STAN-CS-78-666

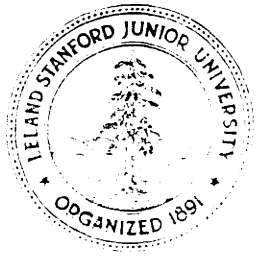
The SCALD Physical Design Subsystem

T.M. McWilliams and L.C. Widdoes, Jr.

Technical Report No. 153

March 1978

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



The SCALD Physical Design Subsystem

T. M McWilliams and L. C. Widdoes, Jr.

Technical Report No. 153

March 1978

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

ABSTRACT

The SCALD physical design subsystem is described. SCALD supports the automatic construction of ECL-10k logic on wire wrap cards from the output of a hierarchical design system. Results of its use in the design of an operational 15-MIPS 5500-chip processor are presented and discussed.

INDEX TERMS: design automation, computer-aided design, Stanford-1 computer, structured design, SCALD system

Sect ion Page

1. Introduction

The S- 1 Structured Computer-Aided Logic Design System (SCALD) has been used to design and implement an operational, 15-MIPS, 5500-chip, ECL-10K processor. The SCALD system has been described in a companion paper, *SCALD: Structured Computer-Aided Logic Design,* and it is recommended that that paper be read first.

SCALD supports a design process which is separated into logical and physical design; the logical design is relatively independent of the details of packaging, but the physical design deals explicitly with packaging considerations. The Macro Expander is used for the logical design, and is applicable across a broad range of technologies and packaging techniques, but the Wire Lister is used for physical design and is specialized to handle a narrow range of component technologies and physical designs essentially only broad enough to include the object machine described in Section 2. This specialization of the Wire Lister was necessitated by a desire to completely implement the object machine in a timely fashion in the context of an initial step in the development of more general tools.

The major goals in the design of the Wire Lister were:

- To allow the engineer to have control over important and difficult physical design decisions, for example, the positions of chips, while freeing him from specifying details which either are not important or can be easily determined automatically. In this regard, the packaging system was chosen to minimize the number of physical design decisions which could not be automated.

- To provide documentation which would support debugging and maintenance of a design specified by a structured macro language.

Section 2 describes the object machine built with SCALD, emphasizing the packaging details. Section 3 explains the global structure of the Wire Lister. Section 4 through Section 8 detail the various subprograms within the Physical Design Subsystem. Section 9 summarizes our experience with the Wire Lister. Section IO outlines our ideas about ways that the Wire Lister should be improved.

2. The Object Machine

SCALD has been used to design an operational processor (the S-1). The S-I is implemented on 12 wire-wrap boards. A wire-wrap board contains 44 50-pin connector feed-through patterns (with all pins uncommitted), and 500 IC patterns each containing 16 DIP pins, 8 SIP pins, 8 ground pins, and two capacitor sockets. A resistor SIP containing 6 100-Ohm resistors tied to a common VTT bus is plugged into each SIP location to provide resistors for termination. The 32-pin IC patterns are packed at a density of one per 0.6 square inches. Special 24-pin adaptors plug in at right angles to the IC patterns. Each board measures approximately 23 inches by 19 inches. Each contains three power planes, used for VCC, VTT, and VEE. Pin 16 of each IC pattern is soldered to VCC, pin 8 to VEE. In IC patterns containing a DIP with multiple grounds, the additional pins are grounded by soldering clips to the VCC plane on the component side of the board.

Identical boards are mounted in pages of four boards each; two boards form one side of a single

page, and two boards form the opposite side. Chips are mounted facing into the channel formed by the two sides of a page. When the spine edge and outside edge of a page are closed, the channel is closed, and fans blow air down through the channel over the chips. Each page is independently hinged from the spine. All connections between pages are made by cables which alternate grounds and signals. Long cables are twisted-pair, while short cables are ordinary flat-cable. 25-pair cables terminated with connectors plug into the connector patterns on the boards. The layout of cables is specified by the design engineer, and obeys rules which allow access to any pin in the machine while the machine is running, without changing the air flow or cabling.

Figure 2.-1 shows the processor cabinet with the pages spread. Figure 2.-2 shows a page open for IC replacement.

All runs (electrically connected networks) are routed without stubs (daisy-chained). The ends of each segment are placed at the same level on the wire-wrap pins. The ECL-10K output may be located anywhere along the length of the run, not necessarily at an end. Each run is terminated at one or both ends with a 100-Ohm resistor. In order to minimize crosstalk, some segments of runs on a board are wired using twisted pair, where the shield wire is either connected to two VCC pins or is connected to the complementary signal. The shield wires in cables at-e used in a similar manner.

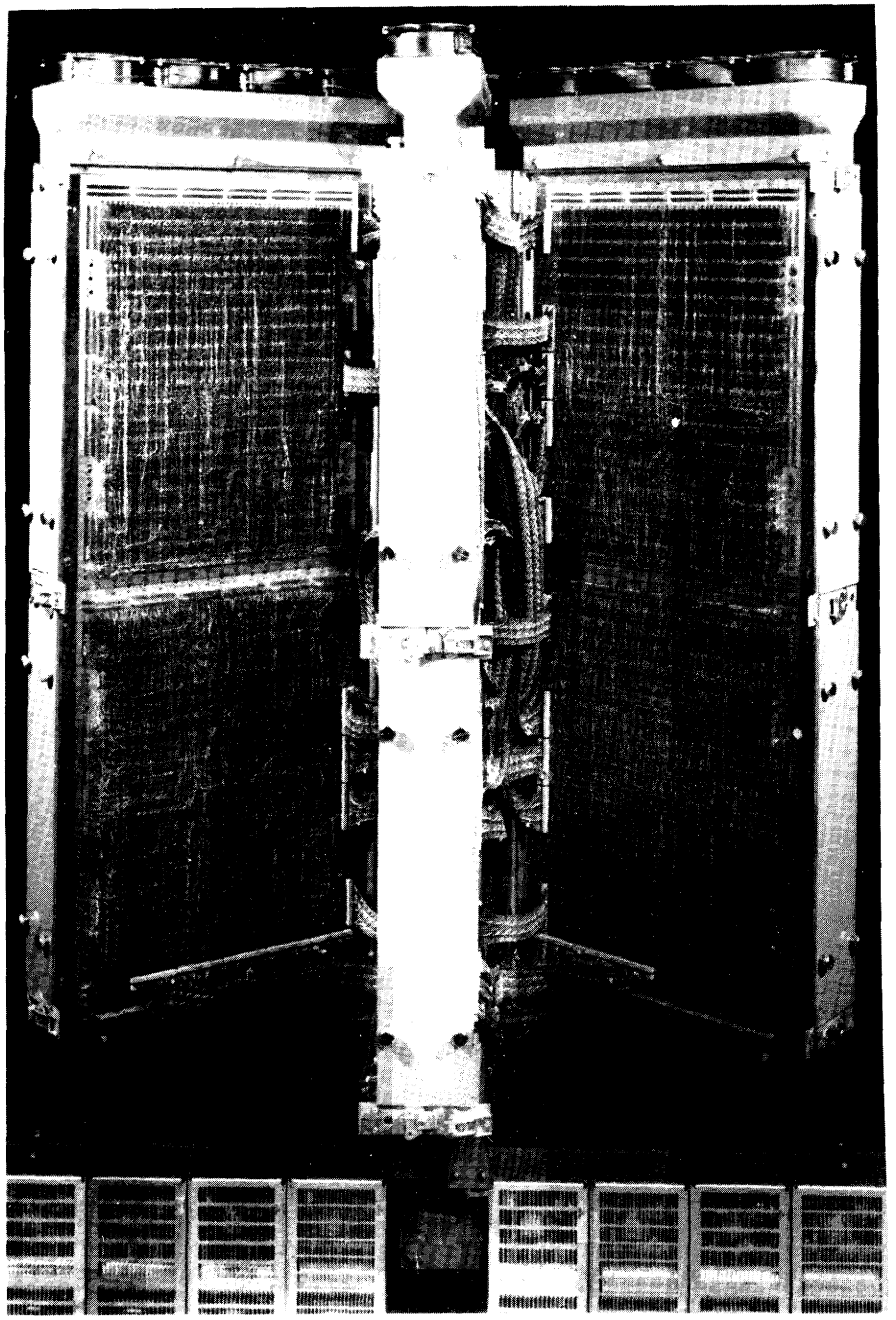


Figure 2.-1
Processor Cabinet

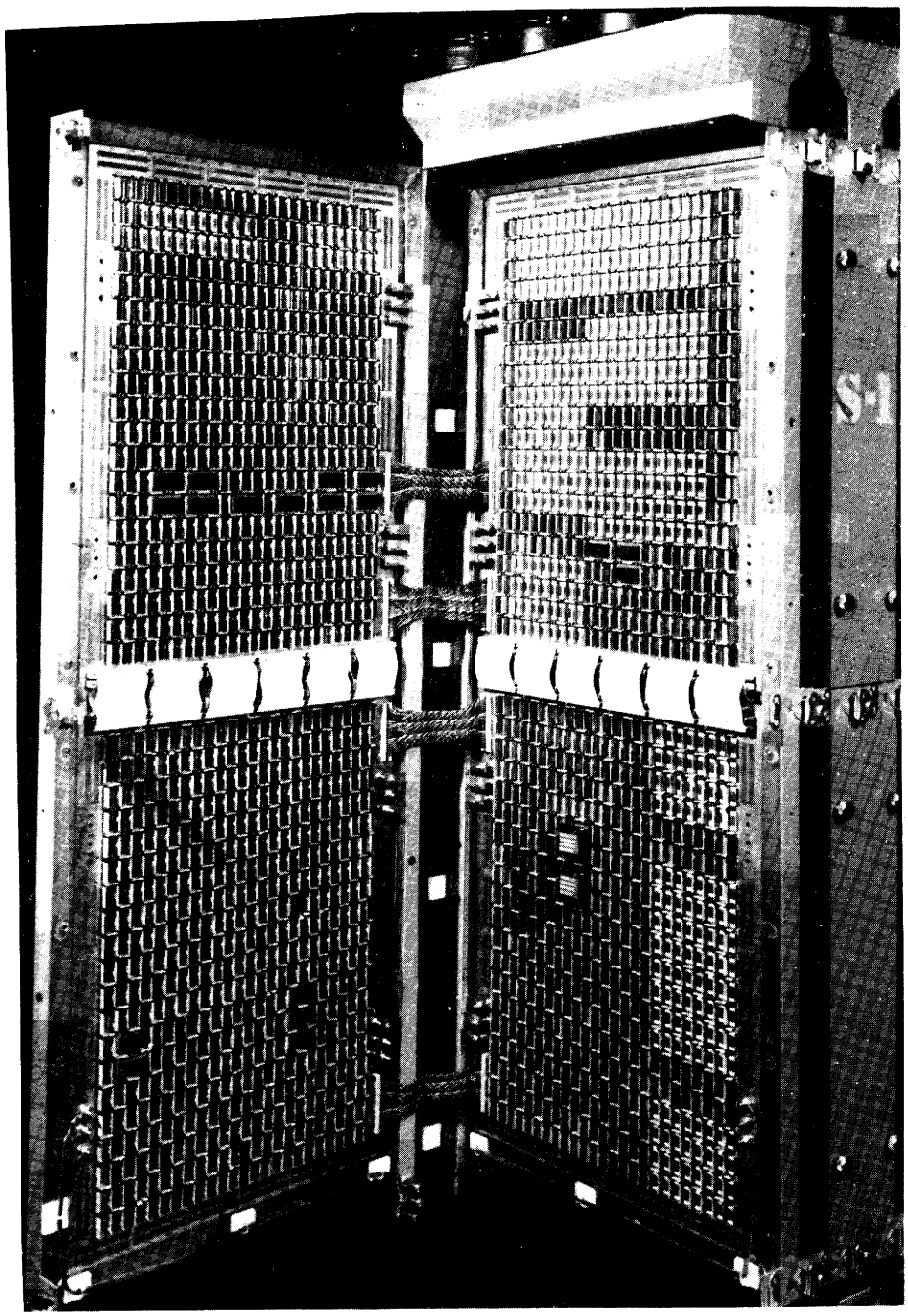


Figure 2.-2
Page Open to Change Chip

3. Wire Lister Structure

Although the assignment of physical locations to terminal path names is actually a function of the Macro Expander, it is described here since it is logically part of the Physical Design Subsystem.

The Macro Expander inputs a hand-generated, block-structured specification of the mapping from terminal path names to physical locations (LAYOUT), and the expansion order is in fact determined by LAYOUT. The Macro Expander outputs a list of the connected points on physical runs, which is input by the router (R). R routes the runs, produces the necessary documentation and wire lists, and outputs a board-state file (NEWBRDS). The programs HINT and ECO (Engineering Change Order) are used to make incremental changes in the design of a constructed machine, BREF produces reference listings for checkout, and TRL simulates the electrical characteristics of runs.

Figure 3.-1 shows the major listings and intermediate files produced and used by various modules of the Wire Lister, and describes the flow of data between parts of the Wire Lister by simply showing the writer and reader of each listing or file. Figure 3.-1 also shows the sizes of the major listings for the S-1 Processor design.

Listing	Writer	Reader	Lines	Contents
DEFS	person	R HINT	2000	Electrical characteristics of ICs.
CBLS	person	R	200	Positions and characteristics of cables.
IOS	R	R	4000	Rssignments of signals to cables.
LAYOUT	person	MRCEXP	4000	Hand layoutspecification.
MAP	MACEXP	person	N/R	Layout naps.
NC	MACEXP	R	10000	Pins of macros which are not connected.
RUNS	MRCEXP	R HINT	70000	Unrouted connect ion list.
SUMM	R	person	12000	Summary and statistics from R run.
OKRUNS	R	person	140000	Runs w i thou t errors.
BADRUNS	R	person	N/R	Runs w i th errors.
NULLRUNS	R	person	N/R	Runs with no outputs.
SORT	R	person	27800	Sorted lists of buses.
NEMBRDS	R	ECO BREF	70000	New state of all boards.
OLDBRDS	R	HINT ECO	70000	Old state of all boards,
TRL IN	R	TRL	N/A	Characterizat ions of runs.
TRLOUT	TRL	person	N/R	Graphical wave forms .
PINS	BREF	person	70000	Signal name connecting to each pin.
BCHK	BREF	person	40000	Checkout listings.

Figure 3.-1
Physical Design Subsystem Files

4. Layout Language

Once the basic design is expressed in the macro language, the next step is to partition it onto boards, and to assign locations on the boards to specific chips. In a conventional design system where all of the logic is drawn out, the logic to go on a specific board is drawn on a separate set of prints, and each component is then labeled with a physical location. In the SCALD Design System, this technique is not possible because not all of the logic is drawn out, and for that matter, a given drawing may generate logic which goes on a number of boards.

The layout language is used to specify the mapping between terminal path names and physical locations. Figure 4.-1 shows the mapping for the logic in Figure 4.-2, which is put onto two boards, and consists of 85 chips.

There are two basic statements in the layout language, the WITH statement and the assignment statement. The WITH statement is used to specify that the logic inside a given macro call is to be laid out next. It has the general syntax:

```
WITH <logical-location> AT <physical-location>;

  <WITH and assignment statements to place contents of macro>

END;
```

where the <logical_ location> is the logical location label given in the macro call, and the <physical_ location> specifies a base location for all the logic laid out inside the WITH statement. The <physical-location> can specify a base board, row of a board, column of a board, and section of a chip, or can default any of these to be zero. For example, the location string "B1R2C3S4" specifies a. base board of 1, row 2, column 3, and section 4. The board, row, column, and section can be given in any order.

The assignment statement is used to place either a terminal component or a macro. If it is used to place a terminal component, then it is of the form:

```
<logical. location> = <physical- location>;
```

When it is used to layout a macro, then it is of the form:

```
<logical location> = <physical_ location-vector>;
```

where <physical_ location_ vector> specifies a vector of board, row, column, and section tuples, which are assigned to each component generated by the macro, in the order they are generated. The macro is expanded in a depth first expansion, where the macro and component calls in a given macro are evaluated in alphabetical order according to their logical location labels. The location vector is specified by the use of an iterative notation. A given dimension, where a' dimension is either a board, row, column, or section, can be specified by the form "L(F,T,B)", where L is a "B", "R", "C", or "S", standing for board, row, column, and section, and F, T, and B are in tegers, which generate a sequence of values from F to T by B. The increment may be defaulted to 1 by using the alternate form "L(F,T)", and a single value may be produced by writing "L" followed by the value. A number of dimensions can then be specified with a space between them, where they will be evaluated from right to left, to form a vector of location tuples. For example, the expression "B(1,2) R3 C(5,3,-1)" evaluates to a vector which has 6 location tuples, ((1,3,5),(1,3,4),(1,3,3),(2,3,5),(2,3,4),(2,3,3)), where each tuple specifies a board, row, column, in

that order. In addition, a sequence of these location specifiers can be given with a comma between them, which are evaluated from left to right. For example, this last expression could have been written as "B1 R3 C(5,3,-1), B2 R3 C(5,3,-1)".

The first thing that is laid out in Figure 4.-1 is the macro "36 BIT ALU 10181", which has a logical location label "A", and a base location at board 1. The next macro laid out is then the "1B REG 10176", with logical location label "RI". The clock line of that register is driven from the signal "REG CK BUF", which is generated from a three-output gate, and has three physical versions. The layout file then specifies that the common clock ("CC") pin of the 10 176 should use version 1 of "REG CK BUF". In the layout specification of the "36B 10174" multiplexer, the first 18-bits are laid out in the high form, and the last 18-bits are laid out in the reverse form, using both the true and complementary outputs on the 10101 gate to drive the select li ties.

```
WITH A AT B1;     !36 BIT ALU 10181
  0 = R1 C3;
  1 = R2 C3;
  2 = R1 C6;
  3 = R2 C6;
  4 = R1 C9;
  5 = R2 C9;
  6 = R1  C12;
  7 = R2 C12;
  8 = R1 C15;
  A = R3 C3;
  B = R3 C7;
  C = R3 C11;
  D = R3 C15;
END;     !OF 36 BIT ALU 10181

R1=B1 R3 C2 SO
      : cc = 1;
R2 = B1 R 3 C(4,6) S(0,5),
     B1 R3 C (8,10) S(0,5)
     : CC = 2*18,3*18;
R3 = B1 R5 C(1,11,2) S(0,3),
     B1  R5 C(13,15) S(0,3);
G1 = B1 R5  C8  SO;
G2 = B1 R5 C 6 S(0,1);
G3 = B1 R3 C1 SO;
t1 = B1 R 4 C(1,15) S(0,1),
     B1 R3 C(12,14) S(0,1)
     / H18 L18;

WITH CTL AT B2;     !PROCESSOR CONTROL
  R = R(1,3) C(1,8)
       : A0 = 1*8, 2*8, 3*8
       : A1 = 1*8, 2*8, 3*8
       : A2 = 1*8, 2*8, 3*8
       : A3 = 1*8, 2*8, 3*8
       : A4 = 1*8, 2*8, 3*8
       : A5 = 1*8, 2*8, 3*8
       : A6 = 1*8 2*8, 3*8
       : A7 = 1*8, 2*8, 3*8;

  C = C11 R(1,2);
  G1= R 3 C9 S0;
  G2= R3 C10 S1;
  G3=  R(1,2) C(9,10) S(0,1);
END;     !OF PROCESSOR CONTROL

END;
```

Figure 4.-1
Layout for Simple Processor

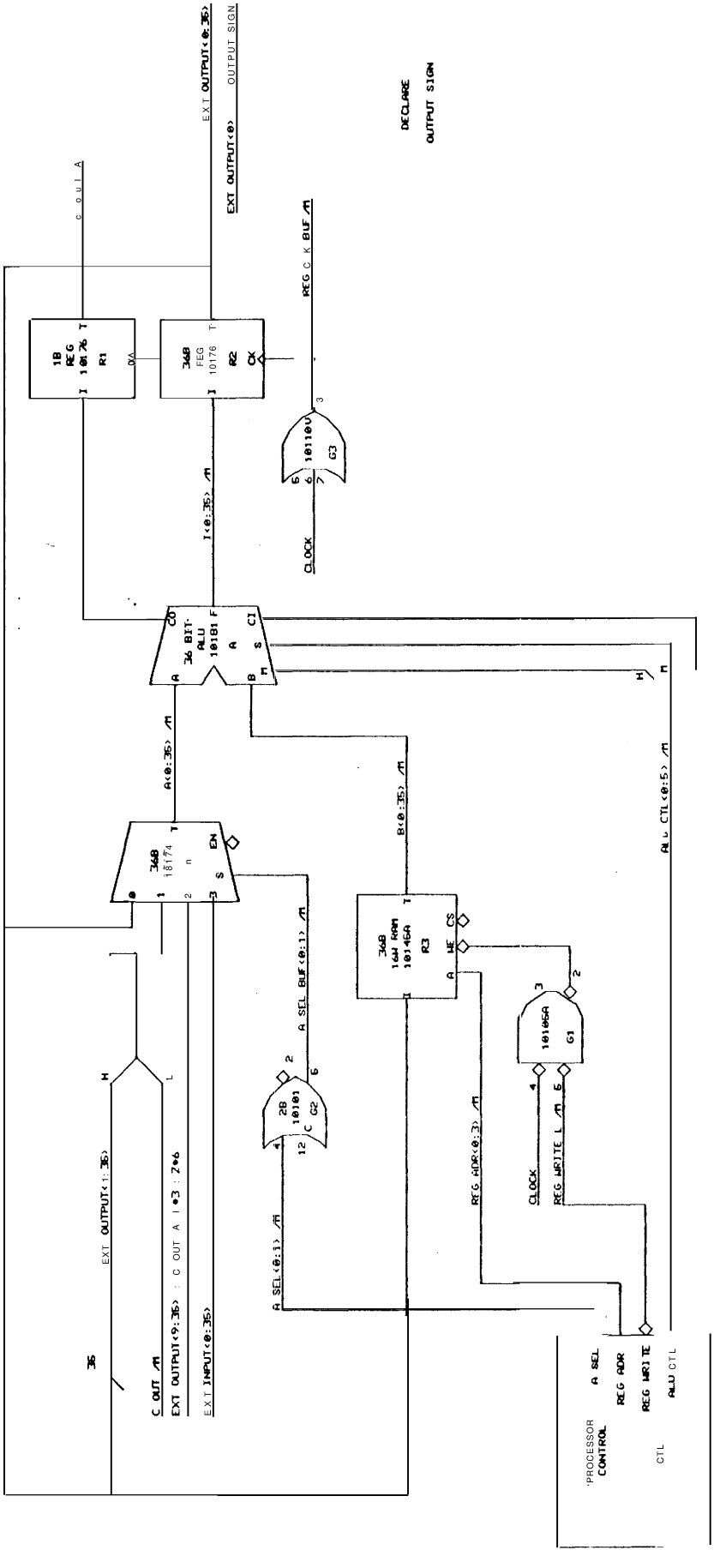


Figure 4.-2
Simple Processor Macro

5. Router

The major function of the Router (R) is to sequentially route the runs contained in the list RUNS output by the Macro Expander, and to produce all necessary wire lists and associated documentation to allow the debugging and maintenance of the object machine.

A sample RUNS file is shown in Figure 5.-i. RUNS is organized in to *buses,* which are vectors of runs, in order to allow the documentation to be organized into buses. Each run consists of a set of connected stops. Buses are sorted in alphabetical order, and runs within a bus are in alphabetical order. Each stop represents a physical pin in the format

<TPN>{-<SEC>}:<SKT>:<PART>.<PIN>{/<VER>}

where TPN is the component's terminal pathname, SEC (optional) is the section number within the component, SKT is the socket name, PART is the physical part name, PIN is the pin number within the section, and VER (optional) is the version number for the run connected to this pin. The section number is assumed to be zero if none is given.

The RUNS file does not make reference to cables or I/O pins. A separate file, IOS, which can either be hand-generated or automatically-generated, assigns I/O pins to runs. An example of IOS is shown in Figure 5.-2. IOS assigns one connector pin of a cable to a run; assignment of the connector pin on the opposite end of the cable is implied by the CBLS file. If a run crosses multiple boards, then IOS assigns one I/O pin per board crossing, in the order that the boards are to be crossed.

R can allocate I/O pins automatically, constrained only by the availability of cables connecting the appropriate boards (specified in the CBLS file). During automatic allocation, if a run needs I/O pins and has none assigned in IOS, R will select the best (in a measure of total run length) set of I/O pins which connect the appropriate boards, permanently allocate those to the run, and output the allocation to the IOS file. The search order is random over the remaining free I/O pins in order to minimizing bunching.

I/O pin allocation, and indeed all other functions of R are purely sequential and involve essentially one pass over the RUNS file; no decision is ever retracted.

The DEFS file is used to make the transformation from the pin number in RUNS to a DIP pin number. An example from the DEFS file is shown in Figure 5.-3. For each IC type, DEFS defines the power pins, and for each section shows the mapping from pins of the section to DIP pin numbers. For each pin, DEFS shows the electrical characteristics including leakage currents and rise times, the pin type (eg. ECL input, ECL output, differential input, VBB, not connected), and a field defining a mnemonic pin function. One pin from several different sections may map to the same DIP pin number.

Since 24-pin DIPS plug into three IC patterns (on an adaptor which in additions covers the positions reserved for two SIP resistor packs), R contains a mapping function which maps 24-pin DIP pin numbers and socket locations onto true physical pin numbers and socket locations. Mappings for adaptors of other sizes or shapes can be easily added.

R routes each single-board section of each incoming run optimally according to a cost function which is presently total wire length (under XY segment routing), but can be varied considerably, even to produce the minimax arm length (where an output divides a run into two arms) which is a good approximation of minimax delay. The computational complexity

and memory requirement of the routing algorithm is $O(N^2*2^N)$, where N is the number of stops. For the S-I Design, a 2-megabyte IBM/370-168 was used to optimally route runs of up to 11 stops. The use of an algorithm which finds an optimal route according to the cost function has proven to be extremely prudent. Since the class of valid outputs of the algorithm can be easily understood, it is convenient to make modifications of the algorithm for special purposes. For example, because stubbing is not allowed, if a run has two I/O pins on a board (ie. if a run goes through a board), then the I/O pins must be on the ends of the on-board section of the run. Such on-board sections are obtained merely by constructing a non-symmetric distance function. Furthermore, when routing to minimize changes to a board which has already been wrapped, the cost function simply diminishes slightly the cost of a segment which connects two previously connected pins.

After routing, R allocates twisted pairs using a parameter-driven algorithm. All segments of a run longer than X inches are twisted, and no segments under Y inches are twisted (where for the S-I, X is 6, and Y is 2). Depending upon the distance of a segment from the low-impedance output and the segment's length, the algorithm chooses to twist segments between X and Y inches.

Non-differential runs which are twisted require ground pins to ground the shield wires. R simply locates the ground pin which is nearest the associated signal pin and allocates it, removing it from future consideration in the ground-pin search. Shield wires of adjacent twisted-pair segments are grounded on the same pin. Differential runs which are twisted do not need ground-pin allocation. Ground pins are also allocated for the shield wires of segments which connect to I/O pins, since alternate wires of cables must be grounded. Where shield wires are too short for twisting according to the twisted-pair algorithm, they are left untwisted.

After routing a run, R allocates terminating resistors automatically. A resistor is always connected to the far end of the longest arm of a run. If an output divides a run into two arms both of non-zero length, then a resistor is placed at the end of each arm, resulting in a 50-ohm DC load for each output. If heavy loading occurs near the output on a single-armed run, then a resistor is connected to the output to decrease the fall time. Two 100-Ohms resistors are found and allocated for each side of a differential run, which is forced to be single-armed. Resistor allocation is similar to ground-pin allocation; the closest unallocated resistor is chosen and allocated. Since resistors are always placed at the ends of runs, the segment lengths to resistors do not affect delays to inputs.

In the cases of both resistors and ground pins, the basic philosophy was to provide enough surplus resources on the boards that a simple algorithm with no backup would be capable of generating small search distances. In practice, the simple algorithm was quite satisfactory, yielding average orthogonal search distances under 0.5 inches for ground pins and resistors.

After routing a run, R makes an estimate of the worst case delay to any input of the run by assuming that the run is a perfectly terminated transmission line with distributed capacitance equal to the intrinsic distributed capacitance of the wire-wrap wire plus the lumped capacitive loads corresponding to the inputs and outputs. Under these assumptions, the approximate delay is given by

$$D = L*T0*SQRT(1+CL/(L*CD))$$

where TO is the unloaded delay per unit length, L is the length, CL is the total lumped capacitance, and CD is the distributed capacitance per unit length. This approximation has proven to be quite satisfactory for screening runs, with typical relative errors of the order of 20% (consistently high).

R outputs the file TRLIN describing runs selected by the screening algorithm based on estimated delay. TRLIN is used by TRL in simulating the complete electrical behavior of the runs.

R checks for many syntactic and semantic errors, most of which are also detected at higher levels in the design process. R does not place restrictions on the loading of runs, since the designer is expected to make all decisions about whether. the actual reflections shown in the output of TRL are acceptable based on his knowledge of the criticality of each signal.

R produces many output files which allow automatic construction and which facilitate the debugging of the processor. The major outputs are the board-state file (NEWBRDS) and the listing of error-free runs (OKRUNS). OKRUNS describes the routing and characteristics of each run in the machine, including inter-board segments of runs (see Figure 5.-4). NEWBRDS is used to allow incremental changes in a constructed machine, as explained below.

```
$BUS=CTL :BR ADR<0:7>;
BR ADR<0>=                    CTL.R#1:2A1:MB7042.15,
                              CTL.C.0.X:2A11: 10916.11;
BR ADR<1>=                    CTL.R#2:2A2:MB7042.15,
                              CTL.C.0.X:2A11:10016.10;
BR ADR<2>=                    CTL.R#3:2A3:MB7042.15,
                              CTL.C.0.X:2A11:10016.9;
BR ADR<3>=                    CTL.R#4:2A4:MB7042.15,
                              CTL.C.0.X:2A11:10016.7;
BR ADR<4>=                    CTL.R#5:2A5:MB7042.15,
                              CTL.C.1.X:2B11:10016.11;
BR ADR<5>=                    CTL.R#6:2A6:MB7042.15,
                              CTL.C.1.X:2B11:10016.10;
BR ADR<6>=                    CTL.R#7:2A7:MB7042.15,
                              CTL.C.1.X:2B11: 10616.9;
BR ADR<7>=                    CTL.R#8:2A8:MB7042.15,
                              CTL.C.1.X:2B11:10016.7;

$BUS=CTL :BRANCH ALW;
BRANCH ALW=                   CTL.R#9:2B1:MB7042.15,
                              CTL.G2-1:2C10:10105A.4;

$BUS=CTL :BRANCH NEG;
BRANCH NEG=                   CTL.R#10:2B2:MB7042.15,
                              CTL.G1:2C9:10104A.4;
```

Figure 5.-i
RUNS File Input to Router

```
TOP :A SEL <
      0          = 1B2  -24;
      1          = 1B2  -30>

TOP :ALU CTL <
      0          = 1B2  -26;
      1          = 1B1  -20;
      2          = 1B1  -28;
      3          = 1B1   -38;
      4          = 1B1   -56;
      5          = 1B2  -22>

TOP :OUTPUT SIGN <
                 = 2T2  -50>

TOP :REG ADR <
      0          = 1B1   -40;
      1          = 1B1   -42;
      2          = 1B1   -44;
      3          = 1B1  -46>

TOP :REG WRITE L <
                 = 1B2  -36>


$END;
```

Figure 5.-2
Example of IOS File

```
$DIP 10101 !FAIRCHILD!

!                          TYPE  USE       TYP MA        MAX MA

MAC 1 PWR
1  1                        0.0  VCC       0             0
16 16                       0.0  VCC       0             0
8  8                       -5.2  VEE       20.           26.

!                          TYPE  USE    HL PF    LOW MA HI MA ----RISE----!

MAC 4 10101
4  4  7  13 10             EI    I        5.0    0.001  0.265
12 12 12 12 12             EI    IC       12.    0.001  0.550
5  5  6  9  11             EO    T        5.0                  1.5  2.2  3.3
2  2  3  15 14             EO    T_L      5.0                  1.5  2.2  3.3
```

Figure 5.-3
Example of Chip Definition in DEFS

```
TOP                  : A<34>                                    7PF  4IN  4BW  0.8NS  2ST  0X  0ER   415LN
                     ----------------------------------------
  (B1 R1 C15)  A15-  2R        Z2          *R100                1.7 IN
  (B1  R1  C14) A14-15  Z1  A2      18      10181      7.0 PF  2.7 IN          A.8.X
  (B1  R3  C14) C14-     2     T     2      10174      5.0 PF                  M#35

TOP                  : A<35>                                    7PF  4IN  4BW  0.8NS  2ST  0X  0ER   417LN
                     ----------------------------------------
  (B1  R1  C15) A15- 3R        Z2          *R100                1.2 IN
  (B1  R1  C14) A14-  2  Z1  A3      21      10181      7.0 PF  2.7 IN          A.8.X
  (B1  R3  C14) C14-15     T     2     10174     S1     5.0 PF                  M#36

TOP                  :A SEL<0>                                  5PF  24IN  4BW  3.8NS  2ST  1X  0ER   420LN
                     ----------------------------------------
  (B1  R5  C6 ) E6 - 5R        21          *R100                0.6 IN
  (B1  R5  C6 ) E6 - 4  E6 -13G 22 I       4 10101      5.0 PF 17.5 IN TWP      G2#1
  (B1  J   B2 ) J31-24 J31-23              *CONN                2.0 IN CBL
  (B2  J   T2 ) J5 -24  J5 -23  Z1         *CONN                3.6 IN
  (B2  R3  C6 ) C6 -15  C6 -16     T    15 MB7042      8.0 PF                  CTL.R#22

TOP                  :A SEL<1>                                  5PF  24IN  5BW  3.8NS  2ST  1X  0ER   422LN
                     ----------------------------------------
  (B1  R5  C6 ) E6 - 7R        Z1          *R100                0.8 IN
  (B1  R5  C6 ) E6 - 7  E6 -10G 22 I       4 10101   S1 5.0 PF 17.0 IN TWP      G2#2
  (B1  J   B2 ) J31-30 J31-29              *CONN                2.0 IN CBL
  (B2  J   T2 ) J5 -30  J5 -29  Z1         *CONN                3.7 IN
  (B2  R3  C7 ) C7 -15  C7 -16     T    15 MB7042      8.0 PF                  CTL.R#23
```

Figure 5.-4
OKRUNS Listing Output from Router

6. ECO Subsystem

SCALD allows incremental design changes in the hierarchical drawings to be reflected in actual hardware updates in a constructed machine. In short, the drawings are edited, then SCALD runs, producing a new set of documentation including an unwrap list and a wrap list which update the machine to the design represented in the new-drawings.

The current machine state is always retained in the board-state file, NEWBRDS (see Figure 6.-1). NEWBRDS contains a record of each DIP, ground clip, single wire, and twisted-pair wire on each board of the machine. NEWBRDS thus forms a permanent, readable record of the physical machine state which is independent of the original input files, and of the programs which processed those files; knowledge of the machine state is never compromised by changes in SCALD.

When changes to a board are made in the laboratory without running the design system, they are carefully entered into a log book, and later edited into the board-state file.

Because R allocates resistors and ground pins automatically, processing runs sequentially in the order of the RUNS file, it is conceivable that a slight change in the drawings, even the change of a signal name, would result in the reassignment of a large number of resistors and ground pins. To avoid this problem, R inputs hints which are generated by the program HINT on the basis of the previous board state and the current RUNS. Thus when R routes runs, it examines the set of hints which specify the resistors, ground pins, routes, and levels used in the previous board state, and R minimizes the number of changes required to implement the new design. All automatic decisions made by R are either absolutely repeatable independent of signal names and input ordering, or are strongly directed by hints based on previous decisions.

When the new board state has been completely defined, the wrap list, unwrap list, and associated documentation are generated by an independent program which simply compares the old board state and the new board state.

```
 STATE =    1;
BRD  1    DIP:  10101      , E 6 ;
BRD  1    DIP:  10105      , E 8 ;
BRD  1    DIP:  10110      , C 1 ;
BRD  1    DIP:  10145A     , E 1 ;
BRD  1    DIP:  10145A     , E 3 ;
BRD  1   CLIP:  C 1- 1   ;
BRD  1  CLIP:  C  1-15  ;
BRD  1   CLIP:  C 2- 1   ;
BRD  1   CLIP:  C 3- 1   ;
BRD  1   CLIP:  C 4- 1   ;
BRD  1   CLIP:  C 5- 1   ;
BRD  1   CLIP:  C 6- 1   ;
BRD  1   CLIP:  C 7- 1   ;
BRD  1   CLIP:  C 8- 1   ;
BRD  1   CLIP:  C 9- 1   ;
BRD  1    THP:   19.5 SUG Z2 , E 6- 4 , E 6-13G, J31-24 , J31-23 , TOP :A SEL<0>
BRD  1   BARE:    2.6 SUG Z1 , E 6- 5R,          E 6- 4 ,          TOP :A SEL<0>
BRD  2   BARE:    5.4 SUG Z1 , J 5-23 ,          C 6-16 ,          TOP :A SEL<0>
BRD  2   BARE:    5.6 SUG Z1 , J 5-24 ,          C 6-15 ,          TOP :A SEL<0>
```

Figure 6.-1
Example NEWBRDS File

7. BREF Subsystem

The BREF Subsystem inputs a board state file and produces various reference listings for quality control and debugging. Specifically, it produces a listing which shows the signal name of the run connected to each pin, it produces listings for clip checking, continuity checking, checking the resistance of runs to ground, and counting the wraps on pins.

8. Transmission Line Analysis

R outputs descriptions of selected runs to TRL, the transmission line analysis program, which prints on the line printer the graphical waveform of a signal edge on each such run (see Figure 8.-1). The selection of runs to be considered for output to TRL is parameterized and is based on the estimated delay of the run. In the S-I development, only one run per bus was output; because the layout was done by hand in a very regular manner, and therefore individual runs within a bus are quite similar.

The run to be simulated is described as lumped resistive or capacitive (but not both) loads connected in a daisy-chain by segments of known propogation delay and impedance. The TRL analysis is an event-driven simulation. The gate output voltage waveform is described by the exponential:

$$V = C^*(1-e^{-At})$$

where typically "C" is 0.8 Volts and "A" is 0.7/ns. The output waveform is propagated *algebraically* to the first load, which *algebraically* reflects part of the energy and transmits part of the energy. Both the reflected and transmitted wavelets are followed in the same way as the original wave was followed, thus the procedure is recursive. Any wavelet with amplitude less than a threshold parameter is ignored. After all wavelets are less than the threshold, the voltage at any point P at any time T can be evaluated by summing the transmitted waveforms evaluated at P and T. TRL prints the voltage waveform at each node along the run. Note that all wavelets are collected and simplified symbolically until the final evaluation.

For reasons of numerical stability, it was found that an effective way to determine which reflections and transmissions should be followed was to assume a sinusoidal output waveform in an initial pass (thus all resistive and capacitive loads simply attenuate when reflecting and when transmitting) and to follow the same reflections and transmissions in the exponential analysis which yields the final waveform amplitudes.

Our experience in debugging the object machine has shown that the accuracy of these simulation results are within the tolerances of the input parameters such as segment impedance and load capacitance.

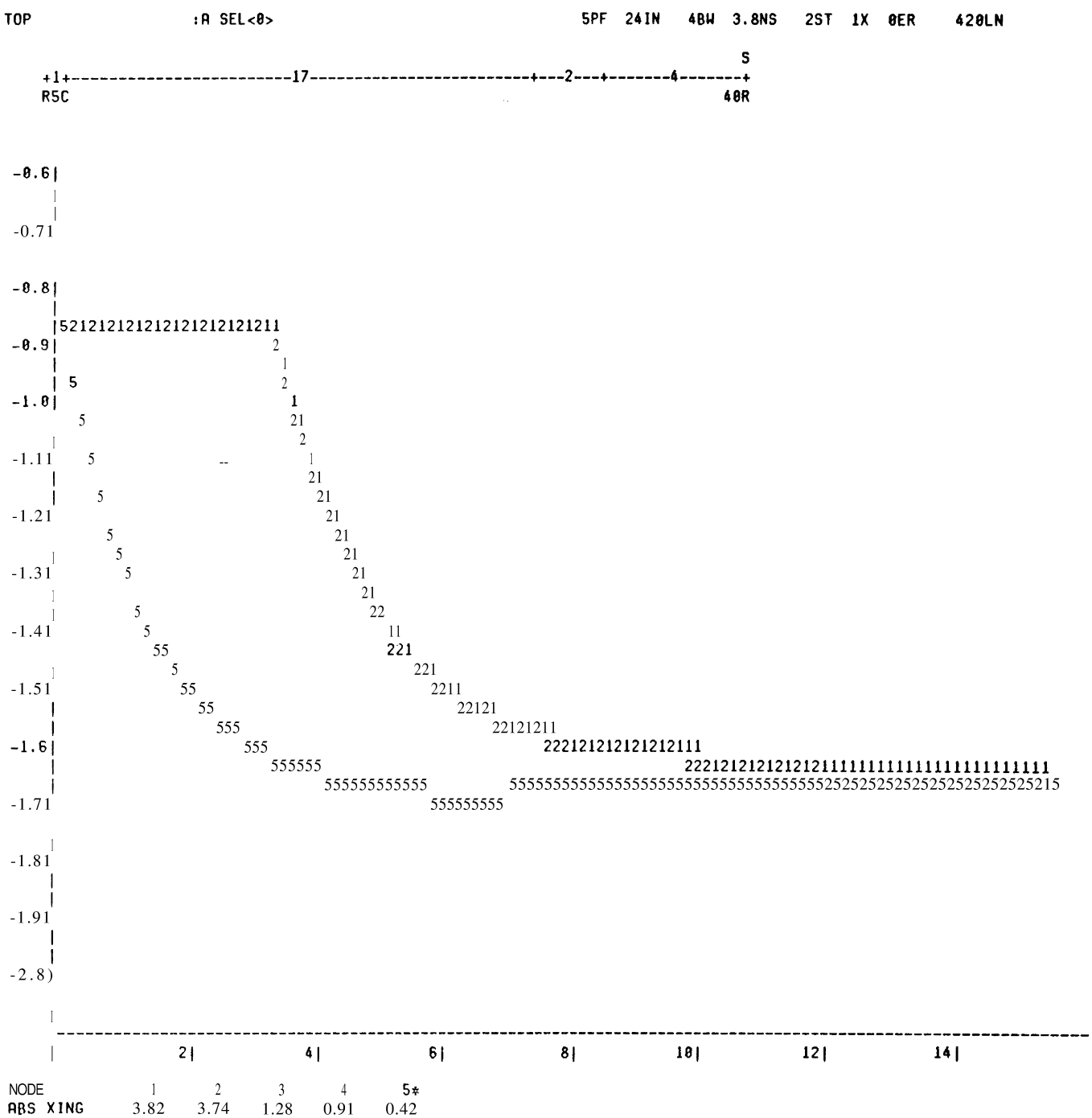


Figure 8.-1
Example TRL Output

9. Experience

The experience in using SCALD to design the S-I has indicated that SCALD is very effective in reducing the time required to implement a large-scale processor. The entire processor design was completed in 24 man-months, while the partitioning and layout required an additional 3 man-months. Debugging the hardware with the SCALD-generated listings has proceeded very well, and the lack of a conventional print set with all of the logic drawn out proved not to be a problem. The ability to design correct logic conferred by the intrinsic clarity of the drawings was verified when the first major portion of the machine (1600 chips) worked at the designed speed ('70 ns per cycle) after the elimination of only 6 minor logic bugs, all involving no more than a single chip.

10. Extensions and Improvements

We have identified the following areas as being candidates for improvements in the Design System:

- A more general language is needed for describing the details of technology and physical packaging. Knowledge of these details is currently largely built into the structure of a 15000-line PASCAL program, and that program would need to be extensively modified to accomodate multiwire ECL-10K, for example.

- The Physical Design Subsystem could be made to drive any of a number of available CAD systems which handle the physical design in various technologies and packaging systems.

- Automatic layout of sub-modules is desirable, but this capability needs to be carefully considered for its effects on the generation of ECOs.

- A version allocator needs to be designed which would eliminate the need to assign physical versions to pins by hand, but this capability also needs to be carefully considered for its effects on the generation of ECOs.

- Feedback from the Physical Design Subsystem to the source language level (the hierarchical drawings) would be highly desirable. In particular, the average and range of the delays of a vector of signals could be automatically entered onto the source drawings to aid in design verification.

11. Conclusions

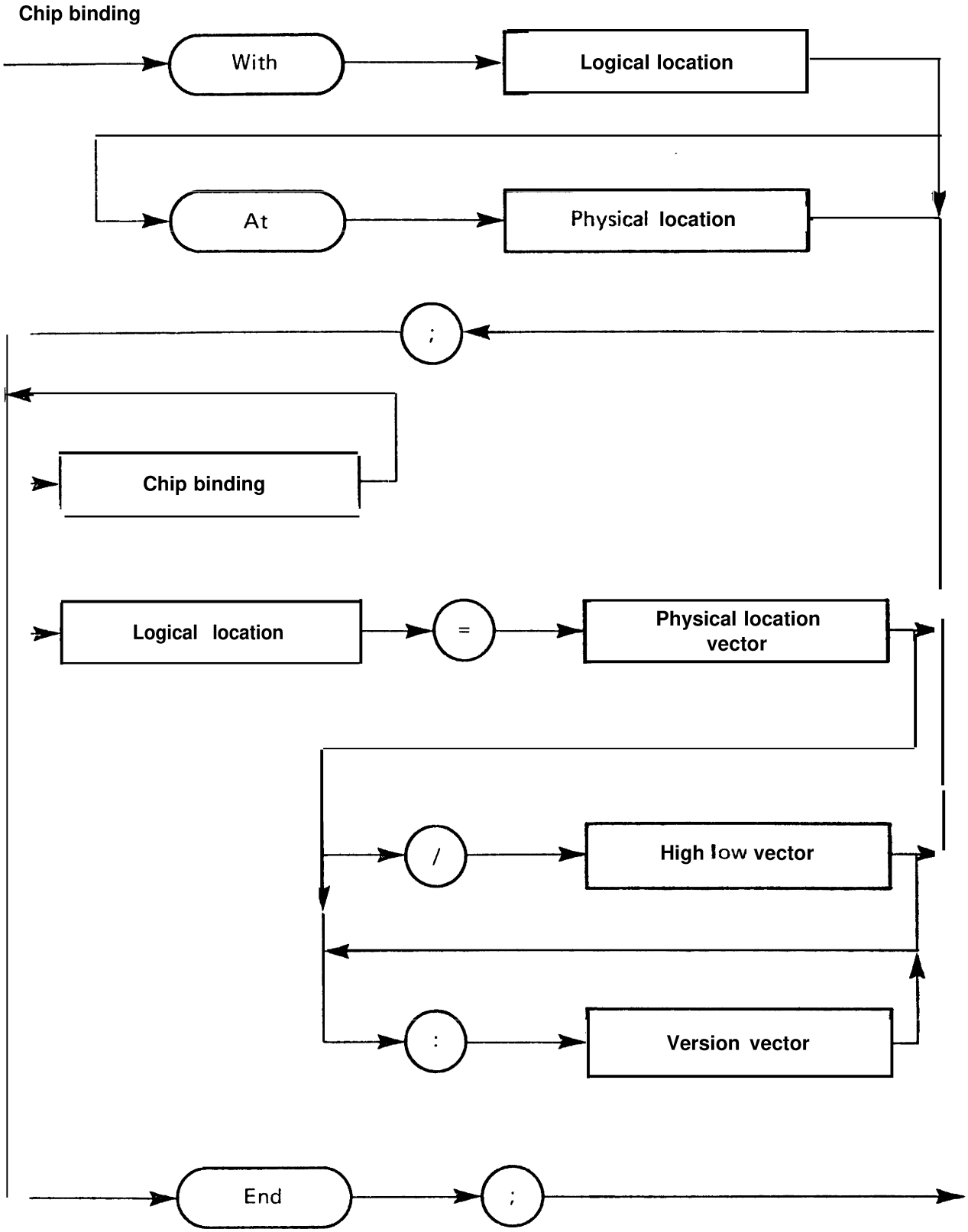
The SCALD Physical Design Subsystem facilitates the implementation and maintenance of a structured design expressed in terms of a Macro Language. The Physical Design Subsystem handled all details of the physical design for the S-l Mark I Processor except layout of components, assignment of versions, and positioning of cables. incremental changes can be made in the structured drawings and are reflected in minimal wrap/unwrap lists. The Physical Design Subsystem is presently specialized to the particular packaging and technology of the S-l Mark I Processor, but can be generalized to allow the rapid creation of additional particular physical' design subsystems.

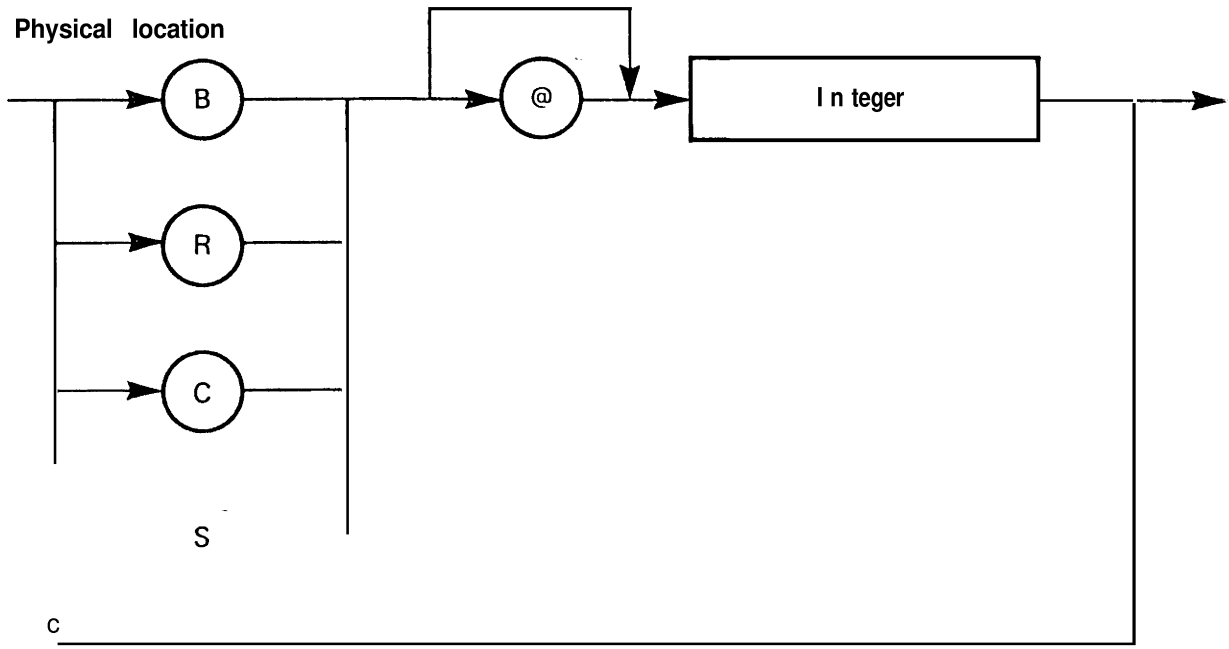
12. Acknowledgements

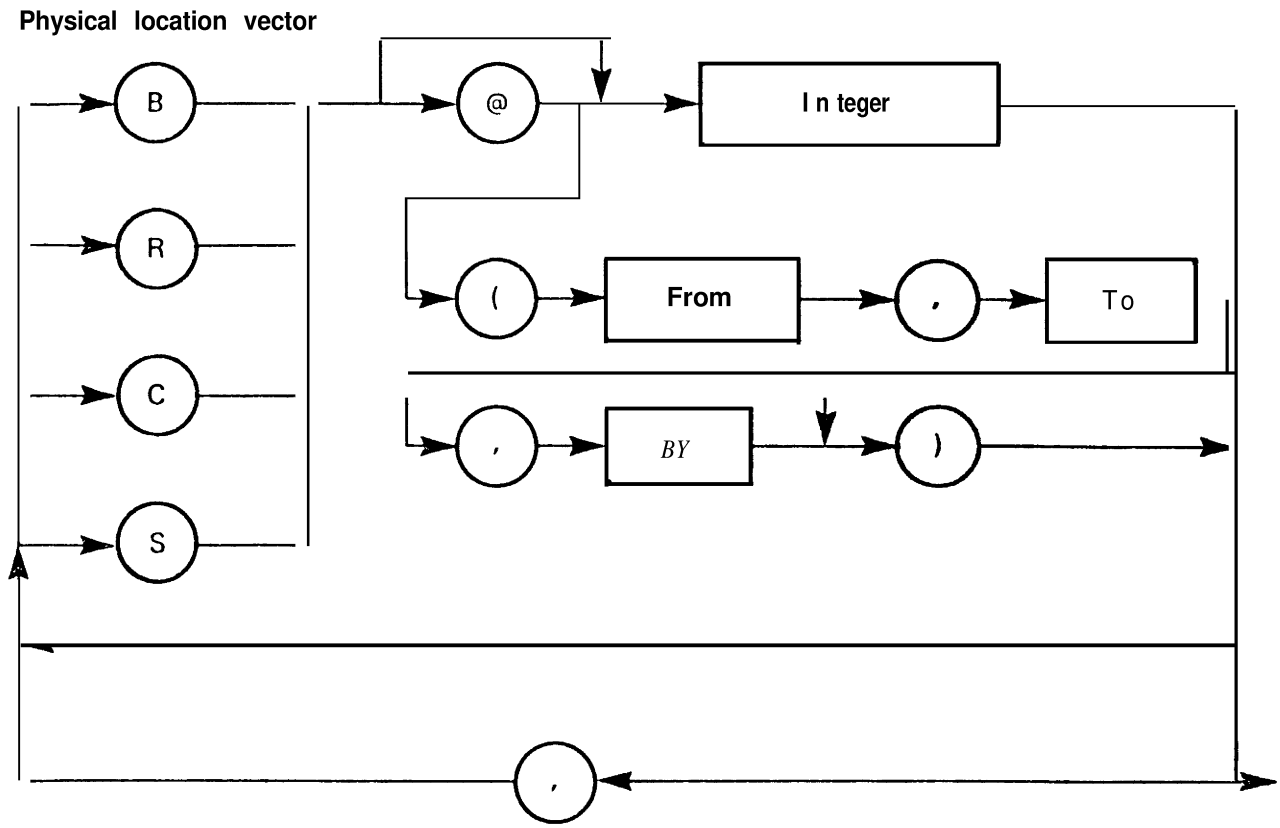
We wish to acknowledge crucial support for this research which has been received from the Office of Naval Research via ONR Order Numbers N00014-76-F-0023 and N00014-77-F-0023 to the University of California Lawrence Livermore Laboratory (where the authors are members of the professional staff), from the Computations Group of the Stanford Linear Accelerator Center supported by the Energy Research and Development Administration under contract EY-76-C-03-0515, and from the Stanford Artificial Intelligence Laboratory. We also wish to gratefully acknowledge the support of our graduate studies which has been extended by the Fannie and John Hertz Foundation. We greatly appreciate the constant encouragement and support which we received from Forest Baskett, Lowell Wood, and Bill vanCleemput throughout this work, the support Sassan Hazeghi provided in writing and responsively maintaining an excellent PASCAL compiler at the Stanford Linear Accelerator Center, and the work Rick McWilliams did in writing TRL and the SUDS-to-Macro-Language Translator.

A1. Syntax of Layout Language

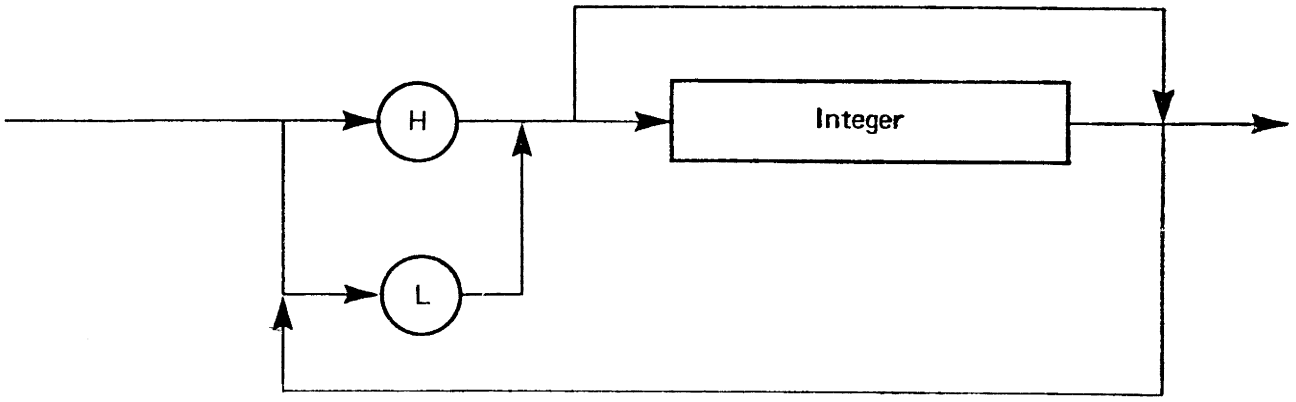
The following syntax diagrams give a detailed definition of the syntax for the text form of the SCALD layout language.


Chip binding
With
Logical location
At
Physical location
;
Chip binding
Logical location
=
Physical location vector
/
High low vector
:
Version vector
End
;

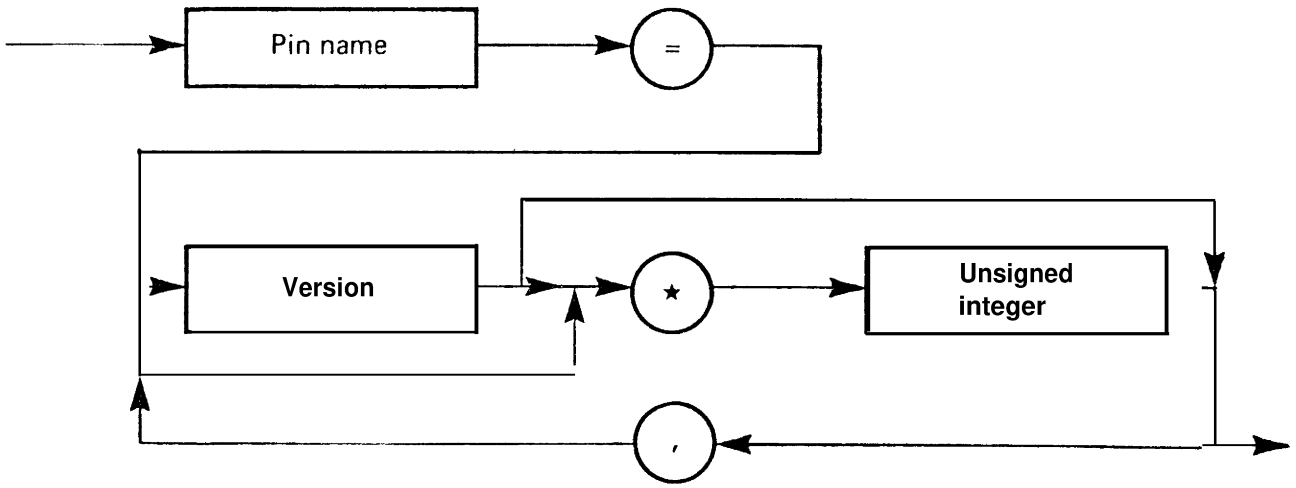

Physical location
B
R
C
S
@
I n teger
c

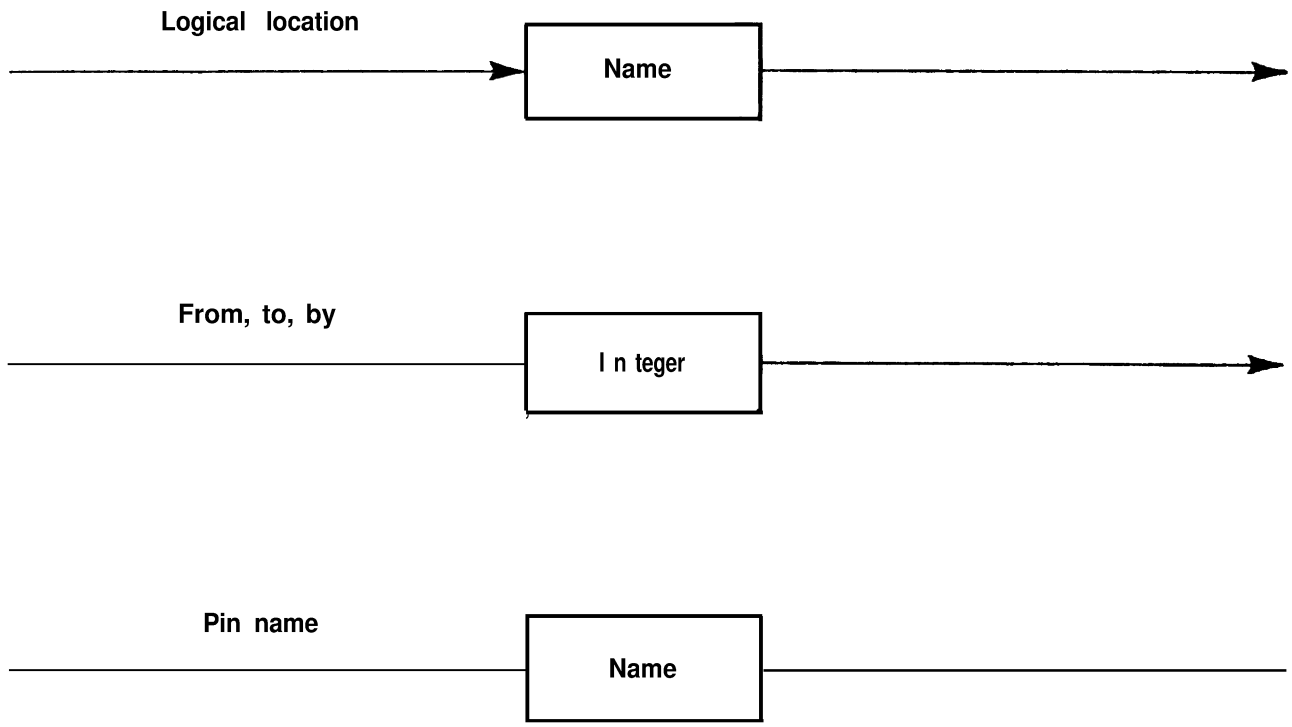

Physical location vector
B
R
C
S
@
I n teger
(
From
,
To
,
BY
)
,

High low vector



Version vector




Logical location
Name
From, to, by
I n teger
Pin name
Name