Prototyping Resilient Processing Cores in Workcraft

Georgy Lukyanov[†], Alessandro de Gennaro[‡], Andrey Mokhov[‡], Paulius Stankaitis[‡], Maxim Rykunov[§]
[†]Southern Federal University, Rostov-on-Don, Russia
[‡]Newcastle University, Newcastle upon Tyne, UK
[§]IMEC, Leuven, Belgium

***Abstract*—We present a methodology for the design and fast prototyping of processing cores with resilient microarchitecture. The resilience is achieved by equipping the core with a family of datapath components optimised for different operating modes and a flexible control structure that allows to change an instruction implementation at runtime depending on current conditions and application requirements. We use asynchronous design techniques to achieve *short-term resilience*, i.e. survival in extreme environmental conditions, such as near-threshold or unstable voltage supply. *Long-term resilience* is achieved through runtime reconfiguration of the processor microarchitecture, which is essential for safety-critical applications that cannot be taken offline for maintenance, such as biomedical implants. By using formal methods one can guarantee the correctness and uninterrupted service during such runtime reconfigurations.**

The presented methodology is supported by open-source tool WORKCRAFT, and has been validated by fabricating two ASICs: an Intel 8051 processing core and a reconfigurable dataflow accelerator. To facilitate fast prototyping of resilient processing cores, we introduce a domain-specific language for their formal specification, software-level simulation and hardware synthesis.

I. FORMAL METHODS FOR RESILIENT SYSTEMS

Many resilient systems rely on runtime reconfigurability to adapt to continuously changing environment without any human intervention. For example, biomedical implants must be able to operate autonomously within patients, adapting to short-term and long-term changes, with required lifetimes in the order of decades. Runtime reconfigurability can be achieved both in hardware and software; the latter is less challenging to implement, however, the former is often unavoidable. In this paper we focus on hardware reconfigurability.

Formal methods provide a systematic approach for developing complex systems in a reusable and correct-by-construction manner. The suitability of these techniques for the specification and verification of reconfigurable systems has been studied for some time now. We refer the reader to a survey by Calinescu and Kikuchi [1] that overviews the use of formal methods for adaptive systems at runtime, and an empirical comparison of different modelling formalisms by Bhattacharyya et al. [2]. In our work [3] we introduced a methodology for the formal specification and synthesis of processor microarchitecture with runtime reconfigurability. The methodology is supported by open-source EDA toolsuite WORKCRAFT [4] and has been validated in silicon [5]. In the rest of the paper we discuss our current work on providing fast prototyping of reconfigurable resilient processing cores in WORKCRAFT using a domain-specific language for microarchitecture description (Section II), and report on our experience in designing resilient processing cores using this methodology (Section III).

II. DSLS FOR IMPROVED DESIGN PRODUCTIVITY

Domain-Specific Languages (DSLs) are designed to have a maximal expression for tasks in a particular domain (for example, VHDL for hardware description or LaTeX for typesetting). However, implementing a language from scratch may be tedious, time-consuming and error-prone. Therefore, DSLs are often embedded into existing general-purpose programming languages, which is particularly convenient for prototyping purposes. Modern functional programming languages such as Haskell offer a wide range of facilities for construction of *Embedded Domain-Specific Languages* (EDSLs) that benefit from features of lightweight formal verification provided by the rich type system and highly-tailored syntax achieved using various functional programming idioms [6].

To design resilient and reconfigurable systems, it is vital to have formal specification methods, simulation facilities and verification techniques. EDSLs can increase the productivity at every stage of hardware design: high-level specification languages help to describe the system functionality in a declarative way, software simulation environments allow to evaluate the system capabilities without fabricating an expensive prototype, and advanced types of the host language provide compiler-checked correctness guarantees for synthesis.

The WORKCRAFT framework provides three DSLs:

- Signal Transition Graphs (STGs), a signal-level DSL for specifying resilient asynchronous controllers [7].
- Conditional Partial Order Graphs (CPOGs), a DSL for specifying reconfigurable processor microarchitectures supported by optimal instruction encoding algorithms [3].
- Dataflow Structures (DFSs), a dataflow-level DSL for specifying dataflow computation graphs [8].

WORKCRAFT can synthesise and export models described in these DSLs into Verilog, a low-level language for hardware description, supported by conventional EDA tool chain.

In our work, we focus on bridging Event-B [9], a high-level formal notation for the specification of system requirements and reconfiguration, and DSLs provided by WORKCRAFT. As a prototype of a bridging language, we present *Farfalle*, an intermediate-level DSL embedded in Haskell for the description of reconfigurable processor microarchitectures. Haskell provides powerful abstractions, e.g. the `Monad` type class [10], that help to build custom EDSLs with strong static typing and compositional semantics.

Here are two simple examples of *Farfalle* code, highlighting the power of Haskell's monadic `do`-notation for building composable EDSL programs in clear imperative step-by-step

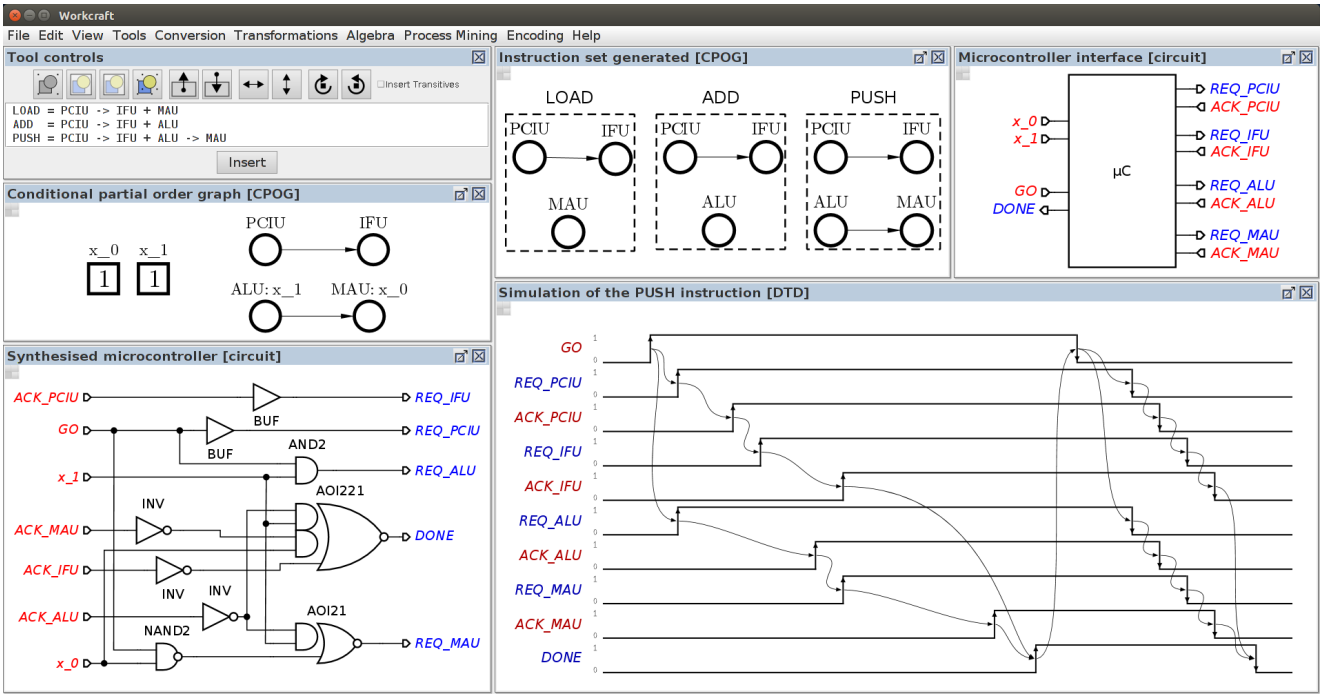

Fig. 1. From formal specification to hardware synthesis and simulation of a simple 3-instruction processing core. A screenshot of WORKCRAFT.

manner. To *increment a register*, one needs to read the value it contains and then write the incremented value back:

```
increment register = do
    value <- readRegister register
    writeRegister register (value + 1)
```

To *fetch an immediate argument*, one needs to increment the program counter `pc` and fetch the value it points to. Note how easily one can reuse previously defined `increment` function:

```
fetchArgument = do
    increment pc
    address <- readRegister pc
    readMemory address
```

Farfalle code can be translated both to Event-B notation for formal verification and to one of WORKCRAFT's DSLs, e.g. a CPOG model for further hardware synthesis, as illustrated in Fig. 1, providing engineers a fast and safe prototyping tool.

III. PROTOTYPES OF RESILIENT PROCESSING CORES

We validated the design flow in Fig. 1 by fabricating two ASIC prototypes: an asynchronous Intel 8051 core [5] and a dataflow accelerator. Both were designed to survive in a wide range of voltages and supported runtime reconfigurability.

Intel 8051 chip was fabricated in the 130nm process using the standard cell library by STMicroelectronics, with the nominal supply voltage of 1.2V. Thanks to runtime reconfigurability the chip could operate in the voltage range 0.22-1.5V. Parts of the chip failed at certain voltages: the SRAM failed below 0.89V, and the conventional program counter was unreliable below 0.74V. However, the processing core, designed using the presented methodology, operated correctly down to 0.22V.

The dataflow accelerator prototype (our DATE'17 University Booth demo) was designed as an asynchronous dataflow processor with reconfigurable pipeline depth. It was fabricated

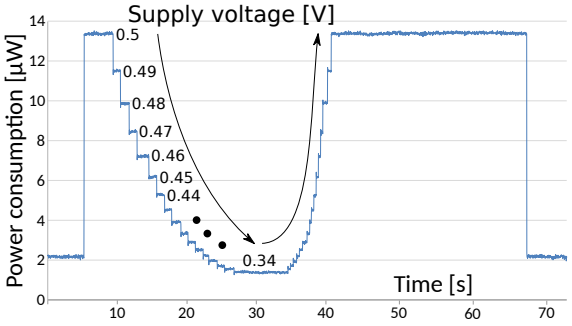

Fig. 2. Resiliency of asynchronous control under unstable voltage.

using the TSMC 90nm CMOS technology for low-power applications and retained functionality in the range 0.34-1.6V. Fig. 2 shows the resiliency of the chip, which adapts to varying voltage by slowing down and reducing power consumption.

REFERENCES

[1] R. Calinescu, S. Kikuchi. *"Formal methods @ runtime"*. In Proceedings of the Monterey Workshop, Pages 122135, Springer, 2010.

[2] A. Bhattacharyya, A. Mokhov, K. Pierce. *"An Empirical Comparison of Formalisms for Modelling and Analysis of Dynamic Reconfiguration of Dependable Systems"*. Formal Aspects of Computing, Springer, 2016.

[3] A. Mokhov et al. *"Synthesis of processor instruction sets from high-level ISA specifications"*. IEEE Transaction on Computers 2014, vol. 63(6).

[4] WORKCRAFT framework homepage: http://www.workcraft.org/.

[5] A. Mokhov, M. Rykunov, D. Sokolov, A. Yakovlev. *"Design of Processors with Reconfigurable Microarchitecture"*. J. Low Power Electronics Application, 2014, vol. 4(1), pp. 26-43.

[6] P. Hudak. *"Modular Domain Specific Languages and Tools"*. Proceedings of the International Conference on Software Reuse, 1998, p. 134.

[7] J. Cortadella et al. *"Logic synthesis for asynchronous controllers and interfaces"*, Springer, 2012.

[8] D. Sokolov, I. Poliakov, A. Yakovlev. *"Analysis of static data flow structures"*. Fundamenta Informaticae, vol. 88(4), pp. 581-610, 2008.

[9] J-R. Abrial. *Modeling in Event-B: system and software engineering*. Cambridge University Press, 2010.

[10] P. Wadler. *"Monads for functional programming"*. Advanced Functional Programming, Bastad Spring School, 1995.