Synthesis of an Amber 23 Open-Core Processor

Leonardo R. Gobatto, Júlia D. Craide, Jonas F. Gava, Vitor V. Bandeira, Ricardo Augusto da Luz Reis
Universidade Federal do Rio Grande do Sul (UFRGS)
Institute of Informatics
Porto Alegre, RS, Brazil
lrgobatto@inf.ufrgs.br, jdcraide@inf.ufrgs.br, jonas.gava@inf.ufrgs.br, vvbandeira@inf.ufrgs.br, reis@inf.ufrgs.br

***Abstract*—The present work implemented the Logical Design and most of the Physical Design of the open code processor Amber 23, generating an integrated circuit in the GDSII stream format, which has a maximum frequency of 195MHz, a density of 6,896% and power consumption of 77,767mW.**

***Index Terms*—Amber 23, open-source, integrated circuits, logical design, physical design, OpenCores, Verilog, microprocessors, microelectronics**

I. Introduction

The objective of the present work is to implement the Logical Design and most of the Physical Design of the Amber 23 open-source processor, available on the OpenCores repository, using the Design Flow of Cadence. Beyond that, the design aimed to maximize the integrated circuit clock frequency and do the density, power, slack timing, connectivity, and geometry verification.

II. Amber 23

Amber [1] is a 32 bits RISC processor compatible with ARM® v2a ISA (Instruction Set Architecture) and supported by GNU Toolset. In the OpenCores platform the Amber Project is classified as: Design done, FPGA proven, Specification done, and OpenCores Certified, it also says it has a stable development status, is compatible with WishBone, and has an LGPL license. There are two versions in the Amber Project core, Amber 23 and Amber 25, both compatible between themselves and with the same ISA.

The main differences between the cores are that Amber 23 has a 3-stage pipeline (fetch, decode and execute), a single unified cache for instructions and data, 32-bit Wishbone interface, and the ability to reach 0.75 DMIPS per MHz. While Amber 25 gets about 15% better performance as it has a 5-stage pipeline (fetch, decode, execute, memory, and write-back), two separate caches for instructions and data, 128-bit Wishbone interface, and a capacity to reach 1.05 DMIPS per MHz. In this paper, the focus is on Amber 23.

Both cores are described in Verilog 2001, and are optimized for synthesis on FPGAs, being tested on the Xilinx SP605 Spartan-6 board, in which the Amber 23 occupies 32% of the LUTs (Look Up Tables) of the board and has a frequency of 40 MHz. A frequency of 80 MHz was obtained in the synthesis on a Xilinx Virtex-6 FPGA, but the core was not fully tested on the device. The Linux 2.4 kernel was booted to verify the circuit.

III. Methodology

As stated in [2] "A design flow is a set of procedures that allows designers to progress from a specification for a chip to the final chip implementation in an error-free way". Cadence's [3] standard cell design flow was used to implement the Amber 23 circuit. This flow is divided into 2 major stages: the Front-end and the Back-end. The Front-end is the initial part of the chip design and is divided into two sub-steps: the System Design and the Logical Design. The System Design is where the chip and microarchitecture specifications are defined, followed by the Logical Design, where the logic synthesis of the chip described in an RTL (Register Transfer Level) language is performed, resulting in a netlist. The Back-end is where the Physical Design stage takes place, where the placement and routing of the chip are made, from the netlist obtained in the logical synthesis, and finally, the layout for manufacturing is generated. The Amber 23 Project has already provided the entire System Design stage, so in this paper, the Logical Design and part of the Physical Design were carried out.

To perform the RTL simulations two software were used, a Cross-compiler from Source Forgery to generate the memory files and Cadence SimVision™ to execute the simulation itself. To perform the logical synthesis, Cadence's GENUS™ tool and the XFab cell library, which uses 180nm technology, were used. For the physical synthesis, the Cadence Innovus™ software was used.

IV. Logical Design

A. RTL Simulation

The first step in the design is the RTL simulation, it serves to verify if the Amber 23 circuit described at a logical level is in accordance with the expected operation, that is, equivalent to the specifications from the System Design. This code at the logical abstraction level is called RTL (Register Transfer Level) and is described using a Hardware Description Language (HDL). The most used RTL languages are Verilog and VHDL, which allow describing the design, operation, and organization of electronic circuits. In the case of this design, the specifications are that the system can execute the ARM® v2a instruction set, providing the testbench with the initialization of version 2.4 of the Linux kernel.

Amber 23 has a testbench with 65 programs [4] that test all parts of the circuit, examples: cache, branches, instructions,

data flow, among others. For the execution of this work, almost all 65 tests were performed, except for some that used or needed external modules to work. In each of these tests, there is a success subroutine, where the value 17 is written in register 10 if the test has passed 100%. So, it is possible to check in a practical way which tests are passing and which are not.

B. Design Constraints

During the logic pre-synthesis, the design constraints will be defined, which describe the clock characteristics of the circuit, such as: transitions, uncertainty, input and output delays, waveform transition, among others. For this purpose, the constraints.sdc file was created, in which the clock and other attributes, such as transition times and delay, were defined.

Next, was created the file with the script used for synthesis, the setup.tcl, in which the use of MOSST cells was configured, that is, Standard cells and not Low Power cells; and the variables SYN_EFF and MAP_EFF, which identify the effort used to optimize synthesis and mapping. During the simulations performed in this project, SYN_EFF and MAP_EFF were changed in the setup.tcl file, as well as the clock period in the constraints.sdc file, the latter being modified according to the desired frequency in the simulation.

C. Logical Synthesis

The logical synthesis step consists of parsing, translating, optimizing, and mapping an RTL code to a specific pattern in a cell library. The main objectives of this synthesis are: to minimize area, minimize power, maximize performance, quickly produce accurate and functional models and produce accurate and predictable results.

The inputs for this step are the RTL codes, design constraints, and cell libraries, that is, the Amber 23 files written in Verilog, the constraints.sdc file and the setup.tcl file. As output, the synthesis generates a Gate Level Netlist (GLD) description using an HDL.

One of the objectives of RTL synthesis in this design was to seek the maximum frequency at which the circuit could work correctly, that is, without errors. Another measure considered is the slack, which is the time between the desired arrival of a signal and the actual arrival of the signal, where we seek to avoid negative slacks, in which the signal would be delayed. Positive slack indicates that it is still possible to make improvements and slack zero is ideal, in which the system works at the desired frequency.

The first round of simulations was done with both SYN_EFF and MAP_EFF variables set to low optimization effort. The synthesis was performed several times in order to progressively increase the simulation frequency according to the table I. With this low-effort configuration, the maximum frequency obtained was 165 MHz.

For the purpose of comparison, the values of the variables SYN_EFF and MAP_EFF were changed for medium effort and the maximum frequency was recalculated, again performing successive simulations as it is possible to see in the table II obtaining the maximum frequency of 195 MHz. Finally,

Table I
LOGICAL SYNTHESIS LOW EFFORT

Frequência [MHz]	Slack [ps]	Potência [μW]	Células	Área total [μm²]
40	0	17.327,483	9.158	383.955
80	0	36.345,834	9.819	397.352
160	0	90.403,686	12.218	455.214
165	0	48.760,273	12.121	452.383
170	-84	94.109,382	12.555	463.036
180	-149	59.141,919	13.894	496.725

Table II
LOGICAL SYNTHESIS MEDIUM EFFORT

Frequência [MHz]	Slack [ps]	Potência [μW]	Células	Área total [μm²]
165	0	71.336,100	9.981	387.170
180	0	80.729,357	10.664	402.440
190	0	96.531,460	11.377	421.306
195	0	97.891,692	11.703	431.948
200	-10	95.552,228	11.777	434.458

simulations were performed with both variables with high effort, according to the table III. And the final maximum frequency was 195 MHz the same one found with medium effort, but the power was slightly lower. Therefore, the final clock obtained has a period of 5.1282 ns and high effort variables were used to perform the physical synthesis. In figure 1 you can see the result of the logical synthesis carried out with the best found parameters.

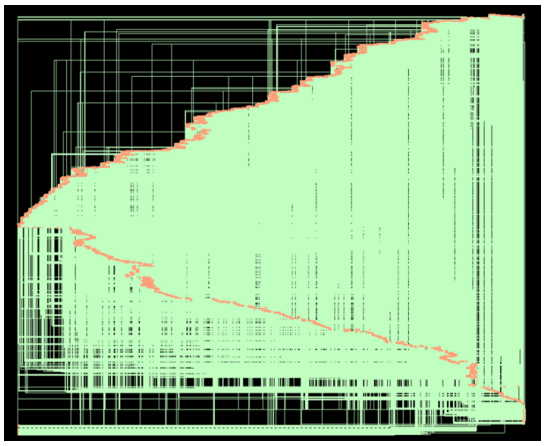

Figure 1. Logical Synthesis result

V. Physical Design

A. Comment about IO PADs

For the final synthesis in this paper, IO PADs were created, which are responsible for connecting internal signals of the integrated circuit with external pins of the chip. During the development of this design, physical syntheses were performed

Table III
LOGICAL SYNTHESIS HIGH EFFORT

Frequência [MHz]	Slack [ps]	Potência [μW]	Células	Área total [μm²]
190	0	95.149,321	11.338	423.682
195	0	92.287,847	11.421	423.008
200	-32	96.657,607	11.847	439.104

without the use of IO PADs, which obtained much higher densities of about 67.194%, as shown in the figure 2. So, the final circuit is limited by the placement of the PADs. To create the PADs, it was: created an iopads module in Verilog; created a top module in Verilog to encompass Amber 23 and the iopads module created; insertion of modules created in Verilog resulting from Logic Synthesis; and generated the IO Pads file, iopads.io, to position the PADS in Innovus.

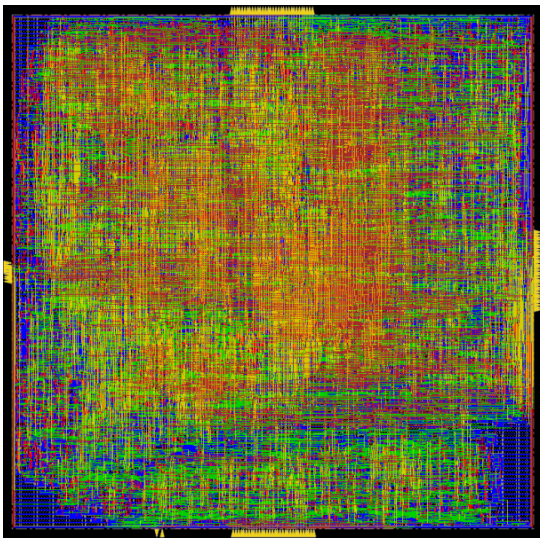

Figure 2. Phisical Syntesis without IO PADs

B. Floorplanning

Floorplanning is the process of allocating space for modules and circuit functional blocks across the chip in order to improve communication between modules and make it possible to derive the final die size. It was defined as a floorplanning of square format, which seeks to obtain 70% of density with margins of 3 μm.

C. Power Planning

In the Power Planning stage, a power distribution network is created, which provides voltage and ground for all modules of the design. It is a very important step, as problems related to power can even lead to failures due to problems with electromigration effects. In this step, horizontal lines of GND and VDD were created in the circuit design.

D. Placement

Placement is the process of placement the standard cells on the floorplan already defined, intending to minimize the length of the wires. An important aspect of the cell library, to allow automatic placement, is that the cells must have a constant height among each other, which can result in a larger area and performance loss compared to full-custom circuits.

The interface of the system was used to carry out the full placement of the design and obtain the placement of the cells. The report on the use of metal layers for placement show some errors and warnings, mainly due to scan chains, which are flip-flop connection patterns, but these will be resolved in future steps.

After placement, a pre-CTS optimization step was carried out, the main problems that can be solved in this step are: increasing and decreasing the size of the cells; adding buffers on networks, and resynthesize ways to improve timing. In the results obtained after the optimization it was possible to notice that there is still a path violation and a density of 6.704%.

E. CTS

In this step, the clock tree (CTS - Clock Tree Synthesis) is synthesized, which synchronizes the clock throughout the chip, minimizing clock skew and latency. For this, buffers and inverters are inserted in the clock paths. The operation to generate the tree results in the time analysis that show the minimum or maximum required for each parameter where, for example, the clock skew is 128.4 ps which is below the required 205.1 ps.

Next, a post-CTS optimization was performed, whose main objective is to optimize routes that may have been disturbed by the CTS and to optimize the clock tree itself. Observing the results, it is possible to see that the clock slack is zero, the density has improved a little, reaching 6.873%, and there are no more path violations.

F. Routing

Routing is an important step in the circuit generation, as it is responsible for connecting the cells, macros, RAMs, and inputs and outputs pins, in the routes specified by the netlist, in order to try to minimize congestion and critical paths. Routing is usually divided into two steps, global and detailed routing. The global routing abstracts the routing problem by breaking the chip area into rectangular portions and tries to minimize the connection path and more congested cells of wires in the same area. The detail route takes into account the real geometry and design rules, as its objective is to make all connections without any violation, trying to follow the plan traced by the global route.

TCL code was used to perform all the routing steps, the first command performed the global route. The following command executed the route detail, finally, the last command performed the global route detail, which gave the wire and metal usage resulting in a total wire length of 820620 um.

After doing the Routing, a Post-Route optimization is performed, which is more accurate than the previous optimizations steps, because it has real wires and not only estimated ones, being able to make more advanced and aggressive changes. This resulted in no path violations, a small improvement in the density, that was 6.896%, and the timing results, where again slack is zero.

G. Fillers

One of the last steps of the Physical Design was to add filler cells, also called dummy cells, in the empty areas of the circuit. They are never activated by the circuit, their only purpose is to increase the uniformity of the circuit layers and thus facilitate the fabrication process.

Once again, TCL commands were used to generate the fillers, the result was the insertion of a total of 105620 filler cells, in which the cells named FEED are the dummys of the used library. There were no path violations and the density remained at 6.896%, however it is interesting to note that the density with filler cells is 100%. In this step, a geometry check was also carried out, which did not show any errors.

VI. Results and analysis

After completing all the steps, several checks were carried out to guarantee the quality and obtain some parameters of the circuit. The first analysis was the power analysis, the result of this Power Report was a total energy of 77,767mW, which represents a significantly lower consumption than the 92,287mW that had been estimated in the Logical Synthesis, and the leakage power was almost zero. After that, parasitic resistances were extracted and delays were calculated with no errors. Using Innovus' interface, it was possible to verify connectivity and circuit geometry, respectively. The results were a success, with zero violations and warnings.

As a result of the entire process, after all the steps and verifications, the circuit in the figure 3 was obtained.

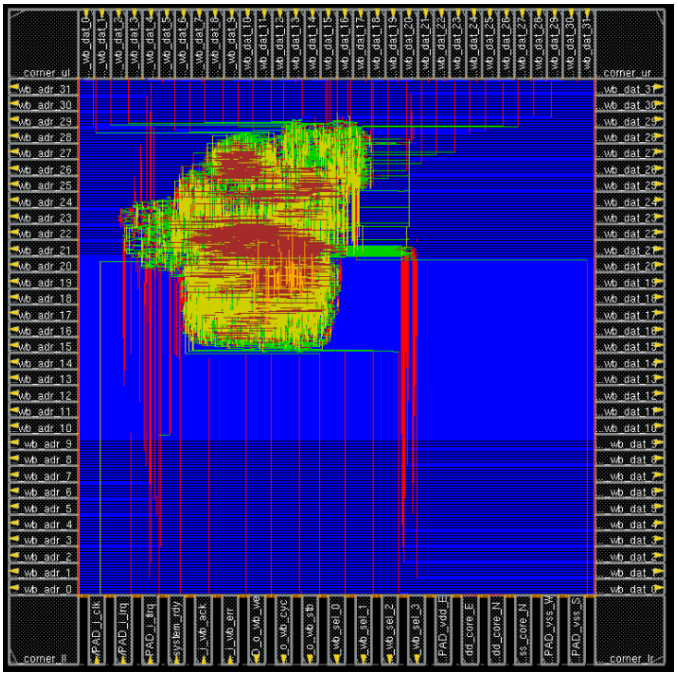

Figure 3. Final circuit

VII. Problems

During the RTL simulation stage of the project, several problems were encountered, the first of which was the configuration of the file_list.f file, which is used to quote and sort Verilog files, to fix dependencies between them, as Amber is a big project it became complex. Another problem is that some files contained an input called "do", which had to be renamed as this is a Verilog reserved word.

Finally, even in the RTL simulation, it was not possible to install the GNU cross-compiler from Code Sourcery, necessary to compile the memory, from the official website due to regulatory problems, making it necessary to use the Internet Archive Wayback Machine, an archiving mechanism for old sites, to get a valid 2011 link that allowed the download.

During the Physical Design, there were problems in the first attempt to make the Power Plan, before the simulation with IO PADs, because both horizontal and vertical power lines were created. However, as the Amber 23's circuit is large, there were violations due to overlap, and this problem was solved by removing the vertical feed lines.

Finally, as an attempt to improve the circuit density, other rectangular layouts were tried for the chip, but these resulted in several overlap violations without much improvement in density. One of these attempts is illustrated in the figure 4 where it is possible to see several overlap errors in the bottom left corner.

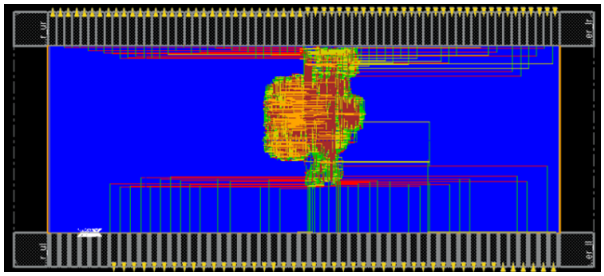

Figure 4. Rectangular circuit with overlap issues

VIII. Final Considerations

In this paper, we discussed the implementation of the Amber 23 Logical and Physical Design, generating an integrated circuit with a frequency of 195 MHz, with total energy consumption of 77,767mW and density of 6.896%. In spite of the progresses achieved, much work remains. One direction for further research is to test different design flows to understand how they impact the occupation and power of the present circuit.

IX. Acknowledgments

Thanks to Cadence Design Systems Inc. for providing the necessary software and excellent support material, Conor Santifort for developing and maintaining the Amber Project, and the OpenCores platform for making available and hosting such a library of hardware projects, without which this work would not be possible.

References

[1] Santifort, Conor. "Amber 2 Core Specification", Amber Open Source Project.OpenCores, 2015.

[2] N. Weste e D. Harris. "CMOS VLSI Design, A Circuits and Systems Pesrpective." Pearson/Addison-Wesley, 2011. ISBN 9780321547743.

[3] "BD03: Digital Physical Design", Version 1.0. IC Brazil Program; Cadence Design Systems Inc, 1990-2008.

[4] Santifort, Conor. "User Guide", Amber Open Source Project. OpenCores, 2013.