# SCALD: Structured Computer-Aided Logic Design

Thomas M. McWilliams and Lawrence C. Widdoes, Jr.

Computer Science Department
Stanford University
and
Lawrence Livermore Laboratory
University of California

## Abstract

SCALD, a graphics-based hierarchical digital logic design system, is described and an example of its use is given. SCALD provides a total computer-aided design environment which inputs a high-level description of a digital system, and produces output for computer-aided manufacture of the system. SCALD has been used in the design of an operational, 15-MIPS, 5500-chip ECL-10K processor.

### 1. Introduction

SCALD (Structured Computer-Aided Logic Design) is a graphics-based design system which allows digital systems to be designed in a hierarchical manner. SCALD's main goal is to reduce the amount of time required to design large digital systems, by allowing the designer to express his design on the same level that he thinks about it, freeing him from the task of actually drawing out all of the logic and creating a wire list. Designs expressed in this high-level notation become much more understandable, both for designers and for maintenance engineers.

SCALD's second important goal is to allow designs to be recompiled rapidly when new circuits become available, allowing designs to repeatedly take maximal advantage of the exponential rates of advance currently characterizing the semiconductor industry. This goal is achieved by expressing a design in terms of high-level modules, which in the future may be implemented as single ICs. In practice, considerable work may still be required to update a design to incorporate recent technology advances, but the required effort is likely to be much less than if the design were not expressed hierarchically.

SCALD has been used to design a very high-performance processor, the S-1, shown in Figure 1-1, which is a 15 MIPS, 5500-chip ECL-10K machine. This design experience has been very favorable; the entire processor was designed and implemented with two man-years of effort.

To provide a vehicle for the presentation of SCALD, the design of a very simple processor has been carried through the Design System. The top level of this design is represented in Figure 1-2 and Figure 1-3; The processor consists of a register file of 36-bits by 16 words, a 4-input, 36-bit multiplexer, a 36-bit arithmetic-logic function generator, and a 36-bit accumulator. The microsequencer shown in Figure 1-3 controls the simple processor; it consists of an 8-bit counter and a control store of 23 bits by 256 words.

### 2. System Overview

SCALD takes as input a high-level description of a digital system, and produces output for the computer-aided manufacture of the system on wire wrap boards. Figure 2-1 shows the three main modules in SCALD. Input to the system is through the Stanford University Drawing System (SUDS) Graphics Editor [Helliwell 1972], which allows drawings to be entered directly on a graphics terminal.

All parts of SCALD except the SUDS Graphics Editor are written in PASCAL, and are therefore highly transportable. The Macro Expander (M) reads the output of SUDS, along with the hand-generated layout specification, expands the drawings into a connection list, and generates a number of listings to aid the designer. The Wire Lister (W) inputs the connection list, a chip definition file, and an old board state, and produces a wrap/unwrap list, a run list, describing each *run* (electrically connected net) in detail, a new board state, and a number of summaries and statistics. Changes can be made to a constructed system by editing the source drawings and then running the entire SCALD System again; the Wire Lister then reads the old board state, which specifies how the system was constructed before the change, and outputs a new board state and a wrap/unwrap list, the execution of which updates the system to correspond to the new drawings.

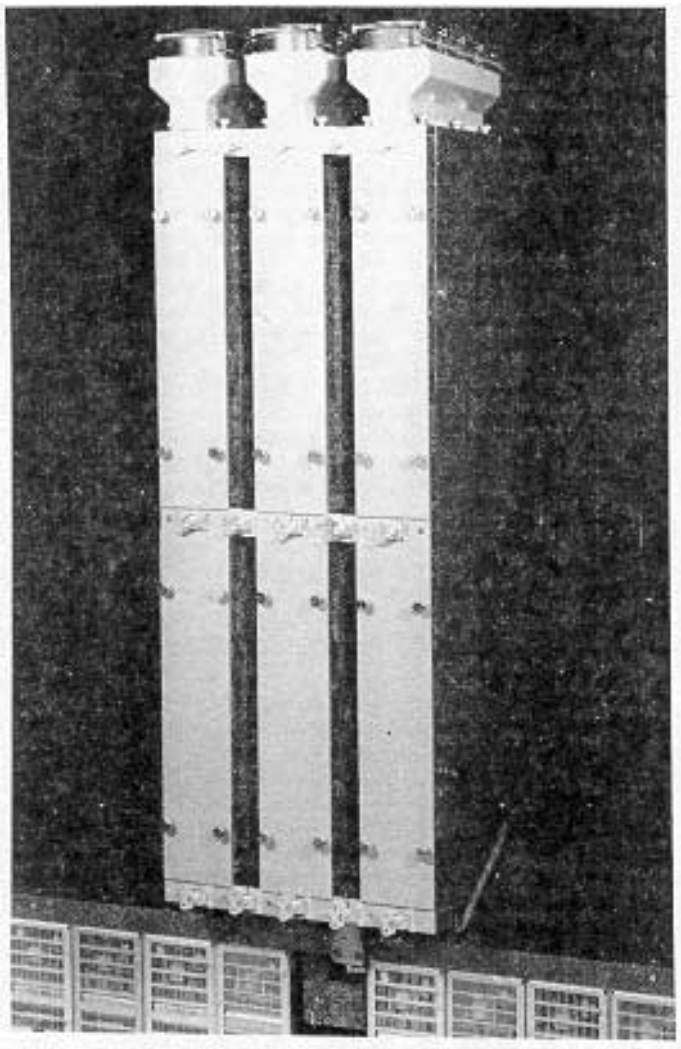

Figure 1-1
S-1 Processor

| Module | Input | Output |
|---|---|---|
| Graphics Editor | Keyboard | Text description of drawings |
| Macro Expander | Text description of drawings<br>Hand layout | Macro call structure<br>Macro definition listing<br>Signal cross reference<br>Connection list |
| Wire Lister | Connection list<br>Chip definitions<br>Old board state | Wrap/unwrap list<br>Run list<br>New board state<br>Summaries and statistics |

Figure 2-1
Main Modules in SCALD

### 3. Graphics Editor

The hierarchical logic diagrams are entered into the SCALD system using the Stanford University Drawing System (SUDS) Graphics Editor. Examples of logic diagrams drawn with SUDS are shown in Figure 1-2 and Figure 1-3. The SUDS system is an interactive graphics editor, written in assembly language, which runs on a PDP-10 equipped with a refreshed graphics terminal. The program is controlled with keyboard commands, and the cursor is controlled either by a light pen, or by the keyboard.

The first step in using the SCALD Design System is to create a library of common body definitions, such as those shown in Figure 3-1. The user has complete control over the appearance of a body. Once a library of bodies is created, drawings can use those bodies. A body is positioned in a drawing by giving commands at the keyboard which place it at the location specified by the cursor. After bodies have been thus positioned, commands are given which connect the bodies with lines, and place text on the lines. In general, the user has complete control of the positioning and interconnection of bodies in a drawing. SUDS allows *bused-through* lines, that is, lines which connect to a body and then pass invisibly under the body (horizontally or vertically) to exit on another side, and this capability was found to be extremely useful in decreasing the clutter in drawings. SUDS includes many commands which allow bodies and lines to be easily moved, and has a macro facility which allows repetitive structures to be drawn very quickly. Hard copy of drawings is available from either a Xerox Graphic Printer or a plotter.

Dominant resource utilization by the SUDS Graphics Editor during the design of the S-1 amounted to 30 hours of KL-10 compute time, and 1000 hours of graphics-terminal time spread over a period of one year.

The SUDS Graphics Editor outputs the drawings represented in a text Macro Language. This text Macro Language serves as input to the Macro Expander. It would be possible to use a different Graphics Editor to supply input to the SCALD Design System if another program were written to translate the drawings into the text Macro Language.

### 4. Macro Language

A design in SCALD consists of a set of macro definitions (macros), which are expanded, starting from a distinguished top-level macro and continuing downward until no macro remains which has a definition (ie., all remaining macros are available devices), to generate a wire list and all the necessary associated documentation for the system being designed (the *object machine*). These macro definitions are entered directly into the SCALD data base using the SUDS Graphics Editor.

Macro calls within a macro definition are represented by appropriate bodies from the Body Library, and may be passed various parameters, the values of which differ from call to call. Connections between bodies are made with lines, which represent signal vectors and may be named; identically named signals are implicitly connected. Signal vectors may be passed as parameters to macros; the formal name of a signal-vector parameter passed to a macro is shown on the macro body where the actual signal vector connects to the macro body

A macro can be called one or more times from other macros, but cannot be called recursively, since SCALD allows no conditional expansion. The ability to define once a function which is used many times greatly reduces the overall design time for a large object machine; it reduces redundancy and thereby facilitates verification and increases changeability.

The use of macros (rather than bodies representing available devices) in the definition of the data path in an object machine results in a great reduction in the number of drawings required and in the density of bodies on the drawings. The object machine's non-repetitive (control) logic can be then distributed throughout the data path, placing it near the logic it controls, thereby enhancing the overall understandability of the logic.

On the macro level, SCALD does not distinguish between inputs and outputs of devices or macros. It is not until after the macros are expanded that the SCALD System checks for runs with an illegal number of inputs or outputs. In general, each run must have exactly one output, unless permission is granted in the drawings (by the use of a *Wire-Or Body*) for multiple outputs.

#### 4.1 Signal Expressions

SCALD allows signals to be grouped together to form a signal vector, represented by a single (possibly named) line in the drawings. For example, the signal vector named "A<0:15>" represents 16 signals. The general notation for the name of a signal vector is "NAME<I:J>", where NAME is a string and I and J are integers; by convention, I is the high-order bit number, while J is the low-order bit number. Signal vectors can be concatenated by writing a colon between their names. A signal vector can be replicated by suffixing its name with an asterisk and a number; for example, the expression "A<0:3>*3" is equivalent to the expression "A<0:3>:A<0:3>:A<0:3>". Holes can be generated in a signal vector by use of the special signal "Z", a one-bit signal which is never connected in the hardware; in the ECL-10K logic family used in the design of the S-1 processor, an open input is a logic zero.

Figure 1-2 contains an example of a complex signal expression: "EXT OUT<9:35> : C OUT /M *3 : Z*6" represents a 36-bit vector, where the high-order 27 bits are "EXT OUT<9:35>", the next 3 bits are the local signal (see Section 4.2) "C OUT", and the low-order 6 bits are not connected.

SCALD also understands primitive *merger bodies*. Figure 1-2 contains a two-merger and a three-merger, which concatenate, respectively, two signal vectors and three signal vectors, forming larger vectors. The two-merger is connected to the "I" input of the 4-input multiplexer, and the three-merger is connected to the "M", "S", and "CI" inputs of the "36 BIT ALU 10181". Mergers are used where concatenation is needed, but preservation of the explicit connectivity of the drawing is desired.

#### 4.2 Signal Types

Signals in a macro definition can be of three types: parameters, locals, and globals. Subject to limitations of scope, signals of the same name in any are implicitly connected throughout all macro definitions.

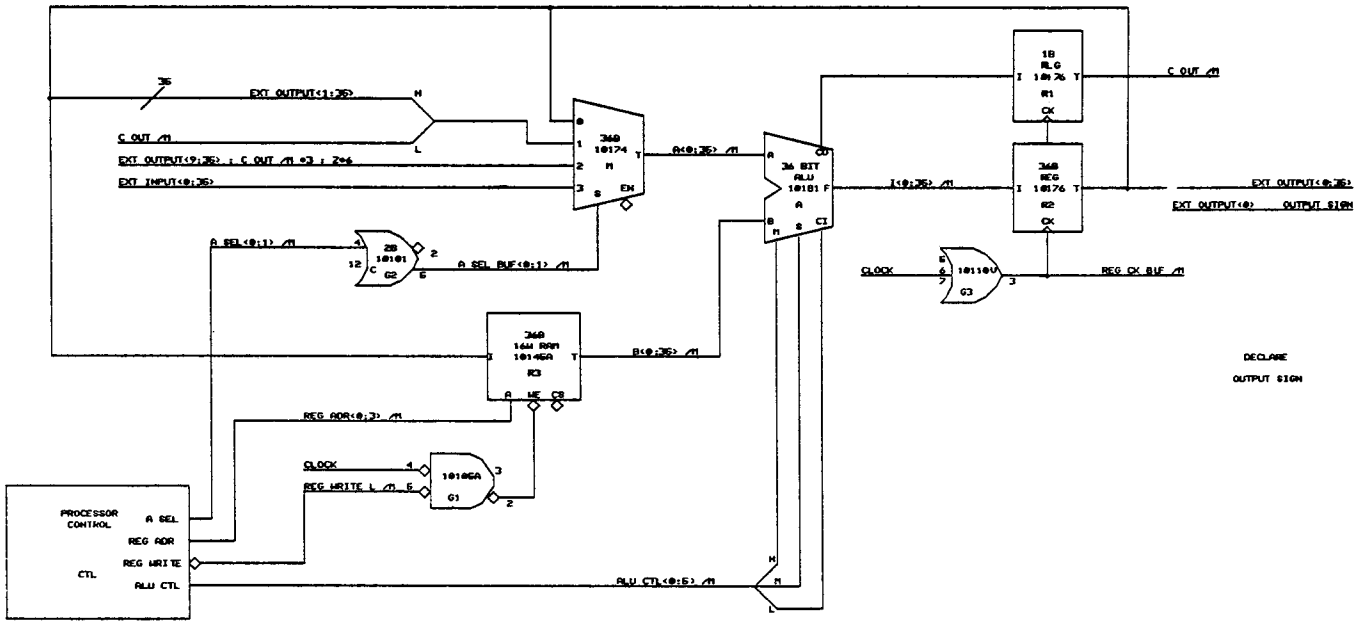


Figure 1-2
Simple Processor Macro

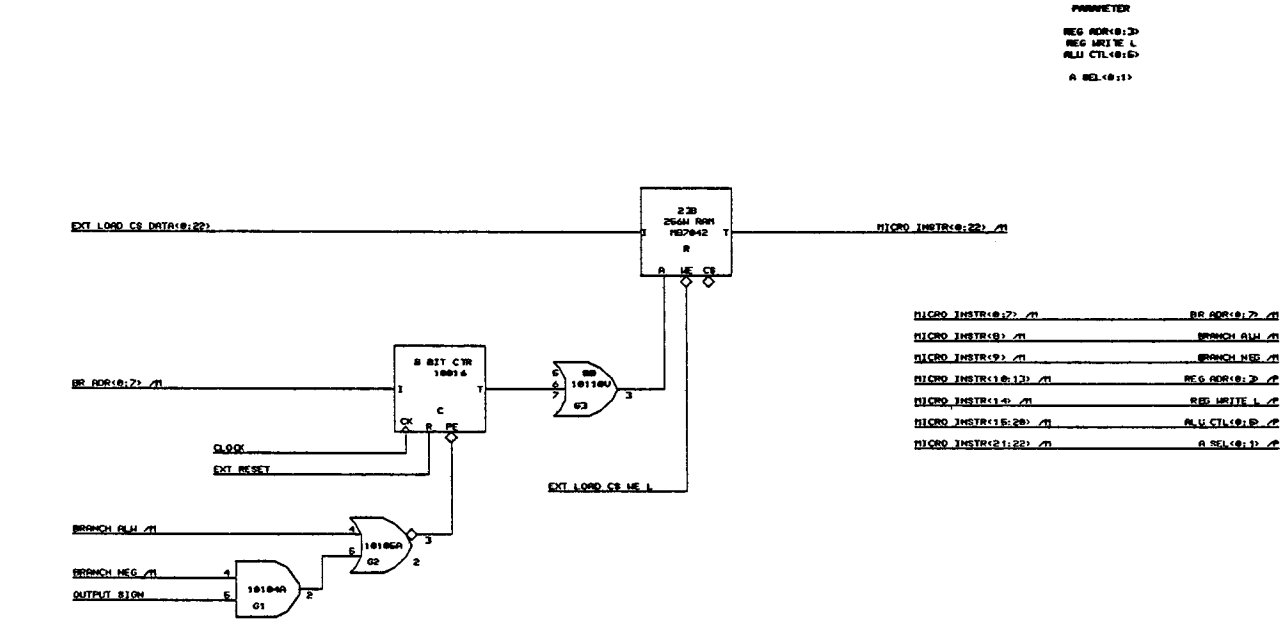


Figure 1-3
Processor Control Macro

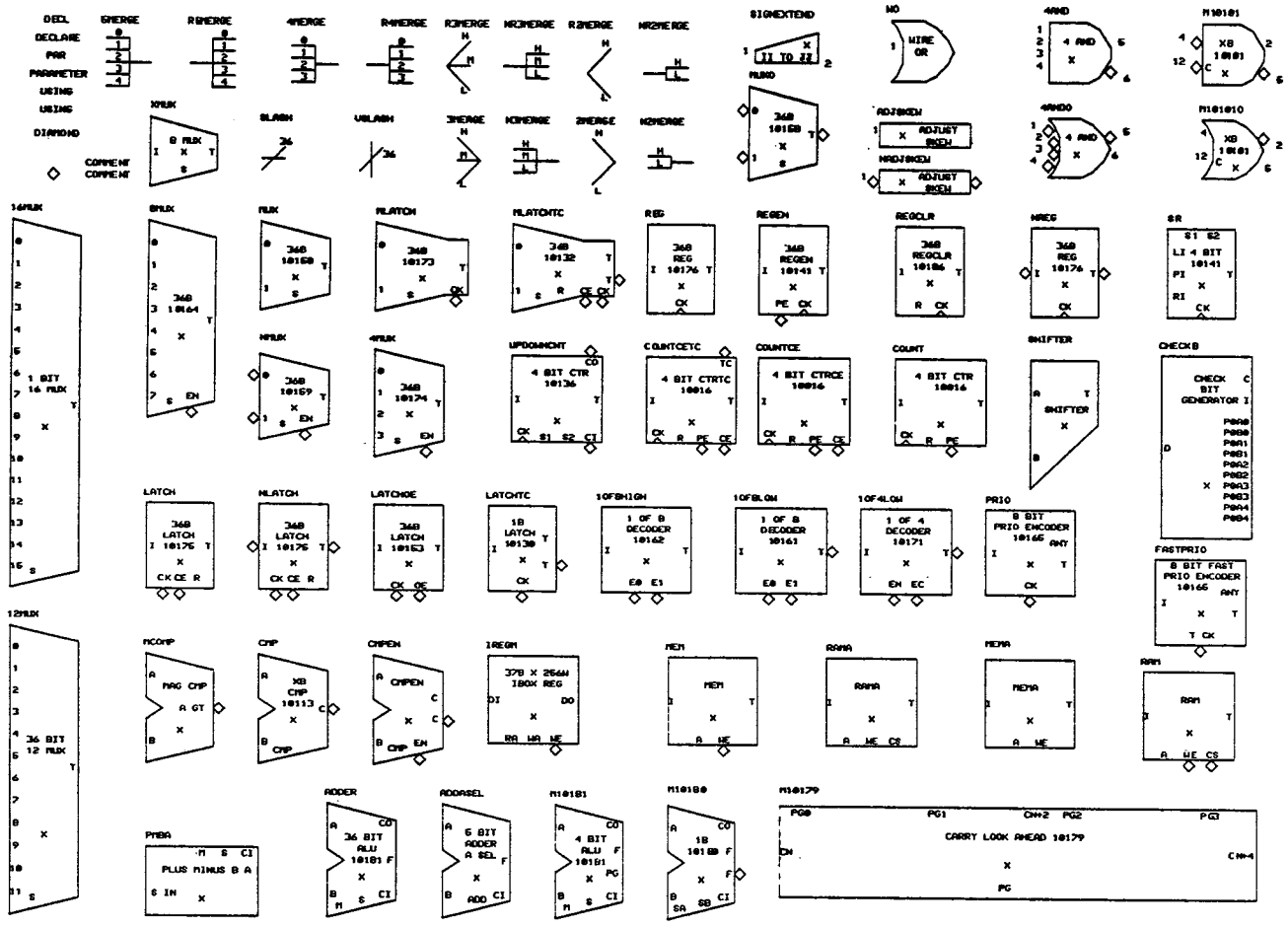

Figure 3-1
Body Library

A macro definition must always declare in a PARAMETER declaration all of the signal parameters that will be passed to it, as shown in Figure 1-3. The parameter signals declared must be the same as those shown in the body at the call site; they are checked for consistency. Where a parameter signal is used, its name can optionally have the string "/P" following it at each use, allowing the macro expander to check for consistency, and improving readability. The scope of a parameter signal is the enclosing macro definition.

A signal name followed by "/M" is a local signal; the scope of a local signal is the enclosing macro definition.

Global signals are those which have no "/M" or "/P" suffix, and which are not contained in a PARAMETER declaration. This syntax for specifying the type of global signals was found through experimentation to be superior to the method of declaring all global signals, as all variables are declared in ALGOL, first because signal names are commonly long, and also because most signals are used infrequently, thus both the absolute and relative overhead involved in maintaining the global declarations was found to be large. Undeclared global signals have unlimited scope. The scope of global signals can be limited to a subtree in the dynamic call structure by declaring them at the root of the desired subtree. In Figure 1-2, the signal "OUTPUT SIGN" is declared, and therefore its scope is limited to its containing macro and to all macros below it in the dynamic call structure.

### 4.3 Versions

In SCALD, there is a difference between a logical and a physical signal. A physical signal is simply a run in the object machine, but a logical signal is a set of physical signals (versions) that essentially always have the same value. Signals in drawings are logical signals. For example, the logical signal "REG CK BUF /M" in Figure 1-2 is driven by a 10110 gate, which has three identical outputs, thus three physical versions. Each output is a different physical signal; loads will be distributed among the three outputs when the object machine is laid out.

### 4.4 High-Low Drivers

In the ECL logic families, many functions have both the true and complementary outputs available. SCALD allows this functionality to be fully utilized. In the definition of a multiplexer chip, for example, it is specified that the values of the select lines can be complemented without affecting the function, if the inputs are permuted in a particular way. When a multiplexer chip is laid out, it can be laid out in its *reverse* form, and SCALD will automatically search to see if there is an unused complementary output on the gate driving the select line. If so, SCALD will automatically utilize it, permuting the inputs to the multiplexer as specified. In Figure 1-2, for example, the "36B 10174" macro can be laid out with 18 bits being driven by the true output of the gate on its select line, and the other 18 bits being driven by the complementary output.

This manner of representing high-low drivers , and similarly, the manner of allocating physical versions, helps to minimize the amount of information in the drawings which is not related to the *logical design*, thus making the logical operation of the object machine more apparent, and places the task of specifying which parts are driven high and which parts are driven low in the layout phase of the design where it belongs, since this determination may depend heavily on positions of chips.

### 4.5 Macro Definition

There are two basic types of macro definitions. The first consists of a complete definition, such as the "SIMPLE PROCESSOR" macro. The second is called an "XB" (X-Bit) macro, which consists of the definition of a single bit or bit-slice of a bit-wise symmetrical function, and expands to a width given by a parameter in the macro call.

A macro definition may have formal signal parameters, through which are passed signal-vectors from the call sites. All of the formal signal parameters of a macro must be explicitly declared in a PARAMETER declaration, as shown in Figure 1-3. These formal signal-vector parameter names must be the same as the signal names which are written on the macro body. The width of a signal-vector is shown in the PARAMETER declaration, and is checked on each call. All signals with the same name within a given same scope, and all points connected together with a line, are wired together. If more than one signal name is written on a line, then the signals are *synonyms*, that is, all connections to each such signal will be wired together.

Macro definitions can consist of one or more pages, where a multiple-page macro has the same macro title on each page.

### 4.6 Macro Call

A macro call consists of a body which has:

- An arbitrary shape.
- Formal signal parameters for passing signal vectors to the macro definition.
- A macro name.
- A label.
- A size parameter.
- A times parameter.

To enhance the understandability of the drawings, different calls of the same macro can have different shapes. For example, it is common to have two shapes for a macro, corresponding to its positive and negative logic forms.

The formal signal parameters of a macro are drawn around the edges of the macro body; each consists of a text string and a point for lines to attach to it. If a parameter is an active low signal, then the body generally contains a diamond at the point at which the lines connect, and the formal parameter in the definition consists of the formal parameter shown in the call, with " L" appended to it, to indicate that it is an active low signal.

The macro name is a text string, which is normally placed in the middle of a macro shape; it can be more than one line long, as shown in Figure 1-2 ("36 BIT ALU 10181"). The macro name connects the macro call with the appropriate macro definition.

The label is a text string, also normally placed in the middle of a macro shape. The label varies from call to call for a given macro body; the label associated with each macro call must be unique within the enclosing macro definition. The labels in Figure 1-2 are "CTL", "R1", "R2", "R3", "A", "M", "G1", "G2", and "G3".

The size parameter is used only for calls on XB-type macro definitions; the value of this parameter is of the form "nB", where "n" is an integer representing the number of bits in this particular macro call. Special syntax in the XB-type macro definition allows SCALD to create signal names which have bit numbers included in them.

The times parameter has a value of the form "n*", where "n" represents an integer. If a macro call has a times parameter of "n*", then the macro will be automatically called n times, using the same values of all parameters during each call, making each label unique (by suffixing), and making all version designations within the called macro unique (also by suffixing). This facility is used for calling functions which must have large fanout without obscuring the logic at the call site.

### 4.7 Terminal Component

A terminal component call consists of a body which has:

- An arbitrary shape.
- Pin names for connecting signals.
- A component name.
- A label.

A terminal component corresponds to either all or part of a chip. It can have different shapes at different call sites to enhance its understandability.

Signals connect to a terminal component only at pins, which have pin names which may be different from the actual pin numbers on an IC.

The component name specifies the chip type.

The label is used to identify the component in the macro definition, and must be unique among macro labels and terminal component labels within a macro definition.

### 4.8 Macro Expansion

The output of the macro expansion is a connection list, which shows all of the pins on terminal components to which each signal connects. Each run on the list is given a unique name. A run name is of the form "PATH_NAME :SIGNAL_NAME".

SIGNAL_NAME is the name used in the macro definition to refer to the signal, except that the name generated for parameter signals is the name of the signal which was passed when the macro was called. If two or more signals are synonyms, then the name of the one which is declared higher in the call structure is used, and if multiple signals within the same level are synonyms, then the one which comes first in the alphabet is used.

For a signal which is global throughout the entire system, PATH_NAME is "TOP". For local and global signals which are declared in a macro, PATH_NAME is created by concatenating, in order, the label in each macro call (with periods between them) in the expansion from the top-level macro down to the macro containing the signal of interest.

### 4.9 Board Layout and Partitioning

During the macro-expansion process, each terminal component generated is assigned a terminal path name, which is generated by concatenating, in order, the labels of all the macro calls in the path down the expansion tree which generated the terminal component (separated by periods), as well as the label on the terminal component. Since each label is unique within a macro definition, all terminal path names are unique.

SCALD inputs a language which maps terminal path names to boards and to positions on boards. A companion paper, *The SCALD Physical Design Subsystem*, describes in detail the language used to construct the S-1 processor (which was manually laid-out). In general, the mapping function can be specified either totally manually, fully automatically, or by some combination of manual and automatic techniques.

### 4.10 Text Form of Macro Language

The output of the SUDS editor is a text form of the Macro Language, which is input to the Macro Expander. The text form of the Simple Processor Macro is shown in Figure 4.10-1. The text Macro Language contains exactly the information in the drawings, but omits information about position and shape. For each macro definition, it consists of declarations which give the file name and macro name, followed by PARAMETER, DECLARE, and SYNONYM declarations. Each body in the macro definition then has an entry which gives either the macro or terminal component name, the logical location label, and the actual signals passed to each signal parameter. SCALD automatically creates signal names for unnamed signals; each such name includes a percent sign to make it different from names input by the designer.

```
MNAME  = SIMPLE PROCESSOR ;
FILE = .EXAM1   ;

DECLARE = OUTPUT SIGN;


SYNONYM = EXT OUTPUT<0>=OUTPUT SIGN;

PROCESSOR CONTROL(LOC=CTL)(A SEL=A SEL<0:1> /M,REG ADR=REG ADR<0:3> /M
  ,ALU CTL=ALU CTL<0:5> /M,REG WRITE L=REG WRITE L /M);
SLASH(SIZE=35)(2=EXT OUTPUT<1:35>,1=EXT OUTPUT<0:35>);
2MERGE()(L=C OUT /M,H=EXT OUTPUT<1:35>,T=%1%T);
10181(SIZE=2B,LOC=G2)(4=A SEL<0:1> /M,5=A SEL BUF<0:1> /M,12=,2=);
10110V(LOC=G3)(6=CLOCK,3=REG CK BUF /M,7=,5=);
REG 10176(SIZE=1B,LOC=R1)(CK=REG CK BUF /M,T=C OUT /M,I=R1%I);
REG 10176(SIZE=36B,LOC=R2)(CK=REG CK BUF /M,T=EXT OUTPUT<0:35>,I=
  I<0:35> /M);
10105A(LOC=G1)(5=REG WRITE L /M,4=CLOCK,2=G1%2,3=);
16W RAM 10145A(LOC=R3,SIZE=36B)(CS L=,WE L=G1%2,I=EXT OUTPUT<0:35>,A=
  REG ADR<0:3> /M,T=B<0:35> /M);
10174(SIZE=36B,LOC=M)(T=A<0:35> /M,EN L=,0=EXT OUTPUT<0:35>,1=%1%T,2=
  EXT OUTPUT<9:35> : C OUT /M *3 : Z=6,3=EXT INPUT<0:35>,S=
  A SEL BUF<0:1> /M);
3MERGE()(L=%2%L,M=%3%M,H=%4%H,T=ALU CTL<0:5> /M);
36 BIT ALU 10181(LOC=A)(CI=%2%L,S=%3%M,B=B<0:35> /M,A=A<0:35> /M,F=
  I<0:35> /M,CO=R1%I,M=%4%H);

END;
```

Figure 4.10-1
Text Representation of Simple Processor Macro

## 5. Output Listings Generated

A number of output listings are generated to help in both the design and debugging of the hardware. The Macro Expander outputs a directory of the macros used in a design, a listing which shows all of the places from which a given macro is called, one which shows all of the macros which use a global signal, and a list of all of the macros and terminal components called by each macro definition, as shown in Figure 5-1 and Figure 5-2. In order to aid in the partitioning and layout of a system, SCALD provides an estimate of the number and types of chips used by a given macro, generated using simple heuristics to account for the packing of multiple sections of a given type into a single chip.

## 6. Conclusions

SCALD has been used to design a 5500-chip ECL processor (the S-1), and in addition to basic facilities for hierarchical design, it contains many features which have been found to be essential either for the understandability of the design or for the efficiency of the machine. Among such features are the following:

- Language constructs for declaring and using local, parameter and global *signal vectors*.
- A mechanism for defining in a single drawing all macros of identical structure but different width.
- Mechanisms for conveniently manipulating multiple physical versions of the same logical signal.
- A mechanism which facilitates the use of both physical polarities of a given logical signal.
- Language constructs for representing bit-wise symmetrical logic.

Structured logic design consists of extending to logic design the essential power of the concepts and the tools which have been developed for simplifying the programming task; the savings in human labor expended in digital systems design realizable by this advance are potentially as great as those which the application of compilers has caused in the specification of complex arithmetic and logical computations. Our experience has shown that the SCALD Design System has greatly increased the understandability of the S-1 Processor, thus reducing the design effort by a large factor, enhancing design correctness, and facilitating the generation of final documentation. The *design itself* serves as a major portion of the final documentation because it is so readily understandable; thus, the need for expensive and relatively inaccurate ex post facto documentation is greatly reduced. Furthermore, the SCALD Design System has increased the changeability of the design; since macros are inherently isolated, changes in one macro definition usually require minimal changes in other parts of the design. Finally, the imposition of structure on the design will facilitate machine verification; that is, it will support simulation of the S-1 at a various levels above the chip level.

## 7. 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 we have received from Forest Baskett, Lowell Wood, and Bill vanCleemput throughout this work, and the support Sassan Hazeghi provided in writing and responsively maintaining an excellent PASCAL compiler at the Stanford Linear Accelerator Center.

## 8. References

vanCleemput, W. M. 1977. "A Hierarchical Language for the Structural Description of Digital Systems," *Proceedings 14th Design Automation Conference, New Orleans, La, June 1977*, pp. 378-385.

Helliwell, D. 1972. "The Stanford University Drawing System", Stanford Artifical Intelligence Laboratory, Stanford University.

```
MACRO:      PROCESSOR CONTROL                                    NUMBER    2
FILES:       .EXAM2

CALLED    1 TIMES FROM:  .EXAM1            SIMPLE PROCESSOR #1

PARAMETER   ALU CTL<0:5>(1),  A SEL<0:1>(1),  REG ADR<0:3>(1),  REG WRITE L(1)

LOCAL       BRANCH ALW(2),  BRANCH NEG(2),  BR ADR<0:7>(2),  MICRO INSTR<0:22>(8),
            C%T<0:7>(2),  G2%3(2),  G2%5(2),  G3%3<0:7>(2)

USING       CLOCK(1),  EXT LOAD CS DATA<0:22>(1),  EXT LOAD CS WE L(1),  EXT RESET(1),
            OUTPUT SIGN(1)

SYNONYM     BR ADR<0:7> = MICRO INSTR<0:7>
            BRANCH ALW = MICRO INSTR<8>
            BRANCH NEG = MICRO INSTR<9>
            REG ADR<0:3> = MICRO INSTR<10:13>
            REG WRITE L = MICRO INSTR<14>
            ALU CTL<0:5> = MICRO INSTR<15:20>
            A SEL<0:1> = MICRO INSTR<21:22>

MACROS CALLED

C           8 BIT CTR 10016 #11 ( CK = CLOCK,  I<0:7> = BR ADR<0:7>,  PE L =
            G2%3,  R = EXT RESET,  T<0:7> = C%T<0:7> )

G1         *10104A #19 ( 2 = G2%5,  4 = BRANCH NEG,  5 = OUTPUT SIGN )

G2         *10105A #20 ( 2 = ,  3 = G2%3,  4 = BRANCH ALW,  5 = G2%5 )

G3          XB 10110V #12(SIZE=8) ( 3<X> = G3%3<0:7>,  5<X> = ,  6<X> =
            C%T<0:7>,  7<X> =  )

R           XB 256W RAM MB7042 #10(SIZE=23) ( A<0:7> = G3%3<0:7>,  CS L = ,
            I<X> = EXT LOAD CS DATA<0:22>,  T<X> = MICRO INSTR<0:22>,  WE L =
            EXT LOAD CS WE L )

   CHIPS  LOCAL SECS  TYPE

      23         0    MB7042
       2         0    10016
       1         1    10104A
       1         1    10105A
       8         0    10110
  ------
      35
```

Figure 5-1
Summary Output Listing from Macro Expander

```
                    DEFINE/USING CROSS REFERENCE LIST

CLOCK                          AUTODECL  .EXAM1          SIMPLE PROCESSOR #1
                               USING     .EXAM2          PROCESSOR CONTROL #2(1)
                                         .EXAM1          SIMPLE PROCESSOR #1(2)

EXT INPUT<0:35>                AUTODECL  .EXAM1          SIMPL PROCESSOR #1
                               USING     .EXAM1          SIMPLE PROCESSOR #1(1)

EXT LOAD CS DATA<0:22>         AUTODECL  .EXAM2          PROCESSOR CONTROL #2
                               USING     .EXAM2          PROCESSOR CONTROL #2(1)

EXT LOAD CS WE L               AUTODECL  .EXAM2          PROCESSOR CONTROL #2
                               USING     .EXAM2          PROCESSOR CONTROL #2(1)

EXT OUTPUT<0:35>               AUTODECL  .EXAM1          SIMPLE PROCESSOR #1
                               USING     .EXAM1          SIMPLE PROCESSOR #1(8)

EXT RESET                      AUTODECL  .EXAM2          PROCESSOR CONTROL #2
                               USING     .EXAM2          PROCESSOR CONTROL #2(1)

OUTPUT SIGN                    DECLARE   .EXAM1          SIMPLE PROCESSOR #1(8)
                               USING     .EXAM2          PROCESSOR CONTROL #2(1)
                                         .EXAM1          SIMPLE PROCESSOR #1(1)
```

Figure 5-2
Cross Reference Output Listing from Macro Expander