Teaching High-Level Design

A. Maeder, N. Hendrich, K. Lagemann

University of Hamburg, Dept. of Computer Science
Troplowitzstr. 7, 22529 Hamburg, Germany
e-mail: <name>@rz.informatik.uni-hamburg.d400.de

Abstract

In this paper we argue that the trend towards *system design* requires to teach *high-level design*, in both lectures and design projects. High-level synthesis and hardware-software codesign are two key words.

The main part of our paper presents our experiences with a design project where we used the MIMOLA software system for high-level synthesis of a floating-point pocket calculator chip. We describe our tool-chain, MIMOLA, MIM2SOLO and SOLO 1400, discuss some problems, and present the designed chips.

1 Introduction

Changing the appearance of "classical" hardware design, the following trends have to be mentioned:
New markets demand for new qualities within the systems. Such markets are: consumer electronics, telecommunications, multimedia ...
Technological development has to be seen on system level, the trends are: larger systems, more components per system, more components per chip — but less chips per system. Looking at the circuits, it can be foreseen, that in the future the limits of silicon will end the previous trend towards smaller and faster structures. Possibly other technologies will evolve...?
Compiler oriented approaches will be the key to handle the task of *system design*. Initial descriptions of algorithms and the architecture result in specifications for the hardware (the input to silicon compilation) and the software — hardware-software codesign, heterogenous designs. To do so, *system engineers* have to know these hardware/software codes, while the actual hardware design will be done using silicon compilation.

2 Teaching High-Level Design

At our institute we teach the subject "high-level design" in one lecture of the three semester cycle *design of integrated circuits*. Concerning high-level synthesis this course contains the following items:

Motivation
- trends: new markets with new demands, higher initial description towards system level
- top-down design flow (interactive, iterative), bottom-up influences
- hierarchical refinement: elements of a structural description (design) at one level are behavioural descriptions (specifications) for the lower one

Advantages
- work is done by system designers, hardware-software codesign
- behavioural descriptions: shorter specifications, handling complexity
- search space exploration through alternatives, redesigns
- faster design cycles (time to market), increased productivity per designer
- technology independence, portability

Concepts & Algorithms
- specification: behaviour, i/o, constraints (processing speed, space, power consumption, testability, serviceability...)

– separated parts: computing data path and sequencing control (FSM, microcoded, collection of FSMs, handshake)
– system types: data flow dominated, control flow dominated
– DSP requirements: execution time fixed by data throughput (specification)
– system : architecture synthesis, system partitioning, pipelining
  rt : code compilation (data flow/control flow graph, dependence graphs), high-level transformations (dead code elimination, constant propagation, common subexpression elimination...), loop interpretation (splitting, folding, merging), scheduling (ASAP, ALAP, list, force-directed, path based, percolation like...), allocation (for functional units, memory and connections – resource binding or module selection), partitioning (timing/area estimation, testability), retiming (minimal cycle/area), design for testability
  logic : resynthesys, state minimization (vertical/horizontal merging), state encoding (one hot, cube collapsing, factorization, path encoding...), technology mapping
– languages and description aids: HDLs (VHDL, Verilog), flow graphs, state charts, BBDs

Problems
– verification for most steps not feasible, done through simulation
– design quality, overhead through synthesis?
– search space force tools to be specialized within the domains: algorithms, architectures, applications ⇒ different design paradigms
– problems of (complete) specification, one unified language? (also for intermediate results)
– DA tools: available, usable, payable, fitting together to tool-chain

3 The calculator design project

Reflecting the necessary evolution towards higher design levels in lab sessions, we offered a design project using high-level synthesis tools during the previous semester. While high-level synthesis had already been used within BSc and MSc theses, this was our first regular course centered on high-level design. The goals for the design project were as follows:

- Previous courses had already proven a kind of "standard" top-down design strategy: Starting from a general (natural language) specification, an algorithmic solution was constructed, using standard programming languages like C or Pascal. Then a first hardware structure on RT level was designed by hand in an "ad hoc" fashion. After the programs were rewritten in a "hardware-like" fashion to simulate this structure. Finally the design has been implemented using a standard-cell design system.

  Therefore the project should establish a *consistent* top-down design flow, using a high-level synthesis system as the front-end of the tool chain. This would eliminate the manual "ad hoc" refinements and allow to explore a large number of design alternatives.
- The main focus should be on the algorithm design, but keeping in mind possible bottom-up effects. The students should get a feeling for how algorithmic decisions influence the lower levels in the design hierarchy and how to take restrictions of the standard-cell system into account during algorithm design.
- Since high-level synthesis enables the designer to explore the design space rapidly, the chosen design-example should allow the parallel development of several alternative solutions for groups of students.

3.1 The Tool-Chain

For this project we choose MIMOLA [Ma 85], MIM2SOLO [HeLoRa 92a] and SOLO 1400 [ES2 91] as the appropriate tools.

The MIMOLA software system is intended for the synthesis of control-dominated, microprogrammed processor structures. Strong points of MIMOLA are the support of multiport memories for the high-level synthesis steps, its retargetable microcode compiler, an integrated interactive simulator, and the automatic test-program generator.

Additionally, MIMOLA is especially suited for teaching purposes, since the input language MIMOLA is based on standard Pascal. Therefore the students are able to program and test their algorithms at home, using a Pascal compiler environment. After adding some MIMOLA specific constructs (namely, the module library), the Pascal algorithms are ready for high-level synthesis.

Our MIM2SOLO tools then offer the necessary link to a standard-cell design system (SOLO 1400 or CADENCE/SOLO 2030) with full support for RAM and ROM macro-cells.

For the project, we chose the SOLO 1400 standard-cell design system as the backend, because it is very easy to use, allows the textual input for the RT-netlist, and we had already gathered a lot of experience with it during previous courses. Other VLSI design systems may offer better layout quality, but often require much more experience from the designer — i. e. the students. Unfortunately, we encountered several annoying problems with the SOLO 1400 system during the project.

The ideal MIM2SOLO design-flow, starting from an algorithmic specification, is sketched in figure 1. The algorithm can be simulated with the simulator MSSS in MIMOLA or executed as a "pure" Pascal program. Then the high-level synthesis steps take place, controlled by option switches, e. g. to define cost weights.

The whole synthesis process is very fast — a complete run for our design `bcdcalc`, from the design analysis up to the microcode generation, took less than 5 minutes CPU-time on a SparcStation 10/30. Using the simulator MSSU, the generated structure can be checked.

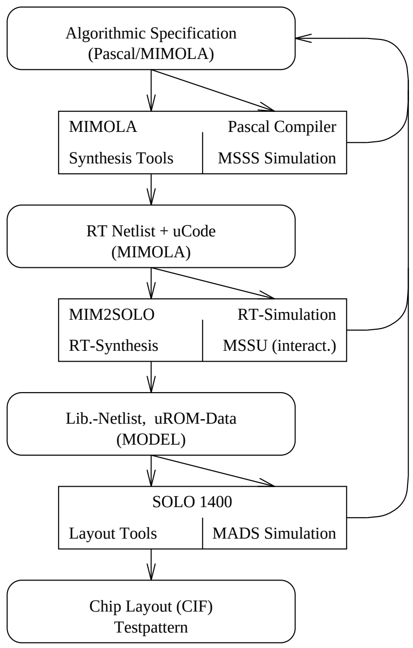

Figure 1 : MIM2SOLO Design-Flow

Then MIM2SOLO generates the gate-netlist from the MIMOLA RT-level structure, produces the μROM personalization file for the ROM generator and translates the simulation stimuli. The physical layout and the final (gate-level) simulation is done with SOLO 1400.

3.2 The design example: a pocket calculator

We choose a pocket calculator as the design example for this project, because:

- The problem is easy (intuitive) to understand. A "vague" verbal description lets freedom for design alternatives, giving a first impression on how a specification might look like.
- A lot of different algorithms and architectures are possible (serial/parallel, binary-/BCD-coded ...).
- Since the task is not limited to a certain functionality, functions could be added or removed to finish in time — depending on the progress of the students.
- No special performance requirements have to be fulfilled.
- Since the design should also contain appropriate interfaces to its environment (e. g. display control, key matrix input) the final IC works with a minimum of external circuitry.
- A pocket calculator can be realized as a control-flow dominated processor with a (possibly) complex microprogram. Therefore it is well suited for synthesis with MIMOLA.

To prove the feasibility, we implemented a "minimal" calculator using BCD-arithmetic for serial addition and multiplication of 8-digit integers. Though some errors were found in this design during the course, it was a first example to give the students at hand.

3.3 Organization of the course

Nine students participated at this course, working in groups at SUN and APOLLO workstations. Due to the length of one semester the course was limited to 14 sessions of 4 hours each. During the first two sessions we introduced the design example, illustrated the design flow with a short presentation of the tools, and discussed possibilities and alternatives for the algorithm design. Afterwards the actual design phase started. Due to the high motivation of the students, two designs out of the variety of initial architectures were actually completed; one of these circuits is submitted for fabricaton.

As a first documentation for the students we wrote short "recipe"-like manuals, describing the capabilities of the tools and the way to use them. Design examples — especially for MIMOLA — and some ideas on the given task completed the training material.

3.4 bincalc

The first design developed during the project is based on a combination of binary and BCD arithmetic. Though we proposed a fixed point calculator, the students wanted to use a floating point representation from the beginning.

The design `bincalc` uses a floating point number representation with a 32 bit mantissa (normalized, implicit leading one, sign bit) and 10 bit exponent with an offset of 512. It implements serial binary arithmetic for addition, subtraction, multiplication and division with algebraic infix notation. All logic for multiplexed seven-segment display control and multiplexed keyboard input is integrated on-chip.

Since the internal computations use binary arithmetic, conversions from BCD to binary and vice versa are necessary. While the input conversion is easy, the binary to BCD transformation is more difficult and two additional BCD-registers and an extra BCD-ALU are used. The bit-true simulations show that results are limited to 8 decimal digits of precision, due to accumulation of truncation errors in the conversion.

The initial Pascal code for `bincalc` had about 1 600 lines of code. Since this was the first time the students wrote code for a high-level synthesis tool, a lot of "good Pascal" style constructs led to poor synthesized results.

After several modifications of the algorithms to improve the synthesis results, we started the physical design, resulting in the chip layout shown in figure 2. The automatically SOLO 1400-generated layout for `bincalc` has an area of $54\,\mathrm{mm}^2$. It contains four macrocells, using about 50 % of the chip area: a microprogramm ROM of 1024×80 bit and RAMs for the operating storage 256×4 bit ($\times 1$ was not possible), for the BCD registers 32×4 bit and the procedure stack 8×10 bit.

3.5 bcdcalc

The second design completed during the course, `bcdcalc`, uses BCD arithmetic for all internal operations, thereby avoiding data conversions. `bcdcalc` is much more efficient than `bincalc` and is submitted for fabrication through EUROCHIP. Though the students already worked on algorithms for higher functions (square root, exponentiation, trigonometrical functions etc.), time was too short to implement them:

- serial BCD arithmetic
- floating point number representation, BCD-coded: 10 digits mantissa with sign bit, 2 digits exponent with sign bit ($\pm 9.999\,999\,999^{\pm 99}$)
- operations: addition, subtraction, multiplication and division
- one memory register with additional operations: `M+` and `M*`
- algebraic input in infix notation
- operator priority and bracketed expressions (10 levels)
- error handling routines: overflow, gradual underflow ...
- multiplexed seven-segment display output, multiplexed keyboard input

Additionally, several measures have been implemented to increase the testability of the circuit:

- memory self-test after reset
- testmode for input and output, by-passing time loops: direct parallel input, simplified output
- scanpath
- multiplexed output of the microprogram ROMs

The first two are defined in the `MIMOLA` code, since a behavioural description was natural, causing only minor overhead after the synthesis steps. However, the latter ones were added by hand to the `MODEL` netlist for SOLO 1400; this approach was necessary because a first algorithmic description in `MIMOLA` led to poor synthesized results.

The `MIMOLA` code for `bcdcalc` has nearly 1 900 lines of code — 650 for the definition of the module library and about 1250 for the algorithms. Since `bcdcalc` was begun after `bincalc`, most of the problems which arose for `bincalc` could be avoided. One major advantage of `bcdcalc` is its memory management scheme: the operands and the operation between two adjacent registers are stored in a register file of 16 registers (256×4 bit RAM), which is addressed through three global variables (4 bit pointer registers). Push and pop operations, necessary for operator priority and bracket handling, are done by simply incrementing or decrementing these variables.

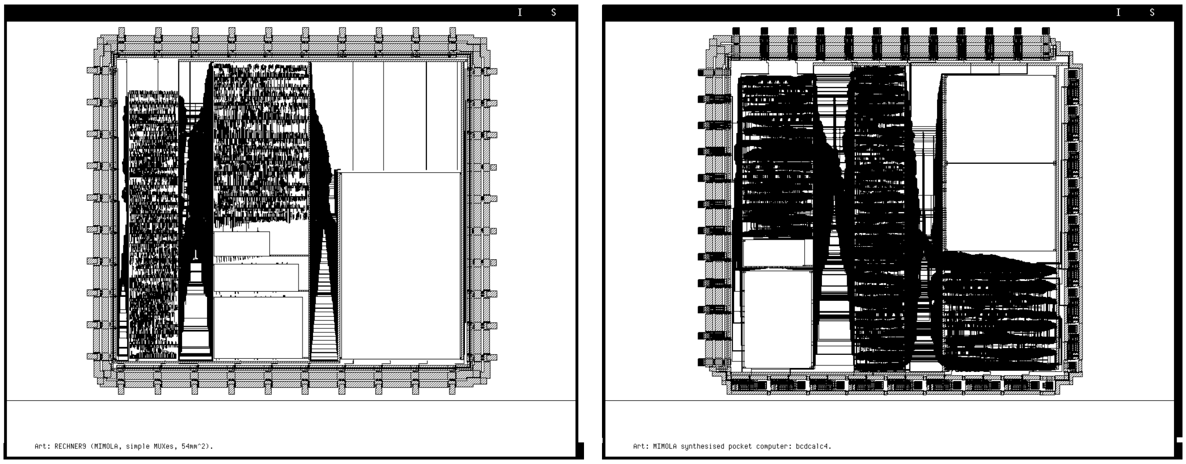

Figure 2: left first, automatically generated layout for `bincalc` (bad macro-cell placement)
right SOLO 1400 layout for `bcdcalc`

The final SOLO 1400-layout of `bcdcalc` with a total chip area of $43.96\,\mathrm{mm}^2$ and a core area of $29.95\,\mathrm{mm}^2$ using the *ecpd15* process is shown in figure 2. The circuit contains 2 144 standard cells and 4 generated macro cells. In the left lower corner the two RAMs are placed: 8×10 bit for the procedure stack and 256×4 bit for the operand memory. Two ROMs of 652×19 bit, placed in the upper right corner, store the microprogram. The design has 42 pads: 12 input, 24 output and 6 power pads. Since the design is core limited, two sides use "flat" Padlib3 cells. According to maximum delay simulations a clock frequency of over 5 MHz can be achieved, limited by memory access times.

4 Experiences with the design project

In the following we will highlight some of our experiences with the design project. A full list of the problems we encountered, together with a detailed description of the designs `bincalc` and `bcdcalc` may be found in [HeMä 93].

Design example & design methodology

The chosen design task worked out well, fulfilling our expectations on algorithm design and design alternatives. The students were able to design in a top-down manner using synthesis and automatic translations — actually the "only" manual intervention was necessary for the SOLO 1400 netlist, because of

- testability aspects (scan-path, multiplexed ROM output)
- optimized implementations of multi-function operators, where the initial MIM2SOLO realizations seemed inefficient
- bugs in SOLO 1400!

MIMOLA Because the MIMOLA software is menu-driven and controlled by options with useful defaults, and because all our students knew Pascal, they had no problems to write MIMOLA specifications and to use the tools.

However, the students had to learn how the expression of *behaviour* influences the resulting synthesized *structures*. For example, the use of complex data types (nested Pascal `records` and `arrays`) in the beautiful initial specification led to an overhead after synthesis, due to the extra adders required for address calculation.

Also, not all students were aware that the use of procedure calls implies a stack to store the call hierarchy and local variables. The designer has to know the nesting depth, recursive algorithms may present problems.

Since arithmetic operators in MIMOLA assume unsigned or 2-complement number representations, the BCD-ALU in `bincalc` was difficult to describe. To have a consistent simulation model, we finally had to code the ALU as a PLA.

SOLO 1400 As explained above, we chose SOLO 1400 as the backend tool, because it is so easy to use. We wanted to spend more time on the algorithmic level and as little time as possible with the physical design. Final layout quality was not a paramount concern for the design project.

Several bugs are known in the SOLO 1400 (3.1.3) software, which require workarounds. For example, deeply nested MODEL constructs may fail, busses more than 32 bit wide (can) cause problems and generated blocks with more than 32 bit wordlength show wrong simulation behaviour, though the simulation models are correct.

Therefore we wrote a simple program to split wide busses into smaller ones. Additionally, a tool `minrom` [Mä 93] was used to reduce the wordlength of the horizontal microprograms by generation of control lines by logic expressions, group coding ... For `bcdcalc` a reduction from 76 to 2×19 bit has been achieved.

However, the "simple" layout algorithms used in SOLO 1400, together with the lack of interactive control for P & R, caused a lot of problems. The following cycle appeared: after placement & routing nets were too long or had to be splitted to accord to the fanout check — the necessary drivers were included in the netlist (`MODEL` file) — a "new" layout is generated — caused by the changed layout, other nets violated the layout rules ...

Compared with previous (larger) designs, the layout algorithms behaved even worse on MIMOLA synthesized circuits — probably due the (excessive) sharing of hardware resources and control bits, leading to very long and branched lines.

5 Summary

With the state-of-the-art tools SYNOPSYS and CADENCE OPUS available through EUROCHIP now, we can offer the industrial standard environment too. Future design projects, first planned for this academic year, will profit of a continuous path from VHDL specification and simulation down to excellent layout tools.

Encouraged by the experiences during this project, we will establish regular design projects and exercises for beginner lab-sessions dealing with: High-Level Design.

References

[ES2 91] European Silicon Structures Ltd., *SOLO 1400 Release 3.1 Reference Manual*, Berkshire, 1991

[HeLoRa 92a] N. Hendrich, J. Lohse & R. Rauscher, *mim2solo: Automatischer Entwurf mikroprogrammierter VLSI Schaltungen mit MIMOLA und SOLO 1400*, Report 201/92, Dept. of Computer Science, University of Hamburg, 1992

[HeLoRa 92b] N. Hendrich, J. Lohse & R. Rauscher, *Silicon Compilation and Rapid Prototyping of Microprogrammed VLSI-Circuits with MIMOLA and SOLO 1400*, Proc. EUROMICRO-92, 287–294, 1992

[HeMä 93] N. Hendrich & A. Mäder, *High-Level Design in der Lehre: Entwurf eines Taschenrechner-ICs*, Report FBI-M.../93, Dept. of Computer Science, University of Hamburg, 1993

[JöMa 89] R. Jöhnk & P. Marwedel, *Mimola Reference Manual V.3.45*, Internal Report 8902, Christian-Albrechts-University Kiel, 1989

[KuMi 90] D. Ku & G. De Micheli, *High Level Synthesis and Optimization Strategies in Hercules and Hebe*, Proc. EuroASIC 1990, 124–129, 1990

[Mä 93] A. Mäder, *minrom: Ein effizienter Algorithmus zur Kompaktierung von horizontalen Mikroprogrammen*, Report FBI-M.../93, Dept. of Computer Science, in preparation, University of Hamburg, 1993

[Ma 85] P. Marwedel, *Ein Software-System zur Synthese von Rechnerstrukturen und zur Erzeugung von Mikrocode*, Postdoctoral Thesis, Christian-Albrechts-University Kiel, 1985