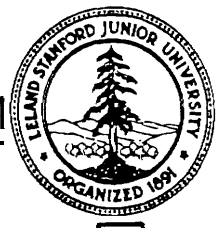
COMPUTER SYSTEMS LABORATORY

STANFORDELECTRONICS LABORATORIES
DEPARTMENT OF ELECTRICAL ENGINEERING
STANFORD UNIVERSITY · STANFORD, CA 94305



DESIGN AUTOMATION AT STANFORD II

Edited by

W.M. vanCleemput

TECHNICAL REPORT NO. 184

February 1780

DESIGN AUTOMATION AT STANFORD II

Edited by

W. M. vanCleemput

TECHNICAL REPORT NO. 184

February 1980

COMPUTER SYSTEMS LABORATORY
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California 94305

DESIGN AUTOMATION AT STANFORD II

An overview of Design Automation at Stanford University

Edited by

W. M. vanCleemput

TECHNICAL REPORT NO. 184

COMPUTER SYSTEMS LABORATORY

Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California 94305

ABSTRACT

This report contains a copy of the visual aids used by the authors during the presentation of their work at the Second Workshop on Design Automation at Stanford, held on February 19, 1980.

The topics covered range from circuit level simulation and integrated circuit process modelling to high level languages and design techniques. The presentations are a survey of the activities in design automation at Stanford University.

TABLE OF CONTENTS

LANGUAGE AND ENVIRONMFNT

FOR

MULTI - LEVEL SIMULATION

by

Dwight D. Hill

ABSTRACT

A methodology is proposed for modeling and simulating computer systems. It makes use of a new language called ADLIB for specifying the behavior of computer subsystems, and a special environment, SABLE, for modeling the way that they interact. The ADLIB language is designed to be compatible with existing computer languages, since it is a proper superset of PASCAL. The union of behavioral and structural design specifications makes it possible to apply type checking to hardware design. Several examples illustrate the description and simulation of systems ranging from distributed computer networks to individual logic gates.

CURRENT TECHNIQUES (PART I!:

MANY LANGUAGES FOR DESCRIPTION, SIMULATION OF A SINGLE TARGET SYSTEM

--> OVERHEAD, ERRORS

HIGH LEVEL LANGUAGES (GPSS,SIMULA),
INTERMEDIATE LEVEL (ISPS),
REGISTER TRANSFER (DDL,CDL),
GATE LEVEL (D-LASAR)
CIRCUIT LEVEL (MSINC,SPICE)

--> EACH USED INDEPENDENTLY, AND ARE INCOMPATIBLE

HIGH ABSTRACTION MODELS

--> POOR RESOLUTION

HIGH DETAIL MODELS

--> POOR EFFICIENCY

CURRENT TECHNIQUES SPECIFICATION & SIMULATION

CURRENT TECHNIQUES (PART 2):

- CURRENT LANGUAGES HAVE CONSTRAINTS LIMIT "DESIGN SPACE"

EITHER:

BEHAVIOR OR STRUCTURE SPECIFICATION

SYNCHRONOUS OR ASYNCHRONOUS TIMING, OR NO TIMING SUPPORT AT ALL

PROCEDURAL OR NON-PROCEDURAL CONTROL

REGISTERS, BITS, OR MULTI-VALUES

HARDWARE OR SOFTWARE IMPLEMENTATIONS

CONSTRAINTS IN CURRENT CAD LANGUAGES

OBJECTIVES:

- CONSISTENT DESIGN SPECIFICATION FORMAT FROM START THROUGH COMPLETION
- STRUCTURE AND BEHAVIOR CAPTURED
- EFFECTIVE SIMULATION
  - LITTLE ADDITIONAL EFFORT
  - WORK DIRECTLY FROM DESIGN SPECS.
  - SIMULATE EARLY
  - ABSTRACT AND REFINED DESIGNS COMPATIBLE
  - SIMULTANEOUSLY SIMULATE MULTIPLE LEVELS

OBJECTIVES: MAKE CAD MORE USEFUL

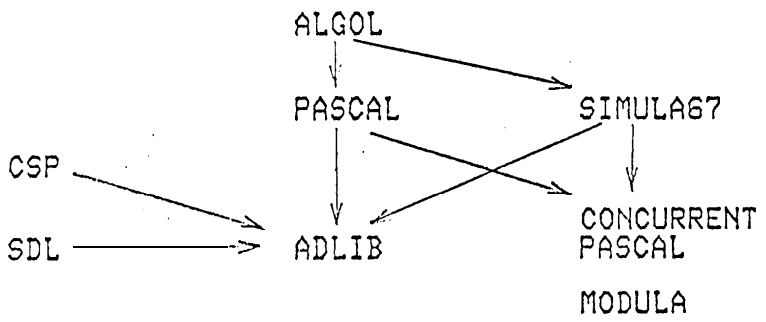
ADLIB ----> SABLE <--- SDL

ADLIB = A DESIGN LANGUAGE FOR INDICATING BEHAVIOR

SABLE = STRUCTURE AND BEHAVIOR LINKING ENVIRONMENT

SDL = STRUCTURAL DESIGN LANGUAGE

"ADLIB," "SABLE," AND "SDL"



ORIGINS OF ADLIB

ADLIB = PASCAL +

CONCURRENT PROCESSES

TIMING

INTERPROCESS LINKS (NETS)

ADLIB MUST BE USED WITH A

STRUCTURE SPECIFICATION SYSTEM

WHAT IS ADLIB?

```
    CASE OPCODE OF ! DECODE INSTRUCTION AND EXECUTE
       XAND: ACC := RGAND(ACC,M[IZ]);
       TAD:  BEGIN
               ACC:= RGADD(ACC,M[IZ]);
               IF RGCARRY <> LINK THEN LINK := 1
                       ELSE LINK :=0;
             END;
       ISZ:  BEGIN
               WAITFOR TRUE DELAY 1;
               INCR(M[IZ]);
               IF M[IZ]=0 THEN INCR(PC);
             END;
       DCA:  BEGIN M[IZ]:= ACC; ACC := 0 END;
       JMS:  BEGIN M[IZ]:= PC; PC := IZ+1 END;
       JMP: PC:= Z;
       IOT:  IF TRACE THEN DUMP('       IOT',[DCACC]);
       ENCODED : ! NEXT SLIDE
   END;  (* CASE OPCODE *)
```

PDP8 SLIDE 2 - INSTRUCTION DECODE

```
       "WAITFOR" CONSTRUCT ALLOWS A COMPONENT
        TO PAUSE UNTIL SOME CONDITION IS MET

       EXAMPLES :

       WAITFOR BUS.ADDR = MY_ADDR CHECK BUS;

       WAITFOR MANTISSA <> 0 SYNC PIPE_CLK;

       WAITFOR DELAY 15.0E-9;
```

LANGUAGE FEATURES : HOLDING A PROCESS

COMPONENT ONE -> -> -> -> -> -> -> NET -> COMPONENT TWO

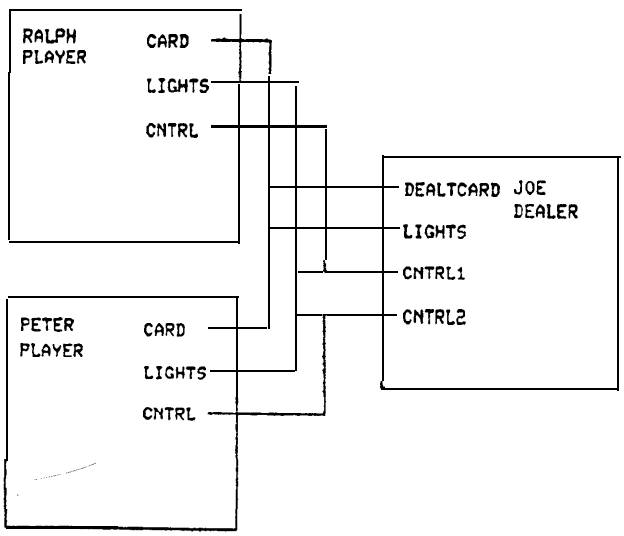
NETS ARE USED FOR BOTH DATA AND CONTROL

USER SPECIFIES REAL PROPAGATION DELAY

REDUNDANT UPDATES AUTOMATICALLY DELETED

NON REDUNDANT UPDATES STIMULATE COMPONENTS

FUNDAMENTAL PRINCIPLES OF ADLIB/SABLE



STRUCTURE OF A BLACKJACK GAME

SUBPROCESSES

RUN INDEPENDENTLY OF MAIN BODY OF COMPONENT, BUT CAN BE ENABLED OR DISABLED AS NEEDED

UNBURDEN MAIN CONTROL LOGIC OF COMPONENT

CAN BE USED TO DESCRIBE INTERNAL TIMING OF COMPLEX COMPONENTS

LANGUAGE FEATURES : SUBPROCESSES

```
SUBPROCESS

TTY_DRIVER : UPON CHAR_RDY CHECK TTY_LINE DO
    BEGIN
    B_POS := B_POS + 1;
    BUFFER[B_POS] := TTY_LINE.CH;
    IF (B_POS > BUFFER_SIZE) OR
     (TTY_LINE.CH IN [ESC,CR,LF]) THEN
     WAKE_MAIN_PROCESS;
    END;


CHANNEL1: TRANSMIT DISK_BUS.DATA CHECK DISK_BUS
            TO MAIN_MEM DELAY 1.5E-6;
```

LANGUAGE FEATURES: USE OF SUBPROCESSES

```
= COMPTYPE PLAYER;
= INWARD CARD : CARD_BUS;
= OUTWARD LIGHTS : DISPLAY_LIGHTS;
= EXTERNAL CNTRL : CONTROL_LINE;
= VAR SCORE : 0..31; HOLDING_ACE : BOOLEAN;
= BEGIN
=
= WHILE TRUE DO BEGIN
=    SCORE := 0; HOLDING_ACE := FALSE;
=    REPEAT
=         REPEAT
=              ASSIGN HIT TO LIGHTS;
=              WAITFOR CNTRL=CARD_RDY CHECK CNTRL;
=              IF CARD.RANK<JACK THEN
=                SCORE := SCORE + ORD(CARD.RANK) + 1
=              ELSE SCORE := SCORE + 10;
=              IF (CARD.RANK=ACE) AND (NOT HOLDING_ACE ) THEN
=                 BEGIN SCORE:=SCORE + 10; HOLDING_ACE:=TRUE END;
=         UNTIL SCORE>=17;
=          IF (SCORE>21) AND HOLDING_ACE THEN BEGIN
=              SCORE := SCORE - 10; HOLDING_ACE := FALSE END;
=    UNTIL SCORE>=17;
=    IF SCORE<=21 THEN ASSIGN STAND TO LIGHTS
=     ELSE ASSIGN BROKE TO LIGHTS;
=    END; !WHILE TRUE
= END; !COMPTYPE PLAYER
```

ADLIB DESCRIPTION OF BLACKJACK MACHINE

MULTI-LEVEL SIMULATION:

"DATA-LEVEL" OF COMPTYPE DETERMINED BY ITS "NETTYPES"

INTERNAL DETAILS OF CODE ARE IRRELEVANT

NETTYPE MISMATCH -> MULTI-LEVEL SIMULATION

"TRANSLATOR" COMPONENTS MEDIATE MISMATCHED NETS

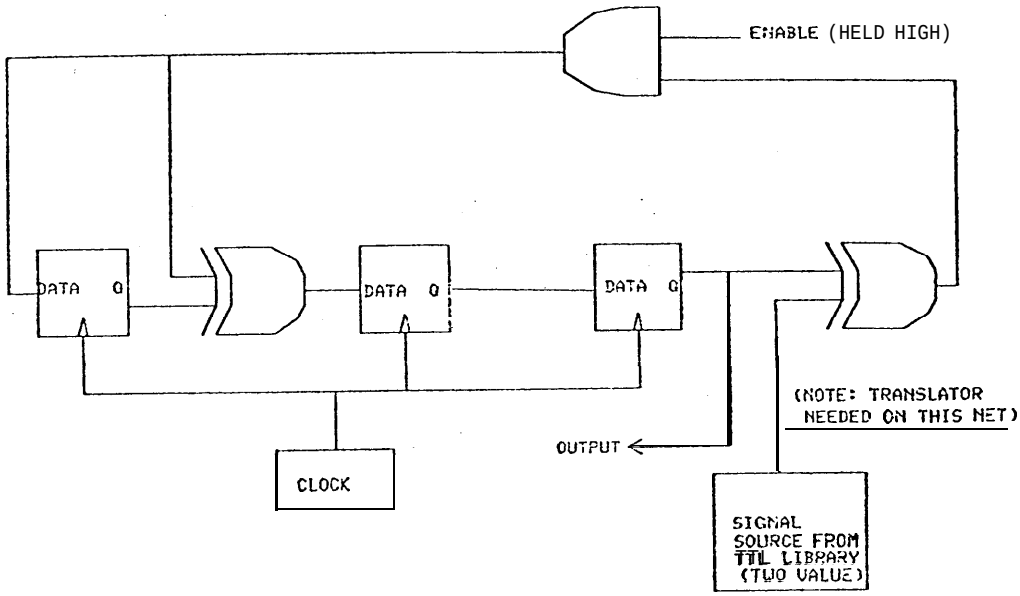
TRANSLATORS MAY BE INSERTED AUTOMATICALLY

EXAMPLES :

BUS SPLIT TO INDIVIDUAL BITS

MULTI-VALUE COMPRESSED TO BOOLEAN

MULTI - LEVEL SIMULATION IN ADLIB/SABLE




```
= @EX SEVEN,HILL:ASSIST                                                =
=  PASCAL/LOTS:  SEVEN [ MULTI_VAL ] PAGE 1..( 500)=
=    (PASCAL COMPILER MESSAGES OMMITTED FOR BREVITY) =
= PLEASE ENTER SHORT AND FULL DELAYS (REAL NUMBERS):=
=  PLEASE ENTER DEFAULT INITIAL VALUE FOR NETS (CR=S )=
= NAME OF TOPOLOGY FILE (WITHOUT EXTENTION) : CRC    =
=  STRRT OF RUN, TIME=0.0, Z9MAXEVENTS=9999,Z9MAXTIME=
=  ENTERING SABLE MONITOR, TYPE '?' FOR HELP           =
=  FOR HELP TYPE "?"                                   =
=  COMMAND: EVENT 100                                  =
=  COMMAND:  TRACE YES                                 =
=  SETTING TRACE TO TRUE                               =
=   COMMAND: RUN                                       =
=   RESTARTING SIMULATION                              =
= SIMULATED EVENT LIMIT REACHED                        =
=  TIME= 7.100000023E+01 NUMBER OF EVENTS=    100      =
=  TIME LIMIT= 9.998999953E+02 EVENT LIMIT=   100      =
=   COMMAND: QUIT                                      =
=     QUITING...                                       =
= END OF SIMULATION, TIME= 7.100000023E+01 NUM EV= 100
=  CPU TIME =     1.194 ELAPSED TIME=    20.318        =
=  NUMBER OF ACTIVATIONS=          224                 =
= EXIT FROM SABLE                                      =
=------------------------------------------------------=
=  RUNNING ADLIB/SABLE (MESSAGES EDITED FOR BREVITY) =
==========================================================
```

```
@ADLIB
 FOR HELP TYPE "?"
*SEVEN.ADL
 FINISHED: NETTYPES... REPORTER   TOMULTI    TOBOOL
MLT_NAND2  MLT_AND2   MLT_OR2    MLT_NXOR2  MLT_NOR2
MLT_XOR2   MLT_DFFLOP MLT_INV    MLT_CAP    PULSE
SIGNAL     MLT_CONST  MLT_PULSE
NO ERRORS DETECTED
EXIT
@SABLE
FOR HELP TYPE "?"
*CRC.SDL,SEVEN/LIST
DATABASE SUCESSSFULLY READ IN
  14 IN = BSOURCE.OUT,XR1.A;
  14                      !
 INSERTING TRANSLATOR TO XR1        A
   8 STUB NET(S) INSERTED
  1 TRANSLATOR(S) INSERTED
 NO ERRORS DETECTED
 EXIT
```

COMPILING ADLIB AND SDL

```
             1 = IN
             2 = CNTRLLER
             3 = DELAY0
             4 = DELAY1
             5 = PROD
             6 = DELAY2
             7 = OUT
             8 = TO_ENAB
             9 = CLK
            10 = *FROM TRAN
            11 = **STUBBY**
                      11111 11111 22222 22222 33333 3333
  TIME   1234 56789 01234 56789 Of234 56783 01234 5678
     0 FHHH HHHHX HXXXX XXF
     0 FHHH LHHLX HXXXX XXF
     0 FHLH LHHLX HXXXX XXF
     0 FHLH HHHLX HXHXX XXF
     0 FHLH HHHLX HXHXX XXF
     5 FHLH HHHLU HXHXX XXF
     5 FHLH HHHLH HXHXX XXF
    10 THLH HHHLD HXHXX XXF
    10 THLH HHHLL UXHXX XXF
```

SEVEN VALUE SIMULATION WITH INTIAL VALUE "H"

IMPACT ON COMPUTER AIDED DESIGN (PART 1):

CONSISTANT FORMAT AT ALL LEVELS

ADLIB -> BEHAVIOR

SDL -> STRUCTURE

UNIFIED IN SABLE ENVIRONMENT

SIMULATION DIRECTLY FROM DESIGN

ELIMINATES TRANSLATION ERRORS

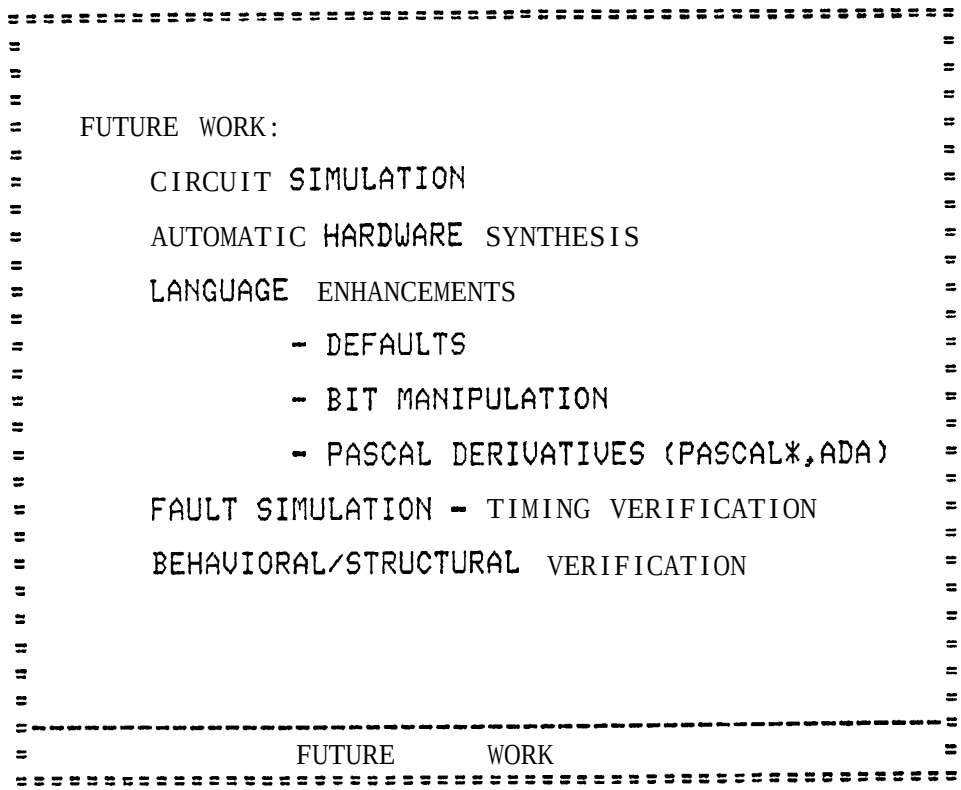
COMPATIBLE WITH OTHER CAD TOOLS

CONCLUSIONS

EXPERIMENTS:

- ARPANET (PERFORMANCE EVALUATION) 490
- PDP 8 (ARCHITECTURE LEVEL & BIT SLICE IMPLEMENTATION) 200 340
- TERMINAL CONCENTRATOR (ALGORITHM TESTED AND GATE LEVEL DESIGN) 230 880
- DATA ACQUISITION SYSTEM 160
- BLACKJACK MACHINE 120
- SEVEN VALUE GATE LIBRARY 430

EXPERIENCE WITH ADLIB


FUTURE WORK:
CIRCUIT SIMULATION
AUTOMATIC HARDWARE SYNTHESIS
LANGUAGE ENHANCEMENTS
- DEFAULTS
- BIT MANIPULATION
- PASCAL DERIVATIVES (PASCAL*,ADA)
FAULT SIMULATION - TIMING VERIFICATION
BEHAVIORAL/STRUCTURAL VERIFICATION
FUTURE WORK

TIMING VERIFICATION

IN

THE SCALD SYSTEM

by

Tom McWilliams

ABSTRACT

A new approach to the verification of the timing constraints on large digital systems has been developed. The algorithm is computationally very efficient and also provides early and continuous feedback about the timing aspects of synchronous sequential circuits as they are designed. It also allows for the design to be conveniently verified in sections, permitting the verification of designs which would otherwise be too large to do on existing computer systems. A system using this algorithm has been implemented, and has been used to verify the timing constraints on the design of the S-1 Mark IIA processor, which consists of 10,000 ECL chips, and is comparable in performance to the Cray-1 CPU.

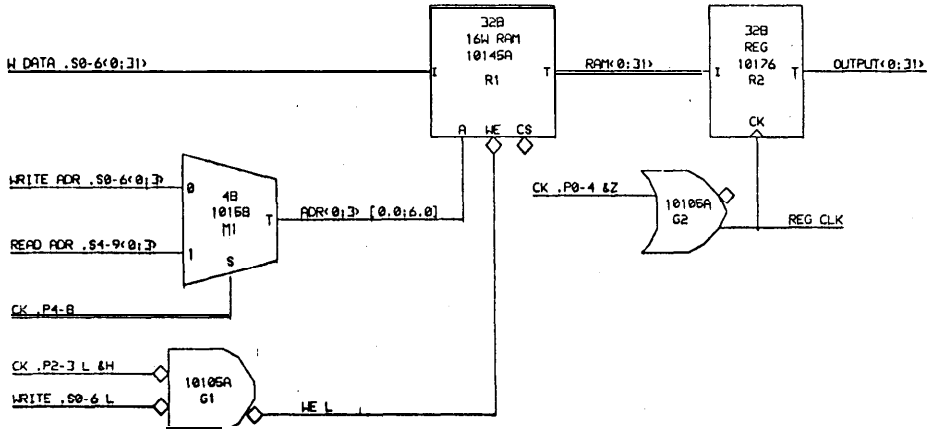
SCALD TIMING VERIFIER GOALS

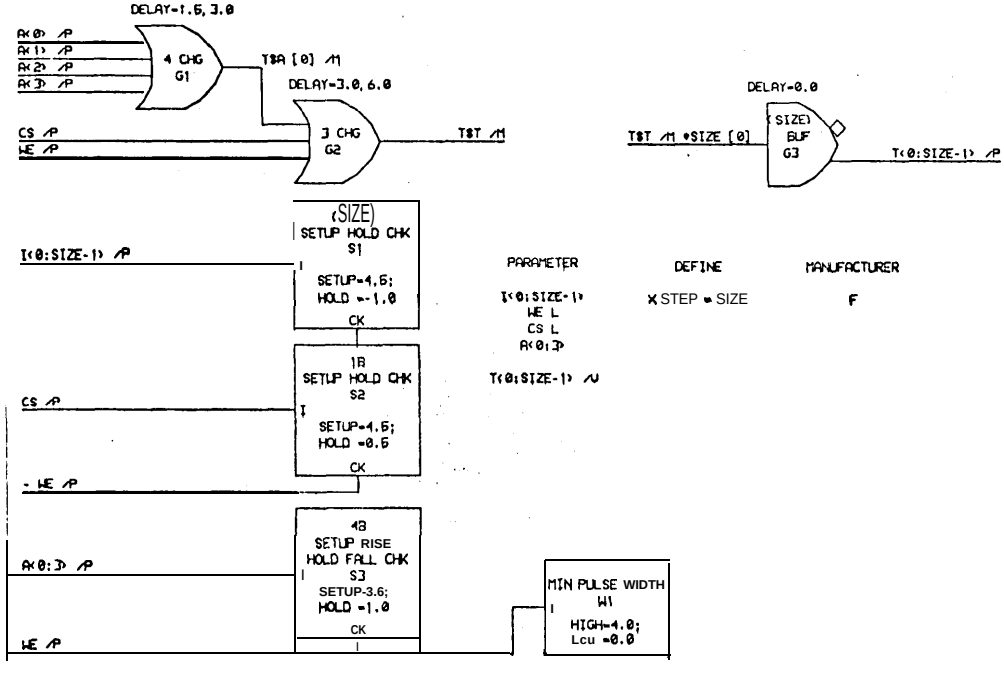
- To verify all timing constraints in large clocked digital systems
- To verify timing constraints early and throughout a design
  - Avoid finding timing errors at the end of the design
  - Automatically provide timing information about part of design already completed, for use in completing design
- To eliminate the necessity to generate complex files to drive the verification (such as is needed in conventional logic simulation)
- To allow additional timing constraints to be specified in the prints
  - For example, the specification of when interface signals can change
- To verify as much as possible of the timing in a "value-independent fashion"
  - In order to minimize the number of cases that need to be tested
    - To reduce CPU time
  - In order to minimize the problem of driving the verification
    - Machine doesn't need to be microcoded to check timing
    - Can verify incomplete designs

TIMING VERIFICATION IN THE SCALD SYSTEM

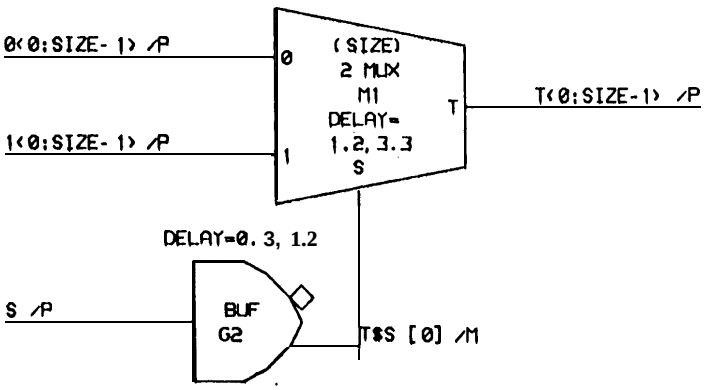
- Checks all timing constraints in large digital systems, taking into account:
  - Component timing properties
    - Propagation delays
    - Setup and hold constraints
    - Minimum pulse width constraints
  - Wire delays
    - User-specified limits
    - Calculated values based on routing, capacitance, and transmission line characteristics
  - Additional designer-specified constraints
- Oriented toward clocked digital systems

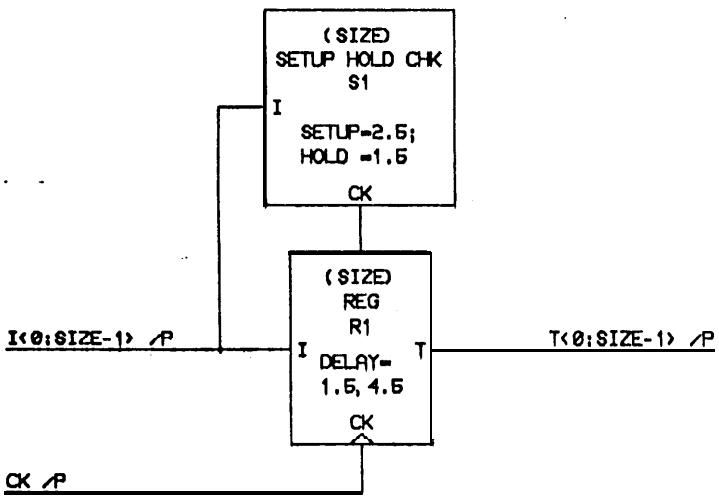
Value	Meaning
0	False
1	True
S	Stable
C	Changing
R	Rising edge
F	Falling edge
U	Undefined (Initial value)





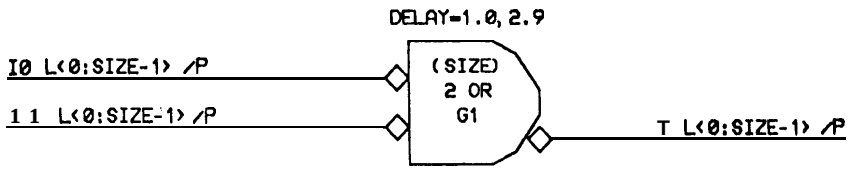
PARAMETER	DEFINE	MANUFACTURER
0<0:SIZE-1> 1<0;SIZE-1> S T<0:SIZE-1>	X STEP = SIZE	MS





PARAMETER	DEFINE	MANUFACTURER
I<0:SIZE-1> CK T<0:SIZE-1> /V	X STEP = SIZE	FMS

PARAMETER	DEFINE	MANUFACTURER
I0 L<0:SIZE-1> 11 L<0:SIZE-1> T L<0:SIZE-1>	X STEP = SIZE	FMS



SETUP AND HOLD ERRORS FOUND

Setup, Hold and Minimum **Pulse Width** errors . . . .

Setup time error; Setup Time = 3.5, Hold Time = 1.0
CK INPUT = WE (+0.0) 0:0.0, R:11.5, 1:15.5, F:17.8, 0:21.8
DATA INPUT = ADR (+0.0) S:0.0, C:0.5, S:11.5, C:25.5, S:36.5

Setup time error; Setup Time = 2.5, Bold Time = 1.5
CK INPUT = REG CLK (+0.0) R:0.0, 1:3.0, F:24.0, 0:28.0, R:49.0
DATA INPUT = RAM (+0.0) S:0.0, C:5.0, S:22.5, C:30.0, S:47.5

SIGNAL VALUE SUMMARY LISTING

Values o f **all signals**

```
ADR<0:3>                S:0.0, C:0.5, S:5.5, C:25.5, S:30.5
CK .P0-4                R:0.0, 1:1.0, F:24.0, 0:26.0, R:49.0  (constant value)
CK .P2-3                0:0.0, R:11.5, 1:13.5, F:17.8, 0:19.8  (constant value)
CK .P4-8                F:0.0, 0:1.0, R:24.0, 1:26.0, F:49.0  (constant value)
OUTPUT<0:31>            S:0.0, C:0.5, S:7.5
RAM<0:31>               S:0.0, C:5.0, S:20.5, C:30.0, S:45.5
READ ADR .S4-9<0:3>     S:0.0, C:6.3, S:25.0
REG CLK                 R:0.0, 1:1.0, F:24.0, 0:26.0, R:49.0
W DATA .S0-6<0:31>      S:0.0, C:37.5
WE                      0:0.0, R:11.5, 1:13.5, F:17.8, 0:19.8
WRITE .S0-6             S:0.0, C:37.5
WRITE ADR .S0-6<0:3>    S:0.0, C:37.5
```

OPERATION OF THE SCALD TIMING VERIFIER

- Does case analysis to handle logic where the values of signals affect timing, and values of signals are not symmetric from cycle to cycle
- For a given case, assumes that the signal behavior is periodic over the cycle time of the circuit
- Evaluates circuit for the first case, and then only re-evaluates those parts of the circuit that change in going from one case to the next

Experience in Using Timing Verifier

- Provides daily feedback about timing errors as the design proceedes
  - Checks design to see that no timing errors have been introduced
    - Uses rule to estimate wire delay initially
    - After layout of boards is done, it uses accurate wire delay predictions based on layout
- Meeting both minimum and maximum delays required significant amount of work
  - Typically two or three timing errors are introduced in a given day of design work
    - With constant feedback, designers learned to make fewer timing errors
      - During initial part of design, many errors would be made during a day's work
    - A number of circuits bad to be entirely redesigned to meet worst case timing constraints
- S-1 Mark IIA processor was verified in two 5,000 chip sections
  - Required 20 minutes of CPU time to verify a given section
    - Executed on S-1 Mark I processor
      - Comparable in performance to 370/168
  - Required 7 to 8 Megabytes of memory

Conclusions

- The Timing Verifier allowed constant feedback to the designer with very little cost
- Use of the Timing Verifier encouraged conventions which greatly improved design readability
- The system resulted in a significant reduction in design time
  - When designing a new section, existing signals can be looked up in a summary listing to see when they are changing
  - Timing errors are found early in the design, before they have a chance to propagate
  - A significant amount of time was saved by not needing to do as many hand calculations while doing the design
- The system allowed a design to be done which executes faster
  - By providing quick feedback about timing, design could be optimized for execution speed more readily

VERIFICATION OF DESIGN CORRECTNESS

WITH ADLIB AND SDL

by

Warren Cory

ABSTRACT

A designer may use Adlib and SDL in a hierarchical fashion to describe a design from its initial phase to the detailed logic design phase. However, the designer must rely on simulation to verify the correctness of each refinement in the design. More formal verification techniques are required.

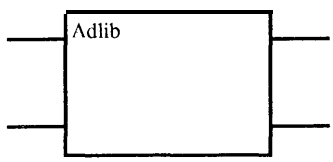
A verification experiment involving a UNIBUS interface design is currently in progress. This experiment will help to evaluate a proposed approach in which the verification problem is partitioned into two simpler sub-problems. This approach is suggested by the similarity of this problem to that considered by IBM in the verification of LCD.

Adlib features of interest for verification

- Components interact only through well-defined interfaces
- Adlib allows description over wide range of levels of abstraction
- Correspondence between levels of abstraction may be defined with translators

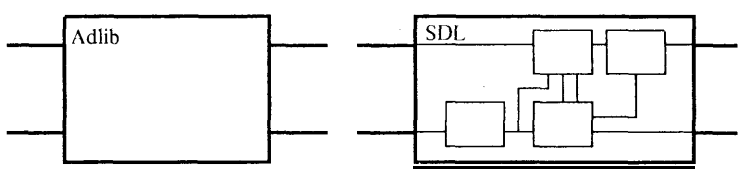
Top-down hierarchical design with Adlib/SDL

(1) Write Adlib description specifying desired behavior of system.



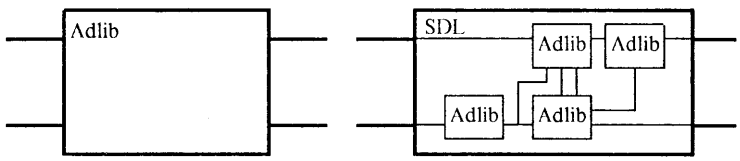
Top-down hierarchical design with Adlib/SDL

(2) When satisfied with this specification, design a structure for implementing this behavior. Describe structure with SDL.



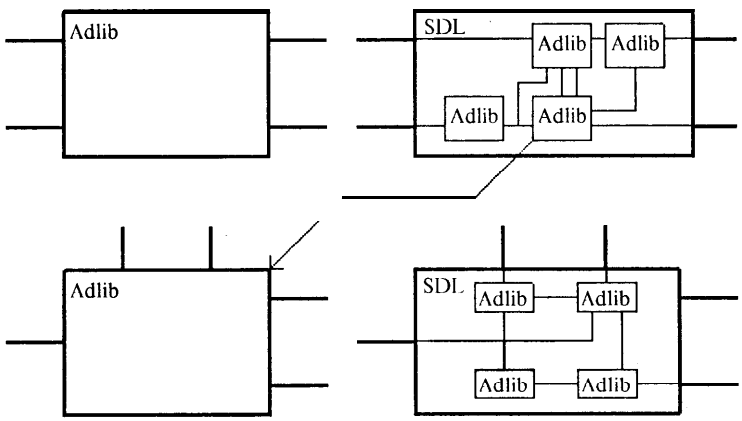
Top-down hierarchical design with Adlib/SDL

(3) Describe behavior of components used in above structure using Adlib. This step corresponds to step (1) at the next lower level in the hierarchy.



Top-down hierarchical design with Adlib/SDL

(4) Repeat to obtain a detailed logic design which uses available physical components or components to be fabricated with other DA tools.



Where does verification come in?

After steps (2) and/or (3).

- Show that structure described in step (2) can support the behavior specified in step (1)

- and/or -

- Show that system described in steps (2) - (3) meets the specifications given in step (1).

Simulation is currently used for this purpose.

This top-down design with Adlib/SDL is similar to design techniques used at IBM with LCD. The major differences between Adlib/SDL and LCD are

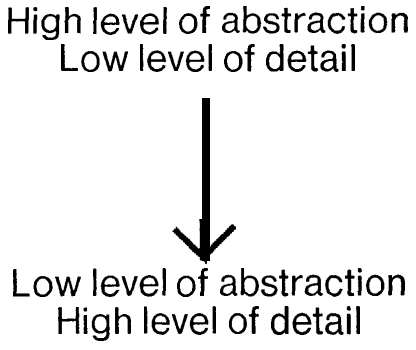
1) LCD may be used only for fully synchronized systems,

2) All LCD descriptions are written at the same level of abstraction (non-procedural RTL), and

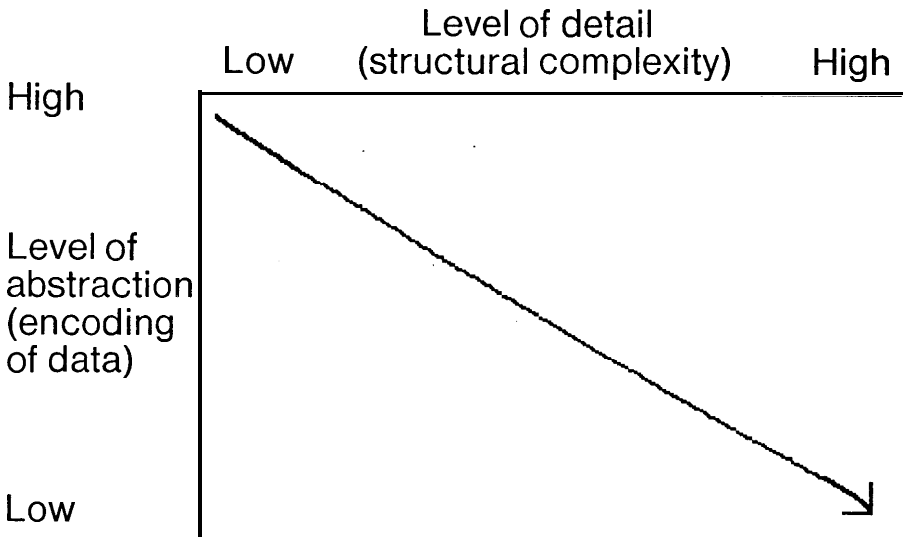
3) In Adlib/SDL, the structure of a system is described explicitly, while in LCD, inter-component connections are implied by the communication of components through global facilities.

IBM has done extensive work on verification with LCD. More on this later...

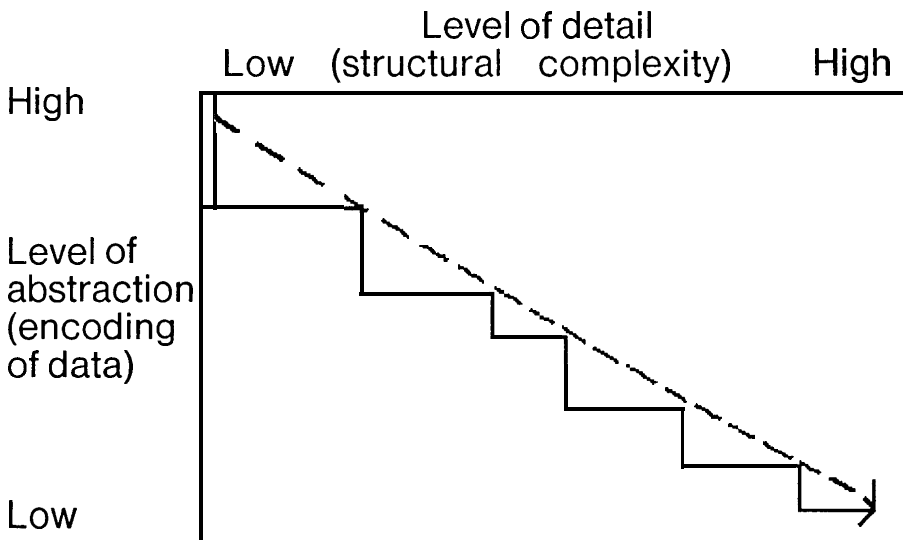
We often view the top-down design process as a one-dimensional refinement.



For verification, it is advantageous to view the design process as a refinement in two dimensions.



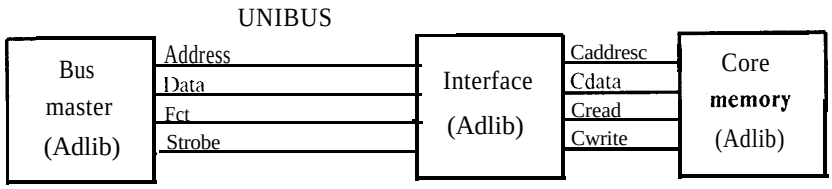
We may now consider transitions to lower levels of abstraction separately from the introduction of more detail.



This partitions the verification problem, as shown in the following example.

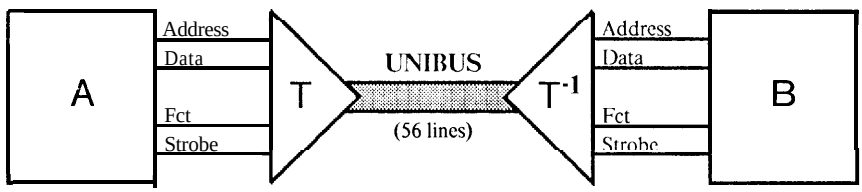
UNIBUS design example: a verification experiment

(1) At high level of abstraction, designer describes interface between UNIBUS and core memory card.



UNIBUS design example: a verification experiment

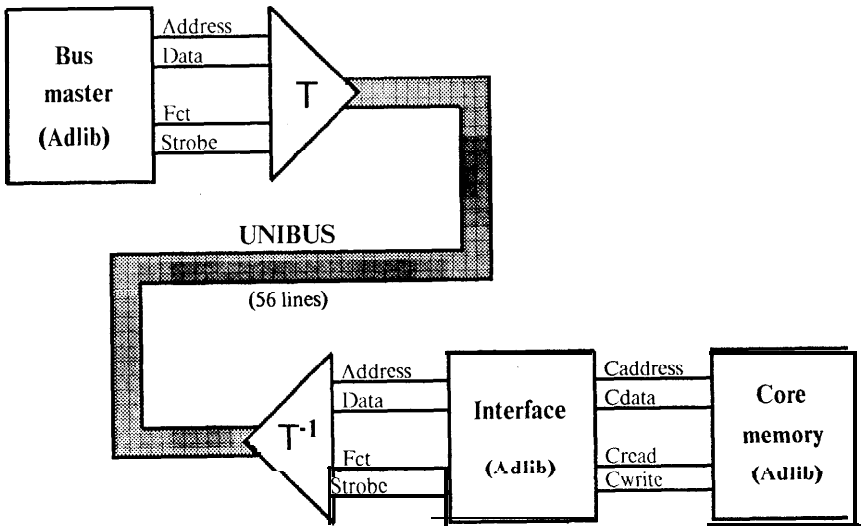
(2) Designer writes translators which define correspondence between crude bus protocol used above and detailed UNIBUS protocol. This is a transition to a lower level of abstraction.



(3) Automatically verify that UNIBUS protocol proposed in step (2) is feasible. That is, show that signals are properly passed between **A** and **B**.

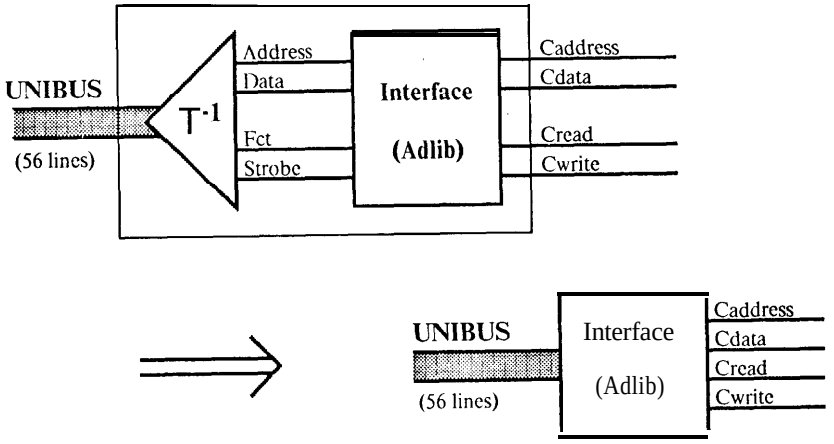
UNIBUS design example: a verification experiment

Model after step (3)



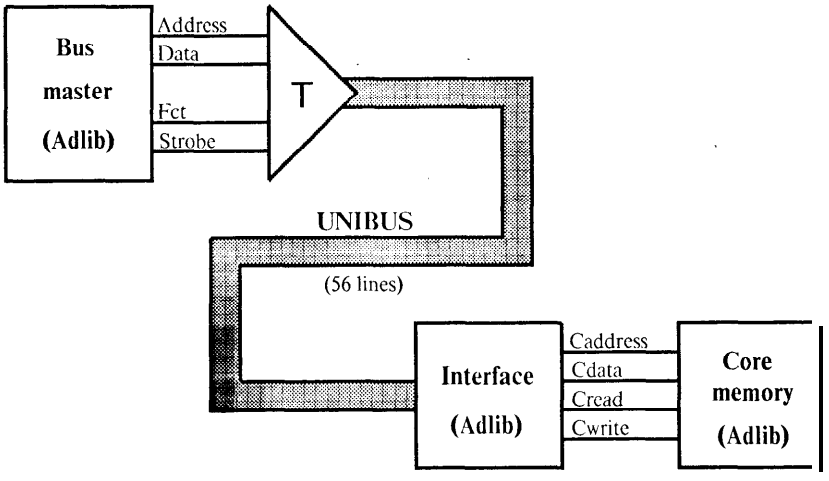
UNIBUS design example: a verification experiment

(4) Merge INTERFACE and T^{-1}, yielding a description of the interface at a lower level of abstraction.



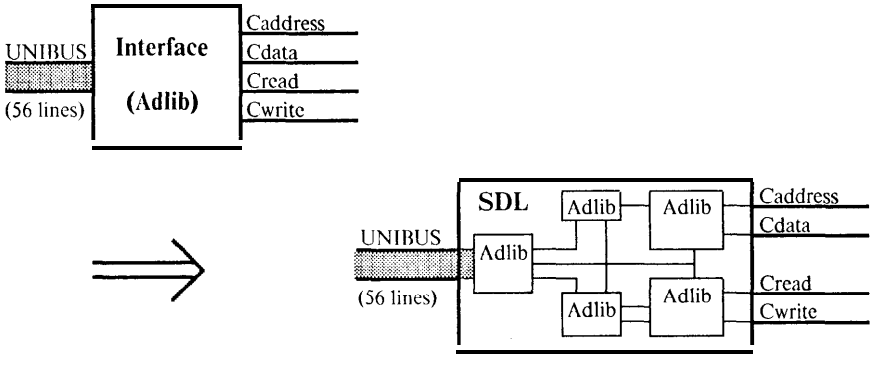
UNIBUS design example: a verification experiment

Model after step (4)



UNIBUS design example: a verification experiment

(5) Use SDL to specify internal structure of INTERFACE. Use Adlib to describe behavior of components used in INTERFACE.



(6) Verify that structural/behavioral description on the right meets behavioral specifications on the left.

UNIBUS design example: a verification experiment

What techniques will be used?

Verification of translators in step (3):
By symbolic simulation, show that proposed protocol properly transmits data.

Comparison of descriptions in step (6):
Symbolically simulate the descriptions in parallel; compare values on nets.

UNIBUS design example: a verification experiment

This last step looks like the original verification problem, but there is an important difference:

The nets in the two descriptions to be compared in step (6) are at the same level of abstraction, and there is a one-to-one correspondence between the nets in the two descriptions.

The techniques used at IBM to verify LCD might now be successfully applied to this simplified problem.

AUTOMATIC SYNTHESIS OF A SYSTEM CONTROLLER FROM DDL-P TO PLA

by

Sungho Kang

ABSTRACT

Direct hardware synthesis from a higher level description of a digital system is one of the ultimate goales of all design automation activities. As an attempt in that direction, a system, which automatically generates the PLA's for the control circuit of a digital machine from a DDL-P description, has been developed. This is a very convenient tool for the design of a finite state machine. Roughly, the control circuit of any digital system for which a state diagram can be drawn can be designed easily using this system.

CONTROL CIRCUITRY in a digital system

1. distributed control
   - random Logic

2. localized control (for a complicated system)
   - random logic
   - PLA

3. centralized control
   - random logic
   - ROM
   - PLA
   - PAL

	Random Logic	PLA
1. combitional network	YES	YES
2. sequential circuit	YES	YES (with FF's)
3. # of basic elements	Large	?
4. minimality	YES, but hard to achieve (multilevel logic)	NO (2 Level logic)
5. optimality in VLSI	??? (interconnections)	attractive
6. automatic design	difficult	YES
7. design tools	some	some

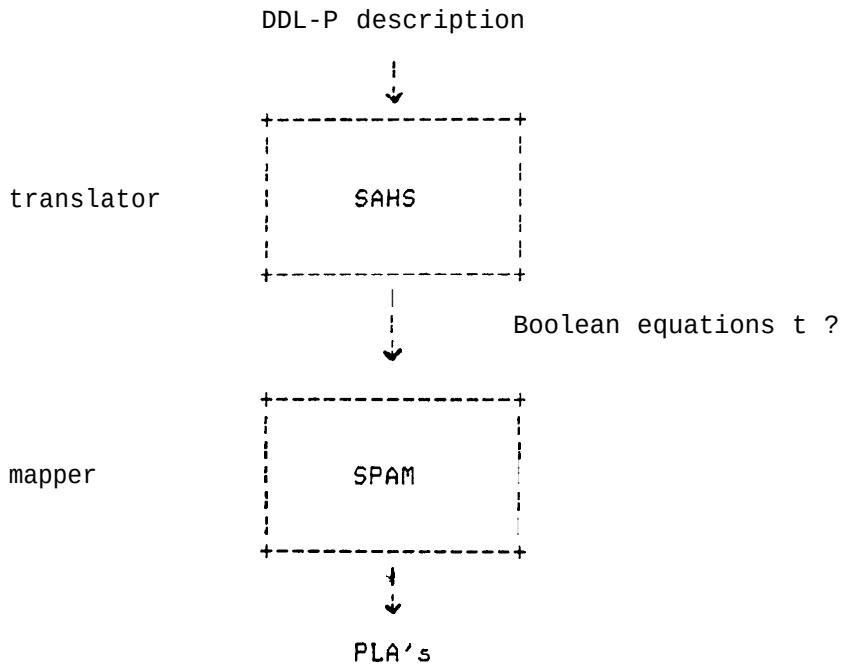
	RON	PLA
1. fundamental difference	uses all input combinations	does not need all input combinations
2. minimality	sometimes very wasteful	difficult to achieve for a large # of inputs and outputs
*** used in a *controller XXX*		
3. addressing logic	usually very complicated	may be simple
4. readability of format	easier	easy
5. changeability		
A. code	simple	difficult
B. structure	difficult	less difficult
6. automatic generation	reported	YES

OBJECTIUES

- automatically synthesize the controller of a digital system using PLA's

1. want to use a higher level language description of a digital system
   - DDL-P : register transfer Level (has been used successfully)
2. efficient PLA mapping
   - chip area
   - speed

- reduce the burdon of a desiginer as much as possible



Chracteristics of DDL-P

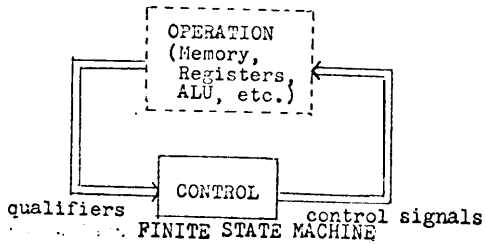
1. register transfer language
   - can be used more liberally
   - if carefully used, can contain all the structural information as well as functional behavior

2. based on ASM theory
   - Mealy model + Moore model
   - cf. Designing Logic systems using state machines by Christopher r. clare

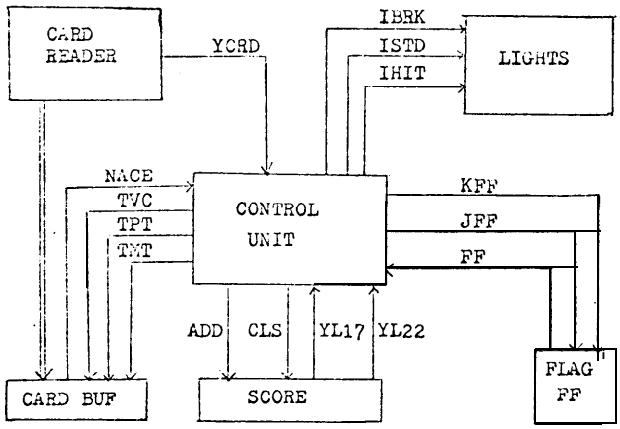
3. suitable for finite state machines
   - whole system synchronized

4. clear boundary between data flows and control flows
   - painful to a designer who wants to describe a digital system in DDL-P
   - good for a designer who wants to synthesize a physical hardware from a DDL-P description

DDL MACHINE MODEL



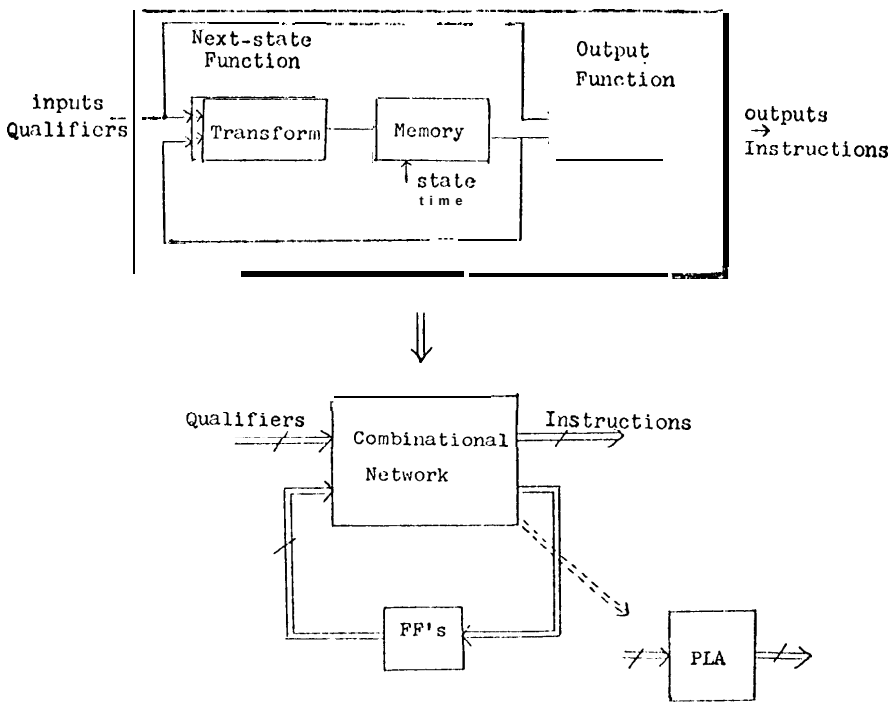
Ex.



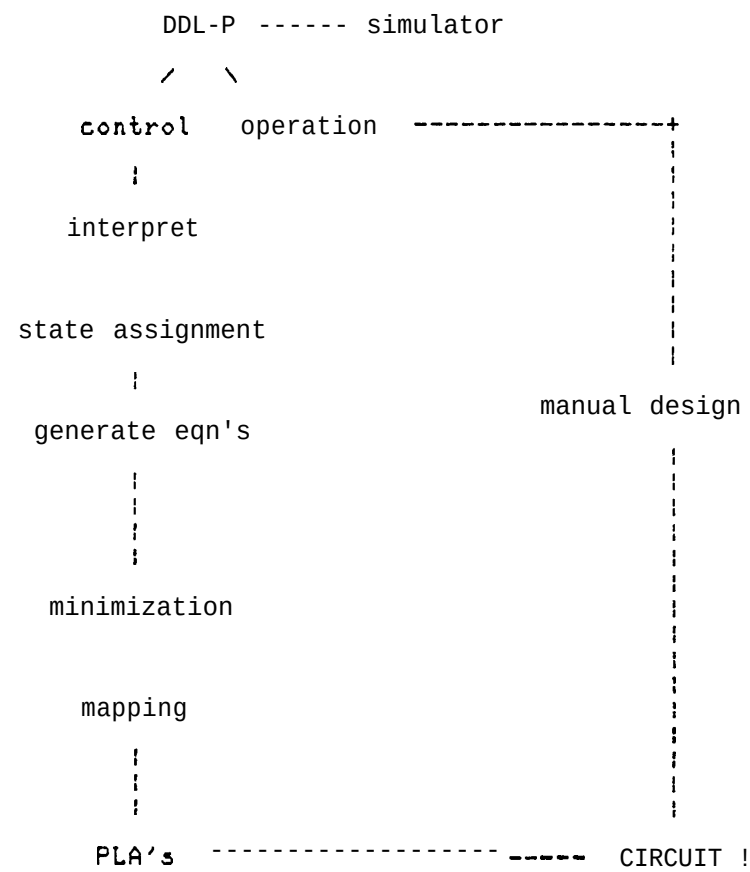
BLACKJACK MACHINE

5 QUALIFIERS
10 CONTROL SIGNALS

STATE MACHINE

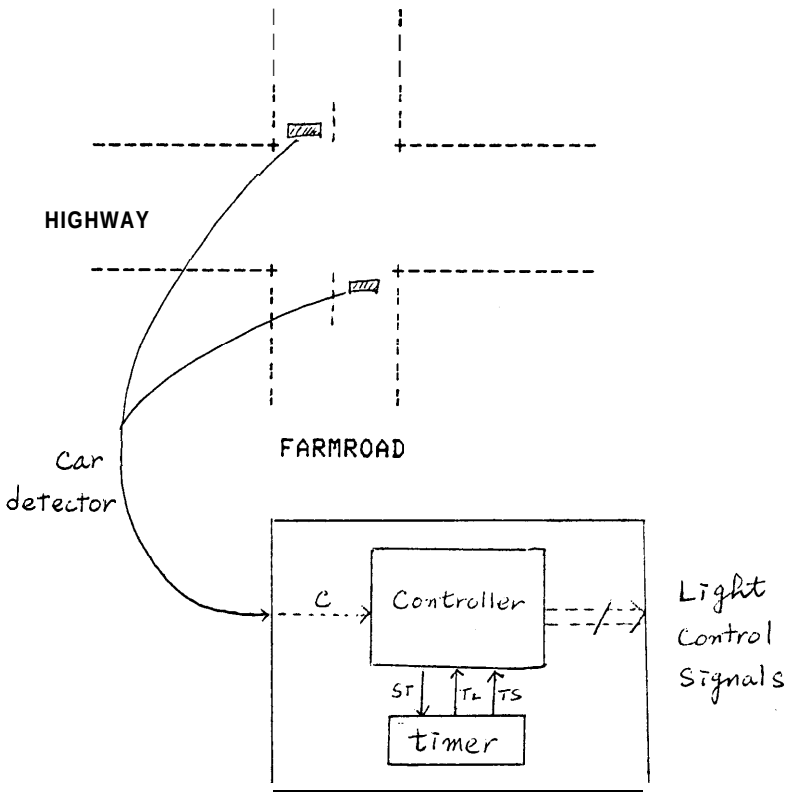


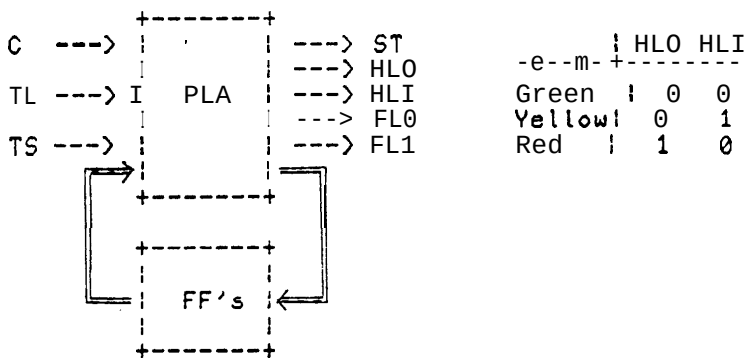
```
' B L A C K J A C K    M A C H I N E . '

REGISTER SCORE[5], CARDBUF[5], FF.
TERMINAL HIT, BROKE, STAND,
   VALUE[1:5] = INPUT(1,VALUE),
   YCRD = INPUT(1,YCRD),
   YL17 = SCORE<17, YL22 = SCORE<22,
   NACE = CARDBUF$1.
OPERATION
   TPT = CCARDBUF _ 5D10], TMT = CCARDBUF _ 5D22],
   TVC = CCARDBUF _ VALUE], IHIT = [HIT=1B1],
   ISTD = [STAND=1B1], IBRK = [BROKE=1B1],
   CLS = CSCORE _ 5D0], ADD=[SCORE _(SCORE(+)CARDBUF)TAIL 5],
   KFF = [FF_1D0], JFF = CFF _ 1D1 ]
CONTROL
   A:  CLS, KFF, ->B/
   B:  IHIT, TVC, ^YCRD^ ->C; ->B./
   C:  ^YCRD^ ->C; ->D./
   D:  ADD, ^NACE+FF^ ->F ; ->E./
   E:  JFF, TPT, ->D/
   F:  ^YL17^ ->B; ->G./
   G:  ^YL22^ ->K; ->H./
   H:  KFF, TMT, ^FF^ ->D; ->J ./
   J:  IBRK, ^YCRD^ ->A; ->J./
   K:  ISTD, ^YCRD^ ->A; ->K./.$
**********

CONTROL
   A:  CLS, KFF, ->B/
   B:  IHIT, TVC, ^YCRD^ ->C; ->B./
   C:  ^YCRD^ ->C; ->D./
   D:  ADD, ^NACE+FF^ ->F ; ->E./
   E:  JFF, TPT, ->D/
   F:  ^YL17^ ->B; ^YL22^ ->JK; ^FF^ ->D., KFF, TMT../
   JK: ^YL22^ ISTD; IBRK., ^YCRD^ ->A; ->JK./.$
```



EX. LIGHT CONTROLLER

(from " Introduction to VLSI systems "
by Mead & Conway)



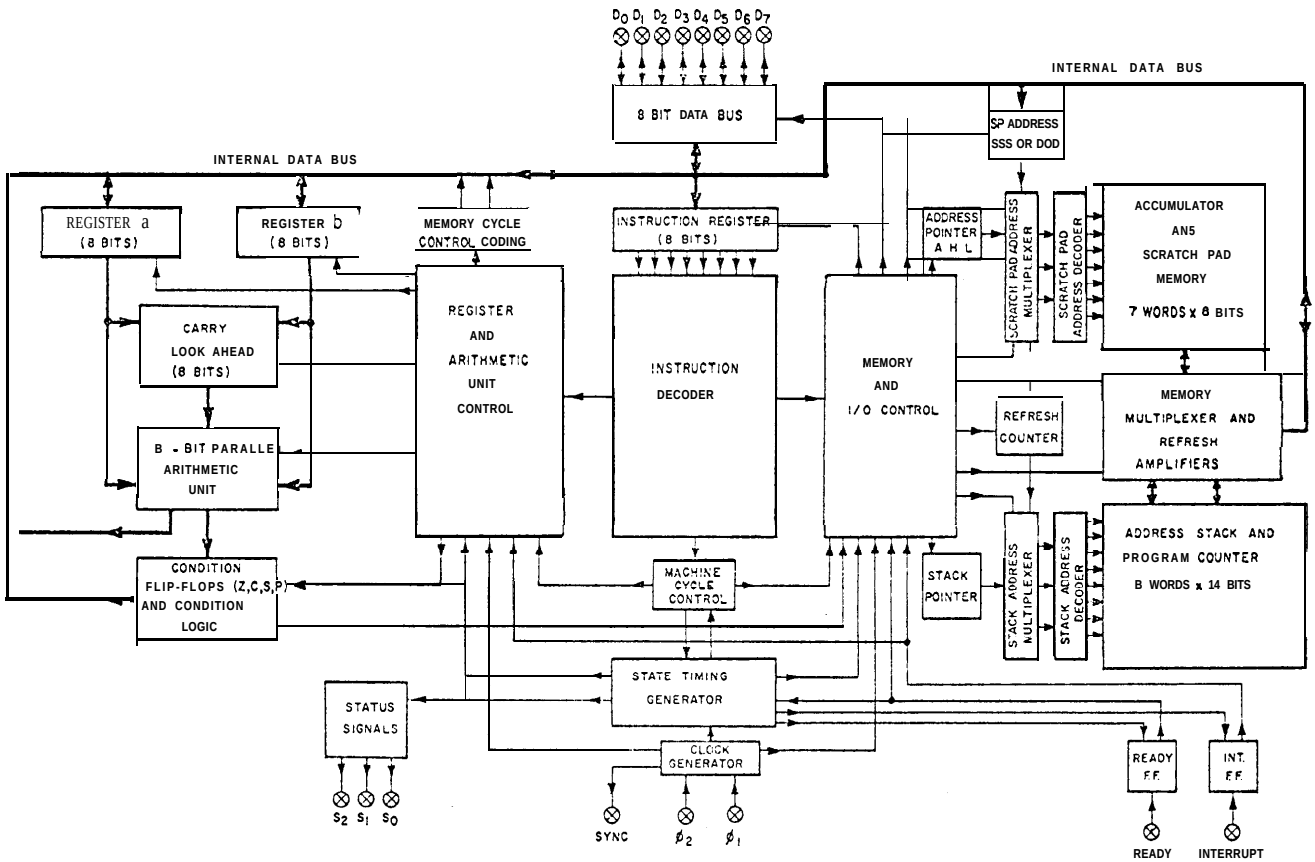


" LIGHT CONTROLLER "

```
TERMINAL
   C,TL,TS,ST,HL0,HL1,FL0,FL1.

C O N T R O L
   HG : IF C * TL  THEN ->HY,ST@,FL0@
                   ELSE ->HG,FL0@ ENDIF/
   HY : IF TS      THEN ->FG,HL1@,FL0@,ST
                   ELSE ->HY,HL1@,FL0@ ENDIF/
   FG : IF -C+TL  THEN ->FY,HL0@,ST@
                   ELSE ->FG,HL0@ ENDIF/
   FY : IF TS      THEN ->HG,HL0@,FL1@,ST
                   ELSE ->FY,HL0@,FL1@ ENDIF/.$
```

```
$ INPUT FILE :  light.ddl

     < OUTPUT EQUATIONS >

1.     ST= -Q2*-Q1*(C * TL) + -Q2*Q1*TS t Q2*Q1*(-C+TL) t Q2*-Q1*TS
2.    FL0 = -Q2*-Q1*(C * TL) + -Q2*-Q1*-(C * TL) + -Q2*Q1*TS
           t -Q2*Q1*-TS
3.    HL1 = -Q2*Q1*TS + -Q2*Q1*-TS
4.    HLO = Q2*Q1*(-C+TL) t Q2*Q1*-(-C+TL) + Q2*-Q1*TS + Q2*-Q1*-TS
5.    FL1 = Q2*-Q1*TS + Q2*-Q1*-TS

     < D FF EQUATIONS >

1.    D1 = -Q2*-Q1*-(-(C * TL)) + -Q2*Q1*(-TS) + -Q2*Q1*TS
          + Q2*Q1*(-(-C+TL))
2.    D2 = -Q2*Q1*TS + Q2*Q1*(-(-C+TL)) + Q2*Q1*-(-(-C
          +TL)) t Q2*-Q1*-TS
.$


   INITIAL SPECIFICATION (F/DC) :      28  /     0 CUBES

  # OF INPUTS/OUTPUTS =      5/    7

        <INPUTS>              <OUTPUTS>

     1.  Q2                1.  ST
     2.  C                 2.  FL0
     3.  TL                3.  HL1
     4.                    5.  HLO
     5.  TS                6.  FL1
                               D1
                           7.  D2


    1.  0011-I......

         ----

   28.  10--0.....I


  SOLUTION       :     10 CUBES


     1.  I----  . ..l...
     2.  10--0  ....1.1
     4.  10--1  1...1..
     5.  1110-  1....11 . . .

     6.  01---  ..l..l.
     7.  110--  1.....1
     8.  ii-l-  1.....1
     9.  01--1  1.1..11
    10.  0----  .1.....
```

```
" INTEL 8008 MICROPROCESSOR "

CONTROL
   M1T1  : HS1,PCLOUT, ^INT^ HSO, CLCY,  SIFF; INCPCL,  CIFF./
   M1T2  : HS0,SPCI, PCHOUT, INCPCH, INTC/
   M1TW  : RDYE, ^RDY^ FETCH, STOIR, STORB, ->M1T3;->M1TW./
   M1T3  : HS2, RDYC, INTE, ^APR+ROT^ ARA.,
           ^RST^ PUSH, CLRRA.,
           ^HLT^ HS1, ^INT^ ->M1T1;->M1T3.;
                 ^M2^ ->M2T1., ^RCF^ ->M1T1../
   M1T4  :HS2, HS1,HSO, ^SSG^ SSSRB.,^INR+DCR^ DDDRA.,
           ^RTG^ POP., ^RST^ RAPCH., ^LMR^ ->M2T1./
   M1T5  : HS2,  HSO, ^LRR^ RBDDD.,  ^INR^ INRADD., ^DCR^ DCRSUB.,
           ^APR^ ALUOP., ^ROT^ ROTRA., ^RST^ RRBPCL.,  -> M1T1/
   M2T1  : HS1, ^MRI^ LOUT; ^INP+OUT^ AOUT;
                 PCLOUT, ^IFF^ HS0, CLCY; INCPCL.../
   M2T2: HSO,  ^MRI^ ^LMR^ SPCW; SPCR.,  HOUT;
               ^INP+OUT^ RBOUT; SPCR, PCHOUT, INCPCH../
   M2TW  :RDYE,^RDY^ ^LMR^ RBOUT; ^-OUT^ FETCH, STORB..,->M2T3;
                 ->M2TW./
   M2T3  :HS2, RDYC, ^APM+API^ ARA.,
           ^M3^ ->M3T1., ^LMR+OUT^  ->M1T1./
   M2T4  : HS2,HS1,HS0, ^INP^ FFOUT./
   M2T5  : HS2, HS0, ^LRM+LRI^ RBDDD., ^APM+API^ ALUOP.,
           ^INP^ RBA., ->M1T1/
   M3T1  : HS1, ^LMI^ LOUT;  PCLOUT, ^IFF^ HS0, CLCY; INCPCL../
   M3T2  : HS0, ^LMI^ SPCW, HOUT; SPCR, PCHOUT, INCPCH./
   M3TW  :RDYE,^RDY^ ^LMI^ RBOUT; FETCH, STORA.,->M3T3;
                 ->M3TW./
   M3T3  : HS2,RDYC, ^CALSUB^ PUSH., ^LMI+JCF^ ->M1T1./
   M3T4  : HS2,HS1,HS0,RAPCH/
   M3T5  : HS2,HS0, RBPCL, ->M1T1/.$
```



		SAHS		SPAN
1. TRAFFIC LIGHT CONTROLLER	(5/ 7)	28	->	10 (0.51 s)
2. BLACKJACK MACHINE	(9/14)	40(+2)	->	18 (1.30 s)
3. INTEL 8008	(31/46)	148(+3)	->	68 (19.31 s)
4. INTEL 8080	(65/66)	181(+3)	->	129 (43.00 s)

(in/out) initial # of products final # of products

SAHS

(Stanford Automatic Hardware Synthesizer)

1. Interpret : source input : DDL-P
   - accept a subset of DDL-P
   - prefer a strict register transfer level description
   - hopefully can be used for other languages (ADLIB ..)
2. state assignment
   - manual, interactive or automatic (simple)
   - what would be real criteria ?
3. generate Boolean equations for the control part
   - operation part should be designed manually

   (for automatic design, fundamental philosophy would be different>

SPAM

(Stanford Programmable Array Minimizer)

1. input : Boolean equations, truthtable or SAHS result
   - some options are available

2. minimizer
   - heuristic
   - simple cost function : equal weight to each product
   - 72 inputs, 144 outputs (practical !!!)
   - essentially for shallow funtions

3. PLA mapper
   - objective : chip area efficiency, speed
   - solution : partitioning, folding

			EMIN (IBM 370/158 ?)	SPAM (DEC 20)
1.	PLA1	(8 / 6) 31 ->	31 (2.23 s)	30 (3.73 s)
2.	PLA2	(6 / 8) 9 ->	9 (0.81 s)	9 (0.51 s)
3.	PLA3	(16 / 7) 18 ->	13 (18.10 s)	13 (1.30 s)
4.	PLA4	(16 / 6) 25 ->	20 (193.26 s)	20 (1.59 s)
5.	PLA5	(16 / 7) 83 ->	44 (86.15 s)	42 (8.46 s)
6.	PLA6	(16 / 8) 67 ->	18 (37.71 s)	18 (6.75 s)
7.	PLA7	(13 / 6) 28 ->	21 (5.08 s)	21 (3.97 s)
8.	PLA8	(15 / 8) 17 ->	14 (20.16 s)	14 (1.29 s)
9.	PLA9	(13 / 6) 6072 ->	140 (3.05.00)	141 (1.40.00) 140 (+ .20.00)
10.	PLA10	(23 / 40) 273(+273)	->	165 (6.30.00)

THE USE OF HIERARCHICAL DESIGN INFORMATION IN PARTITIONING DIGITAL CIRCUITS

by

Thomas Payne

ABSTRACT

New algorithms that use hierarchical logical design information are being developed for the partitioning of digital systems. Information about functional relationships and structural relationships inherent in a hierarchical logical design are used to increase the effectiveness of the automatic partitioning algorithms. Emphasis has been placed on the generation of partitioning algorithms that handle a variety of constraints and realize a variety of partitioning quality criterion. These algorithms generate hierarchical physical realizations. Both an interactive algorithm where the user is required to make the partitioning decisions and an entirely automatic algorithm are being developed.

A Definition

Partitioning is the process of dividing a circuit into physically realizable subparts.

Partitioning Quality

1. Physically Realizable
2. Minimum Costs
3. Maximum Performance

Minimum Costs

- Design Costs
  - Partitioning Costs
  - Placement, Layout, and Routing Costs
- Production Costs
  - Repeated Types
  - Part and Connector Complexity
- Maintenance Costs
  - Testability
  - Reliability
  - Parts Cost

Maximum Performance

- Interconnect Path Length
  - Signal Delay
  - Power (Drivers)
- Testability
  - Test Point Availability
  - Functional Partitioning
- Reliability
  - Connection complexity
  - Power Dissipation

Hierarchical Design

- A design done at several levels of abstraction
- Natural for designer
- More popular as designs become more complex

Information in a Hierarchical Design

- Complete interconnectivity
- Logical entity interrelationships
  - Functional groupings
  - Structural groupings

Interactive Partitioning

- User makes the decisions
- Accurate bookkeeping
- Both the logical and physical designs are hierarchical
- Commands
  - Display
  - Assign
  - Remove
  - Estimate
  - Change Physical Specs
  - Compare
  - Partition Automatically

Automatic Partitioning

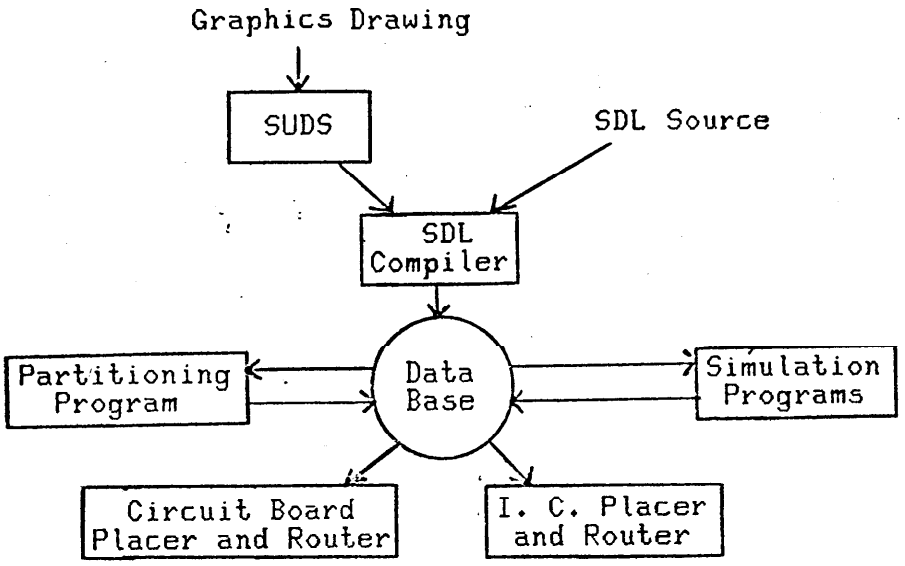
- Tradeoff Assessment
- Planning
  - Special Cases
    - Critical Signals
    - Regular Logic
  - Top Down
- Goal Directed Assignment with Backtracking

Conclusions

- No benchmark results yet
- Interactive Partitioning
  - Improved Efficiency
  - Improved Accuracy
- Automatic Partitioning
  - Accurate

Future Work

- Complete Automatic Partitioning Implementation
- Look at Engineering Change Problem



VLSI CIRCUIT PARAMETERS COMPUTED FROM PROCESS VARIABLES

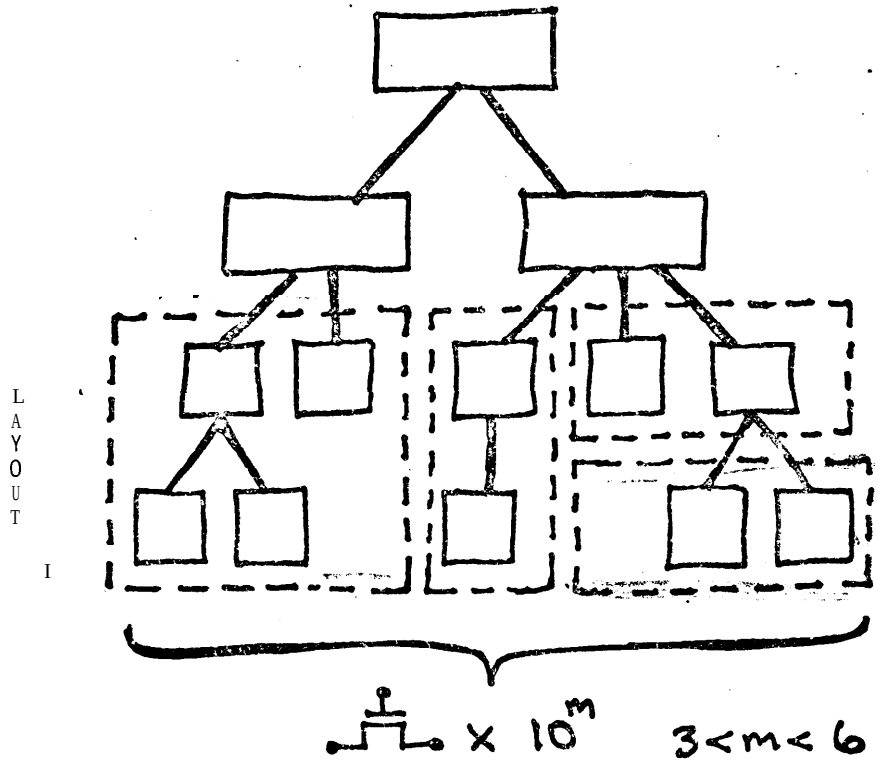
by

Robert Dutton

ABSTRACT

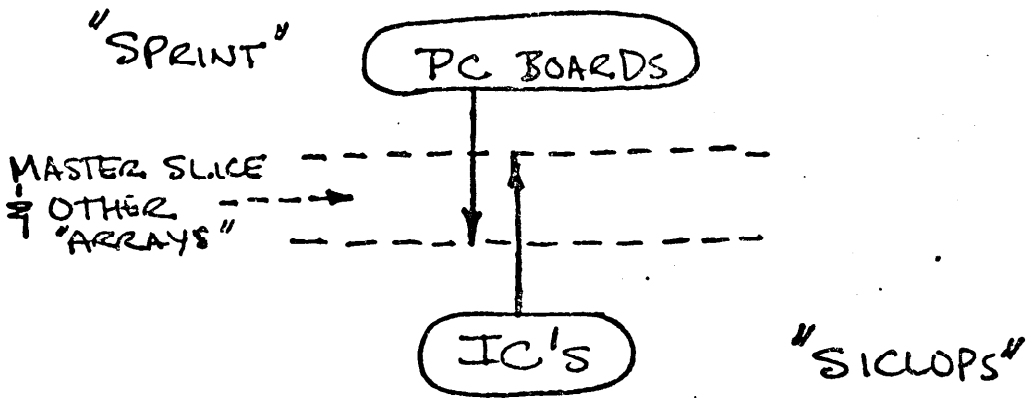
The use of process models such as SUPREM to predict device structures and parameters, interaction with process control. Use of process and device models to predict circuit and system performance.

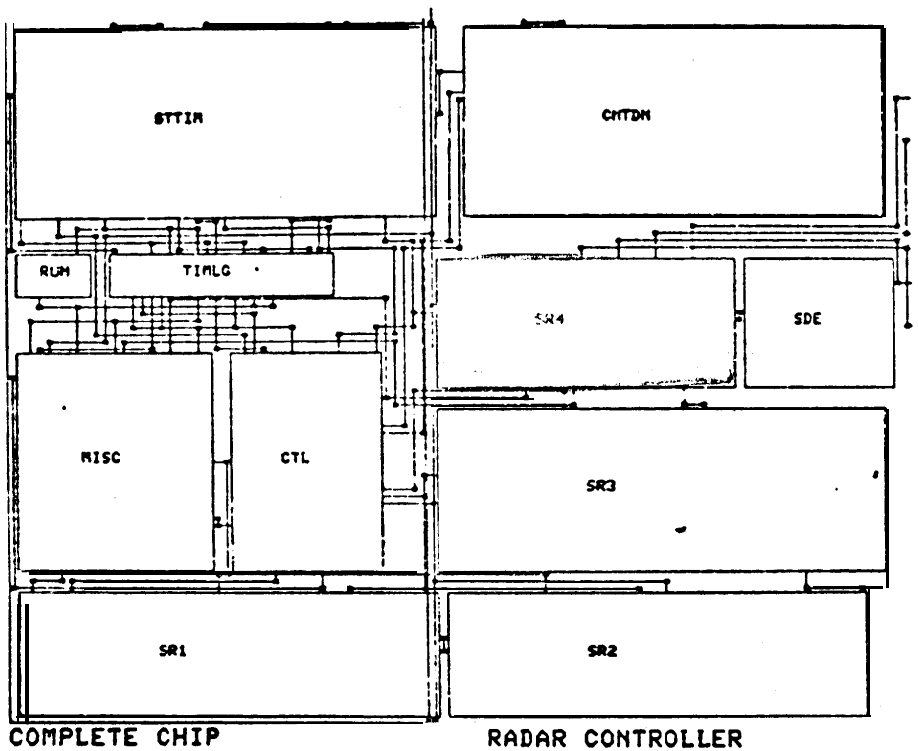
HIERARCHICAL DESIGN

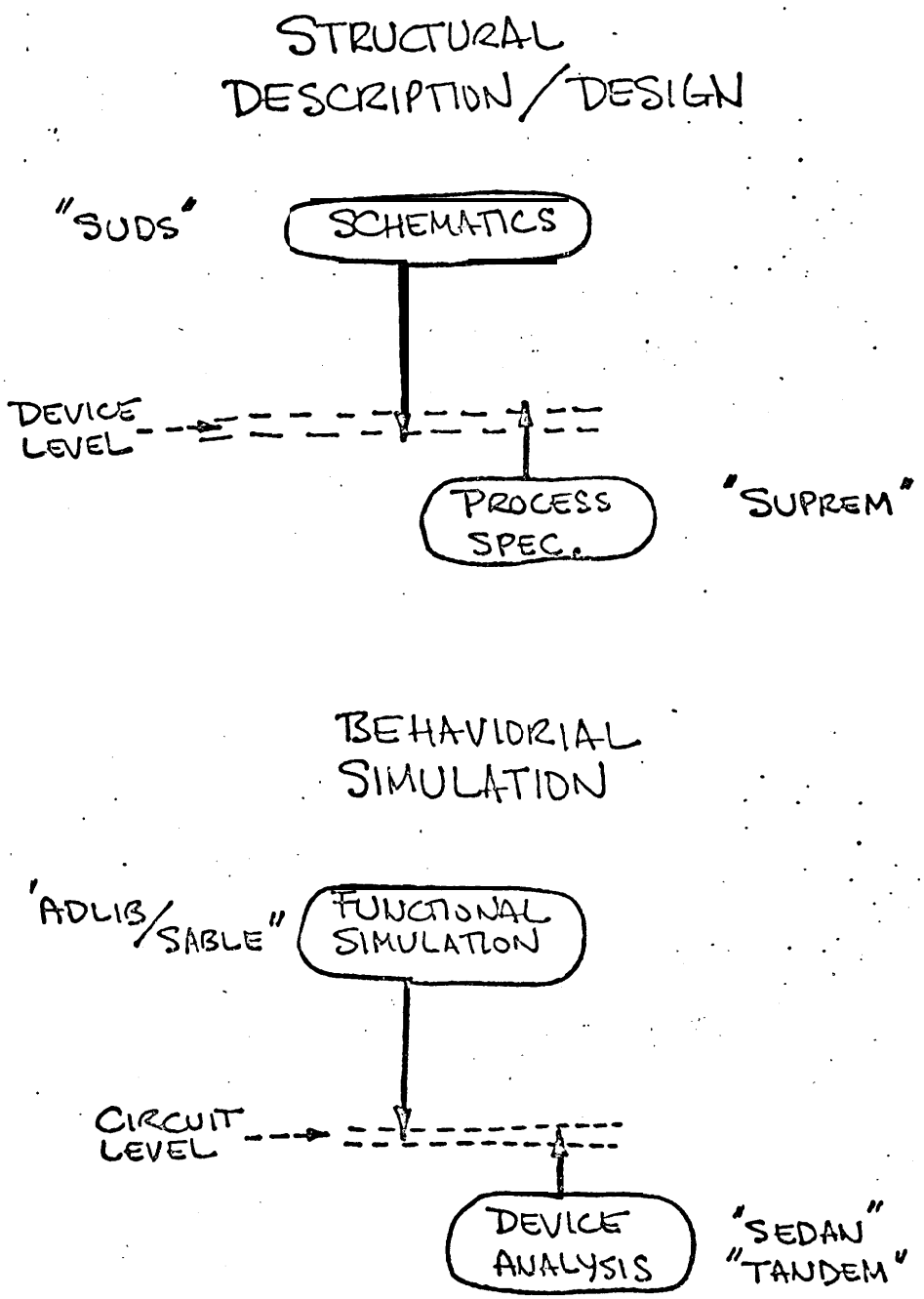


TECHNOLOGY

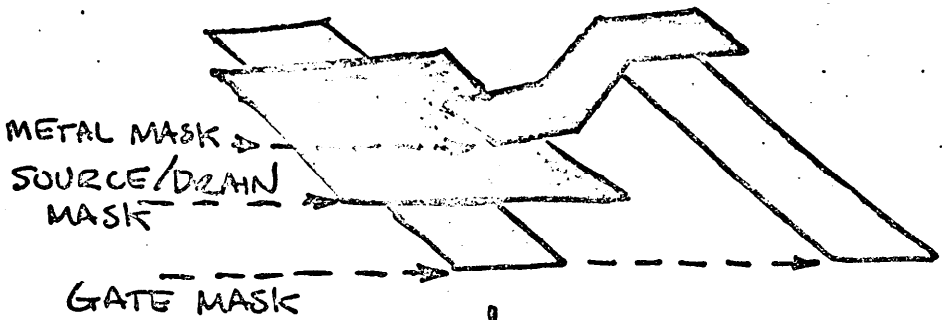
LAYOUT



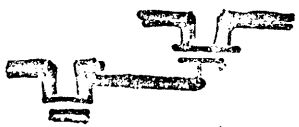

STTIM
CNTDN
RUM
TIMLG
SR4
SDE
MISC
CTL
SR3
SR1
SR2
COMPLETE CHIP
RADAR CONTROLLER


STRUCTURAL
DESCRIPTION/DESIGN
"SUDS"
SCHEMATICS
DEVICE
LEVEL
PROCESS
SPEC.
"SUPREM"
BEHAVIORIAL
SIMULATION
"ADLIB/SABLE"
FUNCTIONAL
SIMULATION
CIRCUIT
LEVEL
DEVICE
ANALYSIS
"SEDAN"
"TANDEM"

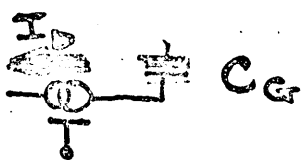
VERIFICATION AND TECHNOLOGY DESIGN



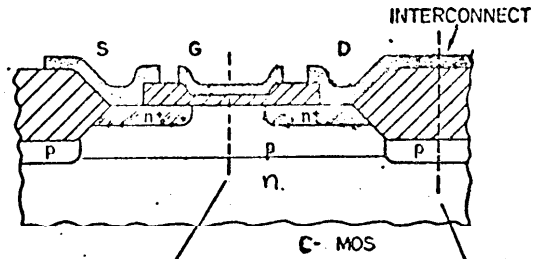
TOPOLOGY EXTRACTION



DEVICE/CIRCUIT PERFORMANCE

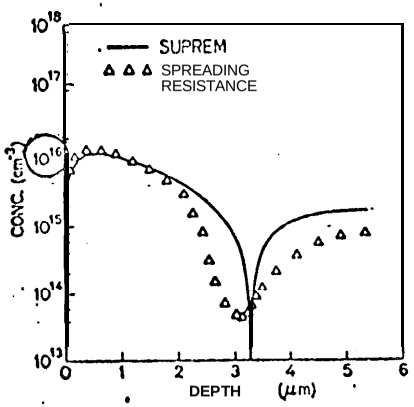


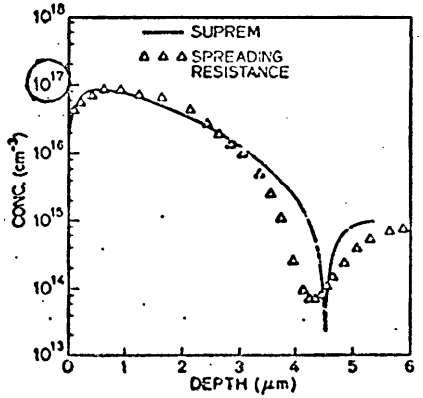
MOS PROCESS MODELING



CHANNEL IMPLANT

FIELD IMPLANT





SUPREM

STANFORD UNIVERSITY PROCESS ENGINEERING MODELS PROGRAM

AN IC FABRICATION SIMULATOR WHICH ACCEPTS PROCESS STEPS AS INPUT AND PRODUCES A ONE-DIMENSIONAL IMPURITY PROFILE AS OUTPUT,

ALLOWABLE STEPS	SECOND ORDER CONSIDERATIONS
ION IMPLANTATION	MULTIPLE SPECIES B, P, A_S, SB
PREDEPOSITION	SPECIES COUPLING - A_S-B, P-B
OXIDATION/DIFFUSION	OXIDATION ENHANCED DIFFUSION (OED)
EPITAXY	CONCENTRATION DEPENDENT OXIDATION
ETCHING/OXIDE DEPOSITION	A_S CLUSTERING

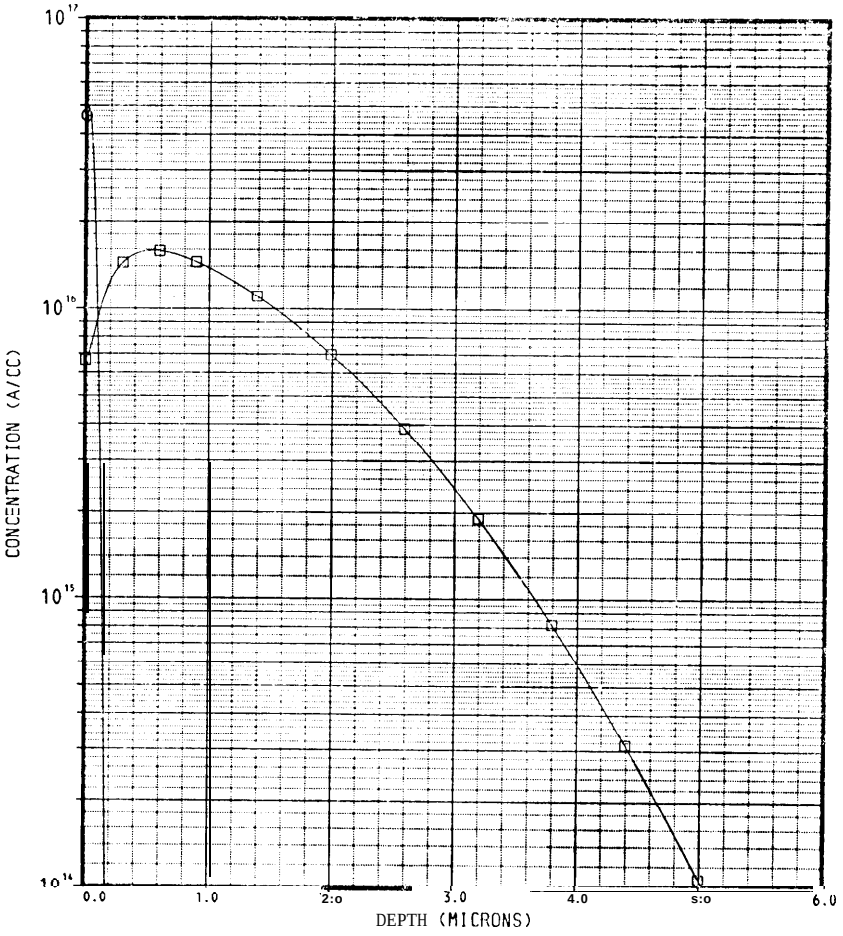
TRANSIENT EPI DOPING
DIFFUSION FROM POLY
OXYGEN ENHANCED STACKING FAULTS (OSF)

*** BELL SUPREM II *** INPUT ***

```
1...TITL  CMOS P-WELL SIMULATION
2...GRID  DYSI=0.01, DPTH=0.6, YMAX=2.5
3...SUBS  ORNT=100, ELEM=-, CONC=1E15
4...
5...COMM  STARTING OXIDE THICKNESS OF 500A.
6...STEP  TYPE=DEPO, TIME-1, GRTE=0.0500
7...
8...PLOT  TDTL=Y, CMIN=14, NDEC=3, WIND=3
9...
10...COMM  P-WELL IMPLANT
11...STEP  TYPE=IMPL, ELEM=B, DOSE=5E12, AKEV=200.
12...
13...COMM  --- STOP PLOTTINGG, START PRINTING ---
14...PLOT  TOTL=N
15...PRINT HEAD=Y
16...
17...COMM  DRIVE-IN IN N2 FOR 1.5 HOURS
18...STEP  TYPE=OXID,TEMP=1100, TIME-90, MODL=NITO
19...
20...STEP  TYPE=ETCH, TEMP=25
21...
22...COMM  ---EXTEND GRID SPACE---
23...GRID  DYSI=0.015, DPTH=1., YMAX=7.0
24...
25...COMM  DRIVE-IN FOR 15 HOURS IN 10% DRY 02
26...MODEL NAME=DRY1, PRES=0.1
27...STEP  TYPE=OXID, TEMP=1100, TIME-900, MODL=DRY1
28...
29:::COMM  FIELD OXID GROWTH IN WET 02 FOR 5 HOURS
30...STEP  TYPE=OXID, TEMP=1025, TIME=300, MODL=WET0
31...
32...STEP  TYPE=ETCH, TEMP=25
33:::
34...COMM  GATE OXIDATION AT 1000 C
35...STEP  TYPE=OXID, TEMP=1000, TIME-5, MODL=DRYO
36...STEP  TYPE=OXID, TEMP=1000, TIME-5, MODL=WET0
37...PLOT  TOTL=Y, WIND=6
38...STEP  TYPE=OXID, TEMP=1000, TIME=5, MODL=DRYO
39...
40...COMM  ANNEAL ---CALCULATE THRESHOLD VOLTAGE---
41...MODEL NAME=SPM1, GATE=AL, QSS0=4E10, CBLK=1
42...STEP  TYPE=OXID, TEMP=1000, TIME=30, MODL=NITO, MODL=SPM1
43...COMM  THRESHOLD TAILORING IMPLANT
44...MODEL NAME=SPM1, CBLK=6E15
45...STEP  TYPE=IMPL, ELEM=P, DOSE=5E11, AKEV=90, MODL=SPM1
46...
47...END
```

STEP = 12 CMOS P-WELL SIMULATION
THRESHOLD TAILORING IKPLANT

ELEMENT	SILICON DIFFUSIVITY	SEGREGATION COEFFICIENT	SURFACE TRANS. COEF.	TOTAL DOSE
BORON	9.162E-05	0.349741	2.526E-03	3.02363E 12
PHOSPHORUS	0.	0.	0.	3.58783E 11



DOSE= 5.0000E 11 ATMS/CM^2 VOLTAGE= 90. KEV RANGE= 0.097 UM
OXIDE THK= 808 A

TANDEM

STATIC SOLUTION (POISSON)

FINITE DIFFERENCE

NON-PLANAR SURFACE

SLOR

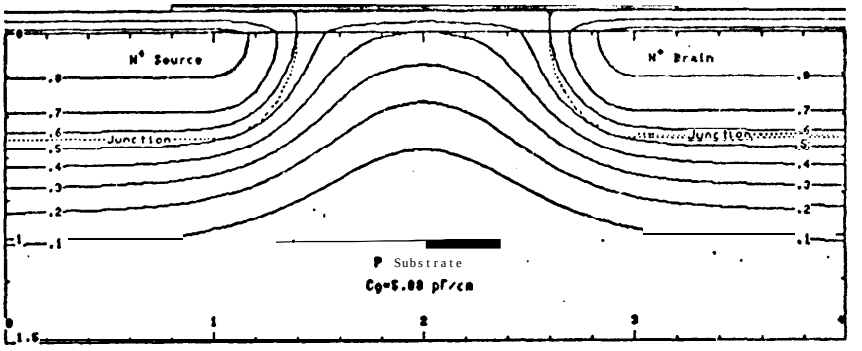
STORAGE - 8mn VARIABLES

EXECUTION - 180 SEC ON HP2117F
25 SEC ON DEC-20

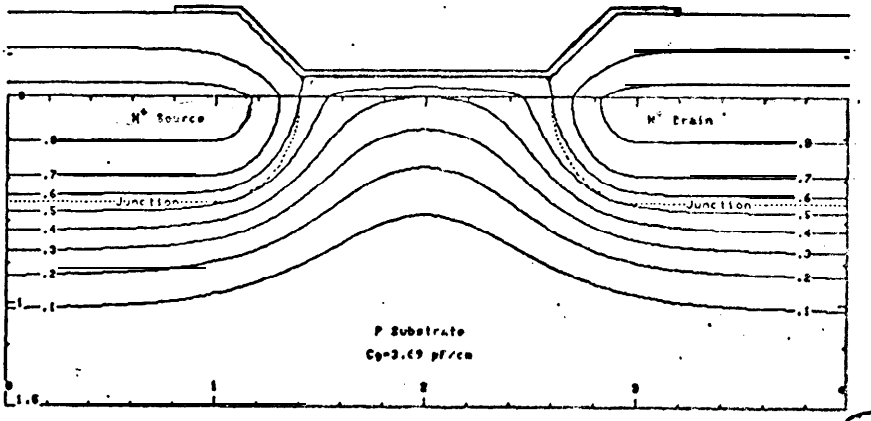
DOPING PROFILES - DIFFUSION, IMPLANT, SUPREM

OUTPUT - TERMINAL GRAPHICS

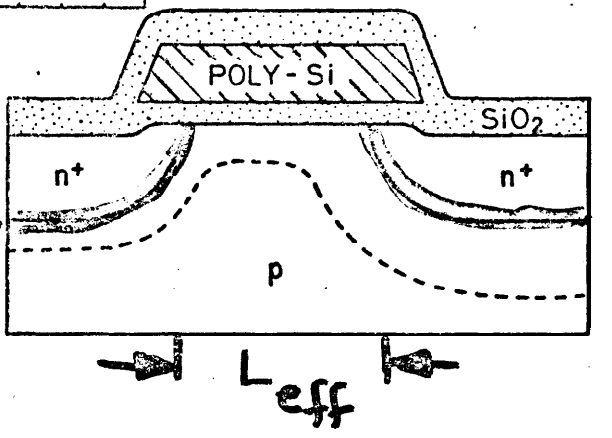
GATE CAPACITANCE

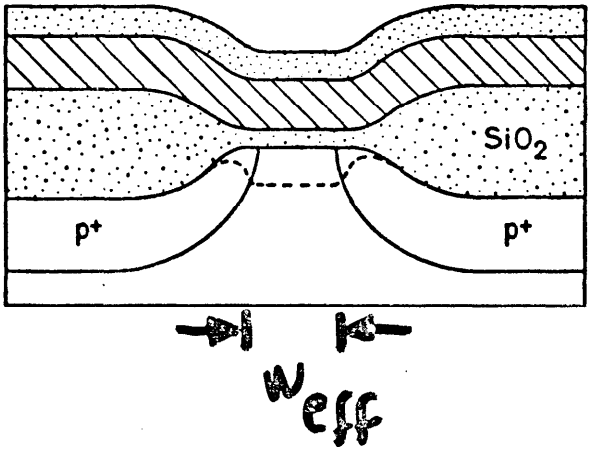


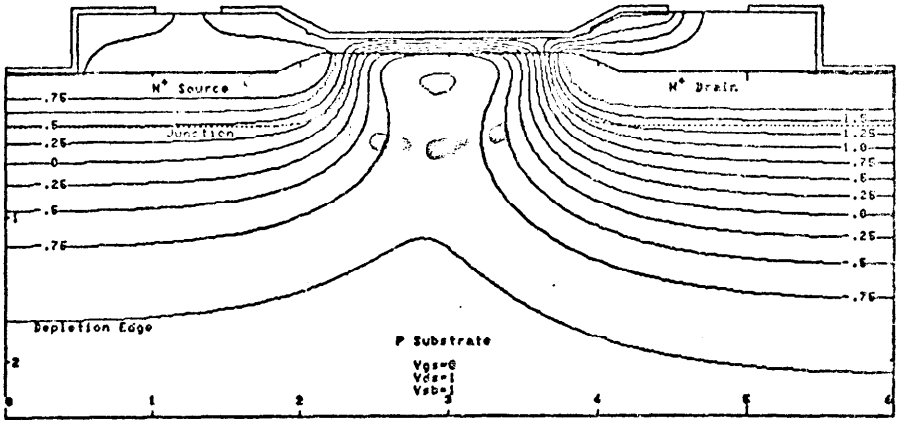
Cg = 5.88 pF/cm



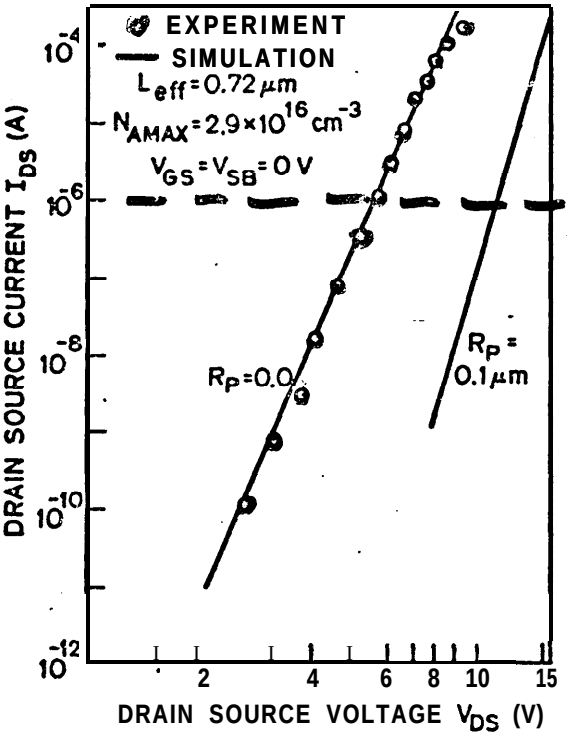
Cg = 3.69 pF/cm

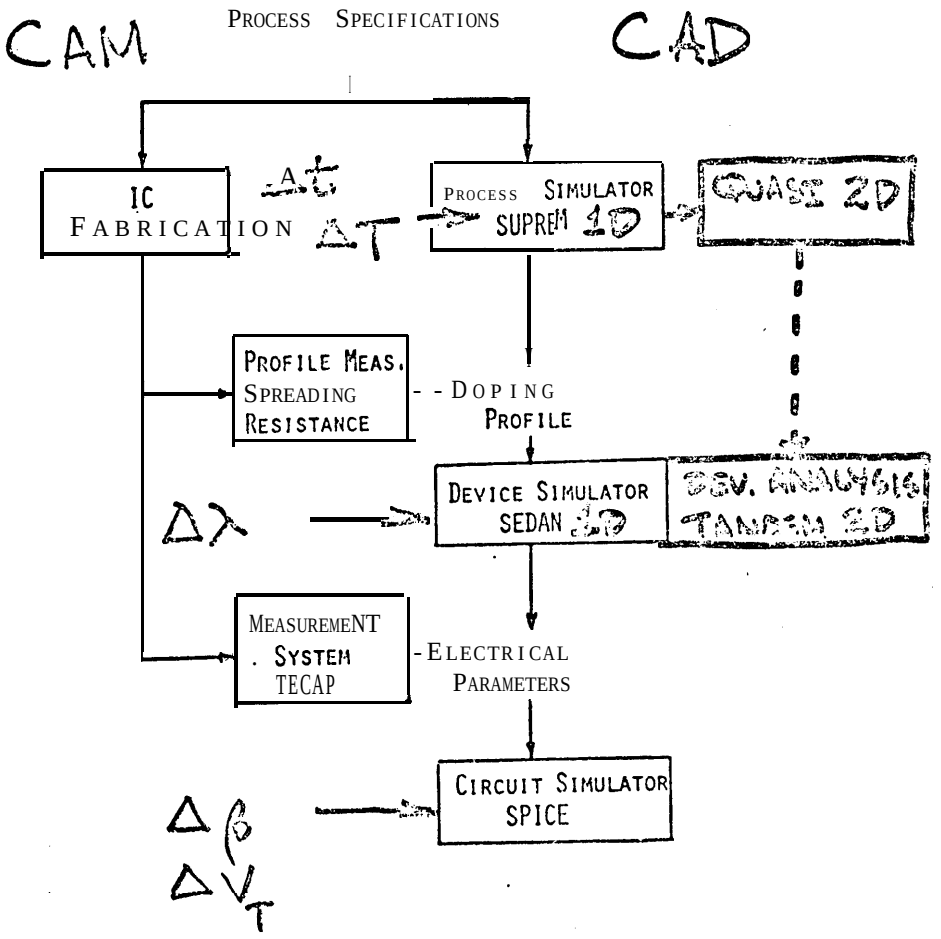






POTENTIAL CONTOURS




CAM
PROCESS SPECIFICATIONS
CAD
IC FABRICATION
Δt
ΔT
PROCESS SIMULATOR SUPREM 1D
QUASI 2D
PROFILE MEAS. SPREADING RESISTANCE
- - DOPING PROFILE
DEVICE SIMULATOR SEDAN 2D
2DV, ANALYSIS TANSEN 2D
ΔX
MEASUREMENT . SYSTEM TECAP
- ELECTRICAL PARAMETERS
CIRCUIT SIMULATOR SPICE
Δβ
ΔV_T

SHORT PRESENTATIONS:

"Hierarchical DRC" by Mark Horowitz

“SUDS-II” by Wayne Wolf

"On-line DRC of PC Designs" by Tom Bennett

“Chip Planning” by Eric Slutz

“CROCODILE” by John Beetem

"Data Base" by Markus Bayegan

"Bus Router" by Tom Blank

"Graphics Terminal" by Andreas Bechtol sheim

HIERARCHIAL DESIGN RULE CHECK (DRC)

MARK HOROWITZ

HIERARCHICAL DRC

IDEA : Use information in layout
- cell calls
- array

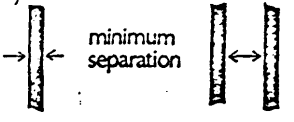
ADVANTAGES: Check cells once
- Smaller input
- Faster execution

HOW: **Check boundary for each** placement

CURRENT DRC

PURPOSE: Check artwork for layout violations

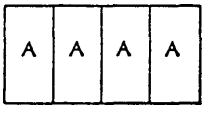
example: minimum width. → ← minimum separation ↔



INPUT TO SYSTEM: Instantiated layout (at rectangle level) no nesting allowed

PROBLEMS: Cells are check in each placement

example:



HIERARCHICAL DRC

IDEA: Use information in layout
- cell calls
- array

ADVANTAGES: Check cells once
- Smaller input
- Faster execution

HOW: Check boundary for each placement

PROBLEMS QUESTIONS

WHAT IS BOUNDARY OF CELL
- Bounding box
- User defined
- Merged layer

CONSTRAINTS

The SUDS-II Drawing System

Wayne Wolf

Advantages of SUDS-II:

* written in transportable language (pascal)

* relatively low-cost terminal required

* simple to learn

* encourages hierarchical design (push & pop commands)

* access to many utility programs through SDL

Current work:

* modify user interface
* investigate component paramaterization
* define optimum hardware mix

DYNAMIC DESIGN RULE CHECKING IN AN INTERACTIVE PC EDITOR

T.C. Bennett

SLAC
Stanford *University*

ABSTRACT

Batch design rule checking has been the standard approach for most DA systems. This approach has several major short-comings:

1. Allow DR violations to exist.
2. Requires DR checking after human intervention.

This method integrates the DRC program into the PC editing cycle. Since DRC on an entire design is a time consuming operation, we find this whole idea unsatisfactory.

At the outset we developed a router which does not produce DR violations during automatic routing. The router can be viewed as consisting of two major parts:

1. Automated line search algorithms.
2. DR enforcement data structures.

The PC editor in our system is an extension of the automatic router: it allows the user to control the line search algorithms as well as deleted critical obstructions. With this approach we obtain two desired goals:

1. No DR violations can be generated by human intervention.
2. The user has the full power of the auto router to connect non-critical connections.

Motivation

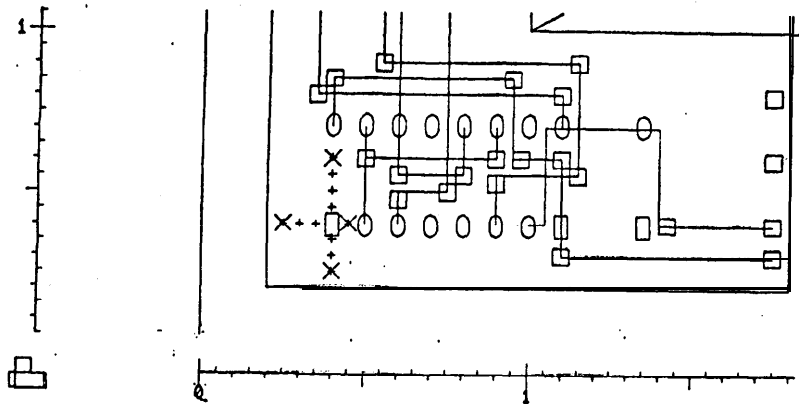
- automated routing is seldom 100% complete
- human intervention is definitely needed
- batch *design* rule checking!
  1) allows additional errors to be made
  2) slow turn around
- need for incremental design rule enforcement
- prohibit DR violations at all times
- utilize existing line-search technique3
- interactive routing aids for the designer

Design Rule Enforcement

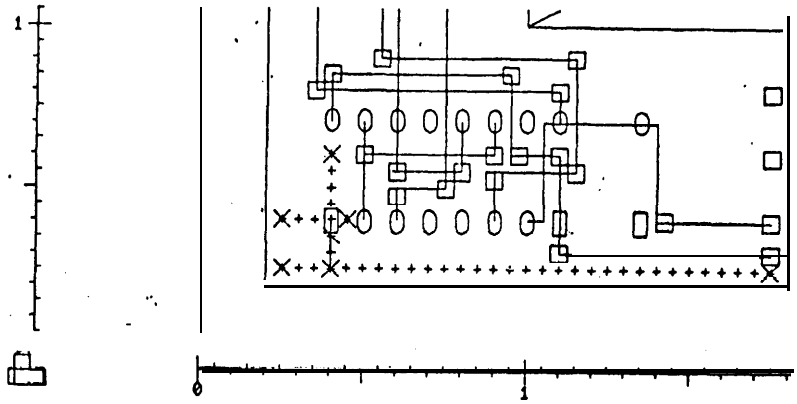
- current net *in* linked list structure
- remainder of design represented by a bit map
- Line search based on Hightower algorithm

Design Rules

- connectivity:
  from circuit specification
- static obstructions:
  board geometry
  component topology
  pads
- dynamic obstructions:
  routed nets
  line segments
  vias

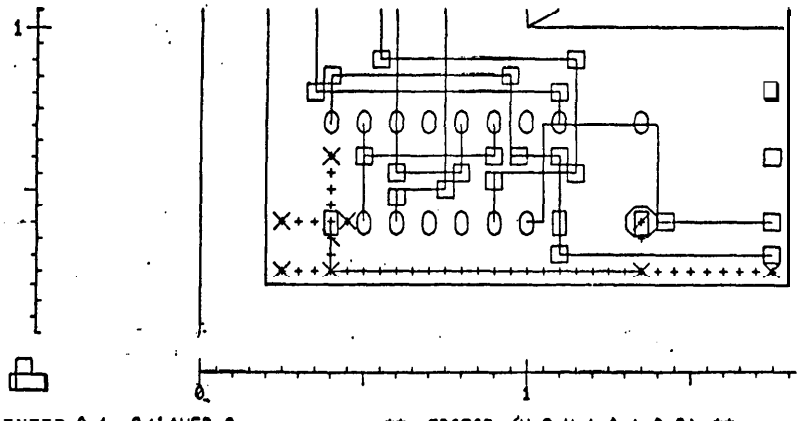



```
** EDITOR (H=2,V=1,C=1,S=2) **
  SET NET
  ROUTE (MODE)
  DELETE (MODE)
  IDENTIFY (MODE)
  DISPLAY
  RESET
  CLEAN
  HELP
  SAVE/RESTORE
  QUIT
```



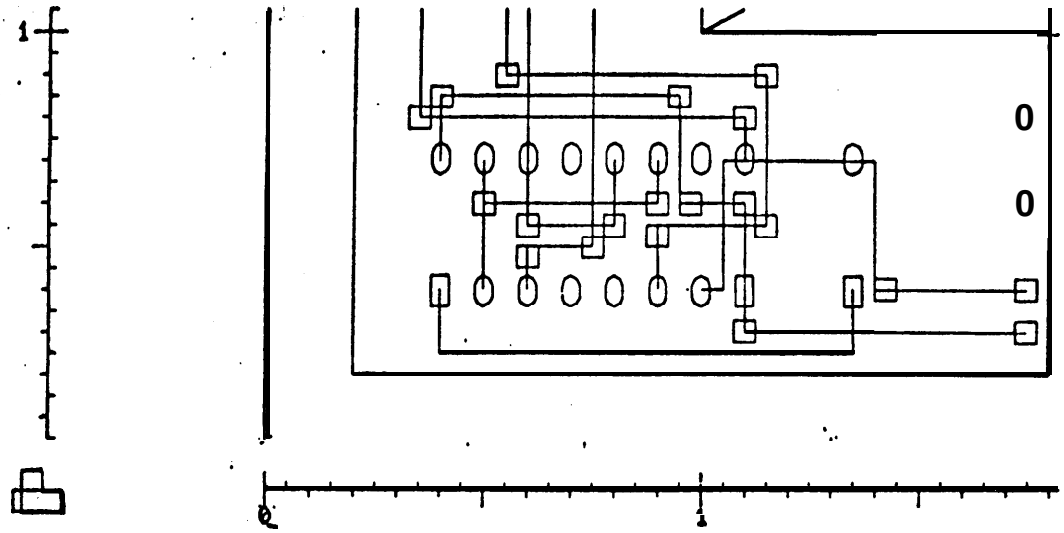
ENTER 0,1..2 :LAYER=?

```
** EDITOR (H=2,V=1,C=1,S=2) **
  SET NET
  ROUTE (MODE)
  DELETE (MODE)
  IDENTIFY (MODE)
  DISPLAY
  RESET
  CLEAN
  HELP
  SAVE/RESTORE
  QUIT
```

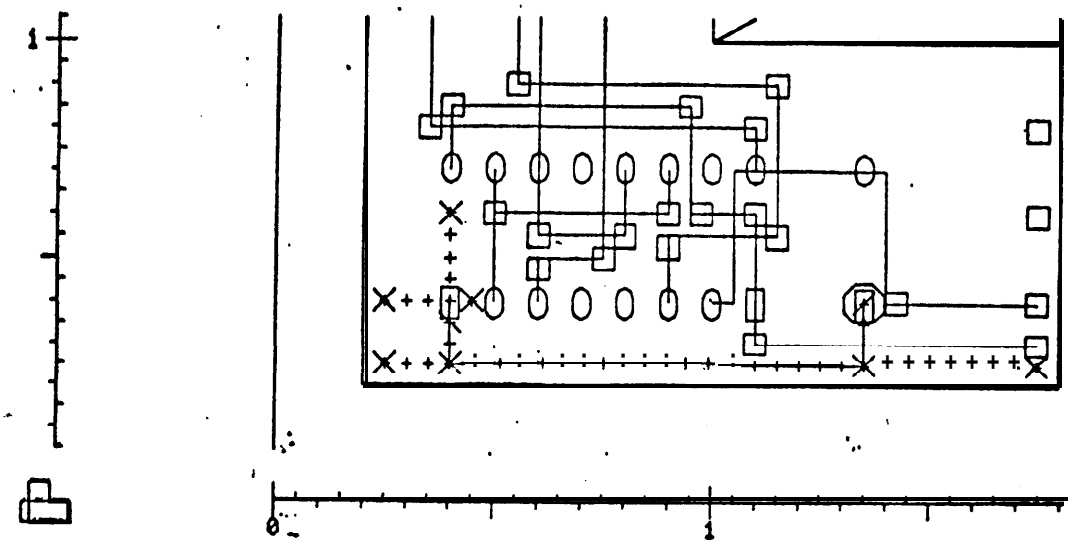


ENTER 0,1..2 :LAYER=?

```
** EDITOR (H=2,V=1,C=1,S=2) **
  SET NET
  ROUTE (MODE)
  DELETE (MODE)
  IDENTIFY (MODE)
  DISPLAY
  RESET
  CLEAN
  HELP
  SAVE/RESTORE
  QUIT
```

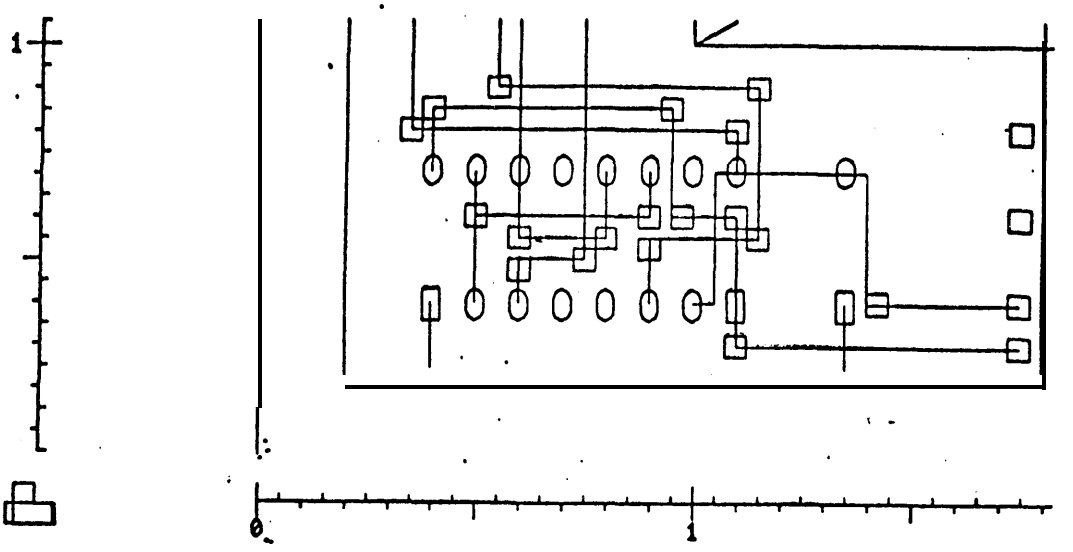



```
** EDITOR (H=2,V=1,C=1,S=2) **
  SET NET
  ROUTE (MODE)
  DELETE (MODE)
  IDENTIFY (MODE)
  DISPLAY
  RESET
  CLEAN
  HELP
  SAVE/RESTORE
  QUIT
```

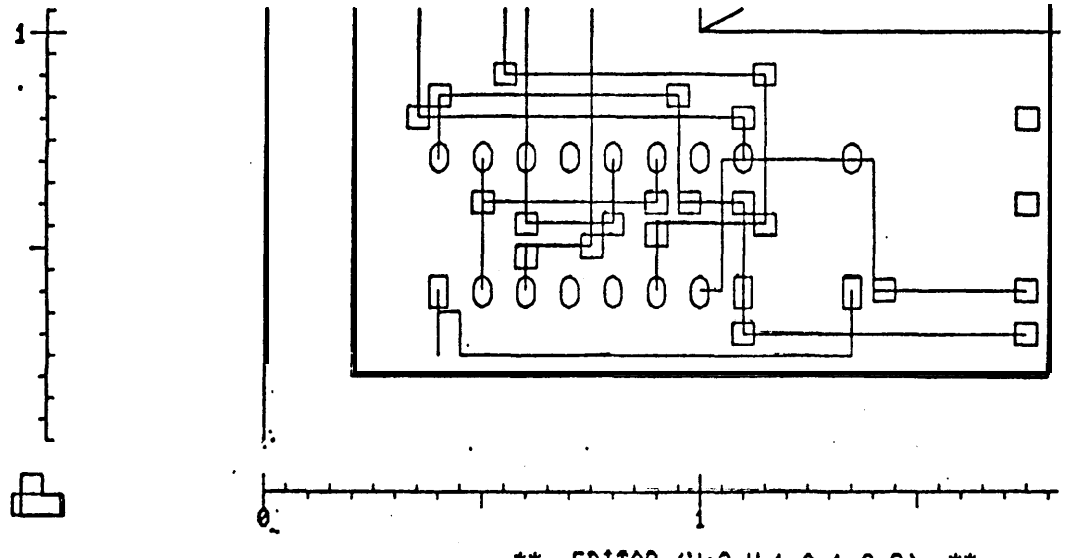



```
ENTER 0,1..2 :LAYER=?
```

```
** EDITOR (H=2,V=1,C=1,S=2) **
  SET NET
  ROUTE (MODE)
  DELETE (MODE)
  IDENTIFY (MODE)
  DISPLAY
  RESET
  CLEAN
  HELP
  SAVE/RESTORE
  QUIT
```




```
** EDITOR (H=2,V=1,C=1,S=2) **
  SET NET
  ROUTE (MODE )
  DELETE (MODE )
  IDENTIFY (MODE )
  DISPLAY
  RESET
  CLEAN
  HELP
  SAVE/RESTORE
  QUIT
```




```
** EDITOR (H=2,V=1,C=1,S=2) **
  SET NET
  ROUTE (MODE )
  DELETE (MODE )
  IDENTIFY (MODE)
  DISPLAY
  RESET
  CLEAN
  HELP
  SAVE/RESTORE
  QUIT
```

SHAPE DETERMINATION

Eric A. Slutz

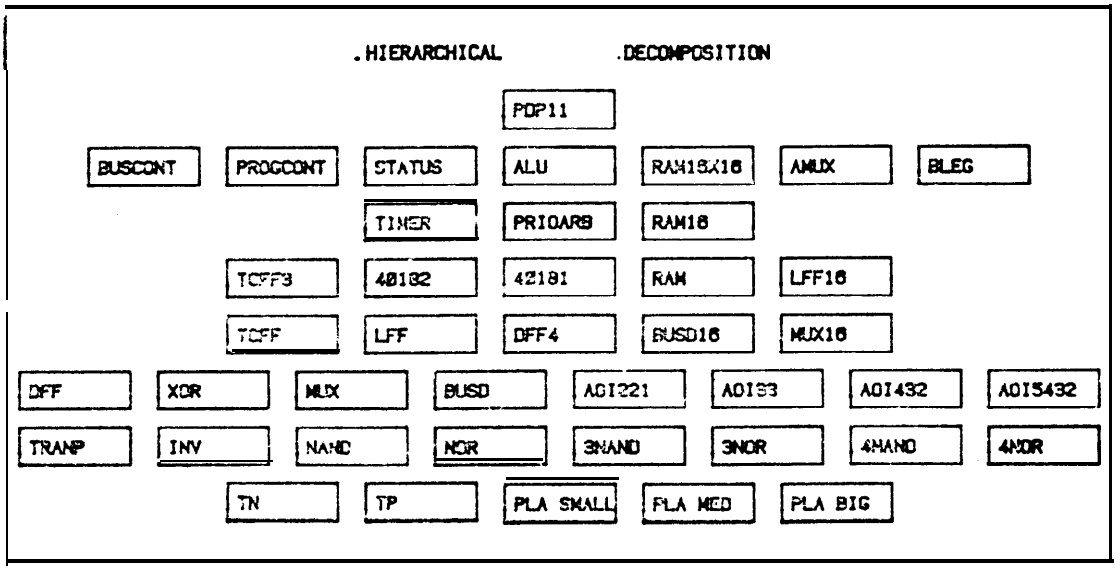
SUMMARY:

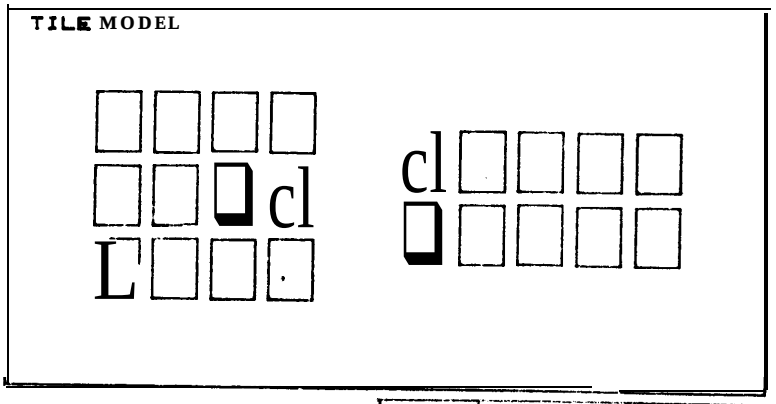
Hierarchical Decomposition

Bottom Up Area Calculation

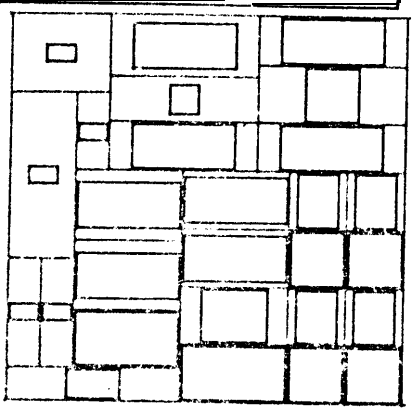
Top Down Sh Determi nat i on:

* Tile I
* Topological Placement
* Shape Adjustment
* Critical Path Abutment

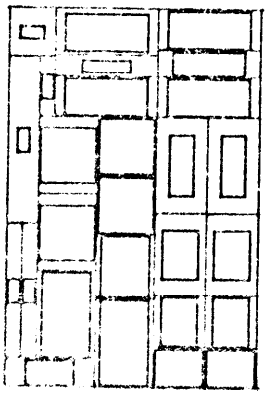




40101
PLACED



40101
ADJUSTED



CROCODILE

A Graphical High Level Language for Describing Electronic Systems

John Beetem

Crocodile Features

(1) Describes both structure and functional behavior in the same diagram.

(2) GRAPHICAL

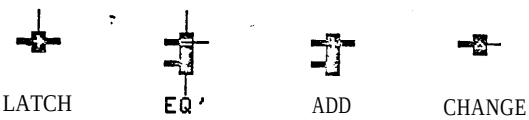
(3) PARALLEL

(4) HIERARCHICAL

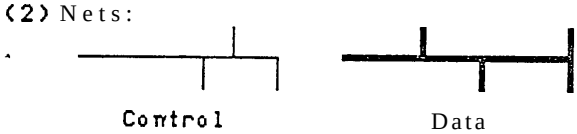
(5) Separation of Control and Data flow.,

Symbols of the Crocodile Language

(1) Primitive Components:



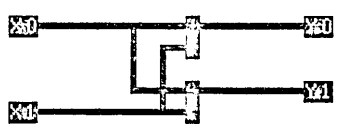
(2) Nets:



(3) External Contacts:

A set of components, nets, and external contacts can be grouped into an Object:

Object FFT2



(4) Non-primitiue Components:

Crocodile Summary

(1) A few simple objects and a general way to connect them: produces a simple but powerful language.

(2) Crocodile has many applications: computer hardware, signal processing, parallel software, etc.

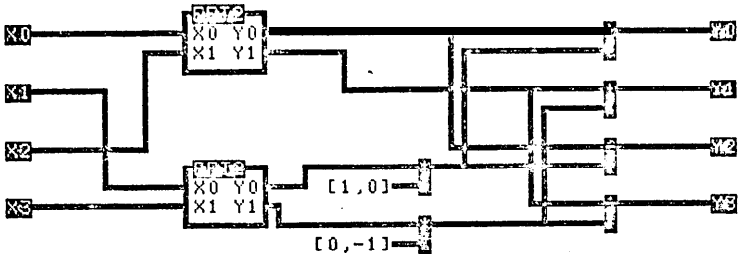
The Crocodile Project

(1) Editor

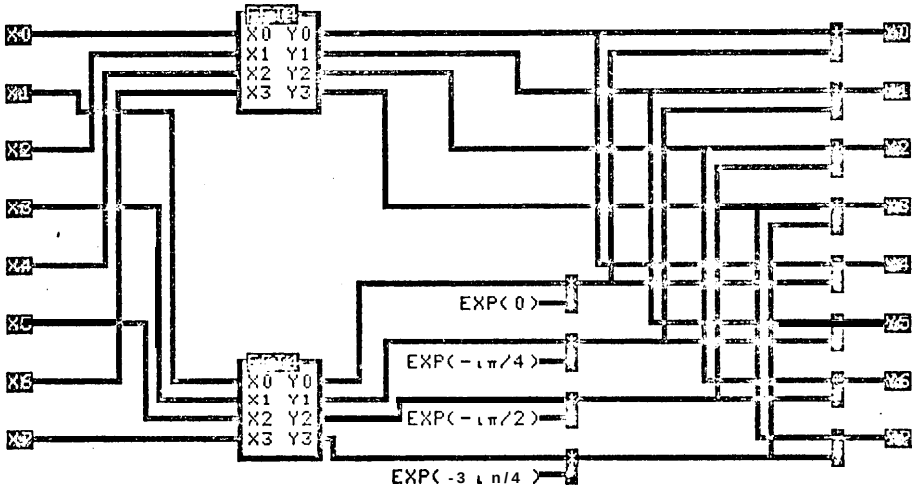
(2) Simulator

(3) Interface to a design database

Object FFT4 .



Object FFT8



DESIGN AUTOMATION DATA BASE

by

Markus Bayegan

BACKGROUND

The increasing number of Design Automation Applications and the growing complexity of design (LSI/VLSI) make the use of Centralized Engineering Data Bases a necessity.

WHAT IS A DATA BASE?

A data base system is a highly structured and formalized system in which a large amount of data can be manipulated concurrently by different programs, without detailed knowledge of implementation.

ADVANTAGES OF DATA BASE

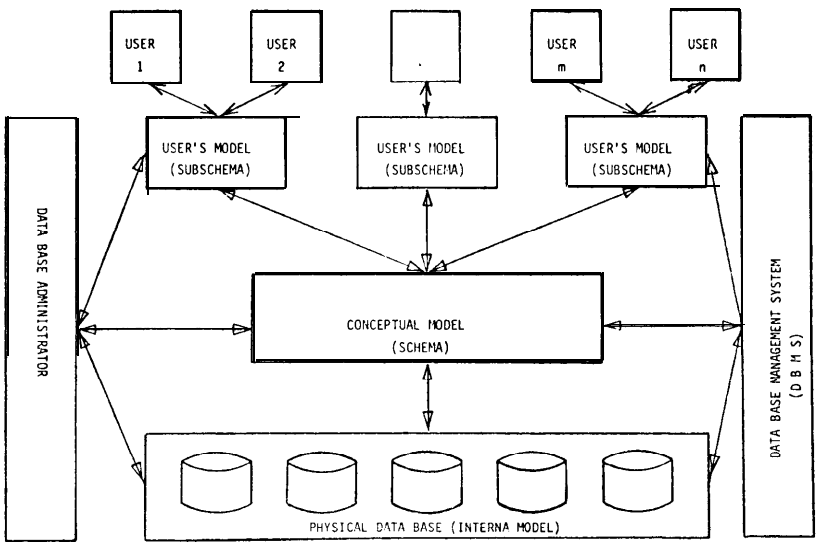
Centralized Control

The result of this is:

- Reduced redundancy.
- Increased consistency in the stored data.
- More effective data - exchange between application programs.
- Easier to maintain data integrity.
- Easier to apply security restrictions for accessing and updating data.

Data Independence

Programs which access the data base don't need to be changed, when the data base formal is updated.



```
GLOBAL_INFO                table
  Field_name               Type        #    First
  NAME                     text        5        1
  CREATION_TIME            text        5        6
  LAST_CHANGE_TIME         text        5       11
  N:ATT                    count       1       16
  P:ATT                    ptr:list    1       17
  P:PROTECT                ptr:table   1       18
  UNIT-FLAG                int         1       19
PRECISION                  int         1       20
  N:WORK_AREA              count       1       21
  P:WORK_AREA              ptr:dir     1       22
  VERSION                  int         1       23
  LEVEL                    int                 24
  *

WORK_AREA                  table
  Field_name               Type        #    First
  NAME                     text        5        1
  N:POINTER                int                  6
  P:POINTER                pointer              7
  *

LOG_DESC                   table
  Field_name               Type        #    First
  NAME                     text        5        1
  ACTUAL_NAME              text        5        6
  ALIAS-FLAG               int         1       11
  N:ATT                    count       1       12
  P:ATT                    ptr:list    1       13
  P:PROTECT                ptr:table   1       14
  N:LOG_PIN                count               15
P:LOG_PIN                  ptr:list    1       16
  N:EQ_GROUP               count       1       17
  P:EQ_GROUP               ptr:list    1       18
  P:INTERNAL_DESC          ptr:table   1       19

  LOG_PIN                  list     1 g/r  expand yes
  Field_name               Type        #    First
  NAME                     text        5        1
  TYPE                     int         1        6
  N:ATT                    count       1        7
P:ATT                      ptr:list    1        8
PARENT_EQ_GROUP                                 9
N E T .                    index       1       10
  *
```

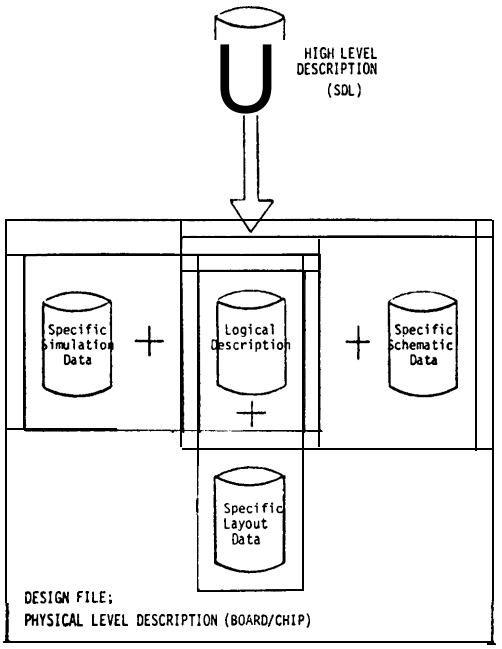
An example of Schema/Subschema language.

WHY NOT USING EXISTING DATA BASE SYSTEMS?

- Existing systems are often tailored for short transitions.
  Engineering transactions are not short.
- Programming language support (COBOL).
- Parallel processing.
- Data integrity.

The Access Routines (Data Sub-language) are:

- Data Base format independent.
- Simple for programming.
- Portable.



A CONCEPTUAL MODEL FOR THE DESIGN PROCESS

BUS ROUTER

by

Tom Blank

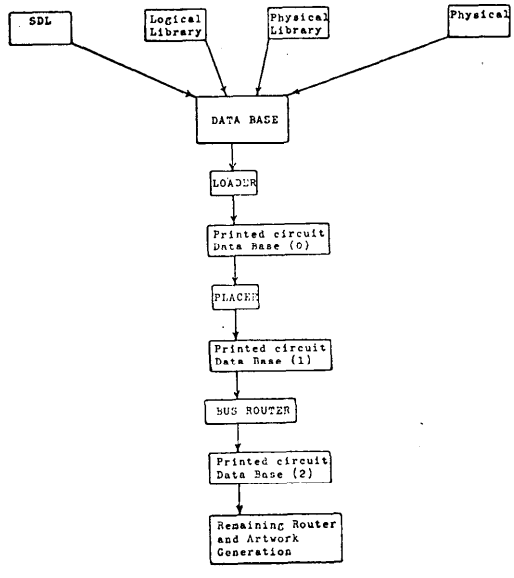
I. Introduction

A. Idea - to devise a printed circuit board routing technique especially for bus structures.

B. Informal bus definition: A collection of nets that have connections on a common group of components.

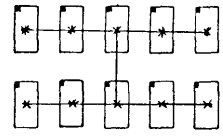
1. Route only DIP and SIP components.
2. Let standard router complete remaining connections.

II. Integration into DA system



III. Implementation

A. Top



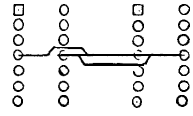
entities: components of a 'super bus'

routing: construction of min. cost spanning tree

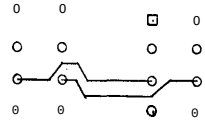
B. Middle

entities: multiple net structures

routing: pattern match



C. Bottom



entities: point to point connection or net.

routing: pattern match

IV. Goal - to incorporate the designer given bus information into the final Printed Circuit board layout.

V. Implications

A. Better PC bd. space utilization.

B. Minimize total routing tins (complete ● ymtcn)

C. Improve percent completion

D. Maximize use of designer input

1. Bus information
2. pattern specification

E. Could be generalized into IC bus routing

A High-Performance Microcomputer

Raster-Scan Graphics System

Andres Bechtolsheim
Computer Systems Laboratory

Applications

Design Automation (VLSI project)
Advanced Text Processing (TEX, Metafont)
As a general departmental display system

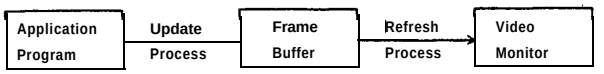
System Architecture

Ethernetbased stations, personal or clustered
centralized file-servers and data bases
remote large-scale computing resources

Station contains

68000 microcomputer with virtual memory
high-performance graphics
keyboard, tablet

Model of a Frame Buffer Graphics System



Goals:

Frame Buffer Size: 1024 by 1024 Bit
Refresh Rate: 64 MBit/sec (non-interlaced monitors)
Update Rate: 16 Mbit/sec (four refresh times)
Data Path Width: 16 Bit

Implications:

Memory Bandwidth: 60 Mbit/sec = 5 MWord/sec
Update Rate: 1 MWord/sec or 1 usec/update
address generation
shifting, masking
frame buffer operation

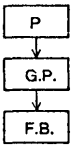
Architectures for Frame Buffer Graphicr

1. Processor-Memory Architecture

Frame Buffer is treated as standard memory
Low performance if operations not microcoded
Consumes significant fraction of main processor bandwidth
To unload memory bus, frame buffer needs to be dual-ported

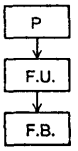
2 Graphics Processor Architecture

Have a subsystem controlled by dedicated processor
Main processor is'unloaded
Performance criteria can be met easily
Functionality limited by Graphics Processor



3. Functional Memory Architecture

Adapt memory organization to Frame Buffer task
Separate Access, Operation, and Control
Provide hardware mechanisms for:
Pixel String Addressability (X, Y, Length)
Raster Operation (Bit-Modification)
Sequential Address Generation in X/Y



The Frame Buffer Operation (RasterOP)

Copy	Bit-Move	Dst ← Src
Paint	Bit-Set	Dst ← Dst OR Src
Erase	Bit-Clear	Dst ← Dst AND NOT Src
Invert	Bit-XOR	Dst ← Dst XOR Src
Copy\	Bit-Move-Not	Dst ← NOT Src
Paint\	Bit-Set-Not	Dst ← Dst OR NOT Src
Erase\	Bit-Clear-Not	Dst ← Dst AND Src
Invert\	Bit-XOR-Not	Dst ← Dst XOR NOT Src

Implementation:

Dst<0:15> ← PLA(Src<0:15>, Mask<0:15>, Mode<0:3>)

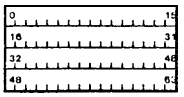
Frame Buffer Memory Organization

64 chips @ 16 kBit =

1,048,576 bits

1024 • 1024

512'512.4

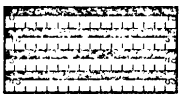


1. Refresh Cycle

Readout 64 bits in parallel

into 64 bit buffer

(64 bits every 1 usec)

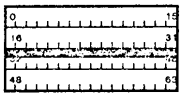


2 Read-Modify-Write Cycle

Readout 16 bits'

Form new data

Write back at same address

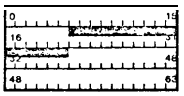


3. Crossing Logical Word Boundaries

Decode each RAS separately

Strobe CAS in parallel

Wire Data Outputs together

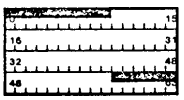


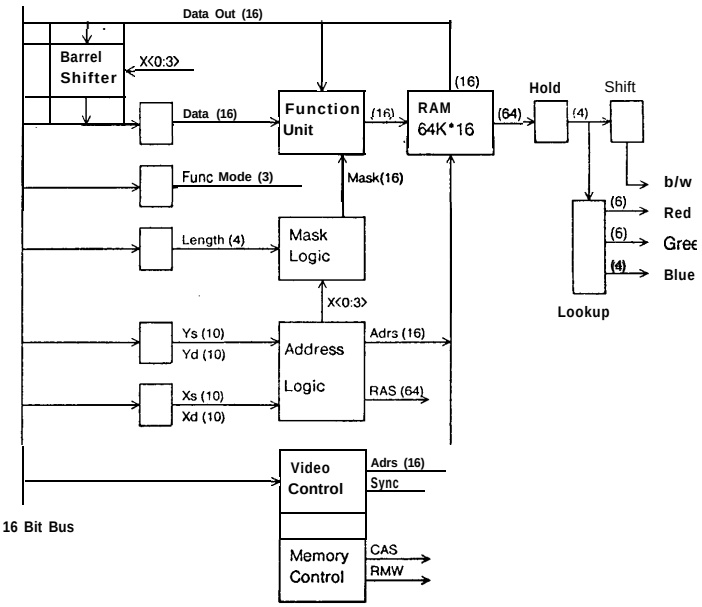
4. Crossing Physical Word Boundaries

Supply two sets of addresses

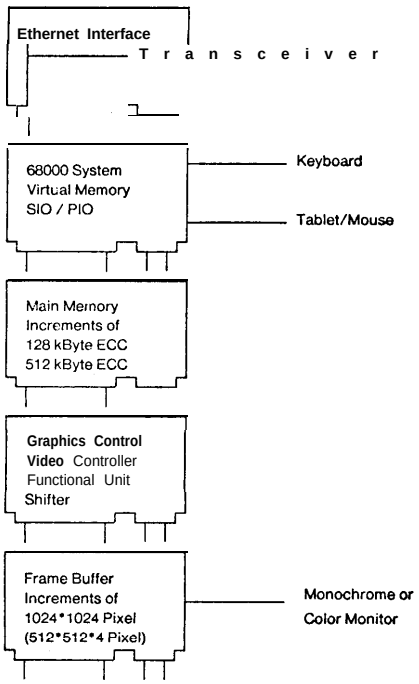
<0:31> ← Address + 1

<32:63> ← Address





Graphics Subsystem Data Paths



Components of the SUN System

CONFERENCE ATTENDEES

Amdahl Corporation

Don Mortimore

AMD

Henry Sun

AM1

Dave Clary
Bob Griffin
Dan Holt
Chi-Song Horng
Bob Kirk
Steve Sapiro

Data General

Jack Crawford
Sabin Head

Digital Equipment

Alain Hanover
Dick Helliwell
Val Pate1

Fairchild

Daniel Fabre
Hem Hingarh
Sanh Srivardhana
Robert Suaya
Dan Wilnai

Four Phase Systems

Carl Hartshorn
Dick Delp

General Dynamics

Len Gaska
Howard Springer
Jim Swenson

Hewlett Packard

Ravi Apte
Dick Dowel1
Jim Lipman
Bill McCalla
Ed Smith
Kamran Elahian
Peter Roth
Rod Price

IBM - San Jose

Gerry Watanabe
Ron Young

IBM - Yorktown Heights

F. H. Dill
Walter Kleinfelder

Intel

Robert Willoner
Todd Wagner
Christopher Goldstein

Microtechnology

Ed Porter
Hung C. Lai
Ming Young

National Semiconductor

Fred Brady
Bill Dawson
Dick Smith
Jackie Tubis
Edward vanBeever

Signetics

Carl Hage

Stanford

Guido Arnout
Ria Simons Arnout
Hugo Deman /Univ. of Leuven
Kim Stevens

Tandem Computers

Al McBride
Michael Kelly
John Barrett
Paul Barnhard

Tektronix

Tom Bohan
Bill Peek