John E. Campbell
Joseph Tahmoush

Design Considerations for a VLSI Microprocessor

The machine architecture and design considerations for an interrupt-driven bipolar VLSI Microprocessor are presented. The processor is designed to a complex architecture and includes an integrated channel and a flexible storage interface. Floating-point functions are optional. A 3-ns custom bipolar technology was developed for the microprocessor, resulting in a very high circuit density package. The 50-mm four-chip air-cooled microprocessor module is packaged on a printed-circuit card with associated repowering circuits and high-speed random-access memory. Important design considerations and tradeoffs associated with the development of this machine, within specific cost, performance, reliability, and schedule objectives, are discussed. Various processor design techniques are described which minimize hardware where performance is not critical. A degree of functional parallelism is utilized, as well as timing flexibility, to attain the required performance.

Introduction

The *VLSI microprocessor* that is the subject of this paper consists of four bipolar VLSI chips contained in a 50 × 50-mm 361-pin module. A very high level of circuit density is obtained in the microprocessor module, with approximately 15 000 equivalent logic circuits and a 50K-bit (K = 1024) *read-only storage* (ROS) contained in it.

Various aspects of the design and development of the VLSI microprocessor are discussed in this paper and in three companion papers appearing in this issue. This paper focuses on the logical design considerations and the *processor system* architecture. Chip design aspects are the subject of the paper by Mathews and Lee [1], which discusses the technology, the chip structure, and the pseudo-custom macro chip design methodology used. The logical design verification strategy used in the development of the processor system is described in the paper by Tran, Forsberg, and Lee [2]; and the physical and electrical design verification procedures are covered in the paper by McCabe and Muszynski [3].

The VLSI microprocessor module is a key part of a processor-component functional building block subsystem, the elements of which are indicated in Fig. 1. These consist of a *processor unit* (PU), a *storage addressing subsystem* (SASS), a high-speed channel, a *storage control unit* (SCU), a *floating-point execution unit* (FP), and a multiple processor interfacing capability.

The processor is designed to a 32-bit architecture; *i.e.*, the machine can operate on byte (8-bit), halfword (16-bit), and fullword (32-bit) operands, and a storage addressing capability of up to 32 bits is provided. The implementation utilizes, in some cases, data flow that is less than 32 bits. This was done to minimize cost and is logically transparent to the system.

The storage addressing subsystem provides dynamic address translation, relocation, and protection. Storage is allocated on a block basis, each block containing 2K bytes. Translation tables are physically located in the same high-speed *random-access memory* (RAM) that contains general registers and work space. These spaces are kept logically separate by the hardware. The channel and the PU share the SASS, and the main storage interface is asynchronous.

The high-speed channel provides an I/O port to pass data between main storage and external devices. Main storage data alignment is handled by the channel, and the I/O port is asynchronous. The channel utilizes the logic of the processor unit to minimize cost; therefore, when the channel is active, the processor execution is suspended, since the data flow is shared.

The SCU controls the main storage array cards [up to 8M bytes (M = 1 048 576) are supported in this implementa-

tion]. *Error checking and correction* (ECC) is used to correct single-bit errors and most double-bit errors. The SCU contains a four-byte data buffer as well as a four-byte instruction buffer. The processor-to-SCU data bus is a two-byte bidirectional bus with four-byte multiplexing capability. The SCU-to-storage interface consists of two unidirectional four-byte buses.

The floating-point unit executes a set of floating-point instructions and is optional. It has been implemented, with a minor change, to an existing module.

The multiple processor interface provides a mechanism to logically share main storage and translation tables among two or more processor systems. It also provides an interface to a shoulder-tap device which provides communication between the processor systems.

The VLSI microprocessor offers *reliability, availability, and serviceability* (RAS) improvements as well as certain functional enhancements which were not available on previous machines of this type. Compatibility of programs is generally maintained, although if new programs use these enhancements they will not run on previous machines. The I/O interface is physically and functionally compatible with previous machines.

The VLSI microprocessor is designed using the *level-sensitive scan design* (LSSD) [4] technique. Parity checking, parity prediction, and ECC were used for enhanced data integrity and RAS characteristics. It is highly ROS-controlled, allowing functional refinement and design changes at a single point that can be changed with minor effect on the manufacturing masks.

The processor system incorporates a high degree of checking. Parity is carried or predicted throughout the machine. As previously mentioned, ECC is used in main storage. Process checks are recorded at each level and by type in a special register in the *program status word* (PSW). Machine checks are also logged by type in an internal register. If a process check occurs, the instruction is suppressed and the PSW instruction counter points to the instruction in error. The SASS prevents individual programs from accessing storage that is not assigned to those programs. Sufficient information is available to isolate errors to the *field-replaceable unit* (FRU) level, which in this case is the card level.

The design of a complex processor using a custom VLSI approach is considerably different from a standard masterslice (gate-array) technology [5] design. With custom VLSI [1], circuits are positioned on a chip as needed. The circuits are implemented in various configurations, and a certain amount of flexibility in circuit design is allowed. The masterslice approach supplies a fixed array of circuits that can be connected while observing specific predefined rules. Such items as circuit type, circuit placement, chip I/O connections, power, and size are predetermined when using masterslice but must be considered when using custom VLSI. The design requirements must be established and understood completely before the actual design is begun.

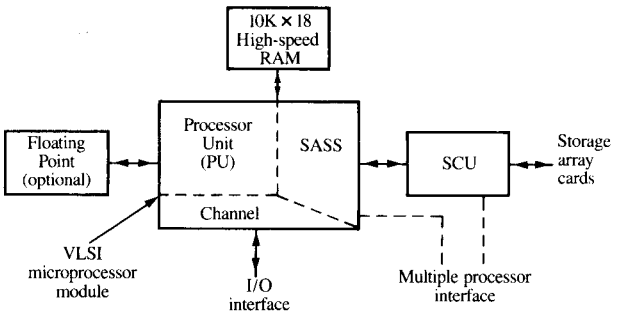

Figure 1 VLSI microprocessor and processor-component subsystem.

The partitioning of the processor and the organization of the logic must be determined prior to the actual logic design. To accomplish this, a thorough understanding of the technology and of the system environment is necessary so that the required design tradeoffs between processor performance, logic density, power, and packaging considerations can be made. The technology aspects of the VLSI microprocessor are discussed in more detail in [1]. A processor card was designed consisting of four custom bipolar logic chips (the VLSI microprocessor module), together with a high-speed RAM (10K × 18 bits) and an oscillator. The organization of the VLSI microprocessor module and its interface with the RAM is shown in Fig. 2.

The 4-wide by 3-high (14.4-cm by 11.7-cm) processor card has a bottom connector with 144 contacts and a top connector with 92 contacts. The four custom bipolar logic chips are packaged in a 50-mm *multilevel ceramic* (MLC) module [6]. Each chip contains the equivalent of 3000 to 5000 circuits. In addition, one of the logic chips contains an embedded 1024 × 50-bit ROS. The design is based on a hierarchical approach using a set of functional macros as building blocks for more complex logical functions [1, 3].

Experience in this project has shown that it is possible to develop a processor using custom VLSI resulting in a first-pass design (the first set of hardware received from the manufacturing site) that is functional.

In the following sections of this paper, the processor system design objectives, the functional partitioning, the

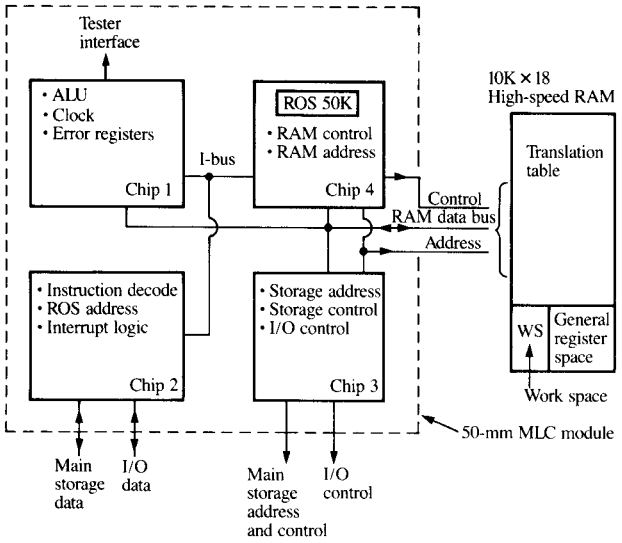

Figure 2 Organizational structure of 50-mm microprocessor module and 10K × 18-bit random-access memory.

ROS control mechanism, the instruction execution, the integrated channel, and the floating-point option are first addressed. Then the design of the interrupt mechanism, the processor clocking, and the *arithmetic and logical unit* (ALU) are described in some detail. This is followed by discussions of the storage protection mechanism and work space considerations, the multiple processor function, and the design and testing procedures.

Design objectives

The design philosophy of the processor system was based on meeting the following criteria:

- Packaging—The complete processor was to be packaged on one card, including a high-speed channel and 20K bytes of RAM.
- Cooling—By forced air.
- Performance—To be greater than that of predecessor machine.
- RAS—To exceed the specified standard for reliability and serviceability and provide a high degree of testability and internal checking, as well as recoverability.
- Compatibility—Logically compatible with previous implementations.
- Cost—Less than the previous implementations.
- Schedule—To produce functional VLSI on the first pass to facilitate system testing.

The VLSI microprocessor was packaged on a single module in order to meet the design requirement of a one-card processor building-block component. The remaining area on the card was used to accommodate the high-speed RAM (used for register space and for storage management tables) and an oscillator. Because of the large number of circuits per chip it was necessary to determine the manufacturing test philosophy before the design was begun. The LSSD technique was used at the chip and module levels. The testing at the card level was to be done in two steps. First, the modules would be dc tested using the *module in place* (MIP) test methodology [7]. After the dc test was successfully completed, a functional test capability would be provided which would consist of running a representative sample of machine code at full speed.

- *Functional partitioning*

The functional partitioning of the VLSI microprocessor was accomplished by first determining a technology macro set to be used in the design. As it turned out, the definition of the macro set was a continuous process, with some additions and deletions as the detailed logic design progressed. Because of the extent of the work involved in the design, verification, and layout of the macros, as well as the one-pass schedule criteria, a minimum macro set was established. Following is a list of the macros that were developed.

- N-way AIs
- N-way XORs
- off-chip receiver
- active pullup (tri-state) driver
- open collector driver
- LSSD latch
- 1024 × 50-bit ROS
- $I \times L \times O$-way PLAs (programmable logic arrays [8])
  $I \times L$ AND array with bit partitioning
  $L \times O$ OR array
  where I = number of PLA inputs, L = number of PLA product terms, and O = number of PLA outputs.

These macros were combined to offer design blocks containing up to nine macros per block. This allowed the designer to use, for example, N-bit register macros and N-bit selector macros. This provided an aid to logic designers as well as to chip layout, since data flow was indicated by macro usage. Because the design was being executed in custom logic, knowledge of the physical properties of the macros as well as the electrical characteristics of the set was required.

Once a basic set of macros was established, the actual partitioning of the machine could be started. Several other major factors that had to be considered during this time were chip power, driver switching activity, and the practical limit of chip size relative to wafer yield and, therefore, cost.

The initial pass for the partitioning indicated that the conventional partition would not result in a machine that could be contained within the chip count required or with an

acceptable I/O pin count at the card or chip level. This partition resulted in excessive logic and lack of wireability. An examination of the system environment in which the processor system was to be used indicated several areas that could reduce the amount of chip logic at some cost in performance and not affect the system throughput. These tradeoffs are examined in more detail subsequently.

Difficulty was experienced in containing the number of I/O connections at all levels of packaging. The card development group developed a card connector with I/O connections at the bottom of the card increased by 50%. This new connector was helpful but still constrained the card I/O design flexibility. Therefore, the storage address bus was multiplexed to relieve the card I/O requirement. A status code was used to inform the external attachments of the type of data on this bus, which is used to transfer information or main storage addresses depending on a status code. This approach saved a significant number of card I/O connections. Similar techniques were used to keep the number of chip and module I/O connections at an acceptable level.

With the initial partitioning completed, the detailed design was started. As the design progressed, the macro set and partitioning were modified to meet circuit and logic design refinements. The final partitioning resulted in four chips of equal size and approximately equivalent I/O count. The three-sigma worst-case power dissipation of each chip was between 3.0 and 3.5 watts.

Machine control

The VLSI microprocessor is controlled by a high-speed 1024 × 50-bit ROS which contains microcode routines for the decoding and execution of instructions as well as processor control subroutines (*e.g.*, interrupt, PSW swap, machine check, channel). See Fig. 2 for the general organization and inter-chip data flow interconnections.

The ROS word is divided into the following control fields:

- Chip 1 control
- ALU control/tester control
- Chip 2 control
- High-speed RAM control
- Chip 3 control
- SASS control
- Clock control
- Next-address/branch code type
- Branch control

The next-address field is used for both conditional and unconditional ROS branching. When branching conditionally, part of the next-address field defines the branch code type. The complete field is used as an address for branching unconditionally. Thus, the branch code type represents both address and data. Ten of the ROS output bits are used for the "next-address field." An additional bit is implemented as the "branch bit." If the branch bit is inactive, the branch is unconditional; if the branch bit is active, the five low-order bits of the next-address field are inspected for the branch condition. For a conditional branch not taken, the present ROS address is incremented by one to generate the next address for the ROS. If the branch is taken, conditionally or unconditionally, the next-address field is the next ROS address. This procedure continues until an "exit" is reached in the microcode sequence. At this time, the ROS address is generated from the instruction decode.

- *ROS design tool*

Previous design experience indicated that an improved method was needed to generate ROS code, to check the ROS, to personalize the ROS, and to document the code. Such a design tool should operate on a single source of data to enable the elimination of manual errors and to facilitate changes and verification.

A PL/I program was developed that took a single-source data set and generated physical addresses, checked function, and produced an easy-to-read document. This data set was also used to generate the ROS personality used on the chip as well as on the simulation models. The physical address generation relieved the ROS coder from having to be aware of physical constraints and allowed him to write code in a logical sequence. This facilitated readability of the automatically generated documentation and also eliminated the possibility of transcription errors. In addition, the labeling and footnoting capability enhanced the readability of the documentation.

A considerable amount of checking was built into the program. ROS control was chosen because it provides unambiguous control function and lends itself to clear, easy-to-read documentation. It is easy to modify and requires only personality changes on one level of the chip mask. The turn-around required to make a ROS (personality) change with this tool is measured in days instead of months for a circuit change. Due to the one-pass VLSI design commitment, the use of control ROS was instrumental in meeting the schedule.

Instruction fetch and execution

The instruction fetch and the decoding of the next instruction is overlapped with the instruction execution. Other machine functions, such as monitoring the external interrupts, channel requests, and instruction-counter synchronization, are performed in parallel with execution (see Fig. 3). Exception cases are handled such that execution is suppressed in accordance with the architecture. An eight-byte buffer is

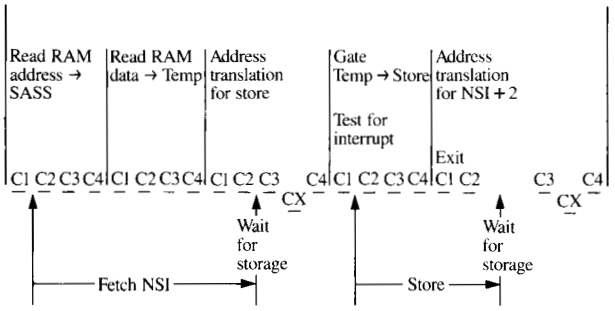

Figure 3 Instruction execution sequence. (Note: NSI = next sequential instruction.)

located in the storage control unit (SCU) as an instruction cache. The PU–SCU interface is asynchronous; a signal (Select) to the SCU indicates that a storage operation was required and that the PU–SCU interface is valid. The SCU then signals with a line (Ready) that the requested data are available on the data bus (if the operation is a read) or that the operation is complete (if the request is a write). If the PU reaches a point that it requires the SCU response, the PU stops executing and waits.

The time available for the SCU to respond to the PU varies depending on what the machine can do while waiting for the response. The microcode is written so that the PU will continue to execute until it requires a response from the SCU. If the instruction being fetched is in the instruction buffer, there is no wait period because the SCU will respond faster than is required. The point at which the PU waits for a response is dependent on the particular request to the SCU. If the request is for a fetch of the next instruction, the PU only requires one machine cycle to decode the instruction, so the wait is microcoded one machine cycle from the end of the instruction. If the request to the SCU is a data fetch, the wait is microcoded at the point that the PU is to operate on that data. The translation table operation for the next SCU operation can coincide with the current SCU operation.

A four-byte (fullword) data buffer is used in the SCU to allow a fullword store operation. Because of I/O pin limitations, the data interface between the PU and the SCU is a halfword. However, the interface between the SCU and the main store is a fullword. The architecture contains instructions that operate on more than one halfword at a time; the SCU implements a fullword data buffer to improve the performance of these instructions. The SCU can access the storage a fullword at a time while operating with the PU on a halfword basis using this buffer. This mechanism is also used in channel burst operations.

Integrated channel

As previously mentioned, the channel shares the processor logic for cost savings. It drives a halfword I/O port asynchronously. Data can be byte- or halfword-aligned and the channel can operate with combinations of byte or halfword devices. FRU isolation is provided by the channel hardware. The channel-SCU interface uses the SASS function.

Floating-point operations

A *floating-point* (FP) module existed for a previous machine, so development of a new module would not have been cost-effective. Instead, the FP module control was modified to map the machine interface of the processor to that of the FP. The key here was use of the variable-length machine cycle of the processor unit, which allowed the processor machine cycle to be mapped to the cycle of the existing FP module.

Interrupt mechanism

The purpose of the interrupt mechanism is to generate an internal mask which is used in monitoring the I/O interrupt request bus. Once an allowable request is found, this mechanism has to store information pertaining to the present level and retrieve necessary information for the level to which the machine has been interrupted (next level).

A significant part of the interrupt mechanism for this processor is contained in a ROS subroutine. The reason for this is that interruptions were infrequent relative to instruction execution, so performance degradation was an insignificant factor. By sharing existing function to handle interrupts, logic was minimized so the microprocessor could be contained within the four chips. The interrupt subroutine seeks out the highest-priority interrupt request among unmasked requests made from either the program or I/O devices. Once that request is found, the internal mask to be used at the next level is generated and stored in the interrupt register. The subroutine then compares the current level with the new level obtained to determine if they are the same. They could be the same since the machine could branch into the interrupt subroutine without changing levels from any one of several control instructions. If the two levels are the same, the subroutine exits and the processor executes the next instruction; if the two levels are different, the subroutine branches to the PSW swap routine.

An internal mask is used to look at the external I/O interrupts. If a higher-level unmasked request is sensed, or if the present interrupt is removed, the hardware requests a swap to the highest interrupt level pending. This design provides a fast detection of changes in external interrupts.

An additional feature incorporated into the interrupt design is a hardware retry. If a parity error is detected during

a sampling of the interrupt bus, the hardware performs an automatic retry. After seven tries, if the error still exists, the error is reported.

Clock design and timing considerations

The VLSI microprocessor clock is an LSSD clock design driven by a single-phase oscillator. A four-bit shift register generates four sequential pulses named C1, C2, C3, and C4. They switch after the rising transition of the oscillator input. The shift register is self-correcting and will not have more than one clock active at any one time after its initial start cycle.

The processor clock also generates four *feature clocks* called F1, F2, F3, and F4. These clocks are used by logic external to the processor; they become active at the same time as their corresponding "C" clock and remain active until the middle of the next sequential "C" clock.

The machine cycle time can be varied in two ways. Y clocks are dead times (no machine clocks are active) that occur between C3 and C4 (see Fig. 3). There can be up to three of them in one machine cycle. The Y clocks are under ROS control.

"Hold" clocks can occur between C2 and C3 and are of an indefinite length. They are controlled in part by an external event, usually asynchronous to the machine clock logic. Under ROS control, the clock logic can hold for storage or I/O operations. A storage hold will prevent C3 from becoming active until the storage has responded to a storage Select signal. An I/O hold will prevent C3 from becoming active until there is a response from an I/O device or a timeout exception occurs.

"*Storage clock*" is a square wave generated by the PU. The negative active levels of "storage clock" are coincident with machine clocks C1 and C3. If there are Y clocks between C3 and C4, the storage clock will remain active until C4 becomes active. If the clocks are being held or are stopped, the storage clock will continue. This clock provides synchronism for SCU–PU controls.

The clock logic can be stopped and then run in a manual mode under the control of a few external signals. A "stop" signal to the clock logic will cause the clocks to stop at the end of the current instruction. After the last C4, the clock logic will signal that it has stopped. If the stop signal is removed, the clocks will resume with C1. While the stop signal is active, the clocks can be run in a single-instruction or single-cycle mode. Single-instruction mode will cause the clocks to run until the end of the next instruction. Single-cycle operation will cause the clocks to run for one machine cycle.

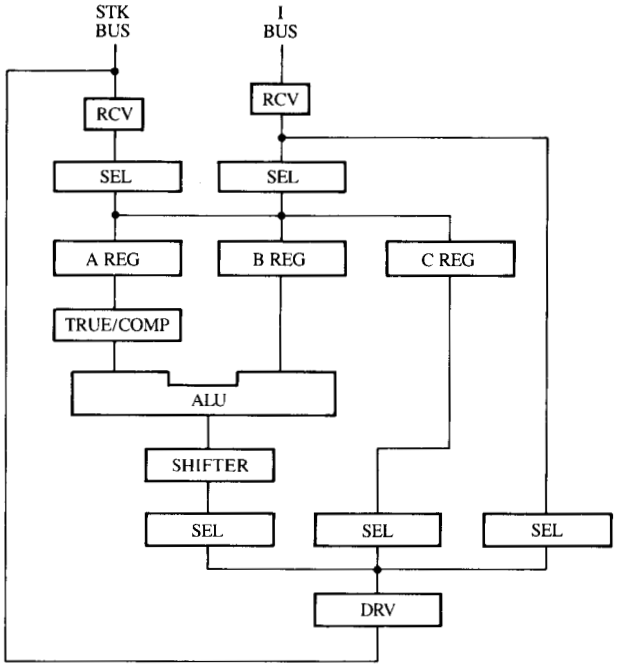

Figure 4 Arithmetic and Logical Unit (ALU) data flow. (Note: RCV = receiver, DRV = driver, SEL = selector.)

In order to minimize the effect of asynchronous external signals which could cause metastability in the logic, the signals are latched with a clock that will be inactive for one more clock period before the signal is used. Thus the primary latch may go metastable but propagation of the metastable signal is unlikely.

Arithmetic and logical unit design

The VLSI microprocessor ALU is used in the configuration shown in Fig. 4. The buses shown are all 16 bits plus 2 parity bits. The ALU design had the following objectives:

- 16-bit data flow, performing the standard ALU functions of SUM, AND, OR, and EXCLUSIVE-OR.
- Medium performance, allowing a propagation delay of two circuits per bit for the SUM operation.
- Output parity prediction based on input data and parity [9].

The ALU bit slice is shown in Fig. 5. Two of the eight circuits shown (those marked with *) are for parity prediction. There are four lines to control the ALU. The control lines are

- *SN*—Active if a SUM or AND operation is required.
- *SO*—Active if a SUM or OR operation is required.
- *SX*—Active if a SUM or XOR operation is required.
- *SNO*—Active if a SUM or AND or OR operation is required.

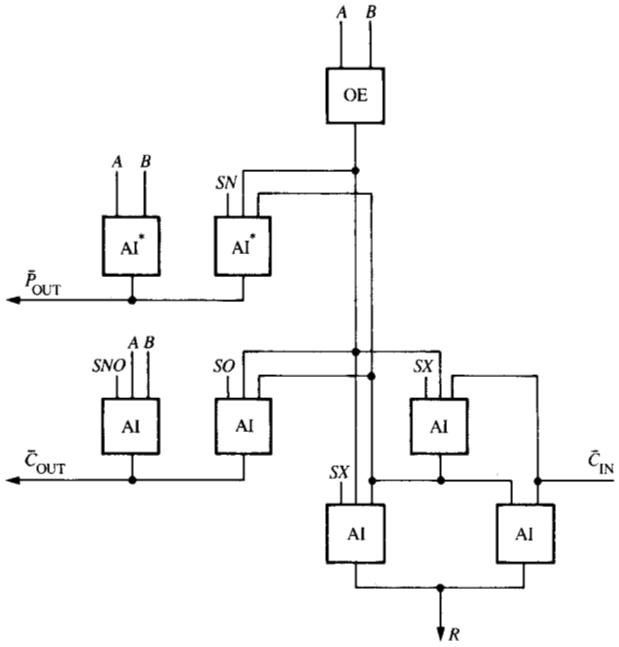

Figure 5 ALU bit slice. (Note: OE = EXCLUSIVE OR, AI = AND-INVERT.)

Additional parity-prediction logic required for each byte of the ALU is shown in Fig. 6 (here, the control lines are indicated as K). The resultant design achieves 100% single-error detection for all circuit faults except for control lines. Errors are detected by a parity check on the output data and predicted parity further on in the data flow.

One reason for the low circuit count is that the sum operation uses ripple carry propagation instead of carry-look-ahead. This allows each ALU bit to be identical and also simplifies the achievement of 100% single-error detection of circuit faults. An added benefit is that the traditional selector on an ALU which is used to select the desired function on the output can be eliminated.

The elimination of the output function selector further reduces the circuit count required. The penalty is that the OR and AND functions are shifted left one bit via the carry propagation lines. Since the ALU requires a shift-right selector for multiplication, a compensating shift right is performed at no cost in circuits or performance for every OR or AND function.

Circuit faults in the ALU are detected by a parity check performed on the output data and predicted output parity. This check is performed before the data are driven off-chip on Fig. 4. Other circuit faults, both before and after the ALU, will show up in the same check; that is why parity prediction rather than parity generation of the ALU output parity is used.

The equations for the predicted output parity are as follows:

SUM: $p(r) = p(a) + p(b) + p(c)$,

XOR: $p(r) = -p(a) + p(b)$,

AND: $p(r) = p(a) + p(b) + p(o)$,

OR: $p(r) = p(a) + p(b) + p(n)$,

where $p(r)$, $p(a)$, and $p(b)$ are the parity bits of the result, the a input, and the b input, respectively; $p(c)$ is the parity of the carry inputs into each bit; $p(o)$ is the parity of a OR b, $p(n)$ is the parity of a AND b; and the "+" operation is a logical OR. Note that the parity tree shown in Fig. 6 generates $p(c)$, $p(o)$, or $p(n)$ depending on whether the ALU operation is SUM, AND, or OR, respectively.

One hundred percent single-error detection can be demonstrated based on the fact that any circuit fault will affect an odd number of outputs which drive a parity check tree. To show this, note the following with reference to Figs. 5 and 6:

- The r outputs are connected to a parity tree further on in the data flow.
- The P_{OUT} lines (P_0 through P_7 and P_{IN} of Fig. 6) are connected to the parity tree shown in Fig. 6. This tree is active for all ALU functions except the XOR. The output is the predicted parity which is eventually checked in the data flow.
- The C_{OUT} line (Fig. 5) affects the R output of the next stage (the three AIs generating R are connected in an exclusive-or configuration) and either none or both of the next stage P_{OUT} and C_{OUT} lines. Thus, C_{OUT} affects an odd number of R and P_{OUT} outputs.
- Any circuit fault in a given stage will either affect P_{OUT}, C_{OUT}, or R alone, or will affect R and both P_{OUT} and C_{OUT}.
- For the XOR function, P_{OUT} and C_{OUT} are either not used or turned off. In this case any circuit fault will either affect the R output of the given stage or the R output of the next stage.

Storage protection

Storage translation and protection is allocated on 2K-byte blocks. Translation space is implemented in separate high-speed RAM, as illustrated in Fig. 2. In order to increase the logical address capability and flexibility, an identifier field (LOCK) is associated with each translation table entry. This LOCK, if not = 0, must match a KEY located in the PSW. If the KEY or LOCK = 0, the match is assumed. This allows

programs to be assigned both private and shared areas on 2K-byte × n blocks. This facility could assist a software storage management mechanism.

Work space

With the organization of the high-speed RAM that was chosen, there was a portion that was not required for general-purpose registers or translation tables. In order to meet the packaging objectives, this portion of the RAM is used as general work space (region WS, Fig. 2). Most of the registers necessary for the interrupt mechanism, the address recovery register, program control functions, and temporary registers are implemented in this area of the RAM.

A copy of the instruction counter (the recovery register) is saved here whenever a branch-type or jump-type instruction is executed. If an error occurs, the address of the instruction is derived from the recovery register and a two-bit hardware counter which keeps track of the displacement of the instruction address from the recovery register. The recovery register enables the instruction counter to be used as a data-address register during storage-to-storage operations.

By implementing control space in work space, hardware registers are minimized and control space is made available to the service tool.

Multiple processor configurations

The processor system can be attached to additional processor systems through the SCU. One or two SCUs can be attached to each processor system. If two are attached, each one addresses the even-odd word, respectively. This allows storage refresh and cycle overhead to be effectively reduced to zero. Each SCU contains contention logic, which allows two or more pairs of SCUs to share the same storage, thus allowing multiple processors to share the same storage with no new hardware. Function is built into the SCUs to allow dynamic duplication of translation tables to effectively allow each processor system to share the storage map (except for a fixed amount assigned to each processor). The processor systems communicate over the I/O interface through a shoulder-tap device.

Design procedure

The design responsibility of the processor system was divided into four groups: the PU/SASS/channel logic design, the SCU logic design, functional test generation, and hardware modeling. A nodal-equivalent hardware model was built [2]. Each custom chip was assigned to two or three people who had complete responsibility for the design. As the data flow progressed, a comprehensive chip description was written by each chip group. The ROS document was available early in the design. As changes were made to the ROS, a PL/I program automatically time-stamped the change in the ROS document.

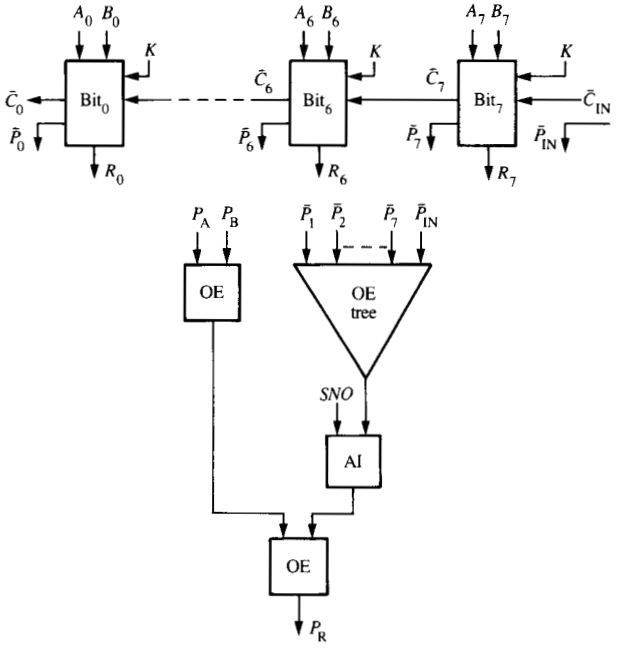

Figure 6 ALU bit slice connection and parity prediction. (Note: C_{IN} is equal to P_{IN}.)

A strict engineering change procedure was established to coordinate the design with the verification models and documentation. Changes had to be acknowledged by all groups before the VLSI change was complete.

RAS design

Parity is carried throughout the machine. ECC on main storage automatically corrects all single-bit and virtually all double-bit errors on both fetch and store operations. The SCU uses the ECC logic to predict parity, and main storage address parity is included in the ECC, so the SCU is highly checked. A diagnostic mode allows software checking of the SCU and storage.

More than 90% of the processor logic is error checked. The errors are reported in a set of registers which are available for software diagnostics and FRU isolation.

Due to the nature of the dense technology used, the failure rate is very low relative to previous technologies. The design itself uses three-sigma worst-case delay assumptions, providing manufacturing margin and increasing reliability. A special interface provides access to internal status for monitoring the machine. An engineering tool was developed to help the system designer build and debug the system.

Integrated test capability

With the density of the physical package of the processor, test requirements at all levels of the design had to be

established. On the basis of previous designs, the required test functions were known. The following are the major functions provided:

- stop
- single-cycle step
- single-instruction step
- read/write register by external means
- read/write storage by external means
- machine status (interrupt level, logical address, etc.)

Most of the information provided is internal to the VLSI microprocessor. To minimize the number of I/O connections needed to provide this information, the information is multiplexed over existing I/O. The multiplexing is controlled by the status bus previously mentioned.

Improved manufacturing test capability

Due to the large number of circuits on the chips, it was decided to follow the LSSD rules. This provided a mechanism to automatically generate dc test patterns and provide a procedure to determine the dc test coverage. This method allowed the standard procedures to be followed in submitting the chips for processing.

The chips (including the clock logic) were designed and checked to ensure that the LSSD rules were followed. Once the module design was complete, the module was checked to ensure compliance with the LSSD rules. This procedure resulted in a chip dc testability [10] greater than 99%, on all four chips, and a module testability of greater than 95%.

In the development of the circuit family for the VLSI microprocessor, circuit speed was an important factor. Like most circuit families that are designed for speed, there are components that are used for the exclusive purpose of speed; *i.e.,* the circuit will still operate properly without the component but at a slower speed. This creates a testing problem when a dc test is the only testing available. In order to ensure that the circuits are functioning properly, they must be ac tested (performance tested).

In the analysis of the problem for the processor system, performing an ac test at the component level and at the product level was considered. Looking at the resources necessary and the existing capabilities, it was determined that the product level test was best. In this case, the product level is the card that contains the VLSI microprocessor.

When the details of the available card testers were studied, the path to developing an ac test was clear. With the addition of one card to the tester, the microprocessor module was capable of executing machine code at machine speed when attached to the tester. The oscillator necessary to operate the processor was located on the added card. With this approach, the speed of the processor under test can be selected by the tester. The added card contained a test program that executed the complete instruction set of the processor with the exception of I/O instructions. The test consisted of 2000 bytes of code and exercised most of the circuits in the processor. Working with the card test facility, the test was assigned an 80% efficiency.

Conclusion

The design of the VLSI microprocessor showed that, with sufficient planning, a VLSI design can result in a one-pass machine. A key point is understanding the criteria to which the machine must adhere. As an example, understanding the machine software environment allowed tradeoffs between performance and circuit count which did not affect system throughput. Establishing automatic devices to facilitate the design and to minimize manual interaction resulted in limiting the main source of design error. The single source input for the ROS illustrated this point during the design cycle. While the ROS coding was a major cause of errors in previous designs for machines of this type, it was not in this design.

A comprehensive multiple-approach test strategy established early in the design resulted in a product that exceeded all test requirements, from the chip level through the final product. Early comprehensive documentation and close communication with all involved areas were essential in the design phase. With the many complex functions required, and with limited hardware resources to provide these functions, the design task could not be accomplished without early comprehensive documentation. The design environment was highly controlled (to control the progress of the design) without placing undue restrictions on designers; thus, a designer had primary responsibility for a function and secondary responsibility for the overall design. This design resulted in a fully functional first-pass machine.

Acknowledgments

Although significant contributions by a number of people were required to implement the design of the processor system, particular mention should be given to J. J. Kavaky, who developed most of the ROS design tools, and to R. J. Norman, who managed the design effort. Others to whom the authors are grateful include B. Bonetti. D. R. Kaufman, J. E. Lee, R. R. Michelin, A. S. Tran, G. R. Thompson, and S. Tung.

References

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. Arnold S. Tran, R. A. Forsberg, and Jack C. Lee, "A VLSI Design Verification Strategy," *IBM J. Res. Develop.* **26,** 475–484 (1982, this issue).

3. J. F. McCabe and A. Z. Muszynski, "A Bipolar VLSI Custom Macro Physical Design Verification Strategy," *IBM J. Res. Develop.* **26,** 485–496 (1982, this issue).
4. 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.
5. J. Z. Chen, W. B. Chin, T.-S. Jen, and J. Hutt, "A High-Density Bipolar Logic Masterslice for Small Systems," *IBM J. Res. Develop.* **25,** 142–151 (1981).
6. A. J. Blodgett, "A Multilayer Ceramic Multichip Module," *IEEE Trans. Components, Hybrids, Manuf. Technol.* **CHMT-3,** 634–637 (1980). See also B. T. Clark and Y. M. Hill, "IBM Multichip Multilayer Ceramic Modules for LSI Chips," *ibid.,* 89–93 (1980).
7. R. P. Oberly, "How to Beat The Card Test Game," *1977 Semiconductor Test Symposium,* Cherry Hill, NJ, 1977, pp. 16–18.
8. *Signetics 82S100 FPLA,* Signetics Corporation, 811 E. Arques Ave., P. O. Box 409, Sunnyvale, CA 94086.
9. D. R. Kaufman and G. R. Thompson, "Parity Check Circuit for Adder With Shared Logic," *IBM Tech. Disclosure Bull.* **23,** 668–669 (1980).
10. E. I. Meuhldorf and A. D. Savkar, "LSI Logic Testing," *IEEE Trans. Computers* **C-30,** 1–17 (1981).

Received October 1, 1981; revised February 12, 1982

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