E-club Project documentation

Team name

EMANON

Team members

Ankit Jalan, Kavita Meena, Manu Seth

Team mentors

Shubham Singh, Khagesh Patel

Project title

FPGA scientific calculator

Aim

To build a scientific calculator on FPGA board using Verilog as HDL, this would handle floating point numbers (32-bit, single precision).

Motivation

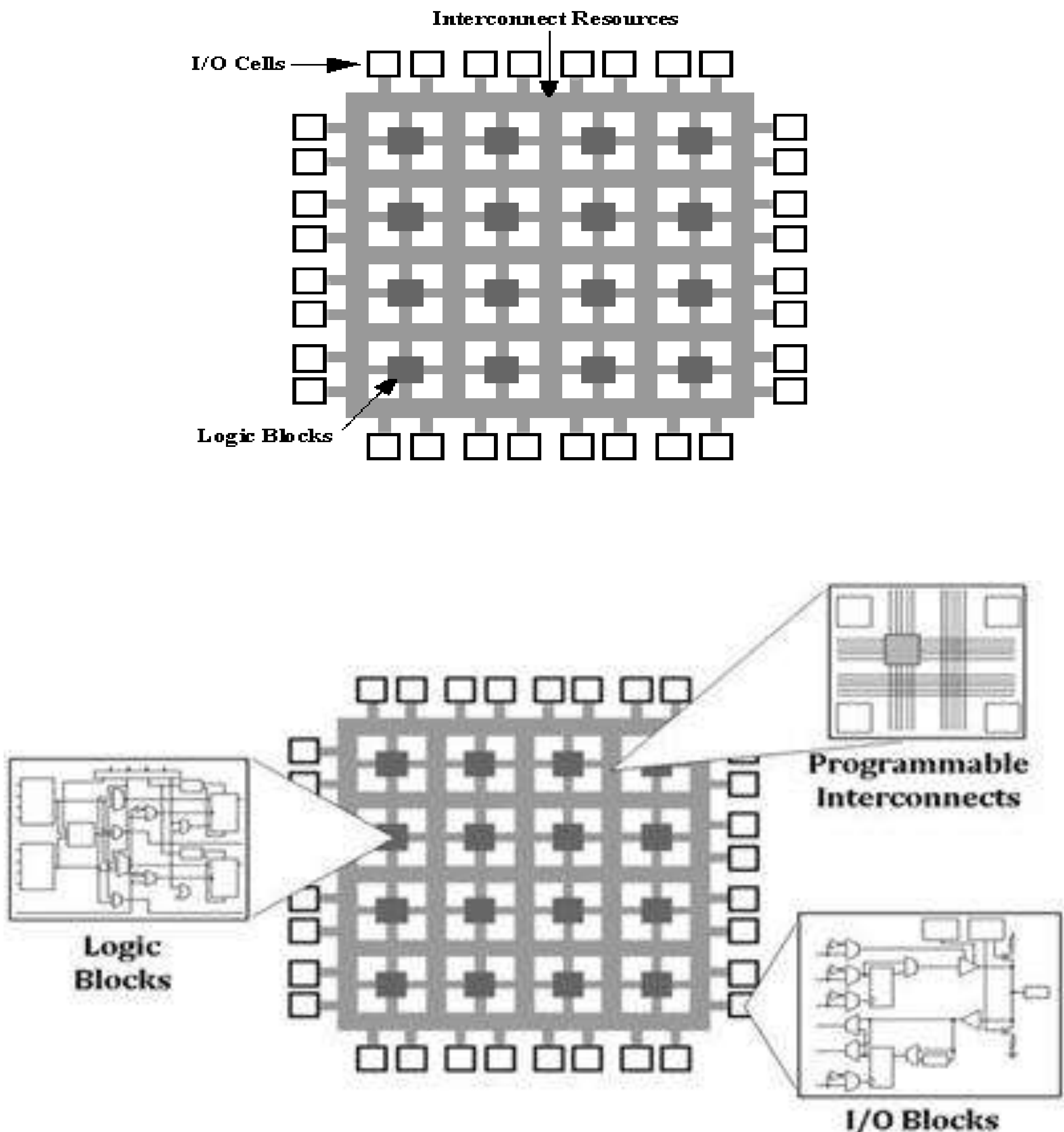
All the complex integrated circuits, before being hardwired and commercially produced, are tested on FPGA board. The circuit of those IC's are firstly coded using a hardware description language and is then synthesized. These synthesizable codes are then implemented on FPGA so that the circuit and its efficiency could be tested and improved.

This motivated us to learn about the HDL's and get a hand-on experience on the FPGA board. Since this was a start, we discussed with the coordinators about the possible projects which we could undertake. Finally we were able to come up with a decision of making a scientific calculator on FPGA.

A brief idea about FPGA module

FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together"—somewhat like many (changeable) logic gates that can be inter-wired in (many) different configurations. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

FPGA's have logic blocks, input-out ports and interconnects which are programmable, which provides flexibility to the range of logic circuits which can be configured on the board.



A brief idea about VERILOG

At the time of Verilog's introduction (1984), Verilog represented a tremendous productivity improvement for circuit designers who were already using graphical schematic capture software and specially written software programs to document and simulate electronic circuits. The designers of Verilog wanted a language with syntax similar to the C programming language, as it was already widely used in engineering software development. Like C, Verilog is case-sensitive and has a basic preprocessor (though less sophisticated than that of ANSI C/C++). Hardware description languages such as Verilog differ from software programming languages because they include ways of describing the propagation of time and signal dependencies (sensitivity).

A Verilog design consists of a hierarchy of modules. Modules encapsulate design hierarchy, and communicate with other modules through a set of declared input, output, and bidirectional ports. Internally, a module can contain any combination of the following: net/variable declarations (wire, registers, integer, etc.), concurrent and sequential statement blocks, and instances of other modules (sub-hierarchies).

Verilog concept of 'wire' consists of both signal values (4-state: "1, 0, floating, undefined") and strengths (strong, weak, etc.). This system allows abstract modeling of shared signal lines, where multiple sources drive a common net. When a wire has multiple drivers, the wire's (readable) value is resolved by a function of the source drivers and their strengths.

There are two assignment operators, a blocking assignment (=), and a non-blocking (<=) assignment. The non-blocking assignment allows designers to describe a state-machine update without needing to declare and use temporary storage variables. Since these concepts are part of Verilog's language semantics, designers could quickly write descriptions of large circuits in a relatively compact and concise form. Its control flow keywords (if/else, for, while, case, etc.) are equivalent, and its operator precedence is compatible. Syntactic differences include variable declaration (Verilog requires bit-widths on net/reg types[clarification needed]), demarcation of procedural blocks (begin/end instead of curly braces {}), and many other minor differences.

Sequential statements are placed inside a begin/end block and executed in sequential order within the block. However, the blocks themselves are executed concurrently, making Verilog a dataflow language.

Subsets of statements in Verilog language are synthesizable. Verilog modules that conform to a synthesizable coding style, known as RTL (register-transfer level), can be physically realized by synthesis software. Synthesis software algorithmically transforms the (abstract) Verilog source into a netlist, a logically equivalent description consisting only of elementary logic primitives (AND, OR, NOT, flip-flops, etc.) that are available in a specific FPGA or VLSI technology. Further manipulations to the netlist ultimately lead to a circuit fabrication blueprint, a bit-stream file for an FPGA.

Software used

We have used *ISE project navigator* provided by *Xilinx* and used *Verilog* as our HDL.

Procedure:

Coding is generally done on the text editor of ISE design suite. Following are the steps to build a module in the design suite.

- Firstly, we select "new project" in the menu which comes when we run the software.
- A window will appear seeking the mane of project.
- After clicking on next button, a window will appear to select the source type. We are working on Verilog, so we will select Verilog module and then click ok.
- Then, we will define input/output ports and the type of ports viz single or bus type.
- After completing the code we will simulate it on ISim SIMULATOR by selecting behavioral model simulating.

A sample module is written here to get a brief idea of the syntax of module.

After a module is written we can check the syntax with the inbuilt function of check-syntax in the software. If the syntax is correct, we are ready to simulate the code but if the syntax is incorrect, we are suggested, by the software, possible mistakes.

A sample module:

```
//-----------------------------------------------------
//one-bit full adder module
//-----------------------------------------------------
module fadder(
input a, //data in a
input b, //data in b
input cin, //carry in
output sum_out, //sum output
output c_out //carry output
);
wire c1, c2, c3; //wiring needed
assign sum_out = a ^ b ^ cin; //half adder (XOR gate)
assign c1 = a * cin; //carry condition 1
assign c2 = b * cin; //carry condition 1
assign c3 = a * b; //carry condition 1
assign c_out = (c1 + c2 + c3);
endmodule
```

Algorithm used for making a calculator

A 32 bit floating point number in IEEE754 standard has 3 parts. The bit-32 is sign bit. This is '0' for a positive number and is '1' for a negative quantity. The bit [31:23], eight bits are the bits assigned for the exponent part. The exponent has value 127 added to the original value.

Thus is the original exponent value is 4, the equivalent exponent in the 32 bit number will be 4 + 127 = 131.

The next 23 bits, i.e. bit [22:0] refers to mantissa. The mantissa is actually 24 bits. The first bit is assumed to be 1. Hence the mantissa is of the form 1. ..01..[23 bits].

Thus a 32 bit floating point number is if the form

$$N = \text{mantissa} \times 2^{exp}.$$

Now, since the Verilog circuits are able to do integer arithmetic, they are not able to perform floating point arithmetic. We have to manipulate the numbers part by part, i.e. we have to separately handle sign, exponent, and the mantissa. Since part – by part arithmetic will be integral, we are able to do so.

Thus, for multiplication we added the exponent and multiplied the mantissa. For division we subtracted the exponent and wrote a module to calculate the modulo of the mantissa. For addition we manipulated the numbers so that they can be represented in a way that the exponents are same, and in that way we were able to handle addition and subtraction. For the factorial and Fibonacci part, these were integral operations so these were not much of a worry about.

The main problem we faced was the conversion of the output result after these arithmetic operations is representing the output in the standard IEEE754 format.

We overcame this problem by making some case-switch statements and manipulating the number bit-by-bit.

Task completed

We have made the modules of basic arithmetic functions for 32-bit floating point numbers like division, multiplication, addition subtraction, inverse and some more additional functions like Fibonacci, factorial for natural numbers.

All the codes written for the above operations are giving correct simulation results and are synthesizable.

Task left

We aimed at implementing a scientific calculator that would take input from the keyboard and print the output on the screen. But due to shortage of time, we were not able to write the codes for keyboard input and screen display. Also, if we had time we would have written a master module that would control all the other modules that we have developed and that would pass the inputs according to the operator sought.

Useful links

- EBook for basic FPGA prototyping by verilog examples:
  http://books.google.co.in/books?id=z8XhRwmWpeQC&printsec=frontcover&dq=fpga+prototyping+in+verilog&hl=en&sa=X&ei=eEvFUYaLG4GJrAe83YDoDQ&ved=0CC0Q6AEwAA

- Xilinx For Windows :
  http://www.xilinx.com/support/download.html

- Verilog tutorials:
  http://www.asic-world.com/verilog/veritut.html

A word of thanks

We would like to thank our mentors Khagesh Patel and Shubham Singh for their guidance, patience, suggestions and their belief in us.
They inspired us to learn a lot and work on this exciting project and checked the progress of our project so sincerely that even when sometimes we were like frustrated and hopeless regarding our project they showed us the way.
Moreover, it was fun doing this project in summers as we got to learn so much, we learned Verilog and certainly it will broaden our viewpoint in the world of IC synthesis.