North-Holland
Microprocessing and Microprogramming 18 (1986) 341–346

VLSI top-down design based on the separation of hierarchies.

L.Spaanenburg, A.Broekema, J.Leenstra and C.Huys

Twente University of Technology
Dept. of Electronic Engineering
P.O.Box 217
7500 AE Enschede (The Netherlands)

Abstract. Despite the presence of structure, interactions between the three views on VLSI design still lead to lengthy iterations. By separating the hierarchies for the respective views, the interactions are reduced. This separated hierarchy allows top-down design with functional abstractions as exemplified by an experimental self-timed CMOS RISC computer design.

1. Introduction.

In the years past, IC design has grown away from a mimicry of printed-circuit board design. In the PCB technology tens of packaged ICs are mounted on an isolating substrate and electrically connected by means of patterns on one or more conducting layers. The equivalent in the silicon technology is the so-called "standard cell" concept. Functional cells with a fixed perimeter are placed on one silicon substrate and subsequently connected. With computer-assistance, the placement of cells is based on wiring area restrictions, not on functional information. As a consequence structural information gets lost and timing conflicts can creep up.

Structured design is called for. But the simple "division of tasks" is not enough. A complex problem can best be solved by splitting it into independent parts, solving the parts individually and finally put the parts together. In PCB design the electronic parts are separately packaged and thereby become almost independent. In VLSI the cell notion is artificial and does not provide by nature independence; hence more structure in the design process is needed.

First the VLSI design space is discussed in terms of views and abstractions. Abstractions lead to timing requirements, while views are interpreted to require a "separated hierarchy"[1] which can be shuffled to optimise an example design in a top-down fashion. Then design elements are discussed to accommodate the need for multi-view self-contained entities. Lastly the concepts are illustrated by the design of a self-timed CMOS microprocessor with RISC architecture.

2. Views and abstractions.

Basically there are three views on a design: behaviour (what to do), structure (how to do) and geometry (where to do). Each of these views can be represented by an external and/or internal specification. For instance an external behaviour specification describes the function by a black-box input/output logic relation such as a truth-table, while on the other hand the internal behaviour specification implies the input/output relations by means of a set of Boolean equations. In turn the external view can be further detailed into perimeter and terminal information, while the internal view is composed of submodels, wires and their mutual connections. Of these representation and/or existence must be noted.

Views are possible on several levels of abstraction[2]. On basis of packaged elements one can discern the abstractions circuit, logic and function. This also reflects the historical evolution of integration. In the silicon technology the following abstractions are more appropriate. The geometry view is based on rectangles, which can be combined to create transistors or even logic gates.

			EXISTENCE	REPRESENTATION
STRUCTURE	INTERNAL	SUBMODEL	COMPOSITION	SYMBOL
		WIRES	NETLIST	LINE
		CONNODE	-	DIAGRAM
	EXTERNAL	PERIMETER	PACKAGE	ICON
		TERMINAL	-	PIN
GEOMETRY	INTERNAL	SUBMODEL	CALL	SYMBOL
		WIRE	ROUTED	MASK LAYER
		CONNODE	-	MASK LAYOUT
	EXTERNAL	PERIMETER	CALLABLE	BOUNDING BOX
		TERMINAL	-	COORDINATE

Figure 1. Meaning of some design model variables.

In standard cell design the cell is introduced, which allows to handle a collection of transistors as a single entity. For more complex designs, the floorplan notes the geometric information as a collection of generalised blocks.
A similar built-up can be seen in the structural view. On the lowest level are the circuit and logic diagrams. A higher level of abstraction gives the modeling by means of registers and the datapaths in between. In an even more abstract way one can state the structure through processes and their communication. Also the behavioural view can thus be layered. Going from top to bottom, one encounters first the system, then the algorithm and finite state machine and finally the Boolean expression.

There are various ways to travel this design space. In principal all start with a behavioural or structural specification on an arbitrary level of abstraction. First of all the bottom-up approach details the behaviour or structure first and then collects the geometrical information from the basic rectangle up. It allows a very efficient local mask layout, but could lead to a very poor global efficiency. On the other hand the top-down approach proceeds from behaviour via structure to geometry in a circular fashion, only gradually creating more detail. This allows a very good global control, but lacks insight into the local problems[3] . To combine the basic benefits of the above two, the "design by example" approach will be introduced here.

3. Design by example through a separated hierarchy.

The introduction of abstraction into the "division of tasks" approach to structured design leads in a natural way to the concept of hierarchy. The design space is imaged by a tree, of which the root gives the system specification. The system is composed of "composite cells" and ultimately of "leaf cells". Composite cells are composed of other cells, leaf cells are not. The "separated hierarchy" restricts the content of composite cells to only calls to other cells.

The hierarchy is at least a layered behavioural description. It can be interpreted for structural and geometrical information. In building the layout, the hierarchy can be traveled upwards, placing cells that are called in one composite cell in ones vicinity, if not next to one another. Where composite cells contain only calls, this order of combining cells can be changed by interchanging the content of composite cells. In other words an exemplary design can be entered as a separated hierarchy. Then the composite cells are shuffled to reflect best a proposed floorplan. Floorplan information is supplied in terms of quality factors. This can be Rent's rule or more complicated schemes, such as the "contact defect ratio"[4] . This tends to create orthogonal designs, that are well fitted to abutment, leaving larger complexes to be routed. This turns the separated hierarchy into a "restricted" one. Herein the notion of "module cell" has been added to reflect the difference between abutted and routed layout entities[5] .

In shuffling a separated hierarchy, a distinction must be made between computation and communication[6] . Cells may appear that have no contribution to the computation being performed. They rather relabel signals; the multiplexer is an example of such a cell. In shuffling communication cells may come and go when needed to merge computation cells or to split them. This is not in conflict with the notion of the separated hierarchy, as these cells are not really part of the initial behavioural description.

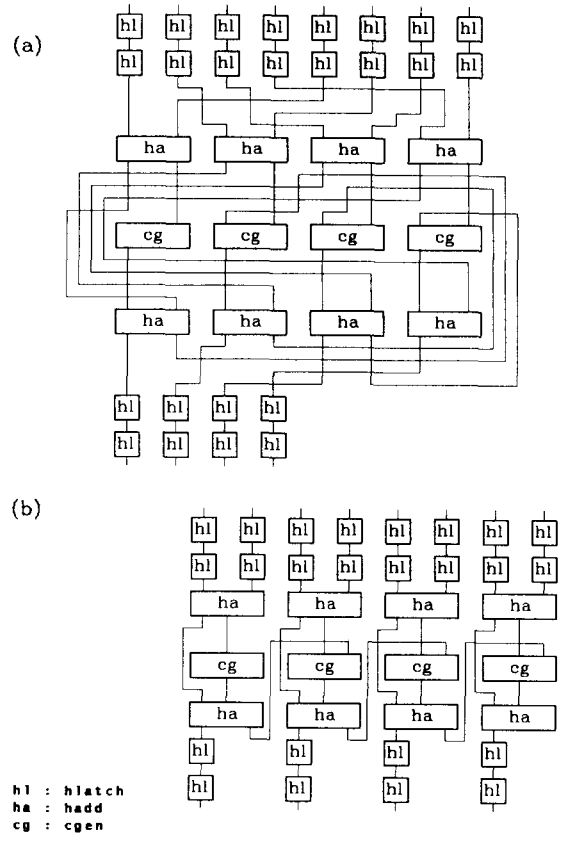

Figure 2. Topological layout of an ALU before (a) and after (b) shuffling.

An example is given in fig.2. The behaviour specifies an elementary 4-bit ALU. Originally the hierarchy showed two input registers, one accumulator and one output register. These in turn were based on latches, halfadders and carry generators. Shuffling changed the tree into a bitwise organisation, which was essentially easier to route and consumed accordingly less area.

4. Functional levels of abstraction.

The moment cells are placed in the geometry domain, the wiring delays become known. Large delays may lead to bad system timing. With the increase in chip size and decrease in minimum chip detail, the size of an isochronous region (i.e.

the region wherein wiring delays can be neglected in comparison to gate delays) becomes smaller than the chip and wires may cross the boundary. A typical example is shown in fig. 3. If register A is clocked to output data to register B, but the clockline suffers too much delay to reach register B in time, data will never arrive.

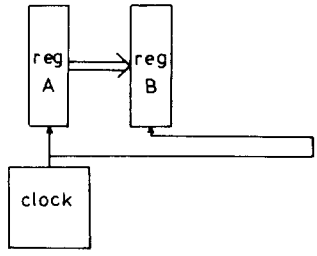

Figure 3. Timing problem caused by clock skew.

By even hierarchically splitting the timing signals, the problem can be eased but the use of a more explicit timing protocol is advantageous[7]. Here a function [8] which is large enough to suffer from the above problem is assumed to consist of

* a communication part, that will monitor timing, and
* a control part, that will start and end the transfer of data on
* a data part.

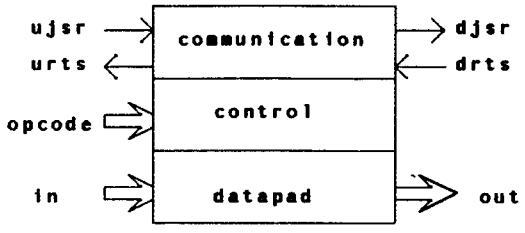

Figure 4. Architecture of the function cell.

The communication part consists of a receiver and a sender. The receiver uses a 2-wire handshake for a REQ/ACK protocol and signals the control part with a second 2-wire handshake. The sender has a similar content.
The control part consists of a small sequencer and an instruction decoder. The instruction will select the datapath, which consists of self-timed logic. After the datapart has finished his operation, it will signal the controlpart to end the operation.

Inside these parts, synchronous logic may still be applicable. But externally the communication parts will give the function block an asynchronous outlook by employing either a handshake or a dataflow protocol[9]. This concept will allow functions to be placed according to the required speed of communication, if the cell perimeter can be adapted to the needs of the resulting floorplan.

5. CMOS logic.

For ages, complementary MOS logic has been composed of a '0' conducting network of NMOS switches together with a '1' conducting network of PMOS switches. Logic function and output capability were given by the same set of devices. As a result, the need for larger output drive also led to larger transistors in the logic function and therefore to a higher loading of the steering gates. Cascode voltage switch (CVS) logic has been proposed to eliminate this problem. It consists of an NMOSTly logic function with a small loadtransistor for supplying a default '1' and a separate output buffer. These two inverting gates in series (logic & buffer) led to a non-inverting overall function; hence explicit inverters are needed. Path driven restoring (PDR) logic eliminates this problem[10]. It consists of a logic switch network, wherein P- and NMOS transistors can be freely mixed and a separate signal level restorer & buffer. It has been shown, that PDR logic implements gates with the least amount of transistors and is very well suited to be configured in any cell size. Fig.5 gives an example.

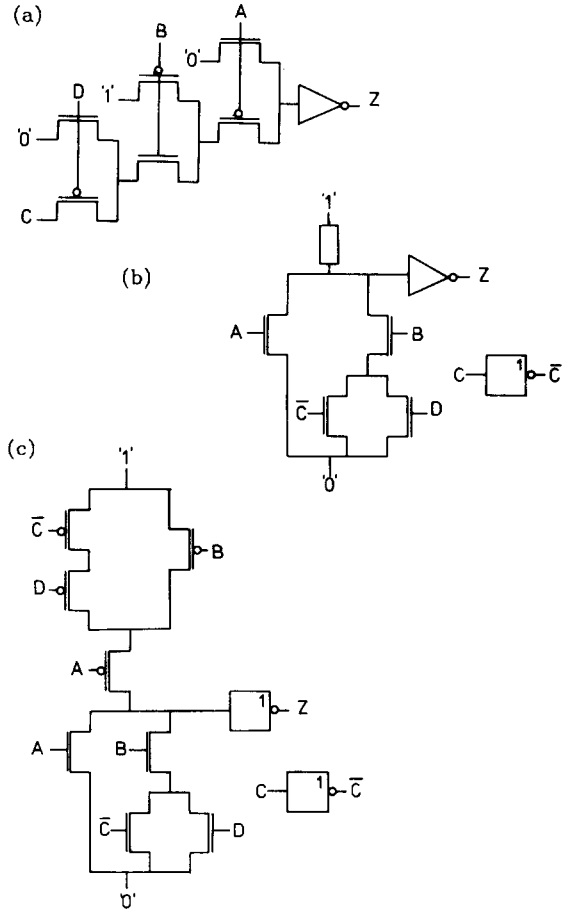

Figure 5. Transistor diagram for the equation Z=A+B(/C+D) in FCMOS (a), CVS (b) and PDR (c).

Self-timed logic can be created in basically two ways: by the introduction of an explicit delay, or by the use of double-rail completion coding. The latter is the most advantageous, but is only suited for use in conjunction with CVS logic. There, very attractive Muller C-elements can be constructed.

Control logic presents a separate problem. Regular layout would require a programmable logic array (PLA). But a PLA is wasteful in area consumption, cannot be configured into any cell size and does not provide optimal system speed. Hence its applicability is limited. The alternative is random logic using a standard cell approach. Here the state-cell approach is taken for the straightforward sequencers and PLA's for decoding like functions. In the state-cell approach[11] the sequential behaviour of a state diagram is realised by so-called state cells and arc cells, connected together in a manner that directly reflects the state diagram specification. The logic for creating transition conditions is assumed to be located elsewhere. In the case of simple sequencers, it is advantageous to find a place inside the structure for the generation of transition conditions. This is handled by allowing arc cells to be steered not only by entire transitions, but also by the literals in transition conditions.

6. The ST2 microprocessor.

The ST2 microprocessor is a reduced redesign of the Philips8500 microcomputer. Provisions have been made to add self-testing facilities at a later stage. For evaluation purposes the datawidth has been set to 8 bits and only 4 on-chip registers have been provided for. Using the function cell concepts, the microprocessor is built as shown in fig. 6.

The initial specification has been entered on specification sheets. All files in the MoDR suite of CAD tools read from and write to these documents. First the separated hierarchy is shuffled to find the best floorplan, which looks as shown in fig. 7.

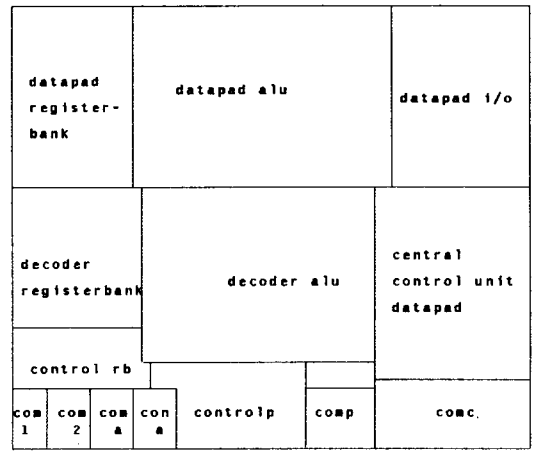

Figure 7. Floorplan of the processor.

Then the datapath has been composed from hand-layout parts, and the PLA and state-cell parts have been generated. Then the functions are assembled and routed to result in an experimental design measuring 9 mm square in a 5um retrograde twin-well CMOS technology (fig. 8). Compared to a previous redesign (also based on the principles of self-timing and self-testing) the overhead is reduced to less than 10%. The part is presently offered for production.

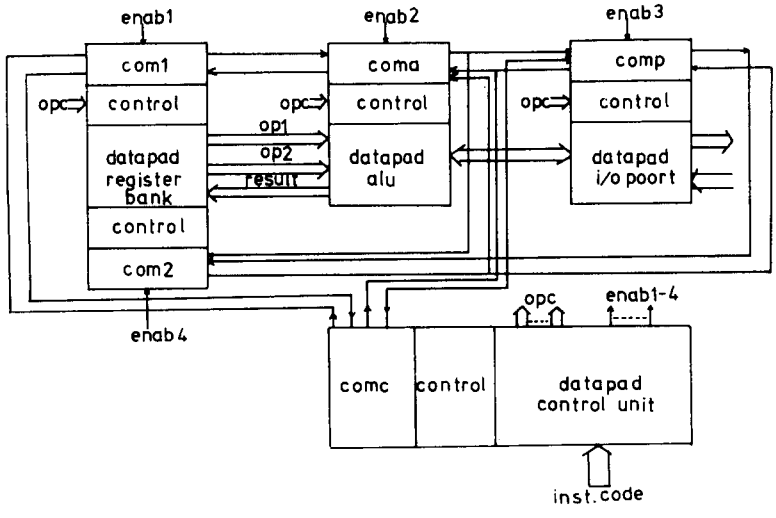

Figure 6. Architecture of the processor.

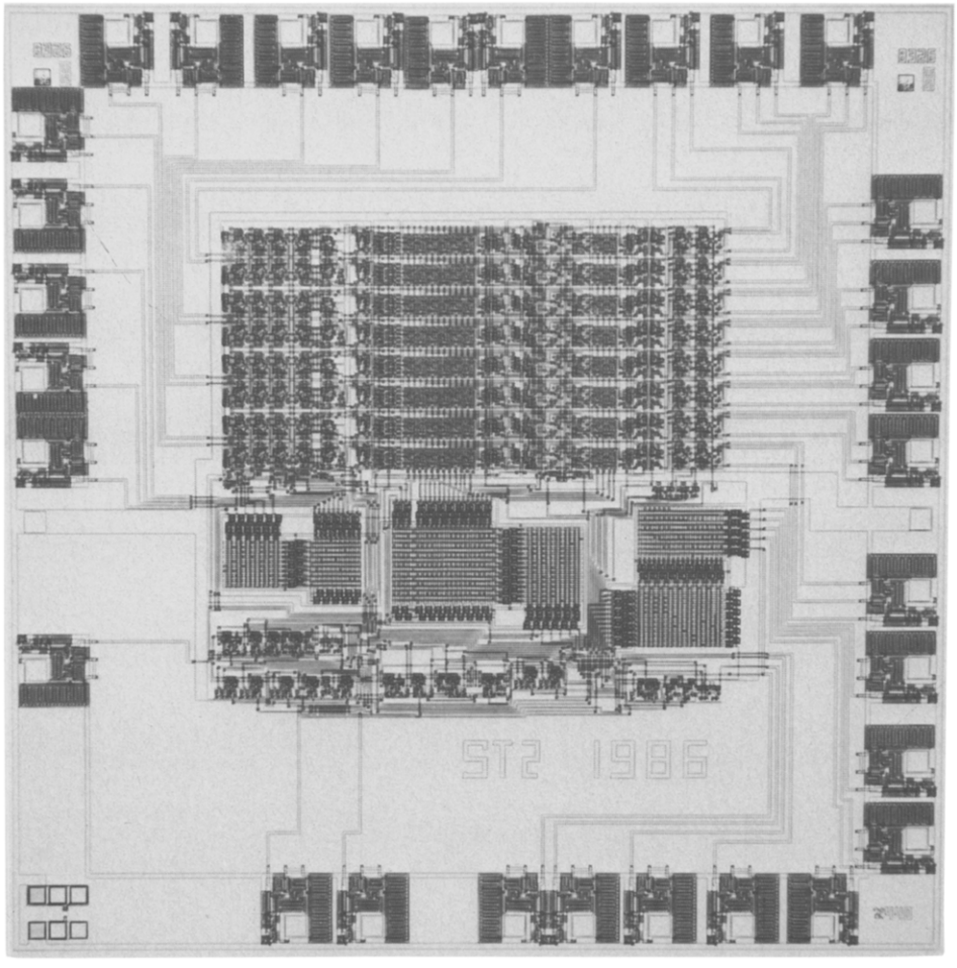

Figure 8. Mask layout drawing of the processor.

7. Discussion.

A methodology for the top-down design of integrated system has been introduced. It is based on the application of the separated hierarchy to eliminate lengthy iterations. Starting from an example specification, a shuffle on composite cells (the so-called "B-shuffle") is performed to find the best match between the specification and the anticipated floorplan. This already eliminates long trial-and-error runs. Then a function cell concept is applied to create a suitable environment to use self-timing. The exploitation of hierarchical testing concepts within this framework is presently in research. At the bottom level of abstraction, novel CMOS circuit techniques support the methodology.

The methodology has been applied in the design of an experimental CMOS microprocessor with RISC architecture. The total mask design was completed in a matter of weeks, a time that was governed by the availability of the CAD hardware and the effort to create the specification documents.

Acknowledgments.

The support of the CACSD night-shift is deeply appreciated.

References.

1. Rowson, J., "Understanding hierarchical design", Ph.D.thesis, California Institute of Technology, Pasadena (CA), 1981.

2. Gajski, D.D., "The structure of a silicon compiler", proc. IEEE-ICCC, pp.272-276, New-York (NY), october 1982.

3. Breuer, M.A. and Kumar, A., "A methodology for custom VLSI layout", IEEE Transactions on circuits and systems, vol.CAS-30, no.6, pp.3568-364, june 1983.

4. Spaanenburg, L. et al., "MOD/R: Top-down only CMOS VLSI design", Microprocessing and Microprogramming, vol.16, pp.83-88, 1985.

5. Ackland, B. and Weste, N., "An automatic assembly tool for virtual grid symbolic layout", VLSI83: VLSI design of digital systems, pp.457-468, North-Holland, 1983.

6. Spaanenburg, L. and Stolmeijer, A., "A short course in structured testable gate array design", Journal of Semicustom ICs, vol.3, no.3, pp.37-45, 1986.

7. Marques, J.A., "MOSAIC: A modular architecture for VLSI system circuits", VLSI81, Addison-Wesley, 1981.

8. Spaanenburg, L. et al., "One-chip microcomputer design based on isochronity and selftesting", Digest EDA'84, pp.161-165, Warwick(England), march 1984.

9. Seitz, C.L., "System timing", in Mead & Conway, "Introduction to VLSI systems", Addison Wesley, 1980.

10. Spaanenburg, L. et al., "Novel switched logic CMOS latch building block", Electronic Letters, vol.21, pp.398-399, april 1985.

11. Spaanenburg, L. et al., "A methodlogy for the fast and testable implementation of state diagram specifications", IEEE Journal of Solid State Circuits, vol.SC-20, no.2, pp.548-554, april 1985.