*papageno*X: Generation of Electronics and Logic for Embedded Systems from Application Software

Tobias Scheipel[a] and Marcel Baunach[b]
Institute of Technical Informatics, Graz University of Technology, Graz, Austria

Keywords: Embedded Systems, Printed Circuit Board, Design Automation, Hardware/Software Co-design, Systems Engineering, Reconfigurable Logic.

Abstract: Embedded systems development usually starts with hardware engineering based on specific requirements of the systems. These requirements are mainly derived from the needs of the not yet developed software to be executed on the system. This process is predictive and many iterations are thus needed, as new requirements often arise during the software development period. In the future, the market will demand more and more sophisticated embedded systems with a much reduced time to market. It will thus be inevitable that system prototypes and series products will need to be created as fast as possible. To enable this, we propose a top-down approach termed ***papageno***X, dealing with the question of "How to generate all layers X of the embedded systems stack including hardware and reconfigurable logic units from application software?". The present work is a work in progress and deals with the definition of the research questions and several ideas and concepts of how to fundamentally solve them. Hence, it aims at introducing ideas to create a generator for embedded systems electronics, reconfigurable logic and software.

1 INTRODUCTION

There is currently a common trend in embedded systems software engineering: code generation. Examples of this include code generation for Application Software (ASW) and Basic Software (BSW) from MATLAB models within automotive Electronic Control Units (ECUs) (AUTOSAR, 2017), and even Real-Time Operating Systems (RTOSs) can be ported automatically to other platforms by generating code from formal models (Gomes and Baunach, 2019). In this work, we propose a concept going a step further and generating an entire embedded system with its hardware from software, as illustrated in Fig. 1. Hence, we formulate the overall research question as *"How can we enable automatic embedded systems generation from Application Software?"*. Hardware in this case subsumes not only reconfigurable logic in Field Programmable Gate Arrays (FPGAs), but also electronics on Printed Circuit Boards (PCBs).

Our newly introduced process, which is primarily designed for prototyping, will also help to evaluate the intended ASW changes after deployment to quantify the consequences on the lower layers and thus assess their feasibility and cost. This is especially important to find out if hardware changes in the system are necessary or if the existing system is still sufficient to execute the ASW. In addition, the process is generally designed to drive the use of reconfigurable logic by seamlessly integrating it into the development and maintenance process.

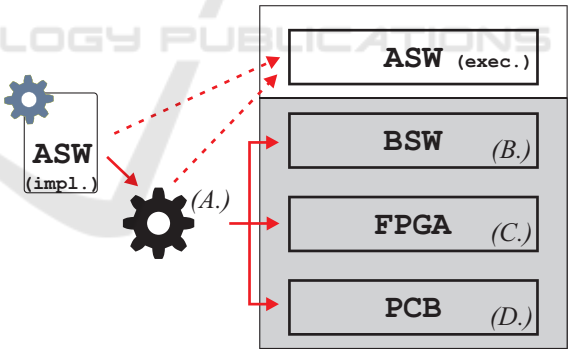

Figure 1: ***papageno***X divided into the main parts, shown in the stack of an embedded system (impl./exec. = implementation and executable of the ASW).

[a] https://orcid.org/0000-0003-0691-6119
[b] https://orcid.org/0000-0002-3716-2682

Scheipel, T. and Baunach, M.
papagenoX: Generation of Electronics and Logic for Embedded Systems from Application Software.
DOI: 10.5220/0009159701360141
In *Proceedings of the 9th International Conference on Sensor Networks (SENSORNETS 2020)*, pages 136-141
ISBN: 978-989-758-403-9; ISSN: 2184-4380

The present paper presents the work in progress and is organized as follows: Section 2 motivates the topic of embedded systems generation from application software. In Section 3, the main idea and the scientific approach for ***papageno***X is outlined. In Section 4 all the envisioned parts of the concept are discussed in detail and the overall concept is sketched with an exemplary use case in Section 5. The paper concludes with Section 6, in which the current state, the progress made and the future work are explained.

2 MOTIVATION

The common process in embedded systems engineering today usually starts with designing the hardware according to requirements defined in natural language. Subsequently, based on educated guesses or best practices, a prototype is assembled, simulated and tested. After this step, software development on the prototype can start. As during software development, new requirements can arise, this process is prone to require multiple iterative redesign cycles in order to result in a suitable hardware solution. In this context, software development is much more dynamic and flexible than the new development of hardware prototypes. As a result, the prototypes are used for as long as possible in order to avoid the time and effort involved in hardware redesign. In some cases, however, the redesign becomes inevitable, resulting in many different prototype variants with diverse properties. AS a result the state of the art is what we call software development following hardware design ("software follows or adjusts to hardware").

With our contribution, we intend to invert the process of embedded systems design towards "hardware follows software", introduced in detail in the following Section 3.

3 SCIENTIFIC APPROACH WITHIN *papageno*X

In order to tackle the problems that arise as described above, we intend to derive the requirements of a system under development directly from previously implemented application code in order to automatically generate suitable and optimized lower layers. As application code does not explicitly specify discrete components but only functional requirements (FRs), the analysis shall also consider additional non-functional requirement (NFR) (e.g., communication load, data retention time).

We have given our concept the name ***papageno***X, an abbreviation for **P**rototyping **AP**plication-based with **A**utomatic **GEN**eration **O**f **X**, where **X** is for the name of the corresponding layer (cf. Fig. 1).

The following structure is proposed to achieve the envisioned goal: The process starts with ASW development (implementation), followed by a thorough analysis of all its FRs and NFRs. Based on this analysis, and a library of available hardware and software modules, a configuration or selection space can be opened over all these components, modules, and possible interconnections. As this selection space can still contain system configurations not suitable for the objective (e.g., due to the unavailability of components), further filtering towards special design decisions must be applied. This yields several potential configurations composed from different components that can be optimized again in order to achieve the best system that is suitable for all requirements. Of course, optimization and selection metrics are also based on FRs and NFRs (e.g., cost, size, energy consumption, Electromagnetic Compatibility (EMC) characteristics). This process is applied to all three layers depicted in Fig. 1 in different extent and is explained in the following sections.

In this work, we additionally envision the use of FPGAs or System on Programmable Chips (SoPCs) to build hardware acceleration units by synthesizing complex algorithms in logic (proper interfacing included) to add further adaptability.

The fundamental paradigm shift towards "hardware follows software" is mandatory to enable the utmost flexibility when designing future embedded systems. Since we understand established development methods as being there for a reason, we will also research methods to quantify the suitability of previously generated hardware configurations after software changes, including possible necessary system modification proposals.

4 ENVISIONED PARTS OF *papageno*X

The toolchain of the ***papageno***X concept will contain a number of powerful features that facilitate automatic systems generation. This set of tools can be divided into four main parts, being *(A.)* ASW analysis, *(B.)* BSW generation, *(C.)* FPGA generation, and *(D.)* PCB generation. All those parts are depicted in Fig. 1 and are discussed in detail below. This work presents solely the ideas of from a work in progress concept and only few new findings at this stage.

4.1 ASW Analysis

Starting from the analysis of applications, the research question to deal with is *"Which information needs to be included in ASW including NFRs to allow the creation a embedded system?"*. This means that the requirements of the software to the system must be extracted and forwarded to the next parts. Since the ASW to be analyzed does not have to be a stand-alone executable, the executable part must also be extracted (cf. dotted arrows in Fig. 1).

The requirements analysis derives a selection space by using constraint solving and design space exploration strategies (Saxena and Karsai, 2011)(Nethercote et al., 2007) and logical expression matching. This selection space contains multiple possible overall structures of the entire embedded system:

- suitable computing platform module(s) and
- other hardware components/modules alongside their interconnections
- BSW modules and drivers for hardware modules

Furthermore, this step also performs a pre-selection of algorithms within the code, which are best suited for mapping to reconfigurable logic (FPGAs) in part *(C.)*.

The results of the requirements analysis is forwarded to the following parts.

4.2 BSW Generation

The next part deals with BSW generation and the question on *"How can we derive the needed BSW features from ASW?"*. To deal with this question, we base our approach on our own RTOS, *MCSmart*OS (Martins Gomes et al., 2017), as it can be used with different computing platforms (e.g., MSP430 (Texas Instruments,), RISC-V (Chen and Patterson, 2016)). This operating system must include a generic driver management concept enriched with non-functional properties to allow modular composition of hardware modules including resource sharing mechanisms. Furthermore, such a management concept enables optimization towards different functional and non-functional requirements (e.g., selectable resource management strategies or scheduling algorithms). The output of this part is a BSW tailored to the ASW's needs.

4.3 FPGA Generation

Parallel to the previous part, the FPGA generation starts based on the ASW analysis. Here, the question *"How can we extract application specific logic and automatically map it to FPGAs?"* is addressed. As software functions are commonly executed on Microcontroller Units (MCUs), our contribution aims at hosting application-specific logic components on FPGA platforms, exemplarily depicted as filled squares connected to a soft core MCU in the FPGA module in Fig. 2.

The ASW analysis must discover mapping candidate software functions (via, e.g., heuristics) in order to enable this functionality within our generator. Research is on deciding which functions should be transferred from software to hardware and how to generate or interface them with respect to FRs and NFRs of the ASW (e.g., by transforming a function call to on-chip I/O operations). We will also investigate suitable ASW design patterns and programming paradigms to facilitate this process. Therefore, the entire concept is supported by research on the creation of a flexible soft-core MCU architecture for hosting application-specific logic (based on e.g., *mosart*MCU (Mauroner and Baunach, 2018), RISC-V (Waterman et al., 2016)). The goal is to work on consistent development processes, where all system functions and algorithms are still developed in software, but then automatically mapped to on-chip extensions during compilation and synthesis and interfaced accordingly. We are also working on partial reconfiguration support at runtime to achieve more hardware flexibility and simplified adaptation. In the expectation of more dynamic software updates, we must also consider modifications to the soft-core architecture. This is also important because modifiable hardware will be available in the future, but there is hardly any runtime support for it.

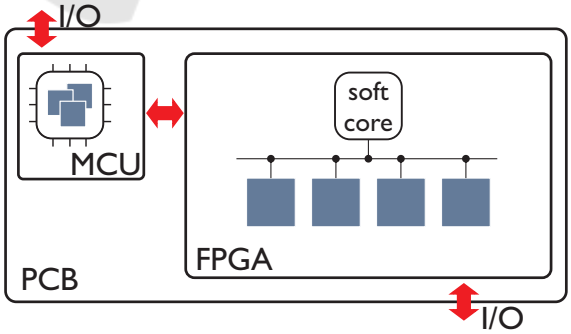

Figure 2: Hosting application-specific logic within an embedded system.

4.4 PCB Generation

The part dealing with the lowest layer in the embedded systems stack is sketched in this section. The

attempt is made to answer the questions *"What information is needed to automatically generate PCBs from ASW?"* and *"How can this information be used to generate a PCB prototype matching all ASW requirements?"*. These questions can be answered in two sub-parts by utilizing dedicated hardware modules as abstractions. The generated board is either a motherboard, where these modules can be plugged in, or a single PCB in which all the necessary electrical components are integrated.

The first of the two sub-parts deals with the translation of constraints to a selection space of possible system configurations. This is an abstract process and does not deal with any electrical properties yet. To do so, enriched module definitions must be created. Our definitions are based on generic JSON (ECMA International, 2017) descriptions including not only their functional, but also their non-functional properties. This approach allows for possible systems configurations to be established and a set of suitable configurations can be filtered out by the toolchain. The result is an intermediate system definition. This is because the electrical feasibility of the connections between the modules has not yet been considered, and no dedicated PCB has been generated yet.

In this second sub-part, the interconnection on electrical level is being dealt with. As stated in (Scheipel and Baunach, 2019), this sub-part uses an interconnected intermediate system definition to convert it into a PCB. Several different inputs are necessary for this:

- hardware module definition files and their corresponding design block schematics and board layouts,
- all necessary interface definitions to adequatly interconnect the previously defined hardware modules, and
- one dedicated system definition file which represents the overall structure of the embedded system (cf. first sub-part).

With these input files, schematics and board plans can be generated. In the first step, the output of our approach is based on the EAGLE (Autodesk, Inc.,) XML format.

5 USE CASE EXAMPLE

An idea to create an embedded system shall be derived from the need of observing or controlling some specific (physical) process. In our case, it starts with the need of "measuring distances for further processing in a car". This requirements definition in natural language spans a selection space of a great number of components and software possibilities for implementing our system. As the engineer knows that there is a need to "measure distance" (I.) and send the distance values to be "further processed in a car" (II.), application software development for the use case can start by using generic Application Programming Interfaces (APIs):

```
1: [...]
2: read_distance(&distance,
                 SensorType.Automotive);
3: send_data(&rec, &distance,
                 CommType.Automotive);
4: [...]
```

This piece of pseudo-code written by the engineer to meet the requirements in language specifies these requirements in more detail, narrowing down the list of components to be considered. In line 2, distance data is read from an automotive-grade sensor (requirement I.; technology need not be specified in detail). Subsequently, it is sent to the recipient `rec` over some automotive in-car communication technology (requirement II.; again, not specified in detail) in line 3. These abstract requirements could also be lists of requirements for further specification.

The next few lines try to give an impression on how ***papageno***X tackles the automatic system generation. Part *(A.)* has already started with the use of certain APIs and is carried out first (cf. Fig. 1). It selects a suitable BSW with drivers and services for part *(B.)*, and selects hardware components for part *(D.)*.

In the analysis part *(A.)*, the requirements are examined more closely in order to obtain a better overview of the actual needs of the ASW. The analysis goes hand in hand with the other parts of the concept.

Subsequently in *(B.)*, the BSW containing an Operating System (OS) is assembled, including all APIs and necessary drivers to use the hardware components selected in part *(D.)*. A tailored version of *MCSmart*OS with a generic driver platform is utilized for this reason. The set of features to enable this functionality is currently under development.

In part *(D.)*, from the set of all possible hardware components and modules several possible system configurations can arise. To simplify the explanation of our use case, the PCB generation part then selects a system containing: a ultra-sonic sensor of type HC-SR04 (Cytron Technologies, 2013) for requirement (I.), and a CAN (International Organization for Standardization, 2015) transceiver with a controller for requirement (II)., both connected to a MSP430 LaunchPad™ (Texas Instruments, 2017) as a computing platform.

Finally, the requirement "measuring distances for

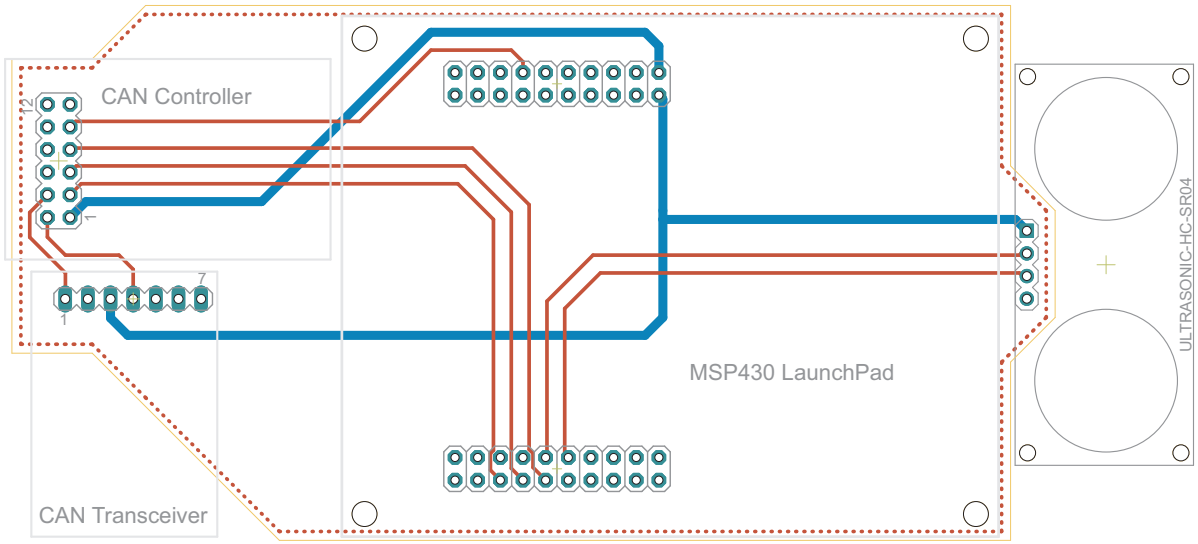

Figure 3: The generated PCB with module placeholders of the explained use case for measuring distances in a car.

further processing in a car" and two lines of code create the embedded system depicted in Fig. 3.

6 CONCLUSION AND FUTURE WORK

In the course of this work we presented a novel approach on how to generate an entire embedded system when only having application code at hand. All the ideas presented are from our current work in progress. As our approach aims at inverting the state-of-the-art process of embedded systems design from "software follows hardware" to "hardware follows software", this opens up new questions in the context of software analysis, and hardware, logic and software generation.

Since this is a work in progress, not all of its parts have yet been implemented. To date, hardware generation on PCB-level is already implemented and partially published, but there is room for improvement throughout the concept. Future steps include the improvement of constraint matching mechanisms for hardware module selection, the improvement of the requirements analysis and the entirety of the reconfigurable logic generation part. Also, the BSW layer must be constantly improved in order to meet the future demands on the concept. As ***papageno***X is generic and module based, it can be easily adopted to different domains in the future. Lowering of the module granularity to, e.g., electrical components level (resistors, capacitors, integrated circuits, and so on) is possible. The next steps in this work are to finish the PCB generation alongside the ASW analysis for FRs and NFRs.

REFERENCES

Autodesk, Inc. EAGLE. [retrieved: Nov, 2019].

AUTOSAR (2017). Classic platform release 4.3.1.

Chen, T. and Patterson, D. A. (2016). RISC-V Geneology. Technical Report UCB/EECS-2016-6, EECS Department, University of California, Berkeley.

Cytron Technologies (2013). *HC-SR04 User's Manual*.

ECMA International (2017). *ECMA-404: The JSON Data Interchange Syntax*, 2 edition.

Gomes, R. M. and Baunach, M. (2019). Code generation from formal models for automatic rtos portability. In *2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 271–272.

International Organization for Standardization (2015). *ISO 11898: Road vehicles – Controller area network (CAN)* , 2 edition.

Martins Gomes, R., Baunach, M., Malenko, M., Batista Ribeiro, L., and Mauroner, F. (2017). A Co-Designed RTOS and MCU Concept for Dynamically Composed Embedded Systems. In *Proc. of the 13th Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, pages 41–46.

Mauroner, F. and Baunach, M. (2018). mosartMCU: Multi-Core Operating-System-Aware Real-Time Microcontroller. In *Proc. of the 7th Mediterranean Conference on Embedded Computing*, pages 1–4.

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., and Tack, G. (2007). MiniZinc: Towards a Standard CP Modelling Language. In Bessière, C., editor, *Principles and Practice of Constraint Program-*

ming – CP 2007, pages 529–543, Berlin, Heidelberg. Springer Berlin Heidelberg.

Saxena, T. and Karsai, G. (2011). A meta-framework for design space exploration. In *2011 18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*, pages 71–80.

Scheipel, T. and Baunach, M. (2019). papagenoPCB: An Automated Printed Circuit Board Generation Approach for Embedded Systems Prototyping. In *ICONS 2019 - The Fourteenth International Conference on Systems*, pages 20–25.

Texas Instruments. MSP430 ultra-low-power sensing and measurement MCUs.

Texas Instruments (2017). *MSP430F5529 LaunchPad™ Development Kit (MSP--EXP430F5529LP)*.

Waterman, A., Lee, Y., Patterson, D. A., and Asanovic, K. (2016). The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.1. Technical Report UCB/EECS-2016-118, EECS Department, University of California, Berkeley.