Three Generations of Asynchronous Microprocessors

Alain J. Martin, Mika Nyström, Catherine G. Wong
Department of Computer Science
California Institute of Technology
Pasadena, CA 91125

Abstract

Asynchronous VLSI offers low power, modularity, and robustness to physical variations. We describe three generations of asynchronous microprocessors designed at Caltech between 1988 and today, and the evolving circuits and design techniques associated with each of them.

I. Introduction

While the VLSI design-and-test community has been grappling with ever mounting issues of clock distribution, parameter variation, power consumption, and design complexity, a number of research groups around the world have quietly been developing techniques that can significantly alleviate these problems by eliminating the clock.

Since the mid-eighties, our research group at Caltech has been designing asynchronous chips of increasing complexity. In 1988, we designed the world's first completely asynchronous microprocessor. This "Caltech Asynchronous Microprocessor" (CAM) is a 16-bit RISC processor with 20,000 transistors. Designed as a proof of concept, the CAM worked flawlessly on first silicon, and showed excellent performance and robustness characteristics [1].

A decade later, we designed the MiniMIPS, an asynchronous version of the 32-bit MIPS R3000 microprocessor. Designed for high throughput, the MiniMIPS required a new design method based on fine-grain pipeline stages combining control and data. With 2,000,000 transistors and a throughput four times that of its synchronous counterpart ported to the same technology, the MiniMIPS remains the fastest asynchronous microprocessor ever designed. Like the CAM, it was functional on first silicon [2].

Currently, we are designing the Lutonium, an asynchronous 8051 microcontroller, specifically for energy efficiency. Using a new theory of energy complexity and new synthesis tools, we expect 0.5 nJ per instruction at 200 MIPS (in 0.18-μm CMOS at 1.8 V) [3].

This article describes these three generations of asynchronous microprocessors and the corresponding circuit families and design methods.

A. Quasi Delay-Insensitive Asynchronous Design

The asynchronous circuits we use are called quasi delay-insensitive or QDI. A QDI circuit does not use any assumption on, or knowledge of, delays in operators and wires, with only a few exceptions. The exceptions are some forks, called isochronic forks, in which the delays on the different branches of the fork are assumed to be similar. QDI circuits are the most conservative asynchronous circuits in terms of delays. But they are also the most robust to physical parameter variations because the circuits' dependence on delays is so small. This robustness makes it straightforward, for instance, to exchange energy and throughput against each other through voltage adjustments.

In order to eliminate all timing assumptions, including the clock, we encode information about a signal's validity within the signal itself. One bit of data can be communicated using a dual-rail channel that consists of two data wires (one representing '0' and the other representing '1') and a request/acknowledge wire. With this encoding, we use four-phase handshakes to synchronize communications. To communicate larger pieces of data, we usually extend the dual-rail channel to 1of$N$ channels, which consist of a request/acknowledge wire and N data wires operating in a one-hot manner.

For instance, consider the receive action $L?x$ (see sidebar for a brief description of the CHP notation), where L is a boolean input-channel with the two input data-wires $L.0$ and $L.1$ and the output acknowledge-wire $L.a$. One of several possible four-phase handshaking expansions (HSE) of this receive action is:

$$\mathit{receive} \equiv [L.0 \;\longrightarrow\; x\downarrow \;[]\; L.1 \;\longrightarrow\; x\uparrow];\; L.a\uparrow;\; [\neg L.0 \;\wedge\; \neg L.1];\; L.a\downarrow$$

Similarly, consider the send action $R!x$, where R is a boolean output-channel with the two output data-wires $R.0$ and $R.1$ and the input acknowledge-wire $R.a$. A possible HSE of this send, which is compatible with the above HSE for the receive, is:

$$\mathit{send} \equiv [\neg x \;\longrightarrow\; R.0\uparrow [] \; x \;\longrightarrow\; R.1\uparrow];\; [R.a];\; (R.0\downarrow,\; R.1\downarrow);\; [\neg R.a]$$

B. General Design Method

Although tools and circuit styles have changed over the years, the general synthesis approach has remained intact. The method is based on semantics-preserving program transformations that can be implemented either automatically or manually. Because these transformations preserve the semantics of a program, the results are correct by construction.

The first step of the method specifies the behavior of an asynchronous circuit as a sequential CHP program. The first transformation, *process decomposition*, replaces the sequential program with a set of concurrent CHP processes; the transformation can then be repeated on the new processes.

The next transformation, *handshaking expansion*, encodes data using only boolean variables (nodes in the eventual circuit), and converts every communication action into a four-phase handshake.

Finally, we compile the handshaking expansions into *production rules*. A production rule has the form $B \to x\uparrow$ or $B \to x\downarrow$, in the first case setting the node x to Vdd (*true*) when the boolean expression B (called the *guard*) is true, and in the second case, setting the node x to GND (*false*) when B is true. For example, we may express 2-input nand gate as the following *production-rule set*:

$$\begin{array}{lcl} a \wedge b & \longrightarrow & x\downarrow \\ \neg a \vee \neg b & \longrightarrow & x\uparrow \end{array}$$

All production rules in a set are executed concurrently. Since no sequencing is provided (much as no sequencing is provided by transistors!), sequencing has to be implicit and handled by the compilation procedure, which guarantees the absence of hazards by ensuring *stability* (if a guard becomes true, it remains true until its production rule is executed) and *non-interference* (the guards for $x\uparrow$ and $x\downarrow$ cannot be true at the same time).

Production-rule sets are our transistor netlists, and so our compilation procedures also seek to make them CMOS-implementable (antimonotonic—only negated literals can be used in up-going rules and only non-negated literals in down-going rules). From a CMOS-implementable production-rule set, the last step is to create physical layout. (We skip the traditional “schematic” and “netlist” steps.)

II. Caltech Asynchronous Microprocessor (1988)

The Caltech Asynchronous Microprocessor (CAM) is a 16-bit RISC with 16 general-purpose registers, four buses, an ALU, two adders, and separate instruction and data memories. Because of the processor’s tidy design, a team of five people moved from specification to tape-out in five months.

A. Traditional QDI Circuits

The CAM circuits belong to the “traditional” QDI logic family. The basis of the logic family is a buffer that inputs a value x on one channel (say, L) and outputs the result of a computation, $f(x)$, on another channel (say, R). In CHP, this buffer is expressed as $*[L?x; R!f(x)]$. The buffer may have several inputs and outputs including conditional ones.

For the sake of simplicity, assume f is the identity function. Then, the handshaking expansion for the buffer is obtained by replacing $L?x$ and $R!x$ with their handshaking expansions, for instance as follows:

$$\begin{array}{l} *[L.r\uparrow;\ [L.0 \longrightarrow x\downarrow \ [\!]\ L.1 \longrightarrow x\uparrow];\ L.r\downarrow;\ [\neg L.0 \wedge \neg L.1]; \\ \quad [R.r];\ [\neg x \longrightarrow R.0\uparrow [\!]\ x \longrightarrow R.1\uparrow];\ [\neg R.r];\ (R.0\downarrow,\ R.1\downarrow)\ ]\ . \end{array}$$

To facilitate the decomposition, we have used a different HSE from what we had in the previous section. Rather than an acknowledge signal, the handshake now uses a *request* signal: $L.r$ for the receive HSE, and $R.r$ for the send HSE.

For a one-bit datapath and a simple function, we can implement this HSE directly. But the implementation becomes unmanageable as the datapath and the complexity of the function grow. In traditional QDI design [4], the standard implementation consists of separating the control part from the datapath. In the above example, we need to introduce a pair of internal signals l and r to connect the control and the datapath as shown in figure 1(b). The HSE for the control is:

$$\mathit{control} \equiv *[L.r\uparrow;\ [l];\ L.r\downarrow;\ [\neg l\ \wedge\ R.r];\ r\uparrow;\ [\neg R.r];\ r\downarrow\ ]\ .$$

The HSE for the datapath consists of two parts, one for writing the data into register x:

$$\mathit{write}\ \ x \equiv *[[L.0\ \longrightarrow\ x\downarrow\ \talloblong\ L.1\ \longrightarrow\ x\uparrow];\ l\uparrow;\ [\neg L.0\ \wedge\ \neg L.1];\ l\downarrow]\ ,$$

and one for reading the values of x:

$$\mathit{read}\ \ x \equiv *[\ [r];\ [\neg x\ \longrightarrow\ R.0\uparrow \talloblong\ x\ \longrightarrow\ R.1\uparrow];\ [\neg r];\ (R.0\downarrow,\ R.1\downarrow)]\ .$$

The signal l is produced by the write part of the datapath to acknowledge that the boolean value presented on wires $L.0$ and $L.1$ has been written into x. Figure 1(c) shows a standard-gate implementation of a one-bit datapath as specified by the above HSE. Figure 1(d) shows how the datapath implementation is adjusted for an arbitrary number of bits and arbitrary function f. The *completion tree* sends a single-bit signal to the control: 1 when all single-bit registers have been written with the valid value presented on the register input-wires, 0 when all register input-wires are reset to the neutral value (0,0). It is usually implemented as a tree of two-input C-elements.

Simplicity and generality are the strengths of this design style, allowing quick circuit design and synthesis. However, this style also puts high lower bounds on the cycle time, forward latency, and energy per cycle. First, in the control, the inputs on L and the outputs on R are not interleaved, putting all eight synchronizing transitions in sequence. Secondly, the completion-tree delay (see figure 1(d)) is included twice in the handshake cycle between two adjacent buffer stages, and it is proportional to the logarithm of the number of bits in the datapath. Finally, the lack of interleaving in the control handshakes makes the explicit storing of the variable x in a register necessary, adding both energy and latency overhead.

B. Results

The CAM was fabricated via MOSIS in HP's 2.0-μm and 1.6-μm SCMOS and was functional on first silicon. In 1.6 μm, its peak performance was 5 MIPS at 2 V drawing 5.2 mA of current, 18 MIPS at 5 V drawing 45 mA, and 26 MIPS at 10 V drawing 105 mA. The test results were remarkable not only because of the speed reached on the first design, but also because of the surprising robustness of the chips to variations in temperature and power supply voltage (Vdd). The microprocessor runs at temperatures ranging at least

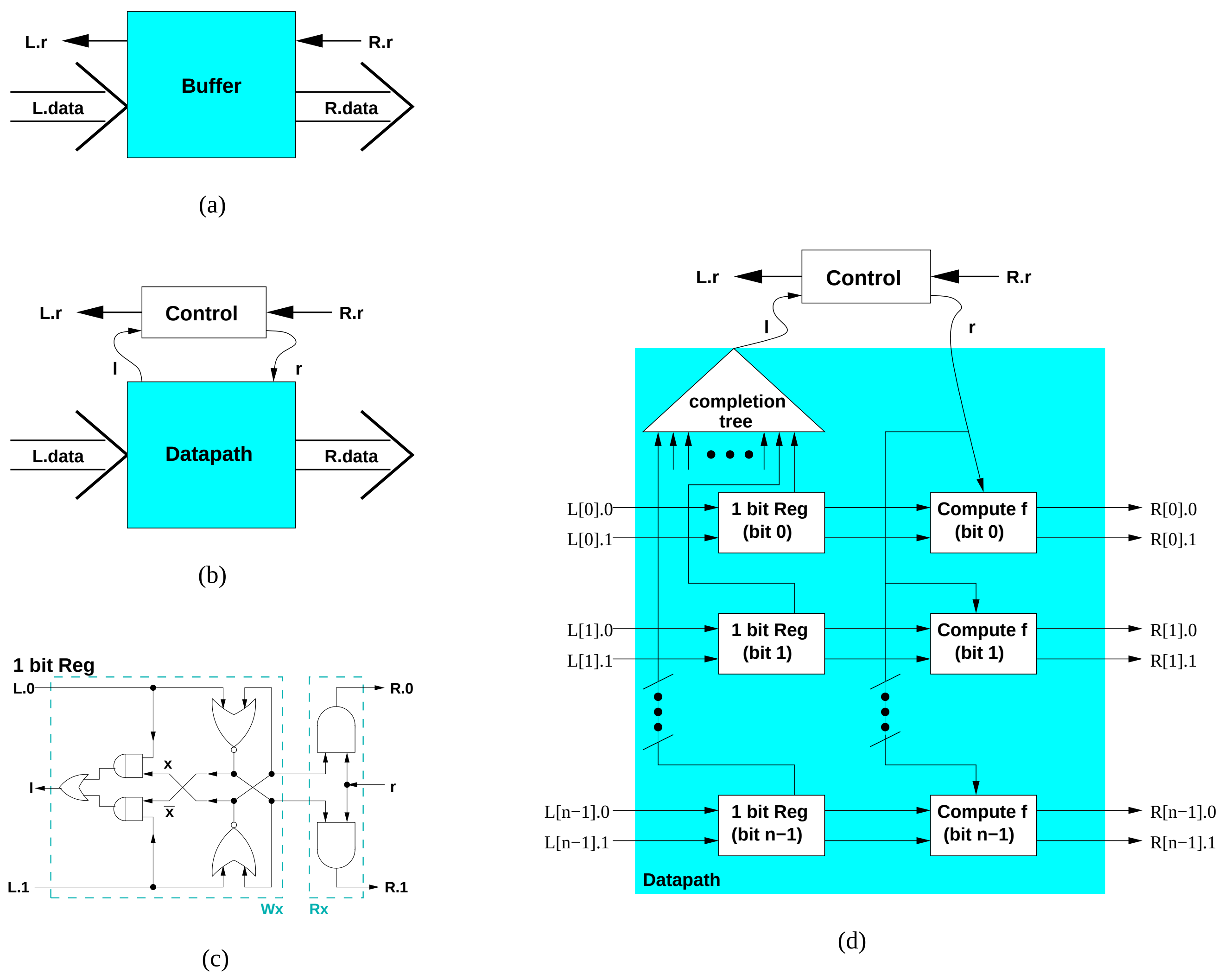


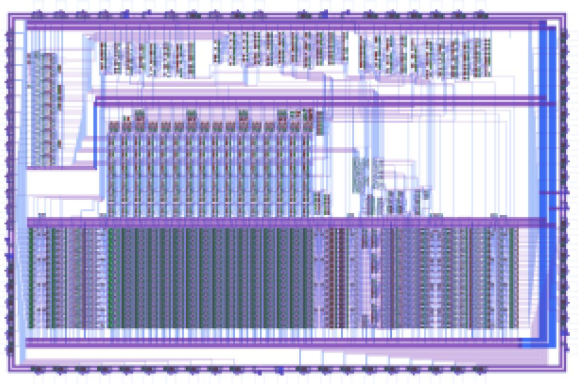
Fig. 1. A simple n-bit buffer implemented in the traditional QDI design style.

from 77 K to 373 K, and with Vdd ranging from 0.35 V (subthreshold) to more than 10 V. (Nominal voltage is 5 V.) The design was later translated into gallium arsenide (Vitesse HGaAs3), in which it ran at 100 MIPS.

III. MiniMIPS (1998)

The biggest design that we have completed to date is the Caltech MiniMIPS. Until the MiniMIPS project, there was a widespread belief that asynchronous circuits could never perform well for wide datapaths because of delays associated with completion trees. The MiniMIPS shattered this belief by introducing a new logic family that offers high throughput without sacrificing the low-power advantages of QDI.

The MiniMIPS is a 32-bit RISC CPU that implements most of the MIPS-I Instruction Set Architecture; the MiniMIPS is similar to a MIPS R3000. The MiniMIPS has two four-kilobyte on-chip caches: an instruction cache and a direct-mapped write-through data cache. Exceptions are precise. We left out the TLB (part of the address-translation mechanism) and the partial-word memory operations.


(a)

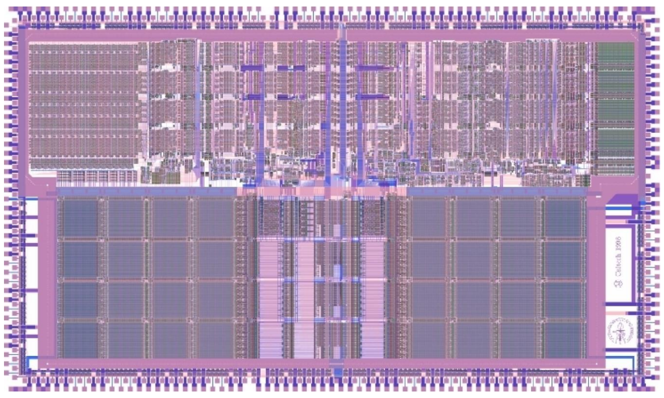

(b)

Fig. 2. Layout for (a) The Caltech asynchronous microprocessor (b) The MiniMIPS.

A. *Asynchronous Pipelined Microprocessors*

Processing an instruction consists of three steps: computing the next program counter, fetching the instruction, executing the instruction. In the MiniMIPS, the execution of an instruction can overlap with the other steps of the next instruction. The pipeline includes execution units, an instruction-decode unit, a register unit that decides when to perform bypasses, and a writeback unit that controls whether the result from an execution unit is written into the registers.

The MiniMIPS pipeline is very different from a clocked execution pipeline, where the execution units and writeback unit are aligned in some specific sequence that all instructions pass through in order. In the MiniMIPS the execution units can execute concurrently, which also allows some degree of out-of-order execution. For example, a result coming out of the adder does not go through any other execution unit or the writeback, but directly to the register unit. Queues holding information about the order in which instructions have issued and exceptions have occurred allow the writeback unit to prevent pipeline hazards.

The MiniMIPS gave us an opportunity to explore several new architectural issues in an asynchronous context, such as caches, precise exceptions, interrupts, register bypassing, and branch prediction.

B. *Fast New Logic Family*

In the MiniMIPS, we introduced a logic family based on very fine-grain pipeline stages implemented as "precharge half-buffers," or PCHBs. The PCHB is a particular implementation of the CHP buffer $*[L?x; R!f(x)]$.

Each PCHB circuit processes a datum small enough that *control and datapath do not have to be separated.* In a PCHB HSE, input and output handshakes (L and R in our example) are interleaved, "reshuffled," in a way that adds concurrency and also processes data directly from the input wires to the output wires. The PCHB reshuffling is as follows:

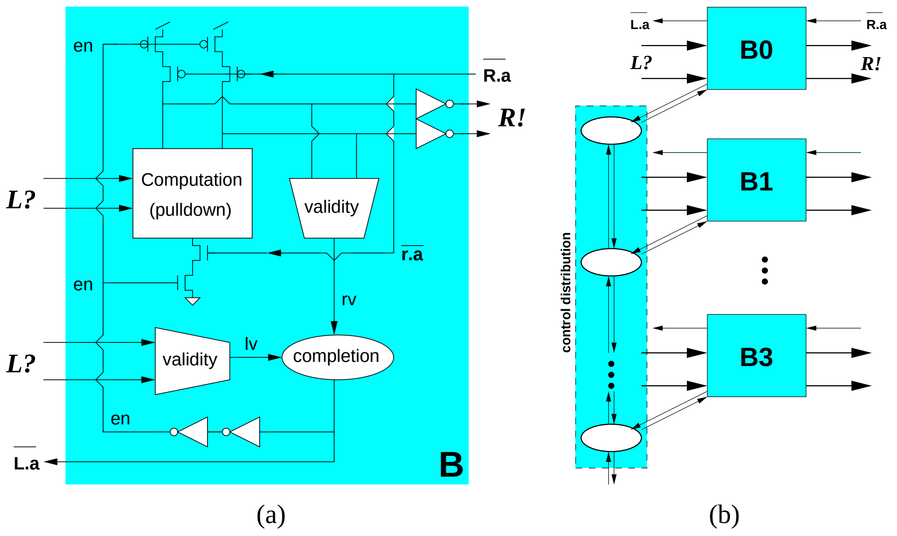


Fig. 3. PCHB circuit template.

$$\begin{aligned}&*[\ [\neg R.a];\ [L.0 \longrightarrow R.0\!\uparrow\ [\!]\ L.1 \longrightarrow R.1\!\uparrow];\ L.a\!\uparrow;\\ &\quad [R.a];\ (R.0\!\downarrow,\ R.1\!\downarrow);\ [\neg L.0\ \wedge\ \neg L.1];\ L.a\!\downarrow\ ]\end{aligned}$$

We have dispensed with the costly internal registers of traditional QDI circuits—the variable x no longer appears in the HSE.

The forward latency of the PCHB, the delay from when the data is valid on L to when the result is valid on R, is only two transitions. The basic CMOS implementation of a PCHB stage is shown in figure 3(a). Observe that all computation is done in the pulldown network with L as input.

We reduced the handshaking overhead (see section II-A), in particular the completion-tree delays, by pipelining the completion mechanism. Instead of a single control for the 32-bit datapath with a single 32-input completion tree, we instead byte-sliced the datapath. The 8-input completion trees inside the byte slices then operate in parallel with the control distribution to the byte slices, as shown in figure 3(b), removing what had been held to be a serious performance limitation on QDI asynchronous circuits.

The choice of as fine-grain a pipeline stage as the PCHB has consequences for the organization of the entire processor. Because each stage can do only a modest amount of computation, any non-trivial computation must be decomposed into a network of PCHB stages, with repercussions on latency, throughput, and energy. While the small size of a PCHB stage helps keep the cycle time low, it may also increase the global latency on computation cycles with feedback, in particular the instruction-fetch loop, which is usually a critical part of microprocessor pipelines. The relationship between a stage period p, a stage latency l, and a pipeline cycle-length n (in terms of the number of stages) is given by the equation $p = n * l$. A crucial step in the design process is choosing the individual stages and the length n of a pipeline in such a way that the equation is satisfied for p and l corresponding to the optimal cycle time and latency of each stage. In the

#	Processor	word	Tech [/μm]	Freq [/MHz]	Power per bit	Energy [/10^{-10}J]	Et^2 [/10^{-26}Js^2]
1	MiniMIPS (sim)	32	0.6	280	0.219	7.8	1.0
2	MiniMIPS (fab)	32	0.6	180	0.125	7	2.1
3	R3000 (cpu)	32	1.2	25			
4	R3000A (cpu)	32	1.0	33			
5	VR3600 (cpu+fpu)	32	0.8	40			
6	R4600	64	0.64	150	0.0719	4.8	2.1
7	21064	64	0.6	200	0.469	23.5	2.1
8	R4400	64	0.6	150	0.234	15.6	7.0
9	SH7708	16/32	0.5	60	0.018	3	8.3
10	P6	32	0.6	150	1.8	120	52

Fig. 4. Comparison of MiniMIPS performance to that of commercial processors similar in complexity and technology. (Sources: #3-5 [7], #6-10 [8].)

MiniMIPS, $p = 18$, $l = 2$, and $n = 9$.

C. Results

The MiniMIPS was fabricated in HP's 0.6-μm CMOS process via MOSIS. The chip is 8$\times$14 mm in size, with two million transistors (1.25 million in the caches). The test performance on small programs is as follows: 180 MIPS and 4 W at 3.3 V; 100 MIPS and 850 mW at 2.0 V; 60 MIPS and 220 mW at 1.5 V. The performance figures running Dhrystone are 185 MHz at 3.3 V on the pins and at room temperature, which translates to about 165 VAX MIPS.

SPICE simulations had indicated that the performance should have been 280 MIPS at 3.3 V, significantly more than we achieved in the lab. First, it appears that the HP process had been detuned over the years, which could be blamed for a 20% performance loss. The other 20% of the loss was caused by a layout error: one of the designers left a long polysilicon wire in a single unit.

Nevertheless, the performance obtained on this first prototype is excellent. It is about four times as fast as a commercial, clocked microprocessor of the same type in equivalent technology. Figure 4 shows comparisons. (The next section explains the metric Et^2.)

The graph of figure 5 offers another comparison. The solid line is the iso-Et^2 line of the MiniMIPS in 0.6 μm; the dotted line shows it adjusted for 0.35-μm technology. The points represent five different processors: the MiniMIPS designed in 0.6-μm technology is compared to the DEC Alpha 21064 and two asynchronous designs in 0.5-μm, the Titac2 and the AMULET2e. In 0.35-μm technology, the comparisons are to the DEC Alpha 21164 and the StrongARM SA110 [12]. Titac2 is an asynchronous MIPS microprocessor designed at the University of Tokyo. In 0.5-μm technology, it runs 52 MIPS at 2 W at 3.3 V [11]. AMULET2e is an asynchronous ARM designed at the University of Manchester. In 0.5-μm technology, it runs 38 MIPS at 0.15 W at 3.3 V [10].

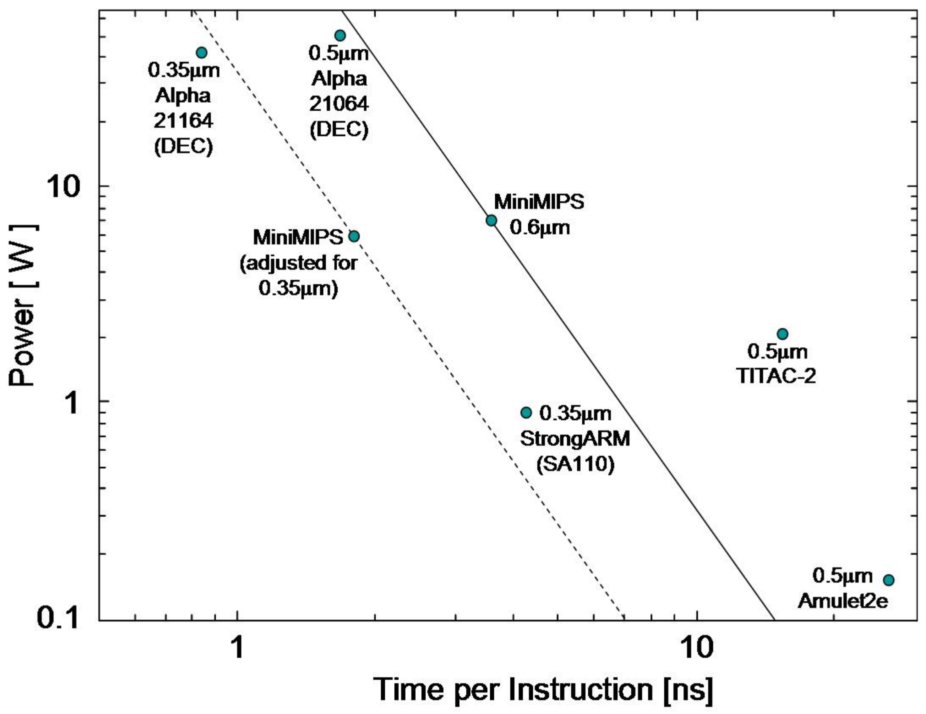


Fig. 5. Performance of other processors relative to the iso-Et^2 lines of the MIPS in 0.6- and 0.35-μm technologies

IV. Lutonium 8051 Microcontroller (2003)

Our latest project is the Lutonium, a QDI asynchronous 8051-architecture microcontroller designed for energy efficiency. Although the 8051's complex instruction set and irregular use of registers do not make it an obvious candidate for an energy-efficient design, it is the most popular microcontroller, and hence it is often found in applications where low energy consumption is important.

The 8051 microcontroller has 255 variable-length instructions, each one to three bytes long. The opcode is encoded in the first byte and the operands, if any, stored in the second and third bytes of the instruction. Instruction memory and data memory are separate. 8051 peripherals include logic ports, timers and counters, I/O ports, and an interrupt controller.

We chose an ambitious, highly concurrent design for the Lutonium. Most instructions run in a single cycle. The standard 8051implementations share a single, global bus for many different kinds of operations. For energy efficiency, our pipelined design instead uses point-to-point channels that can operate concurrently. Other new energy-efficiency features include a deep-sleep mechanism with instant wakeup.

A. Energy-Efficiency Metric: Et^2

When run at their nominal operating voltages, the CAM and the MiniMIPS both demonstrate the energy efficiency of QDI, and its great robustness to variations in physical parameters (supply voltage and temperature). The ability of QDI chips to run at very low voltages, even slightly below the transistor threshold voltage, allows the trading of energy per instruction, E, against cycle time, t, through supply-voltage adjustment.

When both energy and time must be optimized, the question arises what exactly to optimize—how important is energy, how important is time? We combine the two concerns into a single metric Et^2, which is, to first order, independent of supply voltage for CMOS circuits [6]. Because it is independent of supply voltage, we can choose between circuit designs knowing that if one design has lower Et^2 at one voltage, it will also be better at any other voltage within normal operating conditions; this also means that if the supply voltage can subsequently be adjusted, the circuit with lower Et^2 can achieve a lower cycle time than one with a higher Et^2 with the same energy consumption, or equivalently, a lower energy with the same cycle time.

We evaluated the design decisions in the Lutonium based on their prospects for improving Et^2, and we also sized the transistors to give minimal Et^2.

B. Logic Family

The main difference between the Lutonium and MiniMIPS logic families is the target cycle period p of the chip. The instruction-fetch loop of the Lutonium is usually the critical part of the microcontroller owing to the irregular 8051 instruction set. To reduce the number of pipeline stages along this critical loop, we increased the amount of work possible per pipeline stage by allowing 22 elementary transitions per internal cycle (compared to 18 in the MiniMIPS).

C. CAD Tools

In the Lutonium design, the decomposed CHP processes were rewritten using `m3-3`, a Modula-3 library for writing concurrent programs with a similar structure to CHP. Assembly-language test programs were run on this `m3-3` implementation to simulate the decomposition and check for bugs. `m3-3` allows designers to set flags and quickly switch between running the sequential undecomposed version of some unit or the concurrent system of processes of the unit's decomposition.

Part of the production rules have been automatically generated by new tools. All of the production rules have been embedded in the `m3-3` framework, and can be simulated together with the higher-level code derived from CHP. This co-simulation allowed us to design the critical fetch-loop of the microcontroller in production rules before settling on a target cycle period (22 transitions) and before writing the production rules for the rest of the chip. The physical design of the Lutonium is a mixture of full-custom, generated, and standard-cell layout.

V	MIPS	mW	pJ/in	MIPS/W
1.8	200	100.0	500	1800
1.1	100	20.7	207	4830
0.9	66	9.2	139	7200
0.8	48	4.4	92	10900
0.5	4	0.170	43	23000

(a)

Chip	MIPS	MIPS/W	Et^2 [/10^{-25}Js^2]
Lutonium-50	100	600	1.7
Philips 8051 (async)	4	444	6300
Philips 8051 (sync)	4	100	1400
Dallas DS89C420 (sync, ”ultra-high-speed”)	50	100	40

(b)

Fig. 6. Lutonium performance. (a) Performance estimates in 0.18-μm CMOS technology. (b) Comparison of the Lutonium to other 8051 designs (all in 0.5-μm CMOS) [9].

D. *Results*

The estimated Lutonium performance (0.18-μm TSMC CMOS via MOSIS, nominal voltage 1.8 V, threshold voltages 0.4–0.5 V) is summarized in figure 6(a). Our past design experience gives us confidence that the performance of the fabricated prototype will be close to our estimates. In order to compare our design to three existing implementations of the 8051 in 0.5-μm CMOS, we estimated the Lutonium performance for a hypothetical Lutonium-50 in 0.5-μm CMOS. The comparisons are shown in figure 6(b). The Lutonium outperforms its closest competitor by a factor of 25 in Et^2.

V. Conclusion

We have described three QDI asynchronous microprocessors designed at Caltech between 1988 and today. The excellent performance of the CAM and the MiniMIPS, and the expected good performance of the Lutonium are proofs that an asynchronous approach can offer a solution to the challenges faced by today’s chip designer without loss of speed and with improved energy efficiency. Those experiments also show that the QDI style of asynchronous logic does not sacrifice performance to robustness, and therefore that delay assumptions, for instance like those found in bundled-data asynchronous logic, are superfluous.

The description of the three processors also shows an important change in the design style, even though the overall approach has remained essentially unchanged. The need to reduce the length of the critical cycles in order to increase the throughput of the MiniMIPS forced us to abandon the control/data decomposition that was used in the CAM in favor of a data-driven decomposition that produces much smaller components and therefore a shorter critical cycle. In turn, this new approach to decomposition rests on the property of *slack elasticity*, which says that the correctness of the design is unaffected by a change in the slack (amount of buffering) of channels. Unfortunately, not all QDI designs are slack elastic: the CAM was not. Slack elasticity requires a top-level design style quite different from the one used in the CAM. In particular, the coding of control information is now more explicit than it needed to be in the CAM style, which used slack-zero channels to order control signals.

A further consequence of a fine-grain, slack-elastic approach is that the throughput of the pipeline is more sensitive to the amount of slack on the different paths and cycles of the decomposed design. It is therefore necessary, and also possible thanks to slack elasticity, to adjust the amount of buffering by the procedure called *slack matching.* As a result, the design of a high-performance asynchronous processor has become a complex optimization problem, further compounded, in the case of the Lutonium, by the requirement to optimize Et^2 and not only throughput. But the MiniMIPS and the Lutonium experiments have shown that the same approach can be used successfully for both high-throughput and energy-efficient designs.

Acknowledgments

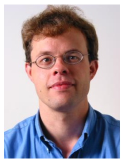
This research was sponsored by the Defense Advanced Research Projects Agency under several grants. It was successively monitored by the Office of Naval Research, the Army Research Office, and the Air Force Office of Scientific Research. We are very grateful to DARPA for their unfailing support in spite of widespread skepticism towards asynchronous logic. We are also grateful to several generations of Caltech students who made the projects successful, and to MOSIS for flawlessly fabricating our chips over the last two decades.

References

[1] Alain J. Martin, Steven M. Burns, Tak-Kwan Lee, Drazen Borkovic, and Pieter J. Hazewindus. The design of an asynchronous microprocessor. In Charles L. Seitz, ed., *ARVLSI'89*. MIT Press, 1989.
[2] Alain J. Martin, Andrew Lines, Rajit Manohar, Mika Nyström, Paul Penzes, Robert Southworth, Uri Cummings and Tak Kwan Lee. The Design of an Asynchronous MIPS R3000 Microprocessor. *ARVLSI'97*. IEEE Computer Society Press, 1997.
[3] Alain J. Martin, Mika Nyström, Karl Papadantonakis, Paul I. Penzes, Piyush Prakash, Catherine G. Wong, Jonathan Chang, Kevin S. Ko, Benjamin Lee, Elaine Ou, James Pugh, Eino-Ville Talvala, James T. Tong, and Ahmet Tura. The Lutonium: A Sub-Nanojoule Asynchronous 8051 Microcontroller. *Async'03*, 2003.
[4] Alain J. Martin. Synthesis Method for Self-timed VLSI Circuits. *ICCD 87*, 1987.
[5] Alain J. Martin. Synthesis of Asynchronous VLSI Circuits. *Formal Methods for VLSI Design*, ed. J. Staunstrup, North-Holland, 1990.
[6] Alain J. Martin, Mika Nyström, and Paul I. Pénzes. Et^2: A Metric for Time and Energy Efficiency of Computation. *Power-Aware Computing*, R. Melhem and R. Graybill, eds. Kluwer, 2002.
[7] Dileep P. Bhandakar. *Alpha Implementations and Architectures*, Digital Press, 1996.
[8] Tom Burd. "General Processor Information". April 2000.
[9] Hans van Gageldonk *et al.* An Asynchronous Low-power 80C51 Microcontroller. *Async'98*, IEEE Computer Society Press, 1998.
[10] S. B. Furber *et al.* AMULET2e: An Asynchronous Embedded Controller, *Async '97*, IEEE Computer Society Press, 1997.
[11] T. Nanya *et al.* TITAC-2: A 32-bit Scalable-Delay-Insensitive Microprocessor, *HOT Chips IX*, 1997.
[12] S. Santhanam. StrongARM 110: A 160MHz 32b 0.5W CMOS ARM processor, *HOT Chips VIII*, 1996.

BIOGRAPHIES

Alain J. Martin is a Professor of Computer Science at the California Institute of Technology (Caltech). He is a graduate of the Institut National Polytechnique de Grenoble, France. His research interests include concurrent computing and asynchronous VLSI design. He is a member of the IEEE and of the ACM.



Mika Nyström is a post-doctoral scholar in the Department of Computer Science at Caltech. He received S.B. degrees in physics and electrical engineering from MIT in 1994, and M.S. and Ph.D. degrees in computer science from Caltech in 1997 and 2001. His graduate research covered a wide range of topics relevant to asynchronous and other high-speed VLSI systems; his dissertation introduces a novel family of ultra-fast asynchronous circuits. He is a member of the IEEE.

Catherine G. Wong is a Ph.D. candidate in computer science at Caltech. She received a B.A.Sc. degree in engineering science from the University of Toronto in 1998, and an M.S. degree in computer science from Caltech in 2000. Her dissertation presents new methods for the high-level synthesis of asynchronous VLSI systems and asynchronous FPGAs. She is a student member of the IEEE and of the ACM.

SIDEBAR: CHP

Infinite repetition of a statement S is $*[S]$.

For x boolean, $x\uparrow$ and $x\downarrow$ assign the value '1' and '0' to x, respectively.

Sequential composition is denoted by the semicolon. Concurrent composition of two elementary actions is denoted by the comma.

The deterministic selection statement $[G_1 \rightarrow S_1 [] \ldots [] G_n \rightarrow S_n]$, where G_i's are boolean expressions (guards) and S_i's are program parts, is executed as: waiting until one of the guards is true, and then executing the S_i with the true guard G_i.

The notation $[G]$ is short-hand for the one-guard selection $[G \rightarrow \mathbf{skip}]$, and denotes waiting for condition G to become true.

The sequence of actions that implements a four-phase handshake is a typical alternation of waits and boolean assignments: $[B]; x\uparrow; [\neg B]; x\downarrow$. Such a sequence is called a *handshaking expansion*, or HSE.

Communication: $R!e$ means "send the value of e over channel R"; $L?x$ means "receive a value over channel L and store it in variable x".