SLOW DIVISION ALGORITHM CODING PART EXPLANATION

In this line define the module named as “restoringDivision” with specified some inputs and outputs

**module restoringDivision(clk,rst,start,X,Y,valid,quot,rem);**

In this lines declare the input and outputs of the module

**input clk;**

**input rst;**

**input start;**

**input [7:0] X, Y;**

**output [7:0] quot, rem;**

**output valid;**

Inputs are clk,rst,start ,X,Y and outputs are quot,rem,valid.

These line declare the internal registers and signals are used in the module

**reg [15:0] Z, next\_Z, Z\_temp, Z\_temp1;**

**reg next\_state, pres\_state;**

**reg [3:0] count, next\_count;**

**reg valid, next\_valid;**

“Z” is 16-bit register and this hold the current value of the divinded.

“next\_Z” is next value of Z.

“Z-temp” and”Z temp1” are temporary registers.

“next\_state” and “pres\_state” indicates the present and next states of finite state machine.

“count” and “next count” holds the current and next values of the counter.

“ Valid” and” next\_valid “ this check that is whether the result is valid or not.

In this line declare the two parameters for finite state machine.

**parameter IDLE = 1'b0;**

**parameter START = 1'b1;**

“IDLE” is representing the idle state and assigning the value 1’b0.

“START” is represent the start state of division operation and assigning the value by 1’b1.

In this line assign the value for reminder and qoutient using the upper 8 bits for “rem” and the lower bits for “quot”

**assign rem = Z[15:8];**

**assign quot = Z[7:0];**

This allows the values to be accessed and externally through the output.

This the a Always block

**always @ (posedge clk or negedge rst)**

**begin**

**if (!rst)**

**begin**

**Z <= 16'd0;**

**valid <= 1'b0;**

**pres\_state <= 1'b0;**

**count <= 4'd0;**

**end**

**else**

**begin**

**Z <= next\_Z;**

**valid <= next\_valid;**

**pres\_state <= next\_state;**

**count <= next\_count;**

**end**

**end**

The always block that triggers the either the positive edge of the clock or negative edge of the rst signal.

It handles the reset conditions, behavior of the registers and output .

If the rst value is active low then it reset the value of register Z,valid,pres\_state, and count to their initial values. Else it updates the register to the respective next values.

This is another always block

This always block that trigger whenever any changes made in the present state and it is a use case statement is based on the current state like pres\_state.

This state is used to determine the next state and update the values of next\_Z,next\_valid,next\_count and next state registers.

**always @ (\*)**

**begin**

**case (pres\_state)**

**IDLE:**

**begin**

**next\_count = 4'b0;**

**next\_valid = 1'b0;**

**if (start)**

**begin**

**next\_state = START;**

**next\_Z = {8'd0, X};**

**end**

**else**

**begin**

**next\_state = pres\_state;**

**next\_Z = 16'd0;**

**end**

**end**

When in the IDLE state, it sets the next\_count and next \_valid to their initial. If the start signal is activated high then it transitions to the START state and modify the next\_Z register with concatenation of 8’d0 and value X. else it remains the “IDLE” state and set the “next\_z” to 0.

**START:**

**begin**

**next\_count = count + 1'b1;**

**Z\_temp = Z << 1;**

**Z\_temp1 = {Z\_temp[15:8] - Y, Z\_temp[7:0]};**

**next\_Z = Z\_temp1[15] ? {Z\_temp[15:8], Z\_temp[7:1], 1'b0} : {Z\_temp1[15:8], Z\_temp[7:1], 1'b1};**

**next\_valid = (&count) ? 1'b1 : 1'b0;**

**next\_state = (&count) ? IDLE : pres\_state;**

**end**

**endcase**

**end**

In the “START” state, it increments the “next\_count” by 1 then shifts the current value of “z” (using the assign operator) to left by 1 .And this value is assign to “Z\_temp”. The Z\_temp subtarcted by Y and the value is assigned to the “Z\_temp1”. The “next\_Z “ is update based on the “Z\_temp1”.

The “next\_valid” is assign to 1 if the “count “ is true else 0.

The “next\_state” is assign to “IDLE” when “count “ is true else it remains the “START” state.

FAST DIVISION ALGORITHM CODE PART EXPLANATION

These line define the module by named “Fast\_division” with specified some inputs and outputs.

**module fast\_division (**

**input wire signed [15:0] dividend,**

**input wire signed [7:0] divisor,**

**output reg signed [15:0] quotient,**

**output reg signed [7:0] remainder,**

**output reg done**

**);**

The inputs are dividend and divisor and the outputs are quotient and reminder, done flag.

This line initialize the internal registers used for the module and each register is declare as “signed” and it has specified bit width.

**reg signed [15:0] div\_op;**

**reg signed [7:0] div\_op\_abs;**

**reg signed [7:0] shift\_reg;**

**reg signed [15:0] quotient\_reg;**

**reg signed [7:0] count;**

**reg [3:0] state;**

The Always Trigger Block

**always @(\*) begin**

**case (state)**

**0: begin**

**if (dividend[15] == 1)**

**div\_op = -dividend;**

**else**

**div\_op = dividend;**

**div\_op\_abs = (div\_op[15] == 1) ? (~div\_op + 1) : div\_op;**

**shift\_reg = divisor;**

**count = 8;**

**quotient\_reg = 0;**

**state = 1;**

**end**

This always block triggers whenever any changes made in the inputs and it is a use case to find the curret state and performs the respective operations.If state is 0 it checks the most signficant bit of the dividend, if it’s 1 the “div\_op” is assign to the negation of the “dividend”. else “div\_op” is assign to the “dividend”. The “div\_op” values is stored in the “div\_op\_abs” using conditoinal operator.

“div\_op” is 1 its peformed 2’s complement by bitwise complement and by adding 1. “shift\_reg “and “quotient\_reg” is set their values and count is count the no of iteration.

In state no 1 the block checks the most significant bit of “div\_op, if it’s 0 it shifts the “shift\_reg” and decrement the count by 1 and it done “div\_op” is positive or zero(The state transitioned to 2). Else its 1 it shifts the ‘shift\_reg” and decement the count by 1 and it is done “div\_op” is negative. (The state transitioned to 3)

**1: begin**

**if (div\_op[15] == 1'b0) begin**

**shift\_reg = shift\_reg << 1;**

**count = count - 1;**

**state = 2;**

**end**

**else begin**

**shift\_reg = shift\_reg >> 1;**

**count = count - 1;**

**state = 3;**

**end**

**End**

In the state no 2 it checks tha absolute value of “div\_op” is greater than or equal to the “shift\_reg”

**2: begin**

**if (div\_op\_abs >= shift\_reg) begin**

**div\_op = div\_op - shift\_reg;**

**quotient\_reg[count] = 1;**

**end**

**shift\_reg = shift\_reg >> 1;**

**count = count - 1;**

**if (count == 0)**

**state = 4;**

**else**

**state = 1;**

**End"**

“div\_op” value is updated by subtracting and right shift operation is performed on the “shift\_reg”,and “count” and the value are decremented by 1.If count reaches 0(the state transitioned to 4). else it go to the state 1.

In state no 3,the “div\_op” is updated and a right shift operation is performed and it check the count is 0 it go to the complete state otherwise it go to the state 1.

**3: begin**

**div\_op = div\_op + shift\_reg;**

**quotient\_reg[count] = 0;**

**shift\_reg = shift\_reg >> 1;**

**count = count - 1;**

**if (count == 0)**

**state = 4;**

**else**

**state = 1;**

**End**

In state no 4 or completion state

**4: begin**

**remainder = div\_op;**

**quotient = quotient\_reg;**

**done = 1;**

**state = 0;**

**end**

**endcase**

**end**

**endmodule**

The final value of “reminder” and “quotient” is assigned as “div\_op” and “quotient\_reg”. And the “done“ value is assigned to 1 for the division algorithm completed.