An Asynchronous Instruction Length Decoder

Kenneth S. Stevens, *Senior Member, IEEE*, Shai Rotem, *Member, IEEE*, Ran Ginosar, *Member, IEEE*, Peter Beerel, Chris J. Myers, *Member, IEEE*, Kenneth Y. Yun, *Member, IEEE*, Rakefet Kol, *Member, IEEE*, Charles Dike, and Marly Roncken, *Member, IEEE*

Abstract—This paper describes an investigation of potential advantages and pitfalls of applying an asynchronous design methodology to an advanced microprocessor architecture. A prototype complex instruction set length decoding and steering unit was implemented using self-timed circuits. [The Revolving Asynchronous Pentium® Processor Instruction Decoder (RAPPID) design implemented the complete Pentium II® 32-bit MMX instruction set.] The prototype chip was fabricated on a 0.25-μ CMOS process and tested successfully. Results show significant advantages—in particular, performance of 2.5–4.5 instructions per nanosecond—with manageable risks using this design technology. The prototype achieves three times the throughput and half the latency, dissipating only half the power and requiring about the same area as the fastest commercial 400-MHz clocked circuit fabricated on the same process.

Index Terms—Asynchronous debugging, asynchronous design, asynchronous testability, domino circuits, handshake protocols, instruction length decoding, pulsed logic, relative timing, self-reset logic, self-timed.

I. INTRODUCTION

THE OBJECTIVE of this research was to demonstrate the ability to design high-speed asynchronous circuits [1] as a potential solution for microprocessor design if and when clocked design becomes too expensive.

We have designed an asynchronous version of the instruction length decoder of a commercial 400-MHz clocked processor [2]. For fair comparison, the prototype was implemented on the same 0.25-μ six-metal-layer CMOS process as the commercial processor. The asynchronous implementation achieved a higher performance at lower power.

The microarchitecture and circuits of the two designs, while achieving the same functionality, were substantially different. The asynchronous architecture exploits multiple interrelated data-dependent frequency domains and pipelining techniques that match a particular problem and data, rather than a chip-wide constraint. For example, the prototype circuit combines three domains operating at average rates of 3.6 GHz, 900 MHz, and 700 MHz.

Fig. 1. Instruction set statistics. Bar graphs show relative dynamic frequencies. Line graphs indicate cumulative frequencies. (a) Instruction length statistics. (b) Opcode type statistics.

Our asynchronous circuit design employs a novel methodology which adds static timing information to handshaking [3]. This enables smaller, more testable, faster, and lower power circuits. However, it introduces a potential problem of increased failure rate if timing margins are tight. This difficulty can be addressed in the future with better design and verification tools [4]. The asynchronous prototype design uses static and domino gates from a standard synchronous library, with a few custom circuits, such as C-elements [5].

The design was motivated by the observation that instruction length decoding could pose a bottleneck in variable length instruction set architectures. As reported in [6], our analysis of the variable length instruction set revealed two principal findings (Fig. 1). First, the average instruction length is about three bytes, and instructions longer than seven bytes are rare. Second, very
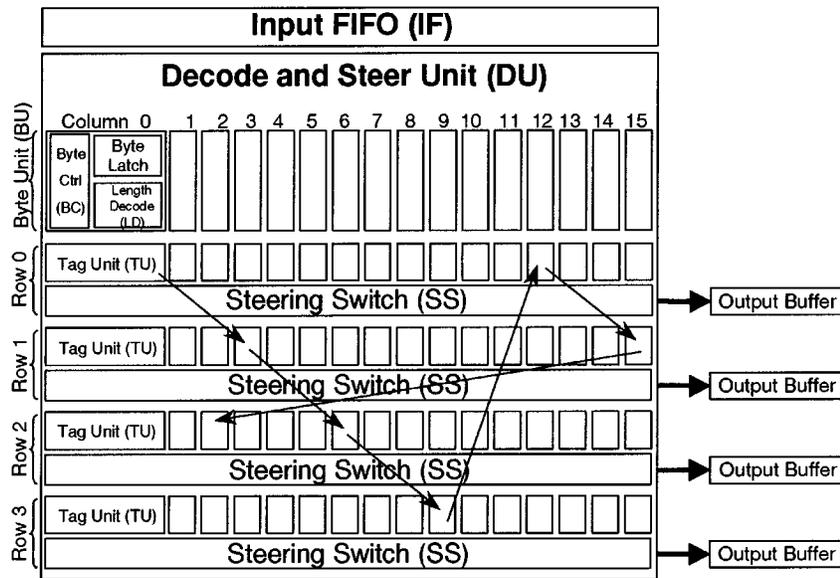
Fig. 2.   Microarchitecture.

few instruction types are used frequently. The asynchronous design exploits these findings.

In the rest of this paper, we present the microarchitecture and circuits, explain the circuit design methodology, and compare the prototype to a contemporary clocked commercial circuit.

II. MICROARCHITECTURE AND BASIC OPERATION

The decoding and steering unit (DU) receives 16-byte-wide instruction cache lines at its input, extracts the instructions, and places each instruction separately into output buffers. The core comprises three stages—a *byte unit* (BU), a *tag unit* (TU), and an instruction *steering switch* (SS), as shown in Fig. 2. The BU receives a 16-byte cache line and speculatively decodes 16 instruction lengths in parallel, assuming that each byte starts a new instruction. The TU in the first byte of an instruction passes a "tag" downstream to the first byte of the next instruction. The SS routes instructions on four separate 62-bit crossbar channels to the output. The TUs are replicated with the four steering switches. This distributed tagging and switching circuit with 16 columns and four rows is connected in a torus that packs the bytes into instructions and steers them into four output buffers. These dimensions are designed to balance the average computation rates.

A. Input FIFO

The input FIFO (IF) holds 32 16-byte wide instruction cache lines. The FIFO is an instruction delivery mechanism designed to operate faster than the DU. Unbiased maximum DU performance can be measured by keeping instruction delivery off the critical path.

The asynchronous FIFO is designed to mimic the microprocessor instruction delivery mechanism and to aid in evaluating the DU. Each instruction byte in the FIFO contains three additional bits derived from the branch target buffer (BTB) information. One bit each indicates whether this byte is used (U),

whether it is the first byte of a predicted taken branch instruction (B), and whether it is a branch target (T). If a cache line contains a predicted taken branch, the B bit will be set and the bytes following the end of the branch instruction up to the end of the cache line are marked as unused; therefore their U bits are cleared. The U bits are also cleared from the beginning of the next cache line up to the byte containing the branch target byte with bit T set. A target bit T will be set in the first cache line following reset.

Data in the Input FIFO can be recirculated so that a continuous, but repetitive, stream of cache lines can be supplied to the core. The continuous operation is essential for performance and power measurements. A second repetitive mode exists where the cache line at the head of the FIFO is repetitively presented to the DU.

The FIFO is loaded serially through a scan register. Once the FIFO is filled, the decoder reads lines from the FIFO. Every byte in the IF is controlled separately, so the IF effectively consists of sixteen separate 11-bit-wide parallel FIFOs. This structure allows the individual bytes to be transferred to the DU when needed, without having to wait for the DU to accept the next line in full. The IF is implemented as a Sutherland Micropipeline where the design of each stage is shown in Figs. 3 and 4.

B. Length Decoding and Steering Unit

The core of the asynchronous circuit is the length decoding and steering unit. The DU consists of 16 identical blocks, or columns, one for each input byte, and four output buffers. Each column consists of a BU, comprising the byte latch, byte control, and length decoder, and four identical tag units and steering switches. The length decoder implementation is optimized for common instructions, such that length decoding for common opcodes is faster than for rare ones [6]. The TUs and SSs are arranged in 16 columns and four rows, wrapped around in a torus. The horizontal toroidal wrap ensures that instructions from different cache lines are correctly packed into the output buffers. Each SS in the four rows is connected to an output buffer. Each
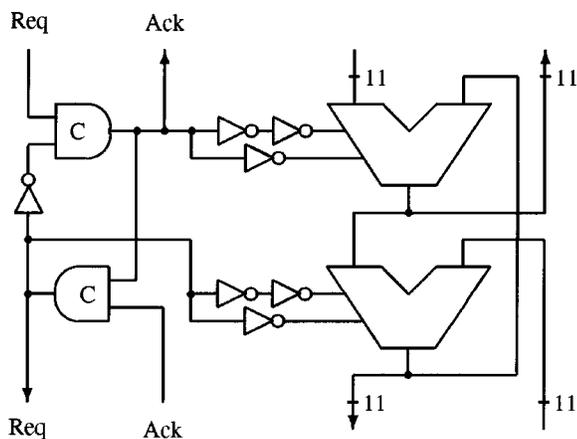
Fig. 3. One cell of the FIFO implementation. The two gates on the left are *C*-elements, and on the right are multiplexors. The design of these gates are shown in Fig. 4.
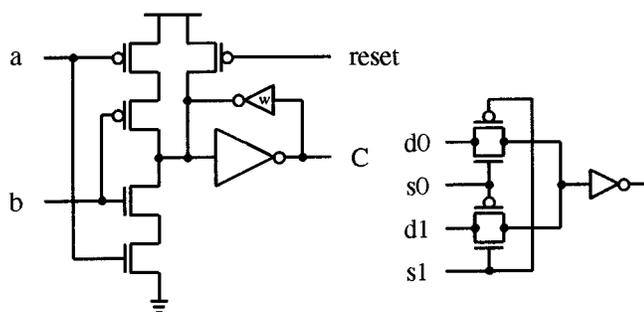
Fig. 4. *C*-element and multiplexor circuits in the FIFO. The output of the *C*-element raises when both inputs raise, and lowers when both inputs lower. This is a dynamic implementation that requires a week keeper.

line is implemented as a distributed self-resetting pulsed domino NOR gate driven and enabled by the data location in each column and row.

Each column receives a byte from its line of the instruction cache at the head of the IF, latches it in the byte latch, and performs a speculative length decoding assuming that an instruction starts at that byte. Each TU waits for the following three events to occur (see Fig. 9):

1) `TagIn`: A tag arrives from one of the neighboring columns upstream, indicating that this is the first byte of an instruction.
2) `InstRdy`: Length calculation for the column is completed and the instruction is ready, meaning that all the instruction bytes are ready in their byte latches.
3) `SSRdy`: The SS of the row is ready to issue a new instruction.

If all three of these events occur, which may happen in any order, the TU performs the following three operations in parallel:

1) sends a tag to the TU in the column of the next instruction's first byte in the next row;
2) transfers the instruction bytes, along with additional information on the length and prefixes, to its row's SS, which in turn forwards them to the output buffer;
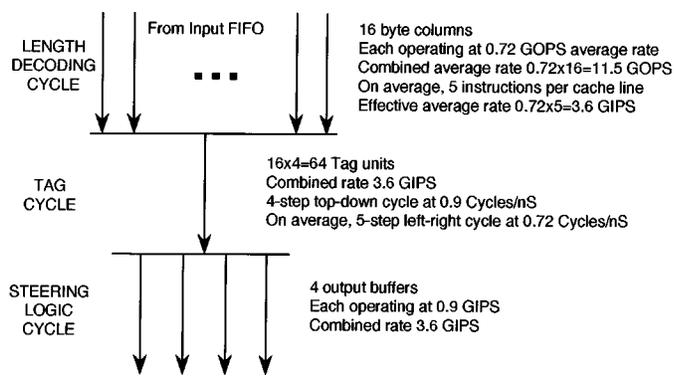
Fig. 5. Computation cycles and execution rates.

3) notifies the BUs in its column that the instruction data has been transferred from the byte latch to the SS.

That is, once the (speculative) length calculation has been completed at the column receiving the tag and the SS in the row of the receiving TU is ready (`InstRdy` and `SSRdy` have been asserted), the next tagged TU can immediately perform the above three operations.

When a BU is notified by one of its four TUs that the instruction has been transferred to the SS (operation 3 above), it opens its byte latch, which permits decoding of the next instruction to begin if it is available. The BU also notifies the other BUs containing the remaining bytes of this instruction that they may open their byte latches. In this way, the length decoding (which is a long latency operation) of bytes from the next cache line starts as soon as the bytes from the previous line have been consumed.

1) Balanced Design: The columns and rows are arranged in a torus. Hence each row is a ring around that torus. As the tag wraps around the torus and crosses from column 15 back to column 0, it falls to the next row. TUs in the fourth row send the tag to the first row. The operation would be balanced if the tagged column had decoded the length of the instruction by the time the tag arrives. Similarly, the corresponding SS would have had to complete the transfer of the previous instruction before the tag arrives. Thus, in a perfectly balanced situation, the `TagIn`, `InstRdy`, and `SSRdy` events would occur simultaneously. Unfortunately, this is not always the case because the latency of length decoding depends on the opcode, and special case handling of branches, long instructions, and prefixes incurs a longer latency.

The following example demonstrates the path of the tag through the TUs, assuming a sequence of 3-byte-long instructions, as shown by the arrows in Fig. 2: Column 0 row 0 → column 3 row 1 → column 6 row 2 → column 9 row 3 → column 12 row 0 → column 15 row 1 → column 2 row 2 → ···.

Operation of the asynchronous circuit consists of independent self-timed cycles. The major cycles are (see Fig. 5):

1) *The length decoding and instruction ready cycle.* This cycle accepts a byte from the IF, decodes the instruction length (as all necessary bytes become available), and generates the Instruction Ready flag (based on the calculated length and the Byte Ready bits from the byte latches of the remaining bytes in the instruction).
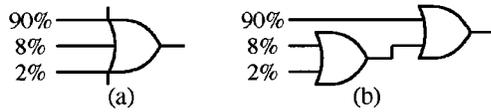
Fig. 6. Average delay optimization for the common case. Inputs labeled by probability. Circuit (b) speeds up the average delay by optimizing the common input signal at the expense of the less common signals.

2) *The steering logic cycle.* This cycle aligns instruction bytes from the byte latches and forward them to the output buffer over the SS.

3) *Tag cycle.* This cycles forward the tag to the start of the next instruction, and synchronizes the above two cycles.

Each cycle has its characteristic cycle time that can be independently optimized based on performance targets. The length decoding cycle is optimized for common instructions [6]. The tag cycle is optimized for common lengths, as discussed below. The steering logic cycle is matched to the throughput and latency of the output buffers. We can compose these cycles, using asynchronous protocols, in a scalable fashion to achieve the target system performance. This architecture is scalable in both the horizontal (length decoding cycle) and vertical (steering logic cycle) dimensions. We can increase the performance through additional parallelism (and area) by adding rows and columns to achieve the target performance.

Each cycle is balanced if its function can be completed just before its results are required. The cycle times are determined by the scale and wrap factors. Assuming an average instruction length of three, each 16-byte cache line holds about five instructions. Therefore the length decoding and tag cycles are balanced if the `TagIn` to `TagOut` latency is one-fifth of the decoding latency. The SS latency is four times the tag cycle latency, hence the TU and SS rows are scaled to four instances to keep the steering logic cycle balanced relative to the other two. The `TagIn` to `TagOut` latency is the critical path and receive the primary focus in the design; the other two cycles were scaled to match the average tag cycle time. Balancing pipelines with significant variation in response time, as is the case with this design, can be difficult [7]. We have recently developed a stochastic performance analysis tool that can help further optimize the design by considering synchronization point locations and delay distributions [8].

These three intertwined cycles demonstrate one advantage of the asynchronous solution. The tag cycle operates at an average rate of 3.6 GIPS (close to 4.5 GIPS in some of the tests, as reported below), consuming on average 720 million cache lines per second. Lines with fewer than five instructions (average length greater than three bytes) are consumed faster, whereas lines with more than five (shorter) instructions are consumed slower. The tag cycle, being the central point of gathering and distributing instructions, is the performance-critical component in this architecture. The steering logic cycles are shielded from variations in the length decoding cycle by the tag cycle.

The length decoder is optimized for common opcodes. Our benchmark analysis indicates that 15% of the opcode types are used 90% of the time (see Fig. 1). Asynchronous circuits can be optimized for the common case as shown in Fig. 6. The length
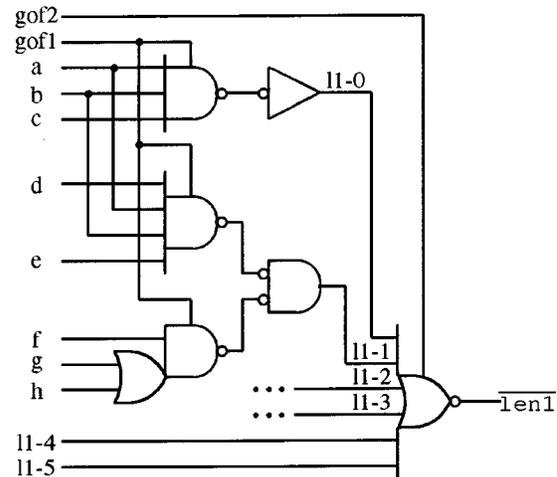
Fig. 7. Part of the logic decoding the length-one one-hot signal. Note that signals `l1-4` and `l1-5` are faster than the other terms. The logic cones for `l1-2` and `l1-3` are not shown for clarity. Gates with inputs on the top are footed domino gates, this input being the "clock" or reset signal.

decoding for common opcodes is done using domino logic; furthermore, the decoding of the most common opcodes is pushed closer to the outputs [6] as shown in Fig. 7. Rare opcodes are decoded using a NOR–NOR programmable logic array (PLA). Fig. 7 shows optimization for the common case. Notice how the common signals `l1-4` and `l1-5` skip directly to the front of the one-hot decoder.

2) Handling Long Instructions: The decoding and steering unit is optimized for instructions up to seven bytes long, which constitute 99.8% of the cases. Longer instructions (up to 11 bytes) are handled through a separate, slower protocol. Thus, each TU can directly tag the seven TUs in seven neighboring columns downstream in the next row down, and be tagged by any of the seven TUs in the seven neighboring columns upstream in the previous row up. The tags are sent via dedicated point-to-point lines. There are seven tag lines at the input and output of each TU.

Instructions longer than seven bytes are transferred to two steering switches and output buffers in two consecutive rows. The first four bytes (head) of the instruction are transferred to the SS in the row containing the tagged TU for the instruction's first byte, and the remaining bytes (tail) are transferred to the SS in the next row down.

If the calculated length is greater than seven, the BU waits for a tag to arrive. If this column is tagged, the BU signals the column containing the fifth byte of the instruction that it holds the first byte of the instruction's tail. The length of the instruction is also passed to the fifth column through three dedicated lines. The fifth byte's BU modifies its length to 4, 5, 6, or 7 (for total instruction length of 8, 9, 10, or 11, respectively) and sends an acknowledgment to the first byte's column. Upon receiving this acknowledgment, the first byte's column modifies its length decoder's output to four. The tagged TU in that column then operates as if the instruction length were four. The first four bytes of the instruction are transferred to the SS (together with an indication that it is the head of a long instruction), and the tag is sent to the TU in the fifth byte's column, in the next row down.
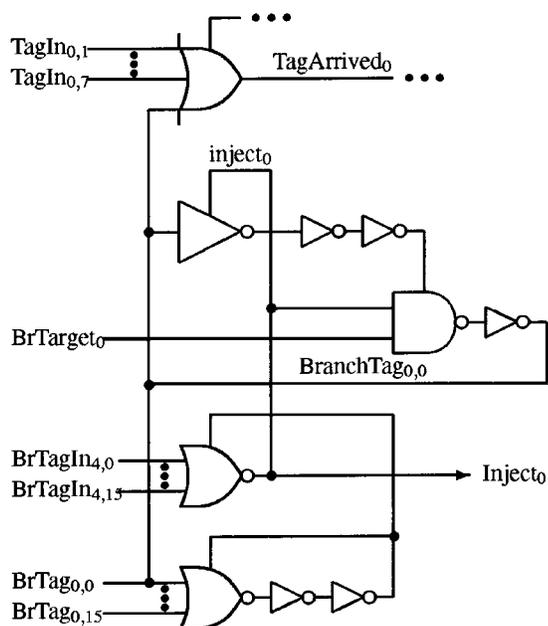
Fig. 8. Branch control circuitry, labeled for row 0 column 0. Interaction with the TU circuit shown.

Fig. 9. TU circuit (simplified for clarity). Branch control circuitry shown in Fig. 8.

The fifth byte's column operates as if it were the first byte of a short instruction. It transfers the tail to the SS in the tagged TU's row, and sends the tag to the first byte of the next instruction.

Instruction prefix bytes, including length-modifying prefixes, are handled in a similar manner.

3) Handling Branch Instructions: When a cache line contains a predicted taken branch instruction, the tag should be routed from the TU of the branch instruction's first byte to the TU of the branch target's first byte. The target always resides in the next cache line (since the fetch unit is designed to fetch the target cache line of predicted taken branches), so the bytes in between the branch and the target instruction are skipped.

The first bytes of the branch and target instructions are marked in the Input FIFO with B and T bits, respectively, and the unused bytes in between the branch and target instructions have their used (U) bits reset. The B and T bits from the byte latch are routed to all four TUs in that column. When a branch instruction is tagged, the corresponding TU foregoes forwarding the tag to the first byte following its length since that byte may not be the start of the target instruction. Instead, `BranchTagIn` is sent to the next row that asserts the `inject` signal as shown in Fig. 8. Each row has a local `inject` signal that is routed to all TU's in that row. The column's T bit will assert the `BranchTarget` signal. When a row's `inject` signal and a column's T bit are both asserted, the branch tag is generated for that TU and the row's `inject` signal is de-asserted. This mechanism forward the tag from the branch to the target instruction without tagging intermediate bytes. From that point onward, the operation continues normally.

Logic decoding the B, T, and U bits is not implemented in the prototype. They are supplied pre-decoded in the IF.

III. DECODER CIRCUITS

We briefly describe two principal circuits in the prototype. The TU circuit demonstrates the use of pulse logic and reduced
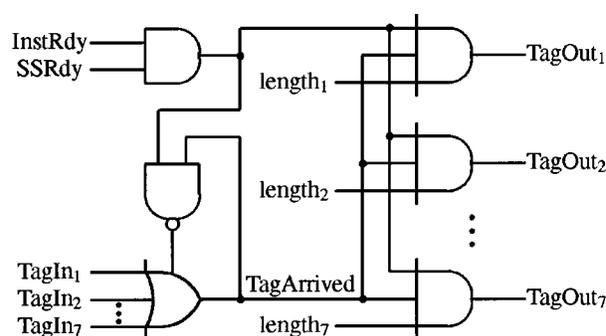
handshake, whereas the byte control circuit provides some insight into the complexity of the design.

A. Tag Unit Circuit

The TU is responsible for transferring the tag from the column containing the first byte of an instruction to the column containing the first byte of the next instruction. There are seven `TagIn` inputs to each TU, and seven `TagOut` outputs (Fig. 9). Additionally, special `TagIn` and `TagOut` lines are used for branch handling.

Transferring the tag to the next TU involves a full request–acknowledge handshake cycle if a speed-independent protocol were used [9]. That would require a `TagOutAck` acknowledge signal for each of the seven `TagOut` outputs. Such a structure will significantly complicate and slow down the TU logic and wiring. In order to simplify the implementation, the `TagOut` signals are implemented as self-timed *pulses*, eliminating the need for acknowledgment signals. However, the pulsed implementation is correct only under the following timing assumptions [10], [3]:

1) When a TU sends the tag pulse to the next TU, the receiving TU is ready to accept it, i.e., the self-resetting signal `TagArrived` (Fig. 9) is off;

2) the `TagOut` pulse is wide enough to cause a state transition in the receiving TU, i.e., the `TagArrived` signal becoming asserted;

3) the `TagOut` pulse is narrow enough so that it is de-asserted before the `TagArrived` indication in the receiving TU is de-asserted.

The first assumption is satisfied by the microarchitecture. When a TU sends the tag downstream, it resets its internal `TagArrived` line. The next time this TU can receive a tag is after the tag has wrapped around the torus horizontally and vertically. The tag must make at least four hops (over the four rows) before returning to the same TU. This delay can be guaranteed to be longer than the time it takes to reset the `TagArrived` line.

The second and third assumptions are satisfied by careful circuit design. The `TagOut` outputs are generated from the `TagArrived` signal, which is in turn generated by a self-resetting circuit.

This timed circuit was "hand-designed" with the relative timing methodology [3] and time-verified with ATACS [11].
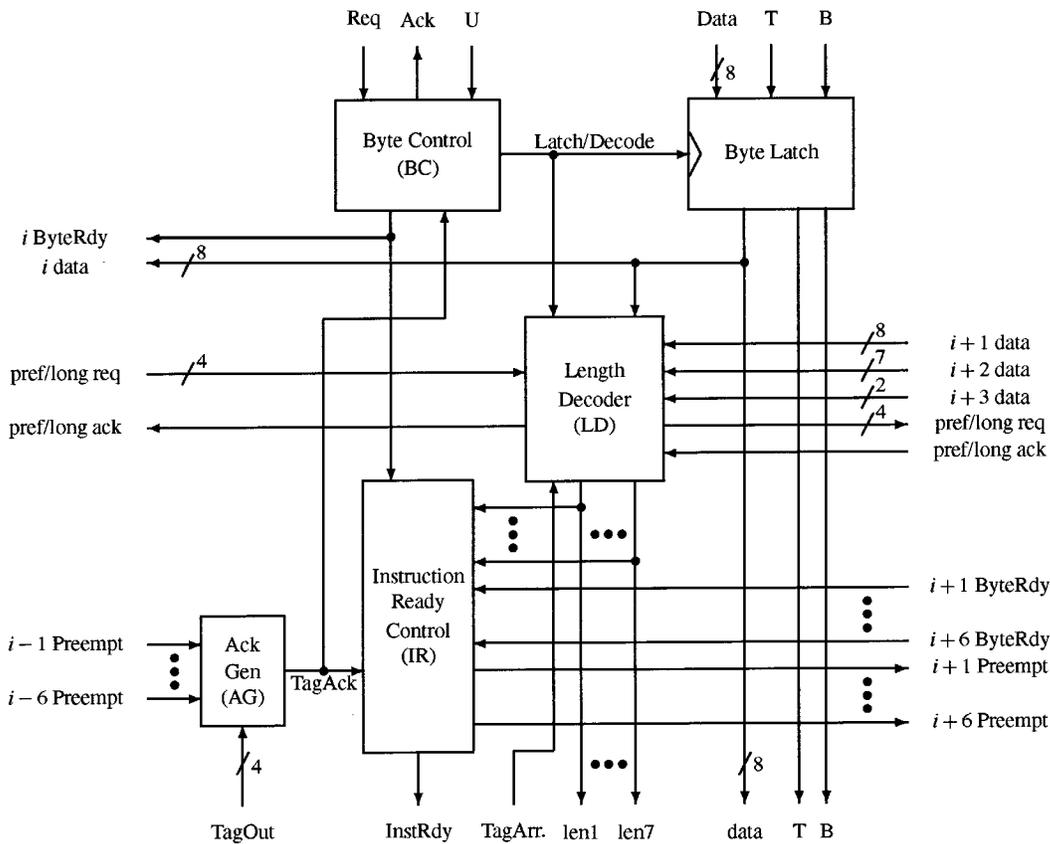
Fig. 10.   Byte unit block diagram.

Current advances in synthesis allow us to automatically synthe-size this circuit [12]. The circuits implementing the handshake interfaces between the TU, the byte control and the SS were also optimized using similar timed circuits and Relative Timing methodology.

B. Byte Unit Circuit

The BU is shown in Fig. 10. Each byte latch is a simple transparent latch. Length decoding may require, for some in-structions, bits from the following three bytes. In addition, if a length-modifying prefix byte precedes the instruction, or if the byte is part of a long instruction, additional control bits from up-stream are required. The length decoder produces seven one-hot encoded length bits. The decoder is implemented as a multistage unfooted domino PLA [6].

The byte control (BC) finite state machine (FSM) acknowl-edges the IF as soon as an incoming byte is latched. If the byte is marked unused (the U bit is set), the BC issues a pulse on the ByteRdy line. Otherwise, it closes the latch and initiates length decoding (by asserting the Latch/Decode signal), and asserts (nonpulsed) ByteRdy. The instruction ready control (IR) waits for both the locally decoded length and the ByteRdy signal from $L-1$ neighboring columns downstream (for length L), before generating InstRdy for the TU. The BC circuit is shown in Fig. 11. Many FSMs such as the byte control were designed using the 3-D synthesis tool [13], [14] and optimized using the relative timing methodology [3]. The actual circuits employ some pulsed signaling (TagAck is pulsed) and partial handshakes.
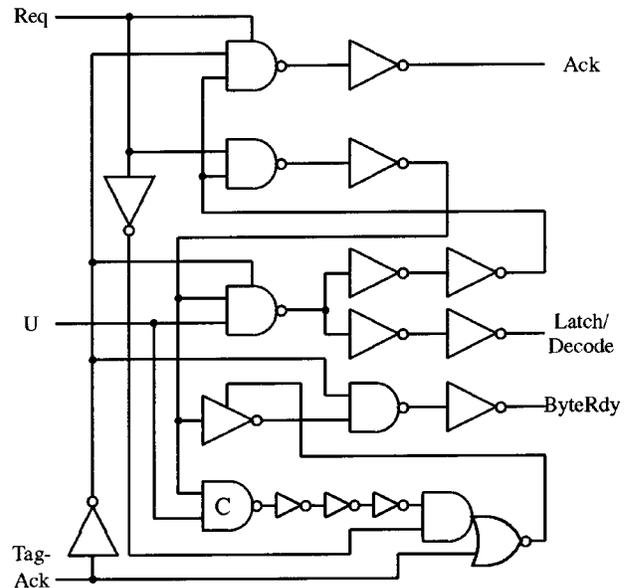
Fig. 11.   Byte control FSM. Req and Ack signals interface the FIFO to the BU.

Once a tag arrives at the column (TagArrived in Fig. 9 is set), the length decoder is notified (this signal is needed for handling prefixed and long instructions). Furthermore, once the tag is sent out (one of the TagOut signals in Fig. 9 is set), implying also that all bytes of the present instruction have been steered out through the SS, the AckGen FSM (AG) instructs IR and BC to get the next byte. IR then sends the corresponding
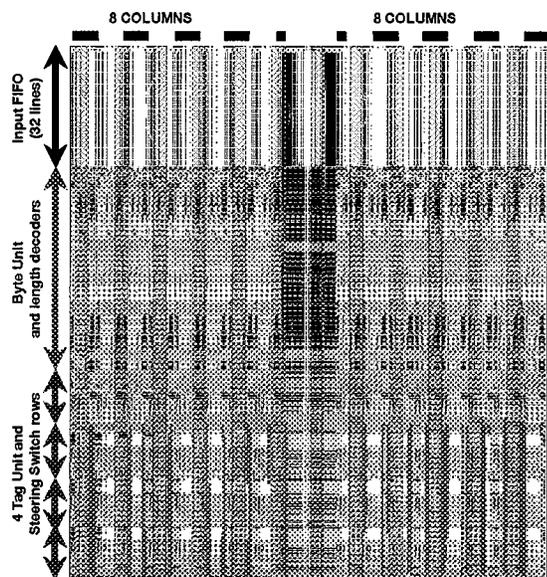
Fig. 12.   Circuit layout plot (3.1 × 3.5 mm).

Preempt signals (acknowledging `ByteRdy`) downstream to the remaining bytes of the instruction so that the length decoders for these columns can abort and reset upon receiving the `Preempt` signals.

At the (nonfirst-byte) columns that do not receive the tag, the LDs may output the length and the IRs may generate `InstRdy`. However, as soon as `Preempt` is received `InstRdy` is lowered before a tag can arrive at this byte column. The length decoders will be reset and a new length calculated before `InstRdy` is re-asserted and a new length is generated.

IV. PROTOTYPE TEST RESULTS AND COMPARISONS

The prototype was fabricated in May 1998 using a 0.25-μm six-metal-layer flip-chip technology. The layout plot is shown in Fig. 12, and only shows the first three layers of metallization so that the circuit structure can be more easily seen. The prototype was tested successfully, and the results are explained and analyzed below.

A. Performance

Decoding and steering performance of the test chip was measured at 2.5–4.5 instructions per nanosecond for average instruction streams. This is approximately three times the peak performance of the fastest synchronous three-issue product in the same fabrication process clocked at 400 MHz, achieving a peak decoding and steering performance of 1.2 instructions per nanosecond. The asynchronous decoder's performance is very data-dependent, and these results are valid for an average instruction stream containing common instructions of up to seven bytes long. The asynchronous design is not optimized for uncommon instructions, and the effects of rare, long, branch, and prefixed instructions on performance are not reported. Note that the steering logic issues four instruction streams rather than three, so the comparison is not completely fair.

Performance of the test chip is reported at nominal V_{cc} (1.8 V) and temperature (27 °C). The prototype was measured

at varying levels of V_{cc} for a subset of the instructions, and was determined to be operational in the range 1.0–2.0 V. The test chips were not tested above 2.0 V. Note that a synchronous processor operating at a fixed clock frequency can tolerate a very narrow range of environmental conditions (e.g., 1.9–2.1 V for the 400-MHz processor [2]). A certain margin is required to ensure that the clocked circuit operates across the specified range. However, no margins need be introduced into the asynchronous design since the circuits are not constrained to operate at a certain frequency. Rather, under unfavorable conditions, such as low voltage, the asynchronous circuit simply slows down. Thus, our asynchronous prototype can operate under the wider range of 1.0–2.0 V.

The latency from the byte latch to the output buffer for common length-two instructions has been found to be only 42% of the 400-MHz clocked circuit's latency. The main reasons for the reduced latency are the absence of clock boundaries at which the fast data must wait, and the ability to pipeline at frequencies matching data-path delays. In a clocked design with a multiple issue rate, several instructions are transferred on the clock edge. Since the first instructions becomes ready before the last (due to the serial nature of length decoding), they must wait before they are transferred to the next pipe stage. In the asynchronous implementation, every instruction is transferred as soon as it becomes available and the time for which an instruction waits is not frequency-dependent.

Table I contains measured performance data for some individual instructions. Tests X0–X8 use different mixes of length-one and length-two instructions. These nine tests consist of a 16-byte-wide cache line filled with 0 to 8 length-two instructions followed by 16 to 0 length-one instructions (test Xi consists of i length-two instructions followed by 16-2i length-one instructions). All the length-two instructions in the Xi tests could be fully length decoded using only the first byte. Test I0 consists of eight length-two instructions containing ModR/M length information in the second byte, complicating length calculation [15]. A noise problem with some instructions resulted in a violation of the setup time at the length decoder inputs, so we opted to use a single cache line for testing all instructions. The single cache line is repeatedly read from the head of the input FIFO, keeping the FIFO loop off the critical path.

The measured performance numbers were compared to those obtained with the COSMOS switch-level unit-delay simulator [16], and found to have an excellent correlation. This enabled us to estimate the performance of tests that failed on silicon.

B. Power

The measured power of the test chips is compared to the simulated power of the logic performing the length decoding and instruction steering of the comparable clocked circuit. The comparison was made using the integer power tests from the commercial clocked processor power test suite. The results show that the asynchronous decoder consumes about one-half the energy of the clocked design.

Since execution times differ greatly between these designs, we calculated the energy required to execute one loop of the test program. For the sake of power measurements, the FIFO was placed in a "frozen" debug mode where it repeatedly supplied the first cache line to the asynchronous core. This made

TABLE I
ASYNCHRONOUS DECODER PERFORMANCE TESTS

Test	Throughput [Inst./nSec]	Silicon/ COSMOS	No. of Instructions	No. of Lines	Test Description
X0	4.42	Si	16	1	16 Length 1
X1	4.41	Si	15	1	1 Length 2, 14 length 1
X2	4.39	Si	14	1	2 Length 2, 12 length 1
X3	4.48	Si	13	1	3 Length 2, 10 length 1
X4	4.44	Si	12	1	4 Length 2, 8 length 1
X5	4.34	Si	11	1	5 Length 2, 6 length 1
X6	4.12	Si	10	1	6 Length 2, 4 length 1
X7	4.12	Si	9	1	7 Length 2, 2 length 1
X8	4.00	Si	8	1	8 Length 2
I0	3.29	Si	8	1	8 Length 2 w/ModRM
PowerI1	2.44	COSMOS	74	21	1st Integer power test
PowerI2	2.49	COSMOS	72	20	2nd Integer power test
Powerf	2.93	COSMOS	81	26	FP power test
Mix0	3.48	COSMOS	77	14	Length 1–5 mix
Mix1	3.35	COSMOS	98	18	Length 1–7 mix
C34	3.10	COSMOS	5	1	4 Length 3, 1 Length 4
C223	3.65	COSMOS	6	1	2 Length 2, 4 Length 3

disassociating the FIFO power from the power dissipated by the decoder core easier. Therefore we measured the power of each instruction individually. The inner loop of the integer power test contains ten different instructions, so we generated ten separate tests, each measuring the power of one of the instructions. Each such test consisted of one instruction from the test set padded by length-one instructions to the end of the line. The power of each individual instruction was calculated by subtracting the power of the length-one instructions. The power for the complete test was calculated by multiplying the frequency of each instruction in the test by the occurrence count. These results compare processors executing at different speeds and only compare a small set of instructions. A more accurate comparison, which is beyond the scope of this research, should include a power-performance curve over a larger instruction mix, as well as measuring a real instruction stream rather than employing the frozen debug mode.

The prototype was not optimized for low power. Its superior efficiency is due only to its asynchronous design and our specific asynchronous design methodologies. For example, clocked methodology typically requires data to move between latches each clock cycle. In this design, data (instruction bytes) are latched in the byte latch and directly transferred to the output buffer only if and when the bytes are needed.

C. Area

The area of the prototype was compared to the area of a 400-MHz clocked circuit performing the similar functionality designed on the same 0.25-μ process. While we had layout and schematics for both designs, calculating an accurate comparison was time consuming due to the following issues, and resulted in some minor inaccuracies:

1) The three-issue instruction steering logic in the clocked design contained considerably more functionality than the comparable four-issue circuit in the asynchronous design.
2) Significant differences existed between the floorplans.

3) The prototype does not handle the instruction pointer, illegal opcodes, and bogus branches.
4) Some of the clocked circuits contain unrelated logic, and isolating the relevant parts is difficult.
5) The prototype layout was not optimized for density due to resource limitations.

Our analysis shows that the test chip occupies 22% larger area than the clocked design, which is a very reasonable area penalty for the improvements in throughput, latency and power. Furthermore, our analysis indicates that there is no evidence of a large area penalty inherent to asynchronous design.

D. Silicon Debugging

Debugging an asynchronous circuit on silicon without direct probing may be an issue since the circuit is self-timed, that is, it is impossible to stop the clock and scan out the state signals. This is especially true with the self-resetting pulsed circuits used in the asynchronous design, since by the time the circuit stops the signals have already returned to their initial states.

A special debug feature was designed to facilitate silicon debugging. Eight bits in the scan-in chain are dedicated to this feature. Each bit, when reset, blocks the resetting of a set of internal state signals. Additional logic required to implement this blocking is minimal. In most cases, it just required adding one input to an existing gate (Fig. 13). When the bit is reset, the self-resetting loop is disabled and the entire circuit will eventually halt. The circuit state can then be scanned out for inspection. Alternatively, operation can be resumed by removal of the debug bit value.

All these additions were made off the critical paths (the reset path is usually noncritical). Frozen state signals can be scanned out and observed. The debugging logic enabled us to identify three different timing-related failures of the first silicon in a very short time.

A pulse signal that was designed to drive a single local gate was later changed to drive an additional distant gate. The additional load from the gate and wire exceeded the pulse drive ca-
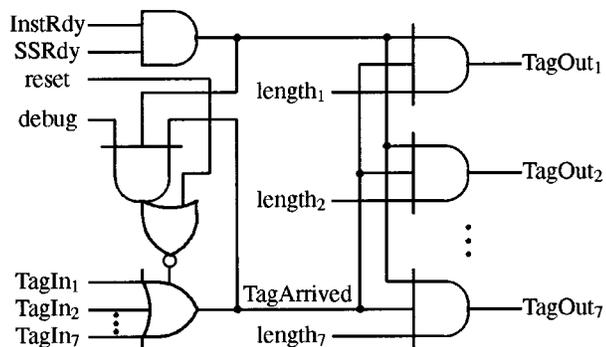
Fig. 13. TU circuit showing the `debug` signal which captures the `TagArrived` pulse.

pability. Through the debugging logic we were able to identify the failing signal. This bug prevented the circuit from operating. Subsequently, the silicon was modified to enable testing of the circuit.

Another timing related bug led to too short a delay on a precharge control line for a clocked domino PLA. This resulted in malfunctioning of the rare instruction NOR–NOR PLA, which was consequently excluded from testing. The third bug, mentioned previously, is a noise condition on certain instructions that resulted in insufficient data setup to the length decoders.

Timing analysis was not performed on the prototype circuit, other than through designer-driven spice runs. These bugs would have been discovered with a complete timing validation flow [3], [4].

E. Testability

Fault analysis of the prototype provides evidence that testability is no reason to avoid asynchronous design. In fact, the fault coverage was no worse than that of similar synchronous circuits. Almost all uncovered faults in the asynchronous design would also be uncovered in similar clocked circuits. However, some issues specific to asynchronous design have been identified. Asynchronous circuits are sequential in nature due to the handshake protocols implemented with finite state machine controllers. Unlike clocked circuits, it is unreasonable to apply scan techniques to convert asynchronous circuits into combinational blocks due to the large state space of the distributed and autonomous handshake control [17]. Further, synchronization points are decoupled, which complicates observing the global state space with a clocked tester. Another testability issue with timed asynchronous circuits is the potential necessity of modeling delay faults.

Since full scan is unreasonable, we opted to use built-in self-test (BIST) to avoid invading the structural design with flops. Cellular automata (CA) [18] were designed to generate test vectors targeted at the terms in the decode PLA of the instruction set [19]. We used one BIST block to test the entire asynchronous circuit (approximately 120 000 transistors). The BIST structures were attached to the interfaces of this block after the design was complete, and thus no logic modifications or design for testability (DfT) were applied to the decoder core. A CA signature analyzer validates correctness by observing the output signals and some important internal states. The BIST scan and debug scan are integrated and share the same flops. The BIST and debug logic has a small impact on performance

and area, estimated at 5% latency penalty (no throughput penalty) and 5% area penalty.

One modification was made to the cellular automaton to generate the sequential patterns needed to test for two-opcode instructions which were too rare to be automatically generated by the automaton. A similar modification was required to generate sequences of prefixes that modified the lengths of subsequent instructions, but was not implemented. The BIST and signature analyzer are clocked at a frequency slow enough to guarantee stability of all nodes at observation time. The circuit still runs at full speed internally.

A 95.9% stuck-at fault coverage was achieved using the COSMOS switch-level fault simulator [16]. Untargeted faults—some sequences of prefix instructions and debug logic—were not included. The BIST logic was not implemented on silicon due to schedule constraints. It was designed at schematics level and simulated. Faults were simulated in one column only, and only in one of the TUs in this column, in order to keep runtime reasonable (145 CPU days). We expect the coverage for all blocks to be nearly identical independent of their position in the array.

The majority of the uncovered 4.1% of the faults were not due to the shortcomings of BIST, but rather to the circuits and design style used. The uncovered faults consisted mainly of unobservable keeper faults, pulse degradation faults in domino keepers (see below), and redundant circuits which were not removed with the relative timing methodology [3]. Other than redundancy, the same types of faults appear in clocked circuits [20], [21].

The circuit in Fig. 14 demonstrates an undetected fault specific to pulsed circuits used in the prototype. Normally, the domino NAND gate G self-reset is controlled by the seven gate delay feedback pulse through r. A stuck-at-zero fault on the keeper in gate G leads to an early reset through the three gate delay feedback path d. This type of fault may or may not result in failure in the actual circuit and should be simulated for realistic noise, coupling, etc., to determine if the shortened pulse results in failure.

V. DISCUSSION

The comparison of synchronous and asynchronous circuits made in the previous section is limited by the fact that we did not have a separate clocked chip implementing the same functionality of our asynchronous prototype. Thus for data such as power, we had to resort to simulation and indirect estimates. However, actual throughput, delay, and silicon area characteristics of the clocked design have been employed in this comparison.

We summarize some of our key observations below. In the early design stage, we learned how to optimize asynchronous circuits mainly for high performance at the microarchitecture level.

1) Optimize for the common cases. The tagging circuit is optimized for instructions up to seven bytes long, and the length decoder for common instructions [6].
2) Employ timing assumptions, direct signaling, and pulsed logic to avoid the full handshake overhead [3].
3) Use a one-hot domino circuit with automatic completion detection, e.g., for the length decoder.
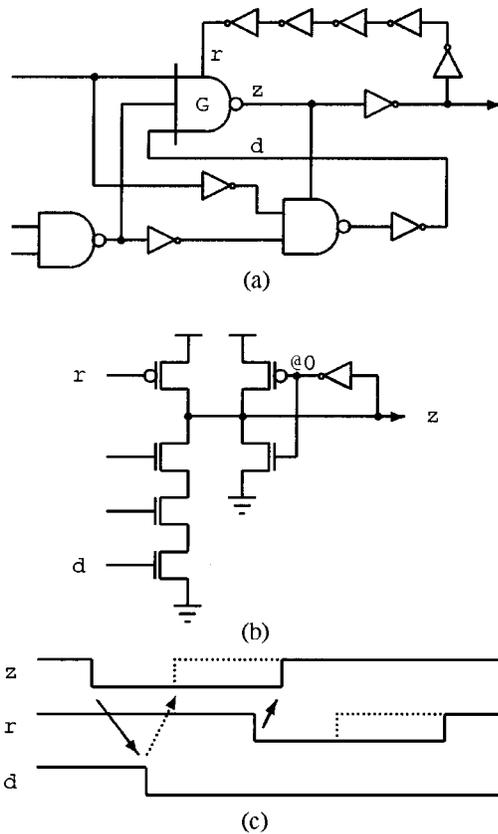
Fig. 14. Pulse length fault. (a) Self-resetting domino circuit generating a seven gate delay pulse. (b) Stuck-at-zero fault on domino keeper of gate G. (c) Dotted waveforms show pulse degradation under fault, arrows show causality.

4) Scalable parallel operation can balance the various operational rates for performance: Four rows of tagging units and output buffers match the tagging time to the instruction steering time.

5) Preempting asynchronous circuits is possible. The length decoder is preempted, reset, and restarted in bytes that do not start an instruction, as well as in the case of prefixes and long instructions.

6) Global synchronization is decoupled. Wide synchronization is inefficient in asynchronous design. The synchronization can at times be deferred by splitting the architecture into concurrent paths and moving the synchronization to a less expensive location. For example, the prototype does not synchronize all sixteen cache line bytes at the input; rather, the bytes proceed along concurrent paths, and only get synchronized at the most opportune time by the TUs.

In the later stages of the design, key observations were mostly related to methods for asynchronous control circuit optimizations [3].

1) Relative timing assumptions were used to simplify the control circuits thus increasing their performance.

2) Relative timing assumptions were added to the formal verification tool ANALYZE [22].

3) Pulsed pipeline control simplified the circuit and increased performance.

4) A footed rather than unfooted domino may yield a faster circuit due to relaxed race conditions.

Self-timed circuits are a potential solution to future design problems like delay variations and clock distribution. We are investigating the adaptive synchronization scheme for communication among units on-chip in the presence of large clock skew [23] and a scheme to embed self-timed modules without significant latency penalty in globally synchronous systems [24]. We are also designing a complete CAD system for timed circuit design [25]–[27], and are working on DfT solutions for the undetectable faults in self-timed circuits. Such CAD and design techniques are a potential solution to the issues we will face in the future, given current trends of increasing clock frequency, interconnect delays, and delay variations.

VI. Conclusion

Our novel design methodology for asynchronous circuits and systems has resulted in a circuit that achieves three times the performance of its high-performance commercial synchronous counterpart, incurring half the latency and consuming half the power, at a comparable silicon area. We have found that the main limitation to exploiting this potential is the lack of appropriate CAD tools [4].

Acknowledgment

References

[1] S. Hauck, "Asynchronous design methodologies: An overview," *Proc. IEEE*, vol. 83, pp. 69–93, Jan. 1995.

[2] Intel Corporation. (2000) Pentium II Processor at 360 MHz, 400 MHz, and 450 MHz Datasheet. [Online]. Available: http://developer.intel.com/design/PentiumII/datashts/243657.htm

[3] K. S. Stevens, R. Ginosar, and S. Rotem, "Relative timing," in *Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, Apr. 1999, pp. 208–218.

[4] K. S. Stevens, S. Rotem, S. M. Burns, J. Cortadella, R. Ginosar, M. Kishinevsky, and M. Roncken, "CAD directions for high-performance asynchronous circuits," in *Proc. Digital Automation Conf. (DAC'99)*, June 1999, pp. 116–121.

[5] M. Shams, J. C. Ebergen, and M. I. Elmasry, "Modeling and comparing CMOS implementations of the C-element," *IEEE Trans. VLSI Systems*, vol. 6, no. 4, pp. 563–567, Dec. 1998.

[6] W. Chou, P. A. Beerel, R. Ginosar, R. Kol, C. J. Myers, S. Rotem, K. S. Stevens, and K. Y. Yun, "Average-case optimized technology mapping of one-hot domino circuits," in *Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, 1998, pp. 80–91.

[7] D. Kearney, "Theoretical limits on the data dependent performance of asynchronous circuits," in *Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, Apr. 1999, pp. 201–207.

[8] A. Xie, S. Kim, and P. A. Beerel, "Bounding average time separations of events in stochastic timed petri nets with choice," in *Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, Apr. 1999, pp. 94–107.

[9] C. L. Seitz, "System timing," in *Introduction to VLSI Systems*, C. A. Mead and L. A. Conway, Eds.   Reading, MA: Addison-Wesley, 1980, ch. 7.

[10] V. Narayanan, B. A. Chappell, and B. M. Fleischer, "Static timing analysis for self resetting circuits," in *Int. Conf. Computer-Aided Design (ICCAD'96)*, Nov. 1996, pp. 119–126.

[11] C. J. Myers, "Computer-aided synthesis and verification of gate-level timed circuits," Ph.D. dissertation, Dept. of Electr. Eng., Stanford Univ., Stanford, CA, Oct. 1995.

[12] J. Cortadella, M. Kishinevsky, S. M. Burns, and K. S. Stevens, "Synthesis of asynchronous control circuits with automatically generated relative timing assumptions," in *Int. Conf. Computer-Aided Design (ICCAD'99)*, Nov. 1999, pp. 324–331.

[13] K. Y. Yun and D. L. Dill, "Automatic synthesis of extended burst-mode circuits: Part I (specification and hazard-free implementation)," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 101–117, Feb. 1999.

[14] ——, "Automatic synthesis of extended burst-mode circuits: Part II (automatic synthesis)," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 118–132, Feb. 1999.

[15] Intel Corporation, "Pentium Processor User's Manual,".

[16] R. Bryant, D. Beatty, K. Brace, K. Cho, and T. Sheffler, "COSMOS: A compiled simulator for MOS circuits," in *24th ACM/IEEE Design Automation Conf.*, 1987, pp. 9–16.

[17] M. Roncken, "Defect-oriented testability for asynchronous ICs," *Proc. IEEE*, vol. 87, pp. 363–375, Feb. 1999.

[18] P. Pal Chaudhury, D. R. Chowdhury, S. Nandi, and S. Chattopadhyay, *Additive Cellular Automata: Theory and Applications*. Piscataway, NJ: IEEE Computer Society Press, June 1997, vol. I.

[19] M. Roncken, K. Stevens, R. Pendurkar, S. Rotem, and P. Chaudhuri, "CA-BIST for asynchronous circuits: A case study on the RAPPID asynchronous instruction length decoder," in *Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, 2000, pp. 62–72.

[20] R. D. Adams, E. S. Cooley, and P. R. Hansen, "Quad DCVS dynamic logic fault modeling and testing," in *Proc. Int. Test Conf.*, 1998, pp. 356–362.

[21] J. T.-Y. Chang and E. J. McCluskey, "Detecting resistive shorts for CMOS domino circuits," in *Proc. Int. Test Conf.*, 1998, pp. 890–899.

[22] K. S. Stevens, "Practical verification and synthesis of low latency asynchronous systems," Ph.D. dissertation, University of Calgary, Calgary, Alberta, Canada, Sept. 1994.

[23] R. Ginosar and R. Kol, "Adaptive synchronization," in *Proc. Int. Conf. Computer Design (ICCD)*, Oct. 1998, pp. 188–189.

[24] A. E. Dooply and K. Y. Yun, "Optimal clocking and enhanced testability for high-performance self-resetting domino pipelines," in *20th Anniv. Conf. Advanced Research in VLSI*, Mar. 1999, pp. 200–214.

[25] W. Belluomini and C. J. Myers, "Verification of timed systems using POSETS," in *Proc. Int. Workshop Computer Aided Verification*, 1998.

[26] R. A. Thacker, W. Belluomini, and C. J. Myers, "Timed circuit synthesis using implicit methods," in *12th VLSI Design Conf.*, Jan. 1999.

[27] H. Zheng, "Specification and compilation of timed systems," M.S. thesis, University of Utah, Salt Lake City, 1998.

Shai Rotem (M'87) was born in Haifa, Israel, in 1954. He received the B.Sc. degree from the Technion—Israel Institute of Technology, Haifa, in 1980.

He has been with Intel Corporation, Hillsboro, OR, since 1980, in positions of VLSI design and architecture of data communication controllers and microprocessors, and CAD design and research in formal verification and asynchronous design. He is currently a Principal Engineer in the Mobile Processor Group's architecture team, responsible for platform architecture definition.

Mr. Rotem is a member of the IEEE Computer Society.

Ran Ginosar (M'82) received the B.Sc. degree in electrical engineering and computer engineering *summa cum laude* from the Technion—Israel Institute of Technology, Haifa, Israel, in 1978, and the Ph.D. degree in electrical engineering and computer science from Princeton University, Princeton, NJ, in 1982.

After working with AT&T Bell Laboratories for one year, he joined the Technion faculty in 1983. He was a Visiting Associate Professor with the University of Utah, Salt Lake City, in 1989 and 1990, and a Visiting Faculty Member with the Strategic CAD Lab, Intel Corporation, from 1997 to 1999. He serves as the head of the VLSI Systems Research Center at the Technion. His research interests include asynchronous systems and electronic imaging.

Kenneth S. Stevens (S'84–M'84–SM'99) received the B.A. degree in biology in 1982 and the B.S. and M.S. degrees in computer science in 1982 and 1984 from the University of Utah, Salt Lake City. He received the Ph.D. in computer science from the University of Calgary, Alberta, Canada, in 1994.

From 1984 through 1991, he held research positions at the Fairchild/Schlumberger Laboratory for AI Research, the Schlumberger Palo Alto Research Laboratory, and Hewlett Packard Laboratories, Palo Alto, CA. He became an Assistant Professor at the Air Force Institute of Technology, Dayton, OH, in 1994, and since 1996 he has been an Adjunct Professor. Since 1996, he has been with the Strategic CAD Labs, Intel Corporation, Hillsboro, OR, where he is currently a Principal CAD Engineer. His primary areas of expertise include asynchronous circuits, VLSI architecture, hardware synthesis and verification, and timing analysis. He holds seven patents and has been the principal author for three papers which received the best paper award. He has been on the Technical Program Committee for the International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC) conference series since 1998.

Peter Beerel received the B.S.E. degree in electrical engineering from Princeton University, Princeton, NJ, in 1989 and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1991 and 1994, respectively.

He joined the Department of Electrical Engineering—Systems, University of Southern California, Los Angeles, in 1994, where he is currently an Associate Professor. His research interests include a variety of topics in CAD and mixed asynchronous/synchronous VLSI design. He has been a member of the technical program committee for the International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC) since 1997 and was Program Co-Chair for ASYNC'98. He is also on the technical program committee for the International Conference on Computer-Aided Design (ICCAD'00) and the Asia Pacific Conference on ASICs (AP-ASIC'00).

Dr. Beerel was a recipient of an Outstanding Teaching Award in 1997 and the Junior Research Award in 1998, both from the School of Engineering, University of Southern California. He received a National Science Foundation Career Award and a 1995 Zumberge Fellowship. He was also co-recipient of the Charles E. Molnar award for two papers published in ASYNC'97 that best bridged theory and practice of asynchronous system design, and was a co-recipient of the best paper award in ASYNC'99.

Chris J. Myers (M'96) received the B.S. degrees in electrical engineering and Chinese history in 1991 from the California Institute of Technology, Pasadena, CA, and the M.S.E.E. and Ph.D. degrees from Stanford University, Stanford, CA, in 1993 and 1995, respectively.

He has been an Assistant Professor in the Department of Electrical Engineering, University of Utah, Salt Lake City, since 1995, where he also serves as Director for the Center for Asynchronous Circuit and System Design. His current research interests are innovative architectures for high performance and low power, algorithms for the computer-aided analysis and design of real-time concurrent systems, formal verification, and asynchronous circuit design.

Dr. Myers received a National Science Foundation (NSF) Fellowship in 1991, an NSF CAREER award in 1996, and a best paper award at the International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC) in 1999.

Kenneth Y. Yun (M'95) received the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, and the S.M. degree in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge.

He is currently an Associate Professor in the Department of Electrical and Computer Engineering, University of California, San Diego. He had held design engineering positions at TRW and Hitachi for six years. His current research interests include the design, synthesis, analysis, and verification of mixed-timed VLSI circuits and systems, in particular, interface design methodologies and tools to facilitate ultra-high-speed communications between synchronous/asynchronous modules. He has been working with Intel Corporation, Hillsboro, OR, as a Primary Consultant on the Asynchronous Instruction Decoder Project. He was a program co-chair with ASYNC'98.

Dr. Yun is the recipient of a National Science Foundation CAREER award and a Hellman Faculty Fellowship. He received the Charles E. Molnar award for a paper that best bridges theory and practice of asynchronous circuits and systems at the International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC) in 1997 and a best paper award at the International Conference on Computer Design (ICCD) in 1998.

Rakefet Kol (S'94–M'98) received the B.Sc. (*summa cum laude*), M.Sc., and D.Sc. degrees in computer engineering from the Technion — Israel Institute of Technology, Haifa, Israel, in 1985, 1989, and 1998, respectively.

From 1985 to 1994, she worked as a Computer Engineer for Rafael, Israel, developing embedded systems. In 1998, she returned to Rafael, where she worked as a System Engineer in charge of developing C4I (computer, communication, command, control, and intelligence) applications and systems. Since 1995, she has also served as an Adjunct Lecturer at the Technion—Israel Institute of Technology, Haifa. She is also currently with Zoran Corporation (in the Israeli operations), where she leads the MPEG4 technology development activity. Her current research interests include computer architectures, asynchronous design, software engineering, and video compression.

Dr. Kol is a Member of the IEEE Computer Society and the Association for Computing Machinery (ACM).

Charles Dike received the B.S.E.E. and the M.S.E.E. degrees from Brigham Young University, Provo, UT, in 1977 and 1984, respectively.

He worked as an Integrated Circuit Design Engineer for Signetics Corporation, Orem, UT, for eleven years. While there, he developed expertise in metastability theory and asynchronous arbitration. He worked in bipolar, BiCMOS and CMOS technologies. He joined Intel Corporation, Beaverton, OR, in 1992 and currently works for the Strategic CAD Labs group of the Design Technology Division on asynchronous logic and interfaces, signaling technology, and LVS methodology. He holds eleven patents and has authored and co-authored several articles.

Marly Roncken (M'99) received the M.Sc. degree in mathematics and computer science from the University of Utrecht, The Netherlands.

From 1985 to 1997, she worked at Philips Research Laboratories Eindhoven, The Netherlands, in the area of VLSI Design Automation and Test. At Philips, she was (with others) responsible for test operations and test research and development in the VLSI Programming and Silicon Compilation Project for asynchronous circuits (Tangram). In 1997, she joined Intel Corporation Strategic CAD Laboratories, Hillsboro, OR, where she investigated the testability of high-speed self-timed domino circuits in the context of the RAPPID design project. Her current focus is on design and test flows for high-speed and low-power embedded systems.