ForSyDe on the Patmos Processor

Ehsan Khodadad*, Ingo Sander†, Luca Pezzarossa*, and Martin Schoeberl*
*Department of Applied Mathematics and Computer Science, Technical University of Denmark, Kongens Lyngby, Denmark
Email: [ehkh, lpez, masca]@dtu.dk
†Division of Electronics and Embedded Systems, KTH Royal Institute of Technology, Stockholm, Sweden
Email: ingo@kth.se

***Abstract*—Due to strict timing requirements, designing embedded real-time systems for safety-critical applications is inherently complex. The design paradigm should not only be able to guarantee that the final system works but also fulfill the safety-critical specifications and requirements. This paper proposes a design approach for embedded systems using T-CREST, a time-predictable hardware platform, and ForSyDe, a modeling framework supporting the formal design of embedded systems. We designed different use cases, performed preliminary experiments, and demonstrated the possibility of using this method. Furthermore, we perform worst-case execution time (WCET) analysis of our example applications.**

***Index Terms*—T-CREST, ForSyDe, real-time systems, time-predictable computer architecture, WCET Analysis.**

I. Introduction

Developing embedded systems for safety-critical applications can present significant challenges due to stringent timing requirements. A robust design paradigm is essential to ensure the correct functionality of the final system and to meet all safety-critical specifications and requirements. Our project aims to provide designers with a complete hardware and software design framework for embedded systems for safety-critical applications. This method uses ForSyDe [9] on top of the T-CREST [10] hardware platform.

ForSyDe, which stands for Formal System Design, is a formal model based on well-defined semantics, making formal analysis and verification easier. In this method, the designer focuses on functionality rather than implementation details. ForSyDe is a well-defined model that complies with a model of computation. Moreover, ForSyDe provides the designers with an abstraction between specification and implementation.

The T-CREST platform is a multi-core hardware architecture specially developed to target time-predictable systems [10]. Patmos, a RISC processor designed for real-time embedded systems, is the central processing unit used in T-CREST [12].

This paper presents our initial efforts and preliminary results toward creating a complete software and hardware design framework dedicated to safety-critical systems by combining T-CREST with ForSyDe. The proposed flow outlines a systematic process for translating high-level models to executable code, paving the way for automated, reliable, and time-predictable code generation.

The main contributions of this paper are (1) the integration of ForSyDe into T-CREST, (2) preliminary experiments to prove the correct functionality and effectiveness of the proposed method, and (3) performing a WCET analysis of individual functions.

This paper is organized into six sections: Section II presents the background on the T-CREST multicore and ForSyDe. Section III describes our port of ForSyDe to the T-CREST multicore platform. Section IV presents the preliminary experiments demonstrating correct ForSyDe code execution and WCET analysis. Section V presents related work. Finally, Section VI concludes the paper.

II. Background

This section introduces the hardware and software we use in this work: the T-CREST platform and the ForSyDe framework.

A. The T-CREST Platform and the Patmos Processor

"Time-predictable multi-Core aRchitecture for Embedded SysTems" (T-CREST) is a comprehensive time-predictable multi-core platform explicitly tailored for safety-critical applications. It mainly consists of several processor nodes connected by a network-on-chip [1]. The system relies on time-division multiplexing to arbitrate resource access and ensure time guarantees.

Patmos [12] is the processor used in T-CREST. It is a RISC-based time-predictable processor specifically designed for real-time embedded systems. The Patmos processor is designed to simplify WCET analysis. To do so, all instruction delays are visible at the instruction set architecture. The Patmos processor is supported by its compiler, simulator, and WCET analysis tool in open source. The Patmos compiler [7] is based on the open-source LLVM platform.[1]

The T-CREST platform also has a WCET analyzer called Platin [6]. Platin is a Ruby-based toolkit that stores information about the program structure and meta-information, such as flow facts and analysis results, in a target-machine-agnostic

Ehsan Khodadad, Luca Pezzarossa, and Martin Schoeberl are with the Department of Applied Mathematics and Computer Science, Technical University of Denmark, Kongens Lyngby, Denmark (email: ehkh@dtu.dk, lpez@dtu.dk, masca@dtu.dk)
Ingo Sander is with the Division of Electronic and Embedded Systems, KTH Royal Institute of Technology, Stockholm, Sweden. (email: ingo@kth.se)

[1] https://llvm.org/

form. It includes flow fact transformation, WCET analysis, graph visualization, and tool configuration. This tool uses PML files generated during compile time to generate reports about WCET results on Patmos.

The Patmos processor and T-CREST are written in Chisel,[2] an open-source hardware description language. Chisel adds features to the Scala programming language to describe hardware. Then, Verilog code is generated, which can later be synthesized and implemented for almost any FPGA.

B. The ForSyDe Framework

ForSyDe is a modeling framework for designing embedded systems using synchronous and data flow MoCs, aiming to be correct by construction as an embedded domain-specific language (e-DSL) in the Haskell[3] programming language. ForSyDe is an excellent starting point for formal specification and synthesis of predictable embedded platforms. This is because it supports a few powerful and formal building blocks in the form of process constructors and skeletons, and has a formal base in the form of the tagged signal. These concepts and building blocks are a prerequisite for rigorous design.

The general definition of the ForSyDe computational model uses a similar notation to the network of concurrent processes introduced in [3]. In this context, a signal is an infinite sequence of events, each with a tag and a value. The position in the list defines the tag of an event. On the other hand, A process maps input signals to output signals; it accepts, manipulates, and emits signals. A process in ForSyDe is synchronous if its input and output signals share the same signal rate (how often they produce or consume data).

In ForSyDe, designers are forced to use *process constructors* to construct a process. Process constructors are high-order functions that take zero or more functions and zero or more variables and produce a process. They are designed separately for each MoC, including synchronous and data flow.

The synchronous library in ForSyDe defines process constructors for the synchronous MoC. They are either combinational or sequential. Combinational process constructors are used for processes without internal states; they depend only on the current input values. `mapSY`, the basic combinational process constructor, constructs a process that maps the input function to the input signal. Thus, there is always a separation between communication/synchronization, provided by the process constructor, and computation, provided by the combinational function (Figure 1). `zipWithSY` is another synchronous process constructor that creates processes with multiple input signals. In `"adder x y = zipWithSY (+) x y"`, e.g., `adder` is a process that takes two signals and produces an output signal. Here, `adder` is constructed by `zipWithSY` process constructor and the `(+)` combinational function.

Sequential process constructors are used for processes with internal states; they depend not only on the current input values but also on the current state, and their initial state is one of the input parameters. The base sequential process constructor in ForSyDe is `delaySY`, which delays the signal one event cycle by adding an initial value at the beginning of the output signal, implying one event (the first) at the output signal with no corresponding event at the input signal [9]. For example, `delaySY 0 [1..5]` returns `[0,1,2,3,4,5]`.

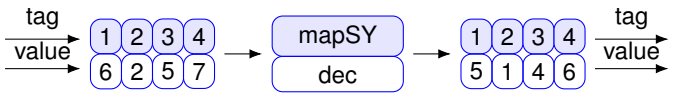

Fig. 1: The block diagram of a `P = mapSY dec where dec x=x-1` process constructed by `mapSY` process constructor.

Listing 1 and Figure 2 show a sample code in ForSyDe using ForSyDe-shallow[4] library's `zipWithSY` and `delaySY` process constructors, in which we define a function (`system`) that accepts a list of `Signal Integer` and returns a list of the same type, so the output signal is calculated by element-wise adding the input signal to an intermediate computation signal called `internal`. This signal is the one-cycle delayed version of the output signal. Thus, the first internal element will be the default value 5, and subsequent elements will be the delayed output values. The input signal `[1..5]` leads the internal signal to `[5,6,8,11,15]` and the output to `[6,8,11,15,20]`.

```haskell
import ForSyDe.Shallow
system :: Signal Integer
       -> Signal Integer
system input = output where
  output   = adder input internal
  adder    = zipWithSY (+)
  internal = delaySY 5 output
```

Listing 1: ForSyDe sample code.

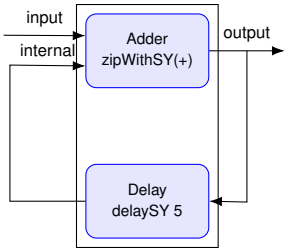

Fig. 2: ForSyDe example block diagram.

Another useful sequential process constructor in the ForSyDe-shallow library is `mooreSY` that describes a Moore-style FSM (see Figure 3). It takes two functions, `"nextState"` and `"output"`, and one `"initState"` value. Considering `"mooreSY nextState output initState"` as the prototype of this function, `"output"` is the output function computing the output from the current state, and `"nextState"` is the next-state function updating the state based on each input. In `"mooreSY (+) (*2) 0 (signal [1..5])"`, e.g., the next-state function is (+), the output function is (*2), the initial value is 0, and the input is a signal of Integers from 1 to 5. Then, the input signal and delayed signal are combined element-wise by `zipWithSY (+)`. Since the `"initState"` value is zero, the first output of the `nextState` is `1+0=1`, and the others are the sum of the input signal element and the previous one-cycle-delayed state. Thus, the internal state is `[0,1,3,6,10,15]`, and is mapped to the output function `((*2))` results in `[0,2,6,12,20,30]`.

[2]https://www.chisel-lang.org/
[3]https://www.haskell.org/
[4]https://hackage.haskell.org/package/forsyde-shallow

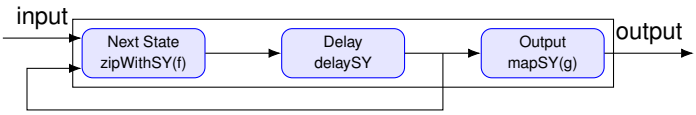

Fig. 3: The block diagram of the "mooreSY f g i" process constructor.

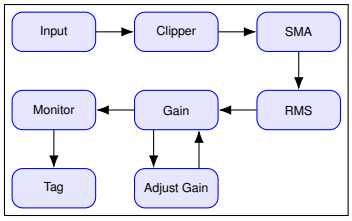

Fig. 4: Audio DSP model with feedback block diagram.

Application	Function	WCET
Audio DSP	Clipper	1374
	WindowSY	6453
	SMA	7597
	RMS	7914
	Gain	1255
	Monitor	1374
	Tag	1208

TABLE I: Individual WCET bounds in clock cycles.

III. ForSyDe on the T-CREST Embedded Platform

As discussed in the previous sections, ForSyDe offers the system designer modeling libraries with a sound formal base for a set of well-defined models of computation to ensure designs are unambiguous, analyzable, and verifiable. We strongly believe that ForSyDe's powerful high-level patterns (the process constructors like `mapSY` and `mooreSY` and skeletons), which can have a corresponding implementation interpretation (such as a C- or VHDL-template) in the target language, can give support to rigorous design for embedded systems in safety-critical areas. Combining the ForSyDe software framework with a time-predictable hardware framework like T-CREST provides designers with a complementary hardware-software framework.

However, since T-CREST supports only the C language, we need a translation of ForSyDe's Haskell into T-CREST's C. Although the hardware synthesis tool has been implemented in ForSyDe-Deep based on the translation rules to VHDL in [9], there is no compiler for software synthesis so far in ForSyDe. There are various practical reasons for not having a software synthesis tool. One of the main reasons is that ForSyDe mainly focuses on system design from a high level of abstraction, where several models of computations and skeletons are supported, and considerable work on design space exploration focuses on looking at higher-level concepts for improving the correctness of the specification and formal design transformations.

For the software translation of synchronous ForSyDe models, [5] describes a synthesis scheme from ForSyDe to C and VHDL, which is based on the hardware synthesis approach presented in [8]. It is important to note that in both studies, the focus has been on the translation processes rather than on optimization techniques.

This paper proposes a manual but systematic and automatable translation of the synchronous ForSyDe model onto the T-CREST platform using the code templates for the process constructors used in the model for C. Our main contribution is presenting the possibility of using ForSyDe on the T-CREST platform, so we concentrate on the idea rather than the tool implementation. To do so, we demonstrate a use case as initial efforts in using ForSyDe on top of the Patmos processor.

We demonstrate a DSP-based model for manipulating an audio signal through different ubiquitous functions in the signal processing world; its code is available in `dsp.hs` file located in the `src/ForSyDe/Shallow/Example` folder of `forsyde-shallow-examples` repository.[5] In this sample, the first function, `Clipper`, clips the original signal by removing too-high or too-low values. Then, the output passes through two averaging functions, a simple moving average (SMA) and a root mean square (RMS). Both use a windowing mechanism implemented in `windowSY` by `mooreSY`. In this example, `SMA` defines an averaging function and applies it to a window of size `n` by `mapSY`. Similarly, `RMS` does the same but with a different averaging formula. Then, `Gain` multiples its input signal by a given number to make the audio signal louder. To combine all these functions, we define `Audio` as the main function of our example and connect them so that the output of each is the input of the next function.

We add a feedback feature to the gain to make our model more realistic and complex. To do so, we add a function called `adjGain`, and modify our `Audio` accordingly using `mooreSY`. This function is later used as the `nextState` of the `mooreSY` along with the identity (`id`) function as the output function and the new `initGain` input as the initial state. The output of the `Gain` function, named `adjusted`, is amplified based on the input `rms` signal, which is used in the next line as the multiplier's parameter. Figure 4 shows a block diagram of this system.

IV. Preliminary Experiments

In this section, we present the preliminary experiments for the proposed method. For this purpose, we configured both ForSyDe and Patmos projects as our method's software and hardware parts. Then, we simulated and executed the code mentioned using these tools.

In the software part of our proposed method, we start a project using ForSyDe's stack template[6] and add our Haskell code as an `hs` file (Haskell's extension) in it. In our experiments, we use version 3.5.0.0 of ForSyDe-Shallow and set the resolver to lts-22.39. We first simulate our examples using two Haskell tools: GHCi and Haddock.[7] GHCi is the GHC(Glasgow Haskell Compiler)'s interactive environment. It can evaluate Haskell expressions, interpret their programs, and load GHC-compiled modules.

We run `stack ghci --package=forsyde-shallow <file_name>.hs` command to start GHCi and load our Haskell file. In this command, `--package` tells Stack to include the `forsyde-shallow` package.

Another alternative for testing is Haddock, an online documentation method that produces HTML-based pages and executes the functions in the comments. By adding >>> to

[5]https://github.com/forsyde/forsyde-shallow-examples

[6]https://github.com/forsyde/stack-templates

[7]https://hackage.haskell.org/package/haddock

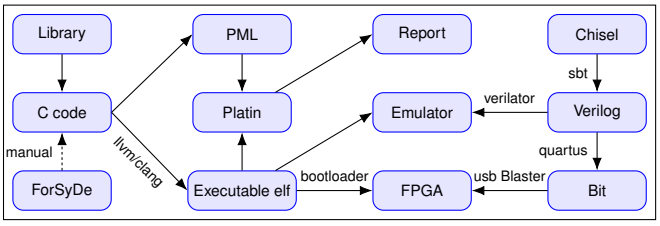

Fig. 5: Tool flow for ForSyDe on the Patmos processor.

Haskell's comment characters, double dashes (`--`), we can run Haskell's one-line code as in GHCi but in the same text editor and file we code in, e.g. `-- >>> Audio gain winSize audio`.

After testing the code with Haskell's tools and ensuring its correctness, we test it on the T-CREST embedded system. We use the DE2-115 board,[8] equipped with an Altera Cyclone IV FPGA, to perform preliminary experiments on the proposed method. In the T-CREST platform, Quartus synthesizes the Verilog code generated from the Chisel source and generates the Patmos bit file. Then, we configure the FPGA with this bit file by Quartus's USB-Blaster programmer. Finally, we download the ELF software executable file through the serial port (Figure 5).

Since ForSyDe does not have an automatic translator from Haskell to C, we translate it manually. We add it as an application to the Patmos project, available in the `c/apps/forsyde` folder of Patmos's repository.[9] Therefore, we can easily compile, execute, simulate, and analyze it similarly to other Patmos applications in that folder. We use the *Patmos-clang* compiler to compile the code, the Pasim simulator to simulate the software, the Patemu emulator to run the software on the emulated hardware, and the Platin analyzer to analyze the WCET of individual functions. Table I shows the WCET results of analyzing the functions of our sample examples as computed by the PLATIN analyzer.

V. Related Work

Many similar works have been developed, extended, and fine-tuned in modern programming languages and combined with other software frameworks and operating systems specifically for embedded computer systems. Among them, we selected Lingua Franca to evaluate and compare with ForSyDe.

Lingua Franca (LF) [4] is a coordination language and framework that can generate C, C++, Python, TypeScript, and Rust code and integrate with legacy code. The results can be deployed on almost every computer system, including T-CREST [2], [11]. The main building block of an LF program is called a reactor. The instructions for what each reactor should do are written in one of the target programming languages.

We compare our proposed method with LF by rewriting Listing 1 and putting it in the `test/C/src/patmos` folder of LF's repository.[10]Comparing them shows not only how less complicated and more concise a ForSyDe model could be than an LF one (7 vs. 49 lines, 85% less), but also shows ForSyDe's ability to nicely combine different complex system components, e.g., the way a default value of an input can be modeled in each.

[8]www.terasic.com.tw

[9]https://github.com/t-crest/patmos

[10]https://github.com/lf-lang/lingua-franca

VI. Conclusion

In this paper, we propose using ForSyDe as the software framework for the T-CREST platform. Combining a time-predictable hardware platform with a solid modeling framework based on a functional programming language and data-flow MoC can lead to reliable and automated embedded systems for safety-critical applications. We believe that this approach not only provides engineers with a new practical workflow to develop high-risk embedded systems and analyze their timing behavior, but also opens a new door for researchers to explore.

References

[1] Evangelia Kasapaki, Martin Schoeberl, Rasmus Bo Sørensen, Christian T. Müller, Kees Goossens, and Jens Sparsø. Argo: A real-time network-on-chip architecture with an efficient GALS implementation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24:479–492, 2016.

[2] Ehsan Khodadad, Luca Pezzarossa, and Martin Schoeberl. Towards Lingua Franca on the Patmos processor. In *2024 IEEE 27th International Symposium on Real-Time Distributed Computing (ISORC)*, 2024.

[3] E.A. Lee and A. Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(12):1217–1229, 1998.

[4] Marten Lohstroh, Christian Menard, Soroush Bateni, and Edward A. Lee. Toward a lingua franca for deterministic concurrent systems. *ACM Transactions on Embedded Computing Systems*, 20(4):36, 2021.

[5] Zhonghai Lu, Ingo Sander, and Axel Jantsch. A case study of hardware and software synthesis in ForSyDe. In *Proceedings of the 15th International Symposium on System Synthesis*, ISSS '02, page 86–91, New York, NY, USA, 2002. Association for Computing Machinery.

[6] Emad Jacob Maroun, Eva Dengler, Christian Dietrich, Stefan Hepp, Henriette Herzog, Benedikt Huber, Jens Knoop, Daniel Wiltsche-Prokesch, Peter Puschner, Phillip Raffeck, Martin Schoeberl, Simon Schuster, and Peter Wägemann. The Platin Multi-Target Worst-Case Analysis Tool. In Thomas Carle, editor, *22nd International Workshop on Worst-Case Execution Time Analysis (WCET 2024)*, volume 121 of *Open Access Series in Informatics (OASIcs)*, pages 2:1–2:14, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[7] Peter Puschner, Daniel Prokesch, Benedikt Huber, Jens Knoop, Stefan Hepp, and Gernot Gebhard. The T-CREST approach of compiler and WCET-analysis integration. In *16th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 2013)*, pages 1–8, 2013.

[8] I. Sander and A. Jantsch. System synthesis based on a formal computational model and skeletons. In *Proceedings. IEEE Computer Society Workshop on VLSI '99. System Design: Towards System-on-a-Chip Paradigm*, pages 32–39, 1999.

[9] Ingo Sander. *System Modeling and Design Refinement in ForSyDe*. PhD thesis, KTH, Microelectronics and Information Technology, IMIT, 2003. NR 20140805.

[10] Martin Schoeberl, Sahar Abbaspour, Benny Akesson, Neil Audsley, Raffaele Capasso, Jamie Garside, Kees Goossens, Sven Goossens, Scott Hansen, Reinhold Heckmann, Stefan Hepp, Benedikt Huber, Alexander Jordan, Evangelia Kasapaki, Jens Knoop, Yonghui Li, Daniel Prokesch, Wolfgang Puffitsch, Peter Puschner, André Rocha, Cláudio Silva, Jens Sparsø, and Alessandro Tocchi. T-CREST: Time-predictable multi-core architecture for embedded systems. *Journal of Systems Architecture*, 61(9):449–471, 2015.

[11] Martin Schoeberl, Ehsan Khodadad, Shaokai Lin, Emad Jacob Maroun, Luca Pezzarossa, and Edward A. Lee. Invited paper: Worst-case execution time analysis of Lingua Franca applications. *Openaccess Series in Informatics*, 121:4, 2024.

[12] Martin Schoeberl, Wolfgang Puffitsch, Stefan Hepp, Benedikt Huber, and Daniel Prokesch. Patmos: A time-predictable microprocessor. *Real-Time Systems*, 54(2):389–423, April 2018.