LUND
UNIVERSITY

ETIN45 – DSP Design

Laboratory Manual

Lab 2: Design Verification

Mojtaba Mahdavi

Jan. 2018

1. Lab preparations

Read again Chapter 1 and the first two pages of Chapter 2 in the ac_datatypes_uv.pdf document. Make sure you understand the concepts of the datatypes ac_int and ac_fixed. Get familiar with the concept of a Radix-2 butterfly (https://en.wikipedia.org/wiki/Butterfly_diagram)

NOTE:
You should perform all tasks and also fill Table I. Then, ask the TA to check your results to get approved in this lab.

2. Assignment 1 – Testbench Walkthrough

An important part of hardware development is verification. In this lab you will verify the design from lab 1. The following code is the testbench to be used in this lab. Study it and try to figure out how it works.

Note:
In this part of the lab, you should use the code that you wrote in Task 3 of Assignment 2 in Lab1.

```
#include "ac_int.h";
#include <iostream>
#include <fstream>
// Include the C/C++ function header
#include "lab1.h"
// Include the SCVerify header
#include "mc_testbench.h"
int main(int argc, char *argv[]){
  ac_int<8> a = 1;
  ac_int<8> b = 2;
  ac_int<8> result = 0;
  // Test simuli. Five iterations.
  for (int s_idx = 0; s_idx < 5; s_idx++) {
    result = f(a,b,s_idx);
    // Generate some output
    std::cout << "a*b*c = result: " << a.to_int() << "*"
<< b.to_int() << "*"<< s_idx << " = " << result.to_int()
<< std::endl;
  }
return 0;
}
```

2.1. Task 1: Verification

Perform the following steps:

1. Add a file named "lab2_tb.cpp" to your lab directory (where you stored lab1).
2. Start Catapult using the instructions from lab 1.
3. Add the design file corresponding to Task 4.3 in Lab1 to your project.
4. Add "lab2_tb.cpp" to the Catapult project by right clicking the "Input Files" in the "Project Files" pane. Make sure to check the "Exclude" checkbox.
5. Copy the testbench code from this manual to the empty file ("lab2_tb.cpp") that you added to the project.
6. Pipeline the design with II=1, set the frequency to 50MHz and the technology to the Sample 90nm (check lab 1 if you forgot how to do this)
7. Click on the "RTL" in the "Synthesis Tasks" to complete the synthesis process.
8. Enable "SCVerify" in the Flow Manager tab.
9. Now run the simulation for the testbench as follows:
    - Expand f.v1-> Verification -> gcc 4.2.2
    - Double clicking "Original Design + Testbench".
10. Check the console window – what is the output of the testbench?
11. Note down the total area, the area for the multipliers and the area for the register and fill in Table I.

3. Assignment 2 – Datatypes

The present dataypes in the design and the testbench are ac_int<8> meaning that the variables are signed integers with 8 bits in total.

3.1. Task 1: Reduce bitwidth

Change the datatype for the output to 3 bits (you have to change both in the design and in the testbench). Re-run the project (click "Generate RTL").

- What is the total area, the area for the multipliers and the area for the register? Write all requested results in Table I.
- Compare your result with the 8 bit case and note/discuss the differences.

Run the test bench (step 8 in the Testbench Walkthrough).

- What is the result?

3.2. Task 2: ac_fixed

Change the datatypes of all variables (a,b,c and result) to 8 bit signed fixed point datatypes with the range of -1 to 0.9921875 (almost 1).

- What are your selected values for template parameters in "ac_fixed" data type?

You must also add the following line to the design (lab1.h) to include the header file: '`include ''ac_fixed.h`'. Consult the ac_datatypes_uv.pdf (Chapter 2) document if you forgot the specification of the ac_fixed datatype.

Change also the testbench and correct all possible errors. (Tip: change from "x.to_int() to x.to_double() in the output print code).

- Run the testbench and analyze the result of five multiplications. Note that the initial values of a, b, c are the same as in previous task.
- What are the result of five multiplications? Are they as expected? If not why?
- Keep 8 bits for a, b, c, and result and change the other template parameters in "ac_fixed" data type to fix this issue. You should be able to see the correct multiplication results for initial values of a=1, b=2, c=0 as you did in previous task ('c' should be increased by one in each of 5 iterations).
- Run the test bench again and check the results and fill in the corresponding row in Table 1.

3.3. Task 3: Butterfly Radix-2

In this task you will design a Radix-2 butterfly unit that is the base computational element in an FFT.

- Modify the following code to implement the butterfly unit. Note that the inputs of butterfly are two complex numbers. Replace the question marks with template parameters of your choice.

```
#include ''ac_fixed.h''

void butterfly(const ac_fixed<?> &x_0_re,
               const ac_fixed<?> &x_0_im,
               const ac_fixed<?> &x_1_re,
               const ac_fixed<?> &x_1_im,
               ac_fixed<?>       &y_0_re,
               ac_fixed<?>       &y_0_im,
               ac_fixed<?>       &y_1_re,
               ac_fixed<?>       &y_1_im){


  y_0_re = 0; //Modify this code
  y_0_im = 0; //Modify this code
```

```
  y_1_re = 0; //Modify this code
  y_1_im = 0; //Modify this code
}
```

Now, write a testbench to test your design. Modify the testbench in the 'Testbench Walkthrough' section to perform the design verification.

- Run the testbench for five different sets of inputs and make sure that the functionality of design is correct.
- Display the output of butterfly for these five input sets.
- Set the clock frequency to 50MHz and try out Initiation Intervals (II) of 1 and 2.
- Write the corresponding results in Table 1 and analyze the results.

Table I. Synthesis results of different runs for design in Lab 2.

					Area of		
	II	Latency	Throughput	Max Delay	Mult/Add	Registers	Total
2.1-Task1							
3.1-Task1							
3.2-Task2							
3.3-Task3	1						
3.3-Task3	2						