Gordon L. Smith
Ralph J. Bahnsen
Harry Halliwell

Boolean Comparison of Hardware and Flowcharts

Boolean comparison is a design verification technique in which two logic networks are compared for functional equivalence using analysis rather than simulation. Boolean comparison was used on the IBM 3081 project to establish that hardware flowcharts and the detailed hardware logic design were functionally equivalent. Hardware flowcharts are a graphic form of a hardware description language which describes the logical behavior of the machine in terms of the inputs, outputs, and latches. The logical correctness of the hardware flowcharts was previously established via cycle simulation. The concepts and techniques of Boolean comparison as used on the IBM 3081 project are described.

1. Introduction

Boolean comparison is one element of a new hardware design verification methodology that was used on the IBM 3081 project. The entire methodology involves the use of hardware flowcharts, cycle simulation, and Boolean comparison for logic verification, as well as a timing analysis program for timing verification [1–4]. The methodology has been created to meet the need for an effective, economical, and timely means of verifying the correctness of clocked hardware in the LSI environment. The need exists because hardware debugging is no longer a viable technique for locating a large number of design problems; both the lack of probe points and the large amount of time required to change a chip make LSI hardware debugging slow and difficult. This paper describes the concepts and techniques of Boolean comparison as embodied in this methodology.

Boolean comparison programs may be found as early as 1965 in unpublished program documentation of the system described in [5]. However, little was published until 1976 and thereafter [6–10]. During development of Boolean comparison for the IBM 3081, four algorithms were implemented. Three, while very capable, were unable to completely analyze the 3081 logic because of the amounts of time and storage required. Some of these algorithms were influenced by work on test generation and logic synthesis [11–14]. The algorithms finally adopted because they handled the 3081 logic are the Differential Boolean Analyzer (DBA), written by R. J. Bahnsen, and DBA with Equivalent Sets of Partials (DBA/ESP), suggested by G. L. Smith and implemented by Bahnsen. DBA and DBA/ESP are described later in the paper.

Section 2 of the paper provides an overview of Boolean comparison concepts and the verification methodology. This is followed by a description of the process in which the various models, Boolean comparison, and results analysis are described. A special topics section follows, including a discussion of algorithms, relaxation of requirements, and internal signals.

2. Overview

Boolean comparison concepts

With the methodology used, two logic design representations exist which purportedly describe equivalent sequential machines with the same state assignments. The engineer defines a one-to-one correspondence between the latches, between the primary inputs, and between the primary outputs of the two representations. The Boolean comparison programs then determine whether or not the functions driving each corresponding pair of latches or outputs in the two representations are Boolean equivalent. If this equivalence is satisfied by a pair of representations, it follows that the two machines described by the representations sequence in

exactly the same way [15]. If both machines are placed in corresponding initial states and corresponding input patterns are applied at each clock interval, then both machines pass through corresponding states and the outputs agree at each clock interval. This conclusion is, of course, dependent upon timing assumptions that permit next states to be determined from the time-independent functions driving each latch. Because the IBM 3081 uses a clocked design and timing constraints are checked by a timing analysis program, assurance can be given that the functions driving each latch determine the next state behavior.

Boolean comparison is useful in design verification if two design representations exist where the first is known to function correctly and the second is believed to be functionally identical to the first. Boolean comparison verifies that the two representations are in fact equivalent and that the second, therefore, functions correctly. Of course, Boolean comparison only proves that the second representation is as good as the first, not better. The correctness of the first must be established by other techniques, such as simulation or hardware implementation.

With the methodology used on the IBM 3081, Boolean comparison is performed between detailed hardware logic diagrams of the machine and equivalent higher level hardware flowcharts which represent the same machine. The hardware flowcharts are a graphical form of a hardware description language. These flowcharts describe the operation of the hardware at each clock interval by describing the logical behavior of the latches and outputs in terms of the input and latch values available at each clock interval. However, the detailed implementation of the intervening combinational logic is normally quite different from that of the physically realized machine.

Hardware flowcharts have been used as an integral part of the 3081 design process because of the many advantages they offer for design discipline, documentation, and communication. Prior to Boolean comparison, verification of the functional correspondence of the flowcharts with the hardware was either done by manual checks or by partial checks obtained by simulation. With Boolean comparison, complete verification of functional equivalence is achievable.

- *Verification methodology*

The Boolean comparison program as part of a logic verification system is shown in Fig. 1. A hardware flowchart representation of the logic blocks and connections is entered into the system. The engineer validates the flowcharts via cycle simulation, which simulates the flowchart logic at clock intervals only; no attention is given to the timing of events within clock intervals [4]. Hardware logic is laid out to meet the specifications of the flowcharts.

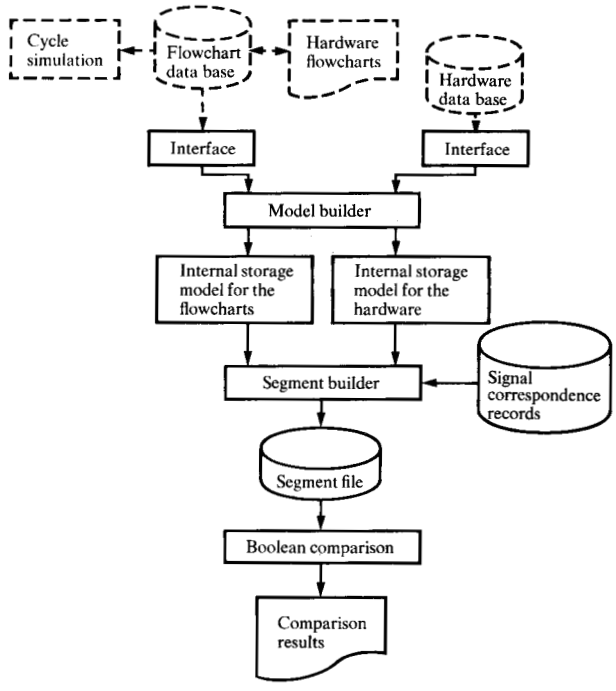


Figure 1 Boolean comparison program.

Flowcharts and hardware logic are converted by the model builder into separate internal storage models consisting of connected Boolean primitives (*e.g.*, NAND blocks). In the case of the flowcharts, a translator converts the flowcharts into Boolean logic in such a way that functional behavior is preserved.

The engineer now provides the Signal Correspondence Records (SCR) list, which associates the inputs, outputs, and latches of the flowcharts with the purportedly functionally identical inputs, outputs, and latches of the hardware logic. By the appropriate use of flags in a control field of the SCR, the engineer indicates the latches and primary outputs for which Boolean comparison is desired during each run. (The generation of the SCR is basically a manual process, although programs are available to aid the process.)

A data structure called a segment is built by the segment builder of Fig. 1 for each SCR entry which is marked for comparison. A segment represents all the logic which drives the output or latch in the flowchart internal storage model plus all the logic which drives the corresponding output or net in the hardware internal storage model plus logic to perform the comparison.

Finally, Boolean comparison is performed on each of the segments. If the Boolean functions are not equivalent, a list

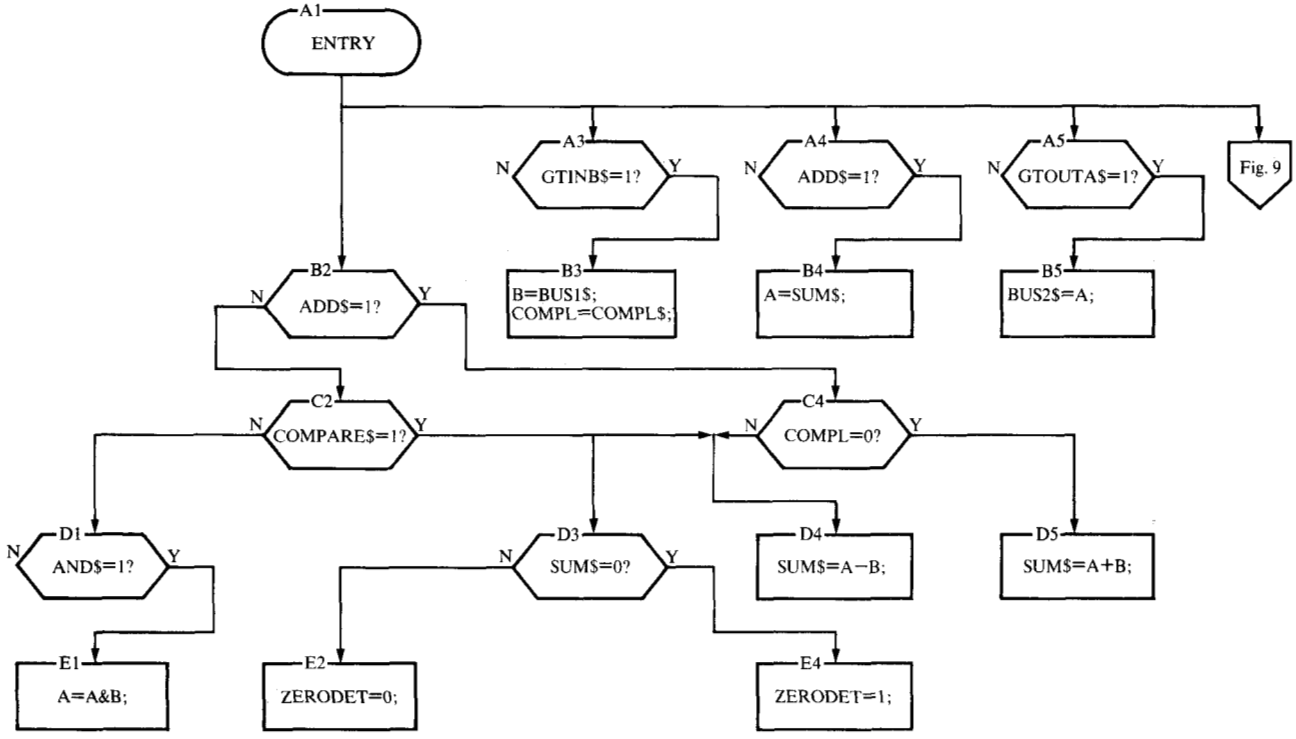


Figure 2 Hardware flowchart for sample machine.

Table 1 Facility declarations for sample machine.

Facility name	*Dimension*	*Type*
A	2	On-over-off triggers clocked by SYSCLK$
ADD$	1	Input signal
AND$	1	Input signal
B	2	On-over-off triggers clocked by SYSCLK$
BUS1$	2	Input signals
BUS2$	2	Output signals
COMPARE$	1	Input signal
COMPL	1	On-over-off trigger clocked by SYSCLK$
COMPL$	1	Input signal
*DONTCARE1$	1	Don't-care signal
GTINB$	1	Input signal
GTOUTA$	1	Input signal
SUM$	2	Internal signals
SYSCLK$	1	Input clock signal
ZERODET	1	On-over-off trigger clocked by SYSCLK$

*Used in don't-care string (see Fig. 9).

of input and latch states (patterns) which cause output mismatches is provided. The engineer analyzes the errata to pinpoint the fault, then corrects the hardware design, the flowcharts, or the SCR, as appropriate, and reruns.

3. The process

• *Hardware flowcharts*

Figure 2 contains a simple hardware flowchart example which displays the various flowchart features. The example describes a sample two-bit machine (see data flow in Fig. 3) which adds, subtracts, ANDs, and compares for equality. Table 1, explained more fully later, is a list of all the declared inputs, outputs, and triggers (which correspond to latches in the hardware) used in the flowcharts. These are known as facilities. The dimension of a facility indicates the number of bits in the facility. Thus a trigger with dimension 2 represents two triggers.

The flowcharts can be viewed as a set of paths which all start at a common top point, *e.g.*, block A1 in Fig. 2. (The various types of blocks are explained subsequently.) During each clock interval, all paths are traced in parallel starting from the top of the flowcharts and continuing down as far as the decision blocks allow. If there is a path from the top to any given assignment block (*e.g.*, block D5 of Fig. 2) for

which all decision blocks (*e.g.*, blocks B2 and C4) are appropriately satisfied during a clock interval, then all statements in that assignment block are executed.

As can be seen in the following text, the structure of hardware flowcharts, which includes branching to one, none, or many blocks, is different from that of program flowcharts, which typically have branching to only one block.

There are five basic structures in flowcharts:

- *Decision blocks,* which cause one of two exits from the block to be followed depending on whether a condition is satisfied or not. The condition is specified by a relation between a pair of expressions involving facilities and possibly constants. An example of a decision block is D3 of Fig. 2. If the condition indicated inside of the block is satisfied, then the yes (Y) leg on the right of the block is followed; otherwise, the no (N) leg on the left of the block is followed.

- *Assignment blocks,* which contain one or more statements that set facilities (triggers or signals) to the value of facility expressions. The facility being set appears to the left of the assignment operator (=). The facility expressions can involve both logical and arithmetic operators. Blocks B3 and D5 in Fig. 2 are examples of assignment blocks.

- *Multiple branching,* by which the entry point to the flowcharts or either exit of a decision block may branch to one or many blocks (also, to no blocks in the case of a decision block exit). Since all paths are traced in parallel, the branches emanating from any given point are considered as members of an unordered set. Paths can diverge or converge, but a path may not loop on itself. Multiple branching occurs at the entry A1 and at the right side of block C2 in Fig. 2. The path A1, B2, C2, D4 and the path A1, B2, C4, D4 diverge at block B2 and converge at block D4.

- *Triggers,* which are clocked facilities that can only change value when a clock pulse occurs at the end of a clock interval. The value set into a trigger depends upon the set and reset conditions generated in parallel during that clock interval. Triggers correspond to latches in the hardware.

  It can be seen from Table 1 that six triggers have been defined, all of which are set by the same clock, SYSCLK$. The implications of the definition of a trigger can be seen in the following: A path exists to block B3, which alters COMPL, and another to block C4, which tests COMPL. Since the effect of the assignment in block B3 does not take place until all paths have been completely processed in parallel for a given clock interval, it follows that the test in block C4 depends upon the value of COMPL at the beginning of the clock interval . The assignment in block B3 can, therefore, only affect tests made in the next or later clock intervals.

  Triggers can be defined to have various set-reset properties. The triggers of the example are defined as on-over-off, implying that a trigger is set to the value one if there is both a set and a reset during the same clock interval and that the trigger retains its value from the previous clock interval unless there is an explicit set or reset during this clock interval. The equivalent logic used for an on-over-off trigger is shown in Fig. 4(a).

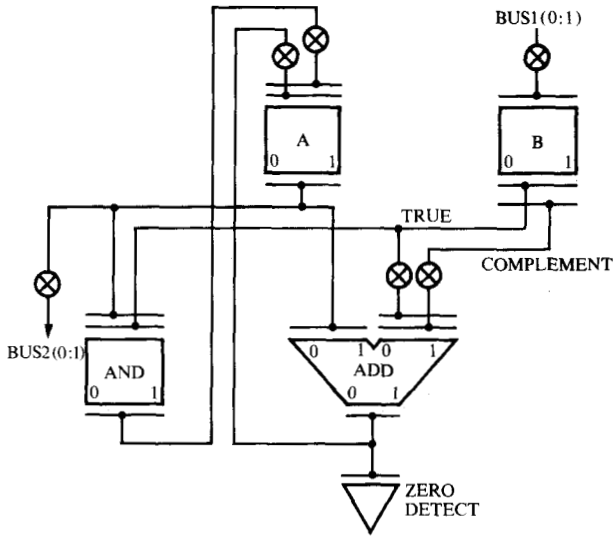
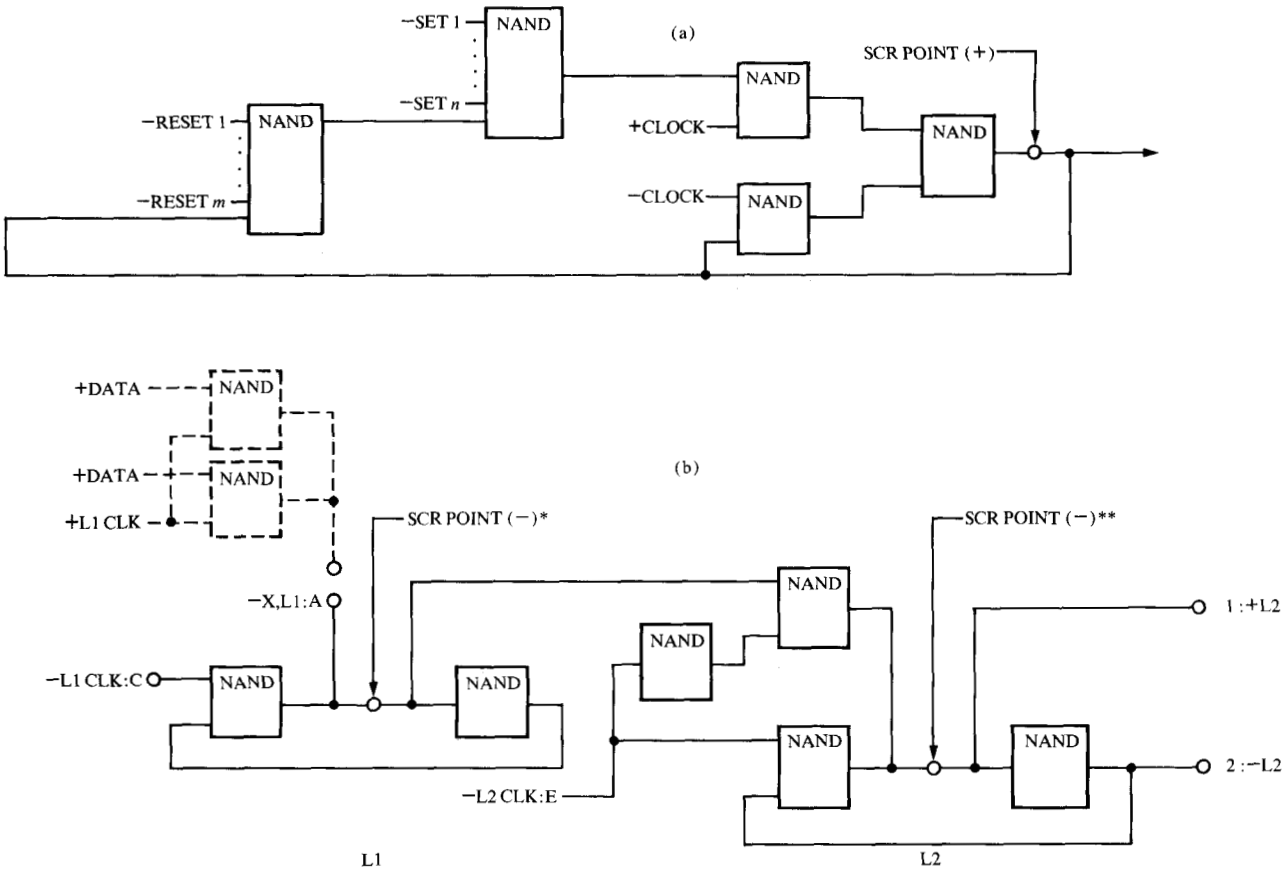


Figure 3 Data flow for sample machine.

- *Signals,* which are facilities that change value immediately upon execution of assignment blocks containing assignments to the signals. All signals are reset to zero at the beginning of each clock interval. During any clock interval, a signal can be set to the value one by assignment statements for that signal. Assignment statements which set a signal to zero have no effect on the signal value. Thus the value of a signal at the end of a clock interval is zero unless at least one set-to-one has occurred. Any decision block that refers to a signal is not executed until all assignment statements in parallel paths which can affect that signal have been executed. There are obvious constraints upon the order in which signals are used in assignment and decision blocks if deadlocks are to be avoided. As an example of signal referencing, the exit chosen for decision block D3 depends upon the results of assignment block D4 in the same clock interval.



*NET IS NAMED BY EXTERNAL NET ATTACHED TO PIN A
**UNUSED IN BOOLEAN COMPARISON OF SAMPLE MACHINE BECAUSE L2 CLK IS TIED ACTIVE

Figure 4 Expansion rules for storage devices. (a) On-over-off trigger (flowcharts); (b) double latch (hardware).

Signals are used for primary inputs and primary outputs. Signals are also used to serve as intermediate values during a clock interval, often resulting in simplification of the flowcharts. They are then known as "internal signals."

Table 1 defines nine input signals, two internal signals, and two output signals. (A don't-care signal, which is discussed under "Relaxation of requirements," also appears in the table.) All signal names end in $ by convention.

• *Hardware*

A hardware implementation of the flowchart of Fig. 2 is shown in Figs. 5(a) and 5(b). A number of implementation errors have been intentionally introduced into the design. It will be the function of the sample Boolean comparison run, described later, to detect the errors.

The logic of the sample machine is implemented in a technology consisting of AND-INVERT (NAND) circuits with dotting. A dot (block outputs wired together) performs an AND function (in positive logic terms) or a −OR function (in negative logic terms). The six triggers of the flowcharts have been implemented with seven double latch devices L1–L2, where the first latch is driven by L1 CLOCK and the second latch is driven by L2 CLOCK. All data ports of a double latch are located externally to the double latch, are dotted together, and are fed into the first latch via pin A. The internal logic of a double latch is shown in Fig. 4(b). This double latch is a simplification of the shift register latch used in the 3081.

• *Signal correspondence and segmentation*

The SCR list (Table 2) is a file created by the engineer to establish correspondences between facilities of the flowcharts and nets of the hardware. Facility names are printed with a four-digit index in order to identify each bit within a multi-bit facility. These correspondences must be created for all primary inputs, primary outputs, and latches; but they may also be created for internal signals.

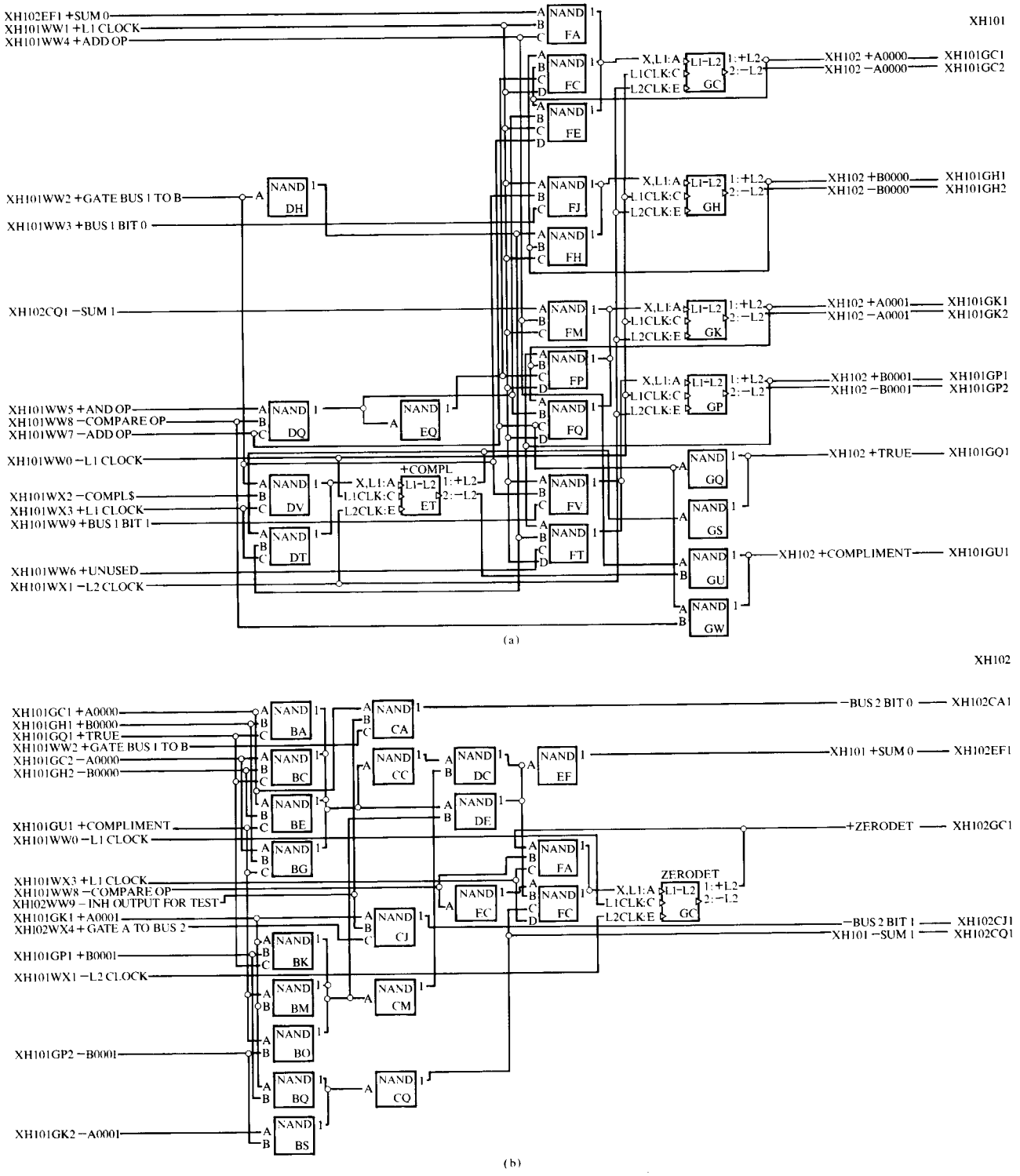


Figure 5 (a) Low-level logic diagram for sample machine: Registers; (b) low-level logic diagram for sample machine: Adder.

A sign is inserted in front of each net name in the SCR to indicate the polarity of the net when it is active; a facility is always assumed positive when active. The engineer inserts an S in the control field of any SCR entry for which a Boolean comparison is desired. All latches and outputs have been so marked in Table 2.

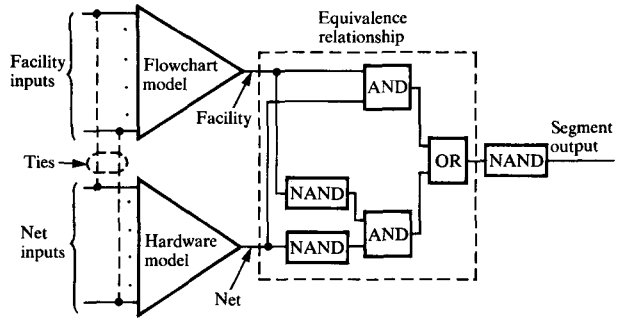


Figure 6 Basic segment structure.

Table 2 Signal Correspondence Records (SCR) for sample machine.

Facility name	*Net name*	*Control*	*Don't care facility name*
ADD$0000	+XH101WW4		
ADD$0000	−XH101WW7		
AND$0000	+XH101WW5		
A0000	−XH101GCA	S	
A0001	−XH101GKA	S	
BUS1$0000	+XH101WV3		
BUS1$0001	+XH101WW9		
BUS2$0000	−XH102CA1	S	
BUS2$0001	−XH102CJ1	S	
B0000	−XH101GHA	S	
B0001	−XH101GPA	S	
COMPARE$0000	−XH101WW8		
COMPL$0000	−XH101WX2		
COMPL0000	−XH101ETA	S	
GTINB$0000	+XH101WW2		
GTOUTA$0000	+XH102WX4		
SYSCLK$0000	+XH101WW1		
SYSCLK$0000	−XH101WW0		
SYSCLK$0000	+XH101WX3		
TIE1	+XH101WW6	1	
TIE2	−XH101WX1	1	
TIE3	−XH102WW9	0	
ZERODET0000	−XH102GCA	S	DONTCARE1$0000

Primary inputs of the hardware which are either unused (*e.g.*, XH101WW6) or not modeled in the flowcharts (*e.g.*, XH102WW9 and XH101WX1) are set to appropriate constant values. These constant values are specified in the SCR by inserting a zero or a one in the control field of an SCR entry. The constant in the control field establishes the value of the facility specified in the SCR entry. The sign in front of a net must be considered when ascertaining the effect of a constant control field value on a hardware net.

The net XH102WW9, which is provided for hardware testing purposes, is excluded from the flowcharts because entry of test logic in the flowcharts compromises their readability. This omission is perfectly legitimate if means exist which are not dependent upon Boolean comparison for verifying test logic. In the case of the 3081, test logic is validated by programs which verify conformance to a set of design rules.

The sample machine used in this paper requires two clocks (L1 CLOCK and L2 CLOCK) to drive the double latches, as is the case in the 3081. Since the functions of the two latches in each double latch device are identical except for an offset in time, it is possible to represent a double latch by a single trigger driven by one clock (SYSCLK$) in the flowcharts. Since all triggers in the sample machine flowcharts are driven by the same clock, it is not necessary to explicitly mention clocks in the flowcharts. For purposes of Boolean comparison, the L1 latch is compared to the trigger. The effect of the L2 latch is eliminated by tying the L2 clock (at XH101WX1) to the active state, allowing information to pass through the L2 latch. In the case of the 3081, this practice is followed generally because the L2 latch design is adequately checked out by test generation programs.

A segment (Fig. 6) is built by the segment builder of Fig. 1 for each selected SCR entry. A segment consists of all logic which drives the facility in the internal storage model for the flowcharts, all logic which drives the net in the internal storage model for the hardware, and logic that produces the inverse of an equivalence relationship between the facility and the net. The equivalence relationship is (facility output = net output) if the net sign in the corresponding SCR entry is positive and (facility output $\neq$ net output) if the net sign is negative. The "backward" search for driving logic does not trace through facilities or nets in the SCR, but it does continue tracing until all logic blocks in the segment are driven by other logic blocks in the segment or by facilities or nets in the SCR. The implications of any inputs with constant values are carried as far forward as possible in the segment. Any block which has a constant value as a result is dropped from the segment. If the backward search either reaches a primary input not in the SCR or detects a loop, the segment is invalid and is not subjected to Boolean comparison.

Boolean comparison runs for the 3081 are made for individual hardware modules of approximately 30 000 circuits each; therefore, the SCRs are structured on hardware module boundaries.

- *Boolean comparison*

For each segment, the Boolean comparison algorithm seeks patterns for the segment inputs which yield the value one at the segment output. The values applied to the facility inputs and the net inputs of the segment are constrained by the relationship (facility input = net input) if the SCR net sign

is positive and the relationship (facility input ≠ net input) if the sign is negative. In effect, the facility inputs are tied to the corresponding net inputs.

Any pattern producing a segment output value of one is a counterexample which proves that the two sides of the segment are not equivalent. All counterexamples are listed up to an engineer-specified maximum number. In most counterexamples, many of the inputs are marked with – to indicate that the value of this input is irrelevant to the counterexample. If no counterexamples are found, the two sides are equivalent.

Table 3 shows the Boolean comparison results for the sample machine. Model 1 names refer to flowchart facilities and model 2 names refer to hardware net names. Segments A0000, BUS2$0001, B0001, and ZERODET0000 show Boolean equivalence. Segment B0000 is not shown because lack of a primary input (net XH101WW3) in the SCR precluded segment building.

Segments A0001, BUS2$0000, and COMPL0000 are all nonequivalent. Counterexamples are listed for each of these segments. Counterexamples include the values of the net and facility outputs and the values of the inputs. Any input which exists in the flowchart side of the segment but not in the hardware side has the hardware net marked with an asterisk, and vice versa.

Table 3 Boolean Comparison results for sample machine.

```
THE FOLLOWING ARE EQUIVALENT
  MODEL 1                OUTPUT =  A0000
  MODEL 2                OUTPUT = -XH101GCA
THE FOLLOWING ARE NOT EQUIVALENT
  MODEL 1                OUTPUT =  A0001
  MODEL 2                OUTPUT = -XH101GKA
   I/O  MODEL 1            MODEL 2              COUNTEREXAMPLES 1 TO 6
    O   A0001                                   01101 1
    O                      -XH101GKA            01101 1
    I   ADD$0000           -XH101WW7            11111 1
    I   AND$0000           +XH101WW5            ------ -
    I   A0001              -XH101GKA            00011 1
    I   B0001              -XH101GPA            01110 0
    I   COMPARE$0000       -XH101WW8            ------ -
    I   COMPL0000        * XH101ETA             -10-1 0
    I   SYSCLK$0000        -XH101WW0            11111 1
THE FOLLOWING ARE NOT EQUIVALENT
  MODEL 1                OUTPUT =  BUS2$0000
  MODEL 2                OUTPUT = -XH102CA 1
   I/O  MODEL 1            MODEL 2              COUNTEREXAMPLES 1 TO 2
    O   BUS2$0000                               10
    O                      -XH102CA 1           10
    I   A0000              -XH101GCA            11
    I  *GTINB$0000          XH101WW2            01
    I   GTOUTA$0000      * XH102WX4             10
THE FOLLOWING ARE EQUIVALENT
  MODEL 1                OUTPUT =  BUS2$0001
  MODEL 2                OUTPUT = -XH102CJ1
THE FOLLOWING ARE EQUIVALENT
  MODEL 1                OUTPUT =  B0001
  MODEL 2                OUTPUT = -XH101GPA
THE FOLLOWING ARE NOT EQUIVALENT
  MODEL 1                OUTPUT =  COMPL0000
  MODEL 2                OUTPUT = -XH101ETA
   I/O  MODEL 1            MODEL 2              COUNTEREXAMPLES 1 TO 2
    O  COMPL0000                                01
    O                      -XH101ETA            01
    I   COMPL$0000         -XH101WX2            01
    I   COMPL0000          -XH101ETA            --
    I   GTINB$0000         +XH101WW2            11
    I   SYSCLK$0000        -XH101WW0            11
THE FOLLOWING ARE EQUIVALENT
  MODEL 1                OUTPUT =  ZERODET0000
  MODEL 2                OUTPUT = -XH102GCA
  DON'T CARE MODEL       OUTPUT =  DONTCARE1$0000
```

• *OResults analysis*

The first step of results analysis is to eliminate any SCR errors that preclude segment building (*e.g.*, B0000). This is followed by processing of Boolean mismatches. A mismatch between the hardware inputs and flowchart inputs of a segment—indicated by asterisks in the result—is often the source of a Boolean mismatch. Analysis of the results for segment BUS2$0000 leads to the conclusion that net XH101WW2 was erroneously used at block XH102CA instead of net XH102WX4. It occasionally happens, however, that a mismatched input has no effect on the value of the segment output and, therefore, presents no problem. In the segment for A0001, COMPL is an input which does not appear in the hardware for that segment because A0001 is actually independent of the value of COMPL. Therefore, the asterisk in the results for A0001 does not represent a problem.

After analysis of any input mismatches of a segment is complete, the counterexamples are examined. If a counterexample represents a state which is significant to the sequencing of the machine, then a logic change to the hardware is required; otherwise, a change is made to the flowcharts as described in the subsequent section "Relaxation of requirements." The counterexamples for segments COMPL0000 and A0001 can be eliminated by inverting the signal phase at pin XH101DVB and by using the input rather than the output of block XH102CQ to drive pin XH101FMA. Experience on the 3081 has shown that the explicit counterexamples provided make it easy to find the cause of problems.

4. Special topics

• *Boolean comparison algorithms*

Both DBA and DBA/ESP use iterative application of a sum-of-products expansion theorem [16] to the function of a segment. The algorithm proceeds by assigning constant values (zero or one) to inputs of the segment [see Fig. 7(a)] in a manner which produces a tree where the nodes consist of functions [see Fig. 7(b)]. The top node of the tree is the segment function. The bottom nodes are constant functions of either zero or one. The other nodes contain intermediate reduced functions. Each node represents a simple reduction of the node immediately above it. At each node, the algorithm chooses heuristically one input variable of the function at that node for use in reduction of the function. Each node

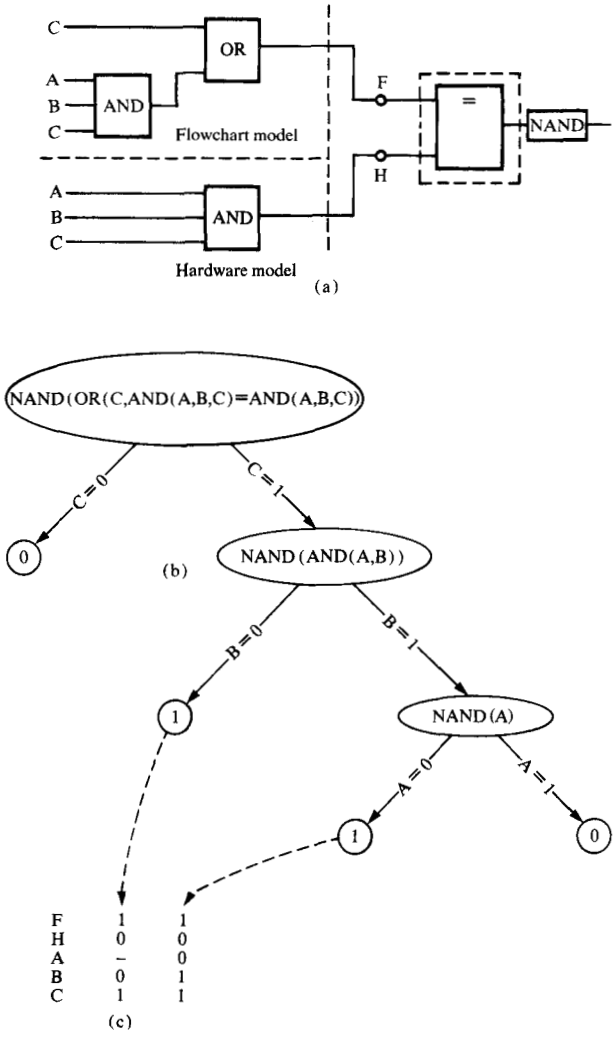


Figure 7 Examples of differential Boolean analyzer (DBA): (a) segment model; (b) tree; (c) counterexamples.

has two branches leaving it, one in which the chosen input is assigned the value zero and one in which the chosen input variable is assigned the value one.

A reduction is accomplished by carrying out the immediate implications of an assignment of zero or one to the chosen input variable. For example, any AND in the function is replaced by the value zero if one of the legs of the AND acquires the value zero, or by the value one if all of the legs of the AND acquire the value one. When the output of any primitive function (*e.g.*, AND, OR, N) acquires a value of one or zero, the reduction process continues by carrying out the immediate implications of this assignment. DBA/ESP additionally recognizes whether any intermediate function is identical to any intermediate function previously processed for that segment; if so, the algorithm backs up immediately. Each bottom node which contains the constant function one represents a counterexample. The counterexample consists of the values assigned to the input variables [see Fig. 7(c)] along the path from the top node to the particular bottom node. Each counterexample also includes the values of the inputs to the equivalence relationship [F and H in Fig. 7(a)].

The practical nature of the algorithms is important. Since the Boolean comparison problem is known to be NP-complete [17], one might anticipate that execution times would increase exponentially with segment size. However, since engineers reduce complexity by imposing some degree of orderliness upon all areas of the logic in order to achieve a comprehensible design, there is usually sufficient regularity for the algorithms to perform efficiently. A manual assist to input selection is provided for those few segments where the heuristic algorithms for selection of inputs are not efficient.

• *Relaxation of requirements*

The Boolean comparison program can handle certain cases where the requirements described in the section "Boolean comparison concepts" are relaxed. As previously shown, SCR entries, usually unused nets or test logic in the hardware, may be tied to one or zero. Also, any individual facility may be modeled by multiple nets in the hardware for powering purposes.

An additional technique is provided for handling cases where the flowchart and hardware logic for a segment differ only for machine states which are unreachable during proper operation of the machine. If a counterexample generated by the system for a given segment represents an unreachable state, two options are available to the engineer. The first is to change the flowcharts (or even the hardware) so that the flowcharts agree with the hardware. This is the preferred method because it minimizes the risk of error when future design changes are made. The second option is to define a "don't-care" signal in the flowcharts which is set by conditions that define unreachable states of the machine. A don't-care signal is associated with a segment by entry of the don't-care signal name in the don't-care field of the SCR entry. Usually, don't-care signals are only defined as needed in order to deal with specific mismatches for unreachable states.

The segment (Fig. 8) generated when there is a don't-care is expanded to include all logic obtained by a backtrace from the don't-care signal in the internal flowchart model. Logic is also added which forces the segment output to zero if the don't-care output assumes the value one. Therefore, no counterexample can be generated which falls within a don't care.

As an example, close inspection of the set conditions for the ZERODET facility in the sample machine reveals that ZERODET is set to 1 or 0 depending on whether or not SUM$=0 in the hardware and is left unchanged in the flowcharts if ADD$=1 and COMPARE$=1. However, if it is stated that ADD$ and COMPARE$ are never activated at the same time in the real machine, then such counterexamples represent unreachable machine states. The preferred technique for eliminating the counterexample is a change to the flowchart (Fig. 2) which eliminates the branch from block C2 to block D3, introduces a second copy of block C2 at another location, C2′, and establishes the path A1–C2′–D3.

For purposes of the example assume that the alternative technique of a don't-care signal is used. Figure 9 shows appropriate don't-care conditions in flowchart form for setting the new signal DONTCARE1$. DONTCARE1$ is entered in the don't-care field of the SCR for the facility ZERODET (Table 2). It can be seen from Table 3 that no counterexamples are produced for ZERODET although the flowcharts and hardware do not agree precisely in the Boolean sense.

By various techniques, don't-care signals can be incorporated into the cycle simulation runs to assure that there has not been overspecification of the don't-cares. In the case of the 3081, this has been done with some but not all don't-cares.

Experience on the 3081 has demonstrated that it is easy to keep the number of don't-cares to a very low level if care is taken to keep the hardware and flowcharts in step as each is created. If hardware is designed with little regard for the details of the flowcharts, then many don't-cares will be required.

- *Internal signals*

As previously mentioned, the SCR may contain correspondences between internal signals and intermediate nets of the hardware. If the logic producing the internal signal shares inputs with the logic driven by the internal signal, the internal signal may not be able to take on both the value zero and the value one independently of the values assigned to the common inputs. A don't-care may then be necessary for the driven segment in order to exclude errata for impossible patterns of the common inputs and internal signal. Consequently, internal signals are entered in the SCR only where the need to improve performance or to reduce the complexity of the Boolean comparison output provides clear justification.

5. Concluding remarks

Boolean comparison has been successfully used on the IBM 3081; all of the approximately 500 000 circuits subjected to Boolean comparison were brought into agreement with the flowcharts. While many factors contributed to this success, the efficiency of the algorithms and the completeness of the process must be identified as being the most important.

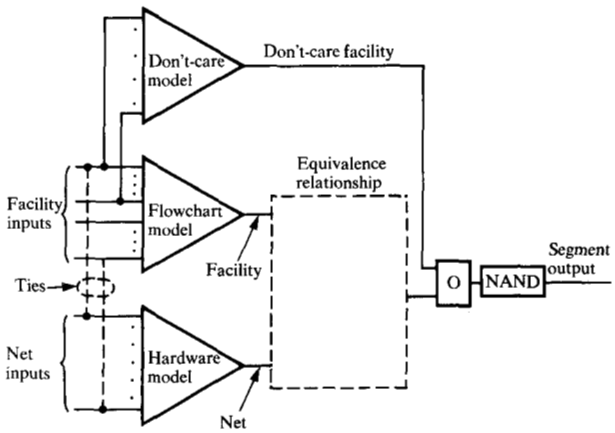


Figure 8 Structure of segment with don't-care.

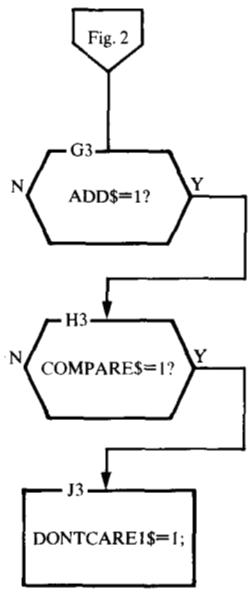


Figure 9 Don't-care flowchart for sample machine.

6. Acknowledgments

While it is not feasible to recognize all contributions, the authors wish to acknowledge the technical contributions of the following people to the Boolean comparison system used on the IBM 3081: David D. Cheng, Wilm E. Donath,

Charles W. Evans, Catherine C. Koo, Hillel Ofek, John G. Rogers, and A. Patrick Torney. In addition, this development was significantly influenced by concepts of J. Paul Roth and Albert Brown. Finally, the management support of Albert Brown, John Chapman, Brian R. Golnek, Judith M. Lauch, and Michael Monachino is acknowledged.

References

1. R. N. Gustafson and F. J. Sparacio, "IBM 3081 Processor Unit: Design Considerations and Design Process," *IBM J. Res. Develop.* **26,** 12–21 (1982, this issue).
2. Michael Monachino, "Design Verification System for Large-Scale LSI Designs," *IBM J. Res. Develop.* **26,** 89–99 (1982, this issue).
3. Robert B. Hitchcock, Sr., Gordon L. Smith, and David D. Cheng, "Timing Analysis of Computer Hardware," *IBM J. Res. Develop.* **26,** 100–105 (1982, this issue).
4. G. J. Parasch and R. L. Price, "Development and Application of a Designer Oriented Cyclic Simulator," *Proceedings of the 13th Annual Design Automation Conference*, ACM and IEEE, New York, 1976, pp. 49–53.
5. J. P. Roth, "Systematic Design of Automata," *AFIPS Conf. Proc., Fall Jt. Computer Conf.* **27,** Part I, 1093–1099 (1965).
6. J. P. Roth, "Hardware Verification," *IEEE Trans. Computers* **C-26,** 1292–1294 (1977).
7. N. Kawato, T. Saito, F. Maruyama, and T. Uehara, "Design and Verification of Large-Scale Computers by Using DDL," *Proceedings of the 16th Design Automation Conference*, San Diego, 360–366 (1979).
8. F. Maruyama, T. Uehara, N. Kawato, and T. Saito, "Hardware Verification and Design Error Diagnosis," *Digest of Papers—The 10th International Symposium on Fault-Tolerant Computing*, Kyoto, Japan, 1980, pp. 59–64.
9. S. B. Akers, "A Procedure for Functional Design Verification," *Digest of Papers—The 10th International Symposium on Fault-Tolerant Computing*, Kyoto, Japan, 1980, pp. 65–67.
10. W. E. Donath and H. Ofek, "Automatic Identification of Equivalence Points for Boolean Logic Verification," *IBM Tech. Disclosure Bull.* **18,** 2700–2703 (1976).
11. J. P. Roth, "Diagnosis of Automata Failures: A Calculus and a Method," *IBM J. Res. Develop.* **10,** 278–291 (1966).
12. J. P. Roth, "Algebraic Topological Methods for the Synthesis of Switching Systems I," *Trans. Amer. Math. Soc.* **88,** 301–326 (1958).
13. H. Halliwell and J. P. Roth, "SCD, a System for Computer Design," *IBM Tech. Disclosure Bull.* **17,** 1517–1519 (1974).
14. J. P. Roth, "VERIFY: An Algorithm to Verify a Computer Design," *IBM Tech. Disclosure Bull.* **15,** 2646–2648 (1973).
15. F. C. Hennie, *Finite State Models for Logical Machines*, John Wiley & Sons, Inc., New York, 1968.
16. C. E. Shannon, "A Symbolic Analysis of Relay and Switching Circuits," *Trans. AIEE* **57,** 713–723 (1938).
17. H. R. Lewis and C. H. Papadimitriou, "The Efficiency of Algorithms," *Scientific American* **238,** 96–109 (January 1978).

Received October 27, 1980; revised August 11, 1981

Gordon L. Smith is located at the IBM Data Systems Division laboratory and Ralph J. Bahnsen is located at the IBM General Technology Division laboratory, both in Poughkeepsie, New York 12602. Harry Halliwell is at the IBM United Kingdom Laboratories Limited, Hursley Park, Winchester, Hampshire SO21 2JN, England.