R. N. Gustafson
F. J. Sparacio

IBM 3081 Processor Unit: Design Considerations and Design Process

Significantly new challenges were presented for the design of the 3081 Processor Unit since it was the first IBM large system implemented in LSI technology. Solutions had to be found for a new set of problems in order to achieve the required product objectives while maintaining an acceptable development cost and schedule. In this paper, the design aspects and the characteristics of the 3081 Processor Unit are described and the tradeoffs that were made due to the implementation of LSI are presented. A design strategy was chosen that included tradeoffs covering the areas of machine organization, performance level, implementation costs, testing and servicing aids, and development schedules. An innovative verification effort was introduced into the design process, capitalizing on a hardware flowcharting discipline and rigorous design rules. On the basis of this development experience, some thoughts concerning VLSI implementations are explored.

Introduction

The 3081, a 303X-compatible product, represents the introduction of the use of LSI technology by IBM in the large-computer-systems area [1]. The 3081 Processor Unit is entirely implemented in LSI technology, with high-density packaging utilized on the chip, module, and board levels. The unit consists of a number of components for CPU, storage, and I/O control and has a dyadic structure containing two central processors.

This paper discusses the 3081 Processor Unit design and its development methodology, focusing on those areas which were affected or influenced by the use of LSI technology. The packaging characteristics and design considerations are described, indicating the objectives and techniques that were used to take maximum advantage of the characteristics of the LSI technology at the chip, module, and board levels. The organization and structure of the processor unit reflect a design in which the performance objectives were achieved by a combination of the dyadic configuration and the aggressive 26-ns cycle time. To support the use of the much less changeable LSI technology, a design verification plan ensuring the detection of a very high percentage of the latent design errors, prior to the actual hardware testing phase, was put into place. The intersection and impact of this verification strategy with the overall development cycle are discussed. The experiences gained as well as the new development tools that were established during the 3081 program have convinced us that the foundation exists for further exploitation of LSI and VLSI as well.

- *LSI characteristics*

The major attributes of LSI which affected the design philosophy for the 3081 Processor Unit were the following:

1. *Long turn-around time:* Because of the much higher level of integration, the turn-around time for changes to the hardware for the correction of design errors or for design modifications was expected to be in the order of weeks, whereas non-LSI designs typically experienced turn-around times measured in hours. Therefore, significantly more design errors had to be removed prior to the building of hardware, and an improved ability was needed to change, or to work around, discovered problems.
2. *Power dissipation:* Dense packaging could be obtained at the chip, module, and board levels; but power distribution, signal coupling, and cooling considerations placed restrictions on the type of circuits that could be used. Because of their speed/power characteristics, T^2L circuits were chosen for these reasons. The number of array chips per *thermal conduction module* (TCM) [2],

their circuit density, and their performance levels were also limited by these considerations. Thus, system performance had to be achieved by a unique balance of hardware and microcode design concepts.

3. *Debugging capability:* The lack of traditional scoping capability could severely impact the ability to trouble-shoot and to isolate problems during the hardware testing phase. Our experience has shown that this limitation can largely be circumvented by the scan-in, scan-out capability that is inherently available in Level-Sensitive Scan Design (LSSD) [3].

• *Design strategy*

The design approach taken in the structure of the 3081 Processor Unit to minimize the limitations of LSI while still obtaining relatively high performance, within the limits of the package and technology, was essentially the following:

1. *Simple organization:* Provision of a relatively simple straightforward design, with limited overlapping or pipelining of instructions. This approach allows maximum use of microcoded controls with minimum hardware assists. A writable control store provides a broader capability to make temporary fixes for hardware logic errors.
2. *Hardware partition:* Organization of the processor unit to take maximum advantage of the circuits and package in order to minimize the machine cycle time. To a much larger degree than previously, with LSI technology this means minimizing both chip and module crossings (interconnections) in logic paths, rather than designing to minimize the number of "logic" levels. Although logic levels still remain important, they are increasingly becoming the parameter that is traded off against "chip" and higher-level packaging "crossings."
3. *System configuration:* Achievement of the system performance objectives with an integrated dyadic configuration composed of two central processors, each having access to central storage and channels via a single system controller.

• *Design rules*

The techniques utilized for level-sensitive design have been used for many years in several large-system designs, largely in the CPU and memory bus areas. The discipline of a level-sensitive design for a common clocking system allows the design of the various parts of a complex machine to a tight cycle time relative to the technology and the integration of these various parts with minimal interface timing problems.

To the writers' knowledge, this technique was first utilized in the 7950 system [4] built in Poughkeepsie and shipped in January 1961. In this system, as indicated in Fig. 1, logic is fed to latches and triggers [5], the latter being the direct source of stable signals for the logic. Latches and triggers

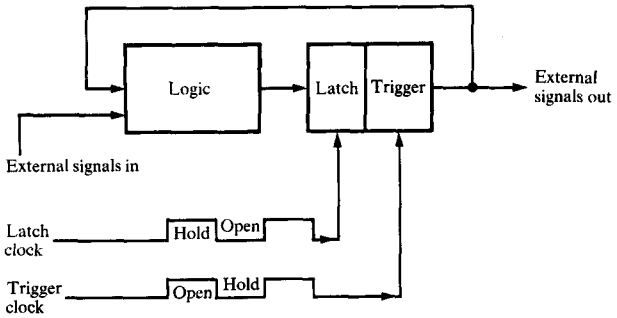

Figure 1 Level-sensitive design in the IBM 7950.

utilize separate clocking lines. The logic is level-sensitive to the degree that it adheres to the following rules:

1. The steady-state output of logic is not dependent on the rise time and fall time of individual circuits, or on the sequence of signals that are inputs to it.
2. The clock or cycle time is no longer than any individual logic path delay, including external inputs; *i.e.*, all outputs have achieved their final state prior to the latch clock transition to the "hold" state. (A minimum logic path delay is also imposed, however, to allow latch and trigger clock overlap that further reduces the machine cycle time.)
3. The latches and triggers cannot control their own clocks in the same cycle during which they are being set by these clocks.

Level-sensitive scan design, as described in [3], is a design technique which was put into place to permit a high level of logic testing during the manufacturing process. This technique was further extended to permit the complete scanning capability of all hardware facilities in the machine by the *processor controller*. Thus, the processor controller has the capability of examining any or all of the facilities within the machine as well as altering their contents. This is the method utilized for system reset and initialization, for the logging out of all facilities on the occurrence of hardware failures, and for the loading of hardware test patterns. This also provided a "scoping equivalent" capability that was successfully used during the hardware test phase. A double-latch version [6] of a *shift-register latch* (SRL) was developed to implement the latch and trigger for system logic and also to serve as one of a pair of latches in a scannable shift register configuration.

As used in the 3081 Processor Unit, the addition of a scan capability (in and out) and the rigorous clocking disciplines needed for LSI chip testing are extremely useful for a disciplined system design as well as for machine debugging and maintenance.

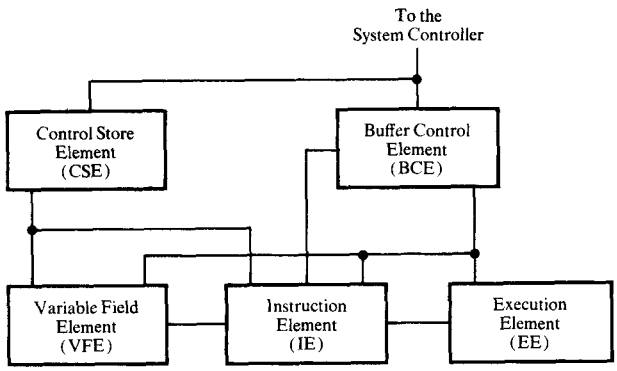

Figure 2 Functional elements of a central processor.

• *Packaging characteristics*

The packaging characteristics of the 3081 Processor Unit resulted from several iterations between machine designers and packaging designers, including the considerations of power distribution and cooling capabilities. The objectives of the machine designers were to package the major components of the machine within one board, and the major elements of those conponents within a single TCM, in order to minimize the number of interconnections between packaging levels and to permit the shortest machine cycle time.

In conjunction with the 704-logic-circuit chip, two array chips were designed having compatible signal levels with logic chips and having placement capabilities on the same TCM as logic chips. Array chips having a capacity of 256 × 9 bits with a 16-ns access time are primarily used for large data buffers (cache and control store). Array chips with 32 × 18 bits and a 7-ns access time are primarily used for working registers and for local storage.

A general problem in the implementation of the 3081 Processor Unit was caused by the nonuniform gain in LSI associated with logic-chip and array-chip densities. Compared with the IBM 3033 processor, at the chip level the average logic-chip density increased by a factor of 40:1 compared with an average array chip increase in the range of 2.5:1. Thus, a disproportionate amount of module area had to be allocated for array functions. A special TCM with a 20% increase in chip sites was designed to help compensate for this problem. Forty-eight of the 118 chip sites have restrictive engineering-change (EC) capabilities and thus are generally used only for array chips. A 32K-byte cache capacity per *central processor* (CP) and a paging mechanism for CP and channel microcode (very similar to CP cache paging) were a result of these array characteristics.

The major components of the 3081 Processor Unit are illustrated in [1, Fig. 2]. The design objective of having major components packaged within individual boards was achieved with one central processor being packaged on a nine-TCM board, and with the *system controller* (SC) and the *external data controller* (EXDC) each packaged within a six-TCM board.

Design considerations

As stated earlier, the design strategy for achieving the objectives of the 3081 Processor Unit were a short machine cycle time, a limited overlap structure with liberal use of microcode, and an integrated dual-processor configuration. Each of these items is now explored, discussing unique characteristics and tradeoffs as appropriate.

• *Machine cycle time*

Major emphasis was placed on achieving the shortest possible cycle time while maintaining the same amount of function or work per cycle as other large processors (without overlap functions). Analysis continued to support our belief that the proper balance between machine cycle time and function per cycle had been achieved (*i.e.*, arithmetic operations, cache access, control-store access). A cycle time of 26 ns was achieved with an average gate delay of 2.5 ns per logic level.

The design and implementation of the CP was the limiting factor in establishing the machine cycle time. The CP was structured into five elements based on functional partitioning, packaging, and the short cycle time objectives. These are discussed in greater detail in a subsequent section. It is cogent to note that functions involving array chips tended to contain the cycle-limiting paths.

• *Limited overlap*

Design errors are more prevalent in complex control functions as compared with more regular data-path functions; our data indicate this to be in the range of 5 to 1. On the basis of our lack of experience with the new verification methods and with the hardware testing technique, we opted to limit the control complexity of the machine and to provide more flexibility with the use of microcode controls. This direction was chosen to minimize the risk of a lengthy testing process. Our experience has been very positive in both the verification and testing efforts; this aspect is discussed in some detail in the paper by Monachino [7].

• *Microcode*

Two types of microcode were selected for the control of the CP and EXDC:

Horizontal microcode is characterized by a wide word which typically controls one machine cycle. The bits within a word directly control the data path (*i.e.*, the gate into a register, etc.). It is utilized when flexibility and speed of

execution are paramount. A rich data path is needed to take full advantage of the efficiencies of horizontal microcode. It is used in the CP and in the *data server element* (DSE) of the EXDC.

Vertical microcode is the type of microcode utilized in the *channel processing element* (CPE) of the EXDC. It is characterized by a less complex format and is essentially a set of basic (Load, Store, Branch, Shift, Add, etc.) assembly-language-type instructions that are tailored to the data path and functions of the CPE. The use of vertical microcode facilitates ease of writing and simulation. With the relatively simple data path of the CPE, the vertical microcode approaches the efficiency of horizontal microcode.

As previously mentioned, due to the density of the available array chips in conjunction with the size of the physical package, only a portion of the microcode could be contained within the CP and the EXDC. However, analysis indicated that, on typical job streams, a paging algorithm similar to that employed by general cache structures would be very effective. Less than 5% internal performance degradation results from a paging of CP microcode. Also, a minimal aggravation to the overrun characteristics of the EXDC has been observed.

- *Dyadic organization*

A major factor in the performance level of the 3081 Processor Unit is the symmetrical dyadic-processor configuration. LSI created the opportunity to package two identical CPs together with the SC and EXDC within one electronics frame. The configuration is symmetrical in that each CP has the same priority and operational characteristics in relation to the central storage and channels.

Organization and structure

- *Central processor*

The CP is composed of five functional elements. Figure 2 shows the functional elements of the central processor and Fig. 3 illustrates how they are packaged within the nine-TCM board. Three separate execution elements (IE, VFE, EE) were designed, rather than one common element, due to the size constraint of the TCM and to achieve the shortest possible cycle time. As is evident, only the EE and VFE are totally contained within a single TCM. (This partitioning will be highlighted as the elements are subsequently described.)

The *instruction element* (IE) controls the instruction sequencing of the CP. It initiates requests for instructions and attempts to maintain a buffer of four doublewords of instructions locally. It performs the instruction-decode and the operand-address-generation functions and initiates all requests for operands. It also executes all arithmetic and logical operations, exclusive of those done in the VFE and the EE described subsequently.

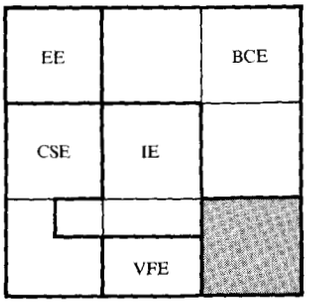

Figure 3 Central processor board layout. (Note: shaded TCM location is unused.)

Other than instruction fetch look ahead, which is performed in parallel with instruction execution, the IE is essentially sequential and does not attempt to overlap the execution of the $(N + 1)$st instruction with the Nth. Instruction execution is controlled by horizontal microcode.

The IE could not be packaged within a single TCM, and it was the most difficult element within the CP to partition across multiple TCMs, within the I/O limitations of the TCM and without impacting the cycle time. The instruction buffers are located on the TCM with the VFE, where they share a port from the cache of the *buffer control element*.

The *variable-field element* (VFE) executes all variable-field-length storage-to-storage instructions. In this class of instructions, the IE and the VFE operate as a team, with the IE performing operand fetches and stores in parallel with the VFE execution wherever possible. The VFE is largely controlled by horizontal microcode.

The *execution element* (EE) executes fixed-point multiply and divide instructions, convert instructions, and all floating-point instructions. The controls for the EE were committed to hardware due to potential partitioning problems with both the EE and *control store element* designs. This was an acceptable risk since the EE functions were well understood and were not as affected by architecture changes.

The *buffer control element* (BCE) contains the 32K-byte cache and has a store-in-cache algorithm, a 128-byte line size, and a four-way set-associative organization with a least-recently-used replacement algorithm [8]. The cache has a two-cycle access with the virtual-to-real address translation accomplished in parallel. The cache itself is packaged on two identical 118-chip-position TCMs, each of which

contains 72 array chips. A vertical partition (one 32-bit word on each TCM) was adopted to achieve the machine-cycle-time objective.

The store-in-cache algorithm was selected because it produced a higher internal performance level than a store-through algorithm, when the characteristics of the system were considered in conjunction with the central storage access time. Modeling results indicated that a store-in-cache algorithm yields superior performance as soon as the central storage access time exceeds the range of ten machine cycles. The 128-byte line size was selected since it delivered the optimal performance when analyzed with the central storage, system controller, and dual-processor characteristics. Store-in-cache also provided a foundation for a checkpoint-retry algorithm for the CP, which will be described subsequently.

The *control store element* (CSE) controls the sequencing of the horizontal microcode of the CP. The CP microword is 108 bits wide and has an addressing capability of 32K microwords. Two microword formats are supported, one for each of the IE and the VFE. A 1K-microword writable static array contains the most-frequently-used microwords. A second 1K-microword array contains 32 lines of 32 microwords each. It is fully associative with a least-recently-used replacement algorithm, and lines are dynamically loaded on demand from a "hardware system area" located in a portion of central storage. Like the cache, the control store arrays are packaged on two unique TCMs with a vertical partitioning; however, a significant number of controls had to be replicated to achieve the cycle-time objectives.

A unique feature of the CP is its hardware checkpoint-retry mechanism. The basic concept is to establish a checkpoint at some instruction N. The contents of the architected registers (PSW, GRs, etc.) are saved in backup registers. As instruction execution proceeds, any store into the cache will cause the old value of a changed doubleword to be saved in a push-down array. In the event of an error condition, the architected registers will be restored to their original value. The contents of the cache will also be returned to its original values by storing back the contents of the push-down array in the reverse order. Regular checkpoints are established during normal instruction execution under specific machine conditions with a negligible effect on CP performance.

• *System controller (SC)*

The major function of the SC is to provide the paths and controls for the communications between the major functional units and with *central storage*. A unique structure was adopted due to the packaging limitation (I/O pins), the store-in-cache algorithm of the CPs, and the support of the dual-processor configuration with minimum performance degradation. The I/O needs of central storage, the SC, and the EXDC provided the requirement for the six-TCM board with its additional I/O capability.

The implementation of the array portion of central storage is on a card-on-board package to achieve the density and cost requirements. Each board contains 4M bytes of central storage, which constitutes a *basic storage module* (BSM). The BSM is configured so that a 128-byte line can be accessed (read or write operation) with a single array operation.

The conventional doubleword interleaving of current large processors is effectively accomplished within the 4M-byte BSM. Of significant importance is that the bandwidth requirement for independent doubleword operations with central storage has been significantly reduced. This is the result of the store-in-cache algorithm of the CPs in conjunction with a 128-byte line, and the 256-byte (per channel) buffering capability for data within the EXDC. Thus, the SC was structured for a line-transfer characteristic. The basic data-bus width of all units connecting to the SC is eight bytes (bidirectional) with a data transfer rate of eight bytes per machine cycle.

To achieve a balanced utilization, a 2K-byte address interleave structure across the 4M-byte BSMs was adopted. This increment was chosen to minimize the complexity of memory reconfiguration and was sufficient to achieve a balanced load.

The store-in-cache algorithm has created a data integrity problem which is managed by the SC. Due to changed data residing in the caches of the CPs, the content of central storage is not always valid. The SC maintains a duplicate copy of each CP cache directory and determines if the central storage request will return valid data. In the event of a conflict, the SC initiates the casting out of the specific line of data from the cache to central storage to then be refetched by the requesting unit.

Each line of data, as it resides in a CP cache, is tagged with a *read-only* (RO) or *exclusive* (EX) status. A line must have EX status before a store operation can be completed within that line. To control simultaneous updates to a specific line by the CP or channels, the SC permits only one copy of a line to exist with EX status. Conversely, multiple copies of a line having RO status can exist simultaneously for use as long as no store activity occurs. If a store must be performed into a line with RO status, a change is effected by the SC to EX status, with the purging of the copy if necessary.

An algorithm predicting the usage (fetch or store) of a line of data was adopted to minimize the performance

overhead of status changes and purging. On the basis of the operation being performed by the CP that required the new line to be accessed, an RO or EX status is assigned. For example, a line that is required for instructions has RO status since reentrant code prevails in today's programs. A line that is required for operands has EX status if a store operation is being performed or if the other CP has a copy that has been changed (copy will be cast out in the process).

The SC also performs an ECC (error checking and correction)-to-parity conversion on data to and from central storage, contains the storage protect keys and time-of-day clock, and manages an eight-position queue containing storage requests.

- *External data controller (EXDC)*

The EXDC performs the channel functions for the 3081 Processor Unit. It provides up to 24 channels with a data rate of up to three megabytes per channel. Four of these 24 channels can be byte-multiplexer channels with aggregate data rates of up to 500 kilobytes per second with a burst size of 32 bytes.

As previously mentioned, the channel area was a prime candidate for the use of microcode controls. Channel designs are difficult to completely verify and test because of the wide variations associated with I/O devices and channel programs. Historic data also bear out the late discovery of problems, as different I/O loads and configurations are encountered.

The EXDC consists of two types of microcode-controlled elements. The *channel processor element* (CPE) is a special processor which handles the control of I/O instructions and interrupts. It supports a queued interface with the central processor for START I/O FAST RELEASE operations and for I/O interrupts, and controls the handling of command/data chaining and all EXDC recovery operations. The CPE is driven by vertical microcode having a two-byte microword and an addressing capability of 32K microwords. A 2K-microword writable static array contains the most frequently used microwords. A second 256-microword array contains 4 blocks of 64 microwords each. The array is dynamically loaded on demand from a "hardware system area" located in a portion of central storage. The processor has a two-byte data-path structure and is packaged on one TCM.

The second element is the *data server element* (DSE), which handles the control sequencing and data buffering for eight channels. It is packaged on a single TCM, and a maximum of three DSEs can be attached to the CPE. The DSE contains 256 bytes of data buffering per channel. It supports two-byte transfers with the interface adapters and 64-byte data transfers with central storage. The DSE is controlled by horizontal microcode having a 54-bit microword in a 750-microword array. The eight channels are controlled by DSE microcode that is time-shared on an equal round-robin basis.

Outboard of the DSE, packaged in a card-on-board technology, each channel has an interface adapter element which drives the I/O interface and contains eight bytes of data buffering.

- *System clocking*

A description of the Processor Unit and the effect of LSI would not be complete without a discussion of the clocking system. After proceeding with a design strategy using a "tunable" clock distribution system, the direction was changed to a very simple "untuned" system, for reasons which shall now be described.

The initial direction was a sophisticated distribution network with a tuning capability achieved by the use of programmable delay chips. It had the quality of providing some clock change capability and was judged to have the least clock skew delay delta to the machine cycle time. Sixty chips per board, a reference generator with an associated comparison network, and several thousand lines of processor-controller microcode were required to support the system. The final untuned system required just six chips per board and completely eliminated the need for a tuning process during manufacturing build and test, as well as in the field when TCMs are replaced.

Delay lines are used to compensate for different physical distances between the oscillator and the final circuits that use the various clock pulses. These delay lines are identical for every machine. The delay variations in the distribution network caused by the variations of the hardware from machine to machine are *not* compensated for by the use of unique delay values for each machine. That variation has been statistically combined with the variation of the logic path delays as part of the overall calculations of the machine cycle time. The distribution networks are also included in the timing-analysis verification to be described subsequently. It is of interest to note that less than 500 picoseconds were added to the machine cycle time over the "tuned" system. The untuned system has proven to be an excellent design, and in the writers' judgment the cycle time penalty is a small price to pay to avoid the problems which would be encountered in manufacturing and field tuning.

- *Testing aids*

The long turn-around time of logic changes with LSI and the lack of scoping abilities initiated the inclusion within the logic of techniques that would allow temporary fixes, and

that would also provide some history of key trigger status for a limited number of cycles to provide the equivalent of scoping.

Unit logic chips were designed and located within each area of the system as "spares." These are chips that had the individual circuits mounted to I/O pins with generally one or two levels of logic. If the logic mistake was not too deeply imbedded within the logic of a chip, a temporary fix would be made with unit logic by discrete-wire rework on the TCM. The machine cycle time might be impacted, but it could be adjusted for the time period during which the temporary fix was installed. The percentage of time this proved useful was greater than expected, with approximately 25% of hardware design errors being temporarily fixed.

History arrays were interspersed among the logic chips in the machine, into which were stored, on a cycle-to-cycle basis, the states of key triggers for the past N cycles. (N was generally 32.) Also kept in the history arrays was microcode-word information for the past 32 cycles. During debugging, the state of the system (all SRLs and arrays) would be scanned out for the cycle in which the machine was stopped. The history arrays were also scanned out to indicate the state of key triggers and microcode words for the past N cycles. This technique proved valuable for all errors, but especially so for intermittent errors occurring infrequently.

Design verification

As previously stated, the primary concern in designing a complex machine in LSI was the turn-around time for the correction of logic errors during the hardware testing phase. If techniques could not be utilized to limit the number of logic errors in the initial design, it could be projected, from past machine debug statistics, that serious logic errors with weeks of turn-around time would so extend the debug time as to make the LSI design infeasible.

In addition to a simplified machine organization using writable control store to minimize the probability and effects of errors, maximum emphasis was placed on utilizing design verification techniques. These techniques (to be subsequently discussed) included the following:

1. *Unit delay simulation.*
2. Detailed *hardware flowcharts* to represent hardware logic; this was the base for *cycle simulation,* which is a cycle-by-cycle simulation of the machine functions.
3. Use of hardware flowcharts for either *hybrid simulation* or *Boolean comparison.*
4. A *timing analysis* program to analyze the logic path delays between shift-register latches and to specify when the logic delays violate the machine cycle time specified.

• *Unit delay simulation*

This simulation technique [9] has been used extensively for many years for the development of many prior machines. It is essentially a program that allows the designer to stimulate (set to a combination of 1s and 0s) the inputs to an area of logic, and that provides the outputs from this area of logic. The program compares the designer's generated logic outputs to the logic outputs generated from the implemented logic.

• *Hardware flowcharts*

Figure 4 illustrates how hardware flowcharts are used to represent and specify logic. In Fig. 4(b) the latch-trigger combinations are of two types: a type (L_0T_0) whose output (G) either is held every clock cycle or is affected by input data through a control; or a type (L_1T_1) whose output (F) holds its present state until either "TURN ON" or "TURN OFF" occurs with "clock time." (If both "TURN ON" and "TURN OFF" occur simultaneously, the presence or absence of the internal dotted line will give one priority over the other.) As can be seen from Fig. 4(a), detailed flowcharts, together with the identification of the latch-trigger type, completely describe the next state of the machine in terms of the current state. (Storage functions and their off-over-on or on-over-off characteristics are listed separately from flowcharts.)

The method of detailed hardware flowcharts has been used as a design technique in certain large systems, or parts of large systems, since 1958 in IBM's Poughkeepsie Laboratory [4]. We believe that the design for complex logical sequences or states is most easily conceived using a pictorial representation, as shown in Fig. 4(a).

A new approach for the verification of the machine design was developed which separated the architectural verification from the logic (hardware implementation) verification. Verification of the Processor Unit architecture was accomplished through the use of a high-level software model built directly from the hardware flowcharts described previously. Since the model is capable of efficiently representing large portions of the machine, high-level functions—generally long instruction sequences (370 architecture)—are exercised. The verification of the implementation was then accomplished with a comparison for equivalence between the actual AND and OR logic blocks of the machine and the hardware flowcharts which were the basis for the high-level model.

The software model constructed directly from the hardware flowcharts can provide for early verification of the machine design. Initial efforts [10] utilized the FORTRAN language and required significant programming effort. The key characteristic of hardware flowcharts which make soft-

ware implementation difficult is the multi-way branch. The process was significantly enhanced with the development of a higher-level programming language that would allow for a more direct translation of hardware flowcharts [11].

This high-level software model, called *cycle simulation,* is now discussed, together with the logic-equivalence programs.

• *Cycle simulation*

The hardware flowchart model in conjunction with a complete list of hardware facilities (shift-register latches and arrays) completely describes the machine. The sequencing of the hardware flowchart model on a cycle-by-cycle basis and the resulting changes in the hardware facilities make up the *cycle simulator* [12].

As compared to a *unit delay simulator,* the *cycle simulator* is much more storage- and time-efficient. Thus, whole areas of the central processor (instruction unit, execution unit, etc.) can be run by simulating, on a cycle-by-cycle basis, the actual logic of the machine. This model is driven by high-level inputs. (The operation code, for example, could be the input.) Entire units (central processor, system controller, etc.) can be modeled for a specific function.

• *Logic equivalence*

A method of comparing the hardware flowchart logic to the actual hardware logic, as implemented in AND and OR logic blocks, was accomplished in two ways: Boolean comparison and hybrid simulation. These are now described.

Boolean comparison [13] This is a completely automated method comprised of a series of computer programs to compare primary outputs, which are generally shift-register latches (SRLs), in both flowcharts and hardware, for logical equivalence of all primary inputs (also generally SRLs).

Hybrid simulation Prior to the development of the Boolean comparison process, a method was developed to automatically accumulate the patterns of inputs and outputs over a multi-cycle time interval for a selected set of facilities from the flowchart simulator. Thus, the capability to automatically generate multiple low-level input and output signals from a global simulator using a few high-level inputs becomes available. These signals then become the input stimuli and the predicted output patterns for the unit delay simulator. The basic shortcoming associated with hybrid simulation is that, unless a very large number of flowchart simulator test cases are provided as input to the hybrid simulation, many branches of the logic will not be tested. For this reason, Boolean comparison became the predominantly used technique because of its comprehensiveness.

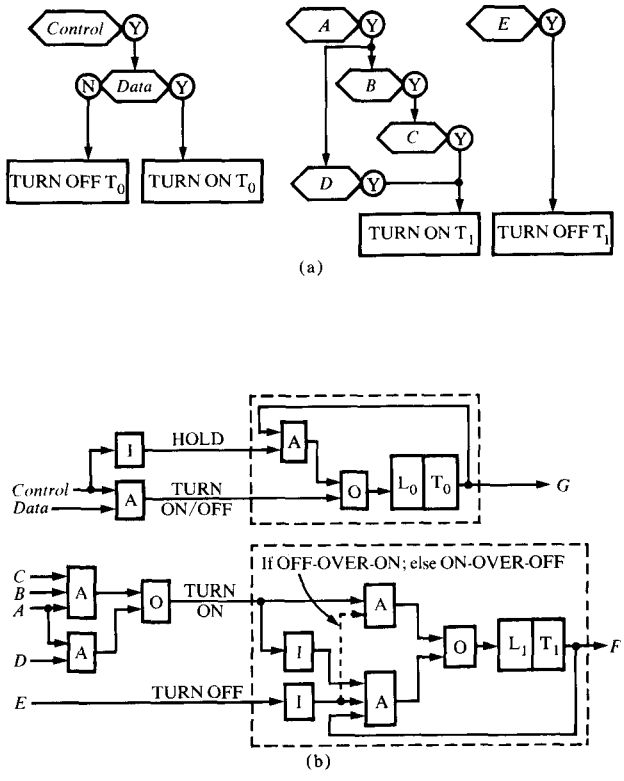

Figure 4 Illustration of the use of hardware flowcharts for the specification of logic: (a) hardware flowcharts, (b) logic implementation of the hardware flowcharts of (a). (Note: L = latch block; T = trigger block; L-clocks and T-clocks assumed.)

• *Timing analysis*

Verification of the machine cycle time objectives was achieved with the development of a *timing analysis* tool [14]. This tool, like Boolean comparison, is a series of computer programs which accurately calculate the individual path delays for all SRL-to-SRL paths within the machine. This is accomplished by evaluating all paths between interconnected SRLs characterizing all physical parameters. The proper delay characteristics are then applied, using the appropriate statistical methods. The net result is the identification of both long and short paths which violate the machine cycle time requirements. Timing analysis has proven to be essential to a design effort aimed at a high-performance machine by minimizing the machine cycle time and by obtaining the best potential times from the technology.

Historically, the cycle time to be obtained has been achieved by designing all logic paths to some rules of thumb (*i.e.,* ten logic levels and two card crossings; or eight logic levels, two card crossings, and one board crossing; etc.) that were fairly conservative and could be easily applied by all designers to all their logic paths. If some paths exceeded the rules, they would be timed with a more detailed description

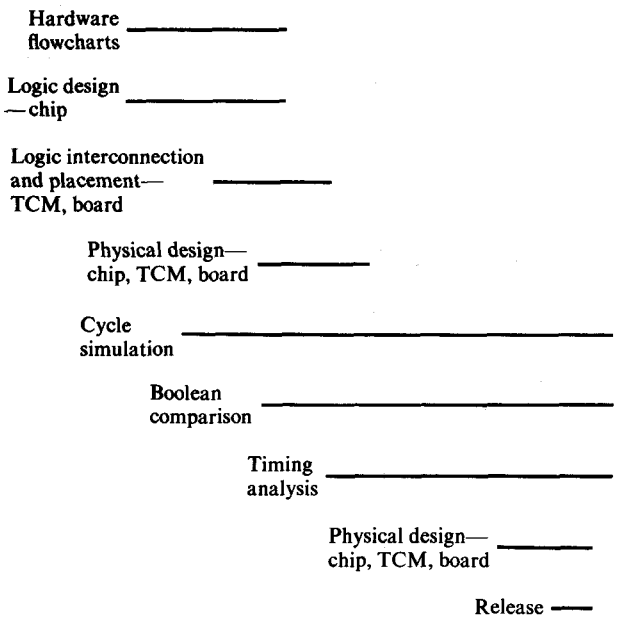

Figure 5 Design schedule.

of the physical characteristics of the path (number of loads, number of pins, precise wire lengths, etc.) using a computer program that would accurately calculate delays using very precise technology rules. The use of this method results in either most paths being significantly under the cycle time to be achieved, or the design rules-of-thumb being reasonably accurate compared to the precise calculated value.

Due to the complexity of the technology rules, a reasonably simple rule-of-thumb could not be obtained for the technology used in the 3081 Processor Unit. The machine cycle time achieved was practical only because of Timing Analysis. Efforts to locate long paths during the hardware test phase proved extremely time-consuming. If an LSI machine did not meet the cycle time in the large majority of paths when sent into hardware test, it would be impractical to locate and correct the many timing errors during test. This is a result of the limited scoping ability with LSI and the long turn-around time for changes.

Design cycle

The design cycle of the 3081 Processor Unit was markedly different from those of predecessor development programs. As previously discussed, a comprehensive verification plan was undertaken to reduce significantly the number of hardware design flaws prior to hardware being built. The following discussion explains how the verification effort was integrated into the design schedule.

Figure 5 illustrates the design schedule. Hardware flowcharting and the design of individual chip types dominate the front end of the schedule. Chips containing data-path functions are the first to be designed. It is not uncommon for several design iterations using different partitions to take place before arriving at an optimal design. Hardware flowcharts are the detailed specifications for the control functions, and these precede the implementation of the associated logic chips. Flowcharts and chip designs are entered into the data bases of the *engineering design system* (EDS) [15], triggering the use of a sophisticated design automation process.

Replication of multi-usage chips, with interconnections and placement of all chips on the higher-level packages, completes the initial implementation phase. A first pass was made through chip, module, and board physical designs to ensure a satisfactory physical design and to provide the actual parameters for Timing Analysis verification. For the start of Timing Analysis, we chose to wait for the completion of initial physical design due to the physical variations that were introduced during that process.

Cycle simulation verification starts as soon as sufficient flowcharts are complete to represent a reasonable functional element. Individual functional elements are simulated on a stand-alone basis, using low-level test cases, prior to being combined for a complete component configuration (central processor, system controller, etc.).

Boolean comparison begins with the completion of the interconnection of the detailed logic. Equivalent partitions are chosen between the hardware flowcharts and the detailed logic equating the primary inputs and outputs. This forms the body of logic to be verified using the Boolean comparison process.

After a sufficient verification level has been achieved with resulting hardware changes applied, a final physical design is completed and then released to manufacturing for the building of hardware. Additional verification work will most likely continue during the procurement of the first hardware, with additional changes to be applied as soon as that hardware is received. The objective is to complete enough of the verification work prior to the initial building of the machine to ensure that the bugs in the main-line logic have been removed and that the remaining defects are within a manageable limit.

Potential problems and strategies in future VLSI technologies

The 3081 Processor Unit had approximately 400 chip types that averaged around 500 circuits per chip. The problems that were encountered and solved in the design and debugging of the 3081 machine using LSI are multiplied, largely in degree, with VLSI. Thus the scoping ability obtained with

chips that average 5000 circuits is substantially less, the possibility of utilizing "unit logic" for temporary fixes of hardware is substantially less, etc. For these reasons it is believed that the techniques developed in the 3081 machine to deal with these "debugging problems" will be extended and improved. The following developments are projected:

1. Faster, more efficient simulation tools, capable of efficiently handling larger sections of the system as an entity, will evolve. This will permit additional design errors to be fixed prior to the hardware testing phase.
2. The ability to manufacture, test, and install a new chip in days rather than weeks will alleviate many of the projected "delays" during the debugging time.
3. The concept of level-sensitive design and the ability to scan in and out of shift-register latches will be continued.
4. The use of "history arrays" will probably be refined and expanded.
5. The design approach which maximizes the use of control store and still maintains high performance, through cycle time and multiprocessing, will probably be followed unless the evolvement of the preceding points 1–4 overcomes, in large degree, the problems of engineering debugging and field changes.

An initially obvious method of bypassing many of the problems of VLSI would be to build a machine in changeable technology, to debug this machine to wring out most hardware errors, and then to remap the machine into LSI or VLSI. This approach is feasible but has the disadvantages of additional resources and time to design, build, debug, and remap the model. Also, the remapped VLSI model will probably be unable to take full advantage of the unique packaging and circuit characteristics of the technology to achieve the best potential performance. How much is lost is subjective, but we believe that the development time and performance gained through the design techniques associated with the 3081 make that approach to VLSI design superior. It appears from our perspective that there are no inherent problems that make VLSI impractical from a machine-development viewpoint.

Acknowledgments

The comprehensive acknowledgment of the engineers responsible for a design effort of this size is difficult. We wish to recognize a few who made significant contributions in establishing the design direction and in defining the functions and structure of the components of the 3081 Processor Unit; for the Central Processor: R. J. Bullions, C. W. Evans, J. A. Gerardi, S. R. Levine, and B. L. McGilvray; for the External Data Controller: P. N. Crockett, M. J. Halma, R. P. Jewett, and A. J. Scriver; and for the System Controller: E. J. Annunziata and B. U. Messina. The key management direction for the design was provided by R. A. Houdtzagers, R. S. James, L. H. Johnson, and R. B. Stuart. We would also like to acknowledge the significant management efforts of A. E. Fitch and M. M. Monachino for the development of the verification tools that were so critical to the program.

References and notes

1. M. S. Pittler, D. M. Powers, and D. L. Schnabel, "System Development and Technology Aspects of the IBM 3081 Processor Complex," *IBM J. Res. Develop.* **26,** 2–11 (1982, this issue).
2. A. J. Blodgett and D. R. Barbour, "Thermal Conduction Module: A High-Performance Multilayer Ceramic Package," *IBM J. Res. Develop.* **26,** 30–36 (1982, this issue).
3. E. B. Eichelberger and T. W. Williams, "A Logic Design Structure for LSI Testability," *Proceedings of the 14th Design Automation Conference,* New Orleans, LA, June 1977, pp. 462–468.
4. F. J. Sparacio, "IBM 7950 Indexing and Control Techniques," *IBM Internal Report TR 00.818,* Poughkeepsie, NY, October 17, 1961.
5. M. E. Homan, "Data Latching Systems," U.S. Patent 3,075,091 (filed February 3, 1960; issued January 22, 1963).
6. E. B. Eichelberger, R. N. Gustafson, and C. Kurtz, "Logic Circuit for Scan-In/Scan-Out," U.S. Patent 3,806,891, 1974.
7. Michael Monachino, "Design Verification System for Large-Scale LSI Designs," *IBM J Res. Develop.* **26,** 89–99 (1982, this issue).
8. Alan Jay Smith, "A Comparative Study of Set Associative Memory Mapping Algorithms and Their Use for Cache and Main Memory," *IEEE Trans. Software Engineering* **SE-4,** 121–130 (1978).
9. A. C. Auch, D. D. Chang, and J. J. Hughes, "Tales—Logic Simulation and Evaluation System," *IBM Internal Report TR 00.1864,* Poughkeepsie, NY, April 1, 1969.
10. The initial concepts of utilizing hardware flowcharts as a base for cycle-by-cycle simulation of the hardware were conceived in 1968 by R. N. Gustafson and D. W. Anderson.
11. C. W. Evans provided the initial ideas for the language syntax to allow hardware flowcharts to be described in a program language (BDL/CS—Basic Design Language for Cycle Simulation) that can be compiled to a list of inputs which are used directly by a cycle-simulation program.
12. G. J. Parasch and R. L. Price, "Development and Application of a Design Oriented Cycle Simulator," *Proceedings of the 13th Design Automation Conference,* San Francisco, CA, June 1976, pp. 48–53.
13. Gordon L. Smith, Ralph J. Bahnsen, and Harry Halliwell, "Boolean Comparison of Hardware and Flowcharts," *IBM J. Res. Develop.* **26,** 106–116 (1982, this issue).
14. 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).
15. 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).

Received August 7, 1980; revised July 7, 1981

The authors are located at the IBM Data Systems Division laboratory, Poughkeepsie, New York 12602.