**Department of Electrical and Computer Engineering**

Author: Zainab Hussein

Title: HW 4

Date: 3-1-2017

1. 18x18-bit multiply accumulate (MACC) using a single Artix-7 DSP whose Verilog diagram is shown in figure1

module MACC(

input logic clk, reset, enb,

input logic [17:0] A, B,

output logic [35:0] AccVal

);

MACC\_MACRO #(

.DEVICE("7SERIES"), // Target Device: "7SERIES"

.LATENCY(3), // Desired clock cycle latency, 1-4

.WIDTH\_A(25), // Multiplier A-input bus width, 1-25

.WIDTH\_B(18), // Multiplier B-input bus width, 1-18

.WIDTH\_P(48) // Accumulator output bus width, 1-48

) MACC\_MACRO\_inst (

.P(AccVal), // MACC output bus, width determined by WIDTH\_P parameter

.A(A), // MACC input A bus, width determined by WIDTH\_A parameter

.ADDSUB(ADDSUB), // 1-bit add/sub input, high selects add, low selects subtract

.B(B), // MACC input B bus, width determined by WIDTH\_B parameter

.CARRYIN(CARRYIN), // 1-bit carry-in input to accumulator

.CE(CE), // 1-bit active high input clock enable

.CLK(clk), // 1-bit positive edge clock input

.LOAD(enb), // 1-bit active high input load accumulator enable

.LOAD\_DATA(LOAD\_DATA), // Load accumulator input data, width determined by WIDTH\_P parameter

.RST(reset) // 1-bit input active high reset

);

// End of MACC\_MACRO\_inst instantiation

endmodule

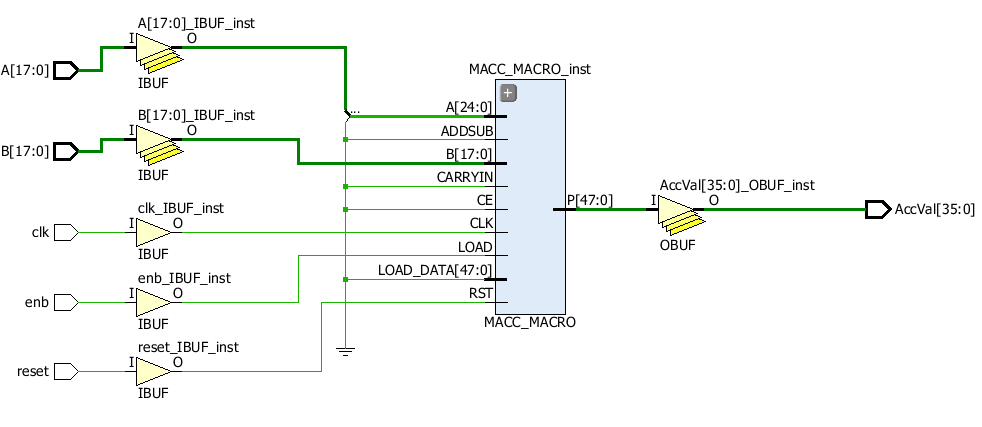


Figure MACC elaborated design

1. Signed fixed-point multiplication

int main() {

// 16-bits => short,

short int fixptmul(short int input1, short int input2){

int product;

product = input1 \* input2;

return product;

}

}

1. Multiply two complex numbers

int main() {

void complexmul(short int a[2], short int b[2], short int prod[2]){

//reminder (a+bi).(c+di) = ac+adi+cbi+bidi

//ac-bd at 0th, cbi+adi at 1st

product[0] = fixptmul(a[0], b[0]) – fixptmul(a[1], b[1]);

product[1] = fixptmul(a[0], b[1]) + fixptmul(a[1], b[0]);

}

}

1. Guess integer thought by user using C code

void main(){

int nguesses; //guesses count

int guess; //user guess

int small; //smaller range

int big; //bigger range

int make\_guess\_result;

int correct;

while(1) {

small = -1; //smaller than lowest possible value =0-1

big = 101; //bigger than highest possible value =100+1

guess = 0;

nguesses = 0;

int make\_guess\_result = 0;

correct = 0; //correct guess

while(small != big) { //guess range specified (0-100)

guess = (big + small)/2; //start middle

nguesses =+ 1; //increase guess count

//user interaction

make\_guess\_result = make\_guess(guess);

if(make\_guess\_result == -1)

small = guess;

else if(make\_guess\_result == 1)

big = guess;

else if( make\_guess\_result == 0)

break;

}

}

return(EXIT\_SUCCESS);

}

1. Acceptance testing plan:

|  |  |  |
| --- | --- | --- |
| Test to perform | Test implementation | Pass/Fail |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Tests to perform

 Straight on scan of a number of codes

 Scan shots at a variety of angles

 Scan shots under high lighting and low lighting

 Scan a dirty code

 Scan a creased code

 Scan in cold environment (like from freezer or refrigerator)

 Perform all of the above from a computer, tablet, and phone

Testing implementation

 Have both Windows and Mac computers and iOS and Android tablets and phones to test

with

 Have codes setup in display to scan

 Use template or holding rig to provide consistent angle scans

 Use both with controllable lighting to provide high and low lighting conditions

 Maintain same dirty and creased codes (possibly print dirty code with false imperfections)

1. Time spent 5 hours