MAC Unit with BIST

Author : Erez Binyamin & Derek Freeman

Lab Instructor	:	Alman Ganguly
TA	:	Andrew Fountain
TA	:	Sabrina Ly
Lab Section	:	L2
Submission Date	:	12/09/19

Table of Contents

ABSTRACT

In this activity a layout for a Multiply and Accumulate (MAC) unit accompanied by a built in self test (BIST) was synthesized using VHDL code to describe the hardware and the Pyxis auto-layout tool to generate the transistor level design. The design was verified using VHDL syntax checking, simulation analysis, DRC transistor design rule checks, and LVS layout design checks. This activity was successful.

DESIGN METHODOLOGY

The design of a MAC with a BIST is actually comprised of a 1-bit full adder, N-Bit ripple carry adder, Multiplier, LFSR, and a MISR units. The 1-bit full adder, LFSR, and MISR were all designed behaviorally. The N-bit ripple carry, Multiplier, MAC Unit, and the top level MAC_BIST were all designed structurally.

i) Multiplier Design

The Multiplier design was done using the method of traditional binary multiplication (Fig 3-4 in Appendix file: Multiplier_ALU.pdf). Partial product vectors were created using an array of ‘AND’ gates and then the partial products were accumulated using an array of ripple carry adders. The advantage of taking this approach to hardware multiplication generates much simpler code (< 20 lines code for the multiplication logic) and synthesizes to a design with grouped hardware components. The advantage of grouped hardware is to increase speed and decrease synthesis faults (Fig 2 in Appendix file: Multiplier_ALU.pdf).

ii)Adder Design

The adder design used for the MAC unit involved a naive ripple carry design. The design was structurally defined using a behaviorally described 1bit full adder. The ripple carry adder simple instantiates ‘N’ 1 bit full adders and connects all of the previous stages Cout’s to the next stages Cin’s. The name ripple carry comes from the effect of the propagation of the carry signal from the LSB adder unit to the MSB adder unit via the connected Cout – Cin pairs.

iii)Fixed Point Arithmetic

From the definition of a “fixed point number” we learn that is simply a representation of a value (number) that has a constant number of digits after the decimal point (fixed point). Fixed point arithmetic, as it is pedantically named, is math (arithmetic) that is done using fixed point numbers. Fixed point numbers are also so defined to conveniently distinguish them from the more complex “floating point numbers”. Fixed point arithmetic was achieved in this design by ensuring that the inputs would always have a fixed number of digits. The task of holding the inputs at a constant number of digits was accomplished by using the VHDL syntax for the deceleration of an input of type STD_LOGIC_VECTOR which requires a bit length specification for the input signal, and inherently ensures that stated bit length does not change.

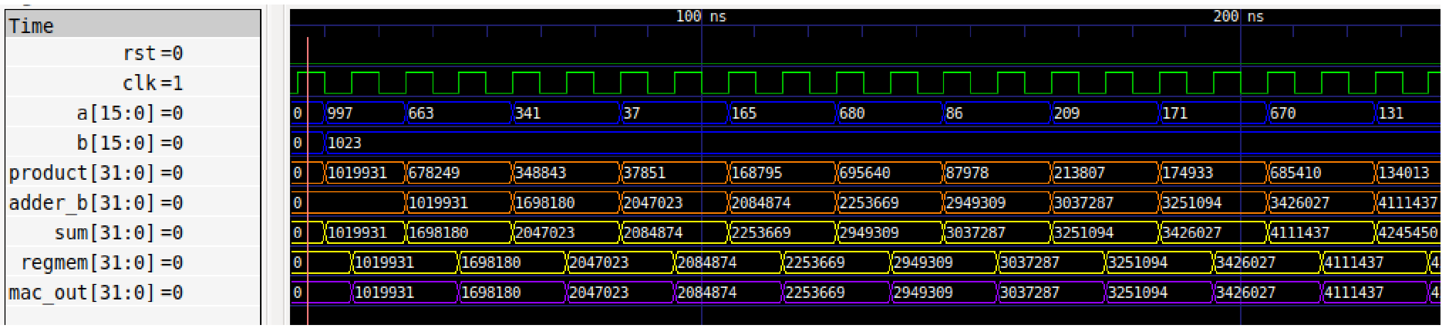
iv)Shift Vs. Parallel MAC register

The MAC register was not designed as a separate component but rather as an internal signal within the MAC unit structural design. Due to this lack of specific definition, the description of weather the register was designed as parallel or shift may seem vague. The closest thing to a full answer would be to state that the MAC register likely synthesized as a Parallel register. Internal signals in VHDL seldom become shift registers, and it is not necessarily likely for them to become parallel registers either, however it is more likely that the synthesis decision to create a parallel register would be made rather than a shift register.

RESULTS & ANALYSIS

The MAC unit was tested with a testbench design that made use of "ieee.math_real.uniform" library to generate random inputs. By using this library to generate random inputs to the MAC unit the design could be reasonably expected to function properly with hardware that generates random inputs.

MAC Unit Simulation


Time
100 ns
200 ns
rst=0
clk=1
a[15:0]=0 0 997 663 341 37 165 680 86 209 171 670 131
b[15:0]=0 0 1023
product[31:0]=0 0 1019931 678249 348843 37851 168795 695640 87978 213807 174933 685410 134013
adder_b[31:0]=0 0 1019931 1698180 2047023 2084874 2253669 2949309 3037287 3251094 3426027 4111437
sum[31:0]=0 0 1019931 1698180 2047023 2084874 2253669 2949309 3037287 3251094 3426027 4111437 4245450
regmem[31:0]=0 0 1019931 1698180 2047023 2084874 2253669 2949309 3037287 3251094 3426027 4111437 4
mac_out[31:0]=0 0 1019931 1698180 2047023 2084874 2253669 2949309 3037287 3251094 3426027 4111437 4

From top to bottom: reset (rst), clock (clk), A input (A), B input (B), product of A*B (product), secondary b input to adder unit from the storage register (adder_b), the sum of product and adder_b (sum), signal to carry stored register memory (regmem), the output from the MAC unit (mac_out)

In the simulation above the A input can be seen changing while the B input remains the same, this was done to test multiple A values against a single B value. For verification of the multiplier unit, one need only look at the signals: A, B, and product (blue, blue, orange). It can be seen clearly that the multiplier is functioning properly:

A	B	Product
997	1023	1019931
663	1023	678249
341	1023	348843
37	1023	37851

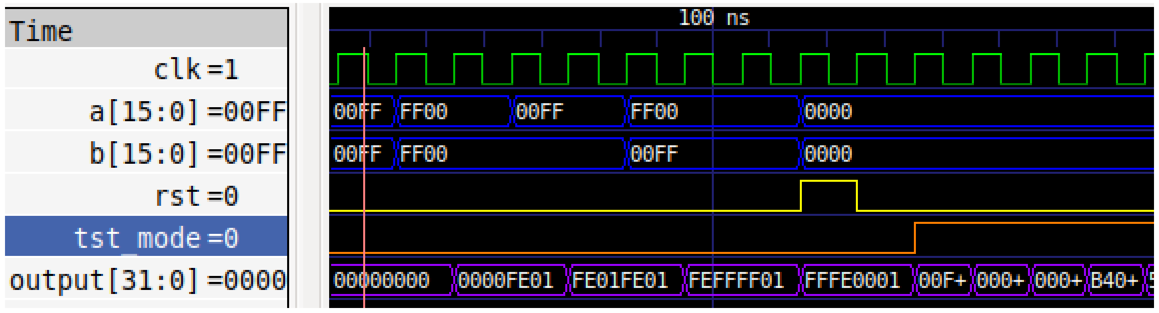
It can also be seen that the adder is working properly to add product (orange) and adder_b (orange) to make sum (yellow):

Product	Adder_b	Sum
1019931	0	1019931
678249	1019931	1698180
348843	1698180	2047023
37851	2047023	2084874

One might notice that the value of the adder_b signal is always equal to the previous value of sum. This is due to the accumulating nature of a MAC unit. The product is being added to the adder_b signal in this example. Functionally, the product is being added to the *previous* sum to generate the *next* sum.

The design that consisted of both the BIST and the MAC unit was tested similarly and it's functionality was verified.

MAC with BIST

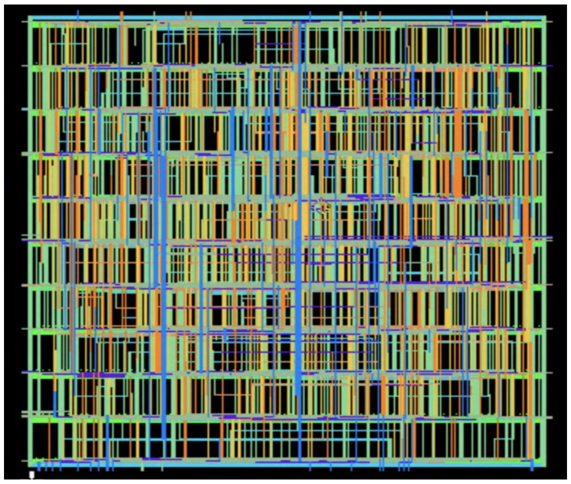


From top to bottom: clock (clk), A input (a), B input (b), reset (rst), test mode control (tst_mode), design output (output)

In the simulation above the MAC can be seen being tested with edge case values (Combinations of 00FF, FF00) with tst_mode (orange signal) set at 0. When tst_mode=0 the A and B inputs assigned from the testbench are the values that enter the MAC and the output is the output from the MAC. At around 140ns the tst_mode signal goes high and the inputs to the MAC become the output from the LFSR and the output from the MAC becomes the output from the MISR. The MISR signatures were shown to be unique for all unique outputs from the MAC.

This VHDL design was processed and auto-instantiated as a layout using the Pyxis design suite. The layout generated by Pyxis is shown below:

Pyxis MAC with BIST layout



The layout shown here is for a 16bit input 32bit output MAC Unit wrapped by a LFSR-MISR BIST

This design passed DRC and LVS checks and is ready for timing and power analysis simulations. The timing and power analysis was not done in this activity but could easily be taken up by another engineer to move this design one step closer to fabrication.

CONCLUSIONS

This activity summarizes all skills necessary to do digital integrated circuit design (DIGIC). VHDL code was written and tested using ghdl and gtkwave simulation tools. The written hardware description code was then synthesized into a transistor layout. The path from logical design to chip fabrication has been fully realized and the relevant tests to perform and evaluate along the way have been practiced and skills have been fine tuned. This lab activity and this course have been successful in their objectives.