(12)

AD-A160 292

unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 85-10-03	2. GOVT ACCESSION NO.	3. ~~RECIPIENT'S CATALOG NUMBER~~
4. TITLE (and Subtitle) UW/NW VLSI Consortium Semiannual Technical Report No. 1		5. TYPE OF REPORT & PERIOD COVERED Technical, interim
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) UW/NW VLSI Consortium		8. CONTRACT OR GRANT NUMBER(s) MDA903-85-K-0072
9. PERFORMING ORGANIZATION NAME AND ADDRESS UW/NW VLSI Consortium, Dept. of Computer Science FR-35, University of Washington, Seattle, 98195		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS DARPA - IPTO 1400 Wilson Boulevard Arlington, Virginia 22209		12. REPORT DATE October 1985
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) ONR University of Washington 315 Univ. District Building 1107 NE 45th St., JD-16, Seattle, WA 98195		15. SECURITY CLASS. (of this report) unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this report is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

DTIC ELECTE OCT 16 1985 S D A

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

VLSI Design generators, VLSI Consortium, module generators, RAM, ROM, multiplier, CFL. [illegible]

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This document reports on the research activities of the University of Washington/Northwest VLSI Consortium for the period 19 March 1985 to 30 September 1985 under sponsorship of the Defense Advanced Research Projects Agency, under contract number MDA903-85-K-0072, program code number 5D30. keywords include:

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DTIC FILE COPY

UW/NW VLSI CONSORTIUM

Semiannual Technical Report No. 1

University of Washington

September 30, 1985

TR# 85-10-03

DTIC
COPY
ED

Accession For

NTIS CRA	☑
DTIC TAB	☐
Unannoun	☐
Justificati	

y

tion/

ailability Codes

A-1	Avail and/or Special

Reporting Period: 19 March 1985 to 30 September 1985

Principal Investigator: Lawrence Snyder

Sponsored by
Defense Advanced Research Projects Agency(Dod)
ARPA Order No. 4563/2
Issued by Defense Supply Service-Washington
Under Contract #MDA903-85-K-0072
(Program Code Number: 5D30)

The views and conclusions contained in this document are those of the authors and should not be interpreted as represented the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency of the U.S. Government.

85 10 15 065

Contents

1 Executive Summary

1.1 Scope of This Report

This document reports on the research activities of the University of Washington / Northwest VLSI Consortium for the period 19 March 1985 to 30 September 1985 under sponsorship of the Defense Advanced Research Projects Agency. The applicable contract for this period is MDA903-85-K-0072.

1.2 Accomplishments

During this period the Consortium completed initial versions of nine generators which are now available to designers. CFL (Coordinate Free LAP), the primary tool used in constructing generators, was enhanced considerably as a result of the feedback of the generator writers. Generators were used in two substantial chip designs, a digital filter and the Quarter Horse, a 32-bit microprocessor. Ongoing work focuses on the generation of output descriptions other than the layout, in particular the DRC interface, the transistor netlist, and the functional description. These output descriptions are a critical requirement for interfacing the generators to a custom CAD system.

The Consortium supported both regular University instruction in VLSI design, as well as a special intensive design class for industry engineers. Classes planned for the coming academic year will focus on architectural alternatives in microprocessor design and use the Quarter Horse as a base for enhancement.

In June the Consortium began distributing Release 3.0 of the VLSI design toolset. Currently over 40 sites have received this release which supports nMOS as well as MOSIS 3 micron CMOS. As the generator project continues, the Consortium intends to make periodic releases of software developed for the project.

2 Release 3.0 of the Consortium Toolset

A new release of the toolset was put together during Spring of 1985 and released in June. A number of new programs and capabilities have been added.

A major improvement in physical design tools has been the addition of Coordinate Free LAP (CFL), a library of "C" procedures that facilitates the layout of VLSI designs. Designed and written by Bill Beckett of the Consortium staff, CFL contains a variety of operators for juxtaposing, transforming and replicating hierarchies of cells. Although CFL has sufficient functionality to allow specification of arbitrary rectilinear mask geometries, it is mainly intended to be used in the chip assembly mode. Hence the typical application would be to use a graphical editor such as Caesar or Magic to generate lower level cells (or tiles) and then use CFL to assemble these cells into higher level modules. Routing facilities are provided which generate a variety of planar and nonplanar wire patterns to connect functional blocks. CFL is approximately 60 times faster than the aging Pascal-based layout facility PLAP, which it replaces.

A number of layout generators are included in the release. Written with CFL, each produces a layout satisfying the MOSIS 3 micron CMOS specification. A multiplier generator produces a NxM two's complement multiplier with ripple carry addition. A decoder generator creates a dynamic nor form decoder with an arbitrary number of address bits and banks. A padframe generator creates a MOSIS-acceptable padframe with input, output and tristate pads instantiated according to user specifications.

Several new programs have been added to allow easier construction of RNL control files for the switch level timing simulator RNL. These include GEN_CONTROL and GEN_TIME. The utility SIMSCOPE displays signal behavior derived from RNL or SPICE on several different graphics terminals.

So far 40 sites have received the distribution tape.

3 Initial Design Generators

During the past 6 months, the Consortium has completed initial versions of a number of design generators. One goal of this work is to have a library of generators available for designers of complex chips. In this way important feedback can be obtained on the utility of the output descriptions produced by the generators. Another goal is to provide insight into the techniques by which a generator is constructed. Once such pieces of information are gleaned from these *ad hoc* generators, we will be better able to define a generator methodology and appropriate data structures to support future versions of generators. The goal, of course, is to systematize generator construction and to support it appropriately with CAD tools.

A complete list of these generators appears as an appendix to this report. All were written for the MOSIS 3 micron process. Several instances created by each generator have been or are in the process of being fabricated.

The current set of generators has been constructed using the following procedure. Given a class of circuits (e.g. static RAMs), the generator writer chooses a subset which may be used in a variety of design situations. For the particular example of the static RAM, this subset was chosen to include both single and dual ported versions and to encompass an arbitrary address range as well as word size. The designer then lays out leaf cells and assembles them to create instances of the design. The designer checks them for both geometrical and electrical design rule correctness and simulates them to ensure correct functionality. The layout is commonly created graphically using Caesar, or more recently, using Magic. With the help of a library of layout procedures known as CFL (Coordinate Free LAP), the designer writes a C program for assembling leaf cells into instances of the circuit class. The parameters for specifying the form of the instance are passed as command line options to the program. A brief man page describes the generator parameterization.

As the initial set of generators was developed it became clear that enhancements to CFL would make the generator writer's job a lot easier. For example it was found that the addition of a "fill" operator extending the material on one border of a cell could be used to adjust the size of the driver transistors and the width of ground/ power buses. In this case and in several others the development of the generators spurred the further refinement of CFL as the generator writer's tool.

Currently, the primary output description is the layout. A border description of the layout is outputted as a byproduct of CFL. This description consists of the locations of all material which lies along the bounding box of the layout, as well as associated labels. This description has been successfully transformed into a format that can be used by Magic's global router. Work is currently underway to allow generators to output only geometry within a certain distance of the bounding box. This description would provide an interface to a design rule checker that could save considerable time in not having to check the inside of the generated component. Work is currently underway to allow generators to output a schematic description as well as the corresponding transistor netlist.

It is clear that the documentation for a generator requires careful consideration. A man page is clearly insufficient for providing the detailed information the generator user would like to know. What is needed is a medium versatile enough to accomodate timing diagrams, circuit schematics and simulation information. Work is in progress to develop such a documentation tool.

Several complex designs have been constructed that have employed a number of generators. An eighth order IIR digital filter design of about 25K transistors utilized the multiplier, PLA, and padframe generators. The addition of the RAM and counter generators to the library will allow a filter designer to put together an arbitrarily configured filter with relatively little effort. The amount of low level custom design required would be very small. The Quarter Horse microprocessor (described in detail in the appendices) employed the PLA, RAM and padframe generators.

Continued enhancement of these major chip designs will provide a test bed for the progress of the generator project. We hope to be able to ascertain the usefulness of our design generators across a spectrum of applications. We also hope the chip designers will be able to give us feedback on the utility of the output descriptions produced by the generators.

4 The Quarterhorse Microprocessor

The Quarter Horse is a single chip microprocessor whose design and implementation in custom CMOS was completed in 90 days. It was undertaken by the Computer Science Department's advanced VLSI design class during the spring quarter of 1985. Seven people contributed to the project which was submitted to MOSIS for fabrication in April.

The Quarter Horse is a 32-bit microprocessor. External memory is addressed as a 32 bit word, while the program counter is 30 bits long. The machine uses word addressing. Instructions are also 32 bits in length. 32 general purpose registers are available for use by the programmer.

A number of different architectures were explored before the traditional "data path" design was decided upon. This consists of several functional blocks (ALU, shifter, register file, program counter) which are laminated together and communicate through a common data bus. These are then controlled by a PLA, which specifies both the clocking and control to execute a particular instruction. Since the PLA is a software programmed element, we were able to delay some critical decisions until the last minute, thus allowing a parallel design approach. In addition to the PLA there is hardware for prefetching the next instruction and for evaluating branch conditions.

An advantage of using a PLA for control, together with instructions of variable numbers of cycles, is that the instruction set can be easily reconfigured on subsequent versions of the design. By reprogramming the PLA, a very simple or complex set can be created. Those currently implemented include most of the instructions available on the RISC microprocessor. In addition, more complex character manipulation instructions are included to demonstrate these advantages.

A number of lessons, outlined in a paper to be presented at the 1985 ICCD conference, were learned while doing the project. In general, these point to the advantages gained from choosing a "flexible architecture which delays binding" and to the use of generators.

A second version of the processor is currently under design. The intention is to create a generator which is capable of creating a unique instance of the Quarter Horse for VLSI experimental purposes.

A technical report describing the Quarter Horse appears as an appendix to this report.

5 Status of the Test Laboratory

5.1 Experimentation With the Laser Prober

The Consortium is a beta-test site for a laser prober prototype developed at Lincoln Labs. This tool provides access to the states of internal nodes of a system without damage to the unit under test. Such a tool, combined with a circuit simulator, has the potential to locate design flaws. With the help of Lincoln Labs in supplying key custom parts, we have been able to duplicate the prototype they have developed.

We have performed experiments on a number of 3 micron chips with nodes easily controllable from the pins of the IC. We have also developed a user's guide to the tool, which is currently under review at Lincoln Labs.

The laser prober experiments have established the proper design and operation of critical building blocks in MOSIS fabricated chips. However, we would like to improve our confidence in the results achieved so far. Our experience indicates that a number of factors make it sometimes difficult to determine a node's state from waveform produced by the prober. One factor is the appearance of a pulse whether the unperturbed device is on or off, although the magnitude of the pulse usually, though not always, differs by a factor of two between the two cases. The comparison of the two pulses can be made easier if the node can be made to switch during the probing operation. Additionally, the response pulse seems very sensitive to the size of the device: the state of large devices is significantly easier to determine than that of minimum sized devices. Finally, if the IC under test is changing state, the current sense amplifier response is dominated by the devices being switched, causing large power supply current changes to be observed by the oscilloscope.

The laser prober has been used on several MOSIS fabricated chips. One particular example is a chip containing 16 16-bit synchronous counters. Initial tests indicated that most of these chips were nonfunctional. In probing these chips our intent was to determine whether the problems were due to the output pads or the metal2 interconnect. Probing showed that neither was responsible and that the problems arose from different (as yet undetermined) yield problems in the counter arrays.

Our conclusion is that laser probing offers a major advantage over contact probing in VLSI circuits. There are however problems in making definite conclusions about the internal states of the IC. We hope that by sharing our experiences with the Lincoln Labs people we can inprove the sensitivity of the instrument.

5.2 Improvements to the Test Facility

The Consortium recently added significant testing capability with the purchase of a Northwest Instruments Test System. This system increases the number of stimulus channels from 16 to 32 and the number of acquisition channels from 32 to 96. The speed at which chips can be tested increases from 20 MHz to 100 MHz. An IBM PC/AT controls the test sequence and supplies the user interface for constructing the stimulus patterns and analyzing the acquired data. An ethernet card allows communication with the VAX 780 so that one may download test vectors created on the VAX. The additional test capability provided by this system will allow better characterization of designs produced by generators. Large chips such as the digital filter and the Quarter Horse microprocessor, for which the existing test gear was inadequate, can now be exhaustively tested.

6 Educational Offerings

6.1 The CMOS Intensive Class

During June and July the Consortium conducted its third intensive class in CMOS design. Eleven attendees from four different organizations participated, seven of whom opted to complete a substantial design as an independent project.

The course itself is divided into two parts. During the first three days of lectures and labs the students learn the fundamentals of CMOS design and the use of the Consortium simulation tools. The next two weeks are devoted to independent work on individual designs. Three days of lectures and labs follow, during which the students learn layout and verification techniques. The next month is spent completing the layouts of individual projects. Designs are fabricated through the MOSIS commercial facility and the returned parts may be tested using the Consortium's test facility.

6.2 UW Course Offerings in VLSI

During the academic year 1985-86 the Consortium will continue its support role for VLSI-related courses of the Computer Science and Electrical Engineering Departments. This year, however, the course sequence will expand with an exciting new architecture course – Architectural Alternatives for VLSI Microprocessors. This course will add a new dimension to the VLSI activities at the University, and build upon the work of previous classes. It is believed that by integrating architectural concepts in the VLSI sequence, the complexity and sophistication of the resulting chips will be increased.

6.3 Seminars

On August 26 the Consortium staff conducted a seminar intended to introduce the engineers in the member firms to the tools in the Consortium toolset. The session was limited in scope — students were not taught how to use specific tools, but rather were shown examples of design situations in which the tools were used effectively. A future seminar is planned which will focus on design generators and CFL.

7 Appendices

7.1 A Summary of Consortium Design Generators

The following design generators have been developed for the MOSIS 3 micron process. Most are installed in the Consortium toolset.

```
PADFRAME        -- generates a MOSIS-acceptable 3 micron CMOS padframe.
    Options:
        Frame size and number of pins.
        Pad type and location on frame.
        Layer for routing connection.

MUX             -- selects one of a number of multiple bit busses
    Options:
        No. bits on each bus.
        No. select wires.
        Inverting/noninverting outputs.
        Style nand(single clock) or nor(2 nonoverlapping clocks).

CAM             -- generates a content addressable memory and associated RAM.
    Options:
        No. words.
        No. bits per word in CAM.
        No. bits per word in static RAM.

MULTIPLIER      -- generates a static style multiplier.
    Options:
        No. bits in multiplier.
        No. bits in multiplicand.
        Signed/unsigned.
        Location of Vdd/GND.

BUFFER          -- increases the drive of a signal with a chain of inverters.
    Options:
        Load capacitance.
        Charging/discharging time desired.
        Inverted/noninverted output signal.
        Width of N transistor of first inverter.
        Stage ratio of the string of inverters.
```

DECODER -- decodes an address.

- Options:
  - Style nand(single clock) or nor(2 nonoverlapping clocks).
  - No. of select inputs.
  - Spacing between decoded outputs.
  - Inverting/noninverting outputs.

RAM -- generates a single or dual ported register file.

- Options:
  - Single/dual port.
  - No. of address bits.
  - No. bits per word.

PLA -- generates a dynamic (2 clocks) PLA.

- Options:
  - Strapping of nor plane inputs with metal2.
  - Stretching of power and ground lines.

ROM -- generates a dynamic style ROM.

- Options:
  - Single clock (NORA) or multiple clock style.
  - No. of address bits.
  - No. of bits per word.

7.2 Automated Generation of Microcontrollers

Barry W. Jinks* — David L. Pulfrey** — Warren S. Snyder***

* Microtel Pacific Research Liaison, UW/NW VLSI Consortium, Seattle, WA98195
** Dept. of Electrical Engineering, U.B.C., Vancouver, B.C. V6T 1W5
*** GTE Laboratories, 40 Sylvan Road, Waltham, MA 02254

ABSTRACT--The concept of an algorithmic microcontroller has been investigated. Software has been created which, when supplied with design parametrics, will generate several representations of the desired instance. This paper discusses the methodology followed during generator creation and the architecture and instruction set of the resulting family of controllers.

I. INTRODUCTION

The use of microcomputers as on-chip building-blocks is an attractive proposition for designers wishing to realize large integrated systems in silicon. Various approaches to this end are being explored, including standard library macrocells, silicon compilers and microcomputer generators. In a recent embodiment of the latter approach[1], the architecture of the microcomputer macrocells is essentially fixed but the user has control over important parameters such as data width, numbers of registers and the memory content and size. In the present work we have taken this concept a step further along the road to flexibility and area minimization by adopting a fully parametric approach to the design of a microcontroller.

By focusing on the microcontroller, rather than on a general purpose microcomputer, we have been able to limit the global instruction set and hence reduce the complexity of the generator design. The microcontroller generator is a software design environment which consists of a suite of subprograms capable of producing a number of different data base representations of a given instance. When furnished with the final system parametrics, the system synthesizes the mask geometries for a microcontroller to be instantiated in a user-specified system.

The microcontroller comprises a microprocessor, memory, communications protocol hardware and analog and digital I/O. It is intended to form a complete control system on a chip for use in telecommunications applications. The present work focuses on the microprocessor and memory portions of this generator.

II. GENERATOR DEVELOPMENT METHODOLOGY

Initial system design and functional level simulation was performed using a behavioral level simulator, after which a LISP-like description of the circuit, including parasitics, was created with MIT's NETLIST program. Leaf cells were then laid out using MAGIC, the new layout editor from Berkeley. Positioning of the leaf cells is performed using Coordinate Free LAP (CFL), a program developed at the UW/NW VLSI Consortium [2].

CFL was designed with generator creation in mind. As such it allows the designer to write a "C" program which embodies the general structure of the generator, even before the leaf cells have been designed. As these cells are created, or changed, CFL automatically aligns them in the correct fashion. CFL also provides the designer with autorouting primitives and, further, creates a complete description of the border of the newly-generated device. This enables manipulation of the device by any higher level program to proceed by accessing only a very small amount of data.

Once the layout phase was complete, the instances were extracted using MAGIC. Simulations with the switch-level simulator RNL were then performed to compare results with the transistor netlist, following which changes to NETLIST, the leaf cells and CFL were made to ensure agreement. Layout verification, and any further necessary model adjustments were then made prior to development of the test procedure. A self-test methodology based on BILBO[3] is currently being developed. This will be driven by the same input parameters as used by the CFL and NETLIST programs such that the result in a signature register (see Figure 1) can be predicted and used to detect simulated faults.

The complete data base representation of a generated microcontroller comprises four files, namely: a simulation file incorporating the transistor netlist, the mask information file containing the layout geometry information and design rule checker information from MAGIC, a border file giving the generator footprint from CFL, and a test file containing the test data.

A satisfactory degree of geometric rule independence is achieved by creating a master rule set from the rules of foundries likely to be employed in the microcontroller fabrication. The leaf cells are designed from this rule set using a graphical layout editor. This approach offers an alternative to the algorithmic technology file approach proposed elsewhere[4]. As our intended applications for microcontrollers are in the telecommunications field, both analog and digital circuitry are likely to be required on the chip. This limits the suitable fabrication technologies to those that use only single layer metal since "double-poly, double-metal" processes are not widely available.

III. DESIGN OBJECTIVES AND PRIORITIES

In establishing priorities for the design, the following axioms were considered to be appropriate to control systems in telecommunications: i) Control algorithms tend

†Funded in part by the Defense Advanced Research Projects Agency under Contract MDA903-85-K-0072.

to concentrate on bit manipulation operations rather than arithmetic operations; ii) the quantity of transient (i.e. read/write) data is relatively small and the data structures are simple; iii) control processors need not be high performance machines; iv) system timing is of critical importance in control applications.

Accordingly, it was determined that the priorities should be ordered as follows: size, simplicity and performance. To keep the design small, considerable effort was devoted to reducing the size of the largest elements (i.e. RAM, PLA, and ALU). To simplify the architecture, a reduced instruction set similar to RISC[5] was chosen. This was further simplified by treating all data as globals which reside in the same memory space, so eliminating the need for memory reference instructions. In addition, the number of control lines has been kept low to reduce bus routing problems. To maintain timing consistency, a constant cycle time for all instructions (including branches) was chosen. Performance issues were addressed in hardware by employing: (1) sub-generators with an algorithmic driver sizing capability[6]; (2) a modest pipeline and (3) separate data and control spaces. The resulting architecture is depicted in Figure 1.

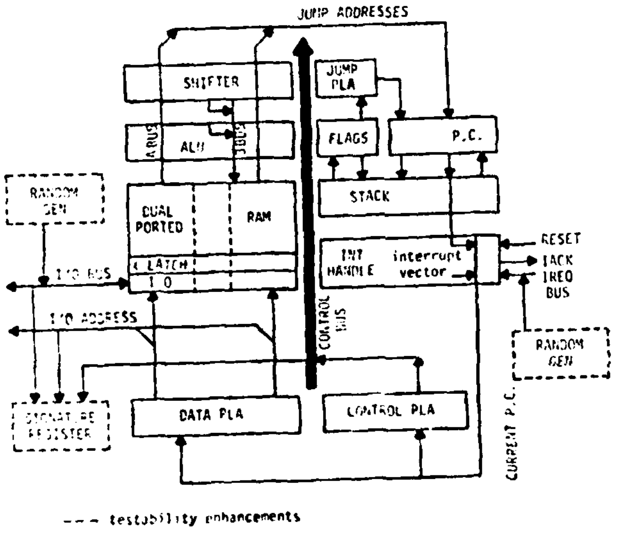

FIGURE 1. MICROCONTROLLER ARCHITECTURE

IV. ARCHITECTURE

Since both program and processor are contained within the same block, the system can be viewed as a finite state machine with the state feedback being the current program counter. A microcoding technique employing PLAs is used to produce the control and data path signals in parallel. Separation of these paths allows PLA minimization techniques to produce a more compact structure. The output of the PLAs remains valid for an entire instruction cycle, see Figure 2.

System timing is based on a two-phase non-overlapping clock scheme, with one machine cycle taking two ticks of each clock to complete. The machine cycle is partitioned into read and write half-cycles by a state clock derived from one of the system clock phases, see Figure 2. The RAM must be read in the first half-cycle and written-to in the second half-cycle. The output of the RAM is dynamically latched by C^2MOS latches, using NORA[7] circuit techniques. The RAM itself is implemented in domino CMOS.

The RAM output, consisting of the contents of registers, I/O data or immediates (which have passed through it) form the input to the ALU and shifter. These elements perform their required operation and force the result back to the register file on the Bbus only.

The RAM is optimized for speed and it is noteworthy that the slowest devices, namely the ALU (slow carry chain) and PLA (high capacitance on the term lines) are able to operate on cycle times which are one half that of the RAM. The RAM, ALU and shifter, which together form the data path, are pipelined so that one is precharging while the other is evaluating.

The data path output may consist of branch addresses which form the input to the program counter. A PLA is used to compare the flags with the instruction condition code. If a subroutine call is to be executed, the current program counter and flags are pushed on the stack.

Interrupts are controlled by the interrupt handler. When an interrupt is being requested; the associated vector is forced onto the current P.C.bus. This causes a jump to the state containing a call to the appropriate interrupt service routine.

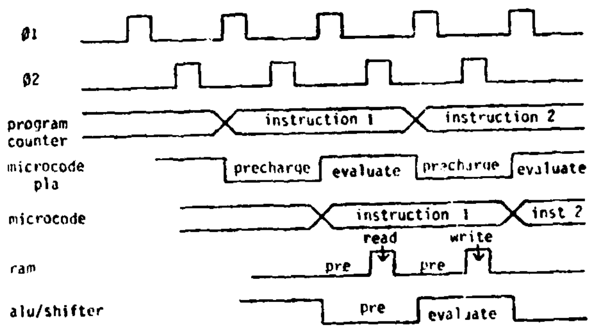

FIGURE 2. SYSTEM TIMING

V. GENERATOR INPUT PARAMETERS

The 9 lower level blocks of the controller are synthesized by 8 unique sub-generators. The input parameters to these generators are the register address width, number of interrupts, number of i/o ports, program counter width, stack depth and data path width. In addition, the ALU, shifter and branch PLA operations are defined. Many of these parameters can be synthesized by analysis of the instructions used in a particular program. This ensures a compact microcontroller which is capable of executing a local subset only of the global set available.

The ALU is an example of where this approach to area reduction is used. In addition to the data path width, the ALU generator has a switch which directs it to synthesize an "arithmetical/logical" or just a "logical" unit, depending on the particular operations required. The shifter generator works in a similar way and synthesizes only the actual paths used in the crossbar switch, instead of all those that are possible.

VI. INSTRUCTION SET

The global instruction set has been devised to provide a limited but powerful set of instructions. All instructions execute in one machine cycle and there are 22 instruction types, see Figure 3.

All of the read/write registers in the processor reside in the dual ported register file. The i/o address space is also memory mapped, therefore, there are no divisions between registers, memory and I/O ports. This greatly simplifies the instruction set while making it completely orthogonal. Several special (x) latches may also reside in the RAM. These allow indirect addresses to be stored and used to access the registers. Since the RAM is designed to have A and B busses read simultaneously but a write to the B bus only, all data path operations are of the form:

(Reg1/data/io) OP (Reg2/io) ---> (Reg2/io)

The source of jump or call addresses is the RAM as well, thus addresses may be immediate or computed. At present, 8 branch conditions are provided for.

By keeping the number of control lines small (<=14), it has proved possible to implement horizontal microcode (i.e. no instruction decode), producing in this case 784 unique instructions. This provides maximum flexibility while maintaining architectural simplicity. The result is that operations which do not use the data path (i.e. flags, interrupt control, return) can be executed in parallel with those that do.

DATA PATH OPERATIONS:

operand pairs:	operations:	
data, reg	ADD	SHLL
reg1, reg2	ADDC	SHLA
*x , reg	SUB	SHRL
reg , *x	SUBC	SHRA
data, *x	AND	RLL
x , reg	OR	RLA
reg , x	XOR	RRL
	MOVE	RRA

*x = an indirect address

OTHER OPERATIONS:

FLAGS LATCH
CALL ON CONDITION
RETURN ON CONDITION
JUMP ON CONDITION
COMPARE
ENABLE/DISABLE INTERRUPT

FIGURE 3. GLOBAL INSTRUCTION SET

VII. RESULTS

The generator has been designed in 3μ CMOS, using a conservative set of the MOSIS design rules. Several instances of the microcontroller have been simulated using NETLIST in conjunction with RNL. This has shown that the speed of a particular instance is highly dependent on the resources used. For example, for moderate word widths, the ALU carry chain delay dominates. If the carry chain is not present (i.e. logical operations only), the program counter becomes the speed determining factor. Most instantiations are expected to perform with a processing rate better than 2 million instructions/second.

Figure 4 is the check plot of a test instance with an 8 bit word width, 31 general purpose registers, 1 indirect address register and an 8 deep stack. Each subgenerator is fully implemented (i.e. all instruction types can be executed) and the program is 280 steps in length. The pad frame has a cavity which is 5.5mm on a side.

Some of the major data path items, notably the RAM and the shifter have already been fabricated. Initial testing shows that the functionality and performance track well with the expected results.

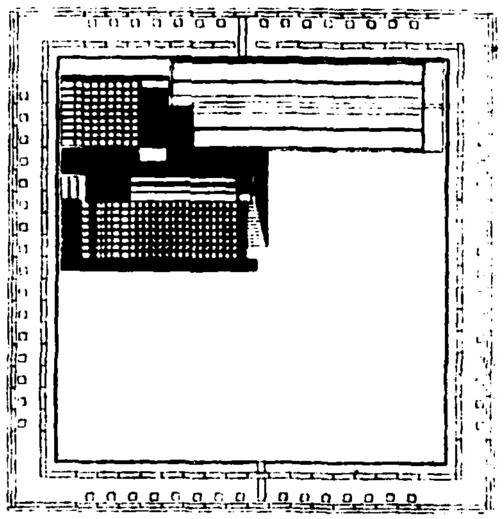
Figure 4. Check Plot

VIII. CONCLUSION

A design environment for the automatic generation of microcontrollers in single chip telecommunications applications has been developed. When the input parameters are supplied, the generator synthesizes the mask geometries for the required microcontroller. By analyzing the instructions used in a particular program, layouts of major components such as the ALU and the barrel shifter are defined. In this manner, a compact layout is produced for each different microcontroller instance. Several microcontrollers with data-paths ranging from 2 to 16 bits have been simulated. The corresponding machine cycle times are predicted to be 180 to 600 nanoseconds.

REFERENCES

1. Buric, M.R., C. Chistensen, and T. G. Matheson
"The Plex Project: VLSI Layouts of Microcomputers Generated by a Computer Program", Proceedings of IEEE International Conference on Computer Aided Design, Sept. 1983, pp. 49-50.

2. Beckett, W.
"Coordinate Free LAP Reference Manual -- Version 3.0", UW/NW VLSI Consortium Technical Manual.

3. Koenemann, B., J. Mucha and G. Zwiehoff
"Built-in Logic Block Observation Techniques", Digest 1979 Test Conference, 79CH1509-9C, October 1979, pp. 37-41.

4. Buric, M. R. and T. G. Matheson
"Silicon Compilation Environments", Proceedings IEEE Custom Integrated Circuits Conference, May 1985, pp. 208-12.

5. Katavenis, M. G. H.
"Reduced Instruction Set Computer Architectures for VLSI", Ph.D. Thesis, University of California at Berkeley, Oct. 1983.

6. Jinks, B. W., W. S. Snyder and D. L. Pulfrey,
"The Algorithic Generation of ROM Macrocells", Proceedings of the Second International Symposium on VLSI Technology, Systems and Applications, May 1985.

7. Goncalves, N. F. and H. J. DeMan,
"NORA: A Racefree Dynamic CMOS Technology for Pipelined Logic Structures", IEEE Journal of Solid State Circuits, Vol. SC-18, No. 3, June 1983, pp.261-266.

I

7.3 The Quarter Horse: A Case Study in Rapid Prototyping of a 32 bit Microprocessor Chip

S. Ho, B. Jinks, T. Knight, J. Schaad, L. Snyder, A. Tyagi, C. Yang

University of Washington, Computer Science Department, FR-35, Seattle, WA 98195

ABSTRACT

The Quarter Horse is a single chip 32-bit microprocessor whose design and implementation in custom CMOS was completed in 90 days. The design effort is presented as a case study in managing *choice complexity*. The factors contributing to the rapid development of a prototype are discussed, as is the processor's architecture.

Our minds are finite, and yet even in these circumstances of finitude we are surrounded by possibilities that are infinite.
— Alfred North Whitehead

We must never make experiments to confirm our ideas, but simply to control them.
— Claude Bernard

INTRODUCTION

The difficulties of implementing large VLSI designs derive from two quite different sources which can be named: mass complexity and choice complexity. Mass complexity refers to the difficulty of specifying the detail in a large VLSI design *when it is known what is to be done*; this type of complexity is primarily a consequence of the sheer quantity of information. (Examples will be given shortly.) Choice complexity refers to the difficulty of selecting among alternatives within the design space *to determine how it is to be done*; this type of complexity is due to the complicated interactions of the components and the interdependence of design decisions. To complete a design is to triumph over mass complexity; to complete a *successful* design is to triumph over both types of complexity.

There are two purposes in distinguishing between mass and choice complexity. First, the distinction helps to clarify what design problems can and can not be solved by CAD tools and VLSI design methodologies. For example, the hierarchical design methodology is directed purely at managing mass complexity: The hundreds of thousands of polygons that make up a layout are organized into a few composite cells and leaf cells. Design rule and other checking facilities also aid in handling mass complexity since most designers can produce correct primitive cells but introduce design rule errors in cell composition or design revision. Simulation tools, on the otherhand, help confront choice complexity by giving designers the ability to evaluate design alternatives in software rather than hardware. Methodologies such as NORA (No Race logic[1]) constrain the design and simplify dynamic behavior, thereby reducing choice complexity. Other tools and methodologies can be similarly classified and it is interesting to assess how they carve away at the overall complexity problem.

The second reason to separate mass from choice complexity is to study ways of handling choice complexity, the more difficult and less well understood of the two. The problem is that simulators, the principal computer aid for coping with choice complexity, quickly become inadequate. The computational load of simulation becomes too large to be feasible; the simple, approximate models that work well on small designs fail to be accurate when scaled up; the space of possibilities becomes too large to be explored fully as the number of interacting and interdependent parts increases. The difficulties of choice complexity increase nonlinearly with the size of the design, causing it to be the more significant limitation.

Clearly, the easiest way to handle choice complexity is through *experience*, since making design decisions is largely a matter of "knowing from experience" the consequences of choosing various alternatives. But it is not possible to start a design and make every decision correctly from experience because were it so, every decision would have been previously made and the design would be essentially a copy of a previous effort — no *design* would be required. So, design decisions must be made when there is no direct experience to guide the designer. He must therefore find ways of acquiring the relevant experience. This paper focuses on the acquisition and application of experience — the management of choice complexity.

The Quarter Horse, a 32-bit microprocessor chip, designed start-to-finish in 90 days, will serve as a case study for discussing choice complexity management. The rapid 90-day design time is itself a means of quickly acquiring experience, but there are many other more fundamental instances within the design where choice complexity was reduced by bringing our experience or other people's experience to bear on the problem. Most, but not all, of these strategies are applicable to other large VLSI designs.

†Funded in part by the Defense Advanced Research Projects Agency, Contract MDA903-85-K-0072, and in part by the University of Washington / Northwest VLSI Consortium.

The organization of the paper is to give an architectural description of the Quarter Horse chip together with an inline discussion of choice complexity. In the Summary section, we will extract some general principles that can be applied to other designs.

THE QUARTER HORSE ARCHITECTURE

In this section we present the Quarter Horse architecture. Concurrently, we identify instances of choice complexity and describe ways that it may be reduced.

Project Objectives

Perhaps the most significant way in which choice complexity was controlled in the Quarter Horse chip was by selecting an achievable, intermediate goal. Rather than building a high performance microprocessor as our first objective, we chose a straightforward, unadorned machine ***which was intended to be redesigned.*** This approach, which exploits the tendency of second implementations to be cleaner, more efficient, more stable, etc., provided many opportunities for simplification.

First, decisions did not *have* to be right the first time, since they could be changed later. We could forge ahead on the likelihood, rather than the certainty, that a decision was correct and thus save exploration time. Second, by ignoring in the first design the features that would improve the performance of the processor (pipelining, caching, or whatever), we saved work in the short run, thus enabling the chip to be completed sooner, and giving us quicker feedback on our decisions. Third, solutions to certain problems could be delayed until the information needed to solve them was available. Global floor planning is such a problem. Until they are stable and the inter-relationships are fixed, there is no point in packing the components together tightly. We placed the components with generous space separating them in order to provide for last minute changes in size; see Figure 1. To do this within the confines of our die size we used the real estate that would have been assigned to the unimplemented performance features.

Therefore, the intermediate goal of the Quarter Horse effort was to build a basic 32-bit microprocessor that could be enhanced to be a high performance machine in a follow-on design. Since it was impossible to predict how the follow-on design would affect the Quarter Horse, the processor could be self contained if it had flexibility and simplicity. We were designing for experience, but if parts of the design were good enough to keep in later designs, that would be an added benefit.

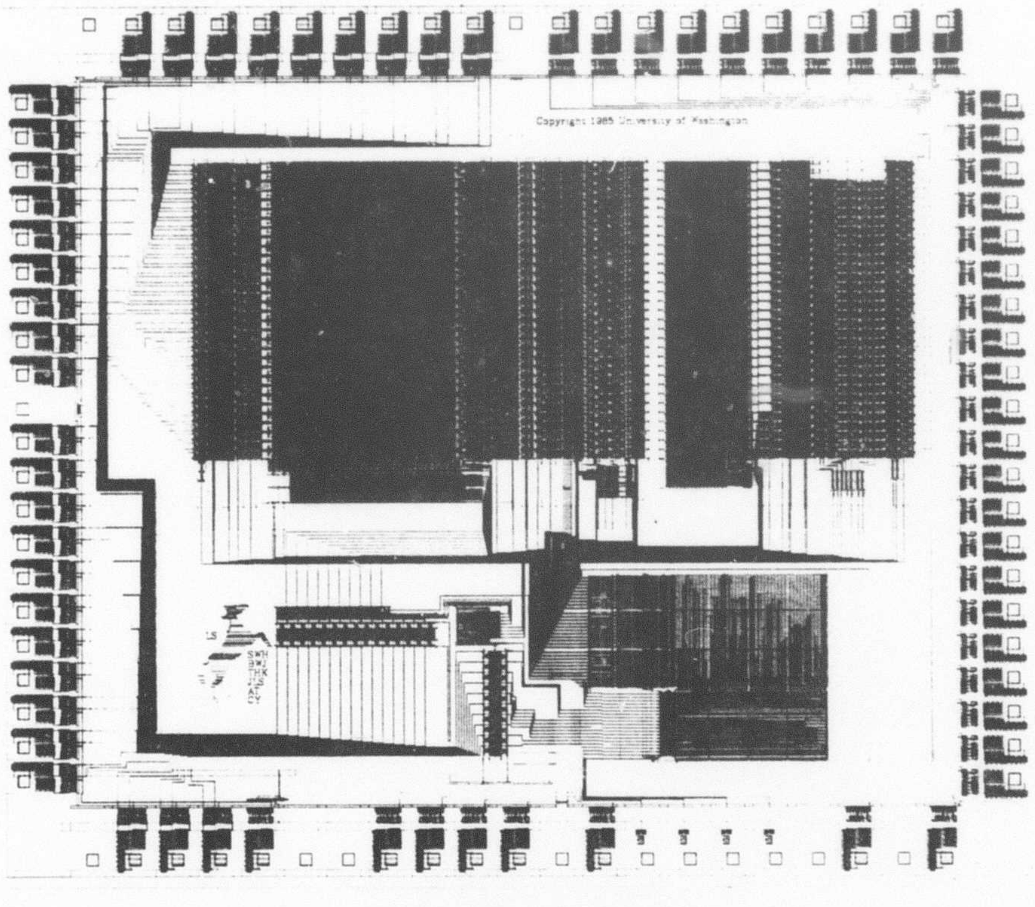

Figure 1. Quarter Horse check plot.

<u>The Architecture</u>

There is a long (by the standards of the IC world) tradition of single chip microprocessors, so one way to reduce choice complexity is to learn from other project's successes and failures. The resulting architecture (see Figure 2) benefitted greatly from this experience. (Explicit credit is given below.)

The processor is built around two 32-bit wide data path busses with a control PLA and a one-instruction prefetch unit. The data path is connected at one end to a 32-bit address port and at the other end to a 32-bit data port.

It is interesting to note that although we advocate aggressive exploration of the design space using every means possible, there are a few choices that just cannot be made easily. These decisions tend to be global and one example is the use of the chip's pins: There is the address port/data port (ap/dp) choice versus the instruction port/value port (ip/vp) choice. In the ap/dp choice, all addresses go through the same pins while the data port must be shared between instructions and values. In the ip/vp choice the pins must be shared between addresses and data. The consequences of the decision affect the internals of the chip significantly and the interface of the microprocessor with the memory system. Although the ip/vp choice was never selected in the microprocessors we studied, our evaluation was that both schemes had assets and liabilities. Our final decision was to choose ap/dp more or less arbitrarily; perhaps it would be worth another 90 days to explore fully the ip/vp choice.

The Quarter Horse has a limited number of instructions like the RISC architecture[2] although we allocated a full 32 bits for the instruction format as was used in the PP4 [3] in order to have flexibility for later expansion. The instruction format is:

```
<------------------ 32 ------------------------->
|op code |SCC|IMM| D | S1 | S2 |      Immediate |
    8      1   1   5   5    5
                            <----------12--------->
```

where the abbreviations are:

SCC	Set condition code
IMM	Immediate data flag
D	Result destination register
S1	Source data register 1
S2	Source data register 2

When immediate values are used S2 is not available.

The architecture uses the register-to-register approach as was used in RISC. Unlike RISC, however, instructions employ a variable number of microcycles which are specified by the controlling PLA. (The purpose and benefits of this choice are explained below.) As a result, instructions can be implemented whose complexity is too great to be completed in a rigid, fixed length fetch/execute cycle. We chose to illustrate this property with instructions using character type data.

The Quarter Horse employs word addressing, as was used in MIPS [4]. Furthermore, we limited the memory space to 2^{28} words, which is quite adequate for our experimental purposes and enables interrupt addresses to be stored in 28 bits. This modest limitation and the fact that the four flag bits fit in the unused positions permit a one word program status word.

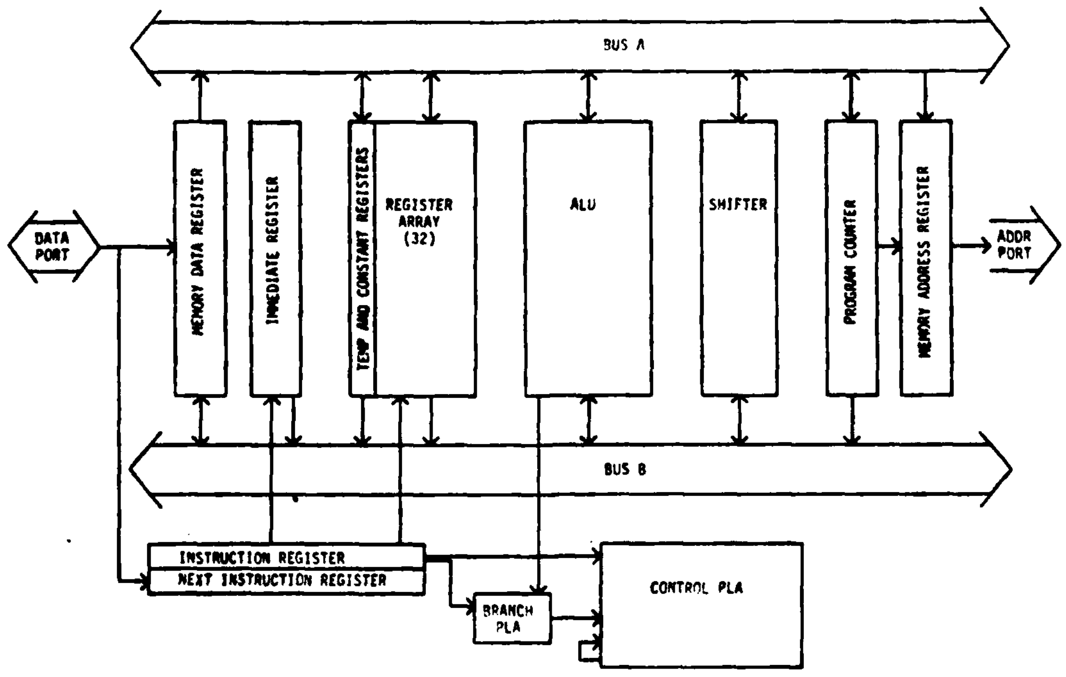

Figure 2. Quarter Horse architecture.

Architectural Components

We briefly discuss each of the components of the architecture with a concurrent discussion on ways of reducing choice complexity.

Register Array The register array has two distinct parts, both of which are built from a basic dual-port static RAM cell: a 32×32 general purpose register set and working register set of two temporary registers and a constant (255) register. The temporary registers and constant are used chiefly for the character operations. The 32 general purpose registers provide considerable flexibility for the software designers.

Design of RAM cells has been a particularly knotty problem for other microprocessor projects. Although the Quarter Horse RAM cell was specially designed, it was based on a dual-ported cell that was produced from a VLSI circuit design generator developed at the University of Washington†. This experience illustrates two ways in which choice complexity can be reduced.

First, the structures like register arrays should be automatically produced using VLSI design generators just as PLAs are automatically generated now. The generator program is an encoding of the "experience" of the generator writer packaged in an extremely usable form. But not every architectural structure can be anticipated, so there may not be a design generator to solve a particular problem. Thus the second, and perhaps the most time-honored means of utilizing other designer's experience, is to modify an existing design. When the existing design is "close" to what is needed this is an extremely effective technique to reduced choice complexity. But when it is not "close" the existing design can possibly be a distraction preventing exploration of rich areas of the design space.

Arithmetic-Logic Unit The ALU of the Quarter Horse is similar to that of the OM2 data path chip designed at CalTech [5]. The chief differences are that it was implemented in CMOS and like the Mosaic design[6], the "R function block" was replaced with an XOR gate.

The way in which choice complexity was controlled in the ALU was by utilizing the best features of designs produced by eight different designers. Students had been asked as a homework assignment in an introductory VLSI class to produce a CMOS version of the OM2 ALU. Eight completed designs were compared and the best parts from these were assembled into the Quarter Horse ALU.

Because the ALU could be produced quickly, it was possible to fix almost at the beginning our 82μ data path pitch. This enabled subsequent design activities to progress with confidence that at least one characteristic of the design would not change.

Shifter The barrel shifter, deemed to be a necessity because of our interest in data types requiring field extraction, was designed by beginning with published approaches [7,8]. The first design was generalized on a second pass to incorporate rotation. The use of a second layer metal CMOS process permitted a significant speed improvement due to reduced capacitance.

†The University of Washington/Northwest VLSI Consortium is presently engaged in a project to study and build design generators - flexible programs that produce circuits for standard architectural components. The RAM design generator was not mature enough to be used for the Quarter Horse directly. The only generators used were the pad frame generator and the PLA generator.

Instruction Register, PC and Memory Registers The instruction register is a pair of registers to support instruction prefetch. These, plus the program counter, memory address and data registers, bus drivers, immediate and sign extension logic were all designed from scratch.

There is little that can be said about the management of choice complexity here except that we tried when possible to use library cells like flip-flops. In the end the designs were mostly original thinking.

The Control PLA Having been impressed by the flexibility provided by the single control PLA of the Mosaic design[6] and a related scheme of the PP4[3], we decided to adopt it rather than embrace the control precepts of other microprocessor chips. The use of the PLA would simplify the addition of new instructions at a later date and it would be the easiest way to control complex instructions requiring many microcycles. At the same time we knew that its performance would be the limiting factor in the speed of the microprocessor. This tradeoff between the costs of performance and the benefits of flexibility were a continual subject of discussion and experimentation. Although we have already realized many of the flexibility benefits, we have not yet determined what the total cost will be. But let us explain.

In terms of choice complexity, what is crucial about the use of the PLA as the single central controlling device of an architecture is that *it delays the binding time of critical architectural decisions.* A principle of computing is to delay binding decisions for as long as possible simply to retain flexibility, and because a PLA is programmable — a complete revision can be produced in an hour — changes can be made trivially right up to the end. The importance of the delayed binding principle for microprocessor architecture design was dramatically illustrated to us when three days before completion of the chip we had to change the general register read protocol! Moreover, as the simulations provided us with information about the performance of the architecture's components, we could continually revise the microcontrol. The conclusion has to be that the architecture itself can aid in controlling choice complexity. But at what cost?

The performance of the PLA concerned us from the very start and we simulated "typical" PLAs to get estimates of performance before adopting the architectural strategy. (The actual, as opposed to the psychological, effect on our work was minimal since we were going to redesign anyhow and the PLA could then be implemented in faster, random logic if it turned out to be too slow.) We took an aggressive 50ns cycle time as a goal and spent a lot of effort trying to achieve it. We experimented with a variety of design styles, domino, NORA, pseudo nMOS, etc., in order to find high performance solutions. We also enumerated a variety of ways in which a slow PLA could be made faster, but none was actually implemented. Most of this activity was to build confidence that the PLA-on-the-critical-path decision was correct. The Quarter Horse chip that was fabricated did not (by simulation evidence) meet the 50ns PLA requirements but by then the flexibility benefits alone justified the decision and slow performance was of less concern. (Measurements on the actual chip were not available at publication time, the simulated time for the PLA was 70ns-80ns.)

There was one other way in which the control PLA reduced choice complexity: By using the PLA for *all* timing, there was no interdependence among the architectural components on clock characteristics, which promoted more independence among the parts. If a single global clock had been used for each component, then there would have had to have been agreement among the designers on such things as the duration of each phase, even with a PLA control. With the PLA doing all of the timing, short duration activities like precharging can be done in one step while ALU computation can be given several cycles, i.e. the "logical clock" used by the component can be asserted for several "physical clock" periods. The lesson is to keep the clock out of the component designs.

Tools

The Quarter Horse chip was designed using Release 2.1 of the University of Washington/Northwest VLSI Consortium Design Tools[9]. Our chief tool for handling choice complexity was the RNL simulator of Chris Terman[10] which was applied to all component designs and to the entire chip. This tool, revised for CMOS and reasonably well calibrated to the MOSIS[11] processes, was an effective way to explore the design space quickly. For certain cells, e.g. RAM, we used SPICE simulation.

While designing the architecture, we built an interactive simulator to allow "register transfer level" simulation of the Quarter Horse. This program, which ultimately produced the microcode for the PLA, was an operational "document" that continually reflected the current state of the high level design decisions. Such tools enable one to try many architectural alternatives and they are worthy of greater exploitation in the future.

The Final Chip

The goal of designing a 32-bit microprocessor was adopted on January 14, 1985 and the Quarter Horse was queued for fabrication on April 15, 1985. It is composed of approximately 25,000 transistors and used the two layer metal 3μ p-well bulk CMOS process provided by MOSIS [11]. (The chip fabrication was not complete at publication time.) A complete description of the architecture can be found in The Architecture of the Quarter Horse Microprocessor [12].

SUMMARY

Multiproject chips have made possible a reduction in the time required to fabricate a prototype chip to 1-3 months. The time required to design a prototype should be similarly brief. The Quarter Horse effort demonstrates that attention paid to choice complexity management permits substantial prototypes to be designed rapidly.

Although the Quarter Horse is not the last word, and there is much still to be studied about choice complexity management, it is useful to recapitulate the points cited above. First,

- **try a throw-away design as an intermediate goal.**

It is an effective way to acquire experience and streamline the effort. Second,

- **read the literature and avoid needless reinvention.**

No matter how inventive or creative a project is, it contains aspects that have been done before. Third, when possible

- **use design generators**

to avoid designing entirely, but if that is impossible, find something close and rework it.

- **Revise existing designs,**

and if several designers can be assigned to a critical part to do independent solutions,

- **merge the best of separate design efforts.**

The sixth rule may not always be applicable, but it was so important to the Quarter Horse, it is worth seeking cases to apply it:

- **Employ a flexible architecture that delays binding**

and plan to implement it with a generator tool such as a PLA. Finally,

- **build tools to aid exploration**

is a rule that will take the form of various simulators such as our architectural simulator.

ACKNOWLEDGMENTS:

We are indebted to the staff of the UW/NW VLSI Consortium for their help throughout the design: Without their direct technical and moral support, the Quarter Horse would not have made it to the finish line.

REFERENCES:

1. Nelson F. Goncalves and Hugo J. DeMan, *Nora: A Race-free Dynamic CMOS Technique for Pipelined Logic Structures*, IEEE Journal of Solid-State Circuits, SC-18(3): 261-266, (June, 1983).

2. Manaolis G. H. Katevenis, *Reduced Instruction Set Computer Architectures for VLSI*, Ph. D. Thesis, Univ. of California at Berkeley, October 1983.

3. W. D. Moeller and G. Sandweg, *The Peripheral Processor PP4: A Highly Regular VLSI Processor*, Proceedings of 11th International Symposium on Computer Architecture, pp. 312-318 (June 1984).

4. J. Hennessey, N. Jouppi, S. Przybylski, C. Rowen and T. Gross, *Design of a High Performance VLSI Processor*, Proceedings of 3rd CalTech Conference on VLSI, California Institute of Technology, Pasadena, California, pp. 33-54, March 1983.

5. Carver Mead and Lynn Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1980, Chapter 5.

6. Christopher Lutz, *Design of The Mosaic Processor*, Masters Thesis, California Institute of Technology, May, 1984.

7. Robert W. Sherburne Jr., Manolis G. H. Katevenis, David A. Patterson, and Carlo H. Séquin, *Datapath Design For RISC*, Proc. of 1982 Conf. on Advanced Research in VLSI, MIT pp. 53-62.

8. John A. Bayliss, Stephen R. Colley, Roy H. Kravitz, Gary A. McCormic, William R. Richardson, Doran K. Wilde, Leon L. Wittmer, *The Instruction Decoding Unit for the VLSI 432 General Data Processor*, IEEE Journal of Solid-State Circuits, Vol. SC-16(5): pp. 531-536, (October 1984).

9. UW/NW VLSI Consortium, *Design Tools Release 2.1*, University of Washington, 1984.

10. Christopher J. Terman, *User's Guide to NET, PRESIM, and RNL/NL*, Technical Report, Massachusetts Institute of Technology, (September 1982).

11. Danny Cohen and George Lewicki, *MOSIS - The ARPA Silicon Broker*, Proceedings of the CalTech Conference on VLSI, California Institute of Technology, Jan. 1981.

12. S. Ho, B. Jinks, T. Knight, J. Schaad, L. Snyder, A. Tyagi, and C. Yang, *The Architecture of the Quarter Horse Microprocessor*, Technical Report, University of Washington, 1985.