POLITECNICO DI TORINO
Repository ISTITUZIONALE

Algorithm validation and hardware design interactive approach

Original
Algorithm validation and hardware design interactive approach / Lazarescu, MIHAI TEODOR; Sartori, M.. - ELETTRONICO. - 1:(1996), pp. 291-294. (Intervento presentato al convegno International Semiconductor Conference tenutosi a Sinaia, Romania nel October 1996) [10.1109/SMICND.1996.557380].

Availability:
This version is available at: 11583/2507491 since: 2020-07-05T15:51:24Z

Publisher:
IEEE

Published
DOI:10.1109/SMICND.1996.557380

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
default_article_editorial [DA NON USARE]

-

(Article begins on next page)

12 October 2024

Algorithm validation and hardware design interactive approach*

ing. Mihai-Teodor LĂZĂRESCU
Polytechnic of Turin, Electronics department
C.so Duca degli Abruzzi, 24, 10129 Turin, Italy
Fax: + 39 (0)11 564 4134; email: lazarescu@polito.it

ing. Mario SARTORI
COREP – LETEO
C.so Duca degli Abruzzi, 24, 10129 Turin, Italy
Fax: + 39 (0)11 564 4134; email: sartori@polito.it

Summary

In this paper we will describe a modality to speed up the design of the VLSI digital (mainly DSP) circuits and to reduce the design errors by increasing the interaction between the ad-hoc software program developed to validate the algorithm and the VHDL description and simulation. A real case of a digital power analyzer will be used for exemplification.

1 Introduction

For the design of the digital VLSI circuits there are now available several high level synthesis tools that accept a more or less high level description of the circuit operation and/or structure and are able to simulate and synthesize the circuit.

The main drawback of the high level description simulators is that they are still very slow. As a large part of the today digital VLSI circuits are DSPs that implement an iterative algorithm, the design flow usually includes a first phase when a dedicated ad-hoc software program is developed to verify the algorithm corectness before even to think at the hardware implementation issues. After this phase, begins the design of the circuit itself and, in this phase, very few results of the previous simulations are used.

We will present in this paper a modality to increase the interaction between the algorithm validation ad-hoc software program development and the high level hardware description that follows. This way, the designer can introduce many hardware constraints directly in the ad-hoc program. Then, using this program, the designer may perform extensive fast-running tests simulating the algorithm much closer to the future hardware implementation than to the theoretical idealization. Moreover, the software program may be organized to have defined a function for each main block of the circuit, a practical way to later verify the simulation results of each implemented block with the results output by the corresponding function in the software program.

2 The problem

Often, the digital circuit designer is confronted with the following problem: having to design a circuit that implements a more or less known and/or validated algorithm. In such cases, the design task is usually divided in two distinct subtasks:

1. Simulation and validation of the algorithm. The designer develops an ad-hoc program in a given high level programming language (like C, Pascal, etc.) and uses it for validating the algorithm.

*Published with the agreement of DTA s.r.l., Milan, Italy

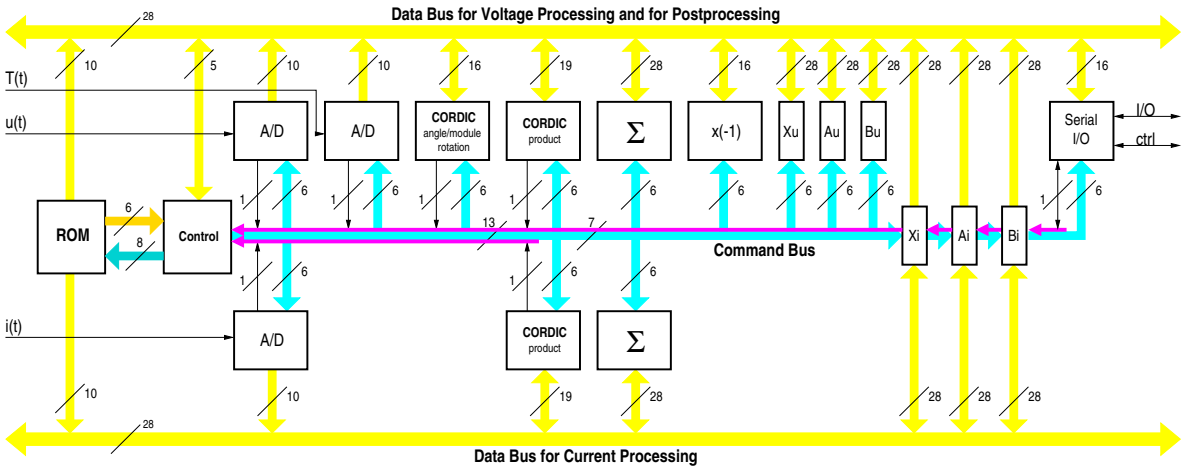

Figure 1: Block schematic

2. Simulation and validation of the hardware description that will implement the algorithm. The designer uses a high level description language (as VHDL) to describe, usually using high level functions, the blocks and the data flow needed to implement the algorithm. At this moment there had to be specified hardware related parameters as: bus dimensions, operands truncations, etc., aspects that were usually ignored in the previous algorithm validation phase.

From the above typical design flow we can see that, when passing to the hardware description that will generate the circuit after synthesis, the designer is once more confronted with algorithm issues like:

- how does the algorithm react to operands truncation? Will it be still convergent operating on integers and not on double precision floating point numbers?
- how large the busses should be?
- how bad will be affected the final results precision if shrinking that bus by one bit?
- given the results error specifications, what does that mean in terms of number of iterations of the algorithm or registers size?

Taking into account that the behavioral or logical simulations of the circuit done to validate the circuit structure in the second phase are very slow compared with the dedicated algorithm simulation program of the first phase, it results clearly that simulations in the second phase of this typical approach is very time consuming and, by the limited number of tests that can be performed in a given design time, it is error prone.

Using a practical example, we will describe a better approach, that takes into account the most part of the hardware limitations from the first design phase. This way, the design time is considerably reduced. The designer will have a good feeling of the hardware requirements starting with the earliest phases of the project. The results of the ad-hoc software program simulations and program blocks (functions) will constitute also a good reference for the hardware description phase, making circuit development and test straightforward.

3 Circuit structure

Our target is to design a digital circuit that performs power harmonic analysis based on the Fourier series decomposition.

The circuit block schematic is presented in figure 1. For the sake of simplicity we decided that a microprogrammed machine structure is the most suited. This way we keep the *Control* block simple and small. A part of the *ROM* is used for storing the microcode and is in tight interaction with the *Control* block.

The two input analog lines are synchronously sampled and converted to digital by the A/D converters and then processed in parallel to keep the clock frequency low and to avoid timing problems.

The digital blocks involved in the processing phase are: *CORDIC product*, $\sum$, *ROM*, *Control*, X_u, X_i, A_u, B_u, A_i, and B_i.

The blocks involved in the postprocessing phase are those connected to the upper bus: *CORDIC angle/module/rotation*, $\sum$, $\times(-1)$, X_u, X_i, A_u, B_u, A_i, and B_i, *CORDIC product*, *ROM*, and *Control*.

This circuit is the result of a project we do in the framework of SUMIS action for DTA s.r.l., an SME located in Milan, Italy. The circuit is designed using SYNOPSYS VHDL synthesis for the AMS 0.8μm digital CMOS technology.

4 The hardware emulation program

Given the considerations detailed in paragraph 2, it would be much more convenient to exploit the high execution speed of the algorithm validation ad-hoc software program in order to emulate as tight as possible the real hardware characteristics as: real bus size, integer operation, results overflow, operands truncation, etc.

We developed a C language program that emulates the following hardware particularities:

- to define the *size of registers, memories, and busses* we set to 1 in dedicated integer variables (called *masks*) a number of LSBs corresponding to the width to be defined;
- the *overflows* are simulated by macros that use the previously defined mask for each hardware element (bus, memory, register) and, using bitwise operations, they discard the bits that fall off the mask and do the sign prolongation needed by 2 complement signed numbers representation. These macros are used each time a new value is assigned to a variable representing a bus or a register;
- *global busses* are represented by global integer variables. Writing a global bus means setting the global bus variable to the contents of the variable representing the desired output register, and reading the global bus to a register means setting the variable representing that register to the global bus variable value;
- the *operands size truncation* is done by a bitwise right shift;
- each block in the block schematic has a corresponding function in the program that performs that block algorithm. That function can be as simple as an integer addition (for the $\sum$ block) or as complicated as an iterative algorithm as for the various *CORDIC* blocks;
- *ROM, RAM* memories are implemented as vectors of integers;
- the *Control* block is mainly a large *switch* statement that decodes the microcode fetched from the *ROM* and calls the function corresponding to the hardware block that is activated by that microcode;
- the *analog input patterns* are automatically generated based on the harmonics amplitude and phase values previously stored in a dedicated data structure and using the C library trigonometric functions.

  The amplitude and/or phase information for each harmonic can be changed automatically during a simulation, allowing to run different test patterns to check the operation of the algorithm, keeping into account the hardware precision limitations simulated by the program;
- the *A/D* converters perform just type conversions from floating point to integer for each input sample;

Using these programming definitions we succeed to emulate the main hardware related limitations that affect the performances of the algorithm.

The program allows changing all busses and registers dimension, as well as the number of iterations for CORDIC blocks, simply by changing the corresponding C definition. The program is very flexible, allowing automatic error calculation and on fly statistic determinations, as to memorize the largest error, the mean error, etc. for each of the output results when running several test patterns.

Moreover, we optimized the program for speed and we use it for running an accurate test with over 500,000 input analog waveforms that takes about 2 weeks to complete on a SUN SPARCstation 20. It is obvious that such an extensive test is far from affordable when using a behavioral or logic simulator for two basic reasons:

- those simulators are very, very slow compared to a dedicated and optimized computer program;
- preparing the input vectors in terms of A/D conversion results would have produced about $2.7 \cdot 10^8$ vectors, much too much to be read by most simulators;
- automatic error calculation and error statistics would not be possible.

Once the hardware emulation program was debugged and checked by passing all the extensive tests, there can be easily set up dedicated tests that activate one at a time only the function that emulate the operation of a single hardware block, and its output results can be used as reference to validate the VHDL description simulation results of that block. This way, we can avoid simulating the complete circuit operation at VHDL description level for a large number of input patterns. We can keep the complete circuit simulations number small, performing only those simulations for testing block interconnections and the *Control* block activity.

5 Conclusions

There are some limitations of the program emulation possibilities.

If the hardware uses parallel processing and the interactions between the parallel processes are such that cannot be done sequentially for simulation purpose, the software simulation may not be possible at all.

Also due to the parallelism, the most I/O interface protocols may not be simulable using a software emulation.

Object oriented languages (as C++) are more suited to express hardware related constraints and hardware blocks functionality.

References

[1] R.D. Harding, *Fourier series and transforms*, Bristol and Boston, Hilger, 1985

[2] R.E. Edwards, *Fourier series: a modern introduction*, vol. 1, 2, New York, Springer, 1979, 1982

[3] Burrus, T.W. Parks, *DFT – FFT and convolution algorithms: theory and implementation*, New York, Wiley, 1985

[4] Ed. by Bede Liu, *Digital filters and the fast Fourier transform*, Stroudsburg, Dowden - Hutchinson and Ross, 1975

[5] Letizia Lo Presti, *Valutazione numerica della Trasformata di Fourier DFT – FFT*, course notes, Politecnico di Torino, May 1988

[6] Brian W. Kernighan, Dennis M. Ritchie, *The C programming language*, 2nd ed., Englewood Cliffs, Prentice-Hall, 1988

[7] Jean-Michel Berge et al., *VHDL '92 (the new features of the VHDL hardware description language)*, Boston, Kluwer, 1993

[8] Bjarne Stroustrup, *The C++ programming language*, 2nd ed., Reading (Mass.), Addison-Wesley, 1991

[9] Douglas L. Perry, *VHDL*, New York, McGraw-Hill, 1994

[10] *IEEE standard VHDL language reference manual*, New York, IEEE, 1988

[11] Randolph E. Harr, Alec G. Stanculescu, *Applications of VHDL to circuit design*, Boston, Kluwer Academic Publ., 1991

[12] Douglas E. Ott, Thomas J. Wilderotter, *A designer's guide to VHDL synthesis*, Boston, Kluwer Academic, 1994