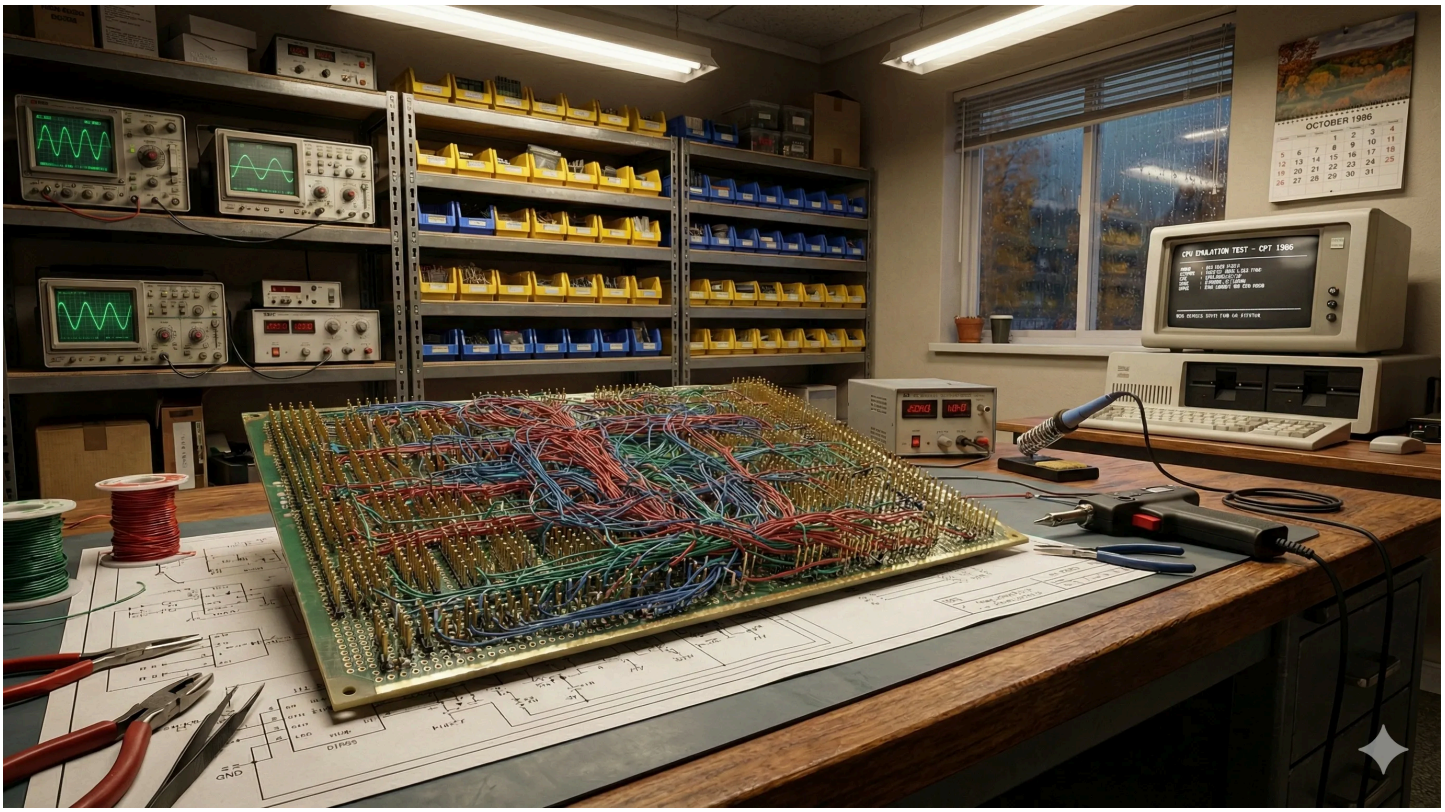
When I Built a CPU

A single project that taught me about ambition, complexity, and value.

8 min read · 2 hours ago

Gunnar Östberg

Listen Share ••• More



CPU build.

In September 1986, I learned what ambition really costs.

I was 24, newly graduated as an engineer. I had been assigned a small but critical sub-project inside a much larger industrial development effort.

The task was simple to describe and difficult to execute: build a dedicated processor from scratch, program it, and make it work in real time inside a prototype machine that did not yet exist.

Ambition

At the time, I was full of confidence. I had designed analog and digital hardware, written software professionally, and felt like a fully fledged hardware-and-software engineer. Looking back, I clearly overestimated how much I knew. But at that moment, I felt ready for anything.

That summer, I accepted a three-month IAESTE internship in Switzerland. It meant postponing the final writing of my master's thesis, but the opportunity felt worth it. I traveled by train from Sweden to the German-speaking part of Switzerland to join a large, high-tech company developing textile machinery.

I was initially placed in the IT department and asked to investigate how to implement a local area network. LANs were a new technology then. Still, I found the assignment uninspiring and asked whether there were any ongoing development projects in computer vision or image processing. To my surprise, there were. Within a week, I was transferred to a department developing a prototype optical inspection system for woven fabric.

There, I was given my real assignment.

I was asked to build a discrete CPU independently, write the program for it, and integrate it into the system so it would work. The processor would serve as a digital signal processor in a four-camera vision system that inspected fabric rolls in real time. The system would detect defects, classify them into 17 categories, and mark their exact location at a speed roughly fifteen times faster than a human inspector.

The fabric moved fast. It also drifted sideways. My processor was responsible for calculating real-time compensation for that movement, based on input from an additional camera, and feeding correction signals to the rest of the system. I would solve that problem using a combination of hardware and software. I had just under three months.

From early hardware sketches on paper and a rough algorithmic idea, the goal was to deliver a fully functional, purpose-built CPU, programmed and tested for its specific task. In practice, that meant designing, building, programming, and validating a unique processor for two-dimensional real-time signal processing in roughly eleven to twelve man-weeks.

I was ambitious. And the system worked.

Only much later did I understand how close I was to the peak of what is sometimes called the Dunning–Kruger curve. In reality, I knew far less than I thought. What I did have was drive, confidence, and a willingness to learn fast. I also had a very capable supervisor who provided subtle guidance without ever taking over.

That combination mattered.

Looking back, I see how essential that level of ambition was. Without it, I would never have dared to take on the task.

My takeaway and story message #1:

> ***Be confident. Be ambitious. Be courageous.***
> ***Don't wait until you think you know everything. You never will. If you are capable and willing to learn, step forward. That is how real growth happens.***

Confidence gets you started.
Reality decides how far you get.

Once the assignment was clear, the nature of the problem became impossible to ignore. This was not about writing clever code or assembling standard components. The processor had to operate under strict real-time constraints, process video-derived data continuously, and interact reliably with a larger system that itself was still under development.

There was no margin for delay. No opportunity to "optimize later." Every clock cycle mattered.

At that point, ambition stopped being an attitude and became a design constraint. Whatever I built had to be fast enough, predictable enough, and simple enough to be trusted inside an industrial machine that would eventually run day and night in a factory environment.

That was when I began to understand what the task really demanded.

Not just intelligence or effort.
But careful engineering, ruthless prioritization, and a willingness to trade generality for performance.

That is where complexity entered the picture.

Complexity

During the fall of 1986, I built a CPU from discrete components and programmed it directly in 48-bit microcode. There were no digital signal processors available as off-the-shelf chips at the time, and a standard CPU running machine code or assembler would have been far too slow for our application.

A general-purpose CPU is like a backhoe loader.

Image: a backhoe loader.

It is versatile, reliable, and good enough for a wide range of tasks. But it is not particularly fast at any one of them.

What we needed was something closer to a Formula racing car.

Image: a Formula racing car.

Extremely fast, highly optimized, and utterly useless for anything outside its narrow purpose.

That was the trade-off.
We deliberately gave up generality in exchange for speed.

Our performance requirements were simply in another league.

The processor had to analyze video data in real time and calculate control signals fast enough to compensate for both high fabric speed and lateral movement. To achieve this, the design had to be purpose-built. Part of the algorithm was implemented directly in hardware, and the rest in microcode, resulting in a hybrid solution optimized for a single, specific task.

From a design perspective, the CPU was constructed using bit-slice technology for the arithmetic logic unit and control unit, combined with memory and discrete logic. The design choices were largely unconstrained. The bit-slice components provided only partial functionality, so the overall architecture had to be defined from scratch. In that sense, this was not an adaptation of an existing processor model. It was a unique CPU, explicitly designed for this application.

Physically, the build was almost deceptively simple. I wire-wrapped roughly a hundred components on a breadboard, connecting thousands of wire points by hand. The tools required were basic: a wire-wrap gun, a holder, wire cutters, and a lot of patience and precision. The intellectual effort, however, was anything but simple. Every signal, timing assumption, and control bit had to be correct. There was no abstraction layer to hide behind.

Once the hardware was stable, the processor had to be programmed. The microcode consisted of a few hundred 48-bit words. There was no interpreter. When power was turned on, the CPU executed only one program. That program turned a general-purpose design into a highly specialized, extremely fast signal-processing engine.

My processor was complicated.
The system it was part of was something else entirely.

Together, the project team developed a prototype of a computer system the size of a house. It consisted of multiple processor boards, specialized electronics, high-speed buses, and several cameras. The bus structures alone were a challenge due to the timing and throughput requirements. And this was just the visible part of the system.

On top of the hardware sat advanced computer-vision algorithms for defect detection and classification, implemented in state-of-the-art software. I know how advanced they were because I had previously worked with researchers in image processing and computer vision. At the time, this was among the most complex engineering systems I had ever encountered, apart from projects like CERN or experimental fusion reactors.

What struck me most was not just the level of complication, but the level of complexity. This machine did not exist in isolation. It had to work reliably in a factory environment, as part of an industrial process at the customer, and as part of a production and service process at the supplier. Speed was only one constraint. Reliability, integration, and manufacturability were equally unforgiving.

The only way to make this succeed was through careful engineering, clear system boundaries, and a strict decomposition into well-defined subsystems and sub-projects.

That project taught me how to handle complexity. Since then, I have never feared it. On the contrary, I have learned to seek it out.

My takeaway and story message #2:

> ***Don't be afraid of complexity. Learn to recognize it, structure it, and master it. That is where real growth happens.***

Value

The CPU project taught me how technical value is created. It also taught me something about how value is recognized.

I negotiated a 175% salary increase for the same job.

I was paid 10 Swiss francs per hour, which was quite good. Since I came from one of three countries, the USA, Sweden, and Egypt, and was registered as an intern, I did not have to pay taxes under Swiss tax rules at the time. My accommodation and food costs were low, so I was able to save most of my earnings.

After about five weeks, I requested a meeting with the company's chief human resources officer. I explained that I was about to graduate as an engineer in Sweden, equivalent to a Swiss *Diplomingenieur* (ETH level), and that I was doing the same work as experienced engineers and was critical to the project's success. I showed my grades and provided the names of my managers. I also asked for a higher salary, even though I knew all interns were paid 10 francs per hour.

The HR director gave no clear answer. When I left, I thought, *Guter Versuch*. At least I had tried. Then I forgot about it.

Three weeks later, I was summoned to my department manager. He thanked me for my work and engagement so far, and showed me a letter from the division manager. It stated that my new salary would be 27.50 francs per hour, retroactive from my start date.

If I hadn't asked, I would never have been able to save enough money to buy an almost-new Volvo 244 GLT when I returned to Sweden.

Today, I don't know where that courage came from.

The company sold the product I helped develop until at least 2007.

My takeaway and story message #3:

> *Know your value. If you don't articulate it yourself, the system rarely does it for you.*

That project stayed with me long after I left Switzerland.

It taught me what ambition enables, what complexity demands, and how value is ultimately recognized.

I have seen the same patterns repeat in large systems ever since. Different technology. Same principles.

Engineering | Software Engineering | Electronics | Systems Thinking | Personal Development

Edit profile

Written by Gunnar Östberg

180 followers · 130 following

CEO Business Clarity. Management Consulting, Product Management, Business Analysis, Business Architecture, Information Systems, Information Technology.

No responses yet