

Rapid #: -8177693

CROSS REF ID: 1199077

LENDER: IQU :: CSEL

BORROWER: IPL :: Main Library

TYPE: Article CC:CCL

JOURNAL TITLE: VLSI design

USER JOURNAL TITLE: VLSI Design

ARTICLE TITLE: Progress Toward the ``Ideal" Silicon Compiler Part 1: the Front End

ARTICLE AUTHOR: Werner, J.

VOLUME: 4

ISSUE: 5

MONTH: September

YEAR: 1983

PAGES: 38-41

ISSN: 0279-2834

OCLC #:

Processed by RapidX: 7/15/2014 10:41:42 AM



RAPID

ILL

This material may be protected by copyright law (Title 17 U.S. Code)

1 Rapid #: -8177693**Odyssey****IP: 128.210.126.171/ILL**

Status	Rapid Code	Branch Name	Start Date
New	IPL	Main Library	7/15/2014 7:02:11 AM
Pending	IQU	CSEL	7/15/2014 7:02:18 AM
Batch Not Printed	IQU	CSEL	7/15/2014 7:02:32 AM

CALL #: TK7874 V558**LOCATION: IQU :: CSEL :: sper**

TYPE: Article CC:CCL

JOURNAL TITLE: VLSI design

USER JOURNAL TITLE: VLSI Design

IQU CATALOG TITLE: VLSI design

ARTICLE TITLE: Progress Toward the ``Ideal'' Silicon Compiler Part 1: the Front End

ARTICLE AUTHOR: Werner, J.

VOLUME: 4

ISSUE: 5

MONTH: September

YEAR: 1983

PAGES: 38-41

ISSN: 0279-2834

OCLC #: IQU OCLC #: 7719803

CROSS REFERENCE ID: [TN:1199077][ODYSSEY:128.210.126.171/ILL]

VERIFIED:

BORROWER: IPL :: Main LibraryThis material may be protected by copyright law (Title 17 U.S. Code)
7/15/2014 7:02:32 AM

Part 1: The Front End

Progress Toward the "Ideal" Silicon Compiler

Jerry Werner, Editor-in-Chief

"The introduction of the automatic layout and wiring program helped reduce the layout processes. As a result, the area where the design cost can be cut down is shifting in the direction of the functional and logical design steps. Thus, it is essential that the automatic logical synthesis and the automatic design method be established."

—From a paper by NTT researchers presented at the 1983 Convention of the Electronic and Communication Society of Japan.

Ever since the phrase "silicon compiler" was coined in the late seventies, it has come to mean different things to different people. It *ideally* implies a design tool that automatically translates a high-level functional or behavioral description of a chip down to the actual layout of the device. With present-day capabilities, only designs that fall into a narrow range of pre-ordained "target implementations" have been designed successfully using this strategy (Werner 1982). Nonetheless, research efforts toward the ideal silicon compiler appear to be accelerating.

Recently published research suggests that the ideal silicon compiler may really involve solutions to two complementary design problems (see Figure 1). The first solution is the translation of a brief behavioral or functional description into a more precise intermediate description that is still implementation independent; and the second solution is the automatic generation of a chip layout from that intermediate description.

One important reason for separating the "front end" of a silicon compiler from the "back end" is the short-lived nature of the target semiconductor processes. "The back end has a life of maybe two years, because the technology changes," says Charles Rupp, manager of the exploratory research group at Digital Equipment Corp. in Hudson, MA. "For companies like DEC, that means that the back end might be used for two or three projects."

Behavioral vs. Structural Specifications

Most programs that purport to tackle the front end of the silicon-compiler problem start with either a behavioral or structural definition of the desired system. A behavioral description might be very similar to a common algorithmic language, containing statements in a form such as IF-THEN-

ELSE, or DO. Or else the behavioral specification might be a special-purpose language, whose syntax more closely emulates the behavior of common hardware blocks. Conversely, structural descriptions specify *which* blocks (or high-level functions) are used, as well as all inputs, outputs, connections, etc.

Although most experts agree that behavioral input exists at a higher level than structural input (even higher than a hierarchical structural description), some people believe that going from a behavioral to a structural description (even a high-level structural description) should be the province of engineers, not of computers. "It's a mistake to start from a behavioral description," flatly states John Gray, president of the Scottish software firm Lattice Logic, Ltd., which claims to have a silicon compiler for gate arrays. His rationale: "To map from a behavioral description to a physical description, you have to make assumptions about an architecture. That's what engineers are really good at. Coming up with an architecture is the creative part of designing. Engineers have all the right insights."

Others see the behavioral-input system as the real challenge—and one that offers a greater potential payback. "I'm not knocking them [structural-input systems]," says John Darringer, a research staff member at IBM's Thomas J. Watson Research Center in Yorktown Heights, NY. "But you'd like to do better than that—accept a description that just says the behavior but doesn't prejudice the implementation. You'd like to get more leverage—with transformations

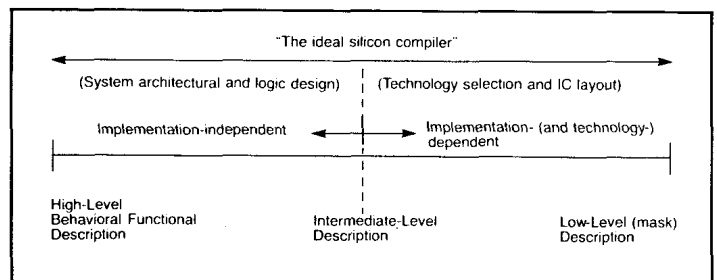


FIGURE 1. Although the concepts involved in an "ideal" silicon compiler span the entire design spectrum, much current research falls into either the left half (the technology-independent part) or the right half (the technology-dependent part) of the spectrum.

to allow you to think about better ways of implementing the hardware."

Still other knowledgeable people don't see a clear distinction between behavioral and structural input. "In reality, you can't distinguish these very much," says Stuart Feldman, a member of the technical staff in the research department at Bell Laboratories (Murray Hill, NJ). "When the smoke clears, you're saying (assuming you're above the geometry level) that 'this output depends this way on these inputs.' The question is whether you're concentrating on the devices or on the values." Feldman is presently coding Version 2 of the Xi (Ξ) language, a dialect of the C language modified for use as a behavioral circuit-description language (Feldman 1983).

However, according to IBM's Darringer, there is a big difference between programs like the Xi compiler (which has a one-to-one map between the behavioral description and the logic implementation) and programs that have intermediate steps to optimize the logic, remove redundant gates, and perform other tasks. "Some people don't make that distinction," he says, adding that programs belonging to the former class "are easier than taking an arbitrary [behavioral input] program and trying to figure out what the hardware is."

A Simple Illustration

Arnie Peskin, the principal investigator on a small silicon compiler project at Brookhaven National Labs (Peskin 1982), explains the type of thinking that goes into translating from an algorithmic statement describing *behavior* to a logic implementation. "A Boolean IF-THEN-ELSE would translate to a simple decoder," he says. "Or if you see 'DO something ten times' then you're invoking a counter."

Peskin, like Darringer and his colleagues at IBM, uses a top-down approach to the problem of creating a program that will translate from a behavioral description to a more explicit logic description. Others, such as the group that developed the MacPitts program at MIT's Lincoln Laboratory (Siskind *et al.* 1982; Fox 1983), are working both top-down and bottom-up. The MacPitts language, which specifies a chip behaviorally, takes into account which bottom-level primitives (called "organelles") are available to implement a data path. Declarations in the source specification for MacPitts determine what will be implemented in the datapath, and what will operate in the random-logic control portion of the chip. "The user does have to know which organelles are presently provided by the compiler," explains Jeff Siskind, principal researcher of the original MacPitts effort and now president of a fledgling silicon-compiler firm called MetaLogic (Bedford, MA). He adds that the low-level library is expandable: new functions can be defined, and the corresponding additions can be made to the MacPitts language.

Peskin, for one, thinks that it is needlessly restrictive to limit the possible behavioral input statements to those functions that have a known hardware implementation. "The user is worried about his entire problem," he says. "I see no point in constraining him. I'd rather let him describe the algorithm fully, in an attempt to see what can be rendered in hardware and what cannot."

Logic Synthesis

In systems that accept an unconstrained behavioral input, the concept of logic synthesis plays a vital role. Logic synthe-

sis is loosely defined as the translation from a high-level specification to a more explicit (but still technology-independent) gate-level (Boolean logic) specification. The concept of logic synthesis certainly isn't new: in the early 1960's, researchers were investigating ways to "synthesize" circuits containing off-the-shelf SSI parts.

But much of the early work in logic synthesis for large systems failed: perhaps because the programs didn't take into account factors such as circuit performance or the limitations of the target implementation. As Darringer and his colleague William H. Joyner, Jr. observed in a much-quoted paper presented at the 1980 Design Automation Conference (Darringer and Joyner 1980): "Previous efforts have dealt primarily with technology-independent primitives and have emphasized circuit minimization. However, larger scales of integration have made other design requirements and technology restrictions as important as circuit count."

One of the most basic restrictions is the target chip technology. "A classical problem is the bias of the implementation," says DEC's Charles Rupp. "TTL and CMOS are biased to NAND structures, and nMOS is biased to NOR structures. If you specify the design in Boolean logic [without taking that bias into account], then you could have a less-than-optimal implementation."

Synthesis Work at CMU

Researchers at Carnegie-Mellon University have been working on logic synthesis and transformation for several years (Parker *et al.* 1979). Much of their work involves the automatic translation from a behavioral description of a computer system in the ISP language (Barbacci *et al.* 1978) into a register-transfer level description.

Some experts in industry believe that CMU's earlier work using ISP started off at an unsuitably high descriptive level. IBM's Darringer, who holds the Ph.D. degree from CMU (1969), says, "With an ISP description, it's not obvious what the registers, operators, and signals are. If you talk to engineers at IBM, they'll say, 'I'll tell you what the operators and registers are; you do the rest of the work.'"

"Clearly, if you try to use that [ISP] directly, the system has to make a lot of assumptions," says Rupp from DEC. "Then you're leaving so much up to the software that if it blows half of its decisions, you have a pretty lousy design." Rupp adds that "A lot of people are speculating that a knowledge-based system would help out a lot there."

Indeed, the development of knowledge-based or "expert" systems for the front half of the silicon compiler is a popular topic in research labs and is just starting to draw attention in the product-development environment. At CMU, researchers recently used a prototype expert system to design the functional architecture of a 6502 microcomputer (Kowalski and Thomas 1983). They interviewed several expert VLSI/system designers at Intel and at Bell Laboratories, and created 130 design *rules* based on the experts' responses. The CMU researchers reported that it took only 4 hours of CPU time on a DEC VAX11/750 computer to design a 6502 architecture that the experts considered acceptable. However, the synthesized chip was never laid out or built to verify the performance of the architecture.

At Brookhaven, Peskin is also trying to incorporate "knowledge-based transformations" in his software. "The

transformation isn't set in concrete," he says. "There are different paths you can take. The user can say 'No, don't use that approach; it's not working out. Use a different one.' And the system learns which transformations are favorable and which are not." Peskin thinks that his software is better at optimization than it was when he described it at the 1982 Custom Integrated Circuits Conference (Peskin 1982). However, no chips have yet been fabricated (although the output of his system is a net list suitable for input into Sandia Labs' standard-cell layout software).

IBM's LTS

An effort is underway at IBM to experiment with logic synthesis and other types of transformations in several product development labs, albeit in a "field-test" mode. John Bendas, a department manager at IBM's Poughkeepsie (NY) facility, is heading that effort. Bendas described the "logic transformation subsystem" (LTS) at the recent Design Automation Conference (Bendas 1983). The LTS includes several different processes to convert a high-level system specification into a flattened net list suitable for input to IBM's master-slice (gate-array) layout system. Those processes include the following transformation steps:

1. *Synthesis*, defined as the transformation of a technology-independent high-level specification into a technology-independent low-level one (e.g., a Boolean description);
2. *Mapping*, or the transformation of a low-level technology-independent specification into a technology-specific one;
3. *Remapping*, or the transformation of one specific technological implementation into a new technology (e.g., TTL to CMOS); and
4. *Abstraction*, or the transformation of a technology-independent low-level specification into a functionally equivalent high-level one. (This process is essentially the reverse of synthesis.)

The IBM system primarily performs *local transformations*. In other words, it works on one block in a local neighborhood of logic. A user programming interface (UPI) was added recently to LTS so that local experts could write their own transformation programs and integrate them into the system.

IBM's work so far looks promising. In one test case, several small master-slice chips (approximately 500 gates each) were remapped into a single larger device (with approximately 3000 gates) by means of LTS. The projected reduction in manpower, as compared to the same task performed manually, was by a ratio of 10 to 1. More recently, the LTS has been used to synthesize the logic specification for the control portion of a processor from a flowchart-oriented hardware description language. Bendas declined to give further information to *VLSI DESIGN* about the status of these test cases, saying that such information was proprietary to IBM.

NTT's ANGEL

Several major research efforts to develop logic synthesis systems are underway in Japan. The most recent developments that have been described at leading technical conferences in Japan come from Nippon Telephone and Telegraph Corp. (NTT). At NTT's Musashino Electric Communication

Laboratory, a "gate generation system" nicknamed ANGEL is presently being developed (Endo *et al.* 1982). The ANGEL system progressively translates a functional-level specification written in Function Description Language (FDL) into the Hierarchical Specification Language (HSL). NTT's cell-based automatic layout program (called PLASMA) then automatically converts the HSL description into a complete IC-mask design.

Figure 2 shows the process flow from an FDL input to a fully flattened HSL output. Based on a comparison of the total number of fan-ins (which is said to be a good measure of circuit area) for circuits that were manually designed vs. the number of fan-ins for circuits that used ANGEL, the gate-level designs generated by ANGEL were from 1.2 to 2.7 times larger. However, the NTT researchers claimed that a 10,000-gate circuit that might take 5 to 10 work years of manual design can be generated in only 3 hours by using ANGEL on a 3-MIPS (million instructions per second) computer.

The NTT researchers found that the FDL description had a profound effect on the efficiency of the resulting HSL-level design. In one case, two different FDL descriptions (that nevertheless described the same circuit function) resulted in HSL designs that differed in size by a factor of nearly three to one.

More recently, NTT used ANGEL to design the equivalent of the Advanced Micro Devices AM2909 microprogram sequencer chip (Kitamura *et al.* 1983). Table 1 shows the results of manual vs. automatic design. The gate-generation program required approximately 30% more total gates and 20% more chip area than the manual design.

The NTT researchers acknowledged that the chip layout did not take into account the original FDL description, and agreed that such a consideration should reduce the total chip area used by interconnections.

Could Silicon Compilation Be a Three-Part Problem?

Although many researchers draw an arbitrary line between the front and rear portions of the "ideal" silicon compiler, the problem may fall into three parts. "Our system now has three stages," says IBM's Darringer. "The first one is genuinely technology independent, the bottom stage [chip layout] is technology dependent, and we have a not-very-

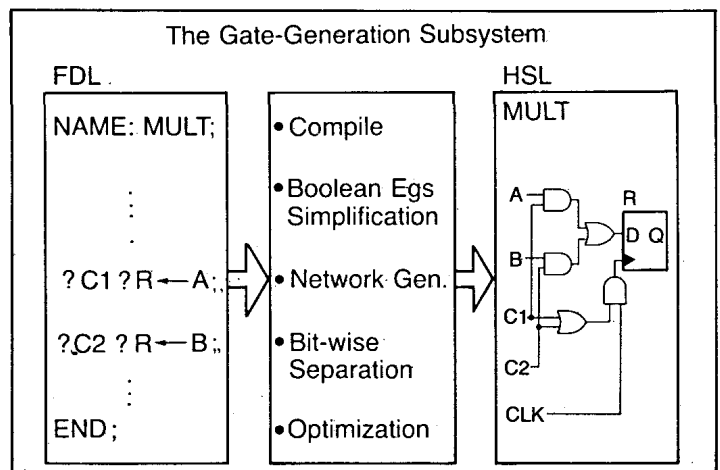


FIGURE 2. NTT's ANGEL system.

THE GATE-GENERATION SUBSYSTEM

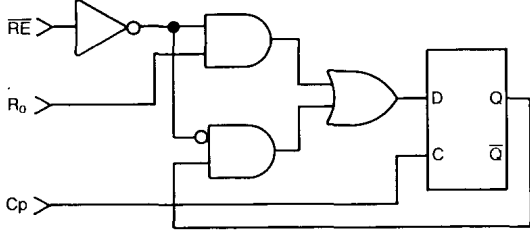
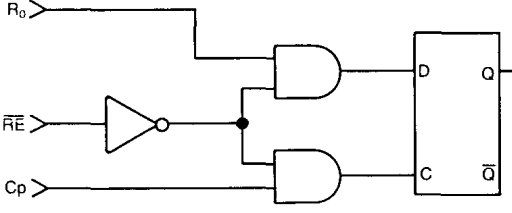
	Manual Design	Gate-generation
Signal I/O	Design takes fan-in and fan-out into consideration	Design takes function only into consideration
Address-register portion	Register value maintained by the feedback of output	Register value maintained by controlling the clock
		
Incrementor portion	Incrementor (approx. 8 gates)	Use adder (approx. 40 gates)
Stack portion	RAM and stack pointer are used	Shift register is used
Total number of gates	Approximately 230 gates	Approximately 300 gates
Chip area	Approximately 0.78 mm ²	Approximately 0.92 mm ²
Wiring-area ratio	Approximately 50.6%	Approximately 51.8%

TABLE 1. Comparison of manual and automatic design of the equivalent of an AM 2909 chip.

complicated middle stage where you *start* to worry about the technology. In the middle stage, you might choose your circuit family (e.g., TTL or ECL), but won't worry about fan-out limitations until much later."

Nevertheless, as the various steps in translating a behavioral description into an input suitable for automatic or semi-automatic IC layout tools become better understood, it is clear that we are asymptotically approaching the goal of truly automatic chip design. □

References

- Barbacci, M., G. Barnes, R. Cattell, and D. Siewiorek. March 1978. "The Symbolic Manipulation of Computer Descriptions; The ISPS Computer Description Language," CMU technical report.
- Bendas, J.B. June 1983. "Design Through Transformation," *Proceedings of the 20th Design Automation Conference*, Miami Beach, FL.
- Darringer, J.A. and W.H. Joyner, Jr. 1980. "A New Look at Logic Synthesis," *Proceedings of the 17th Design Automation Conference*, Minneapolis, MN.
- Endo, M., T. Hoshino, and M. Nagatami. 1982. "An Engineering Project: Gate Automatic Generation Program," *The 25th Convention of the Information Processing Society* (in Japanese).
- Feldman, S.I. October 1983. "The Circuit Design Language Xi (Ξ)," (to be presented at the *1983 IEEE International Conference On Computer Design*, Port Chester, NY).
- Fox, J.R. May/June 1983. "The MacPitts Silicon Compiler: A View from the Telecommunications Industry," *VLSI DESIGN*.
- Kitamura, Y., M. Nagatami, and T. Hoshino. 1983. "Evaluation of the LSI Automatic Design Using the Function Description Language," *1983 All Japan Electronic Communication Society Convention Record* (in Japanese).
- Kowalski, T.J. and D.E. Thomas. June 1983. "The VLSI Design Automation Assistant: Prototype System," *Proceedings of the 20th Design Automation Conference*, Miami Beach, FL.
- Parker, A.C., D.E. Thomas, D.P. Siewiorek, M.R. Barbacci, G. Leive, and J. Kim. June 1979. "The CMU Design Automation System: An Example of Automated Data-path Design," *Proceedings of the 16th Design Automation Conference*.
- Peskin, A.M. 1982. "Toward a Silicon Compiler," *Proceedings of the 1982 Custom Integrated Circuits Conference*, Rochester, NY.
- Siskind, J.M., J. R. Southard, and K.W. Crouch. 1982. "Generating Custom High-Performance VLSI Designs from Succinct Algorithmic Descriptions," *MIT Conference on Advanced Research in VLSI*, Cambridge, MA.
- Werner, J. September/October 1982. "The Silicon Compiler: Panacea, Wishful Thinking, or Old Hat?" *VLSI DESIGN*.