

# AUTONOMOUS COMPUTING SYSTEMS: A PROPOSED ROADMAP

*Neil Steiner and Peter Athanas*

Bradley Department of Electrical and Computer Engineering  
Virginia Polytechnic Institute and State University  
Blacksburg, VA 24061  
email: neil.steiner@vt.edu, athanas@vt.edu

## ABSTRACT

An *autonomous computing system* is a system that functions with a large degree of independence, and assumes a large amount of responsibility for its own resources and operation. As a counterpart to ongoing research in the software domain, this work proposes a forward-looking roadmap for systems that are able to autonomously modify their hardware, and considers what properties such systems require. Of particular interest is the possibility of shifting much of the associated complexity into the systems themselves. This results in simpler interfaces to the outside world, and systems that are able to respond to changes in themselves or their environments with little or no outside intervention.

## 1. INTRODUCTION

One may informally describe an *autonomous computing system* as a system that functions with a large degree of independence, much as a living system does. Stated differently, an autonomous system assumes a large amount of responsibility for itself, and for its internal functioning. This work looks at the infrastructure that is necessary to support hardware autonomy—the ability of a system to change its own hardware—and is strongly influenced by FPGAs, the only mainstream technology to currently support dynamic reconfiguration. Corresponding endeavors in the software domain are larger and more mature [1, 2], but generally use the term *hardware* only to denote commodity desktop computers that are unable to change themselves.

Autonomy is desirable in part because internal and external conditions do change in real systems, and when they do it is desirable for the systems to adjust gracefully. Conversely, the failure to appropriately adjust to changes accelerates system obsolescence. The existence of software operating systems means that software updates can usually be applied without rebuilding an entire system, but that kind of convenience is only now emerging in the hardware domain [3]. Even in the absence of changes, two identical sys-

tems may be placed in very dissimilar environments, which means that any measure of adaptability is still desirable.

There are additional benefits however. A system that assumes more responsibility for itself frees the designer to work at a higher level of abstraction, focusing on smaller self-contained components rather than on system infrastructure. Higher levels of abstraction also make designs more portable to other devices and architectures. And a system's ability to manage its own resources makes it better able to tolerate defects. As system complexity increases, the systems become more difficult to manage externally, and at a sufficiently elevated point on the complexity curve, some form of autonomy becomes a necessity.

Autonomy comes at a cost, and is therefore only acceptable in systems that can justify the required overhead. Similar issues arose in the software domain, with assemblers and compilers and operating systems and object oriented languages at one time or another, but these tools are used extensively today and their benefits are unquestioned.

The ideas presented here bear some similarity to other work. Williams and Bergmann discuss the control, dynamics, and granularity of reconfiguration in embedded systems [4]. Kubisch et al discuss partial dynamic reconfiguration of systems that observe themselves and their environments [5, *see unpublished full version*]. Other groups are less interested in dynamic reconfiguration, but still desire rapid tools to support what they call “Just-in-Time FPGA Compilation” [6] with the portability that it affords. These various approaches typically evaluate available tools and architectures, but find them lacking for the intended purpose, and are forced to propose alternate configurable architectures. The tool and architectural limitations that waylay these projects serve to further motivate the implementation phase of the current work, which attempts to push the limits of what is feasible at present.

## 2. ROADMAP

The steps required to build an autonomous computing system are fairly logical, and are presently within reach in one

---

This work is graciously supported through a Bradley Fellowship from the Via Foundation.

Level	Requirements	Capabilities
0	Configuration port	Reconfigurability under external control
1	Allocation map Primitive router	Partial relocatable reconfigurability under internal control from pre-defined configuration bitstreams
2	Internal model Placer Router Configuration generator	Partial placeable reconfigurability under internal control from technology mapped netlists
3	Synthesizer Mapper	Partial placeable reconfigurability under internal control from technology-independent behavioral descriptions
4	Monitor Sensors Prioritizer	Ability to detect and monitor conditions within the system or within its environment, and to prioritize the conditions in some manner
5	Response library Evaluator Simulator (optional)	Ability to scan its library for possible responses to detected conditions, and to evaluate the suitability of those responses, and possibly simulate them
6	Implementer Requester	Ability to implement a response by synthesizing it and reconfiguring itself, or ability to request assistance from other systems or libraries
7	Inference engine	Ability to infer the required behavior from partial knowledge of conditions, and/or to assemble the behavior out of known primitives
8	Assessor	Ability to evaluate the implemented response, and to update or optimize the response library accordingly, learning from experience and mistakes

**Fig. 1.** Levels of autonomy

form or another, though most have not yet been implemented inside of the systems that they target. This work presents a hierarchy where each level builds upon previous levels, such that a system at a given level includes the requirements and capabilities of all previous levels. The resulting roadmap is shown in Figure 1.

### 2.1. Level 0

Level 0 systems are not autonomous. They are unable to adapt to changing conditions in themselves or in their environment. While the designer may be able to update them, and perhaps even do so remotely, the systems are entirely passive with respect to the update. They do not know what is happening, and they have no control over it.

*Scenario:* An FPGA device is remotely reconfigured with a full or partial bitstream.

*Requirements:* A configuration port.

Full configurations are only acceptable for Level 0 systems, and are antithetical to autonomous systems.

### 2.2. Level 1

Level 1 systems play a minor role in their own reconfigurations. They have some notion of their own utilization and free space, typically divided into slots. They have the ability to accept a partial configuration, to decide which slot to place it in, and to make simple connections that may be required. The configuration may have been provided remotely or retrieved by the system from its own storage.

*Scenario:* A device reconfigures itself with a partial bitstream.

*Requirements:* An allocation map and a primitive router.

### 2.3. Level 2

Level 2 systems maintain an internal model of themselves that is more sophisticated than the slots of the Level 1 allocation map. Instead of blindly accepting a pre-generated configuration, they accept netlists that are compatible with their architecture. Level 2 systems are responsible for placing and routing the netlists, and for updating their internal

model. They have much more control over their own use of resources, and the netlists that they accept are likely to be compatible across an entire family of devices, instead of only being compatible with one specific device.

**Scenario:** *A device extends itself by instantiating a netlist.*

**Requirements:** *An internal model, a placer, a router, and a configuration generator.*

The quality of the placer and router will significantly impact the performance of the updated system. Much research and development has gone into the routing and placing tools used by the industry, but autonomy requires that these tools now be embedded inside the systems that they target. It is worth noting that these tools perform *incremental* placing and routing in the unused subset of the system's resources.

### 2.4. Level 3

Level 3 systems accept operations or behaviors and synthesize and map them into netlists. They are responsible for deciding where to place components and how to connect them, as well as how to implement those components, possibly depending on current system resources and conditions.

The abstract hardware descriptions are suitable for almost any target device, and not just a specific technology family. This progression leads to greater internal complexity, but also allows greater external simplicity and generality.

**Scenario:** *A device extends itself by implementing a component from a hardware model.*

**Requirements:** *A synthesizer and a technology mapper.*

The versatility of abstract or generic components allows systems to model themselves in terms of behavior if that proves to be useful, and paves the way for them to describe themselves to heterogeneous peers. The functionality begins to take a predominant role, and that functionality can theoretically be implemented in any system that has available resources.

### 2.5. Level 4

Level 4 systems have the ability to detect and monitor conditions of interest. These systems require some kind of centralized or distributed controller that can observe or receive notification of changing conditions. The systems may include a variety of distributed sensors, and may also include a prioritizer or classifier that assigns relative levels of importance to conditions that are detected.

**Scenario:** *A system notices that the bit error rate on some incoming link has risen above its norm.*

**Requirements:** *A monitor, a collection of sensors, and a prioritizer.*

Level 4 systems are able to detect changing conditions without necessarily being able to respond to them. This abil-

ity may not appear to be especially useful, but it plays an important semantic role in autonomy, because it motivates change. Changes in living systems are not haphazard, but are instead purposeful and directed, which means that living systems change according to internal or external goals and conditions.

Just as with living systems, conditions to be monitored may be internal or external to the system. It may be just as important to know that an external link is about to be lost as it is to know that an internal FIFO is nearly full, and determining the severity of the conditions may depend partly on whether the two are correlated. In addition, large systems are probably best served by distributed sensors and monitoring, similar to what one encounters in biological systems. The sensors are the "eyes and ears" of the system, because they allow it to observe itself and its environment.

### 2.6. Level 5

Level 5 systems include a collection of responses that can be applied to address conditions that are observed, as well as some kind of evaluator to help determine which responses might be applicable. These systems merely look for candidate responses, and do not necessarily have the ability to apply the responses.

**Scenario:** *The amount of data coming in from a certain link is approaching the system's ability to process it, and the system looks for an appropriate solution. Possible solutions include dropping packets, forwarding some requests to a neighboring system, reimplementing certain functions with faster circuits at the expense of added area, duplicating the existing circuit and splitting the data between them, informing the sender of the impending problem, running statistical analyses to estimate whether the incoming rate is a temporary surge or a sustained increase, monitoring the situation more closely but doing nothing, and so on.*

**Requirements:** *A response library and response evaluator, and possibly a simulator.*

A Level 5 system may or may not find a suitable response in its library, and it uses the evaluator to help make that determination. In some cases it may be desirable to include a simulator that can simulate candidate responses, in order to determine which ones will in fact resolve the problem, so that the evaluator can compare their costs and performances. Because Level 3 and higher systems can synthesize operations and behaviors, the response library can be encoded as a collection of abstract behaviors.

### 2.7. Level 6

Level 6 systems have the ability to apply responses or request assistance. This constitutes the first stage of real autonomy. If a suitable response has been identified, the system can apply it by synthesizing and implementing the cor-

responding behavior, thereby autonomously initiating and completing a modification to itself. If no suitable response is identified, the system can instead query neighbors or peers or supervisors or other resources to see if any of them can supply a suitable response.

**Scenario:** *A system implements a more robust filter to compensate for a deteriorating signal-to-noise ratio.*

**Requirements:** *An implementer and a requester or notifier.*

Although it may seem strange to consider a request for assistance to be a demonstration of autonomy, biological interactions include many examples of requests for assistance large or small, and in particular, people use this approach extensively. A system that asks for help may actually be able to *learn* by adding to its knowledge, much as a student can learn from a teacher or a tutor or a peer or a book. Requests could be sent to neighbors, to supervisors if any, to known external resources similar to public libraries—*why not opencores.org?*—or to the system designer.

## 2.8. Level 7

Level 7 systems have the ability to infer the required behavior from detected conditions, and/or to adapt existing behavioral pieces to the conditions. This level of system falls unambiguously into the artificial intelligence domain: it knows how to make inferences, and it knows how to dissect known behaviors and piece them back together as something that it had no previous knowledge of, in the manner of theorem proving systems. This is a clear case of machine learning.

**Scenario:** *A system determines that it can build a necessary behavior out of smaller existing components, and it implements the new behavior.*

**Requirements:** *An inference engine.*

If the system cannot identify a suitable response, but knows enough about the problem, it may try to infer the function on its own or piece it together out of other components, possibly simulating the response before implementing it.

## 2.9. Level 8

Level 8 systems can analyze the results of implementing changes, and can determine whether their internal response libraries should be updated or optimized for future use. If one assumes a strong artificial intelligence capability, then it is possible to envision a system that learns from experience and mistakes: it adds to its knowledge when applicable, and possibly corrects or replaces previously held incorrect or incomplete knowledge.

**Scenario:** *Having reimplemented some of its functionality for performance reasons, a system analyzes the results and makes adjustments to its response library.*

**Requirements:** *An assessor.*

## 3. CONCLUSION

Autonomous computing systems offer a number of important benefits, including 1) reduced dependence upon external management, 2) simpler interfaces to the outside world, 3) increased portability of designs, 4) simplified defect tolerance, and 5) increased adaptability and customizability.

Autonomy should be especially attractive when systems are large, difficult to access, and subject to defects, or when the problems or environments are complex, non-static, and heterogeneous. The ability to accommodate these conditions may initially make autonomous systems attractive for space, exploration, remote sensing, and military applications. In time, these benefits may also prompt adoption by the commercial sector, for the same reasons that operating systems have made their way into embedded devices.

No comparable roadmap is known to exist for autonomous hardware, and few of the required components have been previously brought together as proposed here, for the sake of facilitating systems changing their own hardware.

The rather abstract ideas presented in this paper have been demonstrated for the first time in a proof-of-concept Level 2 system, an incremental addition to the state-of-the-art. The system interactively accepts mapped EDIF circuit descriptions that it parses, places, routes, connects, and instantiates inside itself while continuing to run.

## 4. REFERENCES

- [1] Alan G. Ganek and Thomas A. Corbi. The dawning of the autonomic computing era. *IBM Systems Journal*, 42(1), 2003.
- [2] DARPA Information Processing Technology Office (IPTO). <http://www.darpa.gov/ipto>.
- [3] Cindy Kao. Benefits of partial reconfiguration. *Xcell*, 4Q05(55):65–67, 2005.
- [4] John A. Williams and Neil W. Bergmann. Embedded linux as a platform for dynamically self-reconfiguring systems-on-chip. In *Proceedings of the 2004 International Conference on Engineering of Reconfigurable Systems and Algorithms, ERSA 2004, June 21–24*.
- [5] Stephan Kubisch, Ronald Hecht, and Dirk Timmermann. Design flow on a chip - an evolvable HW/SW platform. In *Proceedings of the 2nd IEEE International Conference on Autonomic Computing, ICAC 2005, June 13–16*.
- [6] Roman Lysecky, Frank Vahid, and Sheldon X.-D. Tan. A study of the scalability of on-chip routing for just-in-time FPGA compilation. In *Proceedings of the 13th IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 2005, April 18–20*.