An overview of Rapid System Prototyping today

F. Kordon[‡] & J. Henkel[§]

1. The role of Rapid System Prototyping

The International Technology Roadmap for Semiconductors [3] predicts chip complexities by the end of this decade silicon of around 1 billion transistors integrated on a single piece of silicon. It will open new frontiers in terms of applications in areas/devices ranging from security systems (e.g. video surveillance), control systems (e.g. automotive control), individual health systems (e.g. hearing aids) to main stream consumer products in such areas as personal communication (e.g. cell phones), personal computing (e.g. PDA), entertainment (e.g. MP3 players), video/photo (e.g. digital still/video cameras) and many more.

It can be observed that especially the latter group of main stream consumer products is subject to severe competition between major manufacturers and thus leading to two effects from a consumer's point of view:

1. The life cycles of embedded products become increasingly smaller: cell phones represent one of many examples for this trend since they experience the emergence of an average of two new major product lines every year compared to only one years ago.

2. The functionality of these products and hence the complexity of the underlying embedded systems is rapidly increasing: staying with cell phones as an example, the observation is that they feature far more functionality beyond their core functionality of providing a wireless voice channel and thus establishing a phone connection. In fact, common functions of cell phones include web browsing capabilities, SMS (Short Message Service), PDA functionalities and even gaming etc. Almost on a monthly basis there are new features manufacturers of cell phones and service providers announce and eventually integrate into new generations.

Similar scenarios could be discussed in conjunction with other consumer, control etc. devices as well. The observation would often be an increasing portfolio of functionality combined with decreasing product life cycles.

[‡] LIP6-SRC, Université P. & M. Curie, 4 place Jussieu, 75252 Paris cedex 05, France, Fabrice.Kordon@lip6.fr

[§] NEC Laboratories America, Princeton, NJ, USA, henkel@nec-labs.com

The consumer's demand for increasing functionality translates directly into increased complexity of embedded systems on a chip. Indeed, this demand matches well with predictions from the International Technology Roadmap for Semiconductors as pointed out above.

However, already today it can be observed that the maximum possible amount of transistors (silicon-technology-wise) per chip is hardly exploited. This fact is even more obvious when the number of transistors per SOC *without* excluding the embedded memory (ie. easy to design regular structures) is not counted.

As a conclusion, real-world SOCs' complexities currently lag behind the capabilities of current silicon technologies even though there is certainly a demand for higher complexities (i.e. increased functionality) from an application point of view as discussed before.

G. Smith [12] gives a possible answer: the reason is the so-called "gap of complexity". It is measured as the number of available gates per chip for a given silicon technology for SOCs on the one side and the number of actually used gates per chip of a given silicon technology on the other side. The gap is predicted for the case that there will be *no* ESL (Electronic System Level Design) methodologies deployed for designing future complex SOCs. In other words: the gap might be avoided if more ESL methodologies would be deployed in all areas of system level design like specification/modeling, synthesis, simulation/verification and estimation, etc.

We believe that Rapid System prototyping can play a key role in alleviating these problems as it represents a means to rapidly explore design alternatives and to unveil design errors as early as possible what is critical in designing high quality products within short time-to-market windows. Rapid system prototyping becomes critical when new areas in software or hardware development are explored because of new technological issues. It should also be mentioned that on-chip systems cannot be designed as synchronous anymore because the involved high frequencies (up to 10GHz by the end of the decade) make signal propagation from one point to another virtually impossible because of the clock skew problem.

Similar observations are made for 100% software systems [5], especially in new areas such as distributed or embedded systems where new standards or products arise frequently: the community is facing new problems and issues that remain difficult to handle, especially for system or mission critical applications [1]. It is therefore mandatory to rapidly explore design alternatives at various levels of abstraction. Methodologies and architectures proposed in the Rapid System Prototyping community will be a significant source of intellectual property to help solving these upcoming challemges.

2. Diversity of Prototypes

A prototype is any form of specification or implementation of hardware or software (or both) that is built/designed for evaluation purposes. When building a prototype, a designer has typically in mind to design multiple copies once the design is sufficiently evaluated. All prototypes have in common that they are *executable*. Prototypes are also useful for formulating and validating requirements, resolving technical design issues, and supporting computer aided design of both software and hardware components of future designs.

Rapid prototyping refers to the capability to implement a prototype with significantly less effort than it takes to produce an implementation for operational use. It is a way to: collect data and feedback for changing requirements, unveil deviations from users' constraints early, trace the evolution of the requirements, improve the communication and integration of the users and the development personnel, provide early feedback of mismatches between proposed software architectures and the conceptual structure of requirements etc.

Prototypes can either be developed as temporarily designs that are not being used after some evaluations have provided the designer with valuable insights, or they can directly evolve into a product version of the respective design. Each of these approaches has its advantages and disadvantages. It will depend on the designer's constraints which type is the best for a certain project.

THROW-AWAY PROTOTYPES

A *throw-away prototype* is discarded when sufficient information has been obtained during the operational phase of the prototyping phase. Throw-away prototypes are mainly used during the requirement phase (to agree on what have to be implemented). It is quite rarely used, though, during the implementation phase as a way to explore capabilities of a candidate technology before using it on a real scale.

The main advantage is to enable the use of special-purpose languages and tools even if they introduce limitations that would not be acceptable in an operational environment or even if they are not capable of addressing the entire problem. The throw-away approach is most appropriate in the project acquisition phase where the prototype is used to demonstrate the feasibility of a new concept, and to convince a potential sponsor to fund a proposed development project. In such a context, available resources are limited and the ability to communicate the advantages of a new approach via a very low cost demonstration can be critical for establishing a new project.

However, the implementation effort does not contribute directly to the final product. There is also the temptation to skip or abbreviate documen-

tation for throw-away code, which is harmful since lessons learned from the prototyping effort may be lost if not recorded. It also lets unsolved the gap between specification (built using lessons learned from the prototype) and implementation that brings misunderstanding, intrusion of implementation choices, etc.

INCREMENTAL PROTOTYPES

Incremental prototypes can be seen as a ”clever development approach”. The prototype tends to be closer and closer to the final product when development progresses.

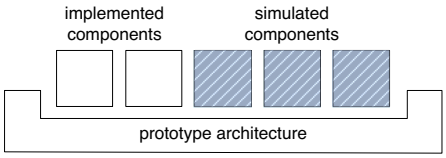

Figure 1. Structure of an incremental prototype.

Figure 1 shows the structure of such a prototype. It is usually based on a ”prototype architecture” that is defined as soon as possible in the development process (usually before 20% of the total estimated effort is spent). This prototype architecture can be seen as an integration to host successive components to be implemented. At a time, there are implemented components and simulated components. When no simulated component remains, the prototype is the first version of a new product.

The main advantage of incremental prototyping is that it does not require more than methodological rules to be operated. Its main drawback is that it entirely relies on the robustness of the prototype architecture.

EVOLUTIONARY PROTOTYPES

When in Incremental prototypes, the model is also the system, evolutionary prototypes do make a difference between model and system (Figure 2).

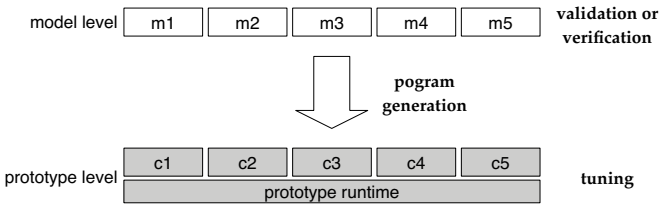

Figure 2. Structure of an evolutionary prototype.

The model is a detailed (executable or formal) specification on which validation (by simulation for example) or verification (using formal tech-

niques) can be applied. Then components are derived from specification modules and linked together using a ”prototype runtime” that acts as glue code. This approach is of interest when many loops between the model and its implementation can be performed and thus, automatic code generation is required.

The main advantage of this approach is to fill the gap between detailed specification and the implemented system. It is also more flexible since the prototype's architecture may change radically from one version to the next one (code is automatically generated). However, evolutionary prototyping heavily depends on sophisticated tools and techniques. The technology needed to support evolutionary prototyping is beginning to emerge.

TOWARDS BODEL-BASED DEVELOPMENT

As previous sections show, the prototyping techniques become more and more complex and must be supported by methodologies to provide superior results. At this stage, prototyping is much more than simply a way to develop a new system. In fact, it becomes a methodological approach for system development [4]. Rapid System Prototyping should not be confused with RAD (Rapid Application Development) or ”Extreme Programming”. Prototyping has nothing to do with these techniques that mainly focus on short-term issues: the obtained code is rarely maintainable. Evolutionary prototyping does exist in hardware, software and mixed hardware/software systems.

It appeared first in the context of hardware systems since the time-to-market pressure does not support ”maintenance” the way software systems do: a flawed chip is never sold and the initial cost of developing the mask is lost. CAD tools do support models (based on the VHDL language for example) and code generation. Since software simulation is too slow, ”soft-hardware techniques” such as FPGA allow the quick elaboration of a prototype that can be used for a higher performance evaluation of the system. This is typically an evolutionary approach that eventually leads to the final implementation.

Software systems suffer from the ”maintenance” market and prototyping techniques so far are mainly in niche markets, where safety is a key issue (e.g. life or mission critical systems) or when maintenance is very difficult (e.g. satellite systems). In those areas, it is of interest to look at the new MDA (Model Driven Approach) proposed by OMG [9]. MDA states that PIM (Platform Independant Model) has to be elaborated first in order to establish the functional view on the system. Then, PSM (Platform Specific Model) is derived. Code generation is derived from the last one. Once again, this is evolutionary prototyping. UML [10], the current standard, is too comprehensive (on the contrary, VHDL is domain-centered). If results are impressive for information systems, more work and experience is necessary to use it in

other areas such as distributed systems [7]. This is also true for real-time or embedded applications. For these types of applications, UML remains a (good) modeling language dedicated to early design only, even if the last UML release [8] brings new interesting features (but does not yet address behavioral aspects).

The lesson we can extract from this evolution is that, sooner or later, ”traditional” programming languages will not be used anymore. This is already the case in hardware (the gate level is not directly handled anymore). Since the objective is to fit the gap between specification and implementation, the next generation of languages will be at model-level, and development will be *model-based* [11].

These new notations will most probably integrate ”implementation directives” (such as the ones found in Architecture Definition Languages). Specification will then become ”modeling”: an operation where all aspects (static and behavioral) of a system are considered and code will be automatically generated automatically. As for years, the semantic level of programming languages will grow.

3. The demand for Rapid System Prototyping

Some large companies have already adopted prototyping as a way to solve these problems. In some areas such as avionics, there is a long experience in prototyping-based development techniques using very elaborated tools and notations. So far, the motivation of industry in these new techniques can be separated in types of interest:

- ***level 1***: prototyping reduces cost and time-to-market of a system.

  For companies producing complex systems (such as embedded, distributed, real-time, etc.), there is an additional reason: the cost of highly skilled engineers increases rapidly since there is more demand than people to fill positions. Automated development approaches could reduce the need for highly skilled engineers since one of them could manage several ”standard” engineers to operate prototyping tools.

- ***level 2***: prototyping increases security and reliability of a system.

  For companies building safety critical systems, a prototype-based approach is even more interesting since it is more likely able to operate formal verification techniques when required. It is now clear that such methods are the only way to provide extremely high levels of reliability in system design and implementation. This is why it is recommended by various certification standards such as DO-178B (for avionic systems).

So far, advanced prototyping is used by industries considering the level 2 (in both hardware and software, even if formal techniques are not yet ready for a wide use [6]). The main reason is that high investments have to be done before these techniques are usable at low costs. As an example, the cost of the so expensive code generator certification (DO-178B) in the SCADE tool is leading to discussions on how certification could evolve to consider the use of these new techniques at a lower cost (and similar reliability) [2].

However, as soon as these techniques will be operational, candidates considering the level 1 will considerably increase the market for prototyping methods and tools. The fact that industrial consortiums and organisations are already looking at these technologies is a sign that many changes will occur in that area.

4. About the Special Issue

The thirteenth IEEE International Workshop on Rapid System Prototyping (RSP) presented and explored recent trends in rapid prototyping of Computer Based Systems. It was hosted by the Technical University of Darmstadt, in Darmstadt, on July 1-3, 2002.

A major intend of the RSP workshop series is to bring together researchers from both hardware and software communities to share their experience with rapid prototyping and related areas. It is of particular interest to see that, despite very different techniques and constraints, how close objectives and methodologies can be.

For RSP'2002, 22 contributions were accepted out of 47 submitted papers. The top twelve were nominated by the program committee as representing especially innovative contributions and the authors were invited to extend them for publication in Kluwer's Design and Automation for Embedded Systems Journal. We received eventually nine papers that represented significantly enhanced versions of the initially published work at RSP 2002. Through an extensive review process we selected finally four papers for this special issue.

We would like to take this opportunity to thank all reviewers for their thoughtful reviews that helped us to make this special issue possible. Our thanks also go to the staff at Kluwer Academic Publishers who were very helpful in both organizational and editorial aspects of this issue.

4.1. Content of this Special Issue

The papers in this special issue have been selected to showcase the wide variety of rapid prototyping architectures/methodologies in both, the hardware and software domain.

Starting with the software domain, Chachkov et al. introduce their approach to modeling a software system by using a formal specification language. The authors use a library-based approach as it reflects the way software systems are typically designed — by extensively re-using software components from a software library. Their specification language CO-OPN is initially used for specification and then a prototype code is automatically generated from that. The specific emphasis of their paper is on interfacing between non-deterministic synchronous prototypes and deterministic asynchronous libraries.

A hardware-architectural approach is proposed by Jain et al. whose architecture is based on a multi-FPGA board. Multi-FPGA boards are necessary to enable the prototyping of complex systems that could not be prototyped on a single FPGA. Interconnect typically represents the major problem of multi-FPGA boards: an interconnect can be programmable or fixed with programmable ones offering a high flexibility at the cost of delay. The authors present a solution to minimize programmable interconnects by multi-hop routing (through the FPGAs). They present an optimal and heuristic solution to the problem.

An application for rapid system prototyping is presented by Piontek et al. by means of an equalizer for orthogonal frequency division multiplexing (OFMD). The authors use the FPGA prototype to conduct a trade-off analysis between complexity on the one side and performance of the other side and thereby simulating various compensation methods for channel distortion. Their hardware simulations have been conducted according to the wireless high-speed LAN standard HipeLAN/2.

In an approach based on the OFDM prototype from above, Ludewig et al. propose a technique to gather signal activity for power-conscious system using a hardware-assisted approach. The basic idea is to use the hardware prototype to observe the signal characteristics on a running system over a long period of time. The observed activity is then used to estimate the power consumption stemming from the interconnect and to refine the interconnect if necessary. Thereby, the authors take into also into consideration the signal activities due to interwire effects.

We hope you enjoy this special issue.

References

1. T. Budden. Decision Point: Using a COTS component help or Hinder Your DO178-B certification Effort? *CrossTalk*, pages 18–21, November 2003.
2. J. Engblom. ARTIST International Collaboration Day on Embedded Software and Systems. Technical report, ARTIST, october 2002.
3. ITRS. *International Technology Roadmap for Semiconductors*. Semiconductor Industry Association, 2002.

4. F. Kordon and Luqi. An introduction to Rapid System Prototyping. *IEEE Transaction on Software Engineering*, 28(9):817–821, September 2002.
5. N. Leveson. Software engineering: Stretching the limits of complexity. *Communications of the ACM*, 40(2):129–131, 1997.
6. Luqi and J. Goguen. Formal methods: Promises and problems. *IEEE Software*, 14(1):73–85, January / February 1997.
7. N. Medvidovic and R. Taylor. A classification and comparison framework for software architecture description languages. *Software Engineering*, 26(1):70–93, 2000.
8. OMG. Initial Submission to OMG RFP's: ad/00-09-01 (UML 2.0 Infrastructure) ad/00-09-03 (UML 2.0 OCL). Technical report, OMG, 2001.
9. OMG. Model Driven Architecture (MDA), Document number ormsc/2001-07-01. Technical report, OMG, 2001.
10. OMG. OMG Unified Modeling Language Specification, version 1.5. Technical report, OMG, 2001.
11. D Quartel, M van Sinderen, and L. Ferreira Pires. A model-based approach to service creation. In *Seventh Workshop on Future Trends of Distributed Computing Systems*. IEEE, 1999.
12. G. Smith. DAC Panel Presentation. In *40th. Design Automation Conference (DAC)*. IEEE, 2003.