

4'b0110: a\_to\_g = 8'b1\_0100000;

4'b0111: a\_to\_g = 8'b1\_0001111;

4'b1000: a\_to\_g = 8'b1\_0000000;

4'b1001: a\_to\_g = 8'b1\_0001100;

4'b1010: a\_to\_g = 8'b1\_0001000;

4'b1011: a\_to\_g = 8'b0\_1100000;

4'b1100: a\_to\_g = 8'b1\_0110001;

4'b1101: a\_to\_g = 8'b1\_1000010;

4'b1110: a\_to\_g = 8'b1\_0110000;

4'b1111: a\_to\_g = 8'b1\_0111000;

default: a\_to\_g = 8'bX\_0000000;

endcase

end

endmodule

//--------------------------------------------------------------------------------

module GCD\_controlunit(

input clk, rst, start, eq, gth,

output valid, Wr\_In0, Wr\_In1, Wr\_C, Sel\_In0, Sel\_In1, Sel\_a, Sel\_b,

output [1:0] State\_Y

);

reg [7:0] Control\_Variable;

reg [1:0] state, nstate;

//wire eq, gth;

wire valid, Wr\_In0, Wr\_In1, Wr\_C, Sel\_In0, Sel\_In1, Sel\_a, Sel\_b; //

//wire [1:0] State\_Y;

parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;

//Reset and Update State

always @(posedge rst or negedge clk) begin

if(rst) state <= 2'b00;

else state <= nstate;

end

//Next State

always@(state or start or eq or gth) begin

case(state)

2'b00: begin if(~start) nstate <= 2'b00;

else if(eq) nstate <= 2'b01;

else if(gth) nstate <= 2'b10;

else nstate <=2'b11;

end

2'b01: nstate <= 2'b00;

2'b10: begin if(eq) nstate <= 2'b01;

else if(gth) nstate<=2'b10;

else nstate<=2'b11;

end

2'b11: begin if(eq) nstate <= 2'b01;

else if(gth) nstate<=2'b10;

else nstate<=2'b11;

end

default: nstate<=2'b00;

endcase

end

//Output

always @(state) begin

case(state)

2'b00: Control\_Variable <= 8'b1100\_1100;

2'b01: Control\_Variable <= 8'b0000\_0011;

2'b10: Control\_Variable <= 8'b0011\_1000;

2'b11: Control\_Variable <= 8'b0000\_0100;

endcase

end

assign Sel\_In0 = Control\_Variable[7];

assign Sel\_In1 = Control\_Variable[6];

assign Sel\_a = Control\_Variable[5];

assign Sel\_b = Control\_Variable[4];

assign Wr\_In0 = Control\_Variable[3];

assign Wr\_In1 = Control\_Variable[2];

assign Wr\_C = Control\_Variable[1];

assign valid = Control\_Variable[0];

assign State\_Y = nstate;

endmodule

//--------------------------------------------------------------------------------

module GCD\_datapath(

input clk, rst, Wr\_In0, Wr\_In1, Wr\_C, Sel\_In0, Sel\_In1, Sel\_a, Sel\_b,

input [15:0] In0, In1,

output eq, gth,

output [15:0] C,

output [15:0] I0\_out, I1\_out

);

wire [15:0] I0\_wire, I1\_wire, ALU;

wire [15:0] A\_wire, B\_wire;

RegisterIn0 Reg0(clk, rst, Wr\_In0, Sel\_In0, In0, ALU, I0\_wire);

RegisterIn1 Reg1(clk, rst, Wr\_In1, Sel\_In1, In1, ALU, I1\_wire);

RegisterC RegC(clk, rst, Wr\_C, I0\_wire, C);

MuxA MA(I0\_wire, I1\_wire, Sel\_a, A\_wire);

MuxB MB(I0\_wire, I1\_wire, Sel\_b, A\_wire);

ALU A(A\_wire, B\_wire, ALU);

Comparator Com(I0\_wire, I1\_wire, eq, gth);

assign I0\_out = I0\_wire;

assign I1\_out = I1\_wire;

endmodule

//------------------------------------------------------------------------------

module RegisterIn0(

input clk,

input rst,

input Wr\_In0,

input Sel\_In0,

input [15:0] In0,

input [15:0] ALU,

output [15:0] out

);

reg [15:0] I0\_wire;

always @(posedge rst or posedge clk) begin

if(rst)

I0\_wire <= 16'h0000;

else if(Wr\_In0) begin

if(Sel\_In0)

I0\_wire <= In0;

else

I0\_wire <= ALU;

end

end

assign out = I0\_wire;

endmodule

//-----------------------------------------------------------------------------

module RegisterIn1(

input clk,

input rst,

input Wr\_In1,

input Sel\_In1,

input [15:0] In1,

input [15:0] ALU,

output [15:0] out

);

reg [15:0] I1\_wire;

always @(posedge rst or posedge clk) begin

if(rst)

I1\_wire <= 16'h0000;

else if(Wr\_In1) begin

if(Sel\_In1)

I1\_wire <= In1;

else

I1\_wire <= ALU;

end

end

assign out = I1\_wire;

endmodule

//-----------------------------------------------------------------------------

module RegisterC(

input clk,

input rst,

input Wr\_C,

input [15:0] I0\_wire,

output [15:0] out

);

reg [15:0] C;

always @(posedge rst or posedge clk) begin

if(rst)

C <= 16'h0000;

else if (Wr\_C)

C <= I0\_wire;

// else

// C<= 0; /////

end

assign out = C;

endmodule

//-----------------------------------------------------------------------------

module MuxA(

input [15:0] I0\_wire,

input [15:0] I1\_wire,

input Sel\_a,

output reg [15:0] A\_wire

);

always @(I0\_wire or I1\_wire or Sel\_a) begin

if(Sel\_a)

A\_wire <= I0\_wire;

else

A\_wire <= I1\_wire;

end

endmodule

//-----------------------------------------------------------------------------

module MuxB(

input [15:0] I0\_wire,

input [15:0] I1\_wire,

input Sel\_b,

output reg [15:0] B\_wire

);

always @(I0\_wire or I1\_wire or Sel\_b) begin

if(Sel\_b)

B\_wire <= I1\_wire;

else

B\_wire <= I0\_wire;

end

endmodule

//-----------------------------------------------------------------------------

module ALU(

input [15:0] A\_wire,

input [15:0] B\_wire,

output reg [15:0] ALU

);

always @(A\_wire or B\_wire) begin

ALU <= A\_wire - B\_wire;

end

endmodule

//-----------------------------------------------------------------------------

module Comparator(

input [15:0] I0\_wire,

input [15:0] I1\_wire,

output eq,

output gth

);

assign eq = (I0\_wire == I1\_wire) ? 1 : 0;

assign gth = (I0\_wire > I1\_wire) ? 1 : 0;

endmodule

**Simulation Waveform:**

**A screenshot of a cell phone

Description automatically generated**

Figure : GCD sim