J. F. McCabe
A. Z. Muszynski

A Bipolar VLSI Custom Macro Physical Design Verification Strategy

The level of complexity and the turn-around time associated with the development of custom bipolar VLSI chips have defined the need for a highly structured physical and electrical design validation approach which can guarantee fully functional first-pass chips, yet be flexible enough to allow logical and physical designers the latitude necessary to achieve specified cost and performance objectives. This paper describes such a design verification strategy and its implied constraints on chip design. The rationale for comparing the logic equivalence of the high-level logical models to the low-level-device physical models is presented, a description of the hierarchical logical-to-physical and electrical checking is given, and its impact on cost and complexity is examined.

Introduction

The ideas discussed in this paper were evolved over a three-year period from 1978–1980, and were applied to the design of a *VLSI microprocessor*, various aspects of which are described elsewhere in this issue [1–3]. In this paper we focus on the design-rules generation and the physical and electrical design verification strategy. Many of the software tools are similar if not identical to those used by IBM's Engineering Design System [4] for masterslice (*i.e.*, gate-array) technologies. The significance of the methodology described lies in the application of some of these tools, as well as structure and rules, to a custom bipolar design from the chip level down to the individual devices. The objective was to define and implement a set of design and checking rules and procedures which would result in a fully functional [5] first-pass custom design, but which would also give the logic, circuit, and chip designers flexibility in logic design documentation and in physical design, power, performance, and silicon area tradeoffs.

An initial set of objectives were established early in 1978 and were followed through to the successful development of the multichip module microprocessor. These objectives, which were key to the design and verification strategy, included the following:

1. Define and design a limited number of building-block circuits with complete freedom of design within the semiconductor process ground rules, using a limited set of device structures.
2. Generate rules governing the application of the building-block circuits.
3. Construct functional macros [6] using the building blocks previously defined.
4. Construct the global chip design using the functional macros and building blocks previously defined.
5. Provide the logic designer with logical macros. However, allow the physical designer to lay out the macros with a choice of physical macro subtypes (same building blocks, different spacing between blocks), or the option to break the macro up into its building blocks if that is more efficient for layout purposes.
6. Make the checking independent of wiring strategy (automatic wiring and/or manual) and provide wiring guidelines but do not constrain the wiring by machine-coded rules.
7. Provide rules-driven systematic checking, utilizing a hierarchical approach wherever possible.
8. Provide rules and data bases compatible with semiconductor manufacturing release procedures and upward

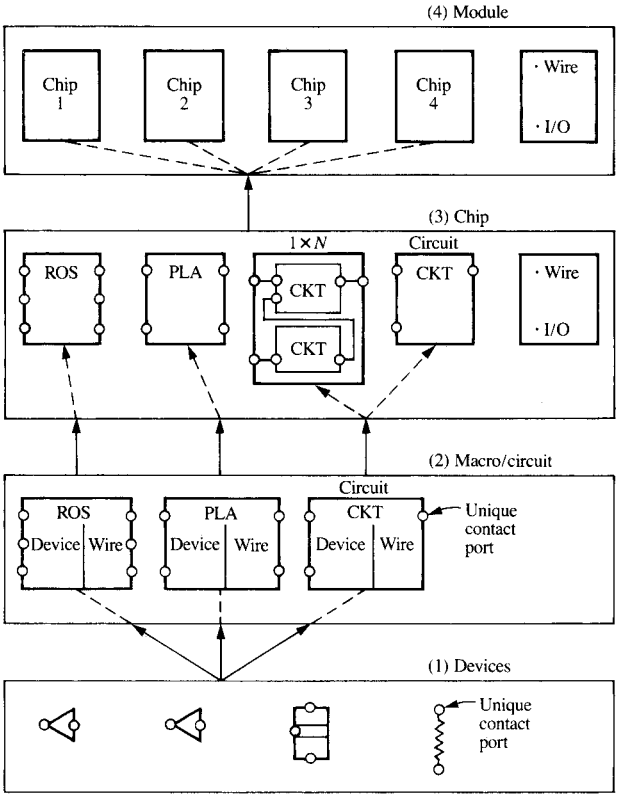

Figure 1 Hierarchical representation of the processor module, consisting of four custom bipolar chips, from the single module to the devices on each chip. (1) Symbolizes the device library where different devices have the contact ports uniquely identified. (2) Symbolizes the primitive circuit and macro library. Devices are wired together to create primitives and devices and/or primitives are wired together to create macros. (3) Symbolizes the chip, with its primitive macros and chip I/O connections wired together, as well as the power distribution wiring. (4) Symbolizes the module with its four chips, signal wiring, I/O connections, and power distribution.

compatible with the next level of package (multichip module).

9. Provide a checking system such that follow-on applications of these macros can be software validated via existing rules.

The key to the successful design was breaking things up into manageable parts (structured design). From the processor (engine) architecture, discussed in the paper by Campbell and Tahmoush [2], a logical data flow was defined, as well as logical and physical circuit and macro objectives. A bipolar custom macro design approach was chosen in order to meet the performance and density objectives while limiting the physical variations. The physical design, placement, and wiring plan used a combination of automatic and custom approaches which consisted of computer-assisted placement tools coupled with automatic as well as manual wiring performed to guidelines, not to predefined machine-coded rules. This design was checked to rules independent of the design tools. Special emphasis was placed on making the rules and checking methodology correct the first time, in order to meet the overall objective of first-pass operational parts. The rules, physical design, and checking were performed in a hierarchical manner, with the semiconductor device as the smallest wireable element.

Table 1 Primitives and macros.

1. *Random logic circuits* made uniquely from devices:

   AND-INVERT (AI), EXCLUSIVE-OR (XOR), AND-OR-INVERT (AOI), SHIFT REGISTER LATCH (SRL), OFF-CHIP RECEIVER (OCR), PUSH-PULL DRIVER (PPD), OPEN-COLLECTOR DRIVER (OCD).

   (These circuits are called random-logic primitives in subsequent discussions.)

2. *1 × N macros* made from the above primitives.
3. *PLAs* made from a combination of logic circuits and personalized devices.
4. *ROS* made from a combination of decode and powering circuits and personalized devices.

Under control of the methodology, the chip model can exist in any one of the following four states:

Physically	*Logically*
Chip (one-block representation with I/O)	Chip model
Interconnected macro logic	Macro models
Interconnected primitive logic	Micro logic
Interconnected devices	Micro logic

Figure 1 illustrates the hierarchical representation. The chip is composed of ROS, PLA, 1 × N macros, primitive circuits, chip I/O ports, and power distribution, plus signal wires. The 1 × N macros are made up of primitive circuits and represent a byte's worth or less ($N \leq 9$, including parity) of function. The primitive circuits, PLAs, and ROS are constructed from unique combinations of devices.

In this case the rules describing the macros and primitives, both logically and physically, had to be generated because, although the silicon process ground rules were similar to those of existing designs, the devices and circuits were unique to this project. A listing of the primitives and macros is given in Table 1. Additional information on the circuits may be found in [1].

In the following sections of this paper, the rules methodology, the common data base, the physical verification, the

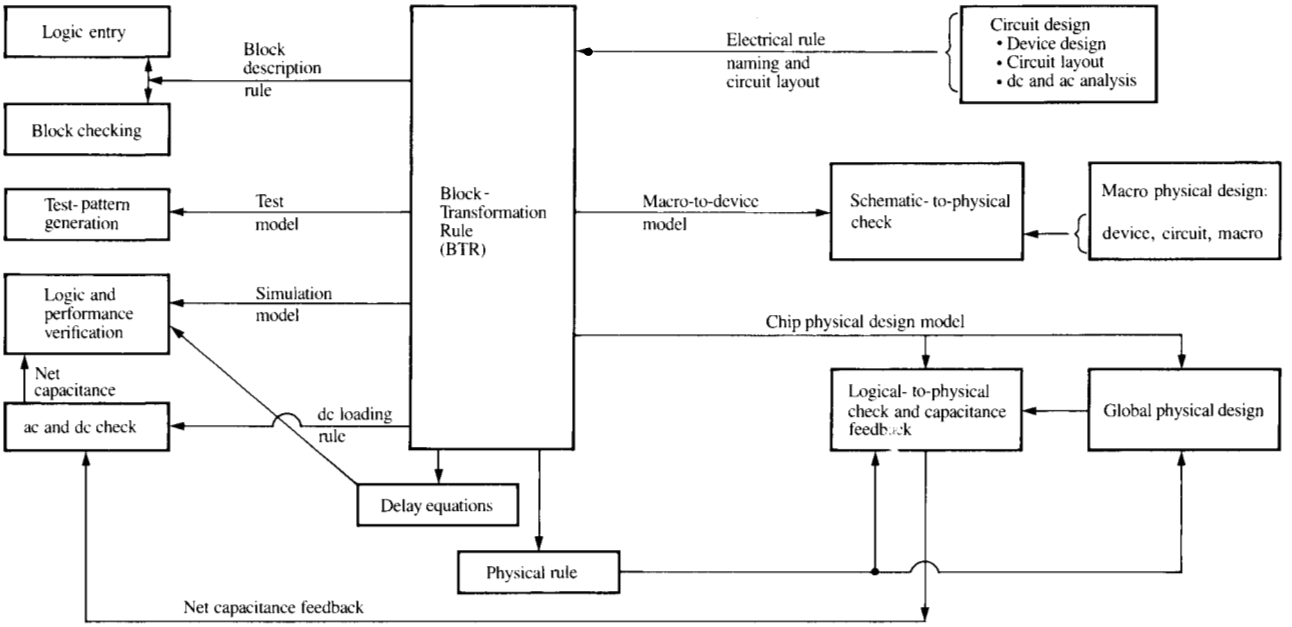

Figure 2 Block-Transformation Rule application flow. Shows how this rule ties (glues) together the physical and logical rules associated with the design and validation software tools.

electrical verification, the electrical checking procedures, and the performance verification aspects of the physical design verification strategy are described in some detail. This is followed by a discussion of some key results and conclusions of the program.

Test generation was based on a design technique referred to as Level-Sensitive Scan Design (LSSD) [7]. The methodology used for the logical verification of the VLSI microprocessor design is discussed in the paper by Tran *et al.* [3], and the physical design graphical tools that were used are described in the paper by Mathews and Lee [1].

Rules methodology

The logic data base and rules structure were similar to those used in IBM's engineering design system [4, 8] but were uniquely tailored to meet the objectives of our hierarchical custom design, as previously defined. The hierarchical approach required the logical data base to be completely represented at several different physical levels (macro, primitive, and device). Since we wanted to maintain only the logic designer's macro input, the other levels were to be automatically generated by rules defining the content of the macros. These rules were unique in that they defined a macro to its device level, which is not generally the case with masterslice designs. This approach was taken because of the number of different silicon circuit designs.

With the language used to define the logic of the system, each logical element (random logic primitive, macro, ROS, and PLA) is described by identifying the block and defining its input and output nets. The block name points to a *block-transformation rule* (BTR). This is the focal-point rule for the methodology, and one exists for each primitive, macro, PLA, and ROS. As the name implies, the purpose of the block-transformation rule is to describe the transformation of a primitive or macro block into its physical and logical description. It also describes to the logic designer how to code the block into the logic description so that the block will be correctly interpreted by the design automation programs.

The BTR describes the logical and physical equivalents, as well as the electrical data, needed to transform the block into a format compatible with the following applications:

- Simulation, including delay simulation,
- Test-pattern generation,
- Physical design, and
- Physical design checking.

The BTR structure is hierarchical in nature, with macros pointing to primitives and primitives pointing to elemental logic functions, device elements, and electrical characteristics. Figure 2 illustrates how the block transformation rule acts like the technology glue, binding together the different aspects of design (circuit, logic, and physical) and validation

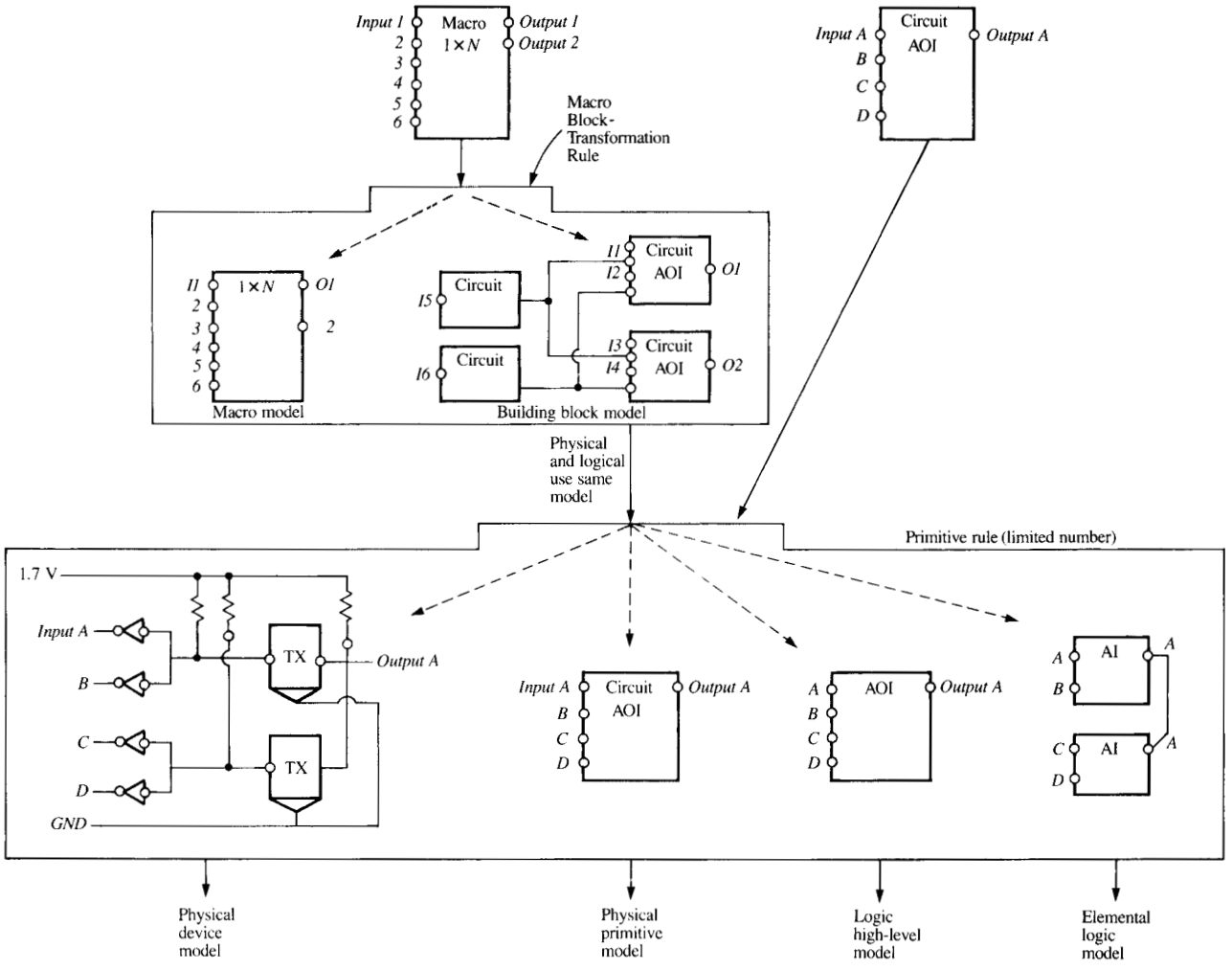

Figure 3 Block-Transformation Rule expansion methodology. A primitive circuit is transformed in one pass to its elemental devices or logic equivalent. A macro goes through two transformations.

(simulation, physical-to-logical, performance, and test-pattern generation). Figure 3 illustrates the hierarchical nature of the BTR structure.

Considerable care was taken to minimize redundancy and duplication, thereby reducing the possibility of errors as well as the number of BTR rules. The key electrical and delay pointers and logical equivalents are only defined once in the base-level primitive BTR. The language used for defining BTRs allowed looping; therefore only one primitive expression need be coded; then it is automatically repeated any number of times to produce from 1 to N of these circuits. The same coding technique allowed a single rule to represent a class of physical circuits and could select a single circuit automatically by inspection of I/O pins. This significantly reduced the number of rules and resulted in one rule per electrically different primitive or macro circuit. In our application, one macro rule represented up to 18 physical and logical variations, and the same model was used for both logical and physical processing.

The primitive BTR rule not only contained up to five physical variations but also contained different logical models (high-level and/or micro-block) depending on user and application program requirements. Single-block high-level models usually require less simulation time than the equivalent micro blocks and were used whenever efficiency was a prime factor.

Another significant point is the verification of the BTR rule. Test cases were generated which validated each model. However, an additional check was made to guarantee that

the models within the rule, which are usually generated independently, were functionally equivalent to one another. That is,

- The primitive, PLA, and ROS *logic* models were logically equivalent to their *physical* models, and
- The logic models (high-level and micro-block) were equivalent.

A prime example of the need to compare models is the "ROS," which consists physically of up to 50K bits in addition to decode and sense circuits. It is represented by two levels of transformation down to the device level. Its logic is, however, represented by a single standard high-level logic model which references the 50K personalized bit pattern. Reference [9] describes the ROS in detail. The physical model of the ROS was transformed to approximately 17 000 devices; these devices were transformed to their respective elemental logic functions,

Reverse diode = AND,
Forward diode = OR,
Transistor = I (Invert),

and were then compared, via a static-type logic analysis, to the single high-level logic function defined in the rule. This procedure was used for the more complicated circuits. Figure 4 illustrates this comparison.

The high-level model and logical micro blocks can be equated at the circuit level and/or at the chip level. We used the latter, and made a logical comparison of the high-level function of the chip *versus* the micro-block function of the chip.

Common data base

The *chip logical data base* defines the interconnections between the logical elements (primitives, $1 \times N$ macros, PLAs, ROS). By means of the block rules, previously defined, the primitive and $1 \times N$ macro logical elements are functionally defined. The logical functions of the PLA and ROS are defined by sets of 1/0 bit-pattern files for ROS [9] and for PLAs [10].

Via the block-transformation rules, the logical chip description is converted to several different but logically equivalent forms for use in different parts of the design and verification systems. Figure 5 illustrates where the different logic forms fit in the chip data flow. These forms are listed in Table 2.

Table 3 describes the hierarchical complexity of our application, from the one physical module to 15 000 equivalent AIs plus a 50K-bit ROS, and from one high-level model representing the total processor to 20 000 micro simulation blocks representing the one-module processor.

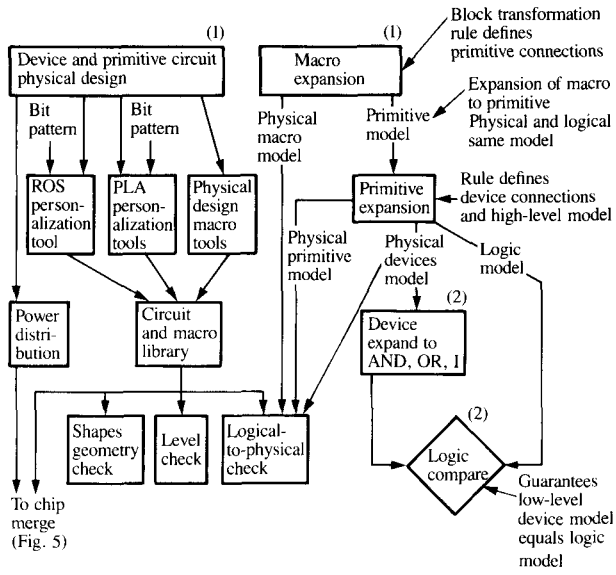

Figure 4 (1) Primitive and macro graphic design and checking; and (2) Macro physical model compared to the logical.

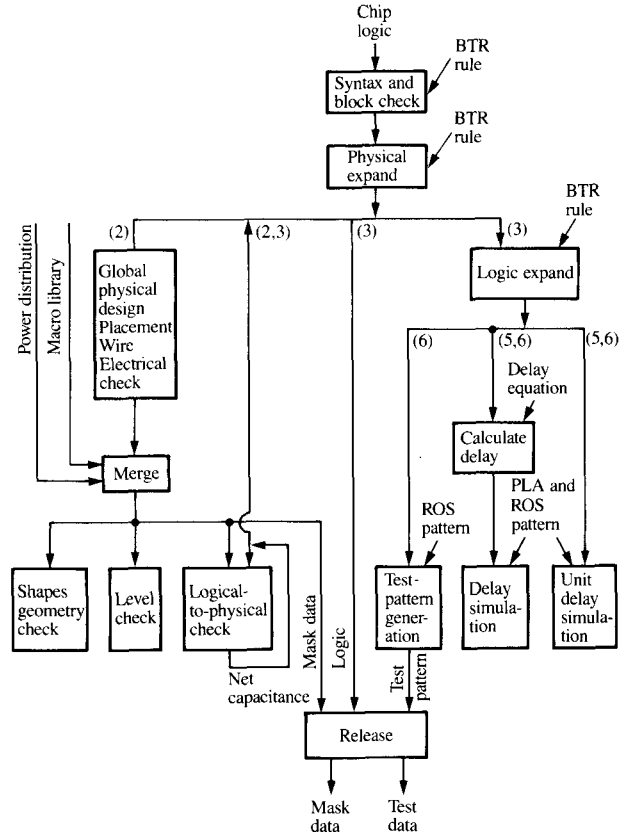

Figure 5 Chip design and validation flow: shows the merging of the macros and power distribution into the chip graphic design and the release to semiconductor manufacturing. Also shows the application of different logic expansions (the numbers refer to the forms described in Table 2).

Table 2 Hierarchical logical and physical logic forms.

Form	*Application*	*Comments*
1. Macro logic coded by logic designer.	Architectural verification.	Normally independent of the physical implementation.
2. Macro logic (physical).	Placement and wiring. Hierarchical logical-to-physical checking.	May be broken into separate parts with special consideration for drivers, receivers, and breaking macros into sections.
3. Primitive logic, PLA, ROS.	Chip level logical-to-physical checking.	
4. Device-level logic.	Device-level logical-to-physical checking.	Normally not done except on a single macro. Discussed in the section on logical-to-physical verification.
5. Primitive behavioral logic.	Simulation.	Examples are SRL, XOR, AOI, AI, PLA, and ROS.
6. Elemental logic.	Test generation and low-level simulation.	Examples are AI, OI, A, O, and ROS.

Table 3 Typical hierarchical complexity of processor models (four custom chips → one custom module).

Number of modules	—	1
Number of custom chips	—	4
Number of logic pages	—	150
Number of logic blocks coded by logic designer	—	2,000
Number of physical circuits placed and wired by chip designer	—	2,500
Number of circuits checked during logical-to-physical checking with only global metal	—	2,500
Number of circuits checked during real logical-to-physical checking with all metal	—	5,000
Number of equivalent three-input AI circuits	—	15,000 + 50K-bit ROS
Number of devices	—	100,000
Number of high-level models representing total processor	—	1
Number of functional high-level blocks simulated	—	5,000
Number of micro blocks simulated for one-module processor	—	20,000

The *physical design data base* is a common graphical language [11] compatible with IBM's design and checking tools as well as with mask fabrication. A library of nested devices, circuit primitives, and macros is maintained. The symbolic circuits (outline shape defining the circuit boundary) and their logic and power service terminals (ports) are positioned at specific locations on the chip image to create a "placed chip image." These circuits are then inter-wired and connected to chip input/output ports. The real library cells are then substituted for the symbolic ones and merged with the power distribution (network) to create the "wired chip image." Figure 5 illustrates how this fits into the chip data flow.

Physical verification

Physical checking falls into the following two essentially different parts:

1. Shape-geometry, which checks a shape width and area as well as spacing and overlap between shapes.
2. Connectivity (logical-to-physical), which compares the presence and interconnectivity of physical circuits, as defined in the graphic language, to the logical models. Forms 2 and 3 in Table 2 describe these models.

• *Shapes geometry*

For shape geometry, IBM's shapes checking design automation tool [12] was used to manipulate and check the graphic shapes. The checks themselves (types of checks and limits) were defined uniquely to meet the manufacturing process requirements as well as the device electrical requirements determined by statistical circuit analysis. In order to apply variable shape rules to different semiconductor structures as well as to assess that only the necessary shapes were present for a particular device, a unique form of device recognition was used which depended on symbolic levels coded into devices. This was not as fail-safe as true geometric recognition but significantly reduced the computer checking time.

Each different device (transistor, resistor, diode) had these special levels defined to its contacts. These were additionally encoded with a numeric attribute which differentiated between sizes of devices of the same type (T1, T2, R1, R2, etc.) and was used for logical-to-physical checking of primitive circuits.

- *Logical-to-physical verification*

This checking is performed hierarchically:

First—primitives and macros
Next—chip

- Check to macro ports (includes only global metal). A significant reduction in computer checking time was achieved over the all-metal case when performing this check. However, there were some detection limitations and the all-metal case was done when the chip was relatively error-free.
- Check to primitive ports (includes all metal).
- Backup capability to check to device contacts (includes all metal). This check was done only once to prove that the hierarchical concept was valid, and was demonstrated not to be required for every chip.

The primitive, macro, and chip models for these checks are created by expanding a single logical circuit or a chip of logic into its physical counterparts, as described in a previous section. (Refer to Figs. 4 and 5, macro and chip data flows.) When using this hierarchical approach, an additional shapes check *must* be included to ensure that the level-naming conventions were followed and that global metal does not touch internal circuit metal. IBM's USC shapes checking tool [12] was used to perform this checking. The objective was to include, in the primitive and chip-level models, *every type of signal and power connection made on the chip between device contact holes.* For a list of specific types of checks, refer to Table 4.

Electrical verification

The basic objective of electrical verification is to ensure the electrical functionality of a chip within specified operating limits, *i.e.*, power supply voltages, temperature, and input and output loading. The electrical verification methodology was designed to achieve this objective by detecting any of the following three possible failure modes: a *communication failure*, caused by noise introduction between a driver and a receiver, and by a possible specification mismatch between driving and receiving logic; a *functional failure*, caused by a logic block failing to perform its designed function; a *reliability failure*, caused by excessive voltages and currents on the chip. The methodology presented here departs significantly from the conventional rules-driven verification approach used in a masterslice (gate-array) environment. It is based on a detailed electrical analysis of the entire chip and chip carrier.

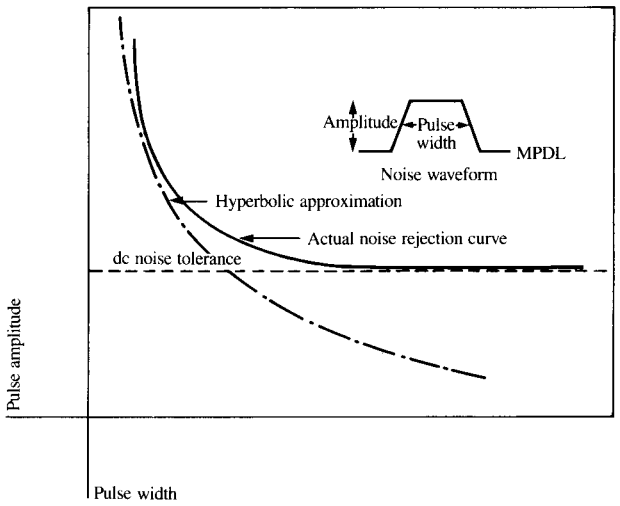

Figure 6 Noise rejection curve represents receiving circuit immunity to noise spikes. Use of the hyperbolic approximation permits us to develop equivalency between noise pulses of different duration. (Note: MPDL = Most Positive Down Level.)

Table 4 Contents of the physical models.

Primitive and macro: • Signal and power connections between device ports. • Device ports connected to primitive and macro signal and power ports. • Subcollector contact to +V, one per resistor. • Substrate contact to ground, at least one per primitive. • Within 1 × *N* macros, connections between primitive ports.
Chip macro level: • Signal connections between global macros and primitives. • Signal connections between global circuits and chip I/O.
Chip primitive level: • Signal and power connections between primitives (all 1 × *N* macros expanded to primitives). • Signal and power connections between primitives and chip I/O (this includes power distribution). • Power-protect diode connections to the power distribution network.
Chip device level (if used): • Same as chip primitive, except to device ports instead of primitives.

A *communication failure mode* is checked by comparing noise margin against estimated noise on a net-by-net basis. Although this concept is a rather simple one, its practical implementation was possible only by making several assumptions and approximations, which we now examine. The net noise margin, illustrated in Fig. 6, is a function of drivers and receivers on a net. The dc portion of a noise sensitivity curve is calculated by subtracting a driver output voltage level

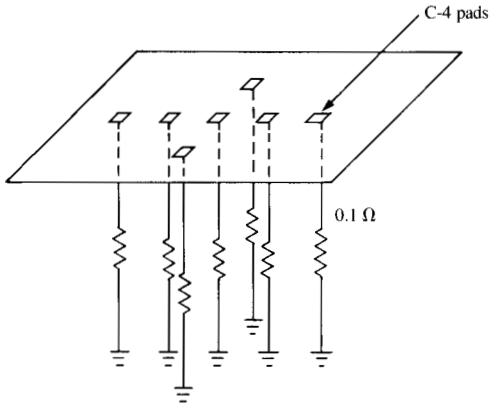

Figure 7 A simple module resistive model is used to establish relative voltages between C-4 pads.

from a receiver threshold voltage. The ac portion represents immunity to fast noise spikes and is assumed to be bound by a hyperbolic curve governed by the equation

$$K = (PW)(AMP),$$

where K is a constant, PW is a noise pulse width, and AMP is a noise pulse amplitude.

The hyperbolic noise sensitivity curve approximation is significant since it permits a development of equivalency between different pulse width noise spikes; *e.g.*, a receiver immune to a 1-V 1-ns noise pulse will also be immune to a 250-mV 4-ns noise pulse. By bringing all the ac noise components to an equivalent point at which the hyperbolic curve intersects the dc immunity line, the entire noise calculation can be carried out by dc analysis.

Another simplication is made by reducing multiple noise margins, corresponding to various combinations of drivers and receivers on a net, to a single worst-case noise margin. This is done to reduce the number of software iterations per net. The worst-case noise margin is calculated by subtracting the greatest of the driver output levels from the lowest of the receiver thresholds.

Each circuit is designed to drive a certain maximum number of loads, and the output voltage level is specified at that maximum loading. Since the output level improves as the loads decrease and a typical circuit loading is below maximum, an additional level of conservatism is built into the calculation.

The noise calculation methodology assumes that a net is at its down level. This assumption is justified since the typical worst-case down-level noise margin is about 130 mV as opposed to 600 mV at the up level. Therefore, if one assumes noise symmetry, noise violation should be first observed on the down level before the up level will be affected.

Next, we examine the nature and source of noise found on a chip and project it to a higher-level package (module). From a dc analysis point of view there are two contributing noise components: resistive voltage net drops and ground shifts.

The net resistive voltage drop is a function of the net topology. When several drivers are present in a net, voltage drops are a function of a particular driver being active. Thus, a checking routine must have the capability of iterating over each driver on a net.

The ground shift between a driving and a receiving circuit is caused by a ground power distribution loss. This loss is a function of switching activities on the chip, the major switching function being driver switching. Four distinct driver states corresponding to four processor clocks are recognized, resulting in four distinctly different values of the ground shift.

The ac noise generated on a chip comes from line-to-line capacitive and magnetic coupling (referred to as *crosstalk*) and from more diffuse global magnetic coupling between module substrate, chip power distribution, and signal nets. The resistive voltage drops resulting from switching drivers are covered by ground-shift noise and are considered dc since the duration of a resulting noise pulse exceeds the ac portion of a noise sensitivity curve.

The overall ac noise adds up to about 50 mV dc equivalent. Most of the ac noise estimation is based on a large-scale model simulation [13]. Although we presently have analytical tools capable of precise ac noise calculation, we feel that the additional gain in calculation accuracy does not justify an additional software simulation cost.

A *functional failure mode* is checked by comparing the macro power service terminal voltage requirement against the calculated voltage across these terminals. As in the case of a communication failure mode, the power noise sensitivity curve is used as a failure criterion; however, it is not derived on an individual macro basis but is global for all the circuits on a particular power distribution level.

A conservative approach is taken by disregarding the ac portion of a power noise sensitivity curve. The reason is that noise is generated primarily by switching drivers and therefore occurs simultaneously on signal nets and on a power distribution network. The ac portion of a noise rejection curve at the input terminal is strongly correlated to the ac

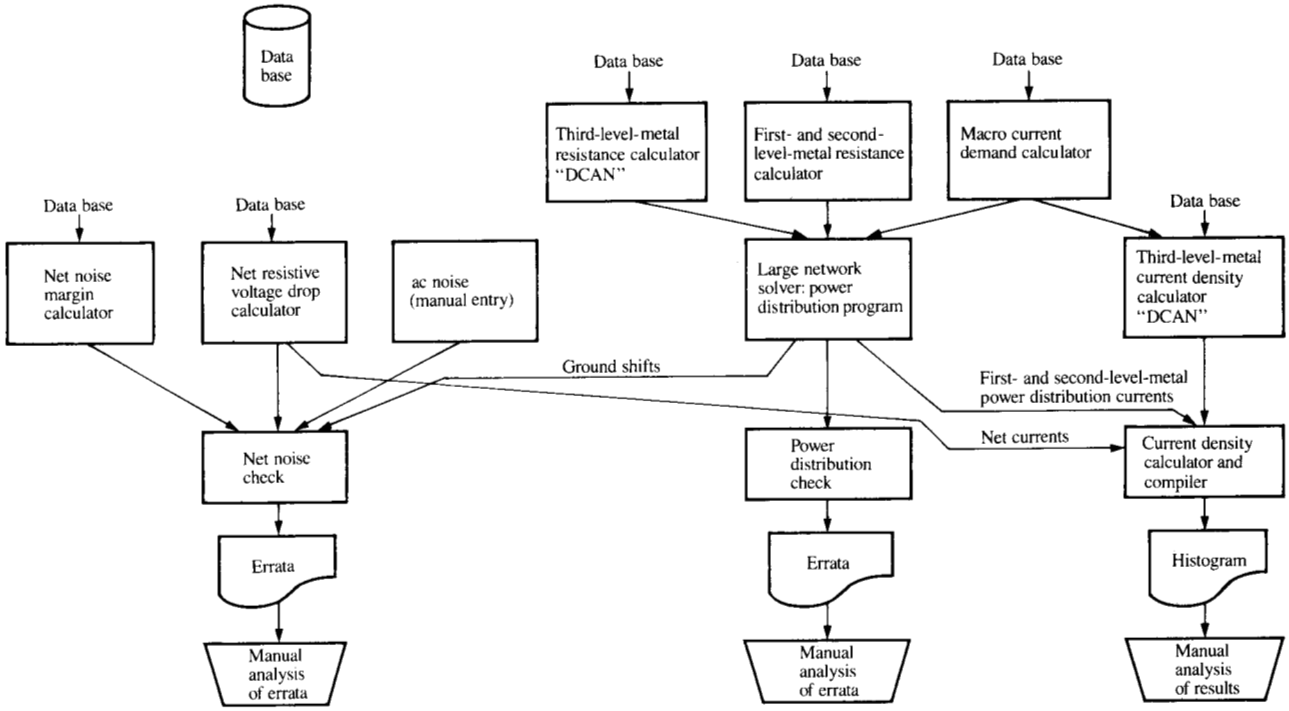

Figure 8 Set of procedures and routines follows three failure mechanisms: communication, function, and current density failure checks.

portion of a noise rejection curve of a macro power terminal. A double accounting of the ac noise may lead to an incorrect conclusion.

Power voltage drops for each macro are calculated by adding ground and voltage distribution drops for that particular macro. Voltage drops on ground distribution are calculated by subtracting the power service terminal voltage from the most positive C-4 [14] module connection. For other voltage distributions the difference is taken between a power service terminal and the most negative C-4 connection.

Power distribution analysis is carried beyond the chip level since the module substrate is the integral part of that distribution. A very simplified modeling of the package (see Fig. 7) is used to estimate the relative potential of C-4s with respect to one another.

Since the chip current demand and the power distribution voltage drops are functions of switching activities, the power distribution analysis is carried out four times, corresponding to four distinct logic states and driver switching activities (as in the case of the ground-shift calculation). The worst-case voltage drops are selected for the final functional failure mode analysis.

The ac noise contribution is estimated to be in the range of 20 mV of equivalent dc noise. This estimate applies to the 1.7-V and 5-V internal power distributions on chip. The 5-V off-chip driver power distribution is considerably larger (100 mV of equivalent dc noise).

A *reliability failure* (physical failure) mode is checked by comparing voltage and current densities against predetermined guidelines which guarantee adequate chip reliability. The voltage criterion is not checked during the chip verification stage. All the circuits are designed not to exceed a certain reverse bias voltage and isolation guidelines. The same applies to current densities on an individual circuit basis. However, a global connection current density check is required. Three items are verified: via, signal net, and power-metal current densities.

Electrical checking procedures

The overall electrical checking procedure is illustrated in Fig. 8. Three distinct paths correspond to the evaluation of the three failure modes previously discussed.

The dc signal-line analysis program checks for possible communication failure mode violations. It calculates worst-case noise margins, estimates resistive signal-line drops, and adds these to the other noise contributions—ground shift and a combined ac noise. The ground-shift voltage drops are supplied by the power distribution analysis routine. This entry mode can be bypassed and a fixed, manually calculated ground shift can be entered. This option was particularly

useful during a global wiring phase of a chip design when power distribution was not totally completed. It provided a quick estimate of possible wiring violations. The ac noise contribution of 50 mV is allocated for each net. Again, it is a manual entry and can be changed as more precise information becomes available.

The program indicates the nets where noise margin was exceeded. The errata consist of a plot of a net found in violation, a list of line resistances, and the common graphical language level (graphic representation) of each section of the net. Noise margins, resistive drops, ground shifts, an ac noise contribution estimate, and a macro block coordinate are also indicated. This sort of detailed description permits a quick assessment of the seriousness of a violation by a chip designer and greatly enhances design correction effectiveness.

The functional failure mode verification is performed by a large network analyzer and power distribution analysis program. It takes a resistive model of the entire chip power distribution, combines it with the current-sink representation of individual macros, and then analyzes the configuration for the resistive voltage drops.

The resistive network is calculated partly by the power distribution analysis program and partly by a separate, stand-alone routine called DCAN [15]. The two programs are used to reduce computation cost. The first- and second-level power distribution is distributed via a wiring system similar to that of net wiring. Therefore, a net resistance can be calculated by adding up segment resistances. Each segment resistance is calculated by the equation

$$R = \rho L/W,$$

where R is the line resistance, ρ is the metal square resistivity, L is the line length, and W is the line width. The third-level power distribution is considerably more difficult to analyze, since it contains irregular shapes. The DCAN program is used to generate a resistive equivalent matrix for the third-level metal. A modified output file is then fed to the power analysis program.

A current demand of an individual circuit is calculated by considering the circuit intrinsic current demand and the output current demand that eventually has to be channeled through the distribution system. It has been assumed that circuits are typically 40% in the on and 60% in the off state, and that the placement does not affect this ratio. The actual output loading determines additional current flow through a ground power service terminal.

Particular attention was given to current assignment associated with the driver macro. Two current values are assigned, one corresponding to the worst-case dc loading and the other to the peak current experienced during transient switching.

A *current density failure* mode is analyzed by two programs: The first calculates the current density for first- and second-level metal, the currents through 1–2 and 2–3 vias, and the current through the power C-4 pads. The second program is the DCAN program, which is used to calculate the current density of third-level metal. Again, DCAN is used because the main program is incapable of processing the graphic language with 45° shapes. Histograms produced by the programs are then manually examined for current-density violation. The data are also used in reliability calculations.

Performance verification

Because of the time and effort required to *accurately* compute internal and between-chip delays, it is not practical to analyze the delay of every circuit on a VLSI chip. The procedure is to first selectively define delay chains for analysis, and then to accurately analyze these chains from a worst-case or statistical point of view. The selection of critical paths was accomplished in two ways:

- Manually, by the logic designer's knowledge of his critical logic path.
- Automatically, via software tools [4] which place delays on logic functions and either simulate functionally across chips and compare to predetermined waveshapes or add blocks of delay and compare to a predetermined expected delay.

The statistical analysis tool [16] is a circuit-modeling program which uses device and wiring parameter distributions.

The key to automatic delay computation lies in the definition of simplified delay equations and the methods used to make assumptions required in order to choose the more accurate equation or constant. The object is to make the results slightly pessimistic and re-examine whatever fails on an off-line tool [16], as described previously. The *delay equations* are derived from linear regression approximations made from the results of the more accurate statistical analysis based on mask and process variations [1]. The basic form of the internal chip equations is

$$\textit{circuit output delay} = (K_0 + K_1 C_T)M,$$

where

K_0 = intrinsic delay of the circuit,
K_1 = constant multiplier for capacitance,
C_T = total pin and wire capacitance, and
M = multiplier for worst case.

• *Delay methodology*

The automatic tools are the same as those used in the IBM engineering design system [4]. Figure 5 illustrates the data flow. However, the procedure for computing net capacitance was tailored to our custom wiring approach. We utilized an existing USC tool [12] to assemble the graphic shapes into nets during logical-to-physical checking, compute the area, and feed a capacitance value back to the input logic for subsequent use by the delay tools. This was different from the standard masterslice approach since our custom wiring did not have wire-type rules defining electrical equivalents. This also added the flexibility to apply different capacitance coefficients to the metal area depending on what shapes were over or under it. Once the delays were automatically generated and placed into the logical expansion, the results were simulated with respect to expected function and timing.

Results and conclusions

During the design period, the only significant physical design error to escape detection was in the "ROS" of the first test chip. This problem was caused by an error in the physical model and was subsequently discovered when the low-level physical model was compared to the high-level logical model. The physical model matched the graphic data but one decoder was out of phase with the logical model.

The first-pass product chips were fully functional and were used in system hardware models [3]. Final validation of each chip was completed in 24 to 48 hours after the last change.

The computer costs to perform the software validation were initially considerable and much effort was expended in minimizing run times. The following areas were addressed:

- *Hierarchical checking* and the previously mentioned method of device recognition reduced run times four-to-one for logical-to-physical checks. However, they complicated the validation with additional checking for the following reasons: (1) to ensure that errors did not exist between internal elements of one checking level and the global elements of the next level; and (2) to ensure that level-naming conventions were followed and that symbolic shapes used as blockages, etc. in the next level of checking were correctly placed. These two items were checked automatically but required considerable thought to include all possible cases.
- Graphic design techniques were used when designing the physical books, which eliminated time-consuming data manipulation during checking (*e.g.*, unioning).
- The order of checking and minimizing shape counts also had a significant impact on run time. The order of checking was designed to perform checks on large numbers of shapes first, then to eliminate these shapes prior to performing subsequent checks. This eliminated the need for the software to process large numbers of nonrequired shapes. It was also found effective to perform a common operation once and then to use the result for several different checks.

In addition to further reducing the computer costs, future objectives should also include defining the additional audit controls to permit a paper qualification on chips using the existing library of books.

Acknowledgments

Space does not permit comprehensive acknowledgment of all the individuals who have contributed to the success of this effort or similar efforts at IBM in custom physical design verification. The following individuals deserve special mention for their personal efforts, and serve as examples of others whose contributions have not been referenced herein. Among the circuit, graphic, and logic designers who substantially influenced this methodology are K. F. Mathews, L. P. Lee, H. Askin, G. Fleming, C. Peterson, T. Kelly, G. Tuma, and J. Lee. Within the rules development group were J. Parsley, C. Meade, and S. Bennett. Among the local tool groups were R. Proctor, R. Golden, L. Goodwin, T. Ng, and A. Niekrewicz. Within IBM design automation development groups, as well as other groups doing custom design, were T. Lavery, K. Clark, D. Lambert, T. W. Kludt, F. V. Legge, and E. J. Nosowicz.

References and notes

1. K. F. Mathews and L. P. Lee, "Bipolar Chip Design for a VLSI Microprocessor," *IBM J. Res. Develop.* **26,** 464–474 (1982, this issue).
2. John E. Campbell and Joseph Tahmoush, "Design Considerations for a VLSI Microprocessor," *IBM J. Res. Develop.* **26,** 454–463 (1982, this issue).
3. Arnold S. Tran, Richard A. Forsberg, and Jack C. Lee, "A VLSI Design Verification Strategy," *IBM J. Res. Develop.* **26,** 475–484 (1982, this issue).
4. P. W. Case, M. Correia, W. Gianopulos, W. R. Heller, H. Ofek, T. C. Raymond, R. L. Simek, and C. B. Stieglitz, "Design Automation in IBM," *IBM J. Res. Develop.* **25,** 631–646 (1981).
5. *Fully functional*, as defined in this paper, means that the VLSI parts operated in a hardware model of the intended systems environment.
6. A *macro*, as defined in this paper, is an organized group of lower-level building blocks. Macros are additionally discussed elsewhere in this issue [1], and in Table 1.
7. E. B. Eichelberger and T. W. Williams, "A Logic Design Structure for LSI Testability," *Proceedings of the 14th Design Automation Conference,* New Orleans, LA, 1977, pp. 462–468.
8. P. W. Case, H. H. Graff, and M. Kloomok, "The Recording, Checking and Printing of Logic Diagrams," *Proceedings of the Eastern Joint Computer Conference,* Philadelphia, PA, 1958, pp. 108–118.
9. J. A. Ludwig, "A 50K Bit Schottky Cell Bipolar Read-Only Memory," *IEEE J. Solid-State Circuits* **SC-15,** 816–820 (1980).

10. R. L. Golden, P. A. Latus, and P. Lowy, "Design Automation and the Programmable Logic Array Macro," *IBM J. Res. Develop.* **24,** 23–31 (1980).
11. David R. Lambert, "Graphics Language/One, IBM Corporate-Wide Physical Design Data Format," *Proceedings of the 18th Design Automation Conference,* Nashville, TN, 1981, pp. 713–719. See also *GL/1 Language Specification,* Order No. GE45-1127; available through IBM branch offices.
12. C. McCaw, "Unified Shapes Checker—A Checking Tool for LSI," *Proceedings of the 16th Annual Design Automation Conference,* San Diego, CA, 1979, pp. 81–87.
13. 100× model of chip power distribution was built to measure magnetic interaction between power buses and signal nets. Assessment of ac noise contribution was based on this model.
14. C-4 stands for "controlled collapse chip connection" and refers to the chip-to-module contact connection.
15. C. M. Sakkas, "Potential Distribution and Multi-Terminal DC Resistance Computations for LSI Technology," *IBM J. Res. Develop.* **23,** 640–651 (1979).
16. *Advanced Statistical Analysis Program (ASTAP),* Program Reference Manual, Order No. SH-20-1118-0; available through IBM branch offices.

Received November 6, 1981; revised February 17, 1982

The authors are located at the IBM System Products Division laboratory, Neighborhood Road, Kingston, New York 12401.