From Procedural to Functional Approach in Hardware Evolution

Gheorghe M. Ştefan

Abstract

Any technological evolution is accompanied by transformations in the three fundamental modes of behavior of the human mind: *logos*, *pathos*, and *ethos*. In the present paper we analyze this triad in relation to the hardware revolution, understood not only as a sequence of semiconductor achievements, but also as a transformation in the way technical reality is imagined, specified, designed, and socially integrated. The main thesis is that hardware evolution has moved from mechanical procedure, to Boolean structure, to programmable structure, and now toward trainable and functionally specified structures. This transition is reinforced by the mutual interaction between semiconductor technologies and artificially intelligent design tools. Hardware is no longer only the passive substrate on which software is executed; it becomes an active participant in the functional organization of computation. At the same time, artificial intelligence changes the meaning of design itself: it shifts attention from explicit procedural descriptions toward families of functions, architectures, constraints, and learned transformations. We argue that this transition creates a desynchronization between the exponential dynamics of technological capability and the much slower dynamics of ethical, cultural, and spiritual assimilation. The architectural approach proposed here attempts to mediate between these levels by treating architectures as structured families of functions capable of connecting formal semantics, operational meanings, and deeper human meanings.

1 Introduction

The synchronicity with which the elements of the *ethos–pathos–logos* triad evolved was clearly manifested in the revolutionary transformations of primitive technologies. Agriculture, for example, appeared not merely as a set of isolated tools, but as a synchronized complex involving practical techniques, symbolic order, social organization, seasonal imagination, and spiritual interpretation. The plough, the granary, the calendar, the ritual, and the village were not independent inventions accidentally placed side by side; they formed an integrated civilizational architecture. In this sense, Mircea Eliade in [6] points out definitively:

> *The transition from a hunter-gatherer economy to agriculture represents not only a technical advance, but a spiritual mutation. The sacredness of the earth, the seed, and the plant cycles reorganized the entire religious and social life.*

But that synchronicity is no longer evident in the technological developments of our time. Formal developments associated with *logos*, and especially technological developments associated with *pathos*, evolve with an *exponential* dynamic. Semiconductor exponential scaling [12, 13], digital communication, machine learning, and automated design environments have accelerated the production of technical possibilities. On the third dimension, that of *ethos*, spirituality and normative reflection evolve with a much slower, often *logarithmic* dynamic. The human community receives technologies faster than it can interpret them, regulate them, educate itself for them, and integrate them into stable forms of life.

The consequence is a civilizational phase shift. The *technosphere* expands faster than the *noosphere* can assimilate it. Hardware and software systems acquire a complexity, autonomy, and performative power that exceed the conceptual instruments used by ordinary institutions, legal systems, educational structures, and even many scientific communities. Under these conditions, the central question is no longer only: *what can be computed?* It becomes also: *what forms of meaning, responsibility, and social order are compatible with the new computational substrate?*

The hardware revolution must therefore be understood at several levels. At the technical level, it is the transition from mechanical devices to Boolean circuits, from Boolean circuits to programmable machines, and from programmable machines to trainable and self-optimizing computational fabrics. At the conceptual level, it is a transition from procedural descriptions to functional, architectural, and eventually semantic descriptions in natural languages using Artificial Intelligence (AI). At a cultural level, it is a transition from tools that extend human action to systems that participate in shaping human decisions, thus closing loops in which humans and artificial tools (example: AI) work together to solve problems.

The classical von Neumann computer was organized around a procedural model: an instruction stream controls a datapath, and computation appears as a temporal sequence of elementary operations [20]. The functional view, by contrast, considers computation as the construction and transformation of functions. This view was already present in mathematical logic and computability theory [4, 7], and it was later reformulated in programming-language terms by Backus as a critique of the von Neumann style [2]. The current hardware revolution gives this critique a new physical meaning. The main bottleneck is no longer only the elegance of programs; it is the mismatch between procedural control and massively parallel, spatially distributed, energy-constrained physical substrates.

AI intensifies this transition. In a trained neural network, the operational structure is not a procedure written line by line by a programmer. It is a large family of numerical transformations parameterized by learned weights. Such a system is not programmed in the traditional sense; it is configured by training, constrained by architecture, and deployed as a functional object. The same tendency now appears in hardware design tools: layout, verification, scheduling, synthesis, and architectural exploration are increasingly influenced by heuristic, statistical, and learning-based methods. Thus, the hardware revolution is not only about faster chips. It is about a change in the ontology of design.

The purpose of this paper is to articulate this transformation through the triad *logos–pathos–ethos*. The *logos* level concerns formal structures, computational models, programming paradigms, and semantic descriptions. The *pathos* level concerns technological imagi-

nation, desire, invention, projection, and the affective energy through which societies pursue artificial intelligence, automation, and computational power. The *ethos* level concerns responsibility, communal meaning, spirituality, education, and the normative orientation of technological civilization.

The paper advances the following claims:

- hardware evolution can be described as a sequence of structural transformations: mechanical, Boolean, programmable, and trainable;
- software evolution can be described as a transition from procedural programming to value-level functional programming, function-level functional programming, and specification-level functional programming;
- artificial intelligence and semiconductor technologies now form a feedback loop in which hardware enables AI and AI increasingly participates in the design of hardware;
- the main social risk is not only technical failure, but desynchronization between technological possibility and ethical assimilation;
- an architectural approach, grounded in families of functions and interpretable through category-theoretic concepts [1], can mediate between formal structures, operational meanings, and deep meanings.

2 Logos Level Effects

At the level of *logos*, the hardware revolution appears as a transformation in the formal structure of computation. Computation is not only a practical activity performed by machines; it is a mode of representation. Each technological stage materializes a different formal ontology: mechanism, logic, program, and function.

2.1 Hardware Structure Technologies

The physical support of computing – hardware – has evolved in four major stages:

- the stage of automatic mechanical devices, such as clockwork and gear-based systems;
- the stage of circuits based on Boolean algebra;
- the stage of programmable structures realized as physical and informational structures;
- the stage of trainable structures in which explicit programs are partly replaced by weight matrices and learned transformations.

The first stage is the mechanical stage. The machine is a visible arrangement of parts whose behavior is governed by geometry, force, friction, and temporal synchronization. Its logic is not yet symbolic in the strict sense; it is embodied in the physical arrangement of levers, wheels, cams, gears, and shafts. Mechanical computation is procedural because the sequence of operations is physically constrained by the machine itself. The procedure is the mechanism.

The second stage is the Boolean stage. With Boolean algebra [3] and its technological interpretation in switching circuits [17], computation becomes a symbolic physical process. The elementary distinction is no longer primarily a mechanical displacement, but a logical state. Relays, tubes, and later transistors implement truth functions. The machine becomes a materialized logical calculus. This stage is decisive because it separates the formal behavior of the machine from the immediate mechanical intuition of its parts.

The third stage is the programmable stage. The stored-program computer introduces a new relation between structure and behavior [20]. A fixed physical machine can perform many different computations because its behavior is controlled by symbolic instructions stored in memory. This stage is characterized by the separation between hardware and software. Hardware provides a universal operational substrate; software provides a temporal sequence of commands. The dominant model is procedural: an algorithm is decomposed into instructions, and the machine executes them step by step.

The fourth stage is the trainable stage. Here the central object is no longer the instruction stream, but a structured numerical transformation. Neural networks, systolic arrays, tensor processors, and domain-specific accelerators embody computations that are better described as maps between high-dimensional spaces than as conventional instruction sequences. The state of the machine includes trained parameters. The machine is not merely programmed; it is conditioned by data, architecture, objective functions, and optimization procedures.

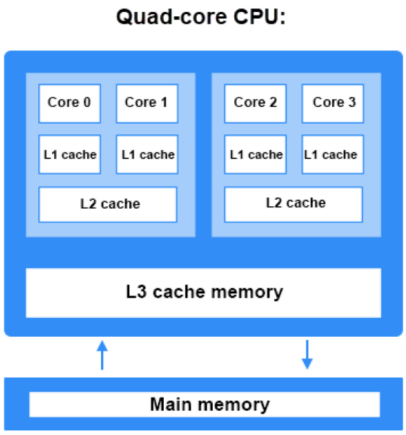


Figure 1: Multi-core, because *"the semiconductor industry ... switched from making microprocessors run faster to putting more of them on a chip – doing so without any clear notion of how such devices would in general be programmed."* [14] (source: Google).

From a market perspective, Intel's multi-core processor (Figure 1), or Nvidia's General Purpose Graphic Processing Unit (GPGPU) (Figure 2), or Google's Tensor Processing Unit (TPU) (Figure 3), or Intel's Video Processing Unit (VPU), or Intel's Neural Processing Unit (NPU) (Figure 4),or Nvidia's Data Processing Unit (DPU) and the like dominate the hardware offering in the absence of a generic abstract model for accelerating intensive computing as exists for single-core applications for which Turing's mathematical model accompanied by the Harvard and von Neumann abstract models represents a rigorous foundation. These hardware structures represent the physical support of data-driven processes in the context of informational structures obtained through the convergence of iterative training processes based on increasingly larger and more diverse data flows.



Figure 2: The oxymoronic General Purpose Graphic Processing Unit which inherits the burden of some functionalities from the time when the architecture on which it is based was used in the field of graphics. (source: Google)

This does not mean that procedural computation disappears. Rather, it loses its exclusive status. The current computational landscape is heterogeneous. It includes:

- sequential processors optimized for control;
- vector and SIMD machines optimized for regular numerical operations;
- GPUs optimized for throughput and data parallelism;

- reconfigurable hardware optimized for spatial computation;
- neural accelerators optimized for dense linear algebra and learned transformations;
- memory-centric and near-memory structures attempting to reduce the energy cost of data movement.

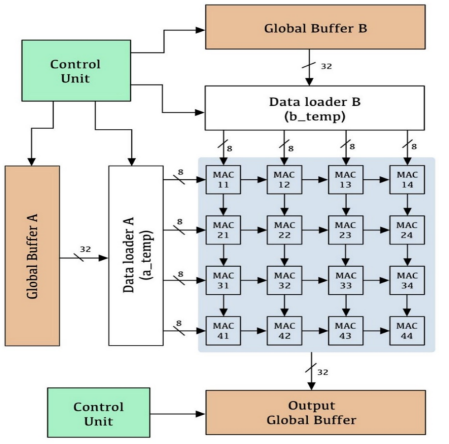


Figure 3: Tensor Processing Unit which is a systolic circuit that proves its efficiency only in the limited domain of matrix operations. (source: Google)

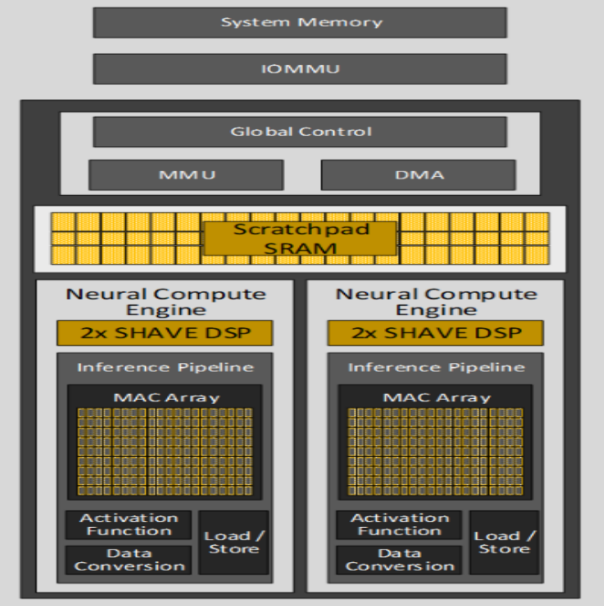


Figure 4: Neural Processing Unit which runs fast for Conv/GEMM-based applications only. (source: Google)

The central problem is that the classical procedural model is no longer the best abstraction for all these substrates. A hardware system composed of thousands or millions of parallel functional units is poorly described as a single temporal stream of instructions. Moreover, an approach similar to that taken for single-core computing is needed: a mathematical model followed by an abstract model that will support various architectural definitions. In this sense, we propose the mathematical model of *partially recursive functions* defined by Stephen Kleene [7], based on which [18] the *abstract recursive parallel model*, suggested by the representation in Figure 5, can be defined.

Definition 1. *The system $MSR_{i+1}(p(\ldots))$, represented in Figure 5, contains:*

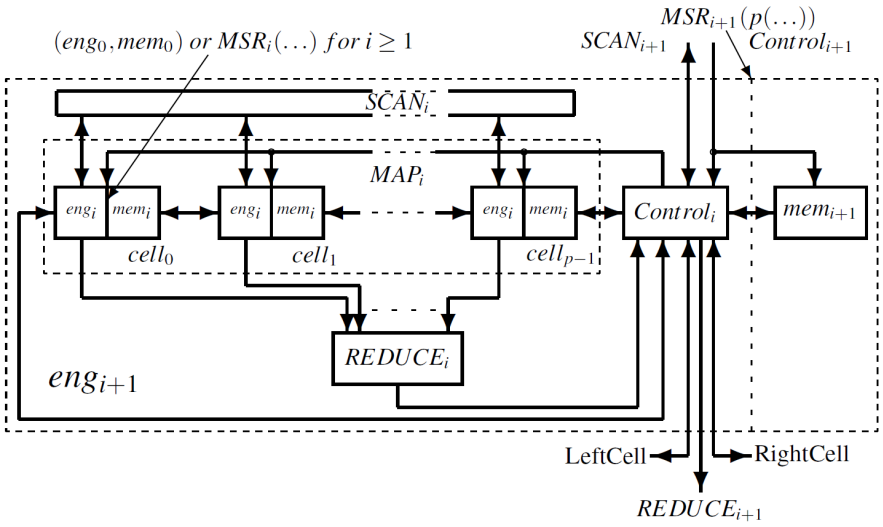


Figure 5: The recursive abstract model for parallelism. Except for $cell_0$, consisting of *execUnit* and *regFille*, each cell, $cell_i$, assumes a pair eng_i, mem_i with the structure of the entire system eng_{i+1}, mem_{i+1}.

- **MAP_i**: *a finite linear array of* p *cells. Each of* p *cells consists of a data memory (and sometimes a program memory) mem_i, and a computing engine eng_i. Each cell executes, according to its internal state, the command (instruction, function) issued by the control unit $Control_i$ through a* log*-depth binary tree network (unrepresented in Figure 5 because it has no functional role)*

- **$REDUCE_i$**: *a log-depth tree network used to reduce a n-dimensional data structure to a $(n-1)$-dimension data structure (for example: a vector to a scalar)*

- **$SCAN_i$**: *a log-depth 2-dimension network of functional units which takes from MAP_i a vector and sends back as a result another vector (for example: computing a permutation or a sum of prefixes)*

- **$Control_i$**: *is a* mono/multi-core *computing unit, with its own data and program memory, which issues, through a log-depth distribution network (not shown in Figure 5),*

commands for the MAP_i array; it receives the output of the $REDUCE_i$ net and is serially connected with the MAP_i array for data transfer

- **$\mathbf{mem_{i+1}}$**: *stores data and programs for the entire system. The transfer between mem_{i+1} and the local memories, mem_i, distributed in MAP_i, is supervised by $Control_i$*

where:

- **$\mathbf{eng_{i+1}}$** *is defined by gathering in one block modules MAP_i, $REDUCE_i$, $SCAN_i$, and $Control_i$, for $i \geq 0$*

defines a **recursive abstract model for parallel computation**, *containing in the lowest level of the hierarchy the pair (***$\mathbf{eng_0}$**, **$\mathbf{mem_0}$***), represented by the smallest and simplest execution unit and a register file.*

◇

Such a system can be actualized architecturally using a family of interacting functions distributed in space, constrained by communication, synchronization, memory locality, energy, and precision.

Definition 2. *An* architectural function *is a function whose realization is not independent of the physical and informational organization that implements it. Unlike a purely mathematical function, an architectural function includes constraints of locality, parallelism, communication, memory hierarchy, timing, and energy.*

◇

This definition is important because it prevents a frequent simplification. In mathematics, a function is often considered extensionally: two functions are equal if they produce the same outputs for the same inputs. In hardware, two extensionally equivalent functions may be radically different architecturally. They may differ in latency, throughput, area, energy, numerical stability, fault behavior, or verifiability. Thus, the hardware revolution forces us to enrich the notion of function.

2.2 Software Structure Technologies

Software has undergone a parallel transformation. At the beginning, programming was closely tied to the machine. A program was a sequence of operations imposed on a concrete physical substrate. Later, programming languages abstracted the machine, but often preserved the procedural character of computation: assignment, sequencing, branching, iteration, procedure calls, and memory updates.

Let us consider the two main types of functional languages identified by John Backus [2]:

- **value-level** functional programming languages, typically represented by John McCarthy's LISP, for LISt Processor [10]; they combine values to form other values until the final values are reached;

- **function-level** functional programming languages, typically represented by John Backus's FP; following the principle of compositionality, they apply program-forming operations to programs to form a succession of programs ending in the final program.

To these we can add:

- **spec-level** functional programming, toward which many AI promoters aspired, starting with John McCarthy's proposal of "common sense" and formulations close to natural language for specifying **what** needs to be done, allowing the machine to determine **how** to do what is desired [9].

These three levels correspond to three different degrees of distance from the machine:

1. at the value level, the programmer manipulates data objects;
2. at the function level, the programmer manipulates programs as composable functions;
3. at the specification level, the human provides constraints, intentions, goals, or descriptions, while the system constructs or searches for an operational realization.

The last level is no longer merely a programming-language issue. It is now becoming a hardware-design issue. A modern design process often begins with an intended behavior, a performance target, a power budget, a technological node, a set of constraints, and a library of possible components. The designer no longer writes every structural detail directly. Synthesis tools, place-and-route tools, verification tools, and increasingly AI-based assistants participate in the generation of the final artifact. The design process itself becomes functional: it maps specifications into architectures.

We may therefore distinguish between three forms of description:

- **procedural description**: the desired process is expressed as a sequence of operations;
- **structural description**: the desired process is expressed as an interconnection of components;
- **functional description**: the desired process is expressed as a transformation with constraints, leaving open a family of possible realizations.

Classical software is dominated by procedural description. Digital hardware description languages are dominated by structural and behavioral description. Modern AI-oriented systems require functional description because the main object is often a transformation whose internal implementation may be discovered, trained, synthesized, or optimized.

Example 1. *A matrix multiplication can be described procedurally by three nested loops. It can be described structurally by an array of multiply-accumulate units. It can be described functionally as a bilinear map*

$$[C = A \times B]$$

with constraints on precision, throughput, latency, memory traffic, and energy. In the context of hardware acceleration, the third description is often the most relevant because it admits many possible architectures: scalar, vector, systolic, tiled, streaming, approximate, sparse, or quantized.

◇

The functional approach is not an escape from implementation. On the contrary, it is a disciplined way of organizing implementation choices. It allows the designer to navigate from intention to structure without prematurely reducing the problem to a single procedural expression.

2.3 From Program to Architecture

The procedural program is temporal: it says what must happen first, next, and later. The architecture is spatial and relational: it says what components exist, how they communicate, how they synchronize, and what transformations they collectively realize. This distinction is decisive for the hardware revolution.

In the von Neumann style, the program counter is the symbolic center of computation. In highly parallel accelerators, there is no single center of comparable importance. Control is distributed, data movement dominates energy, and locality often matters more than arithmetic complexity. Computation becomes less a story of instructions and more a geometry of transformations.

This change has a direct philosophical consequence. A program is close to a text. It can be read linearly, interpreted syntactically, and debugged as a sequence. An architecture is closer to a diagram, a category [1], or a system of relations. Its meaning emerges from composition, interface, and invariant preservation. Therefore, the transition from procedural to functional hardware is also a transition from textual thinking to architectural thinking.

3 Pathos Level Effects

The level of *pathos* concerns technological imagination: the affective and projective energy through which a society desires, fears, invents, and accelerates technology. This term should not be understood in its weak English sense as "pathetic", but in its older sense connected to passion, affect, and imaginative drive.

The limits of this technological imagination were foreshadowed from the beginning of electronic computation. Alan Turing, before the large-scale serial production of computers, already discussed machine intelligence and the possibility of artificial behavior comparable to human reasoning [19]. In the same historical horizon, John McCarthy formulated the ambition of artificial intelligence and later organized the famous Dartmouth meeting. The idea of artificial intelligence was therefore not a late application of computing; it was one of the founding projections of computing.

The history of computing can be read as a repeated expansion of pathic expectation:

- first, the expectation that machines would automate arithmetic;

- then, the expectation that machines would automate logic;
- then, the expectation that machines would automate information processing;
- then, the expectation that machines would automate perception, learning, and decision;
- now, the expectation that machines will participate in invention, design, scientific discovery, and even cultural production.

Each expectation produces new hardware requirements. Arithmetic automation required reliable switching elements. Logic automation required formal circuits. Information processing required memory hierarchy and programmable control. Perception and learning required massive numerical throughput. Design automation now requires machines capable of exploring large spaces of possibilities.

The pathos of the present hardware revolution is dominated by artificial intelligence. AI is not merely another application domain. It reorganizes the hierarchy of hardware requirements. Memory bandwidth, energy per operation, communication topology, sparsity, quantization, tensor operations, and dataflow become central architectural concerns. At the same time, AI changes the design process itself. Hardware is designed for AI, but AI also begins to design hardware. This circularity is one of the defining features of the present moment.

3.1 The Feedback Loop Between AI and Hardware

The current revolution is characterized by a feedback loop:

$$[\text{better hardware} \rightarrow \text{larger AI models} \rightarrow \text{better design tools} \rightarrow \text{better hardware.}]$$

This loop is not purely technical. It reorganizes research institutions, industrial investments, military strategies, educational priorities, and cultural expectations. It also changes the psychology of design. The designer is no longer only a person who specifies exact structures. The designer becomes a mediator between intentions, constraints, automated tools, and generated artifacts.

This feedback loop has several consequences:

- the boundary between design and optimization becomes less clear;
- the boundary between program and data becomes less clear;
- the boundary between architecture and training becomes less clear;
- the boundary between human intention and machine-generated structure becomes less clear.

The traditional engineering ideal was complete explicitness: every relevant decision should be traceable to a human designer or a formal rule. AI-assisted design weakens this ideal. Some decisions become statistical, heuristic, or emergent from search. This does not make them irrational, but it requires a new theory of responsibility, validation, and interpretation.

3.2 Imagination and Its Risks

Technological imagination is necessary. Without it, no hardware revolution is possible. But imagination becomes dangerous when it is detached from formal discipline and ethical orientation. The pathos of AI often oscillates between two exaggerations: technological salvation and technological catastrophe. Both are insufficient. The first underestimates risk; the second underestimates human agency.

The real problem is more precise: the acceleration of technical possibility produces a widening gap between what can be built and what can be responsibly understood. This gap is visible in several domains:

- verification of complex hardware/software/AI systems;
- explainability of learned models;
- social dependence on opaque infrastructures;
- concentration of technological power;
- educational lag between tools and conceptual understanding;
- erosion of human skills by premature automation.

The hardware revolution therefore cannot be evaluated only by performance metrics. Speed, area, power, and cost are necessary engineering quantities, but they are not sufficient civilizational quantities. The pathos of invention must be disciplined by logos and oriented by ethos.

4 Ethos Level Effects

The level of *ethos* concerns the normative and spiritual organization of human life. In technological civilization, ethos determines how tools are integrated into education, work, community, political order, scientific practice, and personal formation.

Until recently, communities were organized mainly through syntactic structures: laws, contracts, procedures, doctrines, books, programs, curricula, and institutional rules. These structures operated with meanings determined by reference and context. The word, the document, and the rule were central instruments of social coordination. Advanced information technologies based on semiconductor structures now create the possibility of operating not only with syntactic forms, but also with semantic approximations, learned representations, patterns, and contextual meanings.

This is a profound change. Freed partially from rigid syntactic constraints, human communities may focus more directly on phenomenological complexity. They may operate with meanings that are not reducible to fixed symbolic definitions. However, this possibility is double-edged. A system that manipulates meanings without genuine responsibility can produce confusion, manipulation, or simulated understanding. Therefore, the transition from syntax to meaning must be accompanied by a transition from external regulation to deeper responsibility.

The spirituality of individual manifestations may acquire forms that are currently difficult to realize within communities and human communions. A person may be supported by intelligent systems that amplify memory, expression, design ability, scientific search, artistic imagination, and technical production. But such amplification does not automatically produce wisdom. On the contrary, amplification without orientation can intensify disorder.

The ethos problem can be formulated as follows:

$$[\text{technical amplification} \not\Rightarrow \text{human elevation.}]$$

A society may become more powerful and less coherent at the same time. It may compute more and understand less. It may communicate more and commune less. It may generate more artifacts and lose the criteria by which artifacts are judged.

4.1 Education and Formation

The first domain in which the ethos problem appears is education. The procedural school was organized around repetition, rule acquisition, and sequential mastery. The functional and AI-assisted world requires another form of education: the ability to formulate problems, evaluate generated answers, understand architectures, detect false abstraction, and connect formal results with lived meanings.

In a hardware context, this means that students should not only learn how to write code or draw circuits. They should understand the relation between:

- mathematical function and physical realization;
- algorithm and architecture;
- data movement and energy;
- precision and meaning;
- automation and responsibility;
- specification and verification.

The old educational model assumed that the learner must first master procedures and then later understand structures. The new model should introduce architecture earlier. The student must see that every computation has several possible realizations and that the choice among them is not merely technical. It involves cost, reliability, interpretability, maintainability, and social effect.

4.2 Responsibility in Generated Systems

AI-assisted hardware design raises a difficult question: who is responsible for an artifact whose detailed structure is partly generated by automated tools? A simplistic answer would attribute responsibility entirely to the user of the tool. Another simplistic answer would attribute it to the tool provider. Neither answer is sufficient.

Responsibility must be distributed across the architectural chain:

- the human who formulates the goal;
- the institution that authorizes the project;
- the tool designer who defines the search and optimization mechanisms;
- the verification process that validates the artifact;
- the deployment context in which the artifact acts;
- the community that accepts or rejects the consequences.

This does not dissolve responsibility; it structures it. In a functional and architectural world, responsibility is not attached only to isolated actions, but also to interfaces, constraints, invariants, and permitted transformations.

5 Architectural Approach

Both the description of knowledge and the generation of the technosphere will increasingly be based on families of functions – architectures –which can be interpreted using concepts from category theory [1, 5, 8].

The noosphere and the technosphere will be able to exchange structures at multiple semantic levels:

- formal semantic structures;
- meanings of reference and context;
- operational meanings embedded in tools;
- deep meanings connected with human orientation, value, and spiritual experience.

The architectural approach is proposed as a mediation between these levels. It does not reduce *ethos* to *logos*, nor does it dissolve logos into subjective meaning of *pathos*. Instead, it introduces an intermediate level: organized families of functions that can be formally described, technologically implemented, and humanly interpreted.

5.1 Architecture as a Family of Functions

In the present paper, an architecture is not merely a block diagram. It is a structured family of functions together with rules of composition, constraints of realization, and interfaces to an environment.

Definition 3. *An* architecture *is a tuple:* $[\mathscr{A} = (O, F, C, I, E),]$ *where (O) is a family of objects or states, (F) is a family of transformations, (C) is a set of composition rules, (I) is a set of interfaces, and (E) is a set of environmental constraints.*

◇

This definition is intentionally general. In hardware, (O) may contain data states, memory states, register states, or signal configurations. (F) may contain combinational functions, sequential transitions, dataflow transformations, or learned maps. (C) may express temporal sequencing, spatial composition, pipelining, synchronization, or feedback. (I) may express buses, protocols, instruction sets, APIs, or semantic interfaces. (E) may include timing, area, energy, reliability, fabrication constraints, or social constraints.

The category-theoretic intuition is that an architecture can be seen as a world of *composable transformations*. *Objects* correspond to states or types; *morphisms* correspond to admissible transformations; *composition* corresponds to the construction of larger behaviors from smaller ones. *Functors* can then express translations between domains: for example, from a mathematical specification to a hardware architecture, from an algorithmic description to a dataflow graph, or from a trained model to an accelerator implementation.

This does not mean that category theory automatically solves hardware design. Such a claim would be exaggerated. The value of category theory is more modest and more precise: it provides a language for composition, abstraction, interface preservation, and structural translation. These are exactly the notions needed when moving from procedure to function and from function to architecture.

5.2 From Syntax to Meaning

Classical digital systems operate mainly on syntactic structures. Bits are manipulated according to formal rules. Meaning is external to the machine. A word in memory has no intrinsic semantic force; it is meaningful only through interpretation by a programmer, a protocol, an application, or a user.

AI systems complicate this situation. They do not possess meaning in the human sense, but they operate on internal representations that correlate with semantic distinctions. This produces an intermediate layer between syntax and human meaning. The machine does not understand as a person understands, but it can transform symbols in ways that are sensitive to context, analogy, and learned association.

The architectural approach must therefore distinguish at least four levels:

1. **syntactic level**: formal manipulation of symbols or signals;
2. **structural level**: organization of components and transformations;
3. **operational semantic level**: behavior that preserves useful distinctions in a domain;
4. **deep meaning level**: human interpretation connected with value, purpose, responsibility, and spiritual orientation.

Confusing these levels is dangerous. Treating statistical correlation as deep meaning is naive. Treating all meaning as mere syntax is equally naive. The hardware revolution requires a disciplined separation and mediation among levels.

5.3 Architectural Mediation

Architectural mediation is the process by which formal structures and phenomenological meanings are connected without reducing one to the other. In technical terms, it is the construction of interfaces between levels of description. In philosophical terms, it is the attempt to preserve both formal rigor and human depth.

For hardware design, mediation occurs when a high-level intention is transformed into a specification, then into an architecture, then into circuits, then into physical silicon, and finally into behavior in a human context. At each stage, meaning can be lost, distorted, refined, or amplified.

We may represent this chain schematically as:

$$[\texttt{intention} \rightarrow \texttt{specification} \rightarrow \texttt{architecture} \rightarrow$$

$$\rightarrow \texttt{implementation} \rightarrow \texttt{operation} \rightarrow \texttt{socialeffect}.]$$

The procedural approach focuses mainly on the middle of this chain: `specification`, `implementation`, and |tt operation. The architectural approach attempts to preserve the entire chain.

This is especially important for AI-designed or AI-assisted hardware. If the machine contributes to the transition from specification to implementation, then the human designer must strengthen the transitions from intention to specification and from operation to social effect. Otherwise, the most important meanings remain outside the design process.

6 Toward a Functional Hardware Revolution

The phrase "functional hardware revolution" can be misunderstood. It does not mean that hardware becomes software, nor that circuits become mathematical abstractions without physical cost. It means that the dominant design object shifts from the instruction sequence to the constrained function.

In the procedural paradigm, the central question is:

What sequence of operations must be executed?

In the functional-architectural paradigm, the central question becomes:

What transformation must be realized,
under what constraints, by what family of structures?

This second question is better suited to modern hardware because it admits spatial, parallel, approximate, learned, and heterogeneous realizations.

6.1 Consequences for Computer Architecture

Computer architecture must therefore expand its conceptual base. It can no longer be only the art of designing processors that execute instruction sets efficiently. It must include

the design of functional fabrics, memory-compute organizations, accelerator interfaces, dataflow systems, reconfigurable substrates, and trainable computational structures.

The central architectural quantities become:

- locality of data and computation;
- composability of functional blocks;
- explicit representation of communication;
- precision as an architectural parameter;
- energy as a first-class constraint;
- verifiability of generated structures;
- semantic adequacy of the implemented transformation.

This is not a rejection of the stored-program computer. The von Neumann model remains indispensable for control, generality, and symbolic manipulation. But it is no longer sufficient as the dominant imagination of computation. The future system will be heterogeneous: procedural control will coexist with functional acceleration.

6.2 Consequences for Hardware Design Tools

Hardware design tools must also change. Classical electronic design automation translates descriptions into structures under constraints. AI-assisted design tools add search, prediction, generation, and adaptation. The designer increasingly works at the level of intentions, constraints, examples, performance objectives, and verification properties.

This creates a new kind of design language. Such a language must express:

- what function is required;
- what approximations are acceptable;
- what invariants must never be violated;
- what resources may be used;
- what contexts of operation are expected;
- what evidence is sufficient for trust.

The last item is crucial. In procedural design, trust is often built through inspection and testing. In AI-assisted design, inspection may be insufficient because the generated structure can exceed direct human readability. Therefore, trust must be built through a combination of formal verification, simulation, testing, architectural constraints, and operational monitoring.

7 Conclusion

The hardware revolution is not only a technological acceleration. It is a transformation in the relation between procedure, function, architecture, and meaning. Mechanical devices embodied procedure physically. Boolean circuits embodied logic electrically. Stored-program computers separated hardware from software and made procedure symbolic. Trainable and AI-oriented systems now push computation toward functional and architectural organization.

This transition has effects at the three levels of *logos*, *pathos*, and *ethos*. At the level of *logos*, it requires new formal descriptions of hardware as families of constrained functions. At the level of *pathos*, it intensifies technological imagination through the feedback loop between AI and semiconductor design. At the level of *ethos*, it creates a gap between technical amplification and human assimilation.

The main danger is desynchronization. Technical systems may evolve exponentially while ethical, educational, and spiritual structures evolve much more slowly. The result may be a civilization that is computationally powerful but architecturally incoherent. The proposed architectural approach attempts to mediate this gap. By treating architectures as structured families of functions, and by distinguishing syntactic, structural, operational-semantic, and deep-meaning levels, it becomes possible to connect formal rigor with human responsibility.

The transition from procedural to functional hardware should therefore not be understood as a purely technical matter. It is a civilizational transformation. The question is not only how to build more powerful chips, but how to integrate them into a noosphere capable of understanding, orienting, and humanizing the technosphere it creates.

References

[1] Ana Agore (2023) *A First Course in Category Theory*, Springer.

[2] John Backus (1978) Can programming be liberated from the von Neumann style? A functional style and its algebra of programs, *Communications of the ACM*, **21**(8):613–641.

[3] George Boole (1854) *An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities*, Walton and Maberly.

[4] Alonzo Church (1936) An unsolvable problem of elementary number theory, *American Journal of Mathematics*, **58**(2):345–363.

[5] Samuel Eilenberg and Saunders Mac Lane (1945) General theory of natural equivalences, *Transactions of the American Mathematical Society*, **58**(2):231–294.

[6] Mircea Eliade (1976) *Historie des croyances et des idées religieuses, I. de l'âge de la pierre aux mystères d'Éleusis*, Payot, Paris.

[7] Stephen C. Kleene (1936) General recursive functions of natural numbers, *Mathematische Annalen*, **112**:727–742.

[8] Saunders Mac Lane (1971) *Categories for the Working Mathematician*, Springer.

[9] John McCarthy (1959) Programs with common sense, in *Proceedings of the Symposium on Mechanisation of Thought Processes*, National Physical Laboratory, Teddington, England.

[10] John McCarthy (1960) Recursive functions of symbolic expressions and their computation by machine, Part I, *Communications of the ACM*, **3**(4):184–195.

[11] Carver Mead and Lynn Conway (1980) *Introduction to VLSI Systems*, Addison-Wesley.

[12] Gordon E. Moore (1965) Cramming more components onto integrated circuits, *Electronics*, **38**(8):114-117.

[13] Gordon E. Moore (1975) Progress In Digital Integrated Electronics, *IEEE, International Electron Devices Meeting. Technical Digest* pp.11-13.

[14] D. A. Patterson (2010) The trouble with multicore. *IEEE Spectrum*, **47**(7):28-32.

[15] Frank Rosenblatt (1958) The perceptron: a probabilistic model for information storage and organization in the brain, *Psychological Review*, **65**(6):386–408.

[16] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams (1986) Learning representations by back-propagating errors, *Nature*, **323**:533–536.

[17] Claude E. Shannon (1938) A symbolic analysis of relay and switching circuits, *Transactions of the American Institute of Electrical Engineers*, **57**(12):713–723.

[18] Gheorghe M. Ştefan (2020) *Composition is the only independent rule in Kleene's model of partial recursive functions*, At: `http://users.dcae.pub.ro/~gstefan/2ndLevel/composition.html`

[19] Alan M. Turing (1950) Computing machinery and intelligence, *Mind*, **49**:433–460.

[20] John von Neumann (1945) *First Draft of a Report on the EDVAC*, Moore School of Electrical Engineering, University of Pennsylvania.

[21] Norbert Wiener (1948) *Cybernetics: or Control and Communication in the Animal and the Machine*, MIT Press.