Embedded Systems: Challenges in Specification, Verification, and Synthesis
or
What is New Beyond Reactive and Real-Time Systems?

Amir Pnueli

The John von Neumann Minerva Center for the Verification of Reactive Systems
at the Weizmann Institute of Sciences

EMSOFT, Grenoble, October, 2002

Emsoft, Grenoble, October, 2002

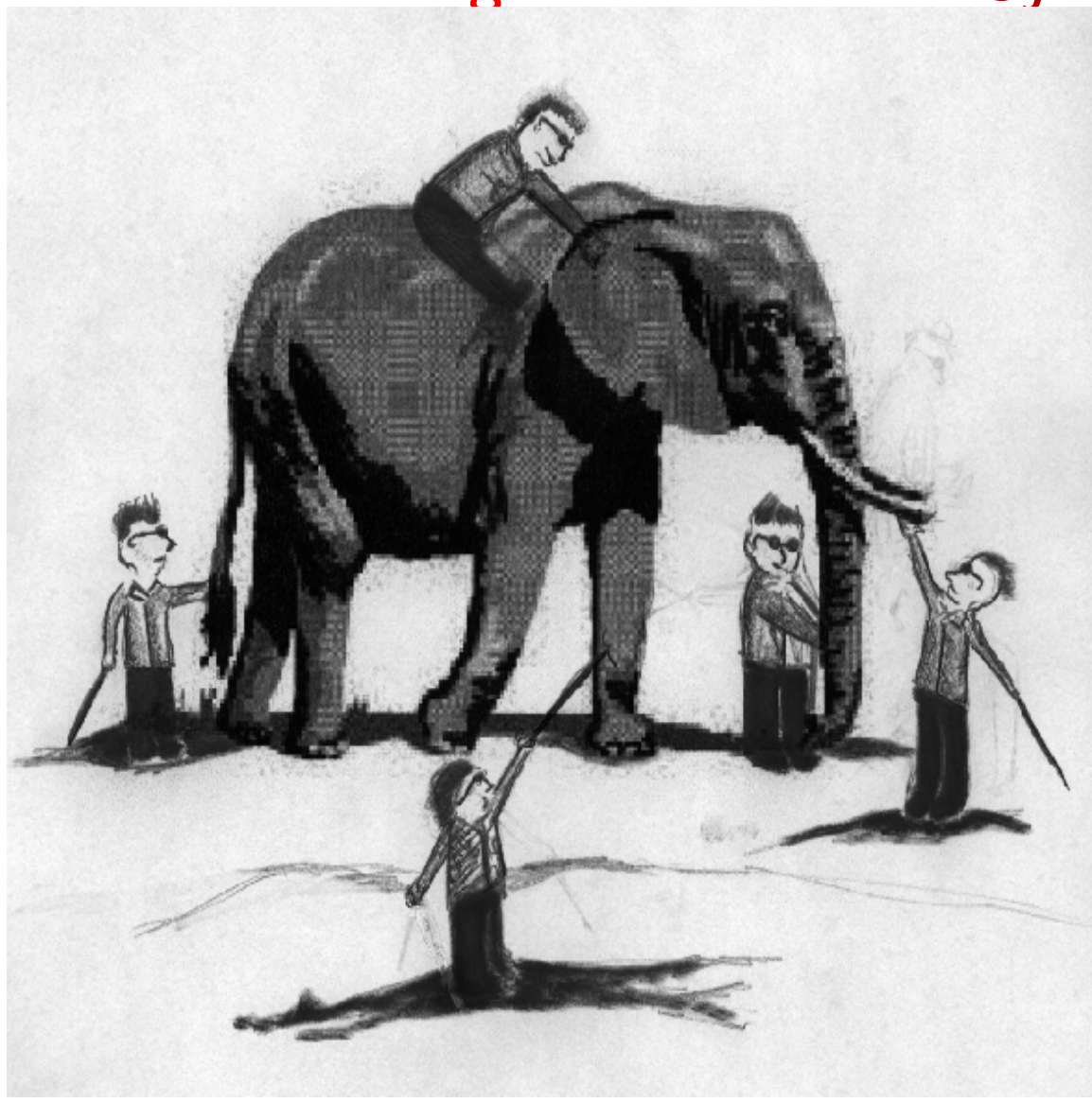
Embedded Systems

Everyone agrees that Embedded Systems will be one of the hottest items in the current millennium, both commercially and technologically.

However, this is where the consensus stops, and opinions differ as to what are the most important technological challenges whose conqueror will master the field. Sample items are:

- Communication and networking infrastructure and standards.
- Customizable and reconfigurable hardware.
- Optimization for power.
- High-level programming & operating systems.
- Distribution and coordination.
- Effective and reliable development process.

What are the Challenges in Embedded Systems?



Five Blind Computer Scientists Examining an Elephant, and Forming their Opinion about Embedded Systems

To What Extent are Embedded Systems a Newly Emerging Application Domain?

Contrary to some expressed opinions, embedded systems are not a new technology. They have been with us for the last decades, embedded in the domains of telecommunication, aviation, automotive, and other process control applications.

If there is an embedded systems revolution, it is in the anticipated huge expansion of the embedded technology into (hitherto) dumb consumer products such as refrigerators, washing machines, etc. Questions are:

- Should we adopt one of the approaches currently endorsed by one of the existing embedded systems sectors?
- Is it time to introduce a horizontally integrated field of embedded systems with its own standards and techniques?
- How can we, in a cost-effective way, attain a level of reliability comparable to that of a Maytag Washing Machine rather than that of Microsoft Windows?

Embedded Systems

Have been with us for a long while, but have often been treated like a poor relative.

- They have been programmed in assembly language forsaking all the new developments in high-level programming languages.
- They have been implemented on inferior platforms, on micro-controllers rather than state-of-the art micro-processors.
- For the holy grail of predictability, scheduling has been constrained to the most rigid regimes, optimization and parallelism has been outlawed.

There is a middle way. With the modern techniques of rigorous analysis and verification, it is possible to use modern technology while maintaining reliability. It is not important to know

What your system is doing at any hour of the night,

but rather that

Your system does the right thing at the right time.

My Own Answers

To the questions:

Are there significantly new challenges introduced by the new vision of embedded systems which were not previously considered in the contexts of reactive, distributed, real-time, or safety-critical systems?

Should we expect radically new solutions?

My Own Answers

To the questions:

> Are there significantly new challenges introduced by the new vision of embedded systems which were not previously considered in the contexts of reactive, distributed, real-time, or safety-critical systems?
>
> Should we expect radically new solutions?

are:

> Probably not! I still believe that the right solution is a rigorous development and analysis process, appropriately incorporating formal techniques.

But now, we really mean it!

A Seamless Development Process: From Requirements to Implementation

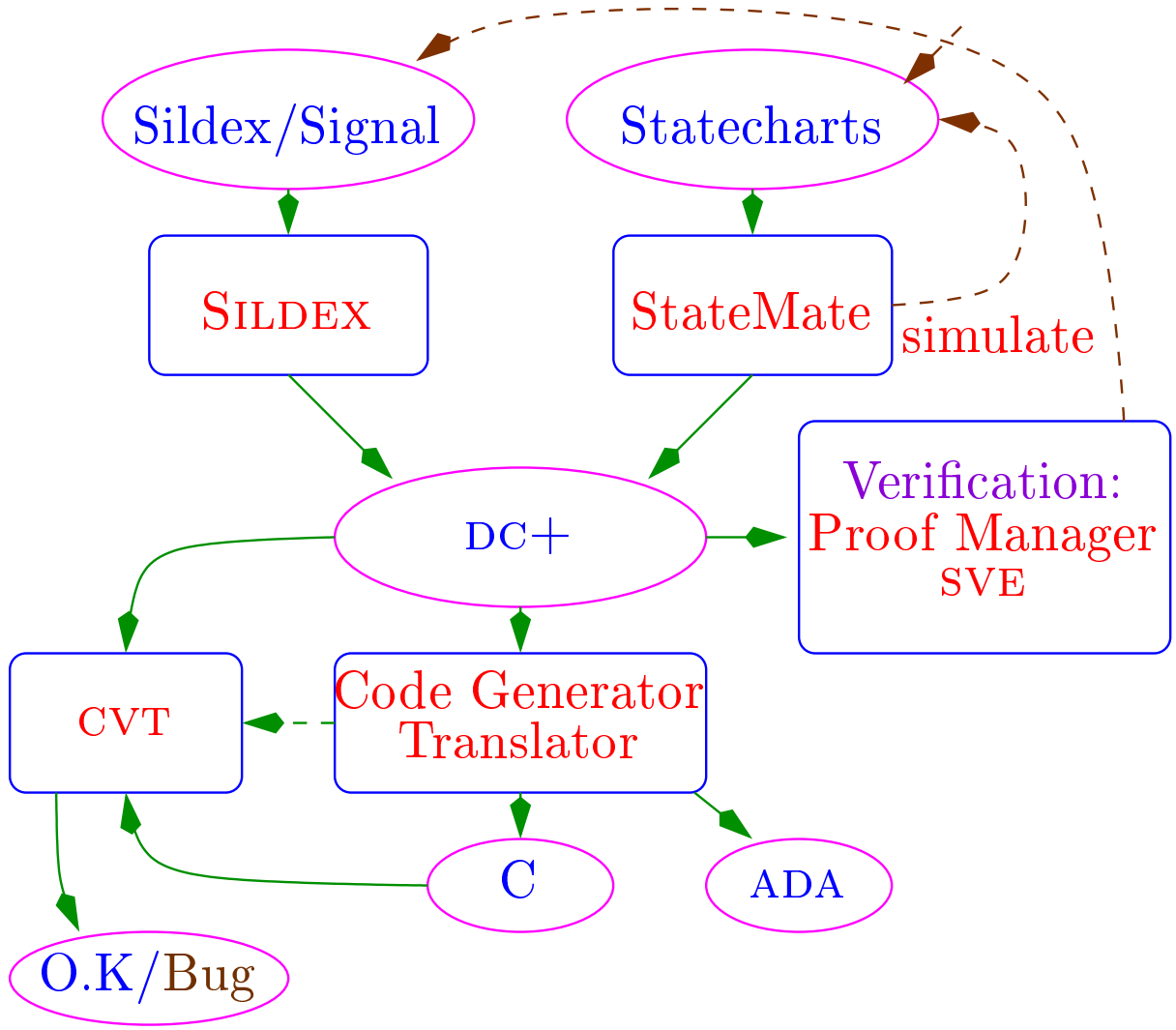
An effective and reliable development process should support a fast turn-around and the ability to produce many customized variations of the same basic product.

Previous Experiments: The *Sacres*/safeair Projects

- Safety Critical Embedded Systems – From specification to architecture.
- A 4th framework European Community *Esprit* research project, 1995–1998.
- Followed by the *Safeair* project, 1999-2002.

The *Sacres* Mission

Developing a methodology and a tool suit supporting the development of safety critical real time embedded systems.



What was New?

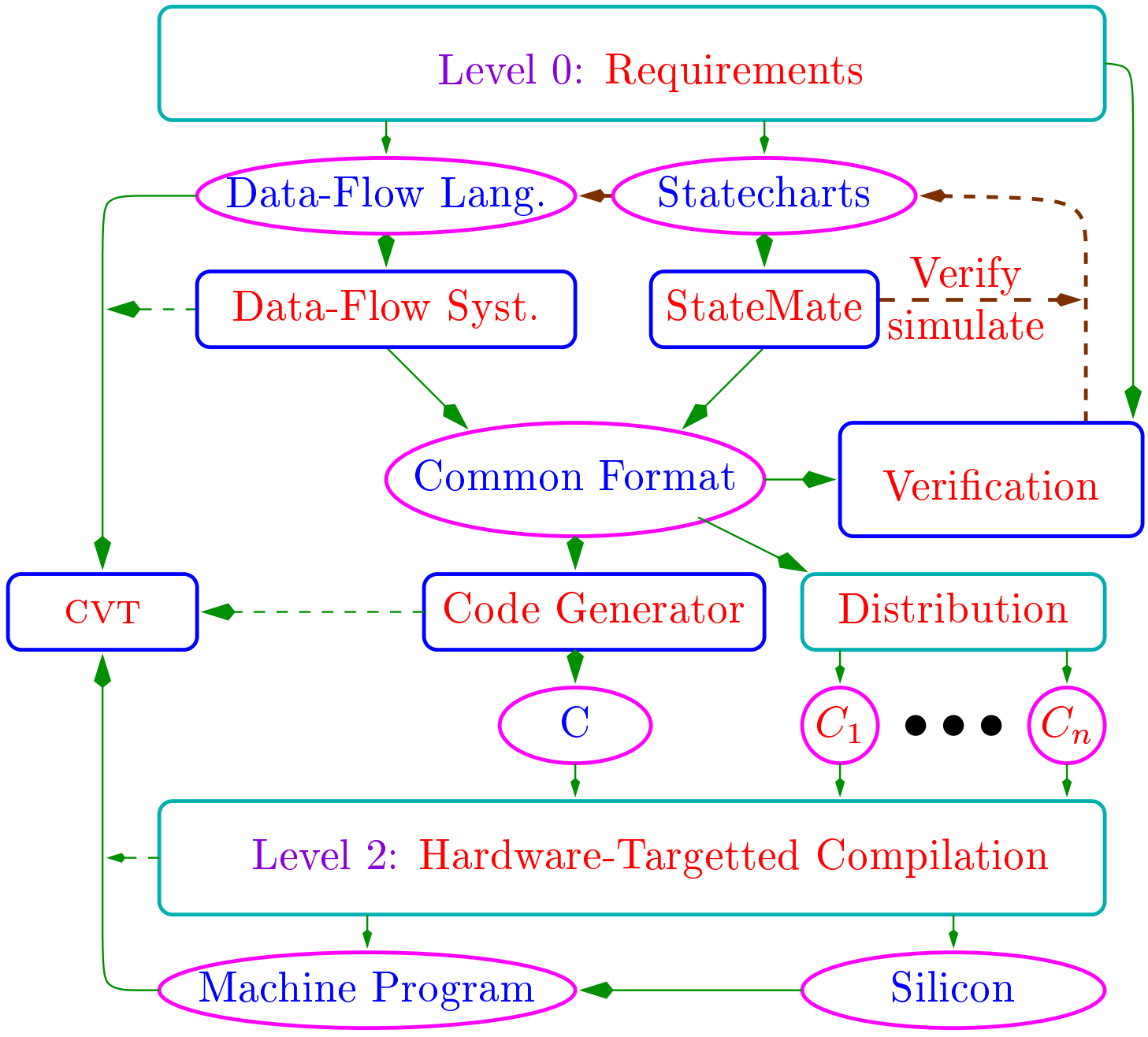
Model-Based early analysis, and development.

What is Missing?

The *Sacres* process covers only a middle layer in a more comprehensive and ambitious development process. The process can be greatly enhanced by adding:

- Level 0 – A requirement layer, in which user requirements are elucidated and crystallized into Statecharts/Sildex- level specification.
- Level 2 – A highly optimizing reconfigurable-architecture driven compiler (e.g. Trimaran) transforming C-level programs into machine-code/silicon.
- Code distribution decomposing a monolithic system description into a distributed implementation while synthesizing the necessary communication and coordination protocols between the components.
- Specification, analysis, and validation of real-time constraints.
- Adding the hybrid dimension. Incorporating into the specification/simulation cycle a representation of a continuous physical environment.

Achieve the Impossible Dream



Level 0: Requirements

Requirements, unlike specifications are not expected to provide a comprehensive integrated definition of the desired system behavior.

Rather, the process of requirement elucidation is an incremental process, possibly employing various different formalisms (languages). Every requirement adds a constraint which the system must satisfy. Possible languages:

- Temporal Logic or its timed variants, possibly in a style less threatening than

  $$\Box\, \neg(p_1 \ \wedge \ p_2), \qquad \Box\, (p \ \rightarrow \ \Diamond\, q).$$

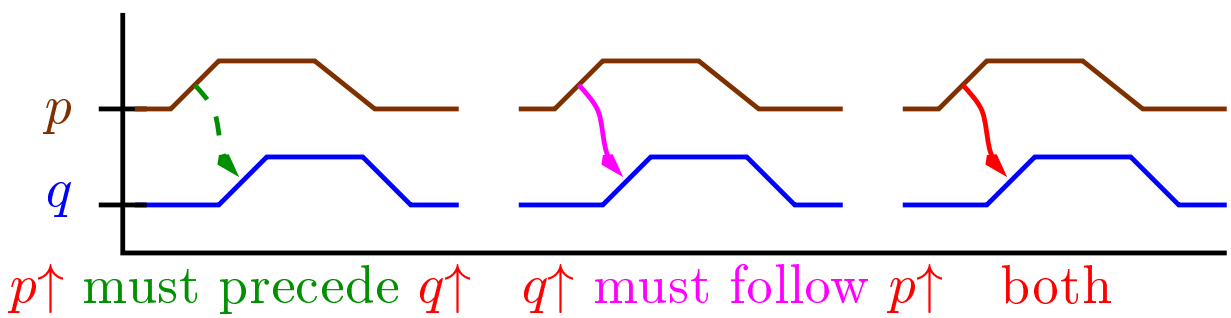
- Symbolic Timing Diagrams. These are wave forms with timing constraints and inter-wave arrows representing causality constraints.

- Live Sequence Charts – a more expressive variant [Harel and Damm 98] of the UML standard Message Sequence Charts.

Symbolic Timing Diagrams

This is a visual notation for expressing temporal requirements, originally developed by [Damm, Josko, Schlör, 95] and extended to include real-time constraints by [Feyerabend, Josko, 97].

The untimed version admits the following edges:



Can be translated into LTL as follows:

$p\uparrow$ must precede $q\uparrow$: $\overline{p} \wedge \overline{q} \rightarrow \overline{q}\, \mathcal{W}\, p$

$q\uparrow$ must follow $p\uparrow$: $\overline{p} \rightarrow \overline{p}\, \mathcal{W}\, (p \wedge \overline{q} \wedge \Diamond q)$

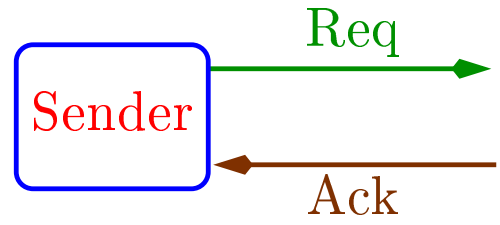
both:

$\overline{p} \wedge \overline{q} \rightarrow (\overline{p} \wedge \overline{q})\, \mathcal{W}\, (p \wedge \overline{q} \wedge (p \wedge \overline{q})\, \mathcal{U}\, ((p \wedge q) \vee \overline{p}))$

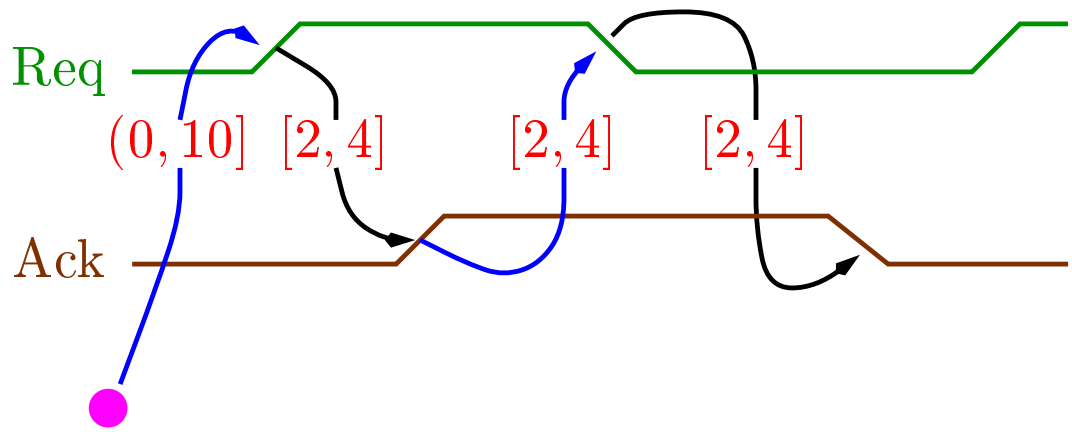
The Timed Version

We add interval timing constraints ($[low, high]$) to the edges, and distinguish between strong (guarantee) and weak (assume) edges.

We may specify the reactive module



by

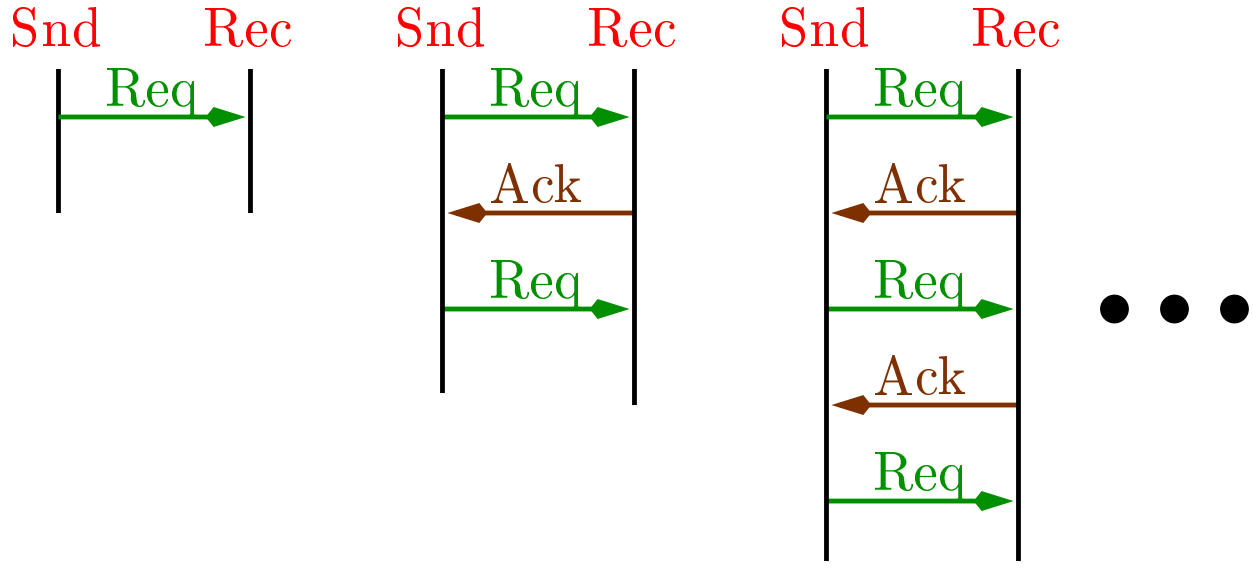


The first constraint can be temporally specified by

$$T = 0 \quad \rightarrow \quad (\overline{\mathsf{Req}} \wedge \overline{\mathsf{Ack}})\, \mathcal{U}\, (\mathsf{Req} \ \wedge \ \overline{\mathsf{Ack}} \ \wedge \ 0 < T \leq 10)$$

Message Sequence Charts (MSC)

A notation widely used for specifying use-cases in object-oriented methodologies, recently standardized as part of the UML standard. For example,

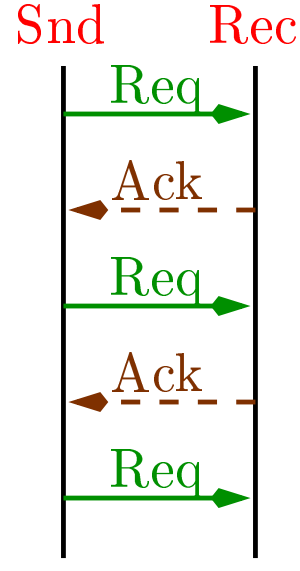


The problem with this specification is that it is completely existential. It only states that the system should at least have these scenarios. What about the missing ones? Are they forbidden or have they just been omitted?

This has the expressive power of CTL without the **A** path quantifier.

Live Sequences Charts (LSC)

Damm and Harel [99] extended MSCs into LSCs, adding liveness and universality, attaining the expressive power of CTL*.

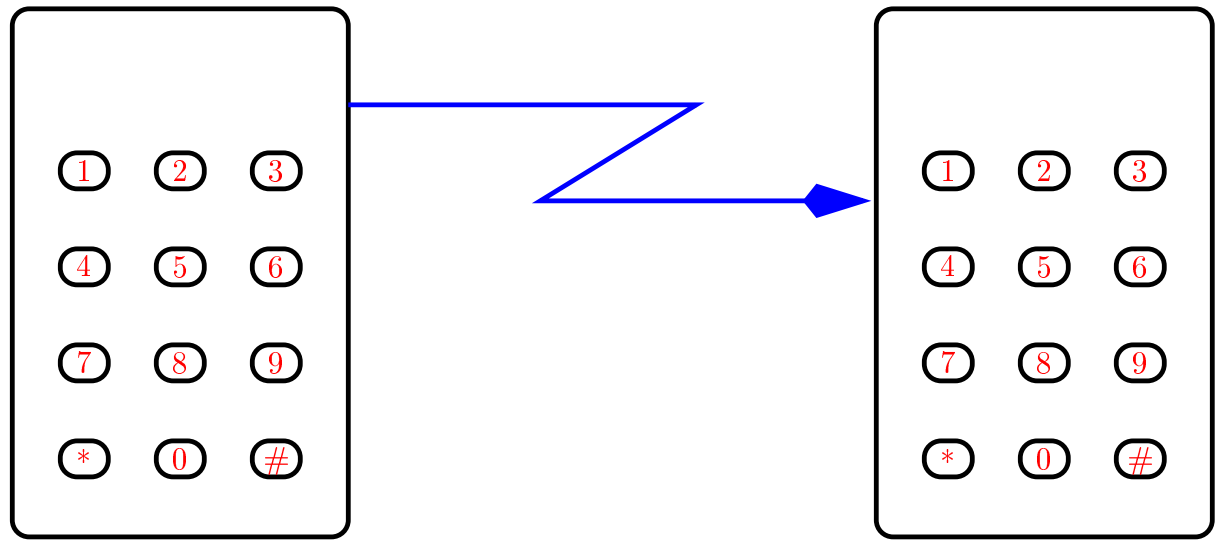


Identifying the Ack-edges as weak (cold) edges which allow breaking the scenario at any such edge but nowhere else.

Fantasizing Even Further

The ultimate dream of requirement elucidation may be based on a rapid prototyping system which will construct a virtual mockup of the driven system with the appropriate interface. The user will create scenarios (desired and undesired) by pushing the right buttons and manipulating the virtual model.

This will be gathered into am assembly of requirements, analyzed for consistency, serve as a base for user investigation, and finally translated into a specification.



Please attend Harel's lecture, where he will explain how he has implemented his vision of such a system by the Play-in/Play-out Engine.

Verification

A comprehensive approach to the formal verification of embedded systems can be based on a compositional verification methodology consisting of

- Unit verification performed by model checking technologies, plus
- System verification performed by deductive and theorem proving techniques.

Such a prototype system has been developed for the *Sacres* project by the OFFIS group, and is currently been adopted by Ilogix, the developer of Statemate.

Early Analysis of Resource Utilization

The essence and main advantages of the extended development process are early awareness and early warning:

Confront potenetial hazards as early as possible!

In embedded systems, where the utilization of resources such as real time, power consumption, memory requirements are crucial, it is important to analyze (and verify) the satisfaction of resource-related constraints at an early phase.

To do so, we must

- Formulate adequate models for the analysis of these constraints at the specification level. For timing constraints we may use models such as timed automata. For power consumption and memory utilization, this is still an open problem. Possibly, we may employ models such as hybrid systems.
- Apply the principle of continuity in the bottom-up direction, deriving data for the formal models from analysis at the implementation level, by methods such as profiling.

All of these are important problems looking for the right researchers.

Extending the Scope of the Translation Validation Techniques

Naturally, when the development process is extended upwards by adding the requirement layer (layer 0) and downwards, adding the compilation into hardware layer (layer 2), it would be necessary to extend the scope of the translation validation techniques to cover these two far ends.

This raises many interesting questions and challenges to the presently used technique. We are currently working on such extensions by constructing a translation validation tool for optimizing EPIC-targeted compilers, such as the SGI-64 compiler.

Let us review the translation validation approach as it was implemented in the *Sacres* project.

Translator vs. Translation Validation

Rather than verify the translator itself (using compiler verification techniques), we chose to verify the results of each run of the translator.

Constructed the CVT tool, which can be invoked after every run of the code generator, translating a DC^+ specification D_1 into a C program C_1. Applied to (D_1, C_1), CVT verifies automatically that C_1 is a correct implementation of D_1 or identifies a bug in the translator.

Advantages of the translation validation approach are:

- Evolution of the translator is not frozen after verification.
- Significantly cheaper than translator verification.
- Completely oblivious to the assumptions intrinsic to the translator. Provides independent cross check.

The only drawback: constant run-time additional cost, can also be justified.

Sacres Simpler Situation: A Synchronous Source Language

This makes the application proceed in well-defined steps, as follows:

loop forever do
[**read inputs**
compute
write outputs]

A Statemate Behavior

loop forever do
[**compute**; **read inputs**
. . .
compute; **read inputs**
compute
write outputs]

A Signal/Sildex Behavior

Consequently, the generated C program has also only a single external loop. To verify correct implementation it suffices to prove an implication

$$\rho_C \quad \rightarrow \quad \rho_S,$$

where ρ_S is the transition relation of a single source step, while ρ_C is a (compressed) transition relation representing the state change effected by a single execution of the C-program loop's body.

Validation of Optimizing Compilers

To deal with hardware-targeted compilers, we have to generalize the code validation techniques.

Structure preserving translations:

First we deal with the case that the target and source can be represented as a directed graph of basic blocks and there exists a mapping κ from target control locations to source locations. The method is based on simulation as follows:

For each basic block `Bi` and `Bj` such that `Bj` is a successor of `Bi` in the IR-graph, validate the condition

$$\varphi_i \ \wedge \ \alpha \ \wedge \ \overset{T}{\rho}_{ij} \ \rightarrow \ \exists V_S' : \overset{S}{\rho}_{\kappa(i),\kappa(j)} \ \wedge \ \alpha' \ \wedge \ \varphi_j'$$

Loop Transformations:

For all other optimizations, we devise special loop transformation rules. For example, consider the two programs:

$$P_{fus} :: \begin{bmatrix} \texttt{for(i = 1; i <= N; i++)} \\ \quad \texttt{B1(i); B2(i);} \end{bmatrix} \qquad P_{dis} :: \begin{bmatrix} \texttt{for(i = 1; i <= N; i++)B1(i);} \\ \texttt{for(i = 1; i <= N; i++)B2(i);} \end{bmatrix}$$

The corresponding rule for such a transformation is give by:

$$\frac{i_0 < j_0 \quad \longrightarrow \quad \texttt{B2}(i_0);\texttt{B1}(j_0) \ \approx \ \texttt{B1}(j_0);\texttt{B2}(i_0)}{P_{fus} \ \approx \ P_{dis}}$$

As part of our work, we managed to discover a bug in the Trimaran compiler.

In Summary

- Without any doubt, we are entering a new era in which embedded systems will occupy a central and prominent place. There will be a great pressure to improve our capabilities for the quick and reliable construction and deployment of embedded systems.

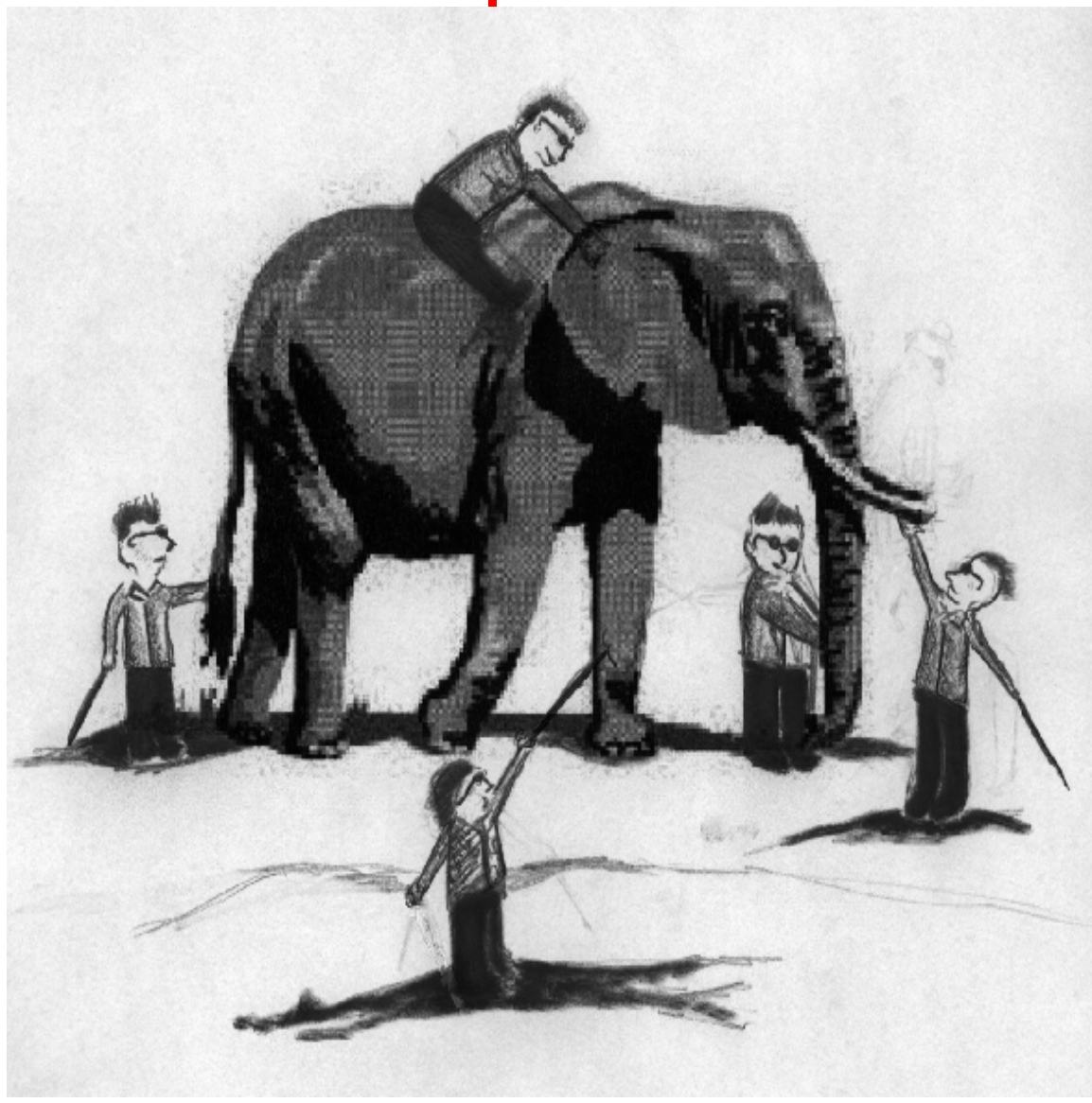
- There are many currently existing techniques which propose partial solutions to the main problems intrinsic to embedded systems.

- To promote the progress of the field, it is necessary to enhance and strengthen the existing techniques, possibly develop additional ones and, most importantly, integrate all of these tools into a seamless development methodology.

- Specifically, we should elevate the entry level and, possibly, consider requirement based analysis and development.

- At the other end, we should relax some of the predictability excused straight jacket constraints and use more powerful rigorous analysis techniqes in order to obtain better efficiency building upon modern software/hardware technology.

Reinterpret Picture: Cooperation Instead of Confusion



Experts, each in their own field, cooperate to achieve a better integrated methodology for the development of embedded systems.