Engineering Design of the CONVEX C2

Tom Jones

CONVEX Computer Corporation
P.O. Box 833851
Richardson TX 75083-3851

ABSTRACT

The CONVEX™ C220 and C240 supercomputers are a family of 64-bit multiprocessors, tightly coupled through a shared main memory. Each processor contains an integrated vector processor. All processor features, including the vector processor, are controlled by a microcoded instruction set. The system is implemented in ECL 100K logic, with a cycle time of 40 nanoseconds.

Two systems are available. The C220 contains two processors, I/O system, and memory in a 31" wide cabinet. The C240 contains four processors, a larger I/O system, and memory in a 57" wide cabinet. This article describes the process by which this computer family was designed, and tries to illuminate the process and rationale used to make project-related decisions.

The first section describes the structure of the C1 and C2, especially as they are different. The second section deals with the product definition process, and the third with technology selection. We regard these as extremely critical phases. Flawless execution of a program that is building the wrong thing, or one using an obsolete or impractical technology will have predictably bad results. A fourth section discusses the way in which the project was organized and staffed, a fifth section discusses the tools used, and the last section relates some of the significant crisis points during the program execution.

We certainly do not claim to have made flawless decisions. Indeed, on several occasions we courted disaster, but were able to pull back from the flame. What we would like to show is some of the real-life problems we faced, and try to relate our approach to resolving them. It is this confrontation with crisis that makes product design so challenging and rewarding. If it were just a matter of reducing logic equations, it really wouldn't be much fun.

C1/C2 Comparison

The C1 system was designed in early 1983 using TTL and CMOS technologies. The bulk of the machine was FAST MSI and SSI logic, with the vector functional units built out of 8K gate CMOS gate arrays. The original C1, later dubbed the C1-XL, had no scalar floating point hardware, but sent the operands to the vector processor.

In 1984, floating point add and multiply functional units were designed in 20K CMOS gate arrays, and the C1 was upgraded to the C1-XP. That machine, with some minor changes, was changed to the C120 with the announcement of the C2. A block diagram of the C1 is included as Figure 1.

That machine embodied our belief that we could use a relatively low-cost memory system. The memory cards are four-way interleaved, and there is a single 10 Mbyte/sec memory path. We accelerated the random access performance of the memory with a 64-Kbyte "P-cache" which was physically addressed. P-cache data is loaded in blocks of 64 bytes, and CPU accesses are held while cache misses are serviced. Most vector accesses bypass the cache, but large stride accesses were encached.

The single memory port in the C1 is demand multiplexed between the CPU and the I/O subsystem. The I/O system transfers data to and from memory via an 80 Mbyte/sec block multiplexer path called the PBUS. The CPU does not program the controllers directly, but sends messages in memory to a number of Channel Control Units (CCUs) on the PBUS.

In addition to the CCUs, the PBUS is used to attach a 68000-based service processor (SPU), which can access, via scan, about 2.200 bits of processor registers. The SPU runs UNIX™ and has a 20-Mbyte hard disk and a floppy tape drive as peripherals.

The C1 CPU executes a single instruction stream, which is accelerated into an instruction cache. The main locus of
control is the Address and Scalar Processor, which is micro-programmed using writable control store.

The Vector Processor contains three distinct micro-programmed controllers that are dispatched to execute vector instructions. Logical addresses are 30 bits, with the most significant bits signifying protection segments. Logical addresses are translated to physical addresses using an address translation cache.

There is a 1-Kbyte data cache in the CPU, which is accessed logically while the address is being translated. This cache is write-through, and is tagged on a word basis.

By contrast, the C2 was designed in 1986, using ECL 100K logic. Over 150 gate arrays were used, of 14 different types, in addition to about 3,000 packages of MSI and SSI logic. Although the vector processor data paths were implemented in 20K gate CMOS, the bulk of the gate arrays were 3K, 7K, and 10K gate ECL.

The Vector Processor data paths were implemented in 20K gate CMOS, the bulk of the gate arrays were 3K, 7K, and 10K gate ECL.

The C2 uses a more expensive and much higher performance memory system than the C1. The C2 memory is organized as pairs of 39-bit (32 data bits with 7 bits of ERCC code) memories, rather than as a single 64-bit memory. There are five discrete paths into the memory system, and each memory board has eight independent banks.

Unlike the C1, the C2's crossbar feature allows CPU and I/O functions to simultaneously access memory. The crossbar is built in gate arrays on each memory board. Besides the crossbar function, these arrays buffer the backplane, and contain stages FIFOs to and from the memory users. Each port operates at 25 MHz, or 200 Mbytes/sec. There is no analog to the P-cache.

The C2 I/O subsystem has its own memory port, which is used by a Peripheral Interface Adapter (PIA). The PIA supports a pair of PBUSes, which allows C1 CCUs, as well as new CCUs developed for the C2, to be attached. In addition to the PIA, this port is used by the Service Processor.

The Service Processor has scan access to approximately 25,000 register bits in the C2, and has a 180-Mbyte disk and a streaming cartridge tape drive as peripherals.

As in the C1, the C2 CPU interprets a single instruction stream and dispatches appropriate micro-controllers in the Address and Scalar Processor or in the Vector Processor. The hardware support in the Address and Scalar Processor is very direct. Almost all instructions execute in a single cycle, the exceptions being complex instructions like System Call.

Each CPU contains, in addition to the instruction cache and address translation cache, a 4-Kbyte data cache that is logically addressed and accessed while address translation is in progress. If a hit occurs in the data cache, the memory reference will be killed before the memory can be started. Since each processor contains its own cache, cache coherence must be enforced by a series of hardware controlled invalidation tag stores.

In addition to the shared memory path, the C2 processors share a Communication Register File, which has some interesting properties. It is organized as 8 frames of 128 registers, each register being 64 bits wide. Registers have intrinsic semaphores, allowing primitive send/receive operators to be implemented. Processes are mounted to frames, and processors dynamically allocate themselves to frames. This facilitates our implementation of dynamic processor scheduling, called Asynchronous Self-Allocating Processors, or ASAP™.

Product Definition

CONVEX began having regular debates on the definition of the C2 before the C1 prototype was even in the lab. In retrospect, our sense of timing was pretty amusing. These debates proved to be philosophical in nature, since we had no resources to commit to the project. Indeed, every scrap of human or material capital at our disposal was thrown into the C1 development activity.
This situation changed in mid-1985. The first production C1 had shipped in March, the early life sustaining activity was winding down, and the moment of decision was confronting us. Not only were we finally able to start another project, we would have to do so, or idle the engineering staff.

The years of debate had hardened the various positions, but had done little to resolve the differences. Engineering, almost to a man, wanted to build a heavily integrated CMOS implementation. CMOS technology had been developing rapidly, some of our competitors were making good use of it, and we felt that we could build a C1-class machine at much lower cost. In a sense, we were captivated by the technical possibilities.

Balanced against that was the vision of Bob Paluck, the president. He had been spending most of his time with customers and hopeful customers, not in the lab. He was convinced that the real opportunity existed at the high end, and he wanted us to build the fastest machine possible.

This highlights one of the prime examples that existing companies have in the definition of products. They have customers, who buy their products, and they also have non-customers, who looked but didn’t buy. This should be differentiated from market surveys, which might be thought to yield the same results. There is a moment of truth, when it’s time to spend real money, that conveys the customer’s real needs and biases. It still takes a lot of interpretation to extrapolate this to decisions about the next product, but at least you have something other than smoke as a start point.

Once the decision was made to increase performance, there were still some issues to be resolved. Parallel processing had become extremely important in the mind of the computer buyer. Sometimes, this was for performance reasons, but just as often, it was an issue of having the latest technology. An integrated vector machine like the C1 might well be the fastest machine on the market for a given application, but it lacked the cachet that parallel processing delivered.

We had consciously decided against parallel processing during the design of the C1. We felt that we would be able to trim months of the time to market by staying with a uniprocessor design. Not only was the hardware simpler to design, the UNIX operating system had been developed on a uniprocessor, and we felt it would port and tune more easily. Finally, we felt comfortable with vectorizing compiler technology in a uniprocessor environment, and thought we could develop it more quickly.

I think that we were pretty much right on all these scores. We shipped our first machine about nine months earlier than our parallel competitors, who started about the same time. That lead turned out to be vitally important as the industry developed.

However, as time went on, the C1 got beat on benchmarks more than once by parallel competitors. There truly were programs that could be parallelized, but not vectorized, although that was not generally the case. More often, we lost because the customer just wanted a parallel machine. When he is spending that much money, especially with a startup vendor, the customer wants evidence that he is getting the latest and greatest technology. In any case, we determined to fix both the technical and marketing problems. C2 would be a parallel machine, of one flavor or another.

There were two popular paradigms for parallel processing, shared memory and local memory. We decided for shared memory, because we believed we could build a compiler to target it while compiling existing programs. It is our belief that if you cannot inherit the existing software base, you are unlikely to succeed commercially. New applications occur at a finite rate. It is by inheriting the old ones that you can grow fast enough to stay ahead of the pack.

For the same reasons, we are believers in coupling a few fast processors, rather than many weaker ones. Programs, especially those pre-existing ones, have annoying amounts of scalar code, which neither vector nor parallel architectures have much effect on. Also, there are limits to the amount of logical parallelism in the source code. Many programs will go asymptotic in performance after only a few processors are in play. The bottom line is that a few fast processors sharing memory are easier to program than any other model, and ultimately, the architecture which is easiest to program will prevail.

Beyond parallelism, we wanted to extend the C2 performance envelope as much as possible. Several major extensions were included. Intrinsic functions, such as sine and square-root, were added to the scalar instruction set, as were vector convert instructions. The vector instruction set was nearly doubled, to allow element-by-element conditional execution of vector instructions. We all had many opinions about what was right and wrong with the C1, but early in the C2 program, we decided to sort out fact and fiction.

The project’s first C1 was modified to generate data for a PC-based performance monitor. Over several months we ran many programs, some small and some enormous, and analyzed the data. This activity allowed us to be very objective about the C1, and led to a performance increase of about 20% beyond what would be predicted by clock scaling. One of the side benefits of this approach is that it is efficient. It is amazing how much time can be wasted in philosophical arguments when the facts are missing.

The cache system came up for a major overhaul. Performance monitoring had shown that the two-level cache system in the C1 exhibited some pathological thrashing problems. In particular, the large write-back cache in the memory system performed poorly on some vector programs. Scalar code segments would accelerate data into the cache, and vector code would subsequently touch the same data. Pipelining broke down severely in these cases.

The C2 solution was to use a single-level, write-through cache in each processor, with a hardware invalidation system between processors. Write-through is not a very popular system these days, because of the memory bandwidth requirements. However, vector processors require extreme memory bandwidth already, so write-through is easy. The
The cache size turned out to be a trade-off between latency and hit-rate.

We could have improved the hit-rate at the cost of doubling the hit latency, from one clock to two. We went with latency, because our intent with large programs, which could benefit from the larger cache, was to vectorize. In addition, all vector references (32 and 64 bit of any stride) reference main memory directly, bypassing the cache. With the memory system structured as two independent 32-bit paths, two 32-bit vector accesses can be executed simultaneously.

Even with these enhancements, compatibility with our existing C1 product was a must. We were making large investments, and so were our customers, in the development of software packages tuned for our architecture. Some of the large third-party applications, like MSC/NASTRAN™ and ANSYS™, take months to validate, even if they port without difficulty. CONVEX simply did not have the manpower to undergo a major struggle to get back to where we had been with the C1. We had to make it trivial to validate C1 packages on the C2, and strict binary compatibility goes about as far as possible in that direction.

We were aware that most computer systems users are more interested in throughput than in performance on a single program. This may seem illogical, since computers are most frequently bought on the basis of benchmarking a single program, or set of programs. However, once the machine is delivered, and a user community climbs aboard, sustainable throughput and robustness become paramount.

What this implied to us was that we needed to develop a system which was competent in both the benchmark and the time-share environment. This may seem trivial, but I don't think that it is. This particularly impacts the mechanisms used to achieve parallelism, and led us to a significant innovation in that area.

Finally, we decided to save as much of the existing C1 system as possible, so that we didn't re-design anything we didn't have to. The truth is that we were strapped for resources, and needed to save effort anywhere that we could.

We chose an I/O system design that would let us use the C1 I/O system until it could be replaced by a higher performance system. There was some debate about this, because the large increase in CPU performance would seem to mitigate a proportional increase in I/O performance. But the C1 I/O system had more performance than the C1 could use, and it was robust and proven. We felt we would have more than enough problems with the subsystems that absolutely had to be re-designed, and didn't need any unnecessary exposure.

Although it was not originally planned, we even used the C1 industrial design. When we picked up the C1 I/O system, the easiest thing to do was to use the C1 peripheral cabinets. So, we just designed the C2 cabinets to match, rather than doing the whole thing over. This had two great effects, above and beyond saving money. First, we were able to upgrade C1's in the field by unbolting the C1 processor cabinet and bolting up a new C2 cabinet.

Second, it avoided the whole debate about industrial styling, which was a nearly endless argument during the C1 program. Only a few people in the company may know enough to critique an architecture, but everyone and his uncle feel competent to judge a styling look. The opportunity occurred to avoid the whole issue, and we took it.

### Technology Selection

One of our objectives was to make arbitrary programs run three times faster on the C2 than on the C1. The only way that we knew how to do this with assurance was by cranking the clock cycle down, without increasing the pipeline depth.

The C1 had a clock cycle of 100 nanoseconds, which allowed us to use FAST (TTL) and CMOS technology. We could have gone to a faster clock cycle in the same technology, by using more pipelining, but it was far from obvious to us that this would take us to the desired performance level, particularly on a broad range of applications. We felt more confident, using faster technology to scale something resembling a C1 processor architecture.

Our initial target for the C2 was 33 nanoseconds, and this made ECL the prime contender.

We knew of competitors who were aiming at those cycle times and better with CMOS, but it didn't look like a viable approach to us. Even though gate speeds on-chip can be very fast, off-chip drivers tend to be slow. Our experience was that pipelined structures such as the vector processor could exploit CMOS because many levels of pipelining could be done on a single chip. The trouble started in more random structures like the scalar processor, and in transmission line environments like the backplane. We ended up with a hybrid approach.

We used CMOS anywhere we could get away with it, and ECL wherever needed. This eventually meant that the vector processor data paths were built from 20K CMOS gate arrays, and everything else was ECL.

Choosing ECL just moved us to the next decision. There were two real contenders in the ECL world, 10KH and 100K, both claiming to be better than the other. 10KH was newer, and becoming popular. We thought it would be cheaper, due to higher volumes, and it used a smaller, "slim-line" package.

100K, on the other hand, had been around for years. It offered both power supply and temperature compensation on voltage thresholds, which had a lot of appeal in a big system. I'm not sure that these features would matter in a small system, such as a workstation, but they were certainly attractive in a system with many large boards.
There is a substantial temperature differential across the board, as well as a voltage differential from front to back.

Finally, the 100K power pins were in the middle of the package, which reduced lead inductance on them, and we expected that to improve noise margins. When everything was considered, all the little things added up, and we went with 100K.

Our original orientation was toward mounting gull-wing, flat packages on double-sided, surface-mount PC boards. We were very concerned that the faster gate speeds of ECL would be diluted by the etch delays, and we wanted to cram the ICs in as tight as possible.

Arrayed against that goal were a host of practical considerations. One was that CONVEX had a well established multi-wire board technology that we were quite comfortable with. Multiwire has big advantages in terms of low tooling cost and fast prototyping and production startup. The DIP through-hole approach in multiwire technology was also well developed, and quite reliable. In the end, it didn't seem to us that the speed advantages of denser packaging were worth the risk and cost required to develop it.

One of the big risks we took was using PAL logic. Although it is widely available in the TTL world, it was only available on data sheets in ECL 100K when we were doing the design.

There were three vendors working on the part. Only one vendor had delivered parts to us to date. One vendor canceled the program, and the other has slipped badly. We spent a lot of time worrying about this part, because if it had flopped, we would have had to re-design the product, severely compromising our cost and performance goals.

We used many hundreds of these parts per machine, so we were really dependent. ECL 10K PALS were available, and prototype 100K parts that more or less worked had been sampled, and we felt pretty sure that eventually one of the three would get there. The real question was whether or not they would get there by the time we were ready to power on the system, and as it turned out, it was real close.

The gate-array technology we used was also available only on data sheets. There were quite a few vendors who claimed that they would be able to deliver the technology we needed when we needed it. The problem was to pick the one, if any, that really could.

In the end, we stayed with the vendor of the C1 gate arrays. Their proposed technology was equivalent to everyone else's, and they had played straight with us for years. Furthermore, when things had gone wrong in the past, they had bent over backwards to support us. When you are treading off into unexplored territory, it's easier to do with people you know and trust.

The thing that everyone worries about with ECL is the power dissipation. Getting rid of the power was not so difficult. Getting it into the chips turned out to be the tough part.

It started with the backplane, something that logic designers just take for granted. The C1 backplane, although large, had never really been a problem, because the thickness was tolerable. We had multiple C1 backplane suppliers right from the start, because the technology was just not that tough. The C2 backplane, though, was a horse of a different color.

We expected routing difficulties, because the wiring density was higher than on the C1, and we were right on that score. The real problem came, though, from the power and ground distribution. It took an awful lot of copper to hold the voltage differentials down on a fully loaded system. The backplane got thick, over 0.3 inches, and this gave the backplane fabricators fits. The wiring density required small vias, and the thickness forced us into aspect ratios on the vias that make the boards tough to make reliably. Although this process is pretty well in hand today, it was touch and go for a long time. There is something grimly amusing about this, that we were in such peril from something that we took for granted.

We also used a large power distribution plane to get the power from the supplies to the backplane, and that turned out to also be very difficult to manufacture. The interesting thing in both cases was that, once the problems were well understood, we were able to execute new designs that overcome the problems. The real problem was that neither we, nor the vendors, was able to correctly anticipate all the problems that we would incur in pushing the commercial state of the art. It emphasizes again the need to select vendors, at least partially, on their willingness and ability to work with you when things go awry.

The backplane was only the most down-stream of the power problems. The C2 uses 360 amp DC power supplies, and those were only a spec sheet at the time that we designed them in. We were able to get two vendors on this item, and that turned out to be handy. In the end, our primary supply fell through because of thermal problems, and the secondary came through late, but fully operational. Having a real second source can make life so much easier, which is why the purchasing people are so adamant about it. However, if you are really pushing the state of the art, it is only occasionally possible.

Another item that logic designers take for granted, connectors, also caused us a lot of worry. We had used 96-pin DIN connectors on the C1. They were cheap, there were lots of vendors, and they were very reliable. Our only real problem was sorting out the quality vendors from the also-rans.

C2 needed a lot more pins, almost twice as many. We also needed to pass a lot more current, and consequently, we ended selecting a high density connector with special power tabs on the sides of the connector body. Once again, the problem was that they existed only on paper.

Fortunately, there were two vendors, but their parts were not interchangeable. This turned out to be another case of the favored horse falling by the wayside, with a rather odd twist at the end. Eventually, we went with the one who
could deliver parts, only to find out that the vendor who
lost held a patent that our vendor was infringing. Happily,
license agreements were finally made, but things got pretty
exciting for awhile.

Looking back at the program, I guess that we exercised a
lot of good engineering judgment, and our luck was out-
standing. There were so many things that could have shut
us down, and we dodged bullet after bullet.

Vendor selection is much more important than many
people think. Finding vendors who will work with you, on
your time schedule, when the best-laid plans go badly
awry, is vital. We look at vendors in the same way as we
look at internal people. If you wouldn't hire them direct, do
you really want them holding your destiny under inde-
pendent management?

One of the best ways to find good vendors, of course, is to
look at the ones you have for existing products, but this
only works if you already have similar existing products.
When you are just starting up, or making big technology
leaps, it's a lot more dangerous.

**Staffing and Organization**

You have to have good ideas to make good products, but
the right idea by no means assures success. Execution is
perhaps even more important, and execution is a function
of your people. Staffing and organizing the project thus be-
came an overwhelming concern from the beginning.

Organization within CONVEX was a delicate issue. One
school of thought was to run C2 development out of the ex-
isting engineering organization, so as to have ready access
to the skills and experience of the C1 staff. The other, and
prevailing, thought was that engineering was dominated by
C1 support, sales, and enhancements, and that only a sep-
brate organization would be so focused as to produce the
desired results.

We ran an experiment of sorts, trying for a while to run it
out of the engineering group, but it really didn't work. We
were just too easily distracted by the daily C1 crises to con-
centrate on the new product.

We decided to create an autonomous unit, and dramati-
cally so. We moved into an unoccupied area in the back of the
CONVEX facility, equipped to be as self-sustaining as
possible. We had our own entrance, our own computers,
even our own refrigerator and microwave oven. It was a
company within a company, oriented completely around
the development of the C2. There was essentially no struc-
ture to support the finer points of career development, such
as a visible future.

We kept to ourselves, as you might expect, which facil-
titated the secrecy that we operated behind. We had made a
decision that we would just as soon not have the project
plans appear in notes.general on the UNIX network, so we
restricted access to all C2 information. We even built a
wall between us and the rest of the building, so that there
was only one way in and out.

All this may have facilitated a feeling of joint purpose on
the part of the C2 group, but it created a lot of hard feeling
within the rest of the company. I once heard a rumor that
another organization's staff meeting had applauded when
they heard that we had slipped. The story may or may not
have been true, but it was hardly surprising. Our inner-fo-
cused elitist approach was deeply offensive to many of
those who were left outside.

Our decision process was democracy backed up by dictator-
ship. There was really no formal channel for conflict resolu-
tion. Most decisions were hashed out between the involved
designers, and if they couldn't agree, an informal group of
senior designers usually could. If they were deadlocked,
one of the managers would decide. The real quality of the
process was that it was fast and effective. We didn't spend
a lot of time fretting about optimal solutions. The clock was
ticking, and we all could hear it.

Despite the fact that we had taken a core set of designers
from the C1 group, we were at only fractional strength,
and bringing on the required staff was an entirely non-
trivial task. Richardson, Texas, is not the center of the
computer universe, at least not yet.

There are, however, some defense contractors who build
complex logic systems, and we hired some excellent and
restless engineers from there. We were also nicely in phase
with the university hiring cycle, and I was extremely
pleased with our success there. We were able to hire a few
experienced computer designers from other geographical
areas.

Perhaps, surprisingly, the CPU designers were not the
hardest positions to staff. Finding really good I/O designers
seemed all but impossible, although we did fine in the end.
Diagnostic engineers are another tough category, as are
tech writers. I think that we, as an industry, do an inade-
quate job of educating and motivating engineers for these
specialties, and that it undermines the quality of our pro-
ducts relative to what they could be.

It took us almost nine months from the time we started the
project until we were fully staffed. Sometimes it seemed
that all we did was recruit, but in reality, the project was
roaring along all that time.

The architecture and support structures were well-enough
defined so that people were able to come in and start pro-
ducing immediately. We could have staffed up sooner by
lowering our standards, but we were never tempted to do
that. When push comes to shove, it's the people who either
win or lose, and you have to keep searching for the win-
ners.

The C2 has been a smashing success, and the people who
made it happen are much in demand in a fast-growing
company like CONVEX. We certainly didn't lay people off
as they finished. They were either reassigned to hot spots
on the C2 project, or assigned to new projects. As the pro-
duct was introduced into manufacturing, the C2 project or-
ganization dissolved. Happily, the resentment of the
separateness also dissolved.

In retrospect, we didn't do everything right, but I think
that the fundamental strategy was sound. Putting a very large chunk of a corporation's resources into an independent project really rankles the departmental organizations, and that's a cost that has to be considered. But it facilitates a level of focus that allows time competition with startups, and it energizes the project with commitment and makes it a lot of fun.

I feel very fortunate to have been involved with the project, that it was a unique opportunity in my life. I hope that the other people that worked on it feel as good about it as I do.

**Tools**

We were convinced that we could not design the C2 with the C1 tool set. Although we had automated schematic capture tools during the C1 project, our computing facility was a single VAX™ 11/780. This machine ran UNIX during the day, when it served as the software development and general timesharing machine. At 8 P.M., VMS™ was loaded, and the night would be spent grinding away at logic simulation and timing verification on the gate arrays.

This technique was sufficient to allow the C1 gate arrays to work the first time. The board design, by comparison, had been designed and wire-wrapped in five months, and debugged for the next year. The board designers wanted to duplicate the C1 ASIC success on the C2 boards.

To that end, we decided to simulate the entire design, first at the behavioral level, then at the gate level. The topology of our simulation approach is shown in Figure 4. Because we expected to burn most of our cycles doing behavioral simulation, we selected the fastest behavioral simulator available. Unfortunately, it had no gate-level simulation, so we had to build our own gate-level models.

This solution was not real elegant, but it worked. No vendor at that time had a complete tool set that would satisfy our requirements, so we took a bit of this, and a bit of that, and glued the pieces together with a lot of C code to build the CAD system that we wanted. I'm not convinced that this situation has changed, although it would be very nice if it had.

The software simulations also got upgraded going from C1 to C2. C1 development had used an instruction-set architecture simulator to develop and test the software for that product. It had been rather slow, however, so the C2 developers sped it up as a first priority. This tool was instrumental in the overall development cycle, because it let us go at hardware debug with fully functional software. We could run the operating system on this simulator, and as a consequence, the operating system came up on real hardware in a matter of days.

Tools is an area that will assume continuously increasing performance in the future. Although all computer designers like to think that it is architecture and design that decide the outcome, I disagree. I think our business is fundamentally driven by semiconductor technology, and the guy who does the best and fastest job of translating new semiconductor technology into new products has a tremendous

![Figure 4—CONVEX C2 Simulation Topology](image-url)
edge. You still have to have superior design talent, but having the design technology to back it up has gone from a convenience to a necessity.

Our efforts in this area may seem like an obvious requirement, but they weren't. After all, for a resource constrained company, every CAD designer meant fewer logic designers. From a resource point of view, it was a zero-sum game, and the issue was getting the maximum bang on what was not a lot of bucks. This is a demonstration of a project organization at work, because the project manager is forced to trade off money spent in the various disciplines.

Execution

It is necessary to plan for success, but the best-laid plans tend to go awry when the project encounters harsh reality. Success at that juncture may well depend on your ability to improvise. It was when everything was going wrong, that I most appreciated our project orientation. Engineers were doing all manner of things that was not within their "job definition" in order to somehow solve the crisis of the day and keep the project moving.

By May of 1986, the key architectural decisions had been made, the critical mass of engineering manpower had been established, and the tools were working pretty well. "Pretty well" is the operative phrase, because active use was exposing all the flaws. Some of the logic and diagnostic designers, as well as the CAD engineers, were improvising furiously in the CAD area. It all came together in the fall, though, and by the end of the year, a full behavioral simulation was in final debug.

Even as we were finishing the behavioral simulation, we were also doing gate-level design and preliminary timing, and one thing that was very clear was that 33 nanoseconds was a pipe dream. Forty nanoseconds was much more reasonable, with our margin requirements. Actually, timing verification was done to 34 nanoseconds, manufacturing normally tests at 36 nanoseconds, and the machine operates in the field at 40 nanoseconds.

We fixed these timing parameters early enough in the gate design stage so that all the boards and gate arrays could be designed to them, and they turned out to be pretty good numbers. They weren't really easy to make, nor did we thrash trying to achieve them, so they must have been about right.

Then came the first great partitioning. We had been running behavioral simulations, splicing in gate level designs to prove them. The day finally arrived when we had to put all the gates on boards, and of course, there were not enough square inches or connector pins to support them. It was back to the design shop.

Several passes later, the design fit and worked, and the next problem emerged. The C240 machine got postponed at this stage, though. To fit the cache coherency logic for four heads would require two additional gate arrays, and it was too late. So the mechanical design was directed toward a two-headed machine. We also decided at this point that the first machines to ship would be single-headed, running the C1's software.

This was an extremely important decision, because it meant that the first thing out the door would be almost entirely a hardware project, with minimal software dependencies. The second item, the two-headed machine, was almost entirely a software project since it was new parallel software running on the same hardware platform as the single-headed machine. This made the hardware resources available to finish the four-headed machine, which was entirely a hardware project since it ran, unmodified, the previously released parallel software.

This was the formal recognition of something that the management knew very well, that the C1 was aging rapidly. There would surely come a day when we could not meet the company's sales objectives with the C1 product.

In a large company, with many products, delays in availability of a single product will not be fatal. Not so in a small, single product company living in an extremely competitive marketplace. We were making tactical decisions to minimize the risk of getting something that would replace the C1 as soon as possible, even if it delayed (it did) the availability of the real design point.

In our naivete, we had assumed that routing the boards would be a major problem. We had designed our chip placement strategy to maximize the routability. In fact, the issue that drove chip placement was circuit delay. Timing verification of the partitioned design showed massive violations, and the partitioning began its next iterations, as we headed toward a design which worked, fit, and had the required timing margins when partitioned. At times, we thought we had the classical definition of the over-constrained problem on our hands, as the days and weeks went by.

The marketplace had continued to develop, of course. C1 was getting older by the minute, and the startups continued to come forth. The program had been running for 18 months, and still there was no prototype. To put it mildly, management was concerned. But we were out of compromises and there really was nothing to do but finish what we had going.

Finally, the logjam broke. Boards were shipped to the vendor, one after another. Our multiwire routing tools were well developed, and once a board passed timing verification, we could route and ship it in less than a week. Gate arrays, PALs, and other components were stockpiled in quantity.

In late July of 1987, the first prototype system was powered on, and we began to discover the differences between the simulated interfaces and the real interfaces to the service processor. For several weeks we ground along, getting initialization and scan sequences right. Then the first diagnostic ran. Six weeks later, we were trying to boot the operating system. In another month, we were running benchmarks.

There was still a lot of work to do, but we were now on a roll. Manufacturing was building machines, and several hundred benchmark codes would be run in the next six weeks. The beta machines shipped at Christmas, and pro-
duction shipments started the next quarter. This is where the C1 compatibility strategy really won. With very few exceptions, C1 applications ran unmodified on the C2.

From project start to beta ship had taken just 27 months, and 20+ production units had shipped the next quarter. This allowed CONVEX to intercept the faltering C1 sales with the C2, propelling us on a new wave of growth. And, the design methodology allowed engineering to release to manufacturing and get on with the C3 project!