EE Times News & Analysis BETA

News & Analysis

PROJECT: You don't need a fab to build your own CPU!

Marco Aiello

6/16/2010 3:31 PM EDT

The complete system

As you can see from figure 1, the system is made up of the MIC 1 CPU, an eight bit rom memory used to store the program, a 32 bit rom used to store the constants of the 'constant pool' area, a 32 bit ram, a full duplex uart (I haven't used FIFO buffers on the tx line modifying instead the OUT instruction with the insertion of a micro programmed delay in it), and some glue logic used to implement the decode of the addressed component by the CPU.

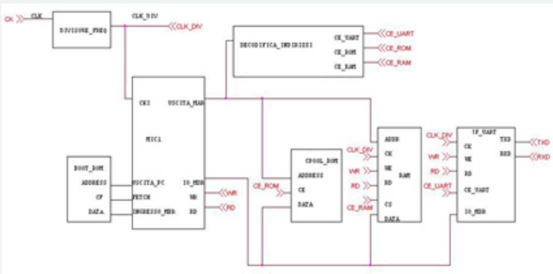

Fig 1: Here's the overall picture of our embedded system. Thanks to FPGA's technology, you can look at the Spartan 3 device as a big box of bricks with which you can play around.

To see a bigger version of this graphic click here.

Given the synthetized version of the CPU can run at a frequency slower than the one given by the quartz oscillator present on the board, I've had to use a frequency divider (the DCIM component) in order to get a clean and scaled down version of the original clock signal.

Using Digital Clock Managers (DCMs) in Spartan-3 FPGAs(reference 2) explains how you can raise or lower the clock frequency and do many other things without adding an external oscillator on the dip socket provided on the board.

The only external resource used, is a simple RS232 cable and a pc running a terminal program, as you can see by figure 1. As on modern machines there's the lack of such a gadget, I've resorted to an USB serial dongle plugged directly in the serial connector of the board.

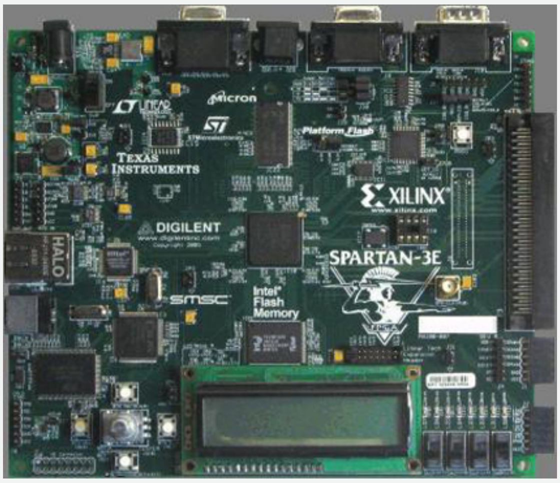

Fig 2: The board offers a lot of peripherals. There's even a VGA plug. I've used only a serial connector in my project, so the whole system can be hosted on a cheaper and simpler board.

As you can understand from the VHDL code posted on the FTP site, describing the system (the 'sistema_mic1.vhd file), once you have developed the components, it is a very simple to assemble the embedded system. Thanks to the board's design and the low usage of FPGA resources, this project can be used as the starting point for more challenging embedded systems.

« 1 **2** 3 4 5 »

PROJECT: You don't need a fab to build your own CPU!

Marco Aiello

6/16/2010 3:31 PM EDT

The MIC 1

The biggest part of the design has been with no doubt the CPU. As my teacher has explained countless times during the course, it’s always a good design habit to divide every machine, no matter what the complexity, into two parts: a datapath and a control unit (microcoded or hardwired, according to your taste). With this wise advice well in mind, I’ve structured the design hierarchically into a datapath and a control unit of course.
With regard to the latter, I have to admit that the most of the work has been already done by Professor Tanenbaum.

Indeed, thanks to the content of the microprogram memory (also called the control store memory to keep the book’s nomenclature) generously provided in reference 1, it’s only a tedious matter of translating all into VHDL statements in order to describe the rom itself.

As you can see from the file 'control_store.vhd' (posted on the FTP site with all the whole source code of the system), it takes more than four thousands lines of code to describe this component, making it with no doubt, the more difficult and error prone part in the system to describe about. But I insist to emphasize that the difficulty has been found only into the translation task in binary format of the symbolic microinstructions given into the book.

The project made use of Xilinx’s ISE program and I’ve maintained the same hierarchy and names given in reference 1 to describe the CPU into VHDL. The other few components of the control part are the 'oring' circuit (described in the 'oring.vhd' file), the micro program counter (described in the 'mpc.vhd' file), the circuit used to get the high bit (stored in the 'high_bit.vhd' file), and lastly a decoder (placed in the '4to16decoder.vhd' file). The whole control part is described in the file 'pc.vhd' and uses only the aforementioned components, connecting them appropriately as described reference 1.

The datapath description is a no more challenging task than the control part one.

Its components are the registers not involved with the ports to the memory (described in the 'high_bit.vhd' file) as the mar, pc, sp, lv,cpp,tos, opc, h, and the remaining two involved with input output operations from and for memory: mbr and mdr.

The mdr register can be found in the file 'mdr_register.vhd', it differs by the aforementioned registers for the presence of a multiplexer placed on its input, in order to switch to it the input on the C bus ; or the one present on the 32 bit data bus.

The mbr register is on 'mbr.vhd' and it is no more than a regular eight bit positive edge triggered register, followed by a sign extension block, and a multiplexer used to select the signed or unsigned extended 32 bit version of the content of the register, according to the control signals.

There are then some negative edge triggered D flip flop used to shift in the right amount the 'fetch' signal used to issue read operations to the eight bit rom memory (running a simulation and watching carefully the whole activity on the datapath you can understand in full the essence of their presence), they are described in the file 'negedge_dff.vhd'.

In the same way, a couple of positive edge triggered D flip flop instead, is used to shift correctly, the 'read' and 'write' signals, in order to allow correct accesses to and from peripherals plugged onto the 32 bit data bus.

A negative edge triggered register is used to get the right timing of the control signals used to delay the enable signals for the writing of the value placed onto the C bus in the right registers, it’s described in the file 'neg_edge_reg.vhd'.

The ALU is placed onto the file 'alu.vhd', while the shifter is in 'shifter.vhd'.

The file 'parte_operativa.vhd ' instantiates all the above components and, besides, adds a multiplexer used to route the output of only a register at once onto the B bus; according to the control signals from the control unit, and a buffer tri state between the output of the mdr register and the data bus. All the CPU is 'assembled' in the file 'mic1.vhd', and it simply joins together the datapath and the control unit, moreover it shifts in the right amount the output signal of the mar register to the address bus lines (as shown and explained on reference 1).

And so we get the MIC 1. I suggest a careful reading of the VHDL source codes (which are fully commented) to get a full understanding of the whole system, thousands of VHDL code are much more explicative than few pages of concise descriptions; of course you must take with you a good supply of eyewash because you’ll need it (I’m joking of course)!

« 1 2 **3** 4 5 »

News & Analysis

PROJECT: You don't need a fab to build your own CPU!

Marco Aiello

6/16/2010 3:31 PM EDT

The calculator's program

Thanks to the strict adherence of the FPGA's version of the MIC 1 system with the one on the simulator (see figure 3) you can safely develop your firmware on the friendly and comfortable environment provided by the latter, and only then, to start the tedious writing process of the program onto the 'rom8.vhd' file and the constant pool values onto the 'constant_pool_rom.vhd' one.

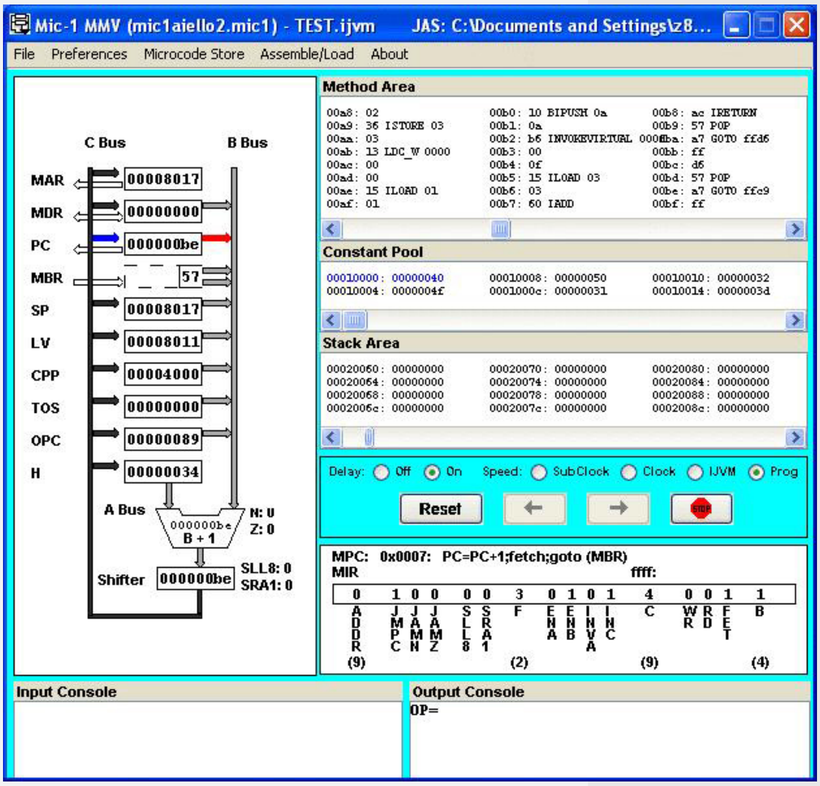

Fig 3: On this screenshot you can see the MIC 1 software simulator. The FPGA version is made using it as a model.

Looking at figure 3, once your program is correctly assembled what you see on the 'Method area' window of the simulator must be hosted into the eight bit rom, while what is onto the 'Constant Pool' one, goes into the 32 bit constant pool rom instead.

When you write a program for the soft core version of the CPU, take into account furthermore the physical dimensions of every memory used, or you will risk for example to incur into a stack overflow/underflow problem, or in a reading of an incorrect value. All this will surely lead you to hours and hours spent at the simulator scratching your head to figure out what has gone wrong, and this isn't a pleasant experience of course! At every change made in one of both of these two files, you have to restart the whole time consuming process of synthesis, translation, mapping, placement and routing, and only after all this lengthy process, you can get the programming file generation ready to download into your own board.

The simulator supports a full understanding of what happens behind the scene; you can set different execution speeds to catch the signals on the data path and the output control signals of the control store rom. I have to admit that it has played a key role in the developing stage of the VHDL description of the MIC 1, because looking at what happens on it helps a lot in the understanding of the CPU's architecture and behavior.

Now let me provide a few words about the demo program I've used for my examination.

As you can see by the source file 'TEST_IMUL_IDIV.jas' posted on the FTP web site, it is very simple, and its purpose is the provision of a simple and articulate task to the whole system in order to show its usefulness and functionality in a 'real' application.

The program is almost self explicating (and fully commented): it simply polls the 'input console' represented by the RXD line of the system's UART receiver (taken as is from reference 5) and searches for a received byte with the 'IN' instruction which gives 0x00 in the lack of a data, or its value (not a 0x00) if there's something.

It waits for the reception of a couple of numbers of two digits (represented with in ASCII format). When it gets them, performs their ASCII to binary conversion, their subsequent multiplication and division (getting the product, quotient and the remainder), the ASCII conversion of the results, and using the 'output console', alias the UART's transmitter (taken as is from reference 5 too), sends the results coded in ASCII, directly on the PC's screen.

PROJECT: You don't need a fab to build your own CPU!

Marco Aiello

6/16/2010 3:31 PM EDT

If something can go wrong it will!

Never have wiser words been true! Almost every time in your life, not always all goes as you planned out. I've learned this lesson well! Indeed, even if the ISE environment doesn't issues no more severe errors or warnings, it doesn't imply that your design works. Once again, even if you get a correct performance of the whole system in the behavioral simulation, you can't be sure at all neither then ; that once you download the program bitstream onto the board's FPGA, it will run right. This is due to many causes, most of them are imputable to a poor VHDL coding style. This is often true particularly for beginners as me, and the task of finding the source of those errors is very hard because the simulation goes fine, and you can't get any hint by it.

Thanks to reference 3, I've learned well the lesson and I have finally acquired a quite decent coding skill, avoiding the use of strange constructs ; that even though allowed (the VHDL has a lot of non synthetizable constructs useful only for simulation purposes), led me to unpredictable results.,

Another pitfall is represented by more subtle timing issues, indeed you must never ignore the plenty of messages given in the 'Transcript' window of the ISE environment. Only after a careful reading of its content, I've had a very bitter surprise: the whole system could run at a maximum frequency well under the one provided by the board's oscillator, so that's why I've used the DCIM component. Prior to use it, I've wrongly described a frequency divider into VHDL, without taking into account the pitfalls given by the placer, the router, the use of various clock domains, and the severe issue of the clock skew.

Thanks to *Advanced FPGA design, architecture,implementation and optimization* (reference 4), I've assimilated well this other lesson too, learning by my many mistakes.

The post translate simulation is a quite valid indicator of your VHDL correct coding: indeed now you can understand if the XST suite has done well its job or not, comparing the behavior of this simulation with the one of the behavioral simulation and searching for a match between both of them.

With the post map and the post route simulation you can see with a reasonable precision the effect of glitches and spurious transitions on all the signals of the system, given by the delay introduced by logic and interconnections, that can lead sometime to an incorrect behavior. But don't think now that your pain is ended, even a simple error on the constraint file can prevent you to get a success !

Alas, I've wrongly twisted the RX an TX line on the board's connector, and I've suffered a lot prior to catch this last ; but nonetheless, subtle mistake.

After this all time consuming odyssey, I've learned that the theory is a beautiful thing such as all seems always simple, elegant and perfect, but the practice is all another hard and dirty story ! So don't dig yourself into this effort if you don't have with you a good amount of hot coffee (or tea according to your taste) and your MP3 player loaded with your favorite songs (Metallica's one of course in my case), to hold high your attention level during the many hours that are awaiting you behind a pc's screen.

To be continued . . .

I have to admit that it is a very great satisfaction to apply what you study for years! Only in this way you realize that the leap between theory and practice is quite thin, and that all the strange things you learn on books, can be very useful in the real word and can be combined together in a creative and new way to get out something useful. Of course, a little bit of practice is a great added value to it!

Learning many things from many disciplines is a very good things too, in fact as Hagakure wrote, "A person who is said to be proficient at the arts is like a fool. Because of his foolishness in concerning himself with just one thing, he thinks of nothing else and thus becomes proficient. He is a worthless person."

It has been very interesting to be faced to all the developing stages of a real CPU. As a one semester university project, it suffers of several limitations. For example the control store memory could be implemented using the Spartan 3's distribuited or block memory used as a rom.

Other room for improvement is given in reference 1, in which you are comfortably driven by hand, in the many improvements possible on the architecture. It should be a good idea too, to use the external rom and sdram memory whit which the board is equipped, in order to lower further the percentage of utilization of the FPGA's resources.

Such as the CPU is a microcoded one, you can develop from scratch a new ISA ; simply changing the content of the sole control store ROM, or designing a new datapath right from the start.

It should be also nice to add a PS2 keyboard controller and a VGA text controller as the ones described in FPGA prototyping by VHDL examples (reference 5) in place of the simple UART.

So what other to say, welcome to you on the new and shiny wagon of embedded systems on FPGA !

References

1. *Modern computer organization*, fifth edition, Tanenbaum, Prentice Hall

2. *Using Digital Clock Managers (DCMs) in Spartan-3 FPGAs*XAPP462.

3. *RTL hardware design using VHDL, coding for efficiency, portability, and scalability*, Pong P. Chu, Wiley Interscience.

4. *Advanced FPGA design, architecture,implementation and optimization*, Steve Kilts, Wiley Interscience.

5. *FPGA prototyping by VHDL examples*, Pong P. Chu, Wiley Interscience.

Resources

1. *Computer organization*, Hamacker, Vranesic, Zaky, Mc Graw Hill, fifth edition

2. *Vhdl programming by example, fourth edition, Perry, Mc Graw Hill*

3. http://en.wikipedia.org/wiki/Intel_4004

4. http://en.wikipedia.org/wiki/Federico_Faggin

5. http://en.wikipedia.org/wiki/Andrew_S._Tanenbaum

6. http://www.ontko.com/mic1/

See online for more information on the Digilent board and on the MIC1 simulator.

About the author

Marco Aiello (caccolillo@yahoo.com) is a student at the fifth (and last) year in electronic engineering at the university of Naples Federico Secondo in Italy. At the time of writing he's four examination to get a degree in electronic engineering. His interests include artificial intelligence in general (neural networks, machine vision, expert systems, genetic algorithms applied to neural

networks training), vhdl design, embedded systems and electronic design (both analog and digital), digital control. In his spare time, he loves to listen Metallica's songs and watch old Japanese films on Samurai, especially the Akira Kurosawa's ones.