Date:	October 31, 1967
From (location or U.S. mail address):	Advanced Computing Systems Menlo Park, California
Dept. & Bldg.:	988/031
Telephone Ext.:	252

IBM

Subject: A Proposed ACS Logic Simulation System (LSS)

Reference:

1. Specifications for Input and Output of ACS/TALES Simulator, A. G. Auch, Dept. B24, SDD Poughkeepsie, September 20, 1967.
2. TALES - ACS Simulation Capability, A. G. Auch, Dept. B24, SDD Poughkeepsie, August 15, 1967.
3. ACS AP #67-115, MPM Timing Simulation, L. Conway, August 25, 1967.
4. ACS AP #66-022, ACS Simulation Technique, D. P. Rozenberg, L. Conway, R. H. Riekert, March 15, 1966.

To: File

L. Conway

L. Conway

LC:aw

347

L. Conway Archives

CONTENTS

Introduction

This memorandum describes a proposed ACS Logic Simulation System(LSS). This system has been only tentatively defined. The purpose of this memorandum is to set down the current thinking and stimulate some feedback from potential users, potential implementers, and other critics on the feasibility and utility of such a system and on the practical details of its implementation and use.

The purpose of the proposed LSS is to provide a mechanism for aiding the debugging of the logical design of the ACS-1. The logical designer may know that for a given section of logic circuitry a certain set of inputs should produce a particular set of outputs (for a given initial internal state) according to the "system level" description of the design which he implemented in the logic circuitry. The LSS will provide a means of inserting the circuit inputs into a logic simulator which simulates the action of the circuitry on these signals and then compares the resulting output with the output expected by the designer. Any mismatches would indicate a logical design error in the circuit (see fig. 1)

A group in Poughkeepsie can provide ACS with a package of programs capable of performing the logic simulation. The ACS designer would provide input to these programs indicating the particular partition of the machine to be simulated and the input-output lines on the interface of this partition. The programs would use this input to extract from the DRKS files the detailed description of the logic of the partition selected. The designer would then need to apply a sequence of inputs to the logic simulator corresponding to a proper sequence of input-output line signals at the interface of the partition. The programs would simulate the logic operating on the input signals and mark any mismatches in the logic output and expected output. The designer would then use these mismatches to debug his logic design.

A major obstacle to the practical application of this proposed system is the difficulty of generating the I/O signals at the partition interface. It does not appear to be at all practical, or even feasible, for the logic designers to generate by hand all the correct test patterns necessary to "moderately" debug all the partitions of the machine.

A method has been proposed to solve this problem by providing a programmed means of automatically generating these interface I/O signals. A detailed timing simulator now exists for the MPM (ref. 3). This simulator times the activity of all MPM hardware, as described at a system level, during the execution of an input program.

Now, suppose we wish to use the LSS to study and debug a particular partition of the MPM. We could carefully define the interface of that partition and rewrite the appropriate sections of the timing simulator such that (i) the same interface existed in the timer as in the logic circuitry, (ii) the same "system" level description is used in the timer to describe the partition that was used to formulate the logical design of the partition, (iii) provide for output to suitable files of the timing simulator interface signals during each simulated cycle of execution.

The timer thus modified could become a practical source of the I/O signals needed to drive the LSS. The timer would have to accurately reflect the MPM only at and within the interface of the partition to be studied. Any errors in this system description would be discovered early in the debugging process. After this phase, many selected programs could be run on the timer to yield as many interface signal sets as are necessary to debug the logic design of the partition to the required level (see fig. 2).

The timer could also assist the designer of the partition in his efforts to find a particular bug when the LSS indicates a mismatch in outputs. The timing charts produced by the timer will give a concise picture of the state of the machine at a system level in the region of time surrounding and including the cycle in which the bug occurred. This may help to determine if the bug is at the level of system specification or logic circuit implementation. Both the timer and LSS can provide the states of specified triggers within the partition and a comparison of these can aid the designer in debugging.

In the following sections of this memorandum some of the details of this proposed LSS system are described and questions are raised which must be answered before any serious development of the system can begin.

The main point to keep in mind is that there are two levels of simulation involved in this scheme -- the detailed simulation of the logic circuitry of a design and the system level simulation of the same design. This two level simulation technique for debugging logic circuitry was originally proposed to ACS in August, 1966 by G. T. Paul. The technique now appears to be feasible because of the availability of an adequate logic simulator and ACS experience with the current timing simulator.

Comments and criticisms are invited, especially on questions concerning the feasibility of the system, its utility to the ACS logic designers, its cost relative to any alternative systems, and the various practical problem of its implementation and use.

FIG1. THE BASIC IDEA OF LSS:

APPLY SAME INPUT TO BOTH LEVELS OF SIMULATION AND COMPARE OUTPUTS. IF OUTPUTS ARE DIFFERENT THEN ERROR EXISTS IN LOGIC DESIGN.

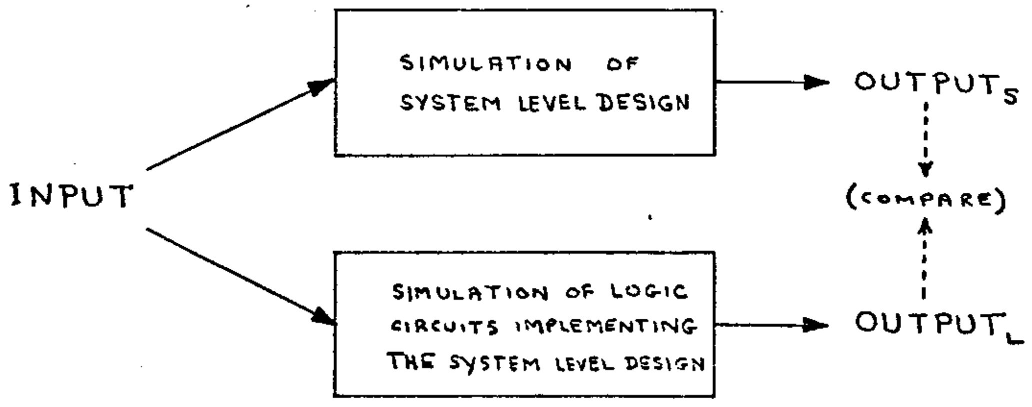
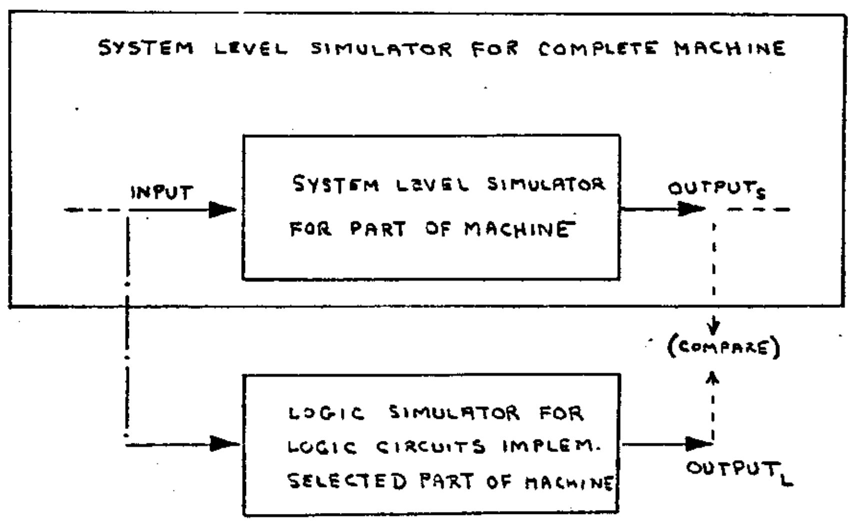

INPUT
SIMULATION OF SYSTEM LEVEL DESIGN
SIMULATION OF LOGIC CIRCUITS IMPLEMENTING THE SYSTEM LEVEL DESIGN
OUTPUT_S
(COMPARE)
OUTPUT_L

FIG 2. AUTOMATIC GENERATION OF SYSTEM LEVEL INPUT/OUTPUT, LOGIC SIMULATOR INPUT:

SYSTEM LEVEL DESIGN IS IMBEDDED IN SYSTEM LEVEL SIMULATION OF ENTIRE MACHINE. WHEN THIS SIMULATOR RUNS WE AUTOMATICALLY GENERATE (AND SAVE) THE I/O AT THE DESIGN INTERFACE. WE MAY LATER APPLY THESE INPUTS TO THE LOGIC SIMULATOR FOR THE SAME DESIGN AND COMPARE THE LOGIC OUTPUTS WITH THE SYSTEM LEVEL OUTPUTS.


SYSTEM LEVEL SIMULATOR FOR COMPLETE MACHINE
INPUT
SYSTEM LEVEL SIMULATOR FOR PART OF MACHINE
OUTPUT_S
(COMPARE)
LOGIC SIMULATOR FOR LOGIC CIRCUITS IMPLEM. SELECTED PART OF MACHINE
OUTPUT_L

351

The LSS Programs

In this section the programs forming the LSS are identified and described. The relationships between the various programs and the designers input and output to the system is described. This specification was developed from information contained in ref. 1 and the notion of using the timing simulator to drive the LSS. This specification is very tentative in nature.

The simulation of the logic of a portion of the ACS-1 machine operating on a sequence of inputs may be viewed as occurring in three distinct phases within LSS.

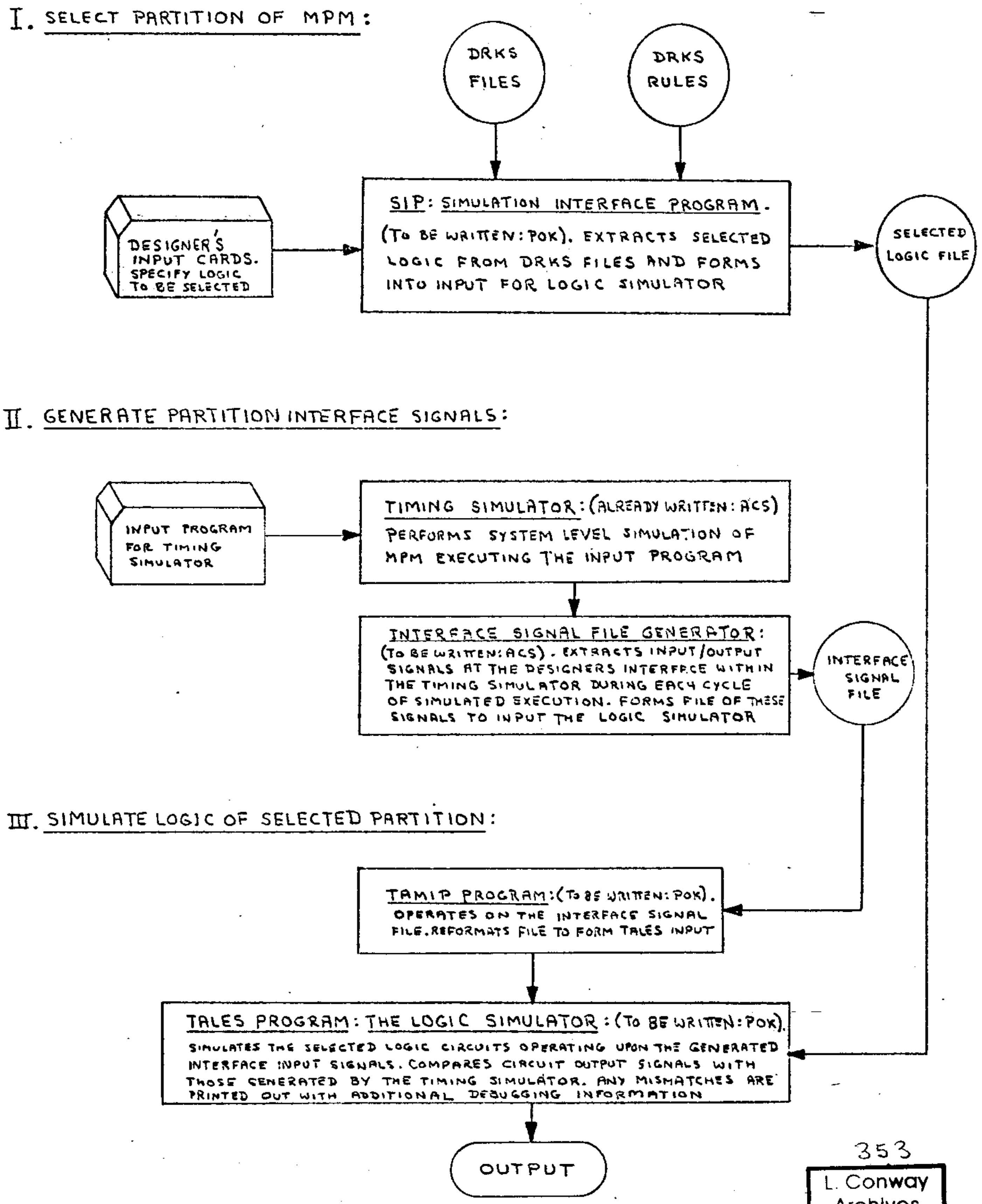
The first phase is the selection of the specific partition of the machine to be studied and the specification of the I/O interface for this partition. The designer will specify the partition and interface in a card input deck. This deck is used by the LSS to extract the detailed information describing the logic circuitry of the partition from the DRKS files and DRKS rules. The program performing this extraction is termed the Simulation Interface Program (SIP), and is to be written by the Poughkeepsie people.

The next phase of the LSS simulation is the generation of a sequence of interface signals for the selected partition. This is done by running ACS program on the modified timing simulator. Once the designer has assisted in forming the proper timing simulator specification for his partition, the production of these interface signals requires no more effort by him. Many programs exist which run on the timer. The designer would merely select those programs which might best be applied to debugging his particular section of the machine. An addition must be made to the existing timing simulator to extract and file the proper interface signals during each cycle of simulated time. Let us call this the interface signal file generator. This program would be written here at ACS.

The final phase of the LSS run is to perform the logic simulation itself. This is done by a program to be called TALES, which is to be developed by the Poughkeepsie group. The interface signal files produced by the timer-interface file generator programs are processed by a reformatting program called TAMIP (also to be written by Poughkeepsie) and then input the TALES logic simulator. The TALES simulator uses the logic files formed by the SIP program to perform the proper logical functions on the input signals to yield interface output signals for each simulated cycle. If the logic simulator output signals differ from the expected output signals produced by the timing simulator, an output listing to this effect will be produced and certain information printed to assist the designer in finding the cause of the mismatch.

In figure 3 the functions of the three phases of LSS are illustrated by flow-charting the relations between the designer's input, the various LSS programs, the DRKS files, and the various LSS internal files.

FIG 3. THE ACS LOGIC SIMULATION SYSTEM



I. SELECT PARTITION OF MPM:

DRKS FILES
DRKS RULES
DESIGNER'S INPUT CARDS. SPECIFY LOGIC TO BE SELECTED
SELECTED LOGIC FILE

SIP: SIMULATION INTERFACE PROGRAM. (TO BE WRITTEN: POK). EXTRACTS SELECTED LOGIC FROM DRKS FILES AND FORMS INTO INPUT FOR LOGIC SIMULATOR

II. GENERATE PARTITION INTERFACE SIGNALS:

INPUT PROGRAM FOR TIMING SIMULATOR
INTERFACE SIGNAL FILE

TIMING SIMULATOR: (ALREADY WRITTEN: ACS) PERFORMS SYSTEM LEVEL SIMULATION OF MPM EXECUTING THE INPUT PROGRAM

INTERFACE SIGNAL FILE GENERATOR: (TO BE WRITTEN: ACS). EXTRACTS INPUT/OUTPUT SIGNALS AT THE DESIGNERS INTERFACE WITHIN THE TIMING SIMULATOR DURING EACH CYCLE OF SIMULATED EXECUTION. FORMS FILE OF THESE SIGNALS TO INPUT THE LOGIC SIMULATOR

III. SIMULATE LOGIC OF SELECTED PARTITION:

TAMIP PROGRAM: (TO BE WRITTEN: POK). OPERATES ON THE INTERFACE SIGNAL FILE. REFORMATS FILE TO FORM TALES INPUT

TALES PROGRAM: THE LOGIC SIMULATOR: (TO BE WRITTEN: POK). SIMULATES THE SELECTED LOGIC CIRCUITS OPERATING UPON THE GENERATED INTERFACE INPUT SIGNALS. COMPARES CIRCUIT OUTPUT SIGNALS WITH THOSE GENERATED BY THE TIMING SIMULATOR. ANY MISMATCHES ARE PRINTED OUT WITH ADDITIONAL DEBUGGING INFORMATION

353

Possible Procedures for Use

So far we have examined the overall functions of the LSS and identified the component programs and files. All of this is very tentative. In this section let us explore some of the many different possibilities which exist for organizing and using the LSS system, and identify those areas which are only tentatively defined and need to be worked on.

Many questions and alternative approaches are outlined which must be resolved before the system can be considered feasible, useful, and economical. Criticism on these specific questions from everyone concerned is needed to formulate the answers to these questions.

Most of these questions center on the organization and management of the system, i.e., what technical form should the system have in order to be usable by the designer? For example, how do we partition the machine, how large or small should the partitions be, and how do we select the interfaces? How should the designers specify the system level description of their partition?

(i) Partitioning the MPM: How large or small should a partition be? From an organizational and system simulator point of view, the larger the better. If a partition is too large, however, the designers may have a difficult time in debugging the logic. This problem might be eased by placing certain triggers internal to a partition in the set of outputs the designer can check. If the partitions are too small and thus many in number, we will have difficulty in managing the study--there will be too many interfaces, and some of them may be inconvenient to specify at the system level.

It seems undesirable to have a single partition so large or so chosen that two different design groups design sections of the partition. The utility of the LSS system is increased by having formal interfaces between the various groups of designers, to allow a successful segmentation of the design. It is natural that the interfaces between design groups would also be interfaces in the system level simulator in LSS.

An approach to choosing partition size might be the following: choose the partitions as large as is possible subject to the following constraints, (i) the boundaries of the various design groups, (ii) the maximum amount of logic which the logic simulator will handle. It is likely that the second limit will usually be met first. This raises the question of whether the logic simulator (TALES) can handle a large enough partition for the LSS to be practical. This question is quantitatively studied (section 4) later in this memorandum, and the answer currently appears to be yes.

(ii) Selecting the Interface: Suppose we wish to formulate a partition of the MPM whose approximate size and boundaries are known. We face the problem of selecting the exact interface that is to exist between this partition and the rest of the machine. This is the problem of selecting an interface which is reasonable both in the logic and in the system level of description. The problems involved in doing this do not appear to be serious if the partition is large, for then certain natural boundaries (the phases) within the MPM may be chosen as interfaces. If the partitions must be very small and many in number, we will have serious problems for the system level description as a whole will become much more detailed and unmanageable. We might not be able to simulate on a cycle by cycle basis, but have to generate and check interface signals at many different times within a machine cycle.

(iii) Describing a Partition: In order to correctly generate the interface signals for a given partition, the timing simulator must accurately reflect the system level description of that partition. An important question to be answered is how is the detailed system level description of a partition to be formed, in what language, and by whom? There is a wide range of possibilities.

Method (a). The designers could give a verbal, nonformal description of their partition to a programmer who would formalize the description by writing the code which performs the system level simulation. This is probably not adequate because it would be too difficult to maintain the description. The designers would have no direct link to the formal description when they desired to make a change.

Method (b). The designers could produce a "semi-formal" description of their partition by creating a combination of flow charts, diagrams, and written description which attempted to document as accurately as possible (outside a formal language) all the details of their design. A programmer could use documents of this type as a direct basis for his coding of the system level simulation. This at least solves the problem of maintenance of the program. A change in a flow chart could fairly easily point to the necessary corresponding change in the simulator code. Even with this method, serious problems arise (even more serious if using Method (a)). Since the designers would not themselves have a complete, formal description at a system level of the thing they have designed, many errors are bound to occur in the system description--errors which would be difficult to debug.

Method (c). We might go a step further in the specification of a partition by the designers and require that they help formulate and have access to a complete, formal description of their partition at the system level. This could be done by having the designers partitipate actively in the production of the formal description. The obvious choice of a language for formal description is the simulation language used in the timing simulation program. This language is an "elementary form" of "Simscript, " and is written in FORTRAN (see ref. 4).

The designers could produce the flow charts, etc., as in Method (b), but then assist in the production of the system simulation code to the extent that they would fully understand and be able to modify (with programming assistance) the system level description.

The system simulation code would then be the formal description for the designer. It would be easy for the designer to introduce changes into the formal description.

Method (d). We can go one step further and require that the designers independently produce a formal system description of their partitions in some language common to all the design groups. This is a goal to strive for in later design efforts. It seems impractical at the present time, however, because of (1) the time required to educate the designers in some formal language, (2) the even greater time required for them to gain "programing" experience--the experience needed to use the language to describe their design at the proper system level. Most logic designers probably conceptualize their design not as a system description being implemented in some logic circuitry, but as the logic circuit implementation itself. That this is likely is indicated by the current lack of detailed system descriptions within engineering and the current wealth of logic circuit diagrams.

Considering the methods (a), (b), (c) and (d) outlined above, it would appear that the most useful and feasible method for currently producing the necessary system level descriptions for the LSS is Method (c).

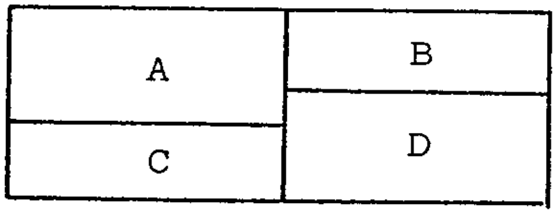
(iv) Selecting the Partition in the Logic: When we have selected and described a partition at the system level, we face the problem of selecting the same partition at the logic circuit level. The description of the logic circuits is formal and is contained in the DRKS files. The Poughkeepsie group will write the SIP program which actually extracts the logic design of a partition and forms the file to input the logic simulator.

The designer's input to specify the logic to be selected by the SIP program has been tentatively defined in reference 1. There will have to be a study by all concerned to produce a specification of the SIP input conventions. Once the procedures for use of the LSS system have been defined, it would be desirable to specify input conventions for SIP which are the simplest possible in nature which meet the needs of the LSS. The smaller and simpler the interface between ACS designers and Poughkeepsie programs the better.

(v) Sequence of Partitions to be Studied: An important property of the proposed LSS using the existing timing simulator as a starting point in the system level description is that the debugging of one partition may proceed independently of that of another partition. We can thus choose a sequence of partitions to be debugged which corresponds to the schedule of design of the partitions.

We could have chosen not to use the timer, but to apply Method (d) of the previous section and develop a formal and accurate system level description of the whole machine. Let us examine some of the problems within this scheme and thus learn the advantages of using the timer.

Suppose the machine could be divided into four partitions:



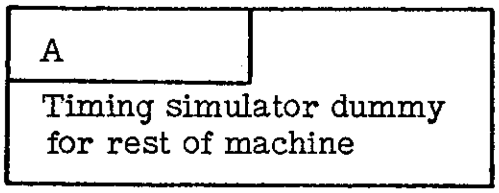
We could have the designers write the programs described A, B, C, and D and then run these as an accurate timing simulator, obtaining input and output signals at the interfaces.

The problem with this is that the system level programs must all exist and be reasonably debugged before the whole system level simulation will run. Of course the individual partition programs could be run separately to yield partition outputs for a given set of partition inputs. But this does not solve the original problem affecting the feasibility of logic simulation--the difficulty of generating by hand all the input-output patterns. It only half solves the problem.

Another difficulty with this approach is that we would be heavily committed to whatever techniques were chosen to implement Method (d).

Clearly we do not need to face these problems and uncertainties. The existing timing simulator can be used to circumvent them as follows:

We chose for LSS debugging the first partition whose design is "completed." Suppose this is partition A.



We already have a working, debugged timing simulator which simulates an approximation to the whole MPM. We write and place into the timer (replacing existing code) the the description of partition A at the system level. Now the remainder of the timer serves as a dummy machine which can properly interact with partition A once the system description of A is debugged. Now we may not get exactly the same feedback from the dummy portion of the machine that we would get from the eventual real machine, but this does not matter. We will get valid feedback which will properly drive partition A. We will automatically get <u>both</u> inputs and outputs of A every cycle while the simulated machine runs an input program.

This allows a considerable degree of freedom in the planning of the debugging process. We may debug the partitions independently and in sequence if we so desire. It is likely that the various partitions will be ready for debugging at different times. We could schedule the debugging to correspond to these design schedules. We would not be committed to the first procedures chosen to debug the first available partition. If a method proves unsatisfactory on the first partition, we can modify our procedures for handling later partitions.

By using this method we can proceed only as far as we choose in applying LSS to debugging the logic. We do not need to determine in advance how much of the logic is to be debugged this way. Some sections of the machine may remain in dummy (original timing simulator) form. Some sections of logic such as functional units (adders, multipliers) clearly can have their logic simulator input-output signals formed by hand or by special programs of much simpler form than system level simulators.

Note that the timing simulator can eventually become an exact system level simulator of the whole machine if that end is desired. This method does not preclude that possibility. Indeed, this method offers a practical means of achieving that end in a step by step approach rather than attempting it directly.

(vi) Debugging a Partition: How does the designer use LSS to uncover bugs in the logic design? Let us consider various procedures which might help in the debugging process.

An important consideration in the debugging of a partition is the selection of some appropriate input programs for the system simulator. We wish to run programs on the timer which exercise as fully as possible the system logic of the partition under study, in order to debug that partition as fully and efficiently as possible. This selection process is yet to be developed.

A question which arises here is how far should the debugging of a partition proceed using LSS. This is a function of input program choice, the available computer time and manpower available for debugging. This question must be studied fully in order to estimate the performance of the LSS system compared to its cost.

An important potential function of LSS which must be explored and developed is that of providing the designer with information to assist his debugging effort in addition to the mere indication of an output mismatch.

One possibility, easily implemented, is to make available to the designer the timing charts produced by the timing simulator (see ref. 3) for the LSS run under study. It has proven possible, with some practice, for individuals to use the timing charts to follow completely the system level functioning of the MPM. The designer would thus have available to him a concise description of the states and functioning of the whole machine in the region of time surrounding and including the cycle in which a bug was found in his partition.

Another possibility is to have the timer and the logic simulator both provide as output the contents of important registers and triggers within a partition in addition to those on the partition interface. This would be especially important if the partition is a large one. Of course we would have to have the timer quantities behave exactly as the logic circuits in order for this to work. This might provide a practical way of allowing large partition size, yet feasible debugging. As an example, suppose a large section of phase 1 of the MPM is to be contained in one partition. It would be very useful in the debugging process if the designer had access to the values of such things as NFA, HISTORY TABLE, DO TABLE, etc., in both levels of simulation (i.e., as "interface output quantities"). Usually these important internal quantities of a partition could be easily made to function exactly the same at both simulation levels.

(vii) Other Modes of Use: During the specification and development of the LSS system we must identify and meet the requirements for any other possible uses of the system and its components.

An example of this is the need to allow manual insertion of interface signals into the Poughkeepsie programs in order to perform the debugging of isolated sections of design for which manual signal insertion is adequate. Examples of such design areas where manual or special program generation of the interface signals is possible are functional units such as adders, multipliers, dividers, etc.

Another function the system might perform is the generation of files suitable for hardware debugging at a later time.

Requirements for Development

The hardware, software, computer time and personnel required to develop, use and maintain the LSS system must be estimated to determine if the system is feasible and economical.

It has been determined that the ACS Mod. 75 computer will have adequate hardware for both the Poughkeepsie programs and the ACS timer-interface signal generator program.

Yet to be explored are possible work schedules, documentation requirements, and forms of communication needed between ACS and Poughkeepsie. It appears possible for the LSS development to proceed without altering engineering design schedules, if a proper scheme of development is chosen. Of course the time required for the designers to specify the system descriptions of their design areas will add to the design schedule time, but it appears likely that this system description will be necessary whether LSS is implemented or not. The requirements for maintenance of the system are yet to be determined. These depend on the role the designers play in specifying and maintaining the specifications of their partitions.

There are two important considerations which strongly affect the feasibility and economics of LSS. These are the computer time required to simulate and the memory requirements of simulation (determines maximum partition size).

Reference 2 indicates that a few seconds of Mod. 75 time would be required for the TALES program to perform the logic simulation of one machine cycle for the largest partition it could handle. The ACS system level simulation of the whole machine will run at a rate of approximately 10 to 15 machine cycles/second on the Mod. 75.

Thus it appears likely that the feasibility of LSS is not impacted by the computer time requirements. The required time is down in the range where the human time and effort in debugging the results would probably be a stronger limitation than available machine time. Of course these machine time requirements could be heavy ones and thus it is very important that the logic simulator (TALES) be made as efficient as possible, for the running of TALES will probably be the major cost of LSS.

Let us now consider the question of memory requirements and their determination of the maximum partition size.

P. Shivdasani has formulated the following study of this question, based on verbal communications with the Poughkeepsie group. His result of 56K ACS circuits as the maximum partition size indicates that we can choose partitions large enough for LSS to be practical (see section 3(i)).

(i) Storage capacity, S, in K bytes, required to run the logic simulator is

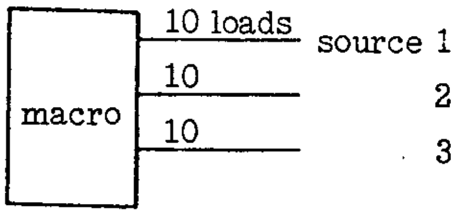
$$S = 98 + 2L\,(10 + \text{avg. fan-in} + \text{avg. fan-out})$$

where L = # of nets to be simulated (in thousands)

Also the fan-out from a block (macro, U. L. or dot) is

$$= \sum_{i=1}^{n} (\text{source}_i \cdot \text{load}_i) \quad \leq 31$$

Thus



fan-out = 30

Another 200K bytes must be allowed for the worst case op. system.

There is also an absolute limit of 32K on L due to the present simulation programs.

Thus if we assume L = 32

fan-out = 31

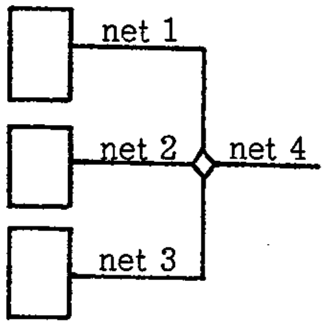
fan-in = 15

We have S = 3882 K bytes which will easily be handled by the two LCS's ACS has on order.

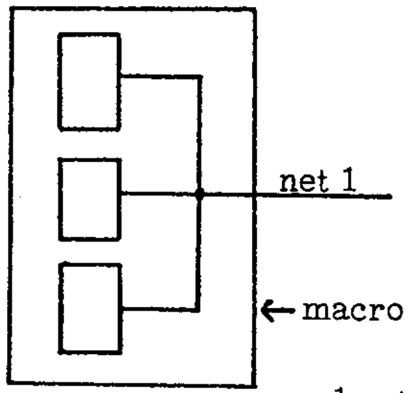
362

(ii) Nets:

A net is defined as a logic source feeding any number of sinks. Thus in U. L. representation each U. L. block leading to a dot is a net.



4 nets



1 net

It is important, then to try and define as many macros as possible.

(iii) Assume 32K nets as maximum partition. Find equivalent in ACS circuits.

a) Let X be the number of circuits corresponding to these nets.

b) Assume 80% of the circuits can be represented in macros and the remaining 20% need a unit logic representation in DRKS.

c) Also assume each macro contains 5 circuits and has two source outputs.

Then nets due to macros $= \left(\frac{.8\,X}{5}\right) 2$

d) Assume an average dot of 4 in U. L. Then we have 5 nets for every 4 circuits.

Or nets due to U. L. $= \left(\frac{.2X}{4}\right) 5$

$$\frac{1.6\,X}{5} + \frac{X}{4} = 32,000$$

$$\text{or } X = \frac{32,000}{.57} = 56\text{K circuits}$$

363

(iv) <u>DRKS</u> does not handle macros made up of U. L. blocks from different <u>portions</u> of the same chip, let alone different chips. So if a high number of U. L. blocks is being dotted externally, the above capability will be desirable to keep the net count down.

Additional Benefits of LSS

There are some additional benefits which might result from implementing the proposed LSS system.

The formal specification of the machine at a system level would give the various design groups a chance to uncover many system level design errors before the logic itself is tested for bugs.

This formal system level description would be useful to many others in ACS.

Of course this description would have to be maintained by the designers to reflect all design changes. If maintained and the timing simulator reflects the description accurately, then the LSS could be used later to generate the interface signals for hardware circuit debugging.

Also, an accurate timing simulator would be very useful to the compiler and system programmers and to any ACS customers who wish to optimize hand code.

TO: L. Conway
Dept. 988
IBM - ACS
2800 Sand Hill Road
Menlo Park, California

Note: If you have any comments, questions, criticisms or ideas concerning the proposed LSS system, jot them down in the space below and mail this page as indicated above.

- -